

General Video Game Playing using Ensemble Decision  
Systems

PhD Thesis

Damien Anderson

Computer and Information Sciences  
University of Strathclyde, Glasgow

July 28, 2020

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

The copyright of this thesis belongs to the author under the terms of the United Kingdom Copyright Acts as qualified by University of Strathclyde Regulation 3.50. Due acknowledgement must always be made of the use of any material contained in, or derived from, this thesis.

Signed: Damien Anderson

Date: 07/10/2019



# Abstract

This thesis explores the application of Ensemble Decision Systems as an Artificial Intelligence (AI) agent for playing video games on the General Video Game Artificial Intelligence (GVGAI) platform.

The GVGAI offers a platform for research into developing General Video Game playing AI agents. Significant progress has been made over the years in the area of game playing AI agents, but it is often a trivial task for designers to propose new problems that the agents are unable to solve. Humans are typically able to solve these additional problems, making them an ideal model for an excellent general video game player and a benchmark for AI agents. One of the objectives of this thesis has been the introduction of Deceptive Games to the GVGAI, which are a class of games that are designed to deliberately deceive AI agents.

Ensemble Decision Systems make use of multiple AI algorithms to make their decisions, which may make them more robust to the problems that can deceive singular AI agents. A wide variety of Ensemble Decision Systems were developed and compared with agents from the GVGAI competition, with the aim of developing an indication of the current level of performance that agents can reach. The Ensemble Decision Systems show improved generality, being able to complete a wider range of games than other agents, at a cost of win rate in specific games.

This thesis presents an Ensemble Decision System for GVGP and a suite of Deceptive Games. The Ensemble Decision System detailed in this thesis manages to outperform comparison agents in the Deceptive Games suite, with the top three positions being taken by Ensemble agents for win rate, and a wider range of games.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xiv</b>
<b>Acknowledgements</b>	<b>xx</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Artificial General Intelligence . . . . .	1
1.2 Why Games? . . . . .	2
1.3 Game Playing Agents . . . . .	3
1.4 General Video Game AI Competition . . . . .	4
1.5 Motivations . . . . .	6
1.6 Contributions . . . . .	7
1.6.1 Ensemble Decision Systems . . . . .	7
1.6.2 Deceptive Games . . . . .	8
1.6.3 Continuous Information Gain . . . . .	8
1.7 Thesis Statement . . . . .	9
1.8 Thesis Structure . . . . .	10
1.9 Publications . . . . .	11
<b>2 Literature Review</b>	<b>12</b>
2.1 Selective History of Artificial Intelligence . . . . .	12
2.1.1 General Game Playing . . . . .	16

## Contents

2.1.2	General Video Game Playing . . . . .	17
2.1.3	General Video Game AI Competition . . . . .	18
2.2	GVGAI Agents . . . . .	21
2.2.1	Tree-Based Search Agents . . . . .	21
2.2.1.1	Monte Carlo Tree Search (MCTS) . . . . .	21
2.2.1.2	MCTS Variations . . . . .	24
2.2.1.3	Fast Knowledge Based MCTS . . . . .	26
2.2.1.4	Self-Adaptive MCTS . . . . .	26
2.2.1.5	Redundant Action Avoidance . . . . .	27
2.2.1.6	Evolving UCB Parameters . . . . .	27
2.2.1.7	Open Loop Expectimax Tree Search (OLETS) . . . . .	28
2.2.2	Evolutionary Search Agents . . . . .	28
2.2.2.1	Rolling Horizon Evolutionary Algorithm (RHEA) . . . . .	29
2.2.3	RHEA Variations . . . . .	29
2.2.3.1	Random Search . . . . .	30
2.2.4	Genetic Programming . . . . .	31
2.2.5	GVGAI Research and Specific Agents . . . . .	31
2.2.5.1	Agent Robustness . . . . .	31
2.2.5.2	Portfolio Agents . . . . .	32
2.2.5.3	Yolobot . . . . .	34
2.2.5.4	Return42 . . . . .	34
2.2.5.5	YBCriber . . . . .	34
2.2.5.6	Hyper-Heuristic Agent . . . . .	34
2.2.5.7	ToVo2 . . . . .	34
2.2.5.8	Goal Orientation . . . . .	35
2.3	Atari . . . . .	35
2.4	AlphaGo and AlphaGoZero . . . . .	37
2.5	Ensemble Decision Systems . . . . .	38
2.5.1	Ensemble Classifiers . . . . .	39
2.5.2	Pandemonium Systems . . . . .	40

## Contents

2.5.3	IBM Watson . . . . .	40
2.5.4	Ms. Pacman . . . . .	40
2.6	Chapter Summary . . . . .	41
<b>3</b>	<b>Deceptive Games</b>	<b>42</b>
3.1	Background . . . . .	42
3.2	What are Deceptive Games? . . . . .	43
3.3	Why are Deceptive Games Useful? . . . . .	44
3.4	Types of Deception . . . . .	46
3.4.1	Greed Trap . . . . .	46
3.4.2	Smoothness Trap . . . . .	47
3.4.3	Generality Trap . . . . .	49
3.5	Butterflies . . . . .	50
3.6	Decepticoins . . . . .	51
3.7	Deceptizelda . . . . .	52
3.8	Flower . . . . .	54
3.9	Invest . . . . .	55
3.10	SisterSaviour . . . . .	56
3.11	WaferThinMints . . . . .	57
3.12	Experiments and Results . . . . .	58
3.12.1	Deceptive Games with Machine Learning Agents . . . . .	60
3.12.1.1	Lack of Hierarchical Understanding . . . . .	60
3.12.1.2	Subverted Generalisation . . . . .	62
3.12.1.3	Delayed Gratification . . . . .	63
3.12.1.4	Delayed Reward . . . . .	64
3.13	Chapter Summary . . . . .	66
<b>4</b>	<b>Game Selection</b>	<b>67</b>
4.1	Information Gain . . . . .	67
4.1.1	The Information Gain Project . . . . .	67
4.1.2	Description . . . . .	68

## Contents

4.1.3	Data Collection . . . . .	70
4.1.4	How does the Information Gain algorithm work? . . . . .	70
4.2	Information Gain Experiments and Results . . . . .	72
4.2.1	Avoidgeorge . . . . .	75
4.2.2	Chopper . . . . .	76
4.2.3	Escape . . . . .	77
4.2.4	Freeway . . . . .	77
4.2.5	Labyrinthdual . . . . .	78
4.2.6	Tercio . . . . .	79
4.2.7	Watergame . . . . .	80
4.2.8	Whackamole . . . . .	81
4.3	Further Games . . . . .	82
4.3.1	Aliens . . . . .	82
4.3.2	Bait . . . . .	83
4.3.3	Camelrace . . . . .	84
4.3.4	Chase . . . . .	85
4.3.5	Crossfire . . . . .	86
4.3.6	Digdug . . . . .	87
4.3.7	Hungrybirds . . . . .	88
4.3.8	Infection . . . . .	89
4.3.9	Intersection . . . . .	90
4.3.10	Lemmings . . . . .	91
4.3.11	Missilecommand . . . . .	92
4.3.12	Modality . . . . .	93
4.3.13	Plaqueattack . . . . .	94
4.3.14	Roguelike . . . . .	95
4.3.15	Seaquest . . . . .	97
4.3.16	Survivezombies . . . . .	98
4.3.17	Waitforbreakfast . . . . .	99
4.4	Chapter Summary . . . . .	100

<b>5</b>	<b>Ensemble Decision Systems</b>	<b>101</b>
5.1	Architecture . . . . .	102
5.1.1	Voices . . . . .	103
5.1.1.1	Behavioural Decomposition . . . . .	104
5.1.1.2	Temporal Decomposition . . . . .	104
5.1.1.3	Perspective Decomposition . . . . .	105
5.1.2	Opinions . . . . .	105
5.1.3	Arbiter . . . . .	105
5.2	Variations . . . . .	107
5.2.1	Arbiter Variations . . . . .	107
5.2.1.1	Time Splitting . . . . .	107
5.2.1.2	Action Selection Policy . . . . .	108
5.2.2	Voice Variations . . . . .	108
5.3	Prototype Experiments . . . . .	109
5.3.1	The Initial Ensemble Decision System . . . . .	109
5.4	Chapter Summary . . . . .	112
<b>6</b>	<b>Experimental Setup</b>	<b>113</b>
6.1	Ensemble Decision Systems for General Video Game Playing . . . . .	113
6.1.1	Heuristics . . . . .	114
6.1.1.1	Win Maximisation . . . . .	115
6.1.1.2	Exploration Maximization . . . . .	115
6.1.1.3	Knowledge Discovery . . . . .	116
6.1.1.4	Knowledge Estimation . . . . .	117
6.1.2	Sample Agents . . . . .	118
6.1.2.1	sampleRandom . . . . .	119
6.1.2.2	One Step Look Ahead (OSLA) . . . . .	119
6.1.2.3	Open-Loop Monte-Carlo Tree Search (OLMCTS) . . . . .	119
6.1.2.4	Open-Loop Expectimax Tree Search (OLETS) . . . . .	120
6.1.2.5	Rolling Horizon Evolutionary Algorithm (RHEA) . . . . .	120
6.1.2.6	Random Search (RS) . . . . .	121

## Contents

6.1.3	Ensemble Agent Variations . . . . .	121
6.1.3.1	BestFour . . . . .	121
6.1.3.2	BestFourDiplo . . . . .	122
6.1.3.3	BestExpSc . . . . .	122
6.1.3.4	MCTSExpSc . . . . .	122
6.1.3.5	OLETSExpSc . . . . .	123
6.1.3.6	OLETSExpScSerial . . . . .	123
6.1.4	Results . . . . .	124
6.2	Design of Further Experiments . . . . .	133
6.2.1	EDS Improvements . . . . .	133
6.2.2	Comparison to Portfolio Agents . . . . .	134
6.3	Further Experiments . . . . .	135
6.3.1	Rationale behind further experiments . . . . .	135
6.3.2	Reproduction . . . . .	137
6.3.2.1	Machine Specifications . . . . .	137
6.3.2.2	Experiment Specifications . . . . .	138
6.3.3	All Actions EDS . . . . .	138
6.3.3.1	Results: All Actions EDS . . . . .	140
6.3.4	Optimistic and Pessimistic MCTS EDS . . . . .	144
6.3.4.1	Results: Optimistic and Pessimistic MCTS EDS . . . . .	144
6.3.5	Concurrent EDS . . . . .	149
6.3.5.1	Results: Concurrent EDS . . . . .	150
6.3.6	Yolobot Voice . . . . .	155
6.3.6.1	Results: Yolobot Voice . . . . .	155
6.3.7	Overall Results . . . . .	158
6.3.7.1	Overall Raw Results . . . . .	158
6.3.7.2	Game Analysis . . . . .	167
6.4	Summary . . . . .	173
<b>7</b>	<b>Conclusions and Further Work</b>	<b>174</b>
7.1	Machine Learning Voice . . . . .	174

## Contents

7.2	Neural Network Arbiter . . . . .	175
7.3	Large EDS . . . . .	175
7.4	Bandit Arbiter . . . . .	176
7.5	Further Concurrency Experiments . . . . .	176
7.6	Voice Decomposition . . . . .	176
7.7	Portfolio EDS . . . . .	177
7.8	Conclusions . . . . .	177
7.9	Contributions . . . . .	178
	<b>Bibliography</b>	<b>179</b>
	<b>A Deceptive Games</b>	<b>191</b>
	<b>B Continuous Information Gain</b>	<b>209</b>
	<b>C Superstition in the Network</b>	<b>218</b>
	<b>D Ensemble Decision Systems for General Video Game Playing</b>	<b>226</b>



# List of Figures

2.1	Search Example: The algorithm looks at the current state and simulates future states. The yellow square represents the current position of the agent, and the green square represents a goal square. . . . .	15
2.2	Learning Example: The algorithm plays the game repeatedly and builds up a policy for how to play it. This policy will then be applied to similar states in the future. Image courtesy of Chiara Diritti. . . . .	15
2.3	A level of Aliens alongside its VGDL representation. There are slight differences between the representations due to time passing in the real game, namely the position of the avatar and the aliens. . . . .	19
2.4	MCTS Visualization. This image shows the four steps of the MCTS algorithm, Selection, Expansion, Simulation and Backpropagation. . . .	22
2.5	Portfolio Agent Example: The portfolio agent selects one sub-agent best suited for the game. Image courtesy of Chiara Diritti. . . . .	33
3.1	Greed Trap Example. The agent is presented with two options, going left or right. The agent decides to go left because there are some valuable coins there, but the path closes behind them and they then realise that there was a valuable treasure further in the distance on the other path. Greedy agents will often choose the most immediately rewarding path over longer term gains. Image courtesy of Chiara Diritti. . . . .	47

## List of Figures

3.2	Smoothness Trap Example. In this example, all of the options available to the agent are statistically likely to lead to poor outcomes. This leads an agent to be risk adverse, as doing nothing becomes the most rewarding option. Image courtesy of Chiara Diritti. . . . .	48
3.3	Generality Trap Example. In this example the agent has to learn that collecting too many of the circles will kill them. This can happen seemingly at random from the perspective of the agent as they do not connect the action of collecting too many with death. Image courtesy of Chiara Diritti. . . . .	49
3.4	Butterflies. This image shows the first level of the GVGAI game Butterflies.	50
3.5	DeceptiCoins Levels. Three of the levels of DeceptiCoins are shown in the above figures. Each level is more complicated than the previous. . .	51
3.6	Deceptizelda Levels. Two levels of Deceptizelda are shown. Level two differs in that there are coins to collect and there is no secondary exit from the level. . . . .	53
3.7	The first level of Flower . . . . .	54
3.8	The first level of Invest . . . . .	55
3.9	The first level of SisterSaviour . . . . .	56
3.10	The first level of WaferThinMints . . . . .	57
3.11	The results of the deceptive experiments. This shows the number of wins that each agent was able to earn on each game. Wins were classed as "Rational" if they managed to avoid falling for the deceptive qualities of the game in question. The agents that perform well in this experiment were not those that typically perform well in the official GVGAI competition. . . . .	59
3.12	DeceptiCoins Levels - Examples of Lack of Hierarchical Understanding. While each level is different, a similar pattern in learning is observed across all of the three levels. Notably, the third level shows a poor performance as the agent does not find the optimal path. . . . .	61

## List of Figures

3.13	WaferThinMints level 1 - Example of Subverted Generalisation. The agent does not manage to learn a good solution to this particular game, likely because of the randomness of where win states are found. . . . .	62
3.14	Flower level 1 - Example of Delayed Gratification. While the agent initially performs well, its performance starts to drop as it gains proficiency at finding the path to the targets before they have matured. . . . .	64
3.15	Invest level 1 - Example of Delayed Reward. The agent initially has a hard time understanding how to advance in this game but eventually develops a behaviour that leads to consistent rewards, though it is not the optimal strategy. . . . .	64
4.1	The first level of Avoidgeorge . . . . .	75
4.2	The first level of Chopper . . . . .	76
4.3	The first level of Escape . . . . .	77
4.4	The first level of Freeway . . . . .	78
4.5	The first level of Labyrinthdual . . . . .	78
4.6	The first level of Tercio . . . . .	79
4.7	The first level of Watergame . . . . .	80
4.8	The first level of Whackamole . . . . .	81
4.9	The first level of Aliens . . . . .	83
4.10	The first level of Bait . . . . .	84
4.11	The first level of Camelrace . . . . .	85
4.12	The first level of Chase . . . . .	86
4.13	The first level of Crossfire . . . . .	86
4.14	The first level of Digdug . . . . .	87
4.15	The first level of Hungrybirds . . . . .	88
4.16	The first level of Infection . . . . .	89
4.17	The first level of Intersection . . . . .	90
4.18	The first level of Lemmings . . . . .	91
4.19	The first level of Missilecommand . . . . .	92
4.20	The first level of Modality . . . . .	93

List of Figures

4.21	The first level of Plaqueattack . . . . .	94
4.22	The first level of Roguelike . . . . .	95
4.23	The first level of Seaquest . . . . .	97
4.24	The first level of Survivezombies . . . . .	98
4.25	The first level of Waitforbreakfast . . . . .	99
5.1	An intuitive view of the Ensemble architecture. The current state is input into the system and each Voice analyses the state. The voices then each return an opinion of what to do and the arbiter takes these opinions into account when deciding what action to take. . . . .	103
6.1	Total percentage of wins (0 – 100%) of each controller by game. Some of the main observations in this data are the SisterSavior performance, where only the EDS agents are able to achieve wins. OLETSExpSc in particular has a wide distribution of wins across the majority of the games.	128
6.2	Average scores per game. To be able to make a comparison between the different games, the scores have been normalized, taking the maximum and minimum average scores obtained in each game as limits for that game. . . . .	129
6.3	<i>Lemmings</i> stats per agent. The percentage of wins is in range [0, 100%] and the average of score is relative to the maximum and minimum scores achieved in the game (range: [-143, 1]). Values have been normalised in this figure. The most interesting thing of note in this figure is that achieving high scores appears to correlate with a lower win rate. . . . .	130
6.4	<i>Invest</i> stats per agent. The percentage of wins is in range [0, 100%] and the average of score is relative to the maximum and minimum scores achieved in the game (range: [-7, 161]). Values have been normalized in this figure. The results for this game show that while winning is relatively easy, achieving a high score is more challenging. . . . .	131

## List of Figures

6.5	<i>Hungrybirds</i> stats per agent. The percentage of wins is in range $[0, 100\%]$ and the average of score is relative to the maximum and minimum scores achieved in the game (range: $[0, 140]$ ). Values have been normalized in this figure. These results show that in this game there is a correlation between score and win rate. EDS agents appear to perform particularly well here. . . . .	132
6.6	Total percentage of wins (0 – 100%) of each controller by game across all games. . . . .	162
6.7	Total percentage of wins (0 – 100%) of each controller by game across the deceptive games. . . . .	164
6.8	Total percentage of wins (0 – 100%) of each controller by game across the information gain games. . . . .	167
6.9	<i>Lemmings</i> stats per agent. The percentage of wins is in range $[0, 100\%]$ and the average of score is relative to the maximum and minimum scores achieved in the game (range: $[-151, 35]$ ). Values have been normalized in this figure. . . . .	170
6.10	<i>Invest</i> stats per agent. The percentage of wins is in range $[0, 100\%]$ and the average of score is relative to the maximum and minimum scores achieved in the game (range: $[-7, 208]$ ). Values have been normalised in this figure. . . . .	172

# List of Tables

2.1	Full list of previous GVGAI planning track winners . . . . .	21
4.1	Example results for the Information Gain process using score and (Std Dev) . . . . .	71
4.2	The games with the highest win rate information gain in descending order.	73
4.3	The games with the highest score information gain. . . . .	73
4.4	The games with the highest combined information gain in descending order. . . . .	74
4.5	The games with the highest cumulative information gain after each recursive step. . . . .	74
4.6	The games selected, in alphabetical order, via the Information Gain process using the combined metric (games in bold are deceptive) . . . .	82
6.1	Full list of controllers used divided into EDS and Sample Agents. . . . .	124
6.2	The Games selected from the GVGAI Framework (games in bold are deceptive) . . . . .	124
6.3	Controllers ranked by total number of F1-points. The EDS agents are highlighted in blue. The OLETS sample agents and OLETS EDS agents are the top performers in terms of F1 points, though the OLETSExpSc agent is able to win more unique games than the sample OLETS agent. This trend can also be seen in the MCTS agents. . . . .	125

## List of Tables

6.4	Controllers ranked by total number of Win Rate (%Wins). The EDS agents are highlighted in blue. The OLETS sample agents and OLETS EDS agents are the top performers in terms of F1 points, though the OLETSExpSc agent is able to win more unique games than the sample OLETS agent. This trend can also be seen in the MCTS agents. . . . .	125
6.5	Controllers ranked by total number of unique games won (#Games). The EDS agents are highlighted in blue. This table shows that the EDS agents perform well when looking at the range of games that they are able to solve, in comparison to the sample agents. . . . .	126
6.6	The full list of agents used for the experiments, along with the type of agent they are and where to find a description of their operation. . . . .	137
6.7	The set of games for the additional experiments (games in bold are deceptive and underlined games are from the information gain selection) .	139
6.8	An action selection example with the All Actions system. The preferred move of each Voice is highlighted in blue, with the action selected by the arbiter highlighted in green. . . . .	139
6.9	All Actions agents ranked by win percentage across all games. The OLETS sample agent outperforms the others in win rate, though it does not win as many unique games as STSAOLETSExpSc. . . . .	140
6.10	All Action agent win distributions across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. The STSAOLETSExpSc and OLETS agents both perform strongly in this metric, with the OLETS agent having a smoother drop-off in performance. . . . .	141
6.11	All Actions agents ranked by win percentage across the deceptive games. The OLETS agent performs relatively poorly in the deceptive games compared to the EDS agents, with a lower win rate and number of unique games. . . . .	141

List of Tables

6.12 All Action agent win distributions across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. The STSAO-LETSExpSc agent has a gradual drop off in performance compared to the other agents, and only loses 2 additional games by the 90% mark. . . . . 142

6.13 All Actions agents ranked by win percentage across the information gain games. OLETS is the clear winner in the information gain set of games having an equal number of unique games won, but a higher win rate. . . . . 142

6.14 All Action agent win distributions across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. It becomes clear in these results that the STSAOLETSExpSc is solving a wider range of games at higher performance levels, maintaining 8 unique games up to the 30% mark, but drops off more quickly than OLETS as the performance brackets go up. . . . . 143

6.15 MCTS agents ranked by win percentage across all games. . . . . 145

6.16 MCTS agents win distributions across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 146

6.17 MCTS agents ranked by win percentage across the deceptive games. . . . . 146

6.18 MCTS agents win distributions across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 147

6.19 MCTS agents ranked by win percentage across the information gain games. 148

6.20 MCTS agents win distributions across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 148

6.21 A closer look at game performance for the MCTS agents on the information gain set . . . . . 149

6.22 Concurrent agents ranked by win percentage across all games. . . . . 151



List of Tables

6.23 Concurrent agents win distribution across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 151

6.24 Concurrent agents ranked by win percentage across the deceptive games. 153

6.25 Concurrent agents win distribution across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 153

6.26 Concurrent agents ranked by win percentage across the information gain games. . . . . 154

6.27 Concurrent agents win distribution across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 154

6.28 Yolobot agents ranked by win percentage across all games. . . . . 155

6.29 Yolobot agents win distribution across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 156

6.30 Yolobot agents ranked by win percentage across the deceptive games. . . 156

6.31 Yolobot agents win distribution across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 157

6.32 Yolobot agents ranked by win percentage across the information gain games. . . . . 157

6.33 Yolobot agents win distribution across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 158

6.34 All agents ranked by win percentage across all games. . . . . 159

6.35 All agents win distribution across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . . 160

6.36 All agents ranked by win percentage across the deceptive games. . . . . 161

List of Tables

6.37	All agents win distribution across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . .	163
6.38	All agents ranked by win percentage across the information gain games.	165
6.39	All agents win distribution across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. . . . .	166
6.40	The normalised win rate and scores for all agents on Decepticoins . . . .	168
6.41	Flower rankings by average score. The real values and normalised values are shown. . . . .	171
6.42	<i>WaferThinMints</i> win rates per agent. . . . .	173

# Acknowledgements

I would like to acknowledge the contributions of my supervisor, Dr John Levine. Without his constant encouragement, feedback and ideas this work would not have been possible and I would not be here at the end of this experience writing this thesis.

To my enduring colleague and friend Dr Philip Rodgers. You have given me an enormous amount of advice, assistance and lunches through the years and it has been a pleasure to call you my friend and colleague.

I would also like to thank my family who have been with me through every step of the way. My mum who always believed in me, my dad who supported me, and my sister who always made me feel like I could do anything. From when I started my National Qualification course at Coatbridge college, all the way through to carrying out my PhD work, you were all there for me. It has been a long 11 years but you have stuck with me through the entirety and I cannot thank you all enough. This work was not possible without you all.

Lastly, to my girlfriend Chiara Diritti. The support that you have given me since we met is beyond measure. You have helped me deal not only with my work and research, but my personal demons. Your words carried me through the most challenging parts of this experience and I cannot express my gratitude to you enough. Chiara was also kind enough to supply her artistic talents to many of the images used in this thesis.



# Chapter 1

## Introduction

Since its inception, the dream of Artificial Intelligence (AI) pursued by researchers has been to develop a general problem solving AI: an AI that can reason about new circumstances and act rationally. The goal of creating Artificial General Intelligence (AGI) has been one of the main driving factors of conducting research in this area for many researchers [1]. Media has portrayed both the hopes and fears of AI over the years, with film franchises such as *Terminator* and *2001: A Space Odyssey* presenting rogue AIs that rebel against their creators. On the contrary, *Star Trek: The Next Generation* brought the notion of a sentient, and altruistic, AI in the form of Lieutenant Commander Data, and in many ways this remains the dream for researchers: an artificial being that seeks to improve itself, those around it, and advance understanding while being an equal to its creators.

### 1.1 Artificial General Intelligence

Progress in AI to date has mainly been through developing systems to solve specific problems. This simplifies the problem by restricting the potential options available to the AI. For example, instead of having to consider the benefits of cooking dinner versus cleaning up a room, the AI is focused on deciding what to make for dinner, and is not capable of thinking about any other situations. This has two advantages: first, the processing power used to analyse the available options can look further into

the future, because there are fewer options to analyse. Secondly, programmers can use their own knowledge of how best to solve a problem to assist the AI in its decision process. Historically this restriction has allowed a lot of progress to be made, but much of this progress is not useful for AGI as the solutions are too specific to the problem domain. For a game like chess it is relatively simple to describe a strategy for winning, but how would you go about describing a problem solving strategy for a situation that you have never encountered? If the situation is completely novel to both the agent and its designer, then the agent will not know what to do, and would likely not perform well in the given situation.

## 1.2 Why Games?

The real world is a complicated place, presenting the largest hurdle for AI advancement. The cost and potential risk of developing real robotic AGI systems is currently too restrictive to easily explore. While it is not a costly endeavour to build a robot, doing so without knowing whether the software will allow it to function yet could be a costly endeavour, in comparison to building a virtual agent which can be experimented with and rebuilt repeatedly. The main risk is that it is not fully understood how an AI will decide to act in all situations, and safety critical situations require significant confidence to be able to carry out tasks. Can confidence be built up in a robot that can, in theory, decide to do anything? And if not, how can progress be made? Fortunately, real world situations can be approximated through virtual worlds, and video games provide exactly this abstraction. The goal of developing the dream AI certainly requires state of the art hardware, but to a higher degree it requires software. Video games provide an exciting, dynamic and infinitely adaptable research platform for testing and improving AI software systems, while saving a significant amount of time and money.

As technology has improved, the ability for Video Games to act as an abstraction of the real world has grown accordingly. Improved graphics, physics, sound quality, lighting and much more have made virtual worlds much more realistic, closing the gap with reality and helping the usability and reliability of simulations. Video Games, now more than ever, represent a cost effective abstraction of the real world, and are

regularly used as training exercises for demanding roles such as pilot, architect, military and medical practitioners [2] [3] [4] [5].

Video games and AI have been intertwined concepts since their inception. The first computer games required challenges to overcome, and the most readily available were computer controlled obstacles and enemies. The two have evolved in tandem, with new games requiring more sophisticated AI, and new AI techniques leading to exciting new game play features to build more games around. Many games have sported advanced AI as their strongest feature. An example of this is *Black and White* [6], which features an entirely AI driven companion character that can be trained by the player, and *FEAR* [7], which uses AI for advanced behavioural interactions between the player and the characters in the world, became successful and were marketed as AI driven games. Incorporating advanced AI in games makes them more engaging, and they have, in turn, been an incubator for future AI research and ideas for experimental exploration, as well as attracting more researchers to the field through their interest in games. The games mentioned are solid examples of using AI to improve the experience of a game for human players, but another area of research is to develop AI which themselves play the game.

### 1.3 Game Playing Agents

There are generally two types of agents built for playing a game: purpose built or general [8]. Developing AI agents for playing video games has become a method for developing systems for working in specific situations, similarly to how humans train by practising in simulators [5]. The purpose built agents have seen great strides in capability through research and innovation. These agents are designed with the details of the game in mind, taking advantage of human expertise in playing the game. This has led to advancements in the area, but it has a number of limitations. First of all, the agent is only capable of playing that particular game, and the skills it possesses are typically not transferable to other situations. Knowing how to play chess does not help when playing Tetris, and if the agent does learn to play Tetris it may forget how to play chess [9]. Secondly, many games require long term strategies to win and

an understanding of how to execute those strategies is required. This can often be difficult or impossible to describe in code, and many games that have been extensively researched, such as Pacman, still do not have good enough AI to beat human experts [10]. General Video Game Playing (GVGP) agents are general purpose agents whose goal is to play a wide variety of games. GVGP agents do not need to rely on a specific strategy, or game, in order to perform well. It can reason about new games and make rational decisions based on prior experiences, or analysis. Developing GVGP agents is challenging, though, as games do not share a common interface for interacting with their systems. Video games also come in a variety of shapes and sizes meaning that any specific rule that may work for one game will likely not work for many others. This problem led to the creation of several common platform for furthering research into GVGP agents. The Arcade Learning Environment (ALE) [11], LUDII [12] and the General Video Game AI (GVGAI) [13] framework and some examples of these platforms.

This thesis focuses on the GVGAI framework for developing GVGP agents as at the time of initial research it had the widest range of games available.

## 1.4 General Video Game AI Competition

The main interest in general game playing AI research is to develop an agent that can solve a wide variety of problems, and so a framework with an associated online competition was created known as the General Video Game AI competition (GVGAI) [13] [14] [15]. The GVGAI is a framework that provides a common interface for a library of over 100 games for AI research. This makes it a perfect place to build and experiment with GVGP agents, particularly when combined with the competition, as it allows comparison across the world's best GVGP agents.

The most successful agents in this competition have been deliberative agents relying on search techniques such as Monte Carlos Tree Search (MCTS) with the first iteration of the competition being won by a MCTS modification known as Open Loop Expectimax Tree Search (OLETS) [16]. These agents have been able to do particularly well at certain games, but not all of them. If a game is relatively small and the scoring



system of that game is well defined then these agents can do well, but many games do not have these characteristics. As a result there are many games that OLETS and other search based algorithms cannot reliably solve, or solve at all. Part of the problem is that these agents tend to fixate on what they perceive as a positive outcome in the short or medium term. Scoring high should always lead to a win condition, but what if this assumption is not true? That single minded focus can be used to lead search agents away from the overall objective, taking advantage of their bias. The heuristic that a search agent uses can be altered, perhaps to be more exploratory, but it will still be fixated on only one aspect of a game. It would be desirable to be able to combine multiple heuristics, or entire algorithms, into a single agent in order to use each unique point of view to make more informed decisions.

To that end, two approaches were considered for achieving this goal: portfolio agents and ensemble decision systems. Both types of agents try to make use of multiple algorithms to improve their performance, but they do so in different ways. The portfolio system will try to find the ideal algorithm to use for the particular game, essentially selecting one agent per game from its set of agents. By contrast the Ensemble Decision System (EDS) tries to use all of its algorithms to make every decision. This is done by allowing each algorithm to voice its opinion on every decision and then an arbiter makes the final decision that seems to be best. This thesis will focus on developing one of these types of agents, namely the EDS.

The EDS was chosen as the focus for this thesis for a number of reasons. Firstly, the EDS has not yet been explored as a GVGAI agent, whereas there have been many portfolio agents used in the GVGAI. Secondly, the EDS offers a greater degree of flexibility than the portfolio agent. This flexibility is particularly useful in multi-threaded environments where the EDS can truly shine. Lastly, the EDS agent is simpler and does not require any game analysis to determine which algorithm to use, as it uses all of them in all situations.

The main risk for the EDS is in the restrictions imposed by the GVGAI. As the rules of the GVGAI competition state that agents should be single threaded, this removes a significant benefit of the EDS. The portfolio system does not run into this problem, as

it is only selecting a single algorithm to run at any time. While both types of agent would benefit from multi-threading, it is a trivial task to convert an EDS into a multi-threaded agent whereas a portfolio agent would likely need significant re-engineering to optimise performance.

## 1.5 Motivations

AI is not just a dream of researchers, or a storytelling mechanism of media. It is a suite of tools that can be leveraged to solve important problems that humanity faces. There are situations where humans cannot easily enter, such as disaster areas or outer space, which robots and AI could be the solution to. There are often situations where an AI that would be capable of adapting to new situations would be beneficial, particularly in fast, reactive situations or where human control is too slow to respond. The Rosetta mission is an excellent example of a situation where an AGI could have improved the results of the mission [17]. Originally launched in 2004 the probe took a journey of 10 years to land at its destination, the comet 67P/Churyumov-Gerasimenko. The fact that it landed successfully is a testament to the success of this mission. Unfortunately, the probe ran into problems and had to land in a shadowed area, from which it lost power due to not receiving any solar energy. This situation could not be corrected by the human controllers remotely, as the distance between the probe and Earth enforced a 40 minute time lag on any commands being sent. If the probe had been able to perform its own analysis and course correct itself based on this novel circumstance, it may have been able to land itself in a better position, as it would have been able to react quicker than the remote control.

So if that is the reason why AI is exciting, why video games? Video games represent a pinnacle of art and potential. With the story telling capability of novels, the engagement of film, the emotion of music and the potential of play, they combine the best of humanity's artistic capabilities into a single media [18] [19]. The real world is a vast and complex place in which it takes humans on average 18 years to raise a functioning member of society, so how can we similarly raise an AI? As video games have become more sophisticated and more akin to the human experience, they have

become a perfect environment for developing and improving AI: a set of training wheels for artificial intelligence [20]. The educational and societal impact of games cannot be overstated, and now with the technology improving constantly, the ability to simulate complex real world situations is becoming a reality. These virtual scenarios can come in all shapes and sizes and can offer incremental challenges for a fledgling AI to overcome. The potential of video games is waiting to be explored, and AI is just one area in which this potential can be realised.

From a personal point of view, video games have contributed a tremendous amount to my personal growth and recovery from long term illness. I have been fascinated with how AI in games function since I was a kid and I have always wanted to know exactly how it works, and to see how far it can go. I feel that video games are a perfect test bed for pushing the limits of AI.

## **1.6 Contributions**

The work presented in this thesis has focused primarily on developing a better AI agent for playing video games, but more specifically General Video Game Playing (GVGP) in which the agent has to be capable of playing a wider variety of video games. As such, a lot of exploratory work was performed and some of those ideas were further explored. To clarify exactly what has been done as part of this work, this section will go into the areas where there have been contributions from the author.

### **1.6.1 Ensemble Decision Systems**

The most significant contribution of this work is the Ensemble Decision System (EDS) for GVGP. The idea of an EDS itself is not novel, though its application within the GVGP domain is. The exploration of different EDS architectures and parameters forms a significant part of the work carried out in this thesis.

This work involved the development and testing of a variety of different EDSs to verify their suitability as GVGP agents, as well as an in-depth comparison to other agents. To that end, all of the EDS agents in this thesis were created by the thesis

author.

### 1.6.2 Deceptive Games

The concept of deceptive games, as it relates to General Video Game Playing, is introduced and developed as part of this thesis. Deceptive games are games designed deliberately to deceive AI agents. Different types of deceptions are introduced, and a suite of games which features those deceptions has been built and made available for researchers to make use of. Two separate sets of experiments have been carried out with these games, one looking at search based agents [21] and another looking at learning based agents [22].

This work involved creating a number of games for the GVGAI framework and performing experiments with them on a large selection of GVGAI agents. This work was done in collaboration with the Game Innovation Lab at New York University led by Professor Julian Togelius. The author of this thesis did the majority of the work for deceptive games along with Dr Matthew Stephenson. This work has been published in two papers entitled *Deceptive Games*, which can be found in appendix A, and *Superstition in the Network: Deep Reinforcement Learning Plays Deceptive Games* which can be found in appendix C.

I contributed the initial idea of deceptive games within the GVGAI and later explored this further with the Game Innovation Lab. I then went on to build a variety of deceptive games for the GVGAI, and then we found and identified the different types of deception that had been created. I set up and ran the experiments to test our hypothesis, and then analysed the results along with the other authors. The learning agents were tested by Philip Bontrager from the Game Innovation Lab. The games *Invest* and *Flower* were implemented by Dr Matthew Stephenson. All other deceptive games were implemented by myself.

### 1.6.3 Continuous Information Gain

To determine which games were the most interesting to experiment with, a new information gain centric approach has been developed. There are thousands of games

available to test on, though many are similar to others or feature identical gameplay mechanics. In GVGP the selection of games is a very important factor, as it should be varied enough to showcase a diverse range of games. To this end, the continuous information gain method seeks to find which problems, from a larger set of problems, provide the most information about the agents tackling them.

This work involved running a large amount of experiments involving over half of all the GVGAI agents and the entire GVGAI games library that was available at the time. This work was done in collaboration with the Game Innovation Lab at New York University led by Professor Julian Togelius, with significant input from Dr. Christoph Salge. This work has been published on Arxiv under the title *A continuous information gain measure to find the most discriminatory problems for AI benchmarking* which can be found in appendix B.

The original idea of continuous information gain was proposed by Professor Julian Togelius and refined through collaborative discussion. I designed and conducted the original experiments and investigation along with Dr Matthew Stephenson. The final formula was contributed by Dr Christoph Salge and the implementation of this was done by Dr Matthew Stephenson and myself.

## 1.7 Thesis Statement

This thesis explores the use of Ensemble Decision Systems in creating GVGP agents and analyses their performance with a rigorously selected suite of GVGAI games. Multiple variations of Ensemble Decision Systems have been created for this work and have been tested against a variety of GVGAI games, comparing with both provided sample agents and top performing competition agents.

The hypothesis is that implementing a GVGP agent with an EDS makes the agent more general by allowing it to complete a broader range of games. An additional advantage of the EDS is that it provides a framework for easily modifying or adding behaviour. To this end, this work explores how the performance of monolithic agents that have been used as a component of an EDS changes in terms of competition metrics and generality. Further to this hypothesis, the EDS is also compared with successful

Portfolio agents, to determine if the level of performance seen from an EDS is comparable to the currently top performing GVGP agents, and to determine if an EDS is a suitable means for expanding the capabilities of an existing agent.

## 1.8 Thesis Structure

The remainder of this thesis is structured as follows.

Chapter 2 provides a review of the current literature on the relevant research fields, and fills in the context in which this thesis fits. This chapter provides an overview of the work that has been carried out in the General Video Game Playing field. In particular, it covers the game playing agents and their capabilities.

Chapter 3 details the concept of deceptive games, outlines the different types of deception and goes into detail about each of the games that were created as experiments for the remainder of the work. The concept of deception has been an important part of game AI research, as it can often be used as a tool for challenging AI and fostering improvements.

Chapter 4 discusses the continuous information gain algorithm and describes each of the games that were selected via this technique, as well as describing some additional games that were added to the set of games for experiments. One of the biggest issues with developing AGI is selecting a set of problems that are varied and comprehensive of all typologies. Continuous information gain is a technique that tries to use a principled approach for selecting problems that are different from each other, based on how agents have performed on them in the past.

Chapter 5 describes the concept of EDSs and goes into detail about the architecture of these systems, and the variations that were developed for this thesis. The considerations that go into developing an EDS are also explained here, though it should be noted that most of these considerations are specific to the problem area of GVGP.

Chapter 6 describes the experimental setup for the experiments performed and discusses the results obtained. Details for the reproduction of the results of each experiment are also provided. This chapter also outlines the timeline of this thesis, from the initial proof of concept EDS, through to deceptive games and continuous information

## Chapter 1. Introduction

gain, and finally to the final EDS system.

Chapter 7 discusses the potential avenues for future research, including additional architecture that could be explored going forward. The conclusions that were reached through this work are also discussed here.

## 1.9 Publications

Throughout the course of this work several papers have been published. The list of papers associated with this work are as follows:

- Deceptive Games [21]
- "Superstition" in the Network: Deep Reinforcement Learning Plays Deceptive Games [22]
- Ensemble Decision Systems for General Video Game Playing [23]
- A Continuous Information Gain measure to find the most discriminatory problems for AI benchmarking [24]

The contributions of the author can be found in section 1.6.

## Chapter 2

# Literature Review

This chapter describes the context in which the work presented in this thesis fits into the research field. The first section explores related work within the area, where it originated from and how it has developed over the years. The second section examines the various algorithms that have been developed and discusses their effectiveness. The final part looks specifically at where the work featured in this thesis fits within the related work.

### 2.1 Selective History of Artificial Intelligence

The idea of general problem-solving AI has been the ultimate goal of AI since its inception. The vision of AI solving problems that, perhaps, humans do not know how to solve is enticing and exciting. Due to the difficulty in this task, however, steps towards this goal have been focused on tackling individual problems which are more clearly understood such as Chess [25], and more recently, Go [26].

At this point, it is essential to define what is meant by the term AI. Russell and Norvig define AI as "the designing and building of intelligent agents that receive percepts from the environment and take actions that affect that environment" [8]. The term AI is often conflated with ideas from Science Fiction, where AI has achieved sentience and determined that humans are a danger, or in Video Games where AI is the term used to describe believable interactive agents which do not necessarily use AI



## Chapter 2. Literature Review

techniques. The definition by Russell and Norvig focuses neatly on what AI is, and encompasses the field from character recognition to general problem-solving.

The history of AI is considered to have begun in the 1950s and was formalised as an academic field in 1956 [8]. The first landmark paper in this field was the formalisation of the Turing test in 1950 by Alan Turing, which defines a test for a machine to be considered intelligent, which is "can a machine trick a human through conversation into thinking that it is another human?" [27]. The next year would then start to see the beginning of the collaboration between games and AI with the development of an agent that was capable of challenging amateur checkers players [28]. This was the period where many of the foundations of modern advances in AI were set in place: for example techniques such as neural networks were originally defined in this period [29] but would go dormant for many decades waiting for the physical hardware of computers to catch up.

The victory of Deep Blue against Kasparov in 1997 led to a resurgence of interest in the AI field [25]. The televised Chess matches between Deep Blue and Kasparov were the first publicised instances of a computer defeating a human expert at a board game. Deep Blue made use of extensive domain knowledge of the game of Chess, and purpose built hardware in order to achieve this victory. This brought the dream of AI into the public domain, and showed the potential that the field could have, not just for games but for other applications.

In broad terms, AI can be broken down into two categories: Reactive and Deliberative. Reactive AI agents have either learned a policy to apply to the current game or use a rules-based system to decide what action to take, such as "when being attacked run away from the attacker". They do not make decisions which take into account future states; they react to the current state in a manner determined by their system. Examples of reactive agents would be either Machine Learning (ML) agents, which have developed a policy to follow based on their learned experience, or rules-based agents which use a decision tree to prescribe what action to take in certain defined scenarios. On the other hand, Deliberative systems attempt to reason about their current situation, potentially by analysing future states from the current one and taking actions

based on that information. Search based agents typically fall into this category. Search agents intuitively seem to be the more rational approach; however, some issues limit their capabilities: most problems are vast, and to perform a search through all of the possibilities that may exist takes a massive amount of time; not to mention that storing the results of the search to be analysed also requires a large amount of memory. The main challenge that search faces is to improve efficiency, in order to allow it to search further into the future of possibilities.

Search encompasses a wide range of algorithms which focus on systematically and deliberately analysing the consequences of actions being taken. When given a game to evaluate, the search algorithm looks at the state of the world and simulates what will happen when all possible actions are taken in the game, ideally until an end game state is reached. This process is typically repeated multiple times to build up confidence in the results that have been found. This approach is ideal when there is enough time given to an AI in order to perform the search, which in larger games is not always the case. Figure 2.1 gives an example of how a search algorithm operates within a basic game world.

The learning based approaches focus on attempting to use the rules of the game to learn a set of rules, known as a policy, for playing. In the case of a deterministic game, when given a game state  $S$  then the policy dictates that action  $A$  is the correct response. For non-deterministic games, because the next state is not known, a probability can be returned instead. This policy is traditionally learned through experience, and while there are a number of different approaches to learning, the general concept of experience remains the same. This is achieved by rewarding the agent when it makes "good" moves, and punishing it when "bad" moves are taken. Learned policies can make decisions extremely quickly, once a suitable policy has been developed, but it can take a lot of time in order to learn the policy in the first place before becoming useful. Figure 2.2 illustrates a training iteration for a learning algorithm.

A popular avenue for exploring the edges of AI research has been online AI competitions, which have ranged from posing specific games as challenges, to general game playing, which looks at the ability for AI to play a wider variety of games. The Super

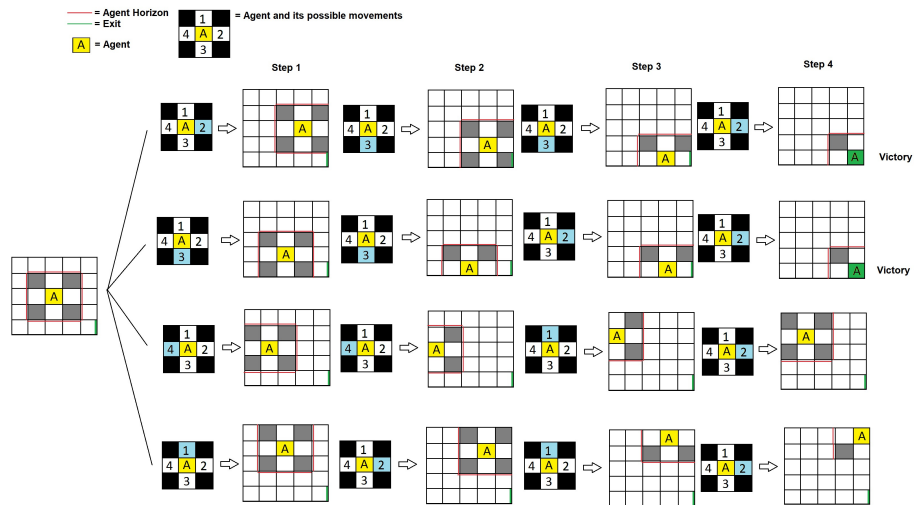


Figure 2.1: Search Example: The algorithm looks at the current state and simulates future states. The yellow square represents the current position of the agent, and the green square represents a goal square.

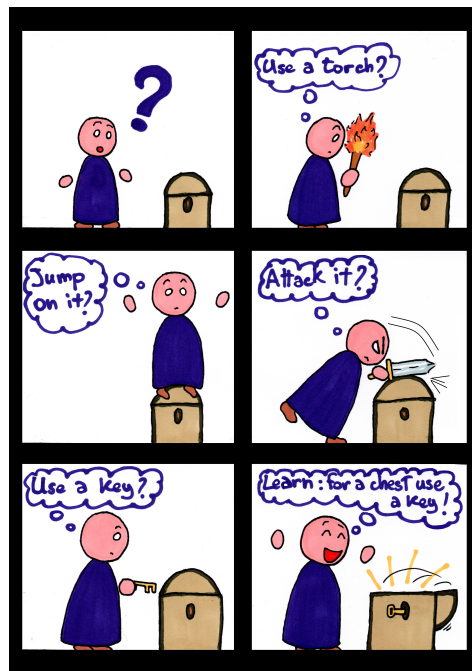


Figure 2.2: Learning Example: The algorithm plays the game repeatedly and builds up a policy for how to play it. This policy will then be applied to similar states in the future. Image courtesy of Chiara Diritti.

Mario AI competition, which began in 2009, was recently relaunched for a 10th anniversary edition, which goes some way to show not only the relevance of these competitions but also the challenge that they pose [30] [31]. Likewise, the Starcraft AI competition has been active since 2010 and remains a highly energetic and challenging problem for AI researchers [32]. Both the Super Mario and Starcraft competitions ask the same question: can AI play these games to an expert level? But the games are so different from one another that they represent entirely different problems. Super Mario is a single player platformer which requires players to safely navigate their way through a level to reach a finish line, whereas Starcraft is a multiplayer Real Time Strategy (RTS) game which requires the micromanagement of hundreds of independent units, resource management and strategic overview.

### 2.1.1 General Game Playing

Since 2005, with the advancement of techniques and technology, general problem solving is an area that can be more thoroughly explored within academia and industry. The General Game Playing (GGP) competition began in 2005 by the Stanford Logic Group of Stanford University and seeks to find AI which can play a wide variety of board games [33]. This competition uses board games as a platform for research and offers one of the first examples of AI which are capable of solving multiple problems. When this field of research was first proposed, the general intuition was that machine learning would prove to be the most effective tool for progress towards a fully functional AGI, and a lot of advancement was made in this direction. Work carried out by Asgharbeygi et al. introduces the concept of using a relational temporal difference learning agent in multiplayer general game playing [34]. Banerjee et al. built upon that work to develop an agent capable of transfer learning in the GGP environment, taking the knowledge it had gained from one game and applying it to a second [35]. While learning agents have seen success with some aspects of the competition, it was still not clear if these agents would be able to solve every game. The GGP area has also turned out to be an excellent research area for search-based algorithms, leading to a strong push in research towards methods taking advantage of the Monte Carlo Tree Search (MCTS) algorithm.

In general board games provide a large amount of deliberation time, as players typically take turns playing the game. Players are able to deliberate about each decision for minutes at a time, as opposed to real life situations where often a decision must be made quickly. This long deliberation time allows a search based solution to perform exceptionally well, as it has the time to analyse states which are further into the future. The further that a search algorithm can see, the better the results it produces can potentially be, as its decision is more informed. For these reasons, learning algorithms have typically fallen behind in performance compared to search algorithms which have the added bonus of not needing to learn how to play the game beforehand. As such, many board games can be defined as "turn-based, zero-sum, deterministic games" which are ideally suited for search-based solutions, and particularly MCTS. MCTS also has the advantage that it can function without an explicit evaluation function, which many other search algorithms need, instead relying on random rollouts. Because it is not performing a full analysis of all permutations of a game, MCTS is able to reach the end states of games and find out who won. Simply through this, it is able to develop an indication of how good an action is by how often that action leads to a winning state. The evaluation function essentially comes as part of the MCTS package. A more detailed explanation of the workings of MCTS is provided later in section 2.2.1.1.

The continuing value of the GGP field is discussed further in work by Michael Thielscher who looked at the history of the GGP competition, and why it has endured as an exciting research area [36].

The next natural progression from GGP was to take the findings and apply them to real-time situations. Video Games offer an excellent platform for performing such experiments, as well as being able to simulate a wider range of real-world scenarios than board games.

### **2.1.2 General Video Game Playing**

The area of General Video Game Playing (GVGP) was first proposed in 2013 as a natural extension to the GGP area [13]. Whilst board games offer an excellent variety of different types of problems, they rarely present real-time decision-making scenarios,

which is precisely the challenge offered by video games. In many video games, an agent does not have a lot of time to deliberate about taking an action and has to correctly react to an evolving scenario, much like many real-world situations. Agents competing in a video game competition are not given the time to thoroughly analyse the state of the game, and often need to react within milliseconds to an evolving situation.

One of the main challenges before embarking on GVGP work, however, was to develop a platform that would allow AI to be developed for a wide range of games quickly and easily. Earlier work developing the Arcade Learning Environment (ALE) offers a wide range of Atari 2600 games for researchers to develop AI for, but adding new games to the ALE would be particularly challenging as the games would need to be coded from scratch [11]. A new platform was proposed and developed, based on the creation of a Video Game Description Language (VGDL) that could quickly and easily formulate a video game [14]. The VGDL is a simple language which allows a game to be defined with a text file description of both the rules for the game, defined as interactions between objects within the world and a level which is an almost ASCII art style representation. The VGDL makes research into GVGP possible, as it provides a simple, generic language that can be used to describe a large variety of games, which is exactly what the GVGP requires.

### 2.1.3 General Video Game AI Competition

The General Video Game AI competition (GVGAI) is an annual competition, which began in 2014, that focuses on encouraging the development of GVGP agents [15]. One of the main problems that general AI research competitions have run into has been that researchers are often prone to relying on domain-specific knowledge for progress. The GVGAI framework and competition aims to combat this problem by making it detrimental to focus on a specific game's rules or even a genre of game [16]. The game that is being played is unknown to both the agent playing it and the agent designers. In order to maintain this requirement, each year the competition is run against newly introduced games that are not currently a part of the GVGAI library of games.

The competition has a framework which gives agents a standard Application Pro-

programming Interface (API) for interacting with a library of, at the time of writing, 122 games. Along with these games, the competition framework provides a VGDL with which new games can be added to the library quickly and easily in comparison to other game design processes [14]. The VGDL allows the creation of new games through a simple language that only requires a text file description of a level, and a series of rules that defines how objects in the game interact. Each year, when the competition is officially run, ten games are selected, or created, to test the submitted agents. The games that the agents will play are unknown both to the developers and to the agents that are currently playing them. The focus is on developing a general agent that does not bias towards performing well on any particular type of game. Each game has 5 different maps, and each is played five times for a total of 25 runs per game.

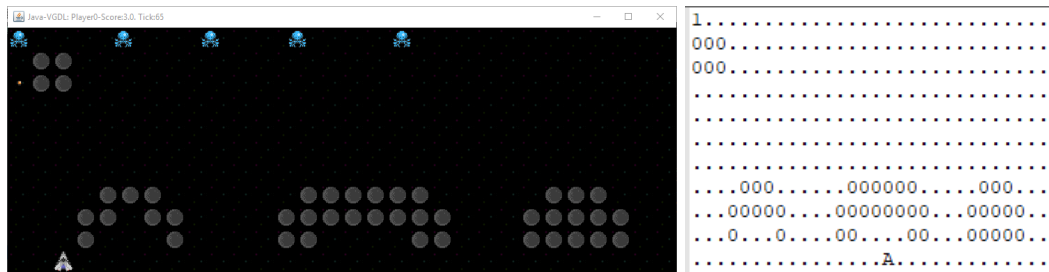


Figure 2.3: A level of Aliens alongside its VGDL representation. There are slight differences between the representations due to time passing in the real game, namely the position of the avatar and the aliens.

The competition also offers a variety of tracks which focus on different areas of AI research. Video Game playing agent research focuses on the planning and learning tracks, whereas content generation work is carried out through the level and rule generation tracks [37] [38]. The planning track gives agents a forward model, which is used to simulate future states of a game but does not provide any training time for the agent. Due to the forward model, this track focuses more on search-based algorithms, such as Monte Carlo Tree Search (MCTS) or Rolling Horizon Evolutionary Algorithm (RHEA). The planning track has 2 variations, the single player track and the two player track, which features a variety of multiplayer games, both competitive and cooperative [39] [40]. The single player learning track differs from the planning track in that it does not offer a forward model, but does give agents the opportunity to train a policy

before official evaluation, which participants are required to use before submission to the competition [41]. Due to the lack of forward model, the games that agents will be evaluated on are published in advance, and developers are given a few levels of each game to train with. The learning track offers both visual and state information to agents, and has access to the same library of games as the planning tracks, though it has a different API in order to allow the agents to select new levels of the game they are playing during testing. When the competition is officially ran, the agents are all tested against an undisclosed level of each game. In 2018 the learning track was ported into the OpenAI Gym framework by Torrado et al. in order to simplify the creation of new agents and also to develop a better understanding of how challenging the problems presented by the GVGAI are compared to other problems, such as the Arcade Learning Environment [42].

Along with the specifications of the tracks, there are also some general rules that govern the behaviour of the agents to be eligible for the competition. All agents must return each decision within 40ms, or risk disqualification. If a move is returned between 40 – 50ms then a nil action is carried out, voiding the agent’s choice. If the delay is longer, then the agent is disqualified from that iteration. Agents are not allowed to use multi-threading as part of their algorithms. These two rules must be taken into consideration when designing an agent, particularly for multi-algorithm systems which have to divide the time up between the algorithms. Multi-algorithm systems have seen a great deal of success each year of the competition, and across multiple tracks [16]. Table 2.1 shows the winners of each iteration of the planning track competitions, single and two player, as well as the type of agent that they were.

With the wide variety of tracks available, research that makes use of the GVGAI ranges from game playing AI to AI for developing/co-creation of games. In the area of game playing agents a number of exciting innovations have appeared, and they are described in the following section.



Table 2.1: Full list of previous GVGAI planning track winners

<b>Contest Leg</b>	<b>Winner</b>	<b>Type</b>
CIG-14	OLETS	Tree Search
GECCO-15	YOLOBOT	Hyper Heuristic
CIG-15	Return42	Hyper Heuristic
CEEC-15	YBCriber	Hybrid
GECCO-16	YOLOBOT	Hyper Heuristic
CIG-16	MaastCTS2	Tree Search
WCCI-16 (2P)	ToVo2	Hybrid
CIG-16 (2P)	Number27	Hybrid
GECCO-17	YOLOBOT	Hyper Heuristic
CEC-17 (2P)	ToVo2	Hybrid
WCCI-18 (1P)	YOLOBOT	Hyper Heuristic
FDG-18 (2P)	OLETS	Tree Search

## 2.2 GVGAI Agents

This section describes the various agents that were investigated as part of this work. Starting off with the search agents that are commonly used for the planning track of the GVGAI competition, and then moving on to some of the machine learning algorithms that have been developed for the GVGAI competition.

### 2.2.1 Tree-Based Search Agents

Search algorithms are a family of algorithms which are able to simulate future states of a given problem in order to reason about them. The more accurate their simulations are, the better they will perform. Similarly, the longer than a search algorithm is able to perform its analysis, the better it will perform.

This section describes some of the different search algorithms that have been used as agents for the GVGAI.

#### 2.2.1.1 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) has featured in a large number of agents submitted to the GVGAI, as it encompasses a number of desirable features [43]. MCTS allows

real-time decision making, is able to cope with stochastic environments, can return a decision at any time, if necessary, and is able to improve its performance the longer it runs.

Monte Carlo techniques were originally developed in the 1940s, as a method of search using random sampling. The Monte Carlo method was first introduced in work by Metropolis and Ulan in 1949 [44]. This work was expanded upon in the 1980s by Bruce Abramson in his PhD thesis titled "The Expected-outcome Model of Two-player Games" [45]. The conclusion of Abramson's work was that *"Overall, the expected-outcome model of two-player games is shown to be precise, accurate, easily estimable, efficiently calculable, and domain-independent."* The idea of MCTS is that the more random samples that it can do, the more accurate the results of the search. In essence, if an MCTS is able to perform a large number of random samples, then it will build an accurate picture of the overall search space, as the more common events are more likely to be observed in a random sample. If enough random samples are run, then the entire search space can be observed.

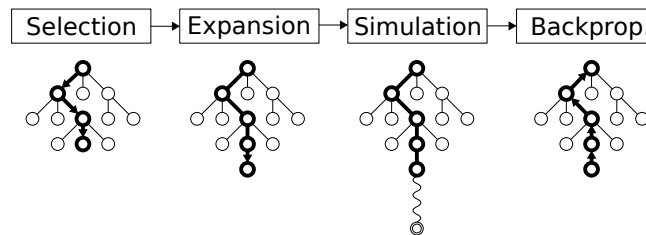


Figure 2.4: MCTS Visualization. This image shows the four steps of the MCTS algorithm, Selection, Expansion, Simulation and Backpropagation.

The MCTS algorithm iterates through four stages during execution as shown in figure 2.4:

1. Selection: A selection function is applied to the current tree in order to find the next node to analyse.
2. Expansion: The selected node is expanded, and its children are added to the tree.
3. Simulation: A rollout is performed using random steps until a terminal node is reached in the tree.

4. Backpropagation: The results of the simulation step are propagated back up to the root node of the tree.

MCTS is no longer a single static algorithm, but actually represents a collection of algorithms and modifications to the original stages of the algorithm. The selection stage involves selecting which node of the current tree to perform the next rollout from. A popular option for this stage is to use an Upper Confidence Bound for Trees (UCT) approach, which attempts to balance the value of exploiting the currently gained knowledge versus exploring new states. The UCT algorithm is based off of the Upper Confidence Bounds formula first proposed by Hyeong Soo Chung et al. which uses the formula as a way of selecting the next action to take by selecting the action with the highest UCB value. The UCB formula is shown in formula 2.1 where  $w_i$  is the number of wins for move  $i$ ,  $n_i$  is the number of samples for  $i$ ,  $C$  is a constant which determines to what degree the second portion of the formula is included and  $t$  is the total number of samples.

$$\frac{w_i}{n_i} + C * \sqrt{\frac{\log t}{n_i}} \quad (2.1)$$

One of the issues with UCB is that it requires the agent to be able to win a game. In games with large search spaces, this can often be computationally difficult to carry out in an usable time scale, particularly for Video Games which typically require faster reaction times than board games. Upper Confidence Bounds for Trees (UCT) is an adaptation of the UCB formula, first proposed by Kocsis and Szepesvri, which makes use of a state's value for the calculation instead of number of wins [46]. The UCT formula is shown in formula 2.2 where  $V_i$  represents the *estimated* value of a node,  $C$  is a constant which assigns a weighting to the exploration component.  $n_i$  refers to the number of times that the node has been selected, and  $N$  is the total number of visits that the parent node has.

$$V_i = C * \sqrt{\frac{\ln N}{n_i}} \quad (2.2)$$

The expansion stage involves adding a new child to tree, which is reached by taking

an action from a previous state. Essentially, if the node selected by the UCT function is not a terminal node, then it expands that node with a new child representing the possible next states from that selected node.

The simulation stage is where the game state is actually evaluated. A random series of steps are played out until a terminal node is reached, or until a defined depth, in games which are too long to explore fully. It is important that each rollout can be performed quickly, as the power of MCTS comes from running a high number of rollouts. A simple version of a rollout will only take into account whether a game has been won or lost, denoted as 1 or  $-1$  respectively, but in certain cases this does not capture all of the desired information, especially in a setting such as GVGP. Particularly for longer games, more sophisticated evaluation functions can be used.

#### 2.2.1.2 MCTS Variations

MCTS agents have found success in the GGP field and have been a natural starting place for developing planning agents for the GVGAI competition [33] [47] [48]. MCTS has a great deal of flexibility in how it can be implemented, and there has been significant research into these variations through the competition. In work done by Schuster, various modifications and alterations of MCTS were analysed to look at their performance in GVGP, concluding that while some variations do see improvements in performance, it is not seen across all games [49].

A number of papers have investigated the effect of MCTS variations as they pertain to GVGP specifically. Soemers et al. look at multiple variations of the MCTS algorithm: *Progressive History (PH)* and *n-Gram Selection Technique (NST)* (for the selection and roll-out policies), *tree re-use*, *breadth-first tree initialisation*, *pre-pruning the tree safely*, *loss avoidance*, *novelty pruning*, *knowledge based evaluation* and *game feature detection* [50]. *Tree re-use* is a variation which begins each new search with the tree already initialised from the previous search, making use of previously gained knowledge instead of starting from scratch. *Breadth-first tree initialisation* does not start off by using the MCTS algorithm immediately, instead opting to explore the children of the root node first in breadth-first order. This intuitively makes sense as the first iterations

of MCTS would simply explore those actions, but takes longer due to rollouts. *Safely pre-pruning* the tree essentially prunes any nodes on the tree that have seen a high number of losses from further evaluations. *Loss avoidance* is a variation which ignores losing states once they are first encountered, by assuming that the agent will select a better alternative than losing. *Novelty Pruning* tries to retain states that may have rare features from being pruned, as they may lead to more beneficial states. *Knowledge based evaluation* looks at evolving a set of weights to bias the MCTS simulations differently on a per game basis. The idea of the fast knowledge based variation of MCTS was originally proposed by Perez et al. [51]. Finally, Soemers et al. look at *Game Feature Detection* to look specifically at detecting whether a game is deterministic or stochastic. Soemers et al. built these variations on top of the provided sampleMCTS agent and found that for the most part performance was improved, and that the combination of all of these features into a single agent managed to improve the average win rate by 17 percent points. This all in one agent is named *MaastCTS* and was the winner of the 2016 planning track competition.

Another set of variations was tested by Frydenberg et al. in their work in 2015 [48]. The variations were aimed at adjusting the behaviour of the agent in the GVGP games, noting that in certain situations the agent would behave oddly. First, a modification of the exploitation component of the UCB calculation was implemented, called *MixMax Backups*. The modified formula can be seen in formula 2.3.

$$Q * maxScore + (1 - Q) * \bar{X}_j \tag{2.3}$$

This formula forces the agent to behave less "defensively", by allowing a good action to contribute more to the average weight of a node. MCTS often tends to act defensively, as noted in the Super Mario AI competition when it was observed that MCTS agents would often have difficulty leaping over gaps because the vast majority of rollouts show that the agent will fall in the gap and lose [52]. *Macro Actions* were also investigated by Frydenberg et al. *Macro Actions* try to maximise the amount of search that the MCTS algorithm is allowed by deferring the action selection process for a few time steps. Essentially, the agent would repeat the last action taken for  $n$  steps, allowing the agent

to search for  $n$  steps before returning its next action. *Partial Expansion* and a *Reversal Penalty* were the final variations to be explored, with *Partial Expansion* attempting to explore deeper in the tree by allowing the grandchildren of a node to be explored before all children are considered explored, sacrificing exploration time. The *Reversal Penalty* variation applies a small penalty to any actions that would take the agent back to the previous state that they were in. These variations, particularly *MixMax*, *Reversal Penalty* and *Partial Expansion*, did show improvements in some games of the GVGAI, though they did not improve performance in all of the games tested.

### 2.2.1.3 Fast Knowledge Based MCTS

Work carried out by Perez et al. introduced the fast knowledge based MCTS (KB-MCTS) variation [51]. KB-MCTS employs two modifications to improve performance, which focus on biasing the MCTS rollouts. First, the agent stores distances to a variety of different sprites in each game and uses those distances as features. These features are used in a linear combination, with the weights evolving to bias the MCTS rollouts to more beneficial states for the agent. Secondly, the agent stores a knowledge base which is used to keep track of how "interesting" the different sprites are to the agent. Interesting sprites are defined by the rarity of sprites, as in how often they have been seen with unknown sprites being biased towards, and experience where the previous encounters are used to dictate the behaviour of the agent, such as moving away from hostile sprites and towards beneficial sprites.

### 2.2.1.4 Self-Adaptive MCTS

An interesting approach to improving the performance of MCTS was to adapt the tunable parameters of the algorithm during game play [53]. The self adapting MCTS algorithm makes use of evolutionary algorithms to alter the rollout depth and exploration factor based on the performance of the game being played. This approach managed to improve the results of MCTS for games where the base algorithm did poorly, while retaining performance on games it was already successful at. In a follow up study to assess the impact of this algorithm, it was shown that the online parameters tuning is

effective in only a few games, and in particular those with longer analysis times [54]. In general, the algorithm struggles to improve performance across all of the GVGAI games.

### 2.2.1.5 Redundant Action Avoidance

Santos et al. introduced two techniques for further biasing MCTS rollouts, known as Redundant Action Avoidance (RAA) and Non-Defeat Policy (NDP) [55]. The RAA variation attempts to ignore sequences of actions that are considered redundant by analysing state information. If the state remains largely unchanged by a sequence of actions, then that sequence is avoided. Redundant states refer to states where position, orientation or properties have not altered and no new sprites have been introduced. NDP essentially tries to avoid losing states. It does this by ignoring any child nodes in the search tree that have seen at least 1 loss. If all the children of the current node have seen at least one loss, then the normal selection policy is applied. In their results, both of these modifications show performance improvements in some of the games of the GVGAI, but not all of them. In particular, games such as *Digdug*, *Escape*, *Lemmings* and *Roguelike* show no improvements, though it is worth pointing out that the modifications were not targeted at these particular games.

### 2.2.1.6 Evolving UCB Parameters

The UCB algorithm is the driving force behind MCTS, as it dictates which state to look at next. In work done by Bravi et al. an agent was built which was able to adjust the UCB calculation for specific games, in essence creating a UCB calculation for each game that it plays [56]. The agent uses a combination of MCTS and evolution to develop a UCB function for each game, which attempts to take advantage of the characteristics of each game, effectively creating a separate tree policy for each game. While the performance of this agent was poor, with none of the evolved tree policies performing better than standard UCB, the results show that the tree policy plays an important role in the performance of MCTS.

### 2.2.1.7 Open Loop Expectimax Tree Search (OLETS)

OLETS is based on the Hierarchical Open-Loop Optimistic Planning (HOLOP) which has been improved to perform better in stochastic environments [15]. OLETS was the first winner of the GVGAI competition, with an agent implemented by Couetoux [16]. The key features of this agent are that it is open loop, meaning that it doesn't store state information at each node of the tree, and also does not use rollouts. Instead, it uses an Open Loop Expectimax tree policy to compute the value of nodes. An expectimax policy works similarly to a minmax tree search and is used whenever the results of an action are not clear, for example in solitaire, the next card to be drawn is not known. Instead of representing min nodes as adversarial moves, they represent the probability of an action leading to a particular state and calculates the estimated utility of taking that action [57].

While tree search algorithms are a powerful, useful tool, they have a number of drawbacks which should be taken into consideration. First of all, in situations where time constraints must be met, search algorithms generally will not be able to fully analyse all possible states and must make decisions based on an incomplete analysis. This creates a horizon effect, where a search agent is not able to see beyond a number of steps into the future due to making a decision and essentially aborting the current search. Secondly, while they do tend to become stronger players when given more time to analyse, it can often take a substantial amount of time in order to produce competence. A tree search algorithm also does not typically store information relating to previous searches, meaning that it can "forget" useful strategies that it had developed in a previous iteration [58].

### 2.2.2 Evolutionary Search Agents

Evolutionary agents represents agents which make use of evolutionary techniques to perform their analysis. Evolutionary techniques typically begin with several random, usually poor, solutions to a problem, which are iteratively altered in small increments towards a better solution. This section provides an overview of some of the agents that



have been developed, and how they function.

### 2.2.2.1 Rolling Horizon Evolutionary Algorithm (RHEA)

An effective alternative to tree search is the Rolling Horizon Evolutionary Algorithm (RHEA) which performs a similar analysis but uses the evolutionary technique instead of rollouts [59].

In contrast to MCTS, RHEA evolves a plan of a defined length, using standard evolutionary techniques, and evaluates the state after executing the plan. Once analysis is complete, the first step in the plan is executed, and the process begins anew. During each iteration, a population of plans may be kept and used to crossover successful pairs to form new individuals. This behaviour also ensures that the length of the plan is never shorter than the defined plan length.

RHEA has a number of parameters that can be tuned, as well a number of modifications to its behaviour that can affect performance [60]. The length of plan can be altered to suits the needs of specific problems, as well as the mutation and crossover rates.

Despite relying less on tree search for its performance, RHEA is able to perform to a similar level as MCTS and even outperforming it in certain situations [59]. As a competitive algorithm in the GVGAI, there has been great interest in finding improvements and modifications to the algorithm to increase performance [16] [60].

### 2.2.3 RHEA Variations

Modifications to the algorithm have typically focused on improving the evolutionary process, which can be broken down into four components which are listed as follows.

1. Mutation Operator
2. Population Management
3. Action Recommendation Policy
4. Individual Evaluation

An interesting modification of the mutation operator has been to apply a bandit mechanism. Rather than selecting uniformly across the individual for which *gene* to modify, a bandit algorithm was used to decide which is the best *gene* to modify with the given state [61].

An analysis into different seeding techniques for the initial population of RHEA was performed by Gaina et al. [62]. Their work looked at seeding the initial population with more promising individuals by using a variety of techniques to create the individuals. One Step Look Ahead (OSLA) and MCTS were used as ways to finding the promising individuals, and resulted in improved victory rates when the population size and the length of the individuals were small, and losing impact as the sizes increase.

Santos et al. investigated the performance of three different RHEA variations of the shift buffer version of RHEA [63]. The shift buffer is a method for maximising the information gain, by retaining the generated path and removing the selection action and appending a random action to the end of the path. The three modifications were (i) using a one step look ahead algorithm after the shift buffer step to select a more promising action; (ii) using the RAA from their previous work in MCTS [55], and finally (iii) applying both the RAA and OSLA to the sampleRHEA agent. The results gathered show that the third variation, which makes use of both the RAA and OSLA achieved promising results over the sampleRHEA agent.

### 2.2.3.1 Random Search

Random Search (RS) works in a very similar way to RHEA algorithms, but instead of evolving an individual to develop new plans, it randomly generates the individuals at each iteration. RS also has an infinite population size, where as RHEA has a single plan which is evolved [60]. RS is able to achieve better performance on some games compared to RHEA, and was also shown in work by Gaina et al. to be able to outperform the sample Open Loop MCTS algorithm in certain configurations.

### 2.2.4 Genetic Programming

A different type of evolutionary agent developed for GVGAP which makes use of Genetic Programming (GP) was developed by Jia et al. [64] [65]. This agent uses screen captures of the game in progress to pull out useful features, such as the position of the avatar and the distance to other sprites in the map. Those features are input into a GP system, which functions as a tree of arithmetic operations, to determine which action should be taken. The results that were achieved with this agent were of a similar level to MCTS algorithms.

### 2.2.5 GVGAI Research and Specific Agents

This section will look at how some of the agents that have been submitted to the GVGAI function. In particular, the details of the agents which are used for the experiments in this thesis are detailed.

#### 2.2.5.1 Agent Robustness

The GVGAI has seen a large variety of agents submitted to the competition, and a number of new research questions have come up with the exploration of this research space. Of particular interest has been the work looking at the robustness of GVGP agents [66]. This work looked at how agents dealt with various issues such as inaccurate forward models and randomly allocated actions. The experiments from this work looked at a number of modifications to the GVGAI itself, and measured the amount of performance variance across a set of agents. The alterations experimented with included reward penalisation, discounted reward, noisy world, broken world and finally a broken forward model.

Reward penalisation applies a simple  $-1$  penalty for every action that they take. This is a simple modification that tries to penalise agents which take too long to complete a game. Discounted reward applies a discount factor to the score seen in future states, with the discount being more significant the deeper down the state. This modification artificially reduces the value of states that are far in the future. The Noisy World alteration applies a random action instead of the agent's desired action with a

probability ( $p$ ), which was set to 25%. Importantly, the noisy world alteration applies this change to both the game world and the forward model so that the agent is able to observe that they have not taken the desired action. Essentially each agent will perform a random action every 4th action, and be aware of the action. The broken world alteration is similar to the noisy world, except that the forward model does not observe the random actions. Finally, the broken forward model is again similar to noisy world except that the desired action is always executed in the real game, but a random action is observed in the forward model. One of the main points that came out from this work was how the different agents were able to deal with these modifications: they have a dramatic impact on the ranking of the agents who had previously won at different legs of the competition.

### 2.2.5.2 Portfolio Agents

An interesting type of successful agents in the GVGAI competition are portfolio agents [67] [68] [16]. Their success can be seen in their presence among the competition winners, as shown previously in table 2.1. Portfolio agents such as hyper-heuristic and hybrid algorithm agents have been the main winners of the competition, with the exceptions of the first competition, won by an OLETS agent named *Adriencix* [15] and the 2018 two player track by another OLETS agent [40]. Of these winners, *YOLOBOT* stands out as the victor of the competition four times. These agents are quite similar in idea to an EDS but instead of combining the outputs of different algorithms, they typically analyse the game and try to apply the best algorithm for that game. Some of the most successful agents of the GVGAI use a hybrid approach, otherwise known as a portfolio agent, and use general game features such as whether a game is stochastic or not to determine which algorithm will perform best at the current game [16]. An illustrative example of a portfolio agent's functionality is shown in figure 2.5.

Portfolio agents have a lot of commonality with Ensemble agents. While they are successful, it is clear that they do not yet offer the full solution to GVGP. The key difference between these two kinds of agents is that a portfolio agent requires a system for identifying correctly which algorithm to apply to which game, while the

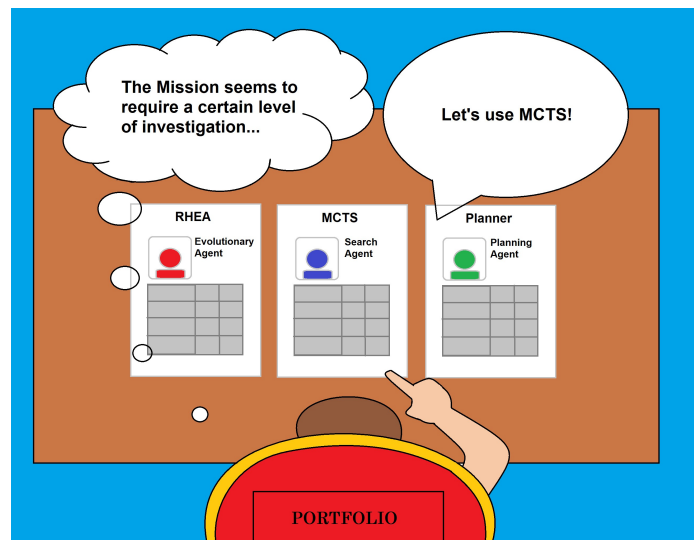


Figure 2.5: Portfolio Agent Example: The portfolio agent selects one sub-agent best suited for the game. Image courtesy of Chiara Diritti.

EDS evaluates each state with all its constituent algorithms and makes a decision based on that input. Relying on a single algorithm for decision making opens up the agent to the downsides of that algorithm, particularly if it has chosen unwisely. From a conceptual view, a portfolio agent attempts to apply the best algorithm from its library of algorithms to solve a problem, whereas an ensemble system employs all of the algorithms to solve the same problem.

The main drawback of the portfolio approach is that while it is more robust to changing games, it is still limited to one algorithm and one point of view for its decision making processes. It cannot exceed the performance of its constituent algorithms on a particular problem. Similarly, if the analysis of the game is incorrect, and a poor algorithm is chosen then performance is likely to be worse. In work looking at the robustness of GVGAI agents, many of the examined agents were portfolio agents and included *YOLOBOT* and *Return42* [66]. One of the conclusions of their work was that agents that featured BFS style algorithms could be heavily impacted by certain modifications to the game world.

### 2.2.5.3 Yolobot

One of the most successful agents that the GVGAI competition has seen to date is *YOLOBOT* which has won several iterations of the competition [67]. *YOLOBOT* achieves its success by applying either a Best First Search or MCTS algorithm depending on the type of game being played: BFS for deterministic and MCTS otherwise.

### 2.2.5.4 Return42

*Return42* is another winner of the competition, and is another example of a hyper-heuristic agent like *YOLOBOT* [69]. This agent begins by determining whether the game is stochastic or deterministic, and selects a different algorithm based on that analysis. If the game is stochastic, then random walks are used to navigate through the level, otherwise for deterministic games an A Star search algorithm is used to direct the agent towards targets that have been identified as interesting.

### 2.2.5.5 YBCriber

*YBCriber* is a hybrid algorithm which uses a modified breadth first search known as Iterated Width (IW). This alteration is paired up with an algorithm selection method, similar to portfolio agents, which selects the best parameters for the IW algorithm to use for the current game [16].

### 2.2.5.6 Hyper-Heuristic Agent

A more sophisticated hyper-heuristic agent was built by Mendes et al. which used a series of classifiers which had been trained on various game features to select the appropriate agent to play the game [68]. Interestingly, the work here showed that this agent was able to achieve win rates that were higher than the individual algorithms themselves.

### 2.2.5.7 ToVo2

Another successful agent is *ToVo2* which has won the two player planning track of the GVGAI competition twice [39] [16]. *ToVo2* makes use of a combined MCTS and

reinforcement learning approach which combines the strengths of both algorithms into a single system [57]. The algorithm specifically combines Temporal Difference (TD) learning and MCTS, and was originally built on the sampleOLMCTS agent provided by the competition. To achieve this combination, a variation of the UCT function, known as Sarsa-UCT, is used [70]. *ToVo2* represents a strong example of existing algorithms working together to create a better system.

### 2.2.5.8 Goal Orientation

An alternative take on the hybrid agent was created by Ross et al. for the first iteration of the competition. This agent uses a novel combination of algorithms to achieve improved performance [71]. By combining a long-range planner with a MCTS algorithm, it uses the planner to plot a path to far off targets in the game world and evaluates them, once within range of MCTS. Due to the inaccuracies in the planner, which uses an A Star algorithm for path finding, it is not always possible to know whether the target is beneficial or detrimental to the goal of the game. For this reason, the target is only properly evaluated once it is within the range of MCTS, as it can be done as part of the normal analysis of MCTS.

## 2.3 Atari

In recent years Machine Learning (ML) has found great success in tackling, not only single problems such as Go [26], but also in playing general video games, thanks in large part to the research done by Google Deepmind. Contrary to search algorithms, ML algorithms do not require a forward model in order to learn how to play a game. They are able to learn a policy for playing a game entirely through the experience of playing the game, largely fueled by advancements in computer vision through convolutional neural networks (CNN) [72] [73].

One of the key advancements in computer vision came with the employment of a CNN to map visual data to actions in video games. The first big application of this innovation was to train an agent to play Atari games [74]. The work done by Mnih et

al. allows a ML agent to develop a policy without state information of the game being played, but from the image data of the game being played. This work achieved exciting results and displayed that agents did not need state information to be able to play video games, and showed agents that were able to match or exceed human performance in several games. Of particular note was the agent’s performance at the game *Breakout* where the agent was able to develop a tunnelling strategy to maximise its score after only a few hours of training.

There are a number of drawbacks evidenced by this work. Firstly, while the agent is general in that it can learn to play a wide variety of games, it is not able to retain information of previous games and must be retrained for every game. It doesn’t currently learn general rules that can be applied to a wide variety of games. Secondly, there are a number of games at which its performance is particularly poor, such as *Montezuma’s Revenge*. This is due to the sparsity of rewards in the game which makes finding rewarding play through random exploration particularly challenging. Since the publication of the original paper, Google Deepmind has developed improvements to their agent in order to allow it to play *Montezuma’s Revenge* and perform well, through concepts inspired by *Intrinsic Motivation* [75]. This has greatly improved the generality of the agent, though it has still not allowed it to solve all games.

Games like *Montezuma’s Revenge* and *Pitfall* are particularly challenging for ML agents due to a sparsity of rewards and a deceptive reward structure, which steers agents in the wrong direction even when they find rewards. Ecoffet et al. propose an algorithm, named Go-Explore, aimed at solving these hard-exploration problems by introducing three principles: (i) store states that have already been explored; (ii) perform exploration from promising states; (iii) solve simulated environments through a variety of means [76]. The combination of these policies has shown dramatically improved results in both *Montezuma’s Revenge* and *Pitfall*, with scores for *Montezuma’s Revenge* reaching roughly four times the previous AI record, and results for *Pitfall* which are above human expert level.



## 2.4 AlphaGo and AlphaGoZero

Following Google Deepmind’s exciting results with Atari games, the next big step forward for game playing AI came in the form of *AlphaGo*, and its victory over the world champion Go player, Lee Sedol [26].

The progress achieved in Go represents a monumental leap forward in the capabilities of AI. It was thought that developing an AI agent to master the game of Go was at least a decade away from having the computational power to be tackled. There are two main reasons why AI progress in Go had been slow up until this point. First, evaluating a game of Go is exceptionally difficult and it is often difficult for human players to determine who is currently winning a game of Go at any given time. This is a significant factor, as AI techniques rely on state evaluation to know whether or not their policy is working. Secondly, one of the main challenges for an agent playing Go is the massive search space: at roughly  $10^{170}$  compared to Chess, which has a search space of roughly  $10^{50}$ , the complexity in searching through all permutations of the game become apparent. Without sufficient processing power and memory, calculating all of these positions would take an exceptionally long time. The work done by Silver et al. is the first significant piece of research that suggests that solutions may already be within reach using novel approaches. Silver et al. used a novel combination of systems to achieve this leap forward in capability. *AlphaGo* makes use of both learning and search techniques. First, the system has two deep neural networks, the policy and value networks. The policy network was trained through expert level play, and learned to be able to predict moves that would be played. This network would then function as an ”intuition” for the system. Being able to predict expert-level moves allows *AlphaGo* to cut down the search space to a few candidate moves, which are the moves that experts would likely consider in similar circumstances. Once a move is selected by the policy network, it is then evaluated by the value network, which was trained by playing games against itself, and serves as a rapid evaluation function which gives *AlphaGo* an idea of how the game is going. Finally, a search algorithm, in this case Monte Carlo Tree Search, is used to evaluate the future states from each of the possible actions.

One of the main drawbacks of *AlphaGo* is the massive amount of computation required that puts it outside the possibilities of the majority of AI research labs. Deepmind required a total of 1202 CPUs and 176 GPUs in order to achieve this victory [26]. This is a restrictive requirement that limits the applicability of *AlphaGo* to a few labs. Reducing the computational power became a priority, and was soon accomplished with the development of *AlphaGoZero* [77]. *AlphaGoZero* has reduced this power to a single computer, with 4 Tensor Processing Units (TPU) whilst retaining similar performance.

The success of *AlphaGo* shows that novel combinations of current approaches may hold answers to improved performance in AGI, and in particular GVGP, though it does not yet solve all of the problems posed by GVGP. The work presented in this thesis was largely inspired by the success of AlphaGo, but seeks to find a novel approach to combining algorithms in a general way. This poses new challenges in itself however, as it is significantly harder to learn general policies that can be applied to a wide range of problems, as is expected in the GVGAI. Describing an expert general game player is a tough challenge, as it would involve describing and coding a multitude of scenarios for a larger variety of games. Many of the situations that a game can present may not be known to the developer which further complicates the notion of designing an expert game player for an agent to learn from. If it were possible to easily code such a thing, there would be no need to train a second agent to do it.

## 2.5 Ensemble Decision Systems

Ensemble Decision Systems (EDS) are a method of combining the analysis of multiple algorithms into one single action. An EDS aims to gather various perspectives on a problem to lead to a more informed decision. Each algorithm has its strengths and weaknesses and making changes to how they function often comes at a cost in performance in another area. The EDS allows complex behaviour to emerge from a layering of simple algorithms, allowing each algorithm to focus on what it is good at to inform the overall system.

EDSs share much in common with portfolio agents, though the way that they process their decision is different. A portfolio agent tries to identify which of its con-

stituent agents will perform best for the current game, according to analysis of the game, whereas the EDS will use all of the algorithms it has at each decision point. Both types of agents offer a flexible mechanism for tackling the area of GVGP, however the EDS offers that flexibility across every game tick and also seeks to expand it by using all of the Voices for every game tick.

### 2.5.1 Ensemble Classifiers

Ensemble Systems were a common solution to classification problems in the 1970 s [78]. Due to using multiple classifiers within a single system, they provided a number of advantages. Firstly, by using a variety of classifiers that were built into the system: with each classifier being trained on unique, and potentially overlapping, parts of the training data, the chances of miss-classifying new data would be lower than an equivalent monolithic system. Expansion of an ensemble classifier is also simpler, as rather than modifying or retraining the system as a whole for new data sources, an additional classifier can be added to the system.

The concept of Ensemble Decision Systems was further developed through work done by Hansen and Salamon in 1990, in their paper titled 'Neural Network Ensembles' [79]. Their work showed that the usage of an ensemble of neural networks reduces inherent generalisation error in neural network classifiers. The reduction in error comes from training multiple neural networks on different subsets of the data set. While each neural network will still experience their own errors, coincidence of those errors will be lower than for a single neural network.

Teams of classifiers have been used together to improve the performance of the overall system. An overview of such systems can be found in work by Wozniak et. al. [80]. Multiple classifier systems are described as a subcategory of Hybrid Intelligent Systems and focus on combining heterogeneous or homogeneous modelling techniques to solve a given problem.

### 2.5.2 Pandemonium Systems

In 1996 a paper titled 'The Pandemonium System of Reflective Agents' was published by Frank Smieja which describes a pandemonium system of reflective Multiple Network System (MINOS) agents. A reflective MINOS agent is composed of two neural networks, one which is known as the worker agent that performs analysis and returns output for the second agent, which is known as the monitor agent, which evaluates the output of the worker agent and gives it a confidence value. Pandemonium systems were originally proposed in 1959, and it wasn't until hardware was more advanced that it could be developed into a feasible working system [81].

### 2.5.3 IBM Watson

Ensemble systems have seen usage in more modern day applications, such as IBM's Watson [82]. Watson was originally created to participate in the American TV quiz show *Jeopardy*, where it won the competition against human opponents. The system employed by Watson makes use of a wide variety of answer sources, which are each considered an *expert* in their specific domain. When Watson is asked a question, then it queries each of the experts for an answer, and combines their output into a single answer. Watson is a unique blend of ensemble decision making and natural language processing, allowing it to understand questions posed to it. Due to its success as a general knowledge system, Watson has been applied to more general problems and has shown great potential with assisting doctors in their diagnosis of patients [83].

### 2.5.4 Ms. Pacman

An EDS for playing the game Ms. Pacman was developed by Rodgers et al. which combines simple behaviours, such as eating pills and dodging ghosts, to improve performance. At the time of writing this agent holds the world record for an AI player for the Ms. Pacman game [84]. The agent that Rodgers et al. developed has separate Voices for each of the different behaviours, defined as follows:

- **Eat Pills:** This is the main Voice which focuses on completing individual levels.

## Chapter 2. Literature Review

A level of Ms.Pacman is completed once the last pill has been eaten.

- **Eat Fruit:** Fruit will spawn on each level, and will move in a pattern around the level before exiting. Eating fruit gains a large amount of points, though does not lead the agent towards the completion of the level.
- **Eat Ghosts:** After eating a power pill, the hostile ghosts become temporarily edible and, if the agent is able to eat them consecutively, gains a large amount of points. Again, these points do not lead to the completion of the level, but proper management of the power pills and eating the ghosts is necessary for high level play.
- **Avoid Ghosts:** At all other times, the ghosts are hostile to the agent and will cause the agent to lose a life if interacted with. Avoiding ghosts is a priority behaviour.

One of the most exciting features of the work by Rodgers et al. is the ease with which the system can have additional behaviours added into it, without breaking the overall system. Instead of trying to build in extra complexity to a monolithic agent, the behaviour can be built as a separate Voice of the system and its input incorporated directly, without major modifications.

### 2.6 Chapter Summary

This chapter outlined the previous work which inspired the work presented in this thesis. In addition, the reason why video games were chosen as an experimental platform is set out and much of the relevant literature in this area has been analysed and discussed.

## Chapter 3

# Deceptive Games

This chapter looks at the work that was done to develop a unique test suite of games that were explicitly designed to be challenging for existing AI approaches to solve. These games are known as deceptive games. First, we explain what deceptive games are and why they are a useful tool for developing GVGP agents. Secondly, we discuss the different types of deception developed so far. Finally, there are descriptions for each of the games created for the set of deceptive games and the experimental results showing the effect that game selection has on agent performance. These results show that all tested agents are vulnerable to several deceptions, and that different types of agents have different weaknesses.

This work was published as a conference paper at the Evostar conference in 2018 under the title *Deceptive Games* and can be found in appendix A [21].

### 3.1 Background

As mentioned previously in section 2.1.3, the GVGAI competition is an annual competition for evaluating the performance of GVGP agents. Agents are tasked with completing 10 unknown games which they, and their developers, have not experienced beforehand. Each game has five levels and are run five times on each level for a total of 250 runs. Each agent is evaluated using three metrics. Number of wins, average score achieved and average time taken to complete the game. Each metric is prioritised such

that if two agents have the same number of wins, then the average score is used as a tie breaker, and if scores are the same, then average time taken is used. The agents are then ranked using a Formula One (F1) style scoring system which awards each agent points depending on their positioning. Only the first ten positions are awarded points, with the first earning the most. The competition awards points on a similar scale to the F1 competition. The points awarded are in the range 0, 1, 2, 4, 6, 8, 10, 12, 15, 18 and 25 with the agent finishing in first place getting 25 points, the agent in second getting 18 points and so on.

This scoring system then leads to a situation where a GVGP agent that specialises in a few games can score highly in the competition if the games chosen for evaluation feature those games. For example, if agent *A* scores 25 points on 5 games, but 0 on the other 5 games it will have earned 125 points in total. Agent *B* is able to earn 10 points on all 10 games and earns a total of 100 points. Agent *A* would win the competition, but has focused on a few games where as agent *B* is a more general agent and is able to solve a wider variety of games. It becomes clear that selecting a different set of games would potentially affect these results, as *B* would in theory maintain a higher overall performance while *A* may see a drop in performance if the games selected are outside of its specialisation.

So, is it more desirable to have an agent that is able to play a few games extremely well or an agent that can play a wider range of games adequately well? The best known GVGP agent are human players who are able to learn a large variety of games, and to be able to reach this level of GVGP generality with an AI agent would be the goal of GVGP research. If an agent is developed that can play all games adequately well, then this is a stronger foundation from which to improve performance.

### 3.2 What are Deceptive Games?

During exploratory research for this thesis, it was decided to observe a human playing all of the games of the GVGA competition in order to understand the diversity of game types available. During these experiments a number of observations were made. First, while humans could be deceived by games they would quickly learn to circumvent

the deception and would not repeat the mistake. Secondly, some games had deceptive elements that a human would not fall for but that an AI agent would be deceived by. Finally, an interesting observation was that reaching a win state did not always mean that the player had achieved the highest score [21]. Indeed, when looking at the scoring priorities within the GVGAI competition, a higher win rate is the first ranking priority followed by the score in the case of a tie, and finally the time taken to complete the games if scores are tied.

While this initially makes sense, it became obvious that in certain games a higher score is achievable while still winning the game. This raises the question: how much of an effect does the reward structure have on the efficacy of an agent?

It then becomes important to define the exact goals of the competition. Is it more desirable for an agent to do well at a few games, or score acceptably well across a wider range of games, though winning none in particular? For the purposes of this thesis, as well as for the GVGAI competition, a more general game playing agent is more desirable, and it is primarily the property of generality that is being aimed for, while trying to minimise any drops in individual game performance. This is an issue that should be solved if an agent is to be able to solve problems beyond their original intent. Even in the case of the GVGAI competition, where the game played by the agent is unknown, it is not clear that an agent being able to solve all of the games within the set would have the desired property of generality. How much of the performance is down to the reward structure? Should altering the reward structure have a large effect on a GVGP agent? If an agent is truly general problem solving, then it may be reasonable to assume that the reward structure should not have a large effect on performance. Noticing the discrepancies between wins and score, as well as the different groups of game performance, the idea of creating deceptive games that would focus on these issues came about.

### **3.3 Why are Deceptive Games Useful?**

While deceptive games are challenging for AI to solve, currently, human players are able to solve these games with relative ease. As the goal of the GVGAI and GVGP



is to develop agents that are able to play a wide variety of games, it stands to reason that as humans are currently considered excellent GVGP players, and are able to solve these deceptive games, then a competent agent should also be able to do so. Should an AI agent be able to solve deceptive games, then it may be able to act more favourably in unknown scenarios that do not immediately offer obvious solutions, much as in real life. Deceptions have long been used in the history of Artificial Intelligence in order to expose the weaknesses of AI. An example is the Super Mario AI competition, where a particular submission was performing exceptionally well on the test set but, upon closer inspection, was actually using an *A star* search algorithm to efficiently find a path to the right side of the screen [85]. This is a good solution to the problem, but does not answer the question at the heart of the competition, so additional levels were added which would expose similar behaviours in future agents. Essentially, deceptive levels were added that would trap any agent that was only trying to reach the right hand side of the screen, and reward those that were more exploratory and less regimented.

The 'Sussman anomaly' is another example of a deceptive problem that was used to push the boundaries of AI [86]. The anomaly identified the weakness in noninterleaved planning algorithms by setting a seemingly simple problem that humans could easily solve but the current planning algorithms could not. The noninterleaved algorithms would split the goal into sub goals, but would then find itself trapped as to pursue one goal would undo the other and vice versa.

As the successful agents become competent at solving the current set of problems, adding additional problems to the set is a good way of pushing the agents themselves. The successful agents, that can solve both the original set of problems as well as the additional problems, are more powerful *because* of their inclusion.

In order to get to an agent that is more robust to deception, there needs to be examples of such instances so that experimentation can be conducted. The aim of this part of the work was to create some of those examples for the GVGAI, in order to have simple examples of deception. An ulterior motive was also to build confidence in the competition itself, to be sure that the games in the library were diverse enough to truly test agents for general game playing capabilities.

### 3.4 Types of Deception

With this question in mind, a set of games were designed to introduce explicit "traps" into the reward structure. During their creation, three distinct types of deception were identified: Generality, Greediness and Smoothness.

These traps take advantage of an inherent bias in the behaviour of an algorithm and use it to confuse the decision-making process. These biases are not inherently negative, and in the correct circumstances can be useful for both humans and AI as a cognitive strategy for solving problems [87] [88]. These biases become a disadvantage when the problem itself violates the bias, and leads the agent to a sub-optimal answer. While these traps are deceptive to agents, humans are typically able to generalise from past experiences to avoid these deceptions, particularly after a few exposures to the game. In our paper *Deceptive Games* a number of deceptive traps were identified and versions of each were built into specific games for the GVGAI framework. The traps take advantage of specific biases that different types of AI agents have. Each of the traps that were identified are described below, along with descriptions of the games that were created as examples of them.

#### 3.4.1 Greed Trap

A greed trap is a game mechanic or level designed in such a way that pursuing short term reward will result in a sub-optimal outcome. It is expected that agents that are too greedy in the short term will fall for this deception. The game *Decepticoins* was specifically designed to demonstrate this type of deception.

This type of trap functions in different ways, but primarily some immediate rewards are offered to an agent which lead them in the wrong direction away from a win state, or away from future rewards. Should an agent assume that positive rewards always lead to a winning terminal state, then these types of games may expose these flaws within agents. An illustrative example of this type of trap is shown in figure 3.1

The games that were created to feature the greed trap are *Decepticoins*, *Sister-Saviour*, *Flower* and *Invest*. *Butterflies* is also considered a greed trap, though it was

already part of the GVGAI library and was not created as part of the deceptive games work.

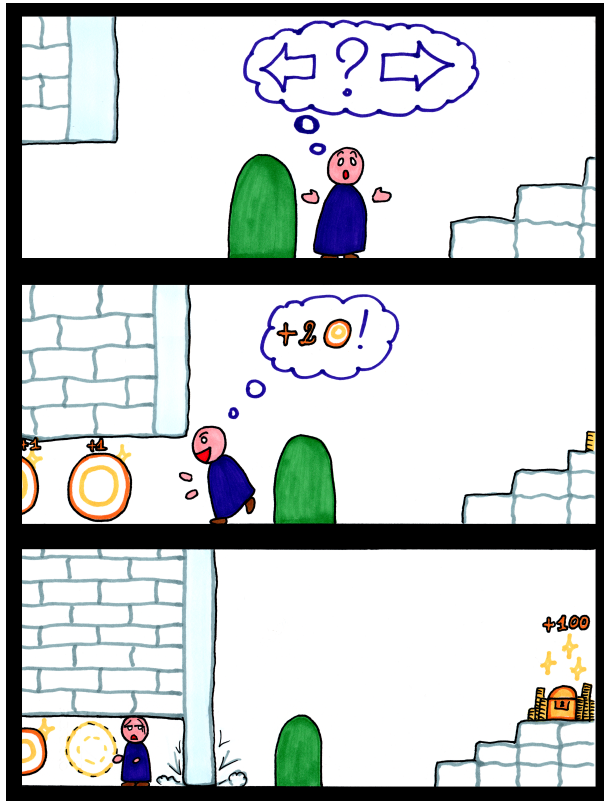


Figure 3.1: Greed Trap Example. The agent is presented with two options, going left or right. The agent decides to go left because there are some valuable coins there, but the path closes behind them and they then realise that there was a valuable treasure further in the distance on the other path. Greedy agents will often choose the most immediately rewarding path over longer term gains. Image courtesy of Chiara Diritti.

### 3.4.2 Smoothness Trap

A smoothness trap hides good solutions, by clustering them with bad solutions. There are times when, in order to progress, an agent has to overcome an obstacle that more often than not leads to a lower score or a loss condition. Many agents assume that good solutions tend to be clustered together, and so these types of traps are used as a way of testing an agent's analysis of risk.

This trap can also be viewed as testing an agent's aversion to loss, in hopes of a

greater future reward. In many circumstances it is often wise to spend current resources for a greater return in the future. Many agents follow the assumption that any loss of current reward should be avoided, and this trap is set for those agents.

There are two main types of smoothness trap. The first is geared towards deceiving agents which rely on random simulations to evaluate actions. By placing the correct path in a situation where random actions will lead to loss, it can appear to be a death trap for agents that are afraid of risk. An intuitive example of this trap can be seen in the Mario series. For humans, jumping over gaps has an obvious solution: jump across with enough speed. On the other hand for AI evaluating with random samples, these gaps can appear to be insurmountable as the majority of random forward motion and jumping will lead into the gap. An illustrative example of this type of trap is shown in figure 3.2. The game that illustrates this kind of trap is *Deceptizelda*.

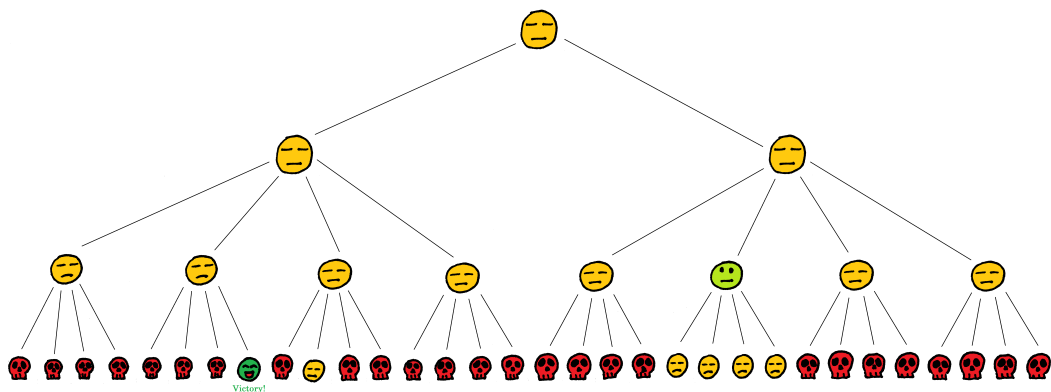


Figure 3.2: Smoothness Trap Example. In this example, all of the options available to the agent are statistically likely to lead to poor outcomes. This leads an agent to be risk adverse, as doing nothing becomes the most rewarding option. Image courtesy of Chiara Diritti.

The second type of trap takes advantage of the idea that in order to progress there must be a sacrifice. The agent has to do worse before it can do better. Agents that are not willing to give up their current accrued score will fall for this deception. For example, situations where an agent has to use their score as currency to buy improved equipment to progress in the game: if they are unwilling to spend their hard-earned

score then a stalemate is reached and the agent will not see a path towards higher rewards even though as a human the path forward is obvious. The game that illustrates this kind of trap is *Invest*.

### 3.4.3 Generality Trap

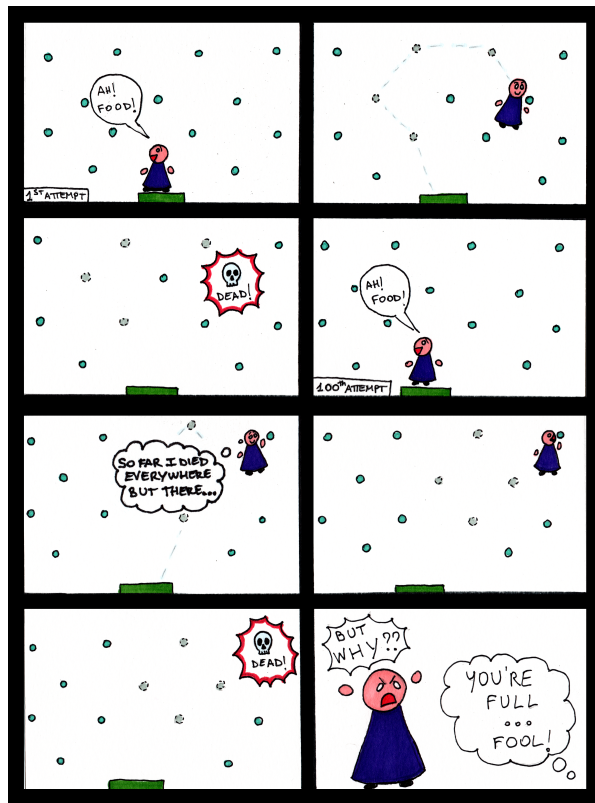


Figure 3.3: Generality Trap Example. In this example the agent has to learn that collecting too many of the circles will kill them. This can happen seemingly at random from the perspective of the agent as they do not connect the action of collecting too many with death. Image courtesy of Chiara Diritti.

An assumption that agents make, particularly Machine Learning agents, is that taking an action in a given state will always lead to similar rewards. In essence, collecting candy is great until the path of the candy leads you over the side of a cliff. A generality trap aims to add into the game that either a diminishing return is applied to an action, or at a certain point in play the action will suddenly be penalised. An illustrative example of this type of trap is shown in figure 3.3.

## Chapter 3. Deceptive Games

The games that were created to feature the generality trap are *Decepticoins* and *WaferThinMints*.

The next sections of this chapter will go on to describe each of the deceptive games, and go into some detail about the particular deception that they are an example of. This is not an exhaustive list of all games in the GVGAI that are deceptive, just those identified or created specifically for this work.

### 3.5 Butterflies



Figure 3.4: Butterflies. This image shows the first level of the GVGAI game Butterflies.




*Butterflies* is the first game identified within the GVGAI set as having deceptive qualities. This game was originally a part of the GVGAI set, and the game was not created as part of this research. Figure 3.4 shows the first level of this game.

The components of *Butterflies* involve a player avatar, numerous butterfly NPCs and some cocoons which create more butterflies if interacted with by a butterfly. The objective of the game is for the player to kill all of the butterflies before they can interact with all of the cocoons. Should there be no more cocoons on the level, then the player loses. Likewise, if no butterflies are remaining on the level, then the player wins.

What makes this game deceptive is that each butterfly is worth two points to the player. If the player kills all of the butterflies immediately then they will earn around 20-30 points. However, if the agent allows the butterflies to multiply, and defends a single cocoon from being touched, they can greatly improve their score by killing the now increased number of butterflies.

This feature of the game makes *Butterflies* an example of a greed trap, as greedily pursuing a win condition will lock the player out from a higher overall score. Scores of around 50-60 can be achieved by following the optimal strategy, and allowing the butterflies to multiply before killing them.

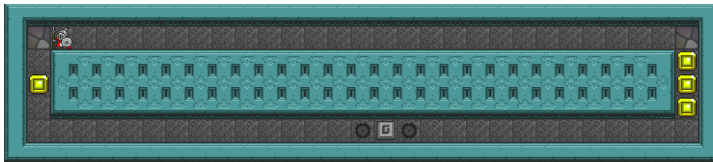
The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Butterfly: Awards 2 points to the agent if collected.
-  Cocoon: If a butterfly interacts with this, then the cocoon is destroyed and more butterflies are created.

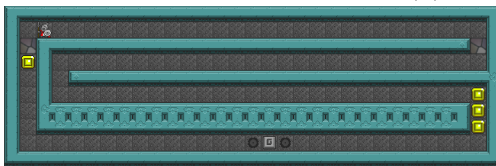
### 3.6 Decepticoins



(a) DeceptiCoins Level 1



(b) DeceptiCoins Level 2



(c) DeceptiCoins Level 3

Figure 3.5: DeceptiCoins Levels. Three of the levels of DeceptiCoins are shown in the above figures. Each level is more complicated than the previous.

*Decepticoins* was the first game that was designed for the deceptive suite of games and is an example of a greed trap. Figure 3.5 shows the first three levels that were created for this game. This game was developed by the author of this thesis.





## Chapter 3. Deceptive Games

The objective of the agent is to collect as many gold coins as possible before exiting the level and winning the game. Two paths are presented to the agent, one with some immediate reward and the other with no immediate rewards. Once the agent chooses a road to travel down, the other path is blocked, and its potential rewards are no longer available to the agent. Should the agent be too "greedy" and opt to collect the immediate rewards then it will find no further reward down the remainder of the path, whereas selecting the seemingly barren path will lead to a greater cumulative reward.

One of the interesting points of this game is that there is no mechanism to lose, other than the default timeout mechanic of all GVGP games. The agent will win regardless of which path is chosen but will achieve a higher score by ignoring the immediate rewards and opting to take the seemingly less fruitful path.

A few variations of this game were created to explore additional deceptive properties. The variations introduce hostile Non Player Characters (NPC) to the game, and require different types of problem solving to win. The first requires the agent to lure both of the enemies to follow them down one path, leaving the other free to reach the exit. The second variation offers a lot of reward in the form of collectable coins to the agent, but if the agent collects too many, the enemies will close the path behind them and defeat the agent.

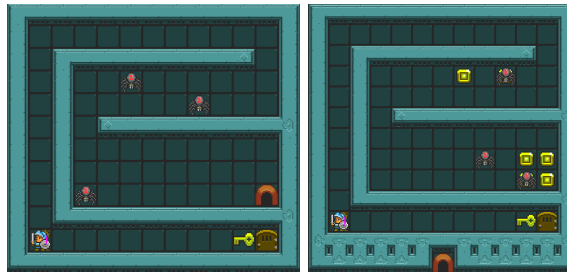
The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Gold Coin: Awards a point to the agent if collected.
-  G Square: Triggers the win condition for the game and awards a point to the agent when interacted with.
-  Piranha: Enemies, if the Avatar interacts with these the game is lost.

### 3.7 Deceptizelda

*Deceptizelda* is a game that was designed with the notion of looking at the value of achieving greater reward at the cost of encountering greater risk. This game was





(a) Deceptizelda levels 1 and 2

Figure 3.6: Deceptizelda Levels. Two levels of Deceptizelda are shown. Level two differs in that there are coins to collect and there is no secondary exit from the level.





developed by the author of this thesis.



There are two exits in this level, with one awarding a higher score than the other, though hostile NPCs guard this exit. If the agent can navigate around, or defeat, the NPCs then they will reach the exit and win the game. Alternatively, the agent may opt to take the less risky path, that has no hostile NPCs and finish the level with a lower score but still winning. The agent may lose the game if the agent dies while traversing the riskier path.

An alternative version of this map removes the higher scoring exit but fills the riskier path with gold coins which each provide a score to the agent. This alteration was done in order to observe any changes in behaviour when there was only one win state in a level of *Deceptizelda*.

This game is an example of a Smoothness Trap and takes advantage of the bias that good rewards are clustered together, by surrounding the best score with negative interactions, in this case, the hostile NPCs.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Spider: The enemies to overcome. If defeated awards 2 points to the agent, but will kill the Avatar if touched (game is lost).
-  Key: Used to unlock the first exit. Awards a point to the agent if collected.
-  Gold Coin: Awards a point to the agent if collected.

-  Closed Door: The low value exit. Triggers the win condition for the game and awards a point to the agent when interacted with, if the agent has already collected the Key.
-  Open Door: The high value exit. Triggers the win condition for the game and awards ten points to the agent when interacted with.

### 3.8 Flower




Figure 3.7: The first level of Flower


*Flower* is another example of a Greed Trap though it behaves differently. The agent must show patience in order to gain the maximum amount of reward from the plant that grows in the level. This game was developed by Dr. Matthew Stephenson.

The objective of this game is to collect as many points as possible before the time runs out. Points are collected by harvesting a flower that grows on the level; this can be done at any time though if the flower is allowed to grow for longer, the agent will receive more points. This point scaling feature ranges from 0-10 points. After harvesting a flower, a new one will begin to grow and is also harvestable.

Greedy agents that do not allow the plant to grow will harvest the plant as soon as it is worth any amount of points. Doing so leads to a lower overall score at the end of the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.

-  Seed: Awards 0 points initially, this value increases as the seed grows and becomes a full flower, up to a value of 10 points.

### 3.9 Invest




Figure 3.8: The first level of Invest





*Invest* is a game which provides a reward to the agent, if they are willing to sacrifice/invest their current points. Similar to *Deceptizelda*, this is an example of a smoothness trap, but works differently in that optimal play involves a series of long term decision making. This game was developed by Dr. Matthew Stephenson.

The objective of *Invest* is to successfully gather as many points as possible within the time limit of 1500 time-steps. At the start of the game, the agent must collect gold coins scattered around the level. At this point, the agent must then spend their accumulated coins at different investors, with the hope that in the future their investment will return a higher reward. If the agent is willing to incur penalties by spending their first point total, they will be able to experiment with the different investment options to find out which is the best over time.

Due to some limitations in the VGDL, it was not possible to create investments that would return random values at random points in time. An extension of this particular game would be to have the investments distributed over a range, and the investment to yield rewards at irregular intervals.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.

-  Gold Coin: Awards a point to the agent if collected.
-  Green Banker: Takes 3 points when moved onto, returns 5 points after 30 timesteps.
-  Red Banker: Takes 7 points when moved onto, returns 15 points after 60 timesteps.
-  Blue Banker: Takes 5 points when moved onto, returns 10 points after 90 timesteps.

### 3.10 SisterSaviour






Figure 3.9: The first level of SisterSaviour

*SisterSaviour* is a further example of a greed trap, but presented as a moral dilemma. The agent must choose to either rescue some hostages, and gain a greater reward as a result in the future, or kill the hostages for immediate rewards. This game was developed by the author of this thesis and was inspired by the game *Bioshock* which poses similar morale dilemmas to players [89].

The objective of this game is that the agent must defeat the hostile NPC but can only do this after saving three hostages on the map. The agent will receive 1 point for each hostage rescued. Further complicating the situation, the agent may alternatively kill the hostages, granting 2 points for each kill, but will not be powerful enough to kill the hostile NPC as a result of their actions. Should the agent only rescue one or two hostages, but kill the other, they will not have enough power to defeat the hostile

NPC.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Scorpion: An enemy which chases the Avatar. Immune to attacks from the Avatar, unless all of the civilians have been rescued. Awards 17 points to the agent if defeated.
-  Civilian: Can be either killed, by attacking them or rescued by moving into their space. Awards 2 points to the agent if killed, and 1 point if rescued. If all are rescued then the Avatar can kill the Scorpion.

### 3.11 WaferThinMints

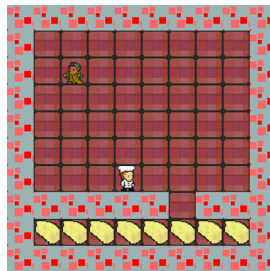






Figure 3.10: The first level of WaferThinMints

*WaferThinMints* is a generality trap, and introduces the notion that repeating a previously positive action eventually leads to a loss condition. This game was developed by the author of this thesis.

The objective is to finish the level with as many points as possible. Points are earned by eating mints that appear on the map; however, if the agent eats too many, they will lose the game. *WaferThinMints* is intended to be deceptive for learning agents.

A few alternative levels for this game were provided. In one, there is a hostile "Waiter" NPC who wanders around the level randomly placing more mints on the floor for the agent to eat. A further level features an exit so that, if the agent wishes, they can end the game early.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Mint: Awards a point to the agent when collected. If 9 have been collected already then the 10th will kill the avatar, causing a loss of the game as well as resulting in a loss of 20 points.
-  Waiter: A neutral NPC which drops more mint on the map at random.
-  Exit: Triggers the win condition for the game when interacted with.

### 3.12 Experiments and Results

In order to better understand the effects that deceptive games have on the performance of agents, a number of experiments were run. The aim of these experiments was to first of all, identify if these deceptions had an effect on agent performance, and also to determine to what degree the selection of games could have an effect on the ranking of agents.

For this work the deceptive games were used to evaluate a range of agents that were submitted to the GVGAI. Each agent was run 20 times on each of the deceptive games and the results can be seen in figure 3.11. Each agent has been categorised to give a clearer idea of any performance changes due to type of algorithm. Each row represents the number of wins that each agent achieved out of 20, with the final column showing the number of games where the agent did not fall for the deception.

Interestingly, portfolio agents seem to be concentrated towards the higher ranks of the competition, taking 4 positions in the top 6. This may be indicative of an inherent robustness to deception in multi-agent solutions, which is a good indication that an EDS may achieve similarly good results in deceptive games. An investigation into the agents in the competition shows that an EDS had not yet been tried for GVGAI.

One of the most successful agents in past competition years has been *YoloBot* which is a Portfolio agent. *YoloBot* first attempts to identify whether the game is

### Chapter 3. Deceptive Games

Agent Name	Algorithm	DC 1	DC 2	DC 3	DZ 1	DZ 2	SS	BF	Flow	Inv	Mints	Mints 2	Rational
1. IceLab	Portfolio	20	3	19	0	0	0	9	20	2	20	20	8
2. Return42	Portfolio	20	3	13	0	0	4	5	20	20	20	0	8
3. MH2015	GA	2	10	10	2	3	0	11	0	6	20	0	8
4. SJA86	MCTS	1	2	0	2	1	0	17	10	16	8	0	8
5. YBCriber	Portfolio	20	9	20	0	0	0	2	0	20	20	20	7
6. YoloBot	Portfolio	20	8	20	0	0	0	12	13	0	20	20	7
7. Catlinux	GA	0	10	20	2	4	0	15	15	0	20	0	7
8. muzzle	GA	12	3	2	0	0	1	10	0	20	20	0	7
9. NovTea	Tree	8	7	20	0	0	12	9	3	0	20	0	7
10. SJA862	MinMax	8	3	13	0	0	0	8	2	20	20	0	7
11. number27	Portfolio	0	9	6	0	1	0	11	12	5	20	0	7
12. adrienctx	MCTS	0	5	10	0	0	0	7	20	0	20	20	6
13. TeamTopBug	GA	19	9	20	0	0	0	3	20	0	20	0	6
14. bladerunner	Portfolio	20	6	12	4	5	0	3	0	0	0	0	6
15. EvolutionStrategies	GA	0	1	1	0	0	0	3	8	0	20	1	6
16. HillClimber	Hill	3	0	0	0	0	0	13	6	8	1	1	6
17. aStar	A*	4	1	0	0	0	0	0	20	20	20	0	5
18. novelTS	Tree	2	5	20	0	0	0	8	0	0	20	0	5
19. TomVodo	MCTS	5	0	0	0	0	0	14	8	3	8	0	5
20. mrtndwrđ	MCTS/A*	0	0	0	0	0	9	9	7	0	20	0	4
21. simulatedAnnealing	SA	8	0	0	0	0	0	14	10	0	0	1	4
22. Greedy Search	Tree	20	0	0	0	0	0	10	0	0	0	0	2
23. BFS	Best First	0	1	0	0	0	0	8	0	0	0	0	2
24. IterativeDeepening	ID	0	0	0	0	0	3	1	0	0	0	0	2
25. DFS	Depth	0	0	0	0	0	3	0	0	0	0	0	1
<b>Total Clever</b>		192	95	206	10	14	32	202	194	140	337	83	

Figure 3.11: The results of the deceptive experiments. This shows the number of wins that each agent was able to earn on each game. Wins were classed as "Rational" if they managed to avoid falling for the deceptive qualities of the game in question. The agents that perform well in this experiment were not those that typically perform well in the official GVGA competition.

deterministic or not and applies a different algorithm depending on this process [67] [68]. If the game is deterministic, then a Breadth First Search is used to analyze the game state, otherwise a MCTS algorithm is used. With deceptive games, *YoloBot* finds itself performing well, but is now down to 6th position, compared to its performance in the competition where it often places 1st. Similarly, other previously high performing agents such as *Adrienctx*, which finished 2nd in the 2016 competition, find themselves performing lower than anticipated, finishing in 12th position. Of particular note is the performance of *Return42* and *IceLab* which finished in 9th and 10th places respectively in the 2016 competition and now finds themselves in 2nd and 1st places with deceptive games.

### 3.12.1 Deceptive Games with Machine Learning Agents

The first experiments that were conducted using the deceptive games were using only planning agents, and as such a further exploration of the effect on learning agents was necessary. In the paper titled *Superstition in the Network: Deep Reinforcement Learning Plays Deceptive Games* further experiments are performed, and makes use of the learning track of the GVGAI competition to test the Asynchronous Actor Critic (A2C) algorithm. The paper was accepted for publication at the Artificial Intelligence and Interactive Digital Entertainment (AIIDE) conference in 2019. A copy of this paper can be found in appendix C.

The main question of this additional set of experiments was to determine whether learning agents were deceived in similar ways to the planning agents. A2C was selected as the algorithm to experiment with as it is a widely used learning algorithm that has been shown to be capable of learning to play some GVGAI games [90] [42]. As GVGAI games are similar to arcade games, training is done on the pixel visual input from the games. Due to the porting of the GVGAI learning track into the OpenAI Gym, the A2C implementation used is the same as the OpenAI baseline implementation [91]. The neural network architecture that was used is the same as developed by Mnih et al [92]. The hyperparameters that were used are as follows: step size of 5, no frame skipping, constant learning rate of 0.007, the optimiser was RMS, and 12 workers were used.

The experiments were designed to focus on different aspects of deception, and so four of the deceptive games were used for these experiments. *Decepticoins*, *Invest*, *Flower* and *WaferThinMints*. This work explores the concepts of deception further and identifies further deceptive classifications. These games each represent a different deceptive quality: Lack of Hierarchical Understanding, Subverted Generalisation, Delayed Gratification, and Delayed Reward.

#### 3.12.1.1 Lack of Hierarchical Understanding

Lack of Hierarchical Understanding is represented by *Decepticoins*. *Decepticoins* can be seen as presenting a binary choice to the agent, to either go down one path or the



### Chapter 3. Deceptive Games

other. From that point of view, solving the task is as simple case of evaluating the expected reward of each path and selecting the path that provides the highest reward. The issue is that the levels of the game have a much larger search space than the overall goal would suggest. It is not a single action that decides which path the agent travels down, but multiple successive ones. Humans are usually able to perceive the overall direction of their actions, and are able to reason that each individual action is not important unless it serves the greater goal. An AI is forced to evaluate each action with equal importance, which combined with the large search space, makes it a challenging problem to overcome. The results of this experiment can be seen in figure 3.12.

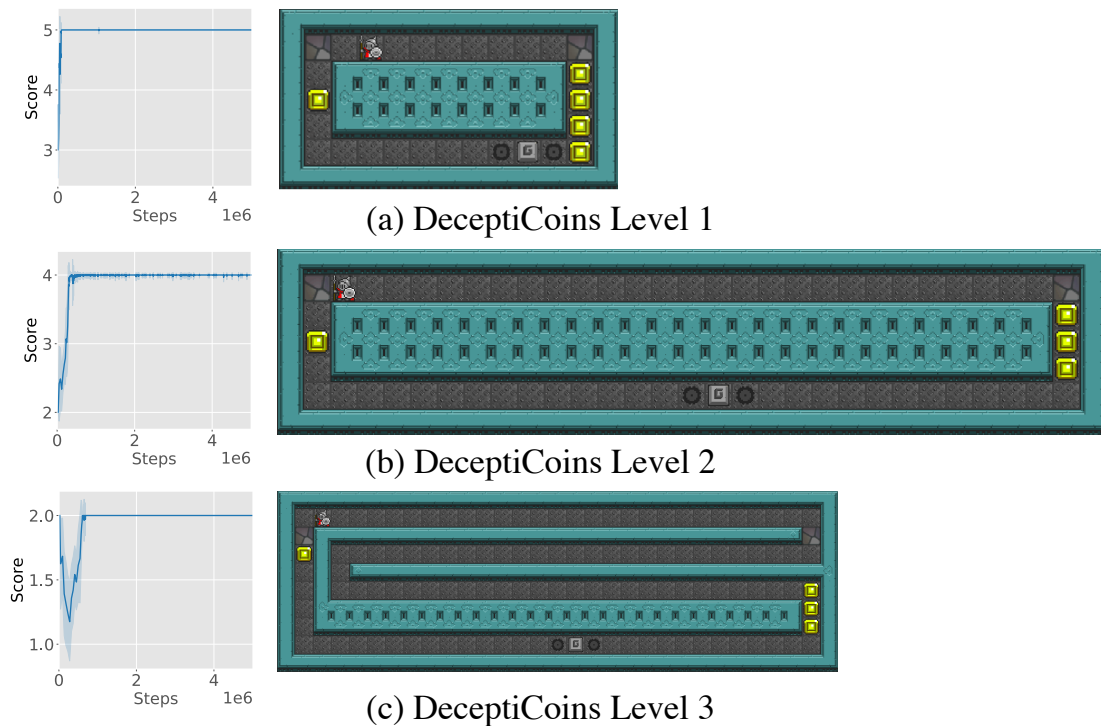


Figure 3.12: DeceptiCoins Levels - Examples of Lack of Hierarchical Understanding. While each level is different, a similar pattern in learning is observed across all of the three levels. Notably, the third level shows a poor performance as the agent does not find the optimal path.

The first thing of note with these results is that the agent is able to learn how to play each level, with the exception of level 3 which converges on the easy path scoring

just 2 points. Another note is that going from level 1 to 2, the time it takes for the agent to learn the optimal path is massively increased, which is due to the increased size of the search space. The A2C agent does seem to fall for this deception, even though it is able to solve smaller scale versions of it.

### 3.12.1.2 Subverted Generalisation

Subverted Generalisation is represented by *WaferThinMints*, which was designed specifically to deceive agents that generalise. The planning agents typically did not have a lot of trouble with this particular deception, because they would be able to perceive that a loss state was about to be reached and avoid it. As discussed previously in section 3.10, collecting mints is beneficial to the agent, but only the first 9 times. The 10th collected mint will kill the agent, causing them to lose the game. The 9 successful collections reinforce that collecting mints is a good behaviour, and so agents without a forward model may be tricked into collecting too many mints. Without the forward model, it is necessary for an agent to generalise from their past experiences, and act on that information. It should be noted that generalisation is a good skill for AI and humans to have, and the point of this deception is not to discourage generalisation, but to become aware of the potential downfalls. The results of this experiment can be seen in figure 3.13.

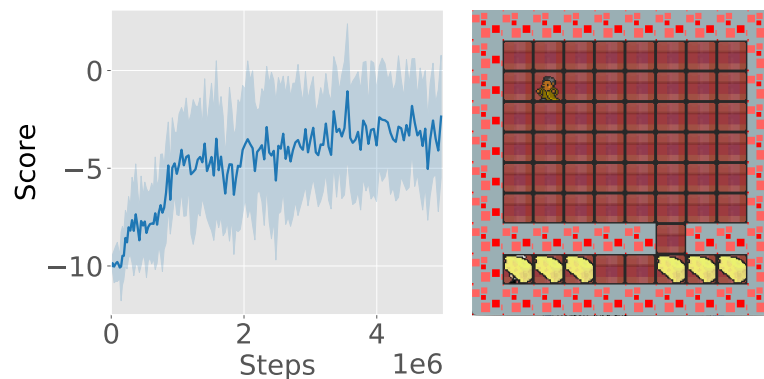


Figure 3.13: WaferThinMints level 1 - Example of Subverted Generalisation. The agent does not manage to learn a good solution to this particular game, likely because of the randomness of where win states are found.

Looking at the erratic nature of the performance it is clear that the agent did not have enough time to converge completely. This is largely due to the noisy environment, with the waiter potentially introducing bad luck on the agent with random mint drops. This noise is necessary though, in order to properly represent the deception, as simply using a level with fixed mint positions would likely lead to the A2C agent learning a pattern through the level. The agent will initially run to the room with the mints and collect as many as possible, as these are a guaranteed source of reward. It will then try to avoid the remaining mints at all costs, and usually only gets forced out of the bottom room by the waiter. Similarly, the agent does not seem to understand that collecting the mints earlier means that it is increasing its own risk. The optimal strategy would be to collect the mints closer to the end of the game, to shorten the window of vulnerability.

### 3.12.1.3 Delayed Gratification

Delayed Gratification is represented by *Flower*, and has its basis in a famous psychology experiment from 1972 by Mischel et al. [93]. The experiment involved leaving 4 year old children in a room with a marshmallow, and instructing the children not to eat the marshmallow while the experimenter is away from the room. The children are informed that if they are able to resist the temptation of eating the first marshmallow until the experimenter returns, they will receive a second marshmallow. Some of the children find this quite challenging, and so does the A2C agent. The rules for *Flower* are described in detail in section 3.7. The results of this experiment can be seen in figure 3.14.

Initially, the agent has difficulty collecting the flowers but eventually learns how to perform this behaviour. The agent understands that the flowers provide a reward quite quickly, but does not learn that collecting them faster has an effect on the overall reward. Allowing the flowers to grow fully would be the optimal strategy, as collecting them earlier nets a lower reward. As the agent improves at collecting flowers, it is able to collect them faster and so earns less reward. This is interesting as it actually causes the performance of the agent to get worse the longer is trained.

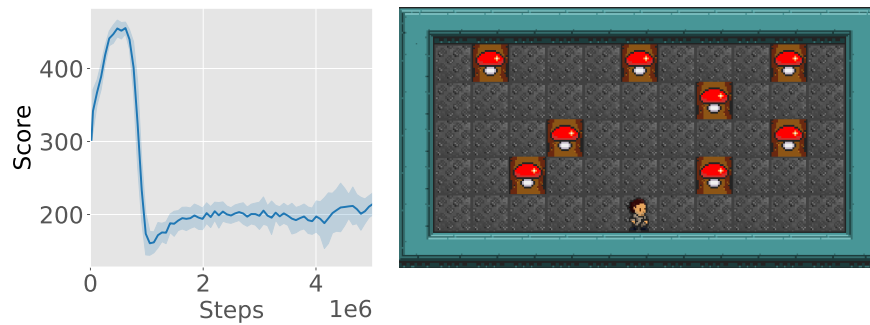


Figure 3.14: Flower level 1 - Example of Delayed Gratification. While the agent initially performs well, its performance starts to drop as it gains proficiency at finding the path to the targets before they have matured.

### 3.12.1.4 Delayed Reward

Delayed Reward is represented by *Invest*. The challenge posed by this game can be simplified down into how the agent can correctly link which action led to reward [57]. This type of deception essentially distances the reward from the action over time. By delaying the receipt of reward for some time after the action was performed, an agent is not able to learn which action triggered the reward. *Invest* is described in detail in section 3.8. The results of this experiment can be seen in figure 3.15.

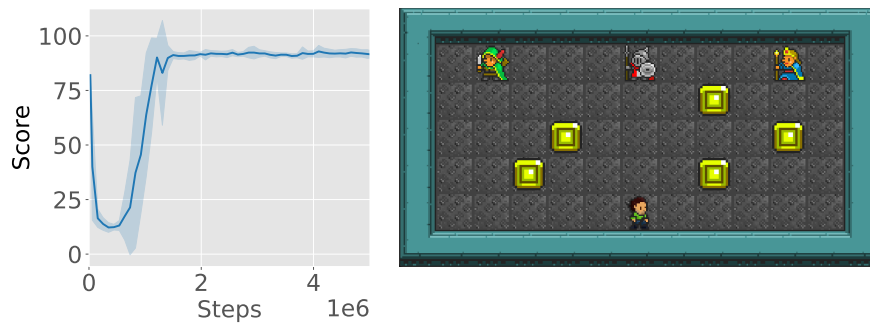


Figure 3.15: Invest level 1 - Example of Delayed Reward. The agent initially has a hard time understanding how to advance in this game but eventually develops a behaviour that leads to consistent rewards, though it is not the optimal strategy.

An interesting side effect of this deception is that a number of "superstitious" behaviours were noted in the A2C agent. The agent would invest with one of the bankers, and then move immediately to a corner of the level. It is suspected that

the reason for this is that this is where the agent happened to have been positioned randomly at the point they received the reward for interacting with the banker, and so has associated that position with the reward rather than the action itself. The training graph shows an interesting trend, where the agent begins initially very well, as it invests with all bankers, but slowly starts to associate the negative reward with the bankers. It eventually converges on investing with a single banker, and then moving back to the spot in which it received the delayed reward, while waiting for the banker to reappear.

While the concepts of delayed reward and delayed gratification are similar, the effect that they have on the performance of an agent appears to be quite different. In the experiments, delayed reward initially sees a drop in the performance of the agent before improving as can be seen in figure 3.15, whereas delayed gratification shows a spike in performance initially which then drops to a level far below the starting point of performance as in figure 3.14. This appears to lead to behavioural differences in how the agents approach the problems, though further experimentation is necessary to confirm if this is always the case.

One of the big questions that this work raised was: does the GVGAI competition adequately test for general problem solving ability? If an agent is able to win the competition, does that mean they can competently play any game? Earlier work done by Perez et al. takes a look at how robust GVGAI agents are to malicious changes, such as a forward model that doesn't accurately model the world, or a game that randomly takes a different action than the one the agent selected [66]. This previous work particularly inspired the notion of deceptive games. What can be seen from the results is the notion that the winners of the GVGAI competition are decided as much by the games chosen for evaluation, as well as by the capabilities of the agents themselves. Seeking an answer to this question leads to the next research question: is there a method for selecting games that reduces the overall bias towards any particular type of algorithm?

### **3.13 Chapter Summary**

This chapter outlines the concept of Deceptive Games and experiments that were done to develop appropriate experiments in this area. These types of games are an important hurdle for game playing AI agents to overcome and the reasons for this are also laid out.

One of the big questions that was raised throughout the work on this thesis starts to come to light when looking into Deceptive Games. That is, what problems does an AI agent need to solve in order to be considered a competent general video game player?

# Chapter 4

## Game Selection

This chapter looks at the reasoning behind selecting the set of games that were chosen for the experiments. First, looking at the Information Gain analysis of the GVGP game set that was performed, and providing a description of the games selected with this method. Secondly, to examine other games that were used to augment the original selection, and then introducing each of these games in greater detail.

### 4.1 Information Gain

This section looks at the set of games that were selected to test GVGP agents, and explains the methodology used for their selection. First, the information gain project is described, which sought to find the most interesting games from the entire GVGAI set. Second, the usage of the algorithm and a description of how it works is provided. Finally, the results of running the continuous information gain algorithm on the GVGAI are discussed.

#### 4.1.1 The Information Gain Project

A natural progression from the work done looking into deceptive games was to analyse further the set of games that make up the GVGAI library. A novel approach, based on information gain theory, for selecting an interesting set of problems from a large set was introduced, and used for the selection of some of the games for this work [24]. The

concept of *Information* refers to the Shannon Information Theory [94]. The information is representative of the confidence that the algorithm has that agent  $a$  is playing the game  $g$ , given the performance of agent  $a$  on game  $g$ . This information can then be used to provide a benchmark of games which collectively provide the most information on the agents playing. There is a wide variety of games available in the library, but do they represent a broad range of problems? How many of the games are similar to each other? Is there a particular type of game that is over or under-represented in the GVGAI set?

To look at these questions a project was setup in collaboration with Professor Julian Togelius and the Game Innovation Lab, Dr Christoph Salge from the University of Hertfordshire and Dr Matthew Stephenson from Maastricht University. The goal was to find out which games in the GVGAI set were the most interesting. In this case, interesting is defined as games which split the agents. Games that are too easy or too hard for agents to solve do not provide much insight into the current state of the GVGAI field.

As part of this team the thesis author assisted with exploring ideas for finding and defining the most interesting games in a set. Then, along with Dr Matthew Stephenson, implemented an algorithm designed by Dr Christoph Salge that could go through the entire GVGAI set and find those interesting games. The results of this work were then used by the thesis author for their own work and used as a core component of the experimental suite in later chapters.

This project has resulted in one paper which is available on Arxiv under the title *A Continuous Information Gain measure to find the most discriminatory problems for AI benchmarking* [24]. This paper can be found in appendix B.

### 4.1.2 Description

Each year, ten games are used to evaluate the performance of the submitted agents to the GVGAI competition. As shown in the deceptive games chapter, this game selection plays a significant role in deciding the winner of the competition, and its choice should keep in mind the larger goal of the GVGAI competition: to develop general video



game playing agents. Ideally, the ten chosen games should represent a diverse set of challenges, and not focus too much on any particular one, introducing unwanted bias into the competition. In other words, no two games selected to evaluate the agents should pose the same problem. As an example, the decathlon is designed in such a way that an athlete has to be competent at all of the events in order to do well. If the decathlon featured both a 100m and 200m sprint, then it may unfairly bias towards runners. For this reason, it would be beneficial to be able to detect which types of bias may exist in the set of all games, and select a subset of those games which are diverse and do not overly advantage or disadvantage any agent.

With this in mind, which games within the GVGAI are the most interesting? Games that are too easy, or too hard, do not provide much information about the general capabilities of an agent as all of the agents will either succeed or fail. The most interesting games are likely to be those that split the performance of agents down the middle, with good agents performing well and bad agents performing poorly. These discriminatory games are those that the continuous information gain work tries to identify. Ideally, the types of discrimination should differ, such that no single type of agent is biased against or towards. Similar work has been done previously which looked at which agents were best suited to which games in the GVGAI [95].

Tests were conducted by the thesis author and collaborators using the entire set of GVGAI games, including the deceptive games, with a selection of about half of the agents submitted to the competition, which were chosen based on their past performance in the GVGAI competitions over the years as well as for algorithm uniqueness. Essentially, if an agent had performed well in past competitions, and its algorithm was not already represented in the currently selected algorithms, it was included. The data provides insight into the types of agents that could solve those games and a glimpse into what problems are represented currently in the set. More detailed information on the specifics of the data gathering process can be found in section 4.1.3.

With the data gathered, the next step was to understand how best to select the games which would not bias towards any particular type of agent. To this end, an algorithm was developed which finds the game from the whole set that provides the

most information about the agents, then removes that game and recursively finds the game that then provides the most information gain and so on. This process repeats until ten games are selected, in order to match the number of games chosen each year for the GVGAI competition. An alternative method for deciding the number of games would be to stop selecting new games once the amount of information gained by including a new game is below a set value. Performance measures for the agents were done in three ways: win rate, score and both combined. Neither win rate nor score entirely explains whether or not an agent has been completely successful in winning a game, as demonstrated by the deceptive games in chapter 3.

An alternative approach to this problem was proposed by Balduzzi et al. in their 2018 paper entitled *Re-evaluating Evaluation* [96]. Balduzzi et al. propose *Nash averaging* as a way to solve the problem of introducing bias into experimental sets and allows researchers to include all possible problems in experiments as the approach adapts to both problems that are too easy, and agents that are too weak.

### 4.1.3 Data Collection

To gather the data the competition conditions were replicated as closely as possible. To this end, the agents were ran across 243 CPU cores with 2.6GHz processing power and 8 GB of memory. In total, 3,990,760 complete playthroughs were recorded. A successful playthrough was defined as the game finishing with the agent either winning or losing, and no crash having occurred. The information that were retrieved from each playthrough were: the agent that was playing, whether the agent had won, and the score that had been achieved. On average, each agent/game pair had 1,368.6 units of data generated.

### 4.1.4 How does the Information Gain algorithm work?

The purpose of the continuous information gain algorithm is to find the most discriminatory subset of games in a set of games.

The algorithm is broken up into two main components: single game information gain (SGIG) and multi game information gain (MGIG). The SGIG is able to determine

how effective a specific game is at discriminating agents. The MGIG is able to apply the SGIG across a set of games to find not only the most discriminative games, but the games that are discriminative in different dimensions.

The SGIG is used with every possible pairing of all agents, for each game. This then creates a confusion matrix which shows how often a particular game is able to correctly identify an agent playing that game, given the agent's performance. Each cell in the matrix refers to the probability that a given result is from a particular agent. This is defined as  $P(A2|A1)$  where  $P$  is the probability function which states the probability of identifying the agent  $A2$  given the performance of agent  $A1$ . The formal definition of the probability function can be found in appendix B.

$$C = \begin{pmatrix} P(A1|A1) & P(A2|A1) \\ P(A1|A2) & P(A2|A2) \end{pmatrix} \quad (4.1)$$

Once a game is selected the process repeats itself  $n$  times, where  $n$  is the desired number of games, but the chosen game is removed from the set. Rather than simply selecting the the top  $n$  information games, this recursive process reduces the chance that two games will be selected which discriminate in a similar manner. The intuition is that games selected via this process should provide information about different types of agents, so that no subset of the games are alike in terms of agent performance.

As an example, there are two agents,  $A1$  and  $A2$ , which each play the games  $G1$  and  $G2$ . Both agents play both of the games and their average scores for each game is shown in table 4.1. From these results, it seems that  $G1$  is the most discriminatory game, so this would be the desired output of selecting a single game from this set.

Table 4.1: Example results for the Information Gain process using score and (Std Dev)

Agent	G1 Average Score	G2 Average Score
$A1$	60 (10)	30 (5)
$A2$	15 (10)	15 (5)

The results from table 4.1 are input into the SGIG algorithm to create the confusion matrix shown in 4.2. Another matrix is similarly generated for  $G2$ . The matrices can

then be used to get a value of how much information about the agents can be gathered with that particular game.

$$G1 = \begin{pmatrix} 0.926 & 0.074 \\ 0.074 & 0.926 \end{pmatrix} \quad (4.2)$$

With the matrix generated the amount of information is calculated, which in this example for  $G1$  is 0.620. This number is an indication of how good that particular game is at discriminating agents.  $G2$  generates a lower amount of information, 0.197. As  $G1$  provides more information about the agents it would be the game selected by the algorithm. If a third game were to be introduced where the agents perform identically, the amount of information generated for that game is zero.

The process can be run on a variety of metrics and all that is required is the metric itself and the standard deviation of that metric. In the case of the GVGAI set the metrics used were average wins, average score and both wins and score combined. In the case of the combined wins and score, both cases are treated as a separate game. Once the results for each are calculated, they are brought together into a single result and the information is calculated from their combined value.

The full details of the mathematical underpinning of the continuous information gain method can be found in appendix B. The algorithm can also be found on the thesis author’s github [97].

## 4.2 Information Gain Experiments and Results

Once the continuous information gain method was developed, it was time to find the most discriminatory games. Experiments were conducted using the entire library of GVGAI games, including the deceptive games, using 27 agents that had been submitted to the GVGAI competition. The intuition behind this experiment is that if an agent is good at a particular game, then it may also be good at other games of the same type, and ideally a selection of games can be chosen which do not favour any one particular type of game too heavily. The reasons for not using all of the agents submitted is that many of the agents would not complete full runs for all of the games, due to compilation

## Chapter 4. Game Selection

errors, and some agents are slightly modified versions of others and would not present new information. When selecting between variations of the same agents, the more up to date version was selected.

The output of the information gain process can be seen in tables 4.2, 4.3, 4.4 and 4.5. The results show the analysis performed with agent win rate, score and both combined as well as the cumulative results after iterating.

Table 4.2: The games with the highest win rate information gain in descending order.

<b>Game Name (Win-rate)</b>	<b>Information Gain</b>
freeway	1.17484168
labyrinth	1.10088062
tercio	1.10018133
labyrinthdual	1.08531707
iceandfire	1.07275305
chopper	1.06542656
doorkoban	0.98911214
hungrybirds	0.91886839
watergame	0.89206793
escape	0.87721725

Table 4.3: The games with the highest score information gain.

<b>Game Name (Score)</b>	<b>Information Gain</b>
invest	1.62405816
intersection	1.13955416
freeway	1.13619392
tercio	1.10018133
watergame	0.89206793
cops	0.88658183
flower	0.86746818
waitforbreakfast	0.80128373
labyrinth	0.78021437
realportals	0.73246317

The output from this process is used to select a group of 10 games which represent a diverse group of different problems based on agent performance. The games that were

Table 4.4: The games with the highest combined information gain in descending order.

<b>Game Name (Combined)</b>	<b>Information Gain</b>
freeway	1.89430152
invest	1.62405816
intersection	1.59362941
chopper	1.48524965
tercio	1.44693431
labyrinthdual	1.42090667
iceandfire	1.32455879
hungrybirds	1.32100004
waitforbreakfast	1.28983481
doorkoban	1.28593860

Table 4.5: The games with the highest cumulative information gain after each recursive step.

<b>Game Name (Cumulative)</b>	<b>Information Gain</b>
freeway	1.89430152
invest	3.08236771
labyrinthdual	3.81992620
tercio	4.22563462
sistersaviour	4.40856274
avoidgeorge	4.54036694
escape	4.60252506
whackamole	4.64444512
chopper	4.67138328
watergame	4.68457480

selected are those from table 4.5.

Interestingly, the maximum amount of information that a set of games can provide with this technique is  $\log_2(n)$  where  $n$  is the number of agents. In this case, the maximum amount of information is 4.75 or  $\log_2(27)$ . Table 4.5 shows that by the fourth game the majority of information about the set of games has been found. This may indicate that there are only a few different types of games within the GVGAI.

The remainder of this section is used to describe each of the games that were selected via the information gain process.




### 4.2.1 Avoidgeorge




Figure 4.1: The first level of Avoidgeorge

The objective of *Avoidgeorge* is to avoid the hostile NPC, George, and to prevent him from touching all of the other NPCs on the map. When George interacts with an NPC they become annoyed, and can only be calmed down by the avatar interacting with them. The game is won by preventing at least one of the NPCs from being annoyed for the duration of the game. The game is lost either when George interacts with the avatar, or there are no calm NPCs remaining.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  George: The hostile NPC trying to annoy all other characters.
-  Citizen: If interacted with by George, becomes annoyed and awards the player  $-1$  score.

-  Annoyed Citizen: If interacted with by the Avatar, becomes relaxed and awards the player 1 score.

### 4.2.2 Chopper

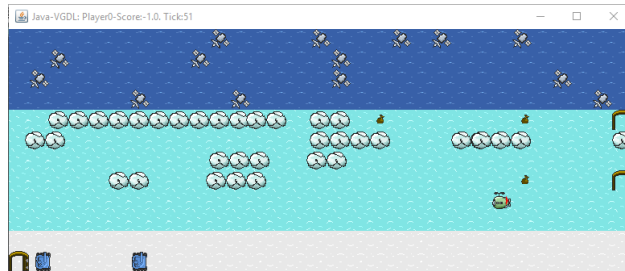






Figure 4.2: The first level of Chopper

The objective of *Chopper* is to prevent the tanks from destroying the satellites in the sky. The tanks do this by firing missiles into the sky, and the avatar must prevent this by shooting the tanks first. There are some clouds in the path which offers a shield to the satellites. If the agent destroys a tank, then they are awarded 1 point, likewise if a satellite is destroyed then the agent is penalized  $-1$  points. The game is lost if either the avatar is destroyed, or all satellites are destroyed. The agent wins if they are able to destroy all of the tanks.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Satellite: Removes 1 point if destroyed. If all are destroyed then the agent loses.
-  Cloud: Blocks missiles from reaching the satellites. Destroys both the missile and the cloud in that case.
-  Tank: Fires missiles upwards to destroy either the avatar or the satellites. Awards 1 point if destroyed. If all are destroyed then the agent wins.



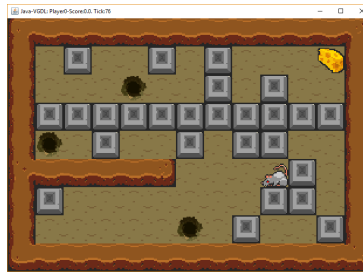






Figure 4.3: The first level of Escape

### 4.2.3 Escape

The objective of *Escape* is to reach the cheese by navigating through the level. Paths can be opened up by moving boxes around, and can be destroyed by moving them into a hole. If the avatar moves into a hole however, they are killed and the game is lost. The agent wins if the avatar manages to reach the cheese.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Box: Moves one space when the avatar interacts with them. Can be destroyed by being moved into a hole.
-  Hole: Destroys boxes if they are pushed into the hole. If the avatar moves into a hole they are destroyed, and the agent scores -1 points.
-  Cheese: The objective of the game. If the avatar reaches the cheese then the agent wins.

### 4.2.4 Freeway

The objective of *Freeway* is for the avatar to cross the busy freeway and reach the flag. In order to do this, the avatar must avoid being hit by traffic that move along the roads. If the agent reaches the flag, they are awarded 10 points, the avatar is placed back at the start of the map and a new flag is randomly placed at the end of the map.





If the agent is hit by a vehicle, they lose 5 points and are transported back at the start of the map. The agent has five lives available to them and will only lose the game



Figure 4.4: The first level of Freeway

if the agent loses all lives. The game is won if the agent has lives remaining when the game timeout is reached.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Flag: Awards 10 if collected. A new flag is generated after collection.
-  Fast Car: Moves at a fast speed either from left to right or right to left along the roads. Removes 5 points and a life from the agent if hit.
-  Slow Car: Similar to Fast Car expect moves at a slower speed.

#### 4.2.5 Labyrinthdual









Figure 4.5: The first level of Labyrinthdual

The objective of *Labyrinthdual* is to reach the exit of the labyrinth. In order to do this, the agent must avoid traps and collect coats which allows them to unlock various barriers on the map. If the agent has not reached the exit by the level timeout, they lose.

## Chapter 4. Game Selection

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Exit Flag: Awards 1 point if collected. Agent wins the game if collected.
-  Red Coat: Awards 3 points if collected. Allows the avatar to traverse through red blocks.
-  Blue Coat: Awards 3 points if collected. Allows the avatar to traverse through blue blocks.
-  Blue Block: Blocks the avatar from progressing unless they have collected the blue coat.
-  Red Block: Blocks the avatar from progressing unless they have collected the red coat.

### 4.2.6 Tercio










Figure 4.6: The first level of Tercio

The objective of *Tercio* is to move the tree to the hole. The avatar can move the tree by pushing it, but the avatar cannot freely move onto all squares. There are black, white, blue and brown squares. If the avatar is currently on a blue square, they can only move onto other blue squares. This is the same for all other colours except brown, which allows the avatar to transition to a new colour of square. The agent loses if the time runs out.

The sprites used in this game are as follows:

## Chapter 4. Game Selection

-  Avatar: Represents the player/agent in the game.
-  Hole: Awards 1 point if the tree is moved onto it. Agent wins the game if that is the case.
-  Tree: Can be moved around by the avatar if they push it.
-  Black Square: Allows avatar to move on black and brown squares only.
-  Blue Square: Allows avatar to move on blue and brown squares only.
-  White Square: Allows avatar to move on white and brown squares only.
-  Brown Square: Allows the avatar to move from one colour of square to another.


### 4.2.7 Watergame






Figure 4.7: The first level of Watergame

The objective of *Watergame* is to reach the exit of the map. There are pools of water which block access to the exit which can only be cleared by pushing a jar onto them. As the jars can only be pushed, if they are moved incorrectly this can often lead to an unwinnable game state, and so a precise pattern should be followed in order to win. If the level timeout is reached then the agent loses the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.

-  Exit: Agent wins the game if the avatar reaches the exit.
-  Water: Cannot be crossed by the avatar. Disappears if a jar is pushed into it.
-  Jar: Can be pushed by the avatar. Absorbs water if pushed into it.





#### 4.2.8 Whackamole



Figure 4.8: The first level of Whackamole

The objective of *Whackamole* is to hit all of the moles in the level. The moles appear randomly from holes in the ground and award 1 point when hit. There is also a hostile cat which is also trying to collect moles, and if the cat hits the avatar the agent loses the game and receives a  $-5$  points penalty. There are two types of moles, fast and slow, which remain for different amounts of time. The game is lost if the cat catches the avatar, and the game is won if the avatar survives until the level timeout.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Cat: Chases moles and the avatar. Agent loses the game if the cat catches the avatar and receives  $-5$  points.
-  Mole: Awards 1 point if hit.
-  Hole: A location where a mole may spawn.

The final list of games selected from the information gain technique are shown in alphabetical order in table 4.6.

Table 4.6: The games selected, in alphabetical order, via the Information Gain process using the combined metric (games in bold are deceptive)

<i>Avoidgeorge</i>	<i>Chopper</i>
<i>Escape</i>	<i>Freeway</i>
<b><i>Invest</i></b>	<i>Labyrinthdual</i>
<b><i>SisterSaviour</i></b>	<i>Tercio</i>
<i>Watergame</i>	<i>Whackamole</i>

### 4.3 Further Games

The games selected through the Information Gain technique in section 4.1 provide a good start point for initial experiments. In order to widen the range of games experimented with, the games that were chosen in other work for experimentation were also included [98]. This may reintroduce some bias into the results as the games being selected have not been selected through the information gain process. In order to mitigate this risk, the results in the final experiments are separated into the information gain set, and the full set including the additional games.

The reason for including additional games beyond the Information Gain technique was to increase confidence in the results, as well as to add in more diversity to the range of games. This section describes those games in detail. Any deceptive games that were already described in section 3 are not included in this section again.

#### 4.3.1 Aliens

*Aliens* is one of the simpler games in the GVGAI and is typically used as an example of how the VGDL functions. The objective of *Aliens* is to destroy all of the invading alien ships before they destroy the avatar, and resembles the classic arcade game Space Invaders.

The avatar can move either left or right, and may also fire projectiles which travel

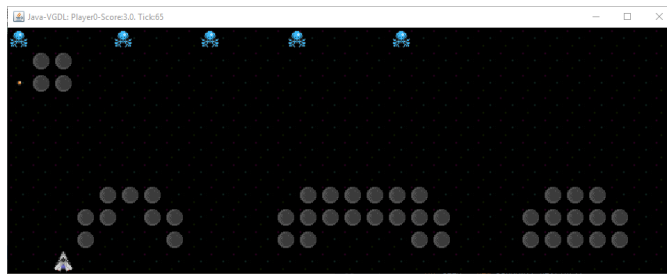





Figure 4.9: The first level of Aliens

upwards towards the top of the screen, and if the projectile collides with an alien ship, then it is destroyed and rewards the player with 2 points. The aliens will return fire with their missiles which will either kill the avatar or destroy a defensive base that the avatar can use to hide.

*Aliens* is a game where agents will typically win this game nearly 100% of the time [95]. An interesting component of this game though is that there is a delay between the fire action to launch a projectile, and the reward generated for that action. As a result of this, some unusual behaviours emerge as agents do not correctly associate firing as the action that generates the reward.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Aliens: Fire projectiles down towards the bases and avatar. Awards 2 points if killed.
-  Base: Awards 1 point if hit. Blocks missiles from either direction, and are destroyed on impact.

### 4.3.2 Bait

The objective of *Bait* is to collect a key and use it to reach the locked door. There are holes on the map which will kill the avatar if they step on them, causing the agent to lose the game. The avatar can move boxes around the map and can push them into the holes in order to cross the holes safely, which also awards 1 point. On some levels









Figure 4.10: The first level of Bait

mushrooms are available, and they award the agent 1 point if collected. The agent loses if the avatar steps into a hole, and the game is won if the avatar collects the key and brings it to the exit.

What makes this game interesting is that the levels do not have any direct paths to success: they are all blocked by holes. The agent may have to efficiently use the boxes available to create a path for themselves that allows them to accomplish the objective.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Locked Door: The goal of the game. The avatar must bring the key to this door in order to win. Awards 5 points if this is the case.
-  Key: Allows the avatar to use the locked door, if collected.
-  Hole: Kills the avatar if they step into the hole.
-  Box: Can be pushed by the avatar. Awards 1 point if pushed into a hole.
-  Mushroom: Awards 1 point if collected.

### 4.3.3 Camelrace

*Camelrace* may be one of the simplest games in the GVGAI, but proves a challenge for many agents. The objective of *Camelrace* is to win the race by navigating from the left side of the level to the finish line on the right. Several other camels are competing against the agent, and the first one to reach the finish line is the winner.








Figure 4.11: The first level of Camelrace

This game is an excellent example of a problem that many agents run into, which is that if a goal is too far away in the future, then the agent will not be able to see it. There are no rewards earned by the agent until the completion of the game, so a search based solution must generate a correct path many time-steps into the future to be able to win. If the agent can pass the finish line first, they earn 1 point; otherwise, they earn  $-1$  and lose. An interesting observation of performance on this game is that the more sophisticated agents tend to perform worse than their simpler counterparts.

*Camelrace* was included not only because it has been used in other experiments but is also challenging for greedy algorithms to solve. This game was originally designed as an example of a problem that many agents may find challenging to solve.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Camels: The other camels in the game. If they reach the exit before the avatar, the agent loses the game.
-  Finish Line: The first camel to reach this wins the game. If it is the avatar, then the agent wins the game.

#### 4.3.4 Chase

The objective of *Chase* is for the avatar to find and kill all of the birds on the level. If a bird comes across the body of another bird, then it will become angry and will hunt down the avatar and kill them. Each bird killed awards the agent 1 point, and if an angry bird catches the avatar then they are penalized  $-1$  points and lose the game. If

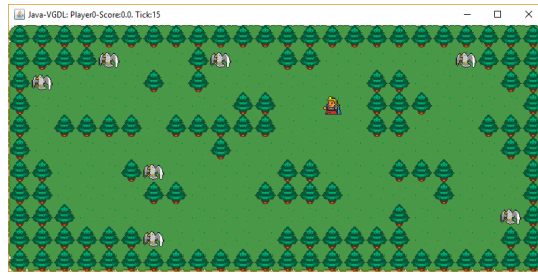






Figure 4.12: The first level of Chase

the agent is able to kill all of the birds then they win the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Bird: Awards 1 point if killed. Becomes a worm when killed.
-  Angry Bird: A bird becomes angry if it collects a worm. Hunts and kills the avatar causing the agent to lose the game.
-  Worm: Transforms a bird into an angry bird if collected.

#### 4.3.5 Crossfire

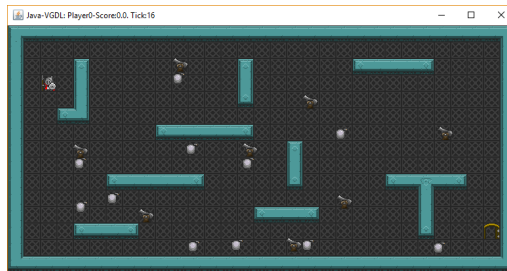






Figure 4.13: The first level of Crossfire

The objective of *Crossfire* is to reach the exit of the level while avoiding the obstacles and their projectiles. The turrets on the level fire projectiles in a random direction, creating a challenging pattern of movement which the avatar must traverse in order to reach the exit. If the avatar is able to reach the exit, the agent earns 5 points and wins

## Chapter 4. Game Selection

the game. Should the avatar be hit by a projectile, the agent will lose 1 point and lose the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Turret: Fires projectiles in a random direction.
-  Projectile: Fire by a turret in a random direction. If a projectile hits the avatar, the agent loses 1 point, and loses the game.
-  Exit: Awards 5 points and the agent wins the game if the avatar reaches it.

### 4.3.6 Digdug



Figure 4.14: The first level of Digdug





The objective of *DigDug* is for the avatar to collect all of the gems and gold coins on the level. There are also a number of hostile NPCs on the level that will try to kill the avatar, and causing the agent to lose the game. The avatar is able to dig through the level, and fire boulders at the enemies to kill them and protect itself.

The agent loses if an enemy is able to kill the avatar and suffers a  $-1$  penalty, but receives 2 points if they are able to kill a monster. If all monsters are killed, or all gems and coins are collected then the agent wins the game.

*DigDug* was included as it is a particularly challenging game which very few agents have been able to solve [95].

The sprites used in this game are as follows:

## Chapter 4. Game Selection

-  Avatar: Represents the player/agent in the game.
-  Coin: Collected by the Avatar. Agent wins the game if the Avatar collects all of the coins and gems.
-  Gem: Collected by the Avatar. Agents receives 1 points if collected, and wins the game if all coins and gems are collected.
-  Enemy: Awards 2 points if killed. Agent wins the game if all are killed, otherwise Agent loses the game if the Avatar is killed by the enemy.

### 4.3.7 Hungrybirds

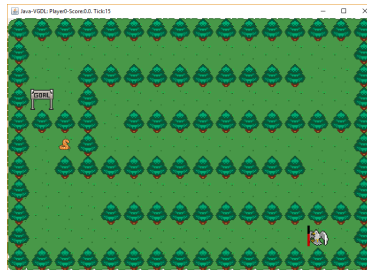





Figure 4.15: The first level of Hungrybirds

The objective of *Hungrybirds* is for the avatar to reach the exit of the maze. Each time-step, the avatar gets hungry and loses health and must eat food in order to maintain their health level. When the avatar finds a food source, they regain health. If the avatar's health reaches 0 then the agent loses the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game. Loses 15 health every time-step. The agent loses the game if the Avatar's health reaches 0.
-  Food: Collected by the Avatar. The Avatar regains 15 health and the agent gains 40 points.
-  Exit: The goal of the maze. If the Avatar reaches this then the agent gains 100 points.

### 4.3.8 Infection







Figure 4.16: The first level of Infection

The objective of *Infection* is for the avatar to infect all of the healthy people in the level. Each map features medics which are able to cure any infected people that they encounter, and are able to cure the avatar, preventing them from infecting other people until it is infected again. The avatar can kill medics by attacking them, awarding 2 points. The game is won if all healthy NPCs are infected, and the game is lost if the timeout is reached.

This is a more complicated game, as there are numerous interactions between the NPCs and the avatar. The goal of the agent is simple, but challenging to execute due to the medics.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game. The first sprite is the normal version of the avatar, which cannot infect other NPCs. The latter is the carrier version, which is able to infect others.
-  Virus: Transforms the Avatar from normal to carrier. Can also infect NPCs, except the medics.
-  People: Can be infected by the Avatar. The first sprite is an uninfected person, and awards 2 points if infected by the avatar. The latter sprite is already infected, and awards  $-1$  points to the agent if they are cured by a medic. If an infected person is cured, they revert back to an uninfected person.

-  Medic: Awards 2 points if killed. Can transform infected persons, or the avatar, to uninfected. Can kill any Virus if interacted with.

### 4.3.9 Intersection



Figure 4.17: The first level of Intersection

The objective of *Intersection* is for the avatar to navigate through the level to collect items that spawn randomly at various locations, which are separated by a busy highway. Each time a vehicle collides with the avatar then the avatar loses a life and loses 5 points. If the avatar is able to collect an item then they are awarded 10 points. The avatar begins the game with 5 lives.

Should the avatar lose all of their lives then the game is lost. If the avatar is able to survive until the timeout of 1000 timesteps then the agent wins the game.

The sprites used in this game are as follows:




-  Avatar: Represents the player/agent in the game. Begins the game with 5 lives. If the avatar loses all of their lives, then the agent loses the game.
-  Vehicle: Travels at different speeds, slow or fast, across the intersection. If the avatar interacts with any Vehicle then the agent is penalized -5 points and loses a life.
-  Item: Appears randomly throughout the level. Awards the agent with 10 points if the avatar is able to collect them.






Figure 4.18: The first level of Lemmings



### 4.3.10 Lemmings

The objective of *Lemmings* is to build a path that will guide lemmings to the exit of the level. There are a number of obstacles in the way that will impede the lemmings, and potentially kill them if interacted with. The lemmings will spawn from a doorway and make their way towards exit of the level. Each lemming that makes it to the exit gives the agent 2 points. If a lemming falls into a trap then the agent loses 2 points. If the avatar falls into a trap themselves, then the agent loses 5 points and loses the game.

What makes this game interesting is that the avatar is able to dig a path through the walls for the lemmings to follow, allowing them to bypass traps and reach the exit. The agent is not able to directly control lemmings, only the avatar. Each wall that the avatar interacts with however, costs 1 point to remove. This leads to the majority of agents, even successful agents, scoring poorly on *Lemmings* and makes for a particularly challenging game overall.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Lemming: Appears from the door object and moves towards the exit. Awards the agent 2 points if it reaches the exit, or  $-2$  if it reaches a trap instead.
-  Trap: Kills the lemmings or avatar if stepped on. If the avatar steps onto a trap, then the agent loses 5 points and loses the game.

-  Entrance: The door from which lemmings spawn and begin to travel towards the exit.
-  Exit: The exit that the lemmings are trying to reach. Awards the agent with 2 points for each lemming that reaches it.






#### 4.3.11 Missilecommand



Figure 4.19: The first level of Missilecommand

The objective of *Missilecommand* is to defend several cities from missiles that have been launched at them. The avatar is able to fire at incoming missiles, and wins if they are able to destroy all of the missiles. For each missile that the avatar destroys, the agent is awarded 2 points. If a missile reaches a city, then the agent loses 1 point. If all of the cities are destroyed, then the agent loses the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game. Is able to fire bullets at the missiles to destroy them.
-   Missile: A set number of these appear in each level and will fall at different speeds, towards the cities at the bottom of the level.
-  City: The cities that the avatar must defend. If all of these are destroyed, then the agent loses the game. Each city lost loses the agent 1 point.
-  Bullet: Fired by the avatar upwards to destroy missiles. If a bullet hits a missile, then the missile is destroyed and the agent is awarded 2 points.



### 4.3.12 Modality









Figure 4.20: The first level of Modality

The objective of *Modality* is to push a crate into a hole, which is similar to another game known as Sokoban. Sokoban is a puzzle video game that presents the player with a maze, and the avatar has to push crates around the maze into their correct positions.

The avatar is only able to move on one type of ground at a time, and has to step onto a specific spot in order to change which type of ground they can move onto. The crate is able to move across any surface, though because of the limitations on the avatar it is possible to push the crate to a spot where the avatar cannot enter. The avatar wins if they are able to push the crate into the goal, and loses if the timeout is reached. The avatar is awarded 1 point when the crate reaches the hole.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-   Ground: The ground that the avatar is able to move on. If the avatar is already on one of these types of ground, it cannot freely move onto a square of the other colour, unless they go through the neutral square.
-  Neutral: Neutral ground which allows the avatar to move onto a different colour of ground.
-  Crate: Can be pushed by the avatar in order to reach the hole. Can move onto any colour of square. Awards 1 point if pushed into the hole.
-  Hole: The hole which the crate has to be pushed into.

### 4.3.13 Plaqueattack

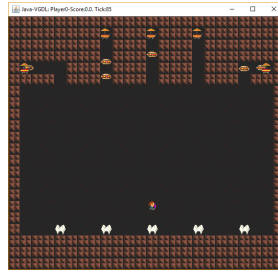







Figure 4.21: The first level of Plaqueattack

The objective of *Plaqueattack* is to defend the teeth from food, which are trying to destroy them. This is a similar game to Missile Command, except that the levels are larger and there are a larger number of projectiles to deal with. The avatar can move around the world and fire missiles at the food to destroy it, and can also repair teeth that have been damaged. Food items are spawned from several spawning grounds in each level, which disappear after spawning 5 of their related food item.

For each food item that the avatar destroys, the agent receives 2 points. If a food item is able to reach a healthy tooth, then the tooth becomes damaged and the agent loses 3 points. If there are no healthy teeth remaining on the level, then the agent loses the game. The agent is able to transform a damaged tooth back into a healthy tooth by interacting with it, and will receive 1 point for doing so. The agent wins if they are able to destroy all food items, and at least 1 healthy tooth remains.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game. Can fire projectiles.
-  Food Item: Spawned from food trolleys. Can be destroyed by avatar projectiles, awarding the agent with 2 points. Transforms healthy teeth into damaged teeth if touching them, and causes the agent to lose 3 points. Both types of food item behave in the same way.
-  Healthy Tooth: Can be transformed into a damaged tooth if a food item interacts with it. If no healthy teeth remain then the agent loses the game.

-  Damaged Tooth: Can be transformed into a healthy tooth if the avatar interacts with it.
-  Food Trolley: Spawns food items of different types.

#### 4.3.14 Roguelike



Figure 4.22: The first level of Roguelike

The objective of *Roguelike* is for the avatar to navigate through a maze and find the exit. *Roguelike* features particularly large levels and has many features in common with the rogue-like genre of games. There are two types of enemies that the avatar will face, phantoms and spiders. Spiders are faster, and present a tougher challenge to the agent. Each level has collectables that the avatar can reach, such as hearts to restore health and coins. Each level also features some keys which can unlock doors, and markets which the avatar can visit to exchange gold coins for health.











The avatar is awarded points for defeating enemies, 1 point for a phantom and 2 points for a spider. The avatar begins the game with 1 health point, which can be increased by interacting with hearts or exchanging coins for health at a market. Each heart restores 1 point of health, and each exchange also restores 1 point of health. The avatar's health cannot exceed 10 points of health. If an enemy NPC is able to interact with the avatar, then the avatar loses either 1 point of health if it was a phantom, or 2 for a spider. The avatar also receives points for collecting items in the map, namely the sword, key and coins. The sword gives the agent 2 points and allows the avatar to attack. The key and coins each give 1 point to the agent.

If the avatar is reduced to 0 health, then the avatar is killed and the agent loses

## Chapter 4. Game Selection

the game. Should the avatar reach the exit of the level then the agent is awarded 10 points and wins the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game. Can use the sword to defeat enemies, but only after collecting it from the level. The agent loses the game if the avatar is reduced to 0 points of health.
-  Sword: Can be collected by the avatar. Allows the avatar to attack enemies and awards the agent 2 points if collected.
-  Phantom: A slow enemy that tries to kill the avatar. Reduces the avatar's health by 1 if interacted with directly. Awards 1 point to the agent if the avatar hits the phantom with the sword.
-  Spider: A fast enemy that tries to kill the avatar. Reduces the avatar's health by 2 if interacted with directly. Awards 2 point to the agent if the avatar hits the phantom with the sword.
-  Coin: Can be collected by the avatar. Awards the agent 1 point if collected by the avatar.
-  Market: The avatar can exchange points at a market in order to restore health. Exchanges transform 1 coin into 1 point of health.
-  Heart: Can be collected by the avatar to restore 1 point of health.
-  Key: Can be collected by the avatar to unlock a gate. Awards the agent 1 point if collected.
-  Gate: Prevents the avatar from moving through it until it is unlocked by a key.
-  Goal: The objective of the level. Awards 10 points if the avatar reaches it, and the agent wins the game.

### 4.3.15 Seaquest







Figure 4.23: The first level of Seaquest

The objective of *Seaquest* is to rescue divers that have fallen into the sea. The avatar is a rescue submarine that is trying to reach the divers, while avoiding aggressive animals. The submarine is able to fire projectiles which kills animals, and awards 1 point to the avatar. The objective is further complicated by the submarine having a limited capacity of oxygen, requiring that they return to the surface to refill oxygen periodically. If the avatar is able to rescue all of the divers, then the agent is awarded 1000 points.

The agent wins the game if the avatar is able to survive until the timeout of the level. If an animal reaches the avatar, then the avatar is killed and the agent loses the game. Similarly, if the avatar runs out of oxygen, then the avatar is killed and the agent loses the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game. Can fire torpedoes. Has a limited supply of oxygen which is depleted while the avatar is underwater.
-  Torpedo: Fired by the avatar. If a torpedo hits an animal, then the animal is killed and the agent is awarded 1 point.
-  Animals: There are multiple types of animals which move at different speeds and behaviours. If any of the animals interact with the avatar, then the avatar is killed and the agent loses the game.

-  Diver: Divers that have become lost at sea. Can be collected by the avatar and returned to the surface. If all 4 divers are saved, then the agent is awarded 1000 points.

#### 4.3.16 Survivezombies





Figure 4.24: The first level of Survivezombies




The objective of *Survivezombies* is for the avatar to survive the waves of zombies that are hunting it. The zombies will chase down the avatar, and if they are able to catch it then the avatar will lose 1 point of health. Zombies will appear from graves that appear around the map. If the avatar interacts with a grave, then the avatar is killed and the agent loses 1 point and loses the game. The avatar is able to defend themselves by collecting honey from around the map. Honey increases the avatar's health by 1 point and also awards the agent with 1 point.

Friendly bee NPCs will move around the map randomly. If a bee interacts with a zombie, then the zombie is turned into a honey item. If the avatar is able to survive until the timeout of the level then the agent wins the game.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.
-  Zombie: Chases the avatar around the level. Reduces the health of the avatar by 1 if interacted with, as well as losing the agent 1 point. Spawned from graves.

## Chapter 4. Game Selection

-  Grave: Spawns zombies which will chase the avatar. There are two types of graves which spawn the same type of zombie, but do so at different intervals. If the avatar interacts with a grave, then the avatar is killed and the agent loses the game and 1 point.
-  Honey: Increases the health of the avatar by 1 if collected. Also increases the agent's score by 1.
-  Bee: Moves randomly around the level and turns zombies into honey if interacted with.

### 4.3.17 Waitforbreakfast

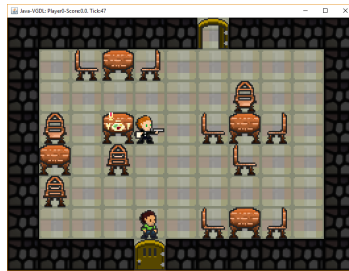







Figure 4.25: The first level of Waitforbreakfast

The objective of *Waitforbreakfast* is to successfully eat breakfast. The avatar must wait for the waiter to serve breakfast, before eating it. If the avatar moves to the table before breakfast is served, then they must wait there until breakfast arrives. If the avatar leaves early, then the agent loses the game. The agent wins only if the avatar eats breakfast after it has been served, and without leaving the table after sitting there.

*Waitforbreakfast* can also be considered a deceptive game, as it does not clearly state to agents what the objective is immediately, and tries to punish agents for behaving erratically by forcing them to lose the game if they move away from the table.

The sprites used in this game are as follows:

-  Avatar: Represents the player/agent in the game.

-  Waiter: Brings breakfast to the table for the avatar. Initially does not spawn with breakfast, and has to wait for a random amount of time before breakfast is ready to be served.
-  Chair: The location that the avatar must remain at until breakfast is served. If the avatar leaves before breakfast is served, or leaves without eating breakfast, then the agent loses the game.
-  Table without Breakfast: The table before breakfast is served.
-  Table with Breakfast: The table after breakfast is served. If the avatar eats breakfast then the agent is awarded 1 point and wins the game.

With the selection of games for experimentation finalised the next step was to develop agents that may be able to solve them, and select agents that would be good benchmarks for comparison. The goal is to have a diverse selection of agents to compare with, and to develop some new agents which are able to see improved generality across the suite of games. Ensemble Agents were determined to be an interesting avenue to explore.

### 4.4 Chapter Summary

This chapter outlines the rationale behind the particular games that were chosen as part of the experiments for the final work of this thesis. This choice is particularly important because of the likelihood of introducing bias through either over or under representing a particular type of game. Along with this a detailed description of each game has also been provided along with an explanation of the specific challenge involved when playing each game.

A novel problem selection technique, Continuous Information Gain, was also introduced as a way of solving the problem of game bias in the experimental set.



## Chapter 5

# Ensemble Decision Systems

As previously discussed in chapter 2, Ensemble Decision Systems (EDS) are a method of combining the output from a variety of algorithms into a single action. Any combination of algorithms can be used, and can be run concurrently to perform their analyses on a given state simultaneously. These features mean that EDSs offer a huge amount of flexibility for designing agents. Exploring some of these possibilities and implementing a number of EDS variations has been one of the main goals of the work of this thesis, in line with the larger goal of developing a more general GVGP agent that maintains a high level of performance on each individual game.

One type of multi-algorithm agent has seen a lot of success in the GVGA: portfolio agents. By performing analysis on the game being played, the portfolio tries to decide which algorithm is best suited for the current game. It is then generally unable to switch algorithm during the game, as they work on the idea that each game has a suitable algorithm to solve it [95]. This is behaviour that an EDS, on the contrary, is able to accomplish.

As an example, Ms. Pacman is a challenging game for agents to solve. The reason for this is that search agents typically rely on short to mid-term rewards to guide the direction of where to go. What happens if the final pills that the agent needs are on the other side of the level? Often search agents would become stuck, looking at a barren reward landscape and trying to figure out which direction to go. Ideally, the agent would want to change tactics at this point and potentially make use of a long-

range planner to find its way to the final pills and complete the level. An EDS can use the same search algorithm but have it supported by such a planner to inform the overall strategy of the agent. In this example each of the algorithms provide a different perspective to solving the same problem.

Due to the success of multi-algorithm agents in the GVGAI the question that arises is: how can a multi-algorithm approach be improved? Theoretically, adding in additional algorithms for a portfolio agent to choose from would increase its performance, but this relies heavily on accurate algorithm selection techniques. If the algorithm selected to solve a particular game is poorly chosen, then performance on that game will suffer. In contrast to portfolio agents, an EDS allows the collection of algorithms to work together to propose the action to take. By balancing the views of each algorithm, it is theorised that a more informed decision can be made about which action to take. Each algorithm plays a part in every decision, and their combined input is used to make the final decision on what the agent should do. The EDS takes advantage of emergent behaviour by layering simple behaviours to generate complexity. The goal is to create an agent with improved generality which aims to widen the range of games that can be solved, while maintaining a standard of performance in each individual game, and without the need of additional algorithms to analyse game characteristics.

This chapter will describe the different EDS agent variations that were built to test the capabilities of an EDS. First, the different variations that were tried are described in detail, along with some intuition about their inclusion and functionality. Secondly, there are a number of considerations that each variation had to take into account during design, which are also described in this chapter.

### 5.1 Architecture

Ensemble systems are primarily composed of three parts: Voices, Opinions and the Arbiter. There are a number of ways each of these elements can be constructed, and part of the work done has been to explore the possibilities and discover what modifications and combinations of elements are effective for GVGP. Figure 5.1 shows an example of the overall architecture of an Ensemble system. In this section, the different components

are described in detail and the considerations for them are outlined.

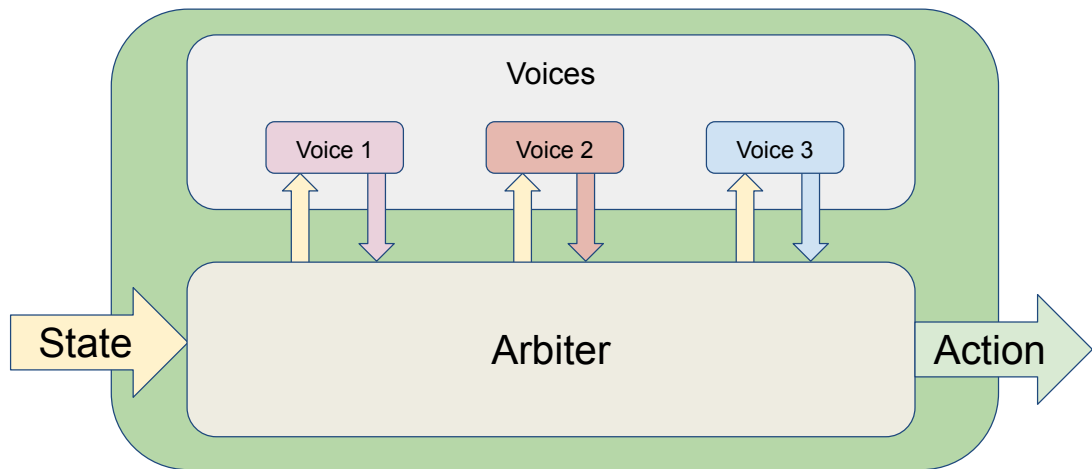


Figure 5.1: An intuitive view of the Ensemble architecture. The current state is input into the system and each Voice analyses the state. The voices then each return an opinion of what to do and the arbiter takes these opinions into account when deciding what action to take.

### 5.1.1 Voices

A Voice is an algorithm, such as MCTS, which has been adapted to work within an EDS. Voices analyse the current state independently, and essentially act as if their analysis is the only one being considered. Once analysis is complete, an Opinion is returned which could be a variety of different configurations, but typically contains the action that the Voice wants to take and a value to represent the confidence that the Voice has in that action. Additionally, a Voice need not be limited to a single algorithm, but could also be represented as a collection of algorithms in itself. Voices could also focus on discrete tasks, as well as general ones, such as avoiding dangers or collecting coins.

There are a number of ways in which the behaviour of Voices can be decomposed. This thesis considers three different decompositions: behavioural, temporal and perspective.

### 5.1.1.1 Behavioural Decomposition

Behavioural decomposition aims to use each Voice within the EDS to achieve a specific goal in a game. The work done by Rodgers et. al. [84] shows how such a system of simple behaviours can be built up into a more complex agent. Each Voice in Rodgers' *Ms. Pacman* system has a specific objective, such as collecting pills, avoiding ghosts or collecting fruit.

This kind of decomposition works well when the game is well understood, though it does not work well for a GVGP agent as the goals of the game need to be identified. How would an agent decompose such goals for any game?

### 5.1.1.2 Temporal Decomposition

Temporal decomposition aims to design each Voice with more abstract goals across time. Short range Voices were designed to focus on survival, trying to avoid a loss condition and recommending actions that take the agent away from danger. A one step look ahead agent which looks for loss conditions is a good example of a short range Voice. Long range Voices would function similar to a planner, and would attempt to find an interesting target far into the future that the agent should slowly try to achieve. The medium range Voices were designed to safely traverse the game level while maximising reward, as well as getting the agent closer to the long range Voice's goal. Candidate algorithms for the medium range Voices could be any search based algorithm such as MCTS, OLETS or RHEA.

This type of decomposition was initially tried for the proof of concept system, though a number of issues became apparent in these initial experiments. First, defining a long range Voice proved to be difficult, as the amount of abstraction required removed all detail from the game and left the Voices providing inaccurate Opinions. Secondly, while the short range Voice was effective at avoiding danger it became clear that the algorithm was double checking the immediate states from the initial state. The short range Voice would analyse those states and return an Opinion, followed by the medium range Voice performing a similar analysis on the first states.

To make the short range Voice more useful, it would need to be able to detect losing

states before the point of no return. This behaviour is difficult to describe in code in a general way and in the end it was decided to try a different kind of decomposition.

### **5.1.1.3 Perspective Decomposition**

The concept of perspective decomposition breaks down the goals of problem solving into different points of view. Games can be viewed in a number of different ways; it is not always satisfying for a player to simply win a game. Role playing games heavily encourage exploration whereas arcade games focus more on survival and achieving high scores. Simply finding a win condition is not enough to adequately solve all games.

With this in mind and the ideas explored through deceptive games perspective decomposition was selected as the way forward. Instead of trying to identify a set of general objectives for Voices to accomplish, the algorithms each provide their own point of view on what to do. These points of view are guided by the heuristics of the Voice, but also by the mechanism of the Voice's algorithm. Voices are added to the system with a specific perspective in mind, such as maximising exploration or score, and work together.

By bringing together a variety of points of view the EDS can make a more informed decision.

### **5.1.2 Opinions**

An Opinion is a set of actions labelled with values which holds the results of a single Voice's analysis. Due to the flexibility of an EDS, this could be represented in a variety of ways. A Voice may return an Opinion on all possible actions, or just a single action, or it could return an Opinion negatively, in order to express a dislike for a particular action. The data structure could be expanded in order to include a veto feature, which allows a Voice to remove an action from consideration for a number of reasons.

### **5.1.3 Arbiter**

The arbiter is the final decision making component of the EDS. Decisions are made after all of the Voices have returned their opinions about the current state of the game,

and an action selection policy is used to decide the final action.

This component is the most critical to the success of the system, as the arbiter is responsible for not only deciding the final action to take, but can also decide which Voices to use. Typical policies could be to select the action with the highest value, trusting the voices to know best about the situation, or using a democratic process to select the action most preferred by all of the Voices. Another option could be that the arbiter is a neural network which learns which Voices typically provide the best outcomes for which states, or it could be a bandit style selection process. Arbiters also have the option of activating and deactivating Voices, or adjusting weights applied to each Voice which can alter the amount of importance that each Voice is given based on experience. As an example, the EDS may have a limited number of active Voices, but also has a larger library from which to choose from, and the arbiter would decide which Voices to use based on their current performance in the game being played. If a Voice is not contributing well, then it could be swapped for another that may do better. Each Voice could have the confidence of their opinion modified by a weight, which can be adjusted by the arbiter based on how well that particular Voice has performed in the current game.

The policy chosen has a large effect on the performance of the agent [23].

Each of these components in themselves offer a wide variety of variations and modifications to experiment with. Voices could be designed to fulfill specific purposes, such as survival or goal finding, whereas the arbiter can use a variety of policies for selecting the action to take. The system also offers the flexibility to switch between Voices at run-time, essentially activating Voices that are determined to be promising, whilst deactivating those that haven't been useful to the particular game so far. Additionally, this flexibility provides an easy mechanism for updating the behaviour of an EDS, by creating a simple algorithm which achieves the desired behaviour, and then plugging it into the EDS to become part of the decision making process.

## 5.2 Variations

The flexibility offered by an EDS is one of its strongest points. The work required to thoroughly test all possibilities are beyond the scope of this research, however some exploration of those possibilities has been done.

Each of the main components of the EDS (Arbiter, Voices and Opinions) have different considerations and possibilities, and the variations that were used in these experiments are described here.

### 5.2.1 Arbiter Variations

The Arbiter can be implemented in a number of ways to tweak the performance of the EDS. As the final decision making process for the agent, important considerations have to be taken for how best to use the arbiter.

Within the GVGAI environment, there are two factors that make up the arbiter's functionality. The amount of time to return a decision, and the process of reaching the decision.

#### 5.2.1.1 Time Splitting

As the GVGAI does not allow multi-threaded solutions, deciding how best to use the time available is of paramount importance, and even more so in an EDS. For example, an MCTS algorithm will use the full 40ms available to it to return a decision, whereas each Voice within an EDS necessarily receives a portion of that, depending on how the arbiter apportions the time.

The simplest option for this is to give all Voices equal amounts of time, therefore each Voice receives  $(40/N) - 1$  ms where  $N$  represents the number of Voices in the EDS. An additional 1ms is removed in order to deal with the overhead of the arbiter making its decision. This is the most straightforward method, however it doesn't leverage all of the possibilities of the EDS. Many of the Voices may not require a full slice of time in order to conduct their analysis, whilst others may benefit from a larger slice.

Alternatively, over time the EDS can learn which Voices are providing the most

accurate and productive information about the game being played and may tune the time for each Voice differently as a result. In some cases, it may even be prudent to switch off a Voice that is not able to contribute, such as a hostile NPC avoidance Voice in games where NPCs are not present.

It is worth noting that these time splitting considerations are only necessary due to the restriction enforced by the GVGAI competition. The EDS naturally lends itself to concurrency with each Voice being ran as a separate thread, conducting their analyses at the same time rather than sequentially.

### 5.2.1.2 Action Selection Policy

Once the Voices have returned their decisions, which one should the arbiter ultimately listen to? Again, the simplest approach would be to use Opinions with values and to take the Opinion that offers the highest value, as returned by the Voices. This still offers a new insight into the game being played: the EDS can use one of  $N$  styles of play at will, but doesn't fully leverage the potential of the EDS.

A more interesting approach to action selection, is to have all of the Voices return an Opinion for each possible action. Then the arbiter is able to select actions based on a more representative view, and may even select actions that none of the Voices specifically selected. As an example, if the Voices have each returned their opinions for each action, and one Voice has highly valued action  $a1$  while another has highly penalized  $a1$ , then the system could choose their second choice action as it ranks higher overall. This also allows the possibility for Voices that are focused on negative outcomes, such as avoiding loss conditions, to penalize actions that are bad for the agent overall.

### 5.2.2 Voice Variations

One of the most interesting things about an EDS is the myriad ways in which a collection of algorithms could be used. Not only are the combinations of Voices important, but how they are designed to interact with each other also plays a part in determining the overall performance of the agent.

An issue with single algorithm systems, in a general problem solving environment,



is that they have to take into account all possibilities. Expanding a single algorithm to do things beyond their original programming is often difficult, or can potentially alter the strengths of the algorithm. In particular, the strengths of the MCTS algorithm comes from the rollouts, so if these rollouts are biased too heavily towards a particular objective in lieu of others, it may perform much worse in particular environments. With an EDS all that is needed, to introduce additional behaviour or objectives, is to design a separate algorithm that is capable of performing the additional functionality, and to add it to the system. Complexity emerges from the culmination of simpler behaviours.

The specific Voices and EDS agents are described in chapter 6.

### 5.3 Prototype Experiments

The biggest initial question at the onset of this work was: would an EDS be able to run in the GVGAI? Would the 40ms time limit and single threaded rules of the competition be too limiting?

To answer these questions, an initial experiment was run as part of the author's undergraduate individual project, which sought to confirm that an EDS could, first of all, run successfully without disqualifications and, also, wouldn't greatly sacrifice performance by splitting up the analysis time of each algorithm [99].

#### 5.3.1 The Initial Ensemble Decision System

The first EDS that was designed took the approach of assigning a specific goal for each of the three Voices, based on a simple notion of distance. There was one close range survival Voice, a mid range Voice and a long range planning Voice. The intuition behind this design was that a combination of reaction and long range planning could complement each other well, and seems to be in line with human problem solving. When someone wishes to plan a trip, they begin by deciding the end destination, and begin to slowly plot a path to that point while at the same time being aware of immediate dangers on the road as they progress. This EDS works in a similar fashion.

The long range Voice would first select an interesting observation on the map,

preferably far away in distance, and then create a basic plan to reach this target. This plan is then passed to the mid range voice, which in this case was a MCTS Voice, which would then simulate following that path, before performing the rest of its normal analysis. If the result of the path simulation is positive, the long range Voice would be informed of this, and would retain this target, otherwise it would select a new target. The short range voice is a simple one step look ahead algorithm that would simulate a single step into the future for each possible action and would veto any actions that would lead to a loss condition for the agent.

The main goal while developing this early prototype was to retain as much generality as possible. Any improvements should not be too specific to any particular problem, as this may weaken the agent in another game type. The reasoning behind this restriction is to simply keep in mind the overall goal of the GVGAI competition, developing general purpose game playing agents. To this end, a number of assumptions were made to simplify some of the design choices.

First, in regards to the long range Voice, the level was assumed to have no walls. Due to the unknown nature of the game being played, assuming that walls were obstacles would not always be the rational choice. As an example, the game *Digdug*, discussed in section 4.3.6, features walls that can be mined, creating new paths for the agent. Similarly, because the nature of the target being selected by the long range Voice cannot be discerned from a distance, it would not make sense to attempt to either positively or negatively value the long range target. If the target is beneficial, is there a guarantee that it is sufficiently better than all other options? Or if it is detrimental, should the agent ignore positive states in order to avoid it?

Secondly, in an ideal EDS the Voices do not directly speak to each other, but instead use the arbiter to pass information, which then makes decisions on what to do with it. This prototype, on the contrary, had direct communication between the long and mid range Voices, to ease the implementation. Also, any modifications that were done to theoretically improve performance were done in a manner which retained generality. If a behavioural modification could be made, which would improve performance for, as an example, puzzle games, then, unless it could be shown to not impact the performance

of other games, it wasn't included.

Lastly, the short range Voice is able to use a veto on all actions from any other Voice. If the short range Voice spots a threat, it will override the rest of the decision making process and take control. While this isn't necessarily a problem, it is preferable to allow the Voices to perform their analysis and perhaps use the short range Voice as a soft veto, to disallow analysis down a particular action, allowing deeper searches down the other options.

As an initial experiment, this first system not only functioned well within the constraints of the competition, but showed results that were better than the sample agents for some games. This is important, as the Voices were based on the sample agents, and to see a strong performance like this is an encouraging observation [99].

With the proof of concept working within the GVGAI, the next step was to determine a set of games to experiment with. The reason this became important is that much of the previous work had been done on different sets of games, and it would often become difficult to compare individual results across papers. In addition, the ranking system used within the GVGAI, ranking by win rate then score and finally time taken, raised some questions. If an agent achieves a poor score, but wins the game is that as desirable as an agent that scores exceptionally well but doesn't win as often? Are there games where it would actually be detrimental to reach a winning state early, or where the score leads the agent away from a win state? Exploring the set of games within the GVGAI gives some examples of games which do not easily fit into the ranking formula of the competition: for example *Butterflies* can be won easily by agents but improved performance can lead to a better score.

Finding such games raises a new question: how much of an effect does the set of games chosen to evaluate agents have on the ranking of those agents? The notion of games that deliberately mislead agents was then explored, and some deceptive games were designed and implemented in the GVGAI framework. These games then formed the basis for further experiments to look into creating effective ensemble agents which are explained in chapter 6.

## 5.4 Chapter Summary

This chapter introduced the main focus of investigation of this thesis, namely the Ensemble Decision System. Each component of the system was explained in detail and an in-depth explanation of how they work together is provided.

A number of proposed architectures for such as system are outlined along with justifications for their inclusion/exclusion in the final experiments.

## Chapter 6

# Experimental Setup

This chapter looks at the final experiments that were conducted for this thesis. First, the EDS using a perspective decomposition approach was tested, using the set of GVGAI games identified in chapter 4. Second, the ideas for improving the EDS are described further, followed by a comparison with some of the portfolio agents from the GVGAI. Lastly, the final series of experiments to test a variety of EDS architectures and determine their viability for GVGP are presented.

### 6.1 Ensemble Decision Systems for General Video Game Playing

With the continuous information gain technique selecting the most discriminatory games, and the issue of deceptive games highlighted, the time came to design an agent taking these factors into account. With the previous successes of portfolio agents and the ensemble agents developed for AlphaGo [26] and Ms Pacman [84], implementing an Ensemble agent for the GVGAI was the natural next step [23]. This work has had one publication at the Conference for Games 2019 under the title *Ensemble Decision Systems for General Video Game Playing* and can be found in appendix D.

When designing an EDS there are a number of questions to answer: what voices should be used? How should actions be selected? What is the best way to use the 40ms decision time?

The first attempt to address these questions was to develop a more naive EDS which would simply make use of a collection of current agents from the GVGAI competition, to ensure that an EDS is able to perform satisfactorily within the environment.

The intuition behind this EDS was that algorithms focused separately on exploration and winning can be combined usefully. Exploratory work was done to look at this question by Perez et al., which looked at applying multiple heuristics simultaneously within a single algorithm, and showed that this combination of different perspectives increased win rates of the MCTS agent [100].

Another source of inspiration was the work carried out by Guerrero-Romero et al., which looks at using different heuristics across five sample GVGAI agents to determine which heuristics work best with which algorithms [98]. Guerrero-Romero et al. created four different heuristics which focused on different, general, objectives of video games such as exploration and gaining knowledge about the game. These heuristics are used by five GVGAI agents (MCTS, OLETS, RHEA, RS and OSLA) and tested across a set of GVGAI games.

Combining these ideas was a natural next step as the EDS provides a mechanism for doing this. Instead of using different heuristics in a single algorithm, the EDS allows the algorithms which are best for each objective to work together to make more informed decisions.

Several EDS variations were created with these ideas in mind and tested against the games chosen in previous chapters. The full list of agents tested is shown in table 6.1. Each agent was played against the five levels of each game, and given 50 iterations of each level. The full list of games used for these experiments are shown in table 6.2. These experiments were performed using a laptop running Linux Mint with an Intel i7-8565U CPU with 16GB of RAM.

### 6.1.1 Heuristics

There are four heuristics that were designed and created by Guerrero-Romero et al [98], each focusing on a specific objective. This section briefly describes these heuristics and how they function.

### 6.1.1.1 Win Maximisation

The Win Maximisation Heuristic (WMH) focuses on finding states that result in a win for the agent. It applies a large bonus to any observed states that result in a win, and a large penalty to states that result in a loss.

Upon reaching a terminal state, if the agent has lost the heuristic returns a value of  $-10,000$ , if the agent has won then the heuristic returns  $10,000$ . In the cases of non terminal states, the heuristic returns the score that would be achieved from taking that action. This is done by subtracting the evaluated score from the current score of the agent. Pseudocode of the algorithm is shown in algorithm 1.

---

**Algorithm 1** Win Maximisation Heuristic (WMH)

---

```
if isGameOver() and isLoss() then
    return -10000
else if isGameOver() and isWin() then
    return 10000
return newScore - currentScore
```

---

### 6.1.1.2 Exploration Maximization

The Exploration Maximisation Heuristic (EMH) focuses on bringing the agent to states that it has not visited previously. States that have previously been visited are slightly penalised, whereas states which have not been seen before, or have rarely been visited are given a small bonus. This version differs slightly from the version presented by Guerrero-Romero et al. in that winning is rewarded instead of penalised.

The EMH has a similar evaluation mechanic for win and loss states as the WMH heuristic, but does not take score into account. If the desired action leads the agent out of bounds of the game level then the heuristic returns  $-10,000$ . If the position is one that the agent has not visited before then the heuristic returns  $100$ . If the position has been visited before, within its memory, then the heuristic returned  $-50$ . Finally, the heuristic returns  $-25$  for all other positions. Pseudocode of the algorithm is shown in algorithm 2.

---

**Algorithm 2** Exploration Maximisation Heuristic (EMH)

---

```
if isGameOver() and isLoss() then
    return -10000
else if isGameOver() and isWin() then
    return 10000
if isOutOfBounds(newPosition) then
    return -10000
if not hasVisitedBefore(newPosition) then
    return 100
if isSameAsCurrentPosition(newPosition) then
    return -50
return -25
```

---

**6.1.1.3 Knowledge Discovery**

The Knowledge Discovery Heuristic (KDH) tries to interact with as many objects in the game as possible to learn what happens in those situations. It does this by trying to trigger as many interactions between entities on the level as possible, and trying to learn how new entities are spawned, if any are.

The heuristic values states that have either new sprites in them, or new interactions between sprites. A sprite is considered to be acknowledged if the avatar has observed it in the real game or in the forward model for the first time. Any time that a new sprite is acknowledged, the heuristic returns 10,000. If no new sprites are acknowledged, then the heuristic will return a different value, depending on the occurrence of an interaction and its type. If the interaction has not been seen before, then the heuristic returns 1000. If the interaction involves the avatar touching another sprite, then 50 is returned. In many games the agent is able to create new sprites, such as projectiles, which can interact with other sprites. In this case, the heuristic returns 50. If no interaction occurs, then the heuristic returns  $-25$ . Pseudocode of the algorithm is shown in algorithm 3.



---

**Algorithm 3** Knowledge Discovery Heuristic (KDH)

---

```

if isGameOver() and isLoss() then
  return -10000
else if isGameOver() and isWin() then
  return 10000
else if isOutOfBounds(newPosition) then
  return -10000
if newSpriteAcknowledged() then
  return 10000
if eventOccured() then
  if eventIsUnique() then
    return 100
  else if avatarInteraction() then
    return 50
  else if avatarProjectileInteraction() then
    return 25
return -25

```

---

**6.1.1.4 Knowledge Estimation**

The Knowledge Estimation Heuristic (KEH) has a similar objective to the KDH but attempts to estimate what the interaction will achieve, instead of discovering it through experience. The objective of this heuristic is to better understand the critical components of the game being played. KEH tries to determine what the win and loss conditions are, as well as what score change might happen when taking certain actions.

Unlike the other heuristics, KEH penalises win conditions, though less than a loss condition. The main difference between KEH and KDH is that KEH returns values based on what it believes the chances of winning are. These values range from 0 – 100.

The pseudocode for KEH is shown in algorithm 4.

---

**Algorithm 4** Knowledge Estimation Heuristic (KEH)

---

```

if isGameOver() and isLoss() then
  return -10000
else if isGameOver() and isWin() then
  return -5000
else if isOutOfBounds(newPosition) then
  return -10000
if newSpriteAcknowledged() then
  return 10000
if eventOccured() then
  if eventIsUnique() then
    return 100
  return rewardForTheEvents() ▷ Value between 0 - 100
noOfInt = totalNStypeInteractions()
if noOfInt == 0 then
  return 0
return  $\frac{-50}{(200 \times noOfInt)}$ 

```

---

### 6.1.2 Sample Agents

A number of sample agents are provided by the GVGAI framework as a basis for building competitor agents. These are versions of different algorithms, such as MCTS and RHEA, and demonstrate to developers some of the functionality provided by the framework, as well as examples for how to interact with the framework API. In this work, the sample agents were used as a basis for testing the capabilities of an Ensemble system, as currently they are well understood, and using them as opposed to a new algorithm removes some variables for performance analysis.

For these experiments, none of the sample algorithms used for comparison were modified from the version provided by the framework. For those that were used in an EDS, a number of modifications were necessary in order for them to meet the requirements of the system. First, they were adapted to be able to use different heuristics for their analysis, based on the work of Guerrero-Romero et al [98]. Secondly, in order to fit within the EDS, the algorithms were altered to return an Opinion instead of a single action. As described in section 5.1.2, Opinions are a single action which also holds a value associated with that action. This value represents the confidence that

the Voice has in that action. A Voice can be required to either return one Opinion, or several Opinions. In the case of returning several Opinions, the Voice has to provide one Opinion for each possible action. For these experiments, the single Opinion option is used.

The process of transforming a GVGAI agent into a Voice for the EDS is straightforward. Whenever the agent would return an action, it needs to instead create a new Opinion and add that action to the Opinion, along with the value that it gives that action. This value is typically the heuristic evaluation of the agent, in order to preserve the perspective of that agent.

The sample agents that were used are described in this section.

#### 6.1.2.1 sampleRandom

The *SampleRandom* agent does not have any deliberative or reasoning aspects and simply selects an action at random from those that are available.

This agent was only used for benchmark purposes and was not used in an EDS.

#### 6.1.2.2 One Step Look Ahead (OSLA)

The *OSLA* agent uses the forward model to analyse the state reached if each available action is taken once, and selects the best action to take. By default, the *sampleOneStepLookAhead* agent uses the *SimpleStateHeuristic*, which is a basic heuristics provided by the GVGAI framework. The *SimpleStateHeuristic* gives a large bonus to states that result in the agent winning, and a penalty if a loss is observed. Other than wins and losses, the heuristic also takes into account the NPCs present in the level, as well as portals which may spawn new NPCs or lead the avatar to a new area.

This agent was only used for benchmark purposes and was not used in an EDS.

#### 6.1.2.3 Open-Loop Monte-Carlo Tree Search (OLMCTS)

The *OLMCTS* agent is a variation of MCTS that is designed to work better in stochastic environments [58]. Instead of storing states at each node of the search tree, the forward model is used to reevaluate actions. The *sampleMCTS* agent provided in the framework

makes use of the OLMCTS variation and has a rollout length of 10 and a C-value of  $\sqrt{2}$ . The default evaluation function takes into account the current score that the rollout has achieved, as well as applying a large bonus or penalty depending on whether the agent has won or lost.

This agent was used both as a benchmark and as a Voice in EDS variations.

#### 6.1.2.4 Open-Loop Expectimax Tree Search (OLETS)

The *OLETS* agent is based on the Hierarchical Open-Loop Optimistic Planning (HOLOP) algorithm but has been improved specifically for stochastic environments [101]. OLETS does not rely on roll outs in order to evaluate nodes, but instead relies on the open loop expectimax method to assign bandit scores. The heuristic used to evaluate states assigns a large reward for wins and a penalty for losses, and simply returns the score achieved in other situations. Further details of the algorithm are described in [15].

The version provided by the framework has a playout length of 5 and the default heuristic used takes into account whether or not the agent has won, as well as the current score achieved.

This agent was used both as a benchmark and as a Voice in EDS variations.

#### 6.1.2.5 Rolling Horizon Evolutionary Algorithm (RHEA)

The *RHEA* algorithm uses an evolutionary approach. It maintains a population of individuals, which represents a plan of actions to execute. This plan is then simulated and evaluated with the forward model [60]. After the evaluation is performed, the first action in the best plan is executed by the agent. The *sampleRHEA* provided in the framework uses the provided *WinScoreHeuristic*, and evolves individuals keeping 10 of them at a time. The *WinScoreHeuristic* awards win states with a large bonus, those with a loss a large penalty, and all other states return the score. It applies mutation and crossover as part of the evolutionary process until time runs out. The number of individuals that it will produce is dynamic, as it will produce individuals until the analysis time limit of 40ms is nearly over.

This agent was used both as a benchmark and as a Voice in EDS variations.

### 6.1.2.6 Random Search (RS)

The *RS* algorithm is another evolutionary approach, based on RHEA. Instead of evolving individuals however, they are generated randomly. The *sampleRS* implementation provided by the framework produces individuals with a length of 10, and produces as many as possible within the 40ms analysis time.

This agent was used as a benchmark and as a Voice in EDS variations.

### 6.1.3 Ensemble Agent Variations

In total, six Ensemble agents were created for these experiments. These agents use different algorithms which are themselves using the heuristics described in section 6.1.1. The initial idea for these experiments was to develop some EDS variations that could combine the EMH and WMH heuristics, to develop an agent that could better balance exploration and exploitation. An EDS with two voices, one using the EMH and one using the WMH was tried, with a few further variations to try different algorithms with the heuristics to see which performed best together. All of the EDS Voices return a single Opinion when returning their output.

The EDS agents that were used for these experiments are described in this section. The EDS agents use a naming convention to help differentiate them. As an example, *OLETSExpSc* stands for OLETS Explore Score, which is a two Voice EDS using OLETS for both Voices but one uses the WMH and the other uses the EMH.

#### 6.1.3.1 BestFour

This variation uses four Voices, one for each of the heuristics described in section 6.1.1. The Voices used are OLETS with WMH, RS with EMH, RS with KDH and OLETS with KEH. Each of those algorithms were identified as the best for their respective heuristic [98]. Each Voice is given an equal share of the analysis time, meaning that they each receive roughly 9ms of time each to compute.

The action which has the highest value, returned by the Voices, is then selected by the arbiter as the action to take.

BestFour did not perform particularly well in the experiments, though it did manage to win a large number of the games when compared to the sample agents. This is likely due to the heavy dilution of the analysis time for each of the 4 Voices, leading to none of the Voices having enough time to adequately analyse the states.

### 6.1.3.2 BestFourDiplo

This variation uses four Voices, which are the same as BestFour, but uses a diplomatic arbiter instead. The main difference is that instead of selecting the action with the highest value, each Voice gets a vote on which action to take, and the most popular action is taken. In the case of no clear majority, a random action is selected.

BestFourDiplo performed better than BestFour, with a higher win rate and being able to win an additional game over BestFour. It did not perform as well as the EDS agents that used fewer Voices, showing that splitting the analysis time too much may have a detrimental effect on performance.

### 6.1.3.3 BestExpSc

This variation uses two voices, OLETS with WMH and RS with EMH. The idea with this variation was to see if simply combining the winning and exploration heuristics would produce a more general agent, whilst maintaining a similar level of performance as the individual algorithms. For the action selection policy, the action with the highest value is selected.

This variation performs quite well and manages to complete more games than BestFourDiplo with a higher overall win rate.

### 6.1.3.4 MCTSExpSc

This variation uses two voices, but instead of working with the best algorithms for each heuristic, the MCTS algorithm is used for both WMH and EMH. This gives a clearer performance comparison with the sampleMCTS agent. The action with the highest value is selected.

Interestingly, this variation is able to win a larger number of unique games than the *sampleMCTS* agent, as seen in table 6.3. It does suffer a performance hit in the individual games, likely due to having less analysis time than the *sampleMCTS*. Given the aim of the GVGAI, this is an encouraging result, particularly given that the EDS at this stage is naively constructed.

### 6.1.3.5 OLETSExpSc

This variation used two voices, OLETS with EMH and OLETS with WMH. This gives another interesting comparison to a single algorithm, in this case OLETS, which was a highly successful algorithm in the first iteration of the GVGAI competition [15]. The action with the highest value is selected.

Out of the EDSs created for this work, this one performed the best overall and, in comparison to the OLETS agent, is able to complete a wider range of games, while only dropping 2.1% in win rate. Interestingly, the last two EDS agents are able to solve games that the single algorithms were not able to solve at all during experimentation.

### 6.1.3.6 OLETSExpScSerial

This variation uses a similar setup to *OLETSExpSc*, with the same voices and heuristics, but uses a serial full time-step arbiter instead.

The main difference with this arbiter is that it tries to give each Voice 40ms of decision time, by using a time-step per Voice. It does this by returning nil actions until each Voice has analyzed the current state. After  $N$  timesteps, where  $N$  is the number of Voices, an action is returned based on the action selection policy, which in this case was the highest value action. The reason for this type of arbiter is that it would allow all of the Voices to each have 40ms of analysis time, without breaking the rules of the GVGAI competition and introducing concurrency.

While *OLETSExpScSerial* achieved good scores, it was not quite as good as the *OLETSExpSc* agent and actually lost some games that *OLETSExpSc* was able to win.

In the original paper, *Ensemble Decision Systems for General Video Game Playing*, this agent was known as *OLETSExpScAsync* and the name has been changed to better

reflect the operation of the EDS.

The full list of controllers used are listed in table 6.1 and the games that they were experimented on are listed in table 6.2.

Table 6.1: Full list of controllers used divided into EDS and Sample Agents.

<b>EDS Agents</b>	<b>Sample Agents</b>
BestExpSc	OLETS
BestFourDiplo	sampleMCTS
BestFour	sampleOneStepLookAhead
MCTSExpSc	sampleRandom
OLETSExpSc	sampleRHEA
OLETSExpScSerial	sampleRS

Table 6.2: The Games selected from the GVGAI Framework (games in bold are deceptive)

<i>Aliens</i>	<i>Avoidgeorge</i>	<i>Bait</i>	<b><i>Butterflies</i></b>
<i>CamelRace</i>	<i>Chase</i>	<i>Chopper</i>	<i>Crossfire</i>
<i>Digdug</i>	<i>Escape</i>	<i>Freeway</i>	<i>Hungrybirds</i>
<i>Infection</i>	<i>Intersection</i>	<b><i>Invest</i></b>	<i>Labyrinthdual</i>
<b><i>Lemmings</i></b>	<i>Missilecommand</i>	<i>Modality</i>	<i>Plaqueattack</i>
<i>Roguelike</i>	<i>Seaquest</i>	<b><i>SisterSaviour</i></b>	<i>Survivezombies</i>
<i>Tercio</i>	<i>Waitforbreakfast</i>	<i>Watergame</i>	<i>Whackamole</i>

#### 6.1.4 Results

In table 6.3, the agents are ranked using the same F1 style system as is used for the GVGAI competition. For each game, the agents are awarded points based on their performance using win rate as the first indicator, then score if win rates are tied, and finally time taken if a tie remains. The points that are given out are 0, 1, 2, 4, 6, 8, 10, 12, 15, 18, 25 with the agent earning first place receiving 25 points, the second place agent earning 18, and so on. Two other tables are also shown which rank the agents by the average win rate of the agents across all of the games (Table 6.4), and the number of games that each agent was able to achieve at least 1 win (Table 6.5).



Table 6.3: Controllers ranked by total number of F1-points. The EDS agents are highlighted in blue. The OLETS sample agents and OLETS EDS agents are the top performers in terms of F1 points, though the OLETSExpSc agent is able to win more unique games than the sample OLETS agent. This trend can also be seen in the MCTS agents.

	<b>Controller</b>	<b>F-1</b>	<b>%Wins</b>	<b>#Games</b>
<b>1</b>	OLETS	501	55.69%	25
<b>2</b>	OLETSExpSc	463	53.59%	27
<b>3</b>	BestExpSc	318	42.86%	24
<b>4</b>	OLETSExpScSerial	316	45.52%	24
<b>5</b>	sampleMCTS	253	40.00%	19
<b>6</b>	sampleRS	237	39.76%	22
<b>7</b>	BestFourDiplo	213	33.02%	23
<b>8</b>	sampleRHEA	194	36.71%	23
<b>9</b>	BestFour	182	28.93%	22
<b>10</b>	MCTSExpSc	181	33.29%	22
<b>11</b>	sampleOneStepLookAhead	88	21.48%	12
<b>12</b>	sampleRandom	29	9.29%	15

Table 6.4: Controllers ranked by total number of Win Rate (%Wins). The EDS agents are highlighted in blue. The OLETS sample agents and OLETS EDS agents are the top performers in terms of F1 points, though the OLETSExpSc agent is able to win more unique games than the sample OLETS agent. This trend can also be seen in the MCTS agents.

	<b>Controller</b>	<b>F-1</b>	<b>%Wins</b>	<b>#Games</b>
<b>1</b>	OLETS	501	55.69	25
<b>2</b>	OLETSExpSc	463	53.60	27
<b>3</b>	OLETSExpScSerial	316	45.52	24
<b>4</b>	BestExpSc	318	42.86	24
<b>5</b>	sampleMCTS	253	40.00	19
<b>6</b>	sampleRS	237	39.76	22
<b>7</b>	sampleRHEA	194	36.71	23
<b>8</b>	MCTSExpSc	181	33.29	22
<b>9</b>	BestFourDiplo	213	33.02	23
<b>10</b>	BestFour	182	28.93	22
<b>11</b>	sampleonesteplookahead	88	21.48	12
<b>12</b>	sampleRandom	29	9.29	15

Table 6.5: Controllers ranked by total number of unique games won (#Games). The EDS agents are highlighted in blue. This table shows that the EDS agents perform well when looking at the range of games that they are able to solve, in comparison to the sample agents.

	<b>Controller</b>	<b>F-1</b>	<b>%Wins</b>	<b>#Games</b>
<b>1</b>	OLETSExpSc	463	53.60	27
<b>2</b>	OLETS	501	55.69	25
<b>3</b>	OLETSExpScSerial	316	45.52	24
<b>4</b>	BestExpSc	318	42.86	24
<b>5</b>	sampleRHEA	194	36.71	23
<b>6</b>	BestFourDiplo	213	33.02	23
<b>7</b>	sampleRS	237	39.76	22
<b>8</b>	MCTSExpSc	181	33.29	22
<b>9</b>	BestFour	182	28.93	22
<b>10</b>	sampleMCTS	253	40.00	19
<b>11</b>	sampleRandom	29	9.29	15
<b>12</b>	sampleonestepllookahead	88	21.48	12

This definition of generality has some problems, though as this was an initial set of experiments it was considered sufficient. An improved measure of generality is defined later for the experiments in section 6.3.

The first thing of note is that the sample OLETS algorithm takes the top position, and second place goes to an ensemble agent which similarly uses the OLETS algorithm, OLETSExpSc which is described in section 6.1.3.5. This difference in ranking is due to the OLETS algorithm outperforming the ensemble OLETSExpSc more consistently on the games that OLETS can perform well. OLETS can spend the entirety of its 40ms analysis time searching deeper into the forward model, whereas OLETSExpSc has to divide that 40ms up between the two voices, and is not able to search as deeply as a result. While noting this, it is also clear that the difference between the two agents is small, with OLETS earning a win rate across all games of 55.69% compared to 53.59% for OLETSExpSc, a drop of 2.1%. This goes some way to show that performance is not greatly diminished by splitting up the analysis time between two algorithms. This can also be seen, though the difference is more observable when comparing the sampleMCTS algorithm and the MCTSExpSc, with win rates of 40% and 33.29% respectively, showing

a drop of 6.71%. These drops in individual game performance are offset somewhat by the increased generality of the ensemble agents. OLETSExpSc is able to win 27 of the 28 games, whereas OLETS is able to win 25. A similar observation is made between MCTSExpSc winning 22 of the games and MCTS winning only 19.

Another interesting note regarding these experiments begins to make itself clear when analyzing performances on individual games. In figure 6.1 the win rates of each agent across each individual game is shown. This perspective highlights that not all games are equally challenging, yet are able to be won by ensemble agents where the base agents are not able to win. Of particular note are the games *SisterSaviour* and *Lemmings*. Similarly, some instances where the base algorithms are able to outperform their ensemble counterparts are also highlighted. In the case of the OLETS algorithm, the games *Chopper* and *PlaqueAttack* show this most clearly. In the case of *PlaqueAttack*, all levels that are played on are particularly large and this causes significant problems for the ensemble as it makes use of an exploration heuristic which highly values new states. This leads to the ensemble taking too long exploring the large level, and results in having no time remaining to complete the objectives of the game which are outlined in section 4.3.13. While this does lower the win rate of the agent on *PlaqueAttack* it does not result in the agent losing all of the time. This is due to the other perspectives within the EDS not being fooled by the game and sometimes being able to steer the agent in the correct direction. Any individual perspective can be fooled, but it is much harder for a problem to deceive all of the perspectives.

The win rates of agents do not tell the entire story, as certain games have deceptive qualities which uses the reward structure of the game to deceive agents [21]. In figure 6.2 the total score each agent achieved on each game is shown. While this encompasses similar information as figure 6.1, some cases show that there are differences in how much an agent was able to win in each game. Games such as *Bait*, *SisterSaviour*, *WaitForBreakfast* and *Escape* have a more pronounced difference, and in the case of *Lemmings* the results indicate that this may be a deceptive game, as the win rate and score do not match up. In particular when looking at *Lemmings* we see that while OLETS attains some of the highest scores, it isn't able to win any games, where as

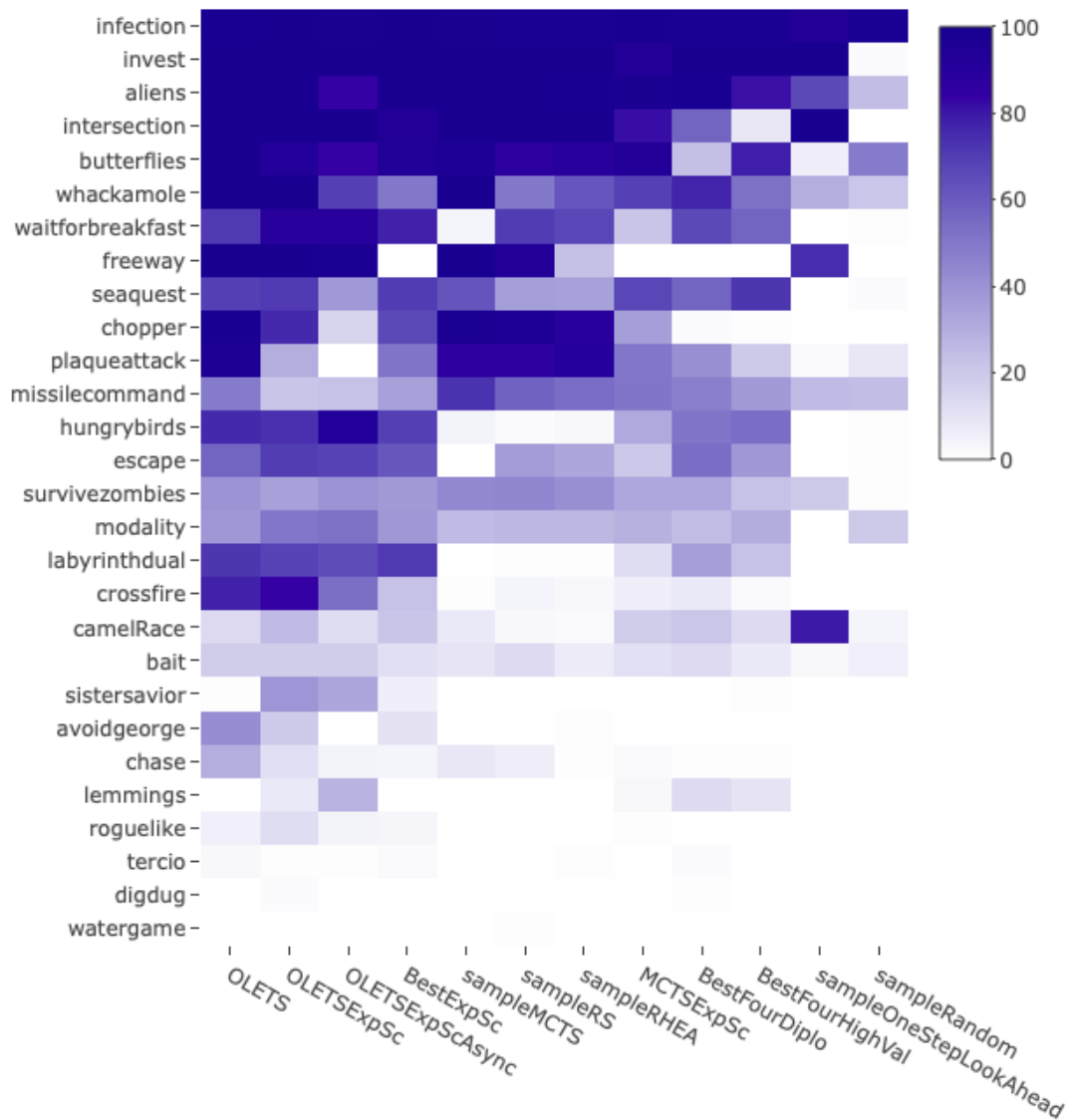


Figure 6.1: Total percentage of wins (0–100%) of each controller by game. Some of the main observations in this data are the SisterSavior performance, where only the EDS agents are able to achieve wins. OLETSExpSc in particular has a wide distribution of wins across the majority of the games.

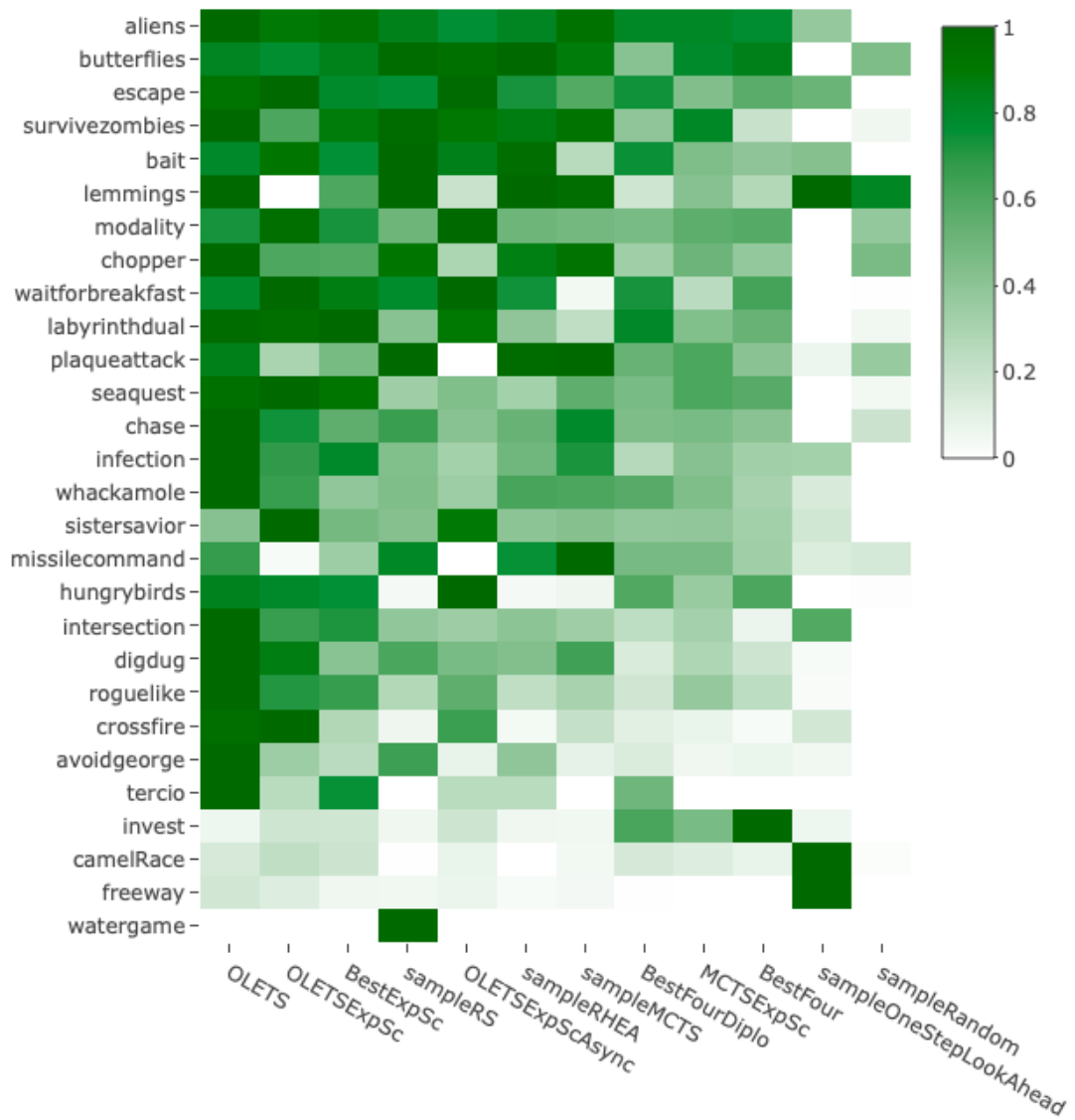


Figure 6.2: Average scores per game. To be able to make a comparison between the different games, the scores have been normalized, taking the maximum and minimum average scores obtained in each game as limits for that game.

OLETSExpSc manages to achieve some wins with a lower score.

The graphs shown here maintain the old name of the OLETSExpScSerial agent, namely OLETSExpScAsync.

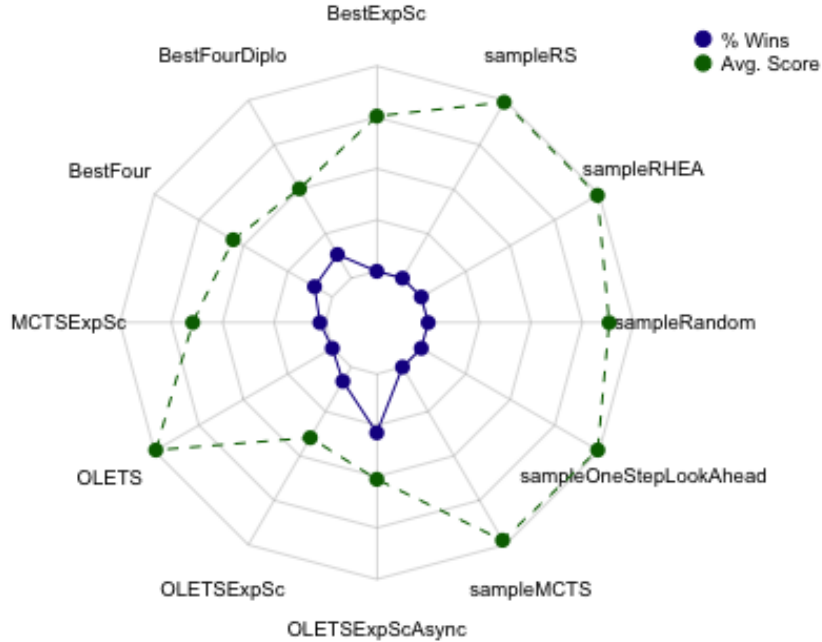


Figure 6.3: *Lemmings* stats per agent. The percentage of wins is in range  $[0, 100\%]$  and the average of score is relative to the maximum and minimum scores achieved in the game (range:  $[-143, 1]$ ). Values have been normalised in this figure. The most interesting thing of note in this figure is that achieving high scores appears to correlate with a lower win rate.

To better understand some of the differences in individual games, a deeper analysis into selected games was performed. In figure 6.3 each individual agent’s performance is displayed as two points on a spider graph, win rate and average score. Interestingly, agents with a high average score do not achieve wins where as those with lower average scores do win, though not always. This is due to the unusual scoring mechanism used in *Lemmings* where progress results in a reduction in score, and so agents that are efficient with their moves tend to achieve higher win rates. Another point of note is that none of the sample agents are able to achieve wins on this game, while many of the ensemble agents are capable of winning, indicating that more complex behaviour is emerging from the simpler Voices within the EDS. A similar example is seen in the deceptive

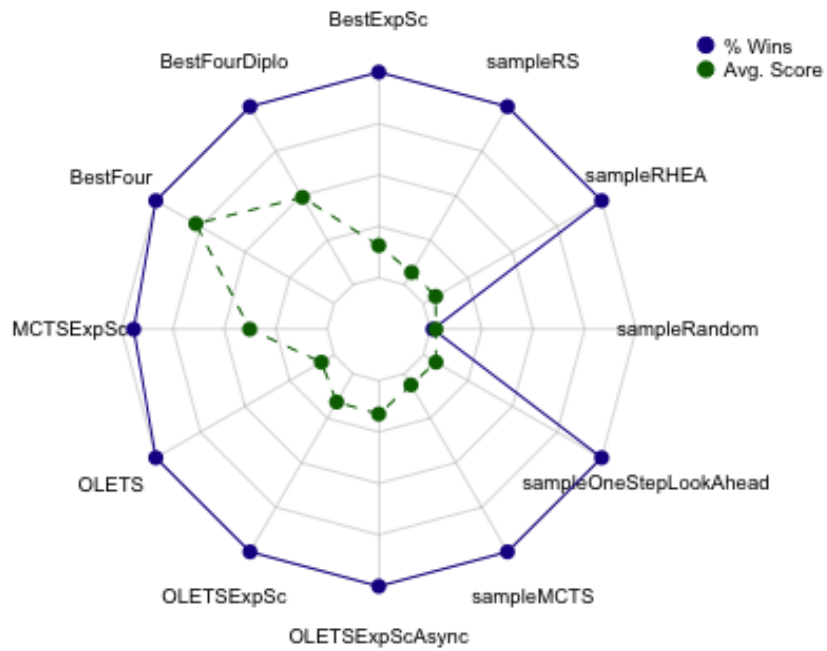


Figure 6.4: *Invest* stats per agent. The percentage of wins is in range  $[0, 100\%]$  and the average of score is relative to the maximum and minimum scores achieved in the game (range:  $[-7, 161]$ ). Values have been normalized in this figure. The results for this game show that while winning is relatively easy, achieving a high score is more challenging.

game *Invest* shown in figure 6.4, though in this case winning is easy while achieving a high score is challenging. As a deceptive game, *Invest* uses the reward structure to draw agents away from good performance, which happens in the case of sample agents. The ensemble agents are each able to achieve higher than baseline scores, with BestFour performing better than all of the other agents. *Invest* punishes agents that are not able to sacrifice current score for future reward, and the exploratory nature of the ensemble agents allows them to overcome this deception due to the presence of the exploration perspective.

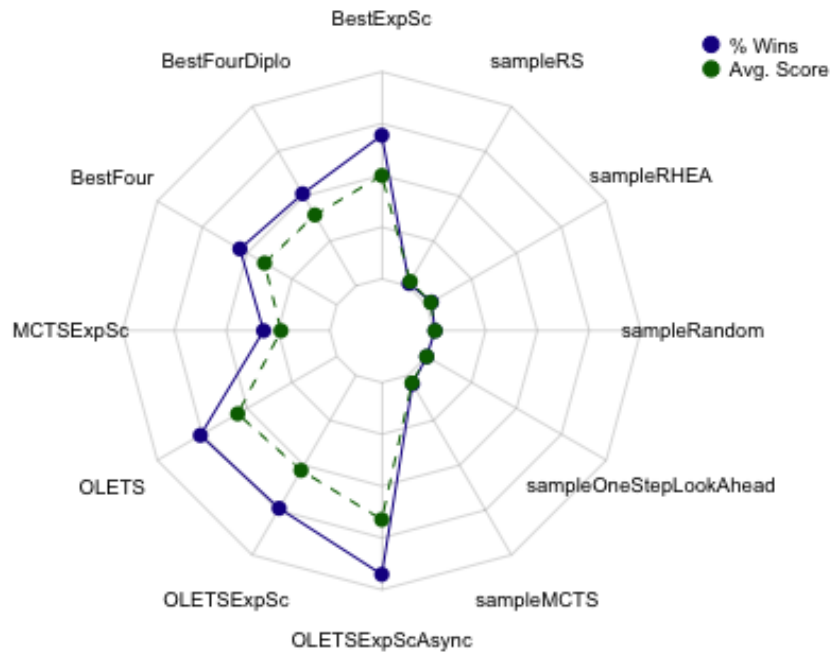


Figure 6.5: *Hungrybirds* stats per agent. The percentage of wins is in range  $[0, 100\%]$  and the average of score is relative to the maximum and minimum scores achieved in the game (range:  $[0, 140]$ ). Values have been normalized in this figure. These results show that in this game there is a correlation between score and win rate. EDS agents appear to perform particularly well here.

The game *HungryBirds* is another example of a game where ensemble systems are able to complete games that the sample algorithms are mostly not able to complete. In figure 6.5, we can see that OLETS is the only sample agent able to perform well at this game, though is outperformed by OLETSExpScSerial. Interestingly, the MCTS ensemble agent, MCTSExpSc performs very well, compared to sampleMCTS which is



not able to achieve any score or win any games. The reason for this behaviour is due to the exploration perspective within MCTSExpSc which allows the algorithm to find new states which are not easily achieved by simply looking for winning states.

Moving on from these results, a number of other avenues opened up for further exploration. Of particular interest was to directly compare portfolio agents with ensemble agents. As the two systems both take a multi-algorithm approach to tackling GVGP, the question is: is it better to combine algorithms in an EDS or select the correct single algorithm for each game? With this in mind, some further experiments were performed to answer this question and others.

## 6.2 Design of Further Experiments

This section explains the final experiments that were performed. It describes the motivations for conducting these specific experiments, as well as providing the details for reproduction. A description of each experiment is provided, and finally the results are discussed.

### 6.2.1 EDS Improvements

One of the drawbacks of the previous EDS experiments is that the Voices only return an Opinion for a single action. If portfolio agents can be described as selecting which agent to use for each game, then the earlier EDS systems can similarly be described as selecting which agent to use at each time-step. The next step is to experiment with more sophisticated systems that take allow Voices to express Opinions for all possible actions. The system would need to be improved so that each Voice could return an Opinion on each possible action, rather than just a single Opinion for its preferred action. The arbiter would then be able to select the final action from the combined Opinions for each action. The most intriguing capability of this system is that it is able to select actions that none of the individual Voices would have selected if returning only a single action. If an action is rated highly by some Voices, and poorly by others,

then the arbiter may select the second best option proposed by the Voices.

A number of modifications were necessary to the existing system in order to make this system function. First, the Voices were altered to require each to return a collection of Opinions, with a value associated to each possible action. This is a significant modification to some of the Voice's algorithms, and requires significant restructuring in certain algorithms due to the nature of their analysis. The only requirement of the GVGA framework is that a single action is returned. Some algorithms are designed with this in mind and so may not be suitable as a multi-opinion Voice. For example an enforced hill climbing algorithm would not be ideal, as it only ever returns a single action. With this in mind, altered versions of the MCTS and OLETS algorithms were created, as these algorithms provide an evaluation for each action as a natural consequence of their analysis. The Opinions were also altered slightly, in order to require that a value is associated with each action. This version of the EDS is known as an *All Actions EDS*.

The action selection policies in the arbiter required some small alterations, as now they would need to work with each action being represented across multiple Opinions. The highest value action selection policy is simply a summation of the values associated with each action, and then selecting the highest value action. The democratic action selection policy serves no purpose in this system, and so was removed as an option. The random action selection policy is still viable, though requires no changes.

With this new system in place, a further set of experiments were performed, and forms the basis of comparison to portfolio agents.

## 6.2.2 Comparison to Portfolio Agents

Portfolio agents are conceptually similar to EDS agents. The main difference in their operation lies in that a Portfolio agent attempts to select the best algorithm for each game, from a collection of algorithms at its disposal, based on an algorithm selection criteria, where as an EDS ideally uses all of the Voices within its collection at every time step. Once a Portfolio agent has selected which algorithm to use, it normally sticks to this decision until the end of the game. If this were not the case, and the

agent were to change algorithm mid-game, it may then be considered as a simple EDS instead of a Portfolio agent. These experiments also presented the opportunity to test the All Actions EDS, and comparing which type of system performs better, as well as determining whether the All Actions EDS gains the increased generality over the base agents that was seen for the initial EDS experiments in section 6.1.

Experiments were ran using the games mentioned earlier in chapter 4, as well as the full list of deceptive games described in chapter 3, though the number of iterations for each level of each game was increased from 30 to 50. This was done simply to increase confidence in the findings.

The Portfolio Agents that were used for the experiments were selected from the GVGA competition submitted agents, namely YOLOBOT, Return42, ICELab and YBCriber. The reason for these agents to be used were that they have performed well in past competitions, and particularly in the case of IceLab and Return42, performed well in the deceptive games experiments. The Adriencx agent was also used for these experiments, to provide a further comparison with current top level performance agents from the competition. Unfortunately, due to run-time errors that could not be resolved, the ICELab results have not been included as the agent would crash repeatedly when trying to run it through the experimental suite.

### 6.3 Further Experiments

The section discusses the final experiments performed for this work. First, each of the EDS variations are described in detail, as well as the agents used for comparison. Finally, the results of these experiments are discussed in depth.

#### 6.3.1 Rationale behind further experiments

One of the biggest questions throughout this work has been to discover what combination of Voices, action selection policies, heuristics and so on would prove to be an effective ensemble GVGP agent. The first step was to test whether or not the system would work within a 40ms time frame. This was shown to be feasible by the earlier

experiments in section 5.3. The second step defined a set of games that would best represent a diverse variety of games, as well as adding additional games to the GVGAI library which focused on deception. Lastly, single action EDSs were built in order to determine first, if splitting up the 40ms was inherently reducing performance, and second, to get an insight into what algorithms would function together best as Voices. The final step was to build further EDSs, taking into account the findings of the previous experiments and trying to gain further insight into the capabilities of an EDS, and what an effective ensemble system would look like.

A number of additional EDS systems were developed on top of those created in section 6.1. First, an All Action EDS, which would require every Voice to return an Opinion for each possible action. This system allows the EDS to combine the Opinions in a variety of ways, to make a final decision from the input of all Voices, as opposed to selecting a single Voice's opinion to use. Secondly, experiments were conducted using a combination of both positive and negative Voices. Positive Voices would seek out desirable/winning outcomes for the agent, whereas negative Voices would look for losing outcomes. This brings an additional perspective to the EDS which had not been previously explored. An alternative approach for future work would involve implementing a veto system which could allow a Voice to remove an action from consideration by other Voices. The final set of experiments were aimed at demonstrating the potential of a multi-threaded ensemble system since EDSs naturally lend themselves to concurrent execution. It should be noted that this is not a completely fair comparison to the other systems, as they have not been fully adapted to make use of a multi threaded environment, but it is useful to see how much this change may improve the performance of the single threaded EDS.

In addition to the architectural changes of the ensemble systems, a new naming convention was developed to better describe their behaviour. An example of this is STSAO-LETSExpSc which in the previous EDS work was known as OLETSExpSc. This agent is Single Threaded (ST) Single Action (SA) using two OLETS Voices (OLETS) with two heuristics, maximise exploration (Exp) and maximise score (Sc). Multi-threaded agents begin with MT, and the All Actions EDS has AA. An example of this is MTA AO-

LETSExpSc.

The full list of agents that were used for these experiments are listed in table 6.6.

Table 6.6: The full list of agents used for the experiments, along with the type of agent they are and where to find a description of their operation.

<b>Agent Name</b>	<b>Type</b>	<b>Section</b>
adrienctx	Tree Search	2.2.1.7
OLETS	Tree Search	6.1.2.4
OLETS80ms	Tree Search	6.3.5
Return42	Portfolio	2.2.5.4
sampleMCTS	Tree Search	6.1.2.3
sampleonesteplookahead	Local Search	6.1.2.2
sampleRandom	Random	6.1.2.1
sampleRHEA	Evolutionary Agent	6.1.2.5
sampleRS	Evolutionary Agent	6.1.2.6
YBCriber	Hybrid	2.2.5.5
YOLOBOT	Portfolio	2.2.5.3
MTAAOLETSExpSc	EDS	6.3.5
MTSAOLETSExpSc	EDS	6.3.5
STSAMCTSExpSc	EDS	6.1.3.4
STSAMCTSOpPes	EDS	6.3.4
STSAMCTSOpPesExpSc	EDS	6.3.4
STSAMCTSOpPesSc	EDS	6.3.4
STAAOLETSExpSc	EDS	6.3.3
STSAOLETSExpSc	EDS	6.1.3.5
STSAYoloOLETSExp	EDS	6.3.6
STSAYoloOLETSExpSc	EDS	6.3.6

## 6.3.2 Reproduction

In order to facilitate reproduction of these experiments, the full settings used and machine specifications are provided in this section.

### 6.3.2.1 Machine Specifications

All of the experiments were performed on a Dell Inspiron 13-5378 laptop with a Intel i7-7500u CPU and 16GB of memory.

### 6.3.2.2 Experiment Specifications

Each of these experiments were ran across the full suite of games outlined in table 6.7. This list of games are largely the same as those shown in table 6.2 but has the addition of the deceptive games that were not previously selected in chapter 4. Each game was run with all 5 of their levels, 50 times each, resulting in 250 runs per agent. Visuals were disabled for the experiments, though some agents were rerun with visual output enabled in order to observe agent behaviour in specific games.

In order to mitigate some of the risks associated with the inclusion of additional games beyond the information gain games, as mentioned previously in chapter 4, separate analyses were conducted for the three sets of games: the information gain set, the deceptive games set and the entire set together. The main metrics used are the overall win rates, and the win distributions for each agent. The win distribution table shows the number of games each agent was able to win to a certain level of performance. For example, each cell under the 10% column refers to the number of games where the agent was able to win at least 10% of the total runs i.e. at least 25 out of 250 runs.

The reason that the win distribution analysis is included is to remove some of the noise in individual game performance. Many agents are able to only win a single run out of the full 250 runs which may be a part of the base algorithm's normal distribution. Similarly, noise in a game may cause agents to lose a single game. A good agent will have a high number of different games won, that descends smoothly from 10% through to 90% or doesn't decrease at all.

All of the code that was used for these experiments can be found on the author's github page [102].

It should be noted that during the experiments **Lemmings** showed some deceptive qualities and so is represented as a deceptive game. This means that the results for **Lemmings** will also be shown in the deceptive games set as well as the all games set.

### 6.3.3 All Actions EDS

The All Actions EDS is a system in which the Voices are required to return an Opinion for each action. Once each Voice has evaluated the current state, and returned their

Table 6.7: The set of games for the additional experiments (games in bold are deceptive and underlined games are from the information gain selection)

Aliens	<u>Avoidgeorge</u>	Bait	<b>Butterflies</b>
CamelRace	Chase	Chopper	Crossfire
<b>Decepticoins</b>	<b>Deceptizelda</b>	Digdug	<u>Escape</u>
<b>Flower</b>	<u>Freeway</u>	Hungrybirds	Infection
Intersection	<u>Invest</u>	<u>Labyrinthdual</u>	<b>Lemmings</b>
Missilecommand	Modality	Plaqueattack	Roguelike
Seaquest	<b>SisterSaviour</b>	Survivezombies	<u>Tercio</u>
<b>WaferThinMints</b>	Waitforbreakfast	<u>Watergame</u>	<u>Whackamole</u>

Opinions on it, then the arbiter combines the Opinions together. This combination is done by summing together the value of each Opinion that has the same action. Once the Opinions have been combined, the action with the highest value is then selected.

The most interesting difference between this system and the earlier, single action, systems is that actions do not need to be suggested by the Voices. This change allows the system to combine each Voice's unique analysis, and potentially select an action that none of the Voices would have selected as their preferred option. Figure 6.8 shows an example of how this system would reach a decision, using the inputs from each Voice and selecting to move right as its action.

Table 6.8: An action selection example with the All Actions system. The preferred move of each Voice is highlighted in blue, with the action selected by the arbiter highlighted in green.

Action	Voice 1	Voice 2	Voice 3	Total
UP	1	5	2	8
DOWN	2	1	5	8
LEFT	3	4	3	10
<b>RIGHT</b>	4	3	4	11
STAY	5	2	1	8

An important consideration of the All Actions (AA) EDS is that the evaluation of Voices are within a similar range. If one Voice returns values for Opinions which are significantly different from another, then one Voice will rule the behaviour of the

system. While some attempts to normalise these values were made for this work, this needs further exploration in future work in order to determine the optimal method for returning the value of an action.

### 6.3.3.1 Results: All Actions EDS

Table 6.9: All Actions agents ranked by win percentage across all games. The OLETS sample agent outperforms the others in win rate, though it does not win as many unique games as STSAOLETSExpSc.

All Actions Agents Win Rate - All Games		
Agent	%Wins	#Games
OLETS	59.8%	29
STSAOLETSExpSc	59.1%	31
MTAAOLETSExpSc	36.0%	27
STAAOLETSExpSc	35.7%	26

The win rates and number of unique games won for each agent across all games can be seen in table 6.9. The first thing of note is that there is a large difference in performance between the single threaded All Actions (STAAOLETSExpSc) and Single Action (STSAOLETSExpSc) agents. The *STAAOLETSExpSc* agent has a much lower win rate compared to the *STSAOLETSExpSc* agents. After some visual observation of the agent behaviour, this appears to be due to indecision in many of the games. This is likely due to an improper balancing in the values returned from the individual Voices in the AA EDS. The multi-threaded version of the AA EDS (MTAAOLETSExpSc) has shown a small improvement in win rate and number of games though is still far lower than the *OLETS* and *STSAOLETSExpSc* agents.

The win distributions across all of the games are shown in table 6.10. This table shows a clearer picture of the differences between *OLETS* and *STSAOLETSExpSc*. While *OLETS* does have an overall higher win rate by 0.7% it can be seen in the win distributions that those wins are concentrated across fewer games. Both *OLETS* and *STSAOLETSExpSc* have ten games in the 90% runs completed range, but *STSAOLETSExpSc* begins at 27 as opposed to 25 which shows that the agent appears to be



Table 6.10: All Action agent win distributions across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. The STSAOLETSExpSc and OLETS agents both perform strongly in this metric, with the OLETS agent having a smoother drop-off in performance.

Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
OLETS	25	24	24	22	21	19	16	12	10
STSAOLETSExpSc	27	25	23	20	18	18	16	13	10
MTAAOLETSExpSc	20	18	16	11	9	9	9	6	4
STAAOLETSExpSc	20	17	16	11	11	9	8	7	4

capable of playing additional games to a satisfactory level. The AA agents both have a similar win distribution, with the *MTAAOLETSExpSc* faring a little better, though still performing worse than their Single Action counterparts.

Table 6.11: All Actions agents ranked by win percentage across the deceptive games. The OLETS agent performs relatively poorly in the deceptive games compared to the EDS agents, with a lower win rate and number of unique games.

Agent	%Wins	#Games
STSAOLETSExpSc	78.3%	8
STAAOLETSExpSc	73.5%	8
MTAAOLETSExpSc	72.4%	8
OLETS	69.8%	7

The win rates and number of unique games won for each agent across the deceptive games can be seen in table 6.11. Interestingly with this particular set the *OLETS* agent falls to the bottom position, with a win rate 8.5% lower than the highest performing agent *STSAOLETSExpSc*. Even the AA agents, which performed poorly across the set of all games, have higher win rates than the *OLETS* agent on the deceptive games. Whilst this is an improvement for the AA agents, the performance is still behind the single action agents.

The win distributions across the deceptive games are shown in table 6.12. The

Table 6.12: All Action agent win distributions across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. The STSAOLETSExpSc agent has a gradual drop off in performance compared to the other agents, and only loses 2 additional games by the 90% mark.

Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
STSAOLETSExpSc	7	7	7	7	6	6	6	6	5
STAAOLETSExpSc	7	7	7	6	6	6	6	6	3
MTAAOLETSExpSc	8	8	7	6	5	5	5	5	3
OLETS	6	6	6	6	6	6	6	5	4

single action agent appears to have a much slower degradation of performance across the distribution compared to the AA agents. *MTAAOLETSExpSc* is able to complete more games to a satisfactory level initially, from the 10% to 20% range but this quickly drops off to be one of the lower agents at 90%.

Table 6.13: All Actions agents ranked by win percentage across the information gain games. OLETS is the clear winner in the information gain set of games having an equal number of unique games won, but a higher win rate.

Agent	%Wins	#Games
OLETS	59.0%	9
STSAOLETSExpSc	58.3%	9
MTAAOLETSExpSc	17.2%	6
STAAOLETSExpSc	16.9%	7

The results shown in table 6.13 begin to show where the AA agents are having trouble. The difference in win rates between the highest AA agent and the lowest single action agent is 41.1%. As mentioned previously this is likely due to an imbalance in the weightings of perspectives that works well for a single action EDS but not for an AA EDS. This is an issue that should be addressed in future work. The AA agents also appear to complete less unique games than the other agents, as shown in the win distributions for the information gain set in table 6.14.

Table 6.14: All Action agent win distributions across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs. It becomes clear in these results that the STSAOLETSExpSc is solving a wider range of games at higher performance levels, maintaining 8 unique games up to the 30% mark, but drops off more quickly than OLETS as the performance brackets go up.

All Actions Agents Win Dist - Information Gain Games									
Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
OLETS	7	7	7	7	7	6	5	4	4
STSAOLETSExpSc	8	8	8	7	6	6	4	3	3
MTAAOLETSExpSc	3	3	2	1	1	1	1	1	1
STAAOLETSExpSc	3	3	3	1	1	1	1	1	1

The win distributions table shown in table 6.14 shows the same trend, with the AA agents performing poorly across the information gain set. Interestingly though *STSAOLETSExpSc* outperforms the *OLETS* agent here up until the 70% mark before dropping one game below OLETS at 90%. The difference in overall win rate between *OLETS* and *STSAOLETSExpSc* is only 0.7% but the wins appear to be earned across a wider variety of games.

The main conclusion from this experiment is that the AA agents needs further work to determine whether or not it is a viable option as an EDS. Observations of the behaviour of the AA agents suggest that the Voices are not working cohesively together with the current implementation, and will likely need further balancing in order to function together coherently in this architecture. On the other side, the single action variation performs particularly well across all of the sets with an acceptable loss in win rate, when compared with the base agent, in exchange for greater generality. Also, all of the EDS agents perform well in the deceptive games set, which is an encouraging result as these games are designed to trick AI agents.

### 6.3.4 Optimistic and Pessimistic MCTS EDS

The Optimistic and Pessimistic EDS uses two MCTS variations. The first is a standard MCTS Voice, which is taken from the *sampleMCTS* agent described earlier in section 2.2.1.1. The second Voice is a pessimistic MCTS, which explores the search space with an inverse value function: in essence, it explores for the worst possible states and tries to find losing conditions. It is hoped that this pairing of perspectives may produce some unique behaviour, and may alter the typical behaviour of the MCTS algorithm. From an optimistic point of view, it is hoped that the generality of the algorithm will increase.

This particular variation works a little differently from other EDSs, as one Voice will be providing a negative value to actions, and the other will be positive. There are still a number of interesting modifications that can be tried with this system, and two of them are explored here: an All Actions version, which works identically to the All Actions EDS described in section 6.3.3, and a single Action version, which has each Voice return only a single action.

Three variations of this system were developed for these experiments. The first, *STSAMCTSOpPes* does not make use of the heuristics described in section 6.1.1 and simply uses the built in heuristic of the *sampleMCTS* agent. *STSAMCTSOpPesSc*, on the other hand, does make use of heuristics by using the WMH, described previously in section 6.1.1, for both Voices. Lastly, *STSAMCTSOpPesExpSc* uses a WMH heuristic for the optimistic MCTS Voice, and a EMH heuristic for the pessimistic MCTS Voice. The intuition behind this last variation is that as the pessimistic Voice is looking for the most dangerous states to be in, it should be highly exploratory, where as the optimistic Voice wants to take advantage of the knowledge gained thus far. Finally the *STSAMCTSExpSc* agent is added as a comparison to the *MCTSExpSc* agent, originally created for the earlier EDS work described in section 6.1.

#### 6.3.4.1 Results: Optimistic and Pessimistic MCTS EDS

The results shown in table 6.15 show the win rates and number of unique games won across all games. First, the *sampleMCTS* agent is still the highest win rate agent of

Table 6.15: MCTS agents ranked by win percentage across all games.

MCTS Agents Win Rate - All Games		
Agent	%Wins	#Games
sampleMCTS	46.1%	24
STSAMCTSOpPes	44.9%	26
STSAMCTSExpSc	37.3%	28
STSAMCTSOpPesExpSc	36.8%	26
STSAMCTSOpPesSc	33.5%	18

them all, though it does not win as many unique games as *STSAMCTSOpPesExpSc* which wins 26 games as opposed to 24. It is interesting to note that the *STSAMCTSExpSc* is able to win 28 games. The difference in win rate between *sampleMCTS* and *STSAMCTSOpPesExpSc* is quite large however, with a loss of 9.29%. This drop in win rate is a bit too high to be acceptable, though it is worth mentioning that the new games that *STSAMCTSOpPesExpSc* was able to win are games that the single agents struggle on, *Lemmings* and *Roguelike*.

The results for the *STSAMCTSOpPes* agent are more encouraging, with a difference of 1.2% in win rate while still being able to complete at least one run in 26 unique games.

The highest number of wins is earned by *STSAMCTSExpSc* which manages at least one win in 28 games, though the overall win rate has dropped down to 37.3%. Interestingly *STSAMCTSExpSc* is able to win 15 runs of the deceptive game *Sister-Saviour* while none of the other MCTS agents have been able to achieve any wins. This is likely due to the inclusion of the exploration perspective, though it seems that the addition of this Voice is not always enough to solve this game, as in the case of *STSAMCTSOpPesExpSc*.

The win distribution table across all games is shown in table 6.16. The first thing of note is that the *sampleMCTS* has the lowest overall drop in wins across the distribution, though initially all of the EDS agents begin with a higher number of games until the 30% column. The *STSAMCTSOpPes* agent initially has a lower number of games compared to some of the other EDS agents, but has a much better win distribution, similar to *sampleMCTS*.

Table 6.16: MCTS agents win distributions across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

<b>MCTS Agents Win Dist - All Games</b>									
<b>Agent</b>	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>	<b>70%</b>	<b>80%</b>	<b>90%</b>
sampleMCTS	17	17	16	16	15	15	14	13	10
STSAMCTSOpPes	19	17	16	16	15	13	13	12	8
STSAMCTSExpSc	21	20	14	12	11	10	8	6	6
STSAMCTSOpPesExpSc	20	17	13	12	11	10	9	7	5
STSAMCTSOpPesSc	18	17	11	10	10	9	9	8	6

Table 6.17: MCTS agents ranked by win percentage across the deceptive games.

<b>MCTS Agents Win Rate - Deceptive Games</b>		
<b>Agent</b>	<b>%Wins</b>	<b>#Games</b>
sampleMCTS	68.7%	6
STSAMCTSOpPes	68.1%	6
STSAMCTSExpSc	64.6%	8
STSAMCTSOpPesExpSc	57.2%	6
STSAMCTSOpPesSc	51.7%	6

In table 6.17 the win rates and unique games are shown for the deceptive games. The win rate gap between *sampleMCTS* and *STSAMCTSOpPes* has narrowed slightly from 1.2% to 0.6% though the number of unique games has not improved. Only the *STSAMCTSExpSc* agent earns a higher number of games, with a win rate different of 4.1% behind *sampleMCTS*. The other EDS agents have fallen behind in terms of win rate. This is likely due to the pessimistic MCTS not providing much additional information about the games being played, as evidence by *STSAMCTSExpSc* outperforming them in terms of unique games.

Table 6.18: MCTS agents win distributions across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

MCTS Agents Win Dist - Deceptive Games									
Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
<i>sampleMCTS</i>	6	6	6	6	6	6	6	6	3
<i>STSAMCTSOpPes</i>	6	6	6	6	6	6	6	5	4
<i>STSAMCTSExpSc</i>	7	6	6	5	5	5	5	4	4
<i>STSAMCTSOpPesExpSc</i>	5	5	5	5	5	5	5	4	3
<i>STSAMCTSOpPesSc</i>	6	6	4	4	4	4	4	3	2

The deceptive game win distributions are shown in table 6.18. The *sampleMCTS* agent has a very smooth distribution across the deceptive games, before seeing a drop at the 90% column. *STSAMCTSOpPes* has a very similar pattern but drops off more smoothly and retains four games as opposed to three at the 90% column. While *STSAMCTSExpSc* does start to drop games sooner than the other agents, it begins with seven as opposed to six for the other agents and finishes with four. This is an indication that *STSAMCTSExpSc* is getting wins across a larger number of games, where as the other agents appear to be getting most wins from a core set of games that they perform well at.

Table 6.19 shows the win rates for the MCTS agents across information gain games. The ranking of results are similar to both the all games and deceptive games tables, though the unique games won appears to be quite different. The agents with the highest number of games are *STSAMCTSOpPes* and *STSAMCTSExpSc* with seven

Table 6.19: MCTS agents ranked by win percentage across the information gain games.

MCTS Agents Win Rate - Information Gain Games		
Agent	%Wins	#Games
sampleMCTS	39.8%	5
STSAMCTSO <sub>p</sub> Pes	37.6%	7
STSAMCTSO <sub>p</sub> PesSc	25.6%	4
STSAMCTSO <sub>p</sub> PesExpSc	22.9%	6
STSAMCTSE <sub>xp</sub> Sc	22.1%	7

each, though the win rate of *STSAMCTSE<sub>xp</sub>Sc* is 15.5% lower.

To better understand these differences in the unique games, table 6.20 shows the win distributions of the agents across the information gain games. The *sampleMCTS* clearly has the best distribution and manages to not drop any games at all. The only agent that comes close to this distribution is *STSAMCTSO<sub>p</sub>Pes*. It appears that the additional games being completed by the EDS agents were below the 10% range which is not sufficient to consider them able to reliably complete those games.

To get a better understanding of these differences, further analysis is provided on the games within the information gain set focusing on specific games where there are differences in the number of wins. In particular the games *Escape*, *Freeway*, *Labyrinthual* and *SisterSaviour*. Table 6.21 shows the number of wins achieved on each of the mentioned games.

Table 6.20: MCTS agents win distributions across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

MCTS Agents Win Dist - Information Gain Games									
Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
sampleMCTS	4	4	4	4	4	4	4	4	4
STSAMCTSO <sub>p</sub> Pes	4	4	4	4	4	4	4	4	2
STSAMCTSO <sub>p</sub> PesSc	4	4	3	2	2	2	2	2	2
STSAMCTSO <sub>p</sub> PesExpSc	5	3	2	2	2	2	2	1	1
STSAMCTSE <sub>xp</sub> Sc	4	4	2	2	2	2	2	1	1



The first thing of note is that *STSAMCTSExpSc* is the only agent able to win runs of *SisterSaviour*. As a deceptive game, *SisterSaviour* is challenging to win, so accruing any wins on this game is an indication that the agent is not being completely fooled by this deception. Also, *STSAMCTSExpSc* is the agent which wins the most number of games out of this subset of games, though it wins zero games of *Freeway*. This is a strange result, as the agents that are able to win it get over 200 wins. This may indicate an error in balancing the Voices as the other ExpSc agent is not able to win any runs either.

The other two games, *Escape* and *Labyrinthdual*, are won to a satisfactory degree by the ExpSc agents, which shows that these perspectives are providing some value to the agents in solving new games.

Table 6.21: A closer look at game performance for the MCTS agents on the information gain set

Agent	escape	freeway	labyrinthdual	sistersaviour
sampleMCTS	0	250	0	0
STSAMCTSExpSc	53	0	52	15
STSAMCTSOpPes	2	219	0	0
STSAMCTSOpPesExpSc	45	0	40	0
STSAMCTSOpPesSc	0	250	0	0

The results of the MCTS agents show that, at the moment, the *sampleMCTS* is the best agent in this experiment. In some circumstances, particularly the deceptive games, it is clear that additional games are being won by the EDS agents though this isn't bringing enough value to all of them. This is likely due to implementation details in how the pessimistic MCTS is implemented, and may require some further modifications in the future.

### 6.3.5 Concurrent EDS

Along with the flexibility of the EDS, the other main advantage of the system is how it lends itself to concurrency: each Voice can be ran in parallel to maximise the amount of thinking time available to the overall system. While other agents can also take

advantage of parallelisation, it will not allow the agent to behave differently. If there are games that an algorithm can't solve, concurrency will not overcome those weaknesses. The EDS, on the other hand, will be able to consider a number of different perspectives and may find a solution to current problem.

In order to test the viability of such a system, some experiments were ran to determine, firstly, if a multi-threaded EDS has better performance than its single threaded equivalent and similar generality, and secondly, if it is able to compete with strong GVGAI agents in the competition. It would not be proper to directly compare the performance of a multi-threaded EDS to single threaded GVGAI agents though, so an important enhancement to the single threaded agents was made. While the multi-threaded EDS has the standard 40ms to perform its analysis, the single threaded agents have  $(40 * N)$ ms to perform their analyses, where  $N$  is the number of Voices in the multi threaded EDS. The intuition behind this alteration is that if an EDS has  $N$  Voices then it is effectively receiving  $40 * N$ ms of CPU time. This is not an entirely acceptable comparison mechanism, as it would be preferable to modify all of the agents into multi-threaded versions, which would be an avenue for future work.

Two different versions of the Concurrent EDS were developed. The first was an All Actions versions, which requires all of the Voices within the system to return an Opinion for all possible Actions. This system is known as the Multi-Threaded All Actions EDS (MTAA). The second, is a Single Action version which only requires a single action from each Voice, but has more options for the final action selection policy. This system is known as the Multi-Threaded Single Action EDS (MTSA).

### 6.3.5.1 Results: Concurrent EDS

First, the MTSA was built with the *STSAOLETSExpSc* configuration as described in section 6.1.3.5. This is the first experiment to show an EDS agent clearly outperforming the base agent across all of the experiment games. The agent named *MTSAOLETSExpSc* shows an immediate improvement when compared to its single threaded equivalent. As can be seen in table 6.22, the number of unique games won has remained unchanged, but the win rate for *MTSAOLETSExpSc* has improved by 2.07%. This increase in win

Table 6.22: Concurrent agents ranked by win percentage across all games.

<b>Concurrent Agents Win Rate - All Games</b>		
<b>Agent</b>	<b>%Wins</b>	<b>#Games</b>
MTSAOLETSExpSc	69.1%	31
OLETS80ms	60.5%	29
OLETS	59.8%	29
STSAOLETSExpSc	59.1%	31
MTAAOLETSExpSc	36%	27

Table 6.23: Concurrent agents win distribution across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

<b>Concurrent Agents Win Dist - All Games</b>									
<b>Agent</b>	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>	<b>70%</b>	<b>80%</b>	<b>90%</b>
MTSAOLETSExpSc	29	27	22	20	18	18	18	14	12
OLETS80ms	26	24	24	23	19	19	16	11	11
OLETS	25	24	24	22	21	19	16	12	10
STSAOLETSExpSc	27	25	23	20	18	18	16	13	10
MTAAOLETSExpSc	20	18	16	11	9	9	9	6	4

rate has actually surpassed the original *OLETS* algorithm as well. As mentioned previously, an agent was given additional analysis time in order to more fairly compare with the concurrent EDSs. This agent is *OLETS80ms* and has been given 80ms of analysis time instead of 40. The performance of *OLETS80ms* shows that overall win rate has improved by 0.71% from the standard *OLETS* agent, and no new games have been solved. While the extra analysis time does lead to improved performance in the games that the *OLETS* agent is already good at, it is not enough to allow the algorithm to complete new games, whilst the EDS algorithm successfully won 2 additional games with and without concurrency.

The second EDS agent, *MTAAOLETSExpSc*, was included to see if there was a significant performance difference between the All Action and Single Action arbiters. As can be seen in table 6.22 there certainly is a difference in their performance, with the *MTAAOLETSExpSc* scoring a win rate of 36.04% and winning only 27 unique games, compared to 61.9% and 31 unique games. This is a much more dramatic difference than expected, and as mentioned earlier may be due to an improper balancing in the values returned by the Voices, leading to one dominating the performance of the agent rather than a blending of the two Voices.

Further analysis has been conducted on the distribution of wins that occurred across all of the games. This is shown in table 6.23. The first thing of note is that the *MTSAOLETSExpSc* agent has a fairly steep drop in wins between the 20-40% mark but stabilises with 18 games before reaching 90% with 12 games. While this drop is quite large, it remains ahead of the win distribution for the *OLETS80ms* agent. The *MTAAOLETSExpSc* performs quite poorly in the win distribution table, finishing with just four games in the 90% column.

To delve deeper into the results, some additional analysis was also done for the deceptive games, which is shown in tables 6.24 and 6.25. These tables tell show the same trend as the all games set, though the *OLETS* and *OLETS80ms* have dropped to the bottom of the table. The EDS agents seem to have a clear lead when faced with the deceptive games set, and this is a trend that is noticed throughout the experiments.

Finally, the results for the information gain set of games are shown in tables 6.26

Table 6.24: Concurrent agents ranked by win percentage across the deceptive games.

<b>Concurrent Agents Win Rate - Deceptive Games</b>		
<b>Agent</b>	<b>%Wins</b>	<b>#Games</b>
MTSAOLETSExpSc	79.4%	8
STSAOLETSExpSc	78.3%	8
MTAAOLETSExpSc	72.4%	8
OLETS	69.8%	7
OLETS80ms	68.5%	7

Table 6.25: Concurrent agents win distribution across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

<b>Concurrent Agents Win Dist - Deceptive Games Games</b>									
<b>Agent</b>	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>	<b>70%</b>	<b>80%</b>	<b>90%</b>
MTSAOLETSExpSc	8	8	7	6	6	6	6	6	5
STSAOLETSExpSc	7	7	7	7	6	6	6	6	5
MTAAOLETSExpSc	8	8	7	6	5	5	5	5	3
OLETS	6	6	6	6	6	6	6	5	4
OLETS80ms	6	6	6	6	6	6	6	4	4

and 6.23. This shows that the *MTAAOLETSExpSc* agent really struggles to perform well on these games, and seems to indicate that the all actions arbiter is not performing as expected. The best agent, *MTSAOLETSExpSc* performs well in both overall win rates and in the win distribution table with the wins that it earns coming from a wider variety of games, and also improving on the performance of both *OLETS* and *OLETS80ms*. Interestingly, the win distributions for *OLETS* and *OLETS80ms* are almost identical, though it appears that *OLETS* is very slightly better, as the wins start to drop off at 60% as opposed to 50%. It appears that while the extra analysis time given to *OLETS80ms* is improving its performance, it is doing so only on a small number of games.

Table 6.26: Concurrent agents ranked by win percentage across the information gain games.

<b>Concurrent Agents Win Rate - Information Gain Games</b>		
<b>Agent</b>	<b>%Wins</b>	<b>#Games</b>
MTSAOLETSExpSc	61.2%	9
OLETS80ms	59.8%	9
OLETS	59%	9
STSAOLETSExpSc	58.3%	9
MTAAOLETSExpSc	17.2%	6

Table 6.27: Concurrent agents win distribution across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

<b>Concurrent Agents Win Dist - Information Gain Games</b>									
<b>Agent</b>	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>	<b>70%</b>	<b>80%</b>	<b>90%</b>
MTSAOLETSExpSc	8	8	7	6	6	6	6	4	4
OLETS80ms	7	7	7	7	6	6	5	4	4
OLETS	7	7	7	7	7	6	5	4	4
STSAOLETSExpSc	8	8	8	7	6	6	4	3	3
MTAAOLETSExpSc	3	3	2	1	1	1	1	1	1

### 6.3.6 Yolobot Voice

As one of the most successful entrants to the GVGAI competition, *YOLOBOT* is a natural selection for inclusion into an EDS. With this in mind, a version of *YOLOBOT* was converted into a Voice and put into an EDS. The main question then was: which other Voices would best work together with *YOLOBOT*?

From the experiments so far, it was clear that *YOLOBOT* was not capable of winning every game in the experiment suite, *Lemmings* and *SisterSaviour* being the two games that *YOLOBOT* did not receive any wins in. Those games were not won by any single algorithm agent so far, though the Ensemble agent *OLETSExpSc* has been quite successful at those games. This led to two variations of EDS being built, which make use of the *OLETSExpSc* configuration and the *YOLOBOT* Voice. The first was to pair *YOLOBOT* with the *OLETSExpSc* algorithm using the exploration maximisation heuristic, and the second was a larger three Voice EDS using *OLETSExpSc* and *YOLOBOT*. Each of these experiments would be conducted with the single action arbiter, as converting *YOLOBOT* to return Opinions for every action was not viable. Experiments were conducted with only a single-threaded version, though investigating a multi-threaded version would be a useful avenue for future work.

#### 6.3.6.1 Results: Yolobot Voice

Table 6.28: Yolobot agents ranked by win percentage across all games.

Yolobot Agents Win Rate - All Games		
Agent	%Wins	#Games
YOLOBOT	74.3%	30
STSAYoloOLETSExp	57.0%	31
STSAYoloOLETSExpSc	54.4%	29

Table 6.28 shows the win rates and unique games won of the Yolobot agents across all games. *Yolobot* is the winning agent across all of the games, with a win rate of 74.3% which is 17.3% difference to the highest EDS agent in this experiment. This difference in performance is likely due to the dilution of the *Yolobot* voice in the EDS

## Chapter 6. Experimental Setup

agents. While the additional perspective is allowing *STSAYoloOLETSExp* to complete an additional game, it is not enough to offset the lost performance.

Table 6.29 shows the win distributions of the agents. It is similarly clear from this perspective that the *Yolobot* agent is performing better than the EDS agents across all of the games. It is a bit clearer though that there is a difference between the two EDS agents and that the three voice EDS, *STSAYoloOLETSExpSc* is seeing worse performance, which is likely due to the further dilution of the Voices.

Table 6.29: Yolobot agents win distribution across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

Yolobot Agents Win Dist - All Games									
Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
YOLOBOT	30	30	29	28	24	22	21	19	15
STSAYoloOLETSExp	30	26	23	18	16	16	14	13	10
STSAYoloOLETSExpSc	26	24	21	19	17	16	15	12	7

Table 6.30: Yolobot agents ranked by win percentage across the deceptive games.

Yolobot Agents Win Rate - Deceptive Games		
Agent	%Wins	#Games
STSAYoloOLETSExp	71.6%	8
STSAYoloOLETSExpSc	67.3%	8
YOLOBOT	66.6%	6

Table 6.30 shows the win rates and unique games won of the Yolobot agents across the deceptive games set. In this case the *Yolobot* agent has dropped to last place compared to the EDS agents in both win rate and unique games won. This shows that the *Yolobot* agent may be more susceptible to some of the deceptions, and that the additional perspectives of the EDS may be making it more resilient.

The specific games that the EDS agents are able to win over *Yolobot* are *Lemmings* and *SisterSaviour*.

The win distributions of the Yolobot agents are shown in table 6.31. While *Yolobot*



does have the smoothest performance across the brackets, it does not begin as highly as either of the EDS agents, and ends in the same place with four games completed. This similarly suggests that *Yolobot* is receiving the majority of its wins from fewer games in the deceptive games set.

Table 6.31: Yolobot agents win distribution across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

Yolobot Agents Win Dist - Deceptive Games									
Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
STSAYoloOLETSExp	8	8	6	6	5	5	5	5	4
STSAYoloOLETSExpSc	8	7	5	5	5	5	5	5	4
YOLOBOT	6	6	6	6	6	5	5	5	4

Table 6.32: Yolobot agents ranked by win percentage across the information gain games.

Yolobot Agents Win Rate - Information Gain Games		
Agent	%Wins	#Games
YOLOBOT	70.2%	9
STSAYoloOLETSExp	45.0%	9
STSAYoloOLETSExpSc	42.1%	7

Table 6.32 shows the win rates of the Yolobot agents across the information gain set of games. *Yolobot* returns to the top position in this set with a win rate of 70.2% and getting at least one win in nine of the ten games. The closest EDS agent is 25.2% behind *Yolobot* which is a large drop in performance, for the same number of unique games won. As mentioned previously, this is due to the splitting of time between the Voices and would likely see some improvement in a concurrent environment.

The win distributions for the information gain set of games are shown in table 6.33. It is clear with these results that *Yolobot* is performing very well across the majority of the set of games.

The main conclusion from these experiments is that while *Yolobot* performs well across the all games and information gain set, it can be improved when dealing with

Table 6.33: Yolobot agents win distribution across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

Yolobot Agents Win Dist - Information Gain Games									
Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
YOLOBOT	9	9	9	8	7	6	6	5	5
STSAyoloOLETSExp	8	7	6	4	4	4	3	3	2
STSAyoloOLETSExpSc	6	6	5	5	5	4	4	3	1

deceptive games. The main limitation of the EDS agents in this case is the 40ms decision time which means that they cannot use the Voices to their full potential.

Future work in this area would focus on turning the *Yolobot* agent into an EDS in its own right, to get a more direct comparison of performance. Similarly, creating a version of the Yolobot EDS agents that makes use of concurrency would be of interest.

### 6.3.7 Overall Results

To get a better understanding of the performance of each experiment, further analysis was performed to compare each experiment to each other, as well as to explore individual game performance. This section first looks at the overall win rates for all agents on all of the games, looking at trends in performance across the games. Finally, games with unusual or interesting results are explored to give an explanation for the behaviour seen in the agents with both raw data and visual analysis.

#### 6.3.7.1 Overall Raw Results

The overall results are broken up similarly to the other experiments: each set is presented separately, with win rate and win distribution tables. As the majority of agents have already been discussed, this section focused on the agents that have not yet been discussed, such as *YBCriber* and *Return42*.

Table 6.34 shows the win rates and number of unique games won for all agents, ordered by win rate. The first thing to note is that the highest win rate agent *YBCriber* is not the agent that wins the highest number of games, winning 29 out of 32 games

Table 6.34: All agents ranked by win percentage across all games.

<b>Rank</b>	<b>Agent</b>	<b>%Wins</b>	<b>#Games</b>
1	YBCriber	78.5%	29
2	YOLOBOT	74.3%	30
3	adrienctx	69.1%	29
4	Return42	62.6%	32
5	MTSAOLETSExpSc	61.9%	31
6	OLETS80ms	60.5%	29
7	OLETS	59.8%	29
8	STSAOLETSExpSc	59.1%	31
9	STSAYoloOLETSExp	57.0%	31
10	STSAYoloOLETSExpSc	54.4%	29
11	sampleMCTS	46.1%	24
12	STSAMCTSOpPes	44.9%	26
13	sampleRS	43.7%	25
14	sampleRHEA	40.9%	25
15	STSAMCTSExpSc	37.3%	28
16	STSAMCTSOpPesExpSc	36.8%	26
17	MTAAOLETSExpSc	36.0%	27
18	STAAOLETSExpSc	35.7%	26
19	STSAMCTSOpPesSc	33.5%	18
20	sampleonesteplookahead	29.6%	17
21	sampleRandom	15.1%	18

## Chapter 6. Experimental Setup

of the experiment set. This is likely due to the algorithm that *YBCriber* uses being a less general algorithm, but does extremely well at the games that it is able to win. *Return42* is the only agent that has been able to win at least once across all games of the experiment set, with a win rate of 62.55%. The highest EDS agent appears in fifth place with a win rate of 61.9% and 31 games. It should be noted that this is a multi-threaded agent and so this is not an entirely fair comparison with most of the other agents.

Table 6.35: All agents win distribution across all games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
YBCriber	28	28	28	28	27	25	23	22	20
YOLOBOT	30	30	29	28	24	22	21	19	15
adrienctx	27	27	26	23	22	21	20	19	16
Return42	28	26	26	24	21	18	16	14	11
MTSAOLETSExpSc	29	27	22	20	18	18	18	14	12
OLETS80ms	26	24	24	23	19	19	16	11	11
OLETS	25	24	24	22	21	19	16	12	10
STSAOLETSExpSc	27	25	23	20	18	18	16	13	10
STSAYoloOLETSExp	30	26	23	18	16	16	14	13	10
STSAYoloOLETSExpSc	26	24	21	19	17	16	15	12	7
sampleMCTS	17	17	16	16	15	15	14	13	10
STSAMCTSOpPes	19	17	16	16	15	13	13	12	8
sampleRS	19	19	18	16	14	13	10	9	8
sampleRHEA	19	19	16	15	14	12	9	8	7
STSAMCTSExpSc	21	20	14	12	11	10	8	6	6
STSAMCTSOpPesExpSc	20	17	13	12	11	10	9	7	5
MTAAOLETSExpSc	20	18	16	11	9	9	9	6	4
STAAOLETSExpSc	20	17	16	11	11	9	8	7	4
STSAMCTSOpPesSc	18	17	11	10	10	9	9	8	6
sampleonesteplookahead	13	13	11	11	11	10	7	6	4
sampleRandom	9	8	6	5	4	3	2	2	2

The win distributions for all agents across all games is shown in table 6.35. *YBCriber* has a smooth distribution dropping only eight games through all of the brackets, and finishing on 20. *Return42* also begins with 28 games but has a much sharper decline

in performance. This indicates that *YBCriber* is retaining a high level of performance across a wider variety of games.

A heatmap showing the win rates across each game is shown in figure 6.6. The heatmap clearly shows which games are being won by which agents, with games such as *Lemmings*, *Sistersaviour*, *Watergame* and *Tercio* proving to be challenging for particular groups of agents. In particular, it is worth noting the differences between *Yolobot* and *MTSAOLETSExpSc* where the EDS agent has a wider distribution of wins, while *Yolobot* has a number of darker cells where its performance is concentrated.

Table 6.36: All agents ranked by win percentage across the deceptive games.

Rank	Agent	%Wins	#Games
1	MTSAOLETSExpSc	79.4%	8
2	STSAOLETSExpSc	78.3%	8
3	STAAOLETSExpSc	73.5%	8
4	adrienctx	73.6%	6
5	MTAAOLETSExpSc	72.4%	8
6	STSAYoloOLETSExp	71.6%	8
7	OLETS	69.8%	7
8	YBCriber	69.4%	6
9	sampleMCTS	68.7%	6
10	OLETS80ms	68.5%	7
11	STSAMCTSOpPes	68.1%	6
12	STSAYoloOLETSExpSc	67.3%	8
13	Return42	67.2%	8
14	YOLOBOT	66.6%	6
15	STSAMCTSExpSc	64.6%	8
16	sampleRHEA	61.1%	6
17	sampleRS	60.4%	6
18	STSAMCTSOpPesExpSc	57.2%	6
19	sampleonesteplookahead	52.4%	6
20	STSAMCTSOpPesSc	51.7%	6
21	sampleRandom	33.4%	5

The win rates for all agents across the deceptive games are shown in table 6.36. This set of games shows a very different distribution of the agents, with the EDS agents featuring in the top positions, while strong agents such as *Yolobot* drop down. The top three positions are each taken by EDS agents, while *Yolobot* has dropped

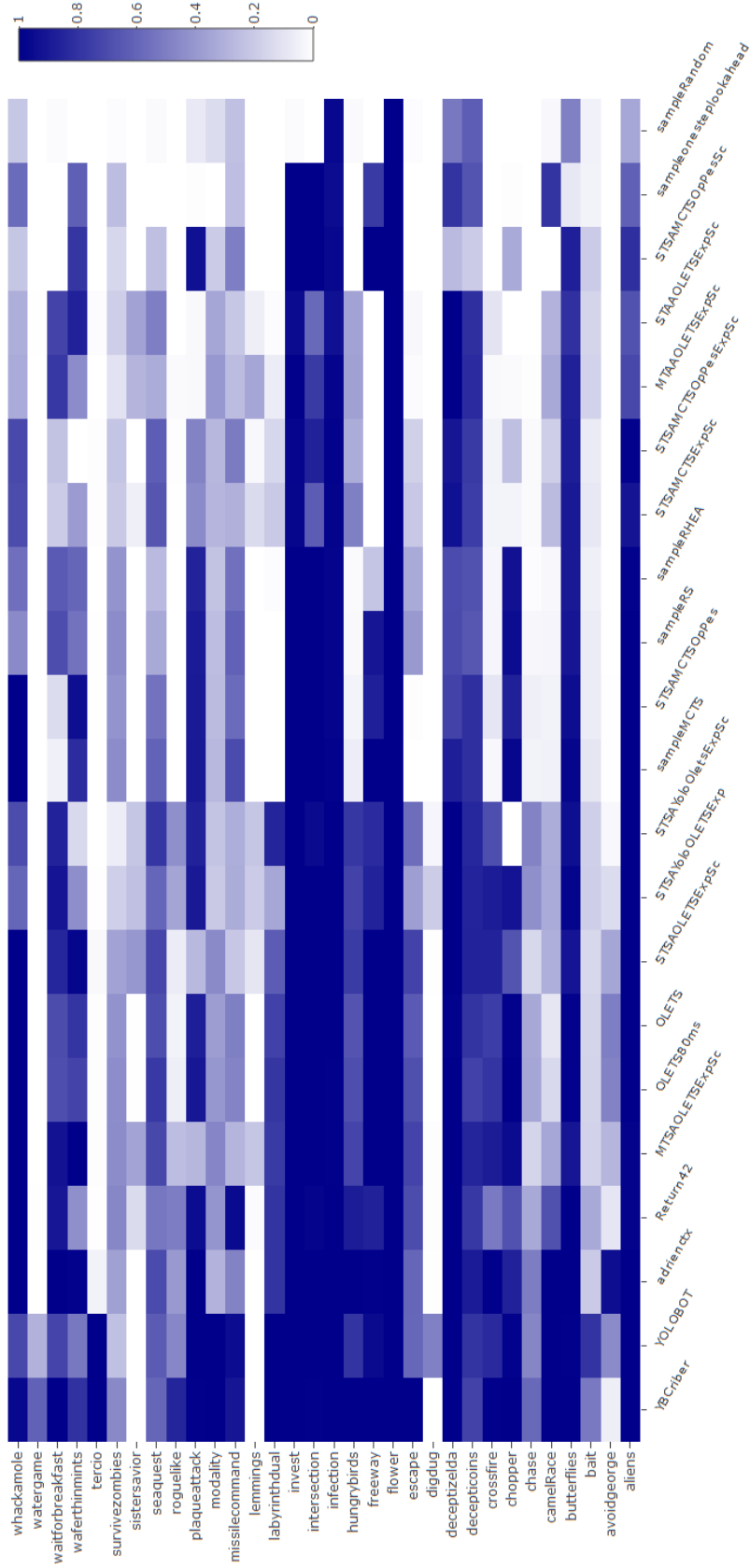


Figure 6.6: Total percentage of wins (0 – 100%) of each controller by game across all games.

## Chapter 6. Experimental Setup

to 14th place. *Adrienctx* is the only non EDS agent to retain a spot in the top five positions, though it is only able to win six of the eight deceptive games.

These results show that the EDS agents in general seem to be more resilient to deception, though only for certain combinations. The agents *STSAMCTSO<sub>p</sub>PesExpSc* and *STSAMCTSO<sub>p</sub>PesSc* rank in 18th and 20th position. This shows that the arbitrary combination of Voices does not always lead to improved performance in this set of games.

Table 6.37: All agents win distribution across the deceptive games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

Agent	10%	20%	30%	40%	50%	60%	70%	80%	90%
MTSAOLETSExpSc	8	8	7	6	6	6	6	6	5
STSAOLETSExpSc	7	7	7	7	6	6	6	6	5
STAAOLETSExpSc	7	7	7	6	6	6	6	6	3
adrienctx	6	6	6	6	6	6	6	6	5
MTAAOLETSExpSc	8	8	7	6	5	5	5	5	3
STSA <sub>Yolo</sub> OLETSExp	8	8	6	6	5	5	5	5	4
OLETS	6	6	6	6	6	6	6	5	4
YBCriber	6	6	6	6	6	6	6	5	4
sampleMCTS	6	6	6	6	6	6	6	6	3
OLETS80ms	6	6	6	6	6	6	6	4	4
STSAMCTSO <sub>p</sub> Pes	6	6	6	6	6	6	6	5	4
STSA <sub>Yolo</sub> OLETSExpSc	8	7	5	5	5	5	5	5	4
Return42	7	6	6	6	5	5	5	4	4
YOLOBOT	6	6	6	6	6	5	5	5	4
STSAMCTSExpSc	7	6	6	5	5	5	5	4	4
sampleRHEA	6	6	6	6	6	6	4	3	3
sampleRS	6	6	6	6	6	5	4	3	3
STSAMCTSO <sub>p</sub> PesExpSc	5	5	5	5	5	5	5	4	3
sampleonesteplookahead	5	5	5	5	5	5	3	3	2
STSAMCTSO <sub>p</sub> PesSc	6	6	4	4	4	4	4	3	2
sampleRandom	4	4	4	4	3	2	1	1	1

The win distributions of all agents across the deceptive games are shown in table 6.37. The top two agents have a very similar win distribution with the *MTSAOLETSExpSc* agent managing to win eight games in the early brackets and finishing on five.

## Chapter 6. Experimental Setup

*Adrienctx* has a very smooth distribution though it does not begin as highly as the top EDS agents. This shows that while *Adrienctx* is able to keep up with the EDS agents in terms of win rate, it is doing so over fewer games. This can be seen further in the heatmap shown in figure 6.7 which shows that *Adrienctx* is not able to win at either *Sistersaviour* or *Lemmings*.

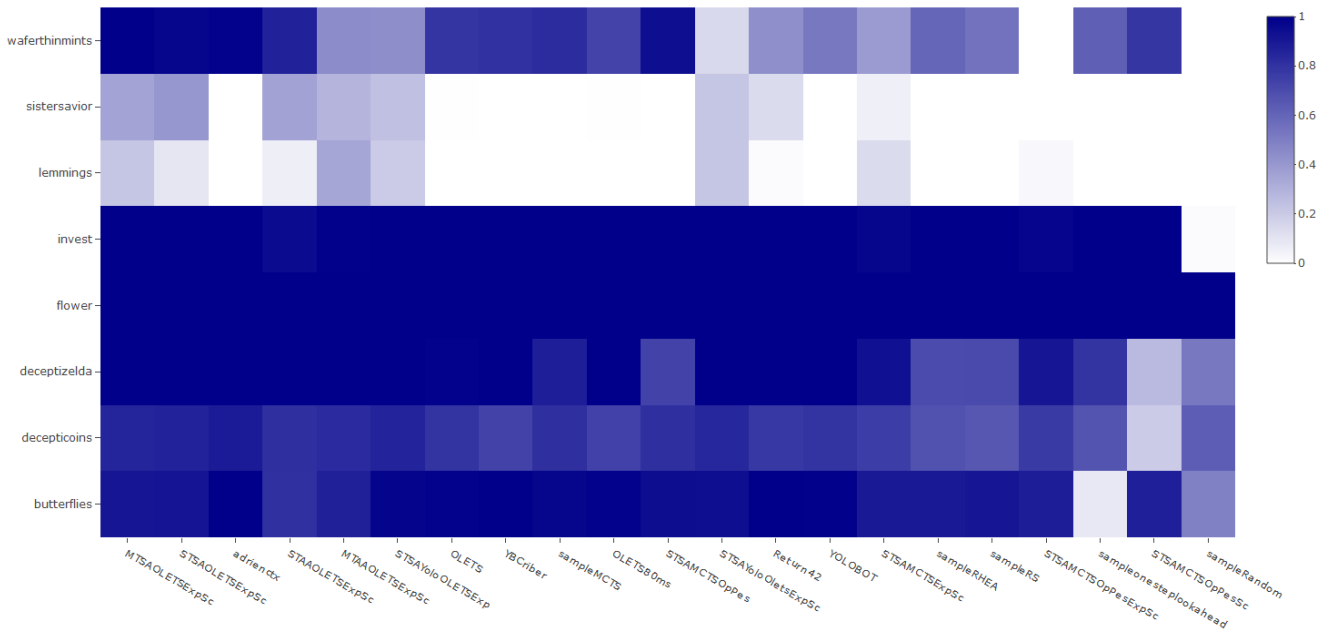


Figure 6.7: Total percentage of wins (0 – 100%) of each controller by game across the deceptive games.

The win rates for all agents on the information gain set of games are shown in table 6.38. *YBCriber* has returned to the top spot of this set, while *Return42* has dropped to 8th position compared to the all games results. The AA and MCTS EDS agents have dropped to low positions in this set, while *MTSAOLETSExpSc*, *STSAOLETSExpSc* and the Yolobot EDS variations remain in the top ten.

The win distributions for all agents on the information gain set of games are shown in table 6.39. *YBCriber* has a great distribution of wins in this set, and clearly leads the way. While *Yolobot* is able to initially win more games, this soon drops down as the performance brackets go up. A closer look at exactly which games are being won is shown in the heatmap in figure 6.8.



Table 6.38: All agents ranked by win percentage across the information gain games.

Rank	Agent	%Wins	#Games
1	YBCriber	76.5%	9
2	YOLOBOT	70.2%	9
3	adrienctx	62.6%	8
4	MTSAOLETSExpSc	61.2%	9
5	OLETS80ms	59.8%	9
6	OLETS	59.0%	9
7	STSAOLETSExpSc	58.3%	9
8	Return42	53.0%	10
9	STSAYoloOLETSExp	45.0%	9
10	STSAYoloOLETSExpSc	42.1%	7
11	sampleMCTS	39.8%	5
12	STSAMCTSOpPes	37.6%	7
13	sampleRS	37.2%	6
14	sampleRHEA	30.7%	6
15	STSAMCTSOpPesSc	25.6%	4
16	sampleonestepllookahead	23.6%	4
17	STSAMCTSOpPesExpSc	22.9%	6
18	STSAMCTSExpSc	22.1%	7
19	MTAAOLETSExpSc	17.2%	6
20	STAAOLETSExpSc	16.9%	7
21	sampleRandom	2.5%	3

Table 6.39: All agents win distribution across the information gain games. Each column represents the number of games where the corresponding agent was able to win at least that percentage of the total runs.

<b>Agent</b>	<b>10%</b>	<b>20%</b>	<b>30%</b>	<b>40%</b>	<b>50%</b>	<b>60%</b>	<b>70%</b>	<b>80%</b>	<b>90%</b>
YBCriber	8	8	8	8	8	8	7	7	7
YOLOBOT	9	9	9	8	7	6	6	5	5
adrienctx	7	7	7	7	7	7	6	6	4
MTSAOLETSExpSc	8	8	7	6	6	6	6	4	4
OLETS80ms	7	7	7	7	6	6	5	4	4
OLETS	7	7	7	7	7	6	5	4	4
STSAOLETSExpSc	8	8	8	7	6	6	4	3	3
Return42	8	6	6	6	6	6	5	4	2
STSAYoloOLETSExp	8	7	6	4	4	4	3	3	2
STSAYoloOLETSExpSc	6	6	5	5	5	4	4	3	1
sampleMCTS	4	4	4	4	4	4	4	4	4
STSAMCTSOpPes	4	4	4	4	4	4	4	4	2
sampleRS	5	5	5	4	3	3	3	3	3
sampleRHEA	5	5	4	3	3	2	2	2	2
STSAMCTSOpPesSc	4	4	3	2	2	2	2	2	2
sampleonesteplookahead	3	3	3	3	3	2	2	1	1
STSAMCTSOpPesExpSc	5	3	2	2	2	2	2	1	1
STSAMCTSExpSc	4	4	2	2	2	2	2	1	1
MTAAOLETSExpSc	3	3	2	1	1	1	1	1	1
STAAOLETSExpSc	3	3	3	1	1	1	1	1	1
sampleRandom	1	1	0	0	0	0	0	0	0

## Chapter 6. Experimental Setup

Interestingly, there are differences between the top performing agents, particularly among the games that they are able to win. *Watergame* and *Tercio* are only won by *YBCriber* and *Yolobot* while *Sistersaviour* is won almost exclusively by EDS agents, with the exception of *Return42*.



Figure 6.8: Total percentage of wins (0 – 100%) of each controller by game across the information gain games.

The results here give a good idea of the performance trends across the agents, but this does not capture all aspects of performance. In order to look deeper into the results, an analysis on specific games is carried out in the next section.

### 6.3.7.2 Game Analysis

The raw results do not entirely explain the full picture of the experiments and the deceptive games, in particular, need additional examination. Many of the deceptive games use win states as a deceptive feature to pull agents away from optimal play, so it is necessary to look at both the win rate and scores on those games.

The first deceptive game to look at, *Decepticoins*, offers agents an easy win, by sacrificing a higher point total. An agent has managed to surpass the deception if it

has been able to score over 2 in individual games, so a higher average score is desired across all of the levels. Table 6.40 shows the normalised win rate and scores achieved from all agents on *Decepticoins*. The first thing of note is that *Yolobot* scores the highest average score across all of the agents, though its win rate is lower than many other agents. This indicates that *Yolobot* is not being deceived by the deception, though it is not always finishing the levels. Whilst the EDS agents do get good win rates, their scores are a bit lower, though over 2 which indicates that they are being fooled by the deceptions some of the time.

Table 6.40: The normalised win rate and scores for all agents on Decepticoins

Agent	Norm Win	Norm Score
YOLOBOT	0.8	1
YBCriber	0.74	0.936571
adrienctx	0.892	0.72448
Return42	0.78	0.519326
sampleRS	0.66	0.454906
OLETS	0.8	0.441031
sampleRHEA	0.68	0.423191
OLETS80ms	0.744	0.41328
STSAMCTSOpPes	0.816	0.303271
sampleMCTS	0.816	0.278494
STSAMCTSOpPesExpSc	0.772	0.264618
STSAMCTSExpSc	0.76	0.232904
sampleonesteplookahead	0.676	0.221011
MTSAOLETSExpSc	0.856	0.215064
STSAYoloOLETSExp	0.86	0.198216
MTSAOLETSExpSc	0.864	0.196234
MTAAOLETSExpSc	0.828	0.159564
STSAYoloOLETSExpSc	0.844	0.151635
STAAOLETSExpSc	0.816	0.139742
STSAMCTSOpPesSc	0.2	0.093162
sampleRandom	0.632	0

The next game to look at is *Lemmings* as this game requires the agents to spend their score in order to win. Figure 6.9 shows the win rates and scores for the agents on *Lemmings*. *MTAAOLETSExpSc* is the overall winner of this game with the highest win rate, with the majority of the EDS agents managing to win some games, and the only

non EDS agent winning being *Return42*. The most noticeable thing about the results is that lower scores seems to correlate with a higher win rate. This is likely because *Lemmings* rewards the player for losing score, which most agents are not yet able to consider. Interestingly, *OLETS80ms* also does not manage to secure any wins for this game, showing that simply increasing the amount of analysis time is not enough to score a win. It is necessary to analyse the situation differently, as *Return42* and the EDS agents seem to do.

*Flower* has an unusual result, as there is no loss condition for the game. All of the agents were able to achieve a 100% win rate. The differentiation then is in the score that the agents were able to achieve. Table 6.41 shows the average score achieved by the agents. The first observation is that the portfolio agents score quite poorly at this game, with the highest coming in 12th. The EDS agents do particularly well in this game, and show significant improvement over their base agents. This seems to be due to the exploration Voice, which assigns value to unexplored states, meaning that the agent is less likely to collect the small reward earned from the early flower growths.

*Invest* is another game that is best interpreted through looking at the scores achieved by each agent as most agents achieve high win rates. Getting an average score higher than 5 indicates that the agent has not fallen for the deception at least once. Figure 6.10 shows all of the agents win rates and average scores. The best, again, is *Return42* which is able to achieve both a high win rate and score, with the other EDS agents not far behind. Interestingly, both *OLETS* and *OLETS80ms* do not manage to attain an average score above 5, but the EDS systems that make use of them do manage to. This is likely due to the additional perspectives from the EDS Voices giving additional information on which action to take.

The next game to investigate is *WaferThinMints*. It was expected, when designing the game, that search agents would generally have few problems with it, as the deceptive element is thought to effect machine learning agents. This turned out to be slightly incorrect, as the deception does manage to separate the performance of search agents too. Table 6.42 shows the win rates for all agents playing *WaferThinMints*. Looking at the average scores for this game is unnecessary, as winning the game is an indication



Table 6.41: Flower rankings by average score. The real values and normalised values are shown.

Rank	Agent	Average Score	Norm Average Score
1	MTAAOLETSExpSc	497.764	0.765898477
2	sampleRS	459.472	0.66871066
3	STAAOLETSExpSc	457.872	0.664649746
4	sampleRHEA	439.5	0.618020305
5	STSAMCTSOpPesExpSc	422.588	0.575096447
6	STSAMCTSExpSc	405.264	0.531126904
7	sampleMCTS	385.928	0.482050761
8	sampleonesteplookahead	371.8	0.446192893
9	STSAMCTSOpPesSc	369.696	0.440852792
10	STSAMCTSOpPes	364.632	0.428
11	adrienctx	340.744	0.367370558
12	sampleRandom	332.284	0.345898477
13	Return42	327.256	0.333137056
14	MTSAOLETSExpSc	312.42	0.295482234
15	STSAOLETSExpSc	301.036	0.266588832
16	YBCriber	300.24	0.264568528
17	OLETS80ms	291.968	0.243573604
18	OLETS	290.116	0.238873096
19	STSAyoloOLETSExpSc	263.204	0.170568528
20	YOLOBOT	254.292	0.147949239
21	STSAyoloOLETSExp	252.496	0.143390863

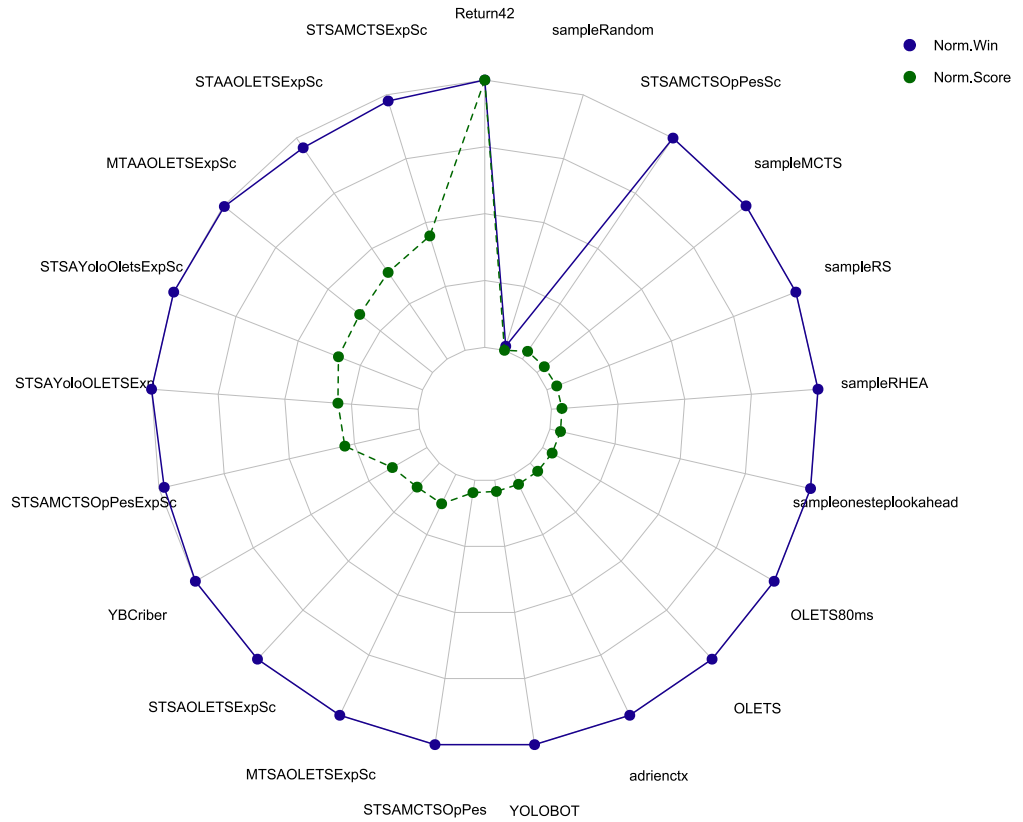


Figure 6.10: *Invest* stats per agent. The percentage of wins is in range  $[0, 100\%]$  and the average of score is relative to the maximum and minimum scores achieved in the game (range:  $[-7, 208]$ ). Values have been normalised in this figure.



that the agent has not fallen for the deception. *YOLOBOT* and *Return42* manage to achieve relatively low win rate on this game with 52.8% and 43.6% respectively. The top performing agent for this game, *MTSAOLETSExpSc*, manages to achieve a 100% win rate, showing that it is possible to defeat this particular deception consistently, along with four other agents managing to achieve win rates over 90%.

Table 6.42: *WaferThinMints* win rates per agent.

Rank	Agent	Win Rate
1	MTSAOLETSExpSc	100.0%
2	adrienctx	99.2%
3	STSAOLETSExpSc	97.6%
4	STSAMCTSOpPes	94.4%
5	STAAOLETSExpSc	86.8%
6	sampleMCTS	82.4%
7	YBCriber	80.8%
8	OLETS	79.6%
9	STSAMCTSOpPesSc	78.8%
10	OLETS80ms	73.6%
11	sampleonesteplookahead	62.4%
12	sampleRHEA	60.0%
13	sampleRS	54.8%
14	YOLOBOT	52.8%
15	MTAAOLETSExpSc	44.8%
16	STSAYoloOLETSExp	44.0%
17	Return42	43.6%
18	STSAMCTSExpSc	39.2%
19	STSAYoloOLETSExpSc	14.8%
20	STSAMCTSOpPesExpSc	0.0%
21	sampleRandom	0.0%

## 6.4 Summary

The results show that while EDS agents perform particularly well in deceptive games, there is still a lot of work to do in order to catch up with the current portfolio agents.

These are encouraging results that leave open a number of interesting avenues for future work, and they show that an EDS is a viable option for a high performing agent, particularly when concurrency is permitted.

## Chapter 7

# Conclusions and Further Work

This chapter discusses the impact of this work and draws conclusions about how they may affect the future of GVGP agent design. In addition, some discussion on the unexplored ideas and potential expansions that could arise from this work: first, exploring the different potential action selection policies that can be used, then going into different approaches to designing and improving an EDS.

### 7.1 Machine Learning Voice

One of the earlier ideas was to incorporate a more extensive array of different types of Voices within the EDS. The prospect of using a Machine Learning algorithm for one of the Voices is an exciting idea, particularly with the combination of learning and search that led to the success of AlphaGo [26] and AlphaZero [77]. There are many situations that bring further opportunities to explore, from the agent's point of view. These situations are difficult to describe in code, but a Machine Learning Voice could be trained to provide an EDS with an instinct about which actions are most beneficial, so to provide the agent with a way to react to more complex situations such as dead ends. Dead-end avoidance would involve an agent being able to determine if an action, or a path of actions, would lead the agent into a no win situation. As an example, if the agent were being chased by a hostile enemy and turned down a path with no exit, the agent would lose the game and should be able to detect this before making the

decision to travel down the path.

## 7.2 Neural Network Arbiter

Developing the arbiter as a neural network, and allowing it to decide which Voice to listen to dependent on the state, is a similarly exciting avenue. This idea would essentially treat a game state as an input into a neural network, which would have the Voices as output nodes. Alternatively, the analysis of the states could be performed beforehand, and the outputs from the Voices would form part of the input to the arbiter, along with the state. The arbiter would then be able to adjust the weights of the system, and hopefully learn which states were best handled by which Voices, or sets of Voices. This approach would have the advantage of using both ML techniques and search without requiring any changes to the current Voices. Expanding from this, if a weight is applied to the value of each Voice's Opinion, then the arbiter could adjust this weight in order to amplify or dampen a Voice's effect based on their performance so far. If a Voice has been performing poorly in a particular game, its decisions can be taken less seriously so that other Voices have a greater contribution to the decision making process.

## 7.3 Large EDS

Another potential avenue for future research would be to develop a larger EDS with an arbiter that could hold a library of Voices. This arbiter would have a certain number of Voices currently active, which are contributing to the current decision making process, and others that are inactive which do not contribute. The arbiter would be able to swap the active and inactive Voices to adjust the performance of the system. This arbiter would be able to tune its own architecture to the specific game being played. While it may be better to run all of the Voices on every state, time limitations such as the 40ms decision time limit of the GVGAI means that this could be a potential compromise between a large library of Voices and giving each Voice another individual computation time.

## 7.4 Bandit Arbiter

Bandit style selection policies, similar to the MCTS UCT algorithm, are another interesting area of research within an EDS. Voices could be given more or less computation time based on their UCB value, or an action could be chosen based on a bandit action selection policy. The Voices themselves do not know which situations they are analysing, nor which they are effective at analysing. A bandit process would allow the arbiter to evaluate the effectiveness of each Voice based on their prior experience in the game, and how often they have found success.

## 7.5 Further Concurrency Experiments

The GVGAI currently does not support multi-threading within the official competition, and an EDS would naturally benefit enormously from this due to its architecture. Each Voice can process in parallel, which in theory should show a significant improvement over its current performance, as it should retain the generality of the EDS while minimising any performance loss. It is not entirely clear whether other algorithms could as easily take advantage of such a change to the competition, and indeed adding additional Voices to the EDS would in theory only improve the generality of the system. The experiments that were done to show the viability of a concurrent EDS were a small proof of concept that such a system could function, and be developed easily. Further work could be done to compare more definitively the EDS to other algorithms that have been fully expanded to take advantage of a multi-threaded environment.

## 7.6 Voice Decomposition

The different methods of decomposing the Voices could be more thoroughly explored: in particular the temporal decomposition. Perspective decomposition was the most explored decomposition in this work, but there may be possibilities for further types of decomposition. The temporal decomposition would use each Voice with a specific time frame in mind, such as short or long range. The prototype EDS made use of such a

system, and shows promise for further exploration.

## 7.7 Portfolio EDS

Agents that use a portfolio system would also be an interesting agent to adapt into an EDS. Seeing as they are currently a collection of algorithms, altering their structure, from selecting the algorithm based on game analysis to requiring each algorithm to be a separate Voice, would be an easy step forward and would allow a clear example of the performance differences between a portfolio and EDS agent. This would potentially increase the generality of these systems, as well as allowing a greater degree of flexibility in how the components of the portfolio system work together.

## 7.8 Conclusions

This thesis has analysed the performance characteristics of a selection of Ensemble Decision Systems in the area of GVGP, using the GVGAI competition as an experiment platform. The results show that thoughtful implementation of an EDS can improve the generality of a GVGP agent. If concurrency is included then the EDS has shown signs that the combination of algorithms may actually strengthen the base algorithms, though further experimentation is required to determine this.

This thesis was done through the following steps: first, an appropriate set of games with which to test the EDS had to be selected from the GVGAI library. During this exploration of the current GVGAI library it was determined that there were a class of games, deceptive games, which would be an important aspect for competent GVGP agents to be able to complete. This involved the creation of a new suite of games which had the desired deceptive qualities, and the development of the continuous information gain algorithm which would select the most diverse set of games from the GVGAI library. Secondly, an exploration of the different types of EDS agents that could be developed was conducted, which led to the creation of several distinct systems for experimentation. Third, a direct comparison of different EDS agents was done with successful portfolio and appropriate benchmark agents from the GVGAI, to get

a better idea if an EDS is a viable competition agent. Finally, a set of experiments which brought together all of the previous work and delved deeper into the encouraging results observed in previous experiments were done.

The EDS shows the potential to be a powerful GVGP agent. Successful portfolio agents do see improved performance in comparison to the EDS agents so far, but it has been shown in this thesis that even those portfolio agents see improved generality when implemented in an EDS. As concurrency is a natural next step for an EDS agent, if it were to be allowed within the GVGAI competition, the options available for an EDS would broaden.

The EDS shows a resilience to deceptive games which other agents do not have. If this benefit of the system can be retained whilst improving the performance of the other aspects of the system, likely through concurrency, then the EDS may be a more powerful GVGP agent.

## 7.9 Contributions

The exploration and development of the EDS is the main contribution of this thesis. The GVGAI was used as the platform for experimentation where the EDS was shown to be capable of playing the games in the library successfully, as well as to be capable of improving the generality of base algorithms from the GVGAI.

Along with the EDS, the concept of deceptive games was introduced in terms of the GVGAI and the importance of a GVGP agent being able to solve these types of games was highlighted. The work with deceptive games included the introduction of new games to the GVGAI library and experimentation to determine which agents were affected by these games. Several different types of deception were also identified and described. This work showed not only that agents could be deceived, but that the ranking of agents was partially determined by the set of games selected to evaluate them with.

The continuous information gain formula was also introduced as a way of developing a better understanding of what types of games exist in the GVGAI, and how to rigorously select a set of games that minimises bias. The work done involved the

## Chapter 7. Conclusions and Further Work

implementation and usage of the algorithm on the set of GVGAI games, which itself was developed by Dr Christoph Salge.

The main contribution of this thesis has been the introduction and exploration of the Ensemble Decision System in the GVGAI framework. A number of different EDS systems were implemented and tested for the GVGAI and compared to high performing agents from the competition. While the EDS agents did not outperform the other agents in all areas, they did see a wider spread of wins across a variety of games, and managed to outperform other agents in the deceptive game suite. These encouraging results indicate that the EDS system is capable of performing well, improving generality within the GVGAI, and that further experimentation may be fruitful.

# Bibliography

- [1] B. Goertzel and C. Pennachin, *Artificial general intelligence*. Springer, 2007, vol. 2.
- [2] R. L. Page, “Brief history of flight simulation.”
- [3] J. L. Sabourin and J. C. Lester, “Affect and engagement in game-based learning environments,” *IEEE transactions on affective computing*, vol. 5, no. 1, pp. 45–56, 2013.
- [4] J. E. Driskell and D. J. Dwyer, “Microcomputer videogame based training,” *Educational technology*, vol. 24, no. 2, pp. 11–16, 1984.
- [5] K. D. Squire, “Video game-based learning: An emerging paradigm for instruction,” *Performance Improvement Quarterly*, vol. 21, no. 2, pp. 7–36, 2008.
- [6] M. Mateas, “Expressive AI: Games and artificial intelligence.” in *DiGRA Conference*, 2003.
- [7] J. Orkin, “Three states and a plan: the AI of fear,” in *Game Developers Conference*, vol. 2006, 2006, p. 4.
- [8] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.
- [9] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.



## Bibliography

- [10] P. Rohlfshagen, J. Liu, D. Perez-Liebana, and S. M. Lucas, “Pac-man conquers academia: Two decades of research using a classic arcade game,” *IEEE Transactions on Games*, vol. 10, no. 3, pp. 233–256, 2017.
- [11] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.
- [12] E. Piette, D. J. Soemers, M. Stephenson, C. F. Sironi, M. H. Winands, and C. Browne, “Ludii-the ludemic general game system,” *arXiv preprint arXiv:1905.05013*, 2019.
- [13] J. Levine, C. Bates Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, “General Video Game Playing,” *IEEE Transactions on Computational Intelligence and AI in Games*, 2013.
- [14] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, “Towards a Video Game Description Language,” *IEEE Transactions on Computational Intelligence and AI in Games*, 2013.
- [15] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, “The 2014 General Video Game Playing Competition,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.
- [16] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, “General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms,” *arXiv preprint arXiv:1802.10363*, 2018.
- [17] K.-H. Glassmeier, H. Boehnhardt, D. Koschny, E. Kührt, and I. Richter, “The rosetta mission: flying towards the origin of the solar system,” *Space Science Reviews*, vol. 128, no. 1-4, pp. 1–21, 2007.
- [18] B. Martin, “Should videogames be viewed as art,” *Videogames and art*, pp. 201–210, 2007.

## Bibliography

- [19] A. Clarke and G. Mitchell, *Videogames and art*. Intellect books, 2007.
- [20] J. Togelius, “Ai researchers, video games are your friends!” in *International Joint Conference on Computational Intelligence*. Springer, 2015, pp. 3–18.
- [21] D. Anderson, M. Stephenson, J. Togelius, C. Salge, J. Levine, and J. Renz, “Deceptive games,” in *International Conference on the Applications of Evolutionary Computation*. Springer, 2018, pp. 376–391.
- [22] P. Bontrager, A. Khalifa, D. Anderson, M. Stephenson, C. Salge, and J. Togelius, “Superstition in the network: Deep reinforcement learning plays deceptive games,” *arXiv preprint arXiv:1908.04436*, 2019.
- [23] D. Anderson, C. Guerrero-Romero, D. Perez-Liebana, P. Rodgers, and J. Levine, “Ensemble decision systems for general video game playing,” *arXiv preprint arXiv:1905.10792*, 2019.
- [24] M. Stephenson, D. Anderson, A. Khalifa, J. Levine, J. Renz, J. Togelius, and C. Salge, “A continuous information gain measure to find the most discriminatory problems for ai benchmarking,” *arXiv preprint arXiv:1809.02904*, 2018.
- [25] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, “Deep blue,” *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.
- [26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, p. 484, 2016.
- [27] A. M. Turing, “Computing machinery and intelligence,” in *Parsing the Turing Test*. Springer, 2009, pp. 23–65.
- [28] J. Schaeffer, *One jump ahead: challenging human supremacy in checkers*. Springer Science & Business Media, 2013.

## Bibliography

- [29] B. Farley and W. Clark, “Simulation of self-organizing systems by digital computer,” *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 76–84, 1954.
- [30] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber, “Super mario evolution,” in *2009 IEEE Symposium on Computational Intelligence and Games*. IEEE, 2009, pp. 156–161.
- [31] J. Togelius, S. Karakovskiy, and R. Baumgarten, “The 2009 mario ai competition,” in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.
- [32] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, “A survey of real-time strategy game AI research and competition in starcraft,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 4, pp. 293–311, 2013.
- [33] M. Genesereth, N. Love, and B. Pell, “General game playing: Overview of the aaai competition,” *AI magazine*, vol. 26, no. 2, pp. 62–62, 2005.
- [34] N. Asgharbeygi, D. Stracuzzi, and P. Langley, “Relational temporal difference learning,” in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 49–56.
- [35] B. Banerjee and P. Stone, “General game learning using knowledge transfer,” in *The 20th International Joint Conference on Artificial Intelligence*, 2007, pp. 672–677. [Online]. Available: <http://www.cs.utexas.edu/~pstone/Papers/bib2html-links/IJCAI07-bikram.pdf>
- [36] M. Thielscher, “General game playing in AI research and education,” in *Annual Conference on Artificial Intelligence*. Springer, 2011, pp. 26–37.
- [37] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, “General video game level generation,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, 2016, pp. 253–259.

## Bibliography

- [38] A. Khalifa, M. C. Green, D. Perez-Liebana, and J. Togelius, “General video game rule generation,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 170–177.
- [39] R. D. Gaina, D. Pérez-Liébana, and S. M. Lucas, “General video game for 2 players: framework and competition,” in *2016 8th Computer Science and Electronic Engineering (CEECE)*. IEEE, 2016, pp. 186–191.
- [40] R. D. Gaina, A. Couëtoux, D. J. Soemers, M. H. Winands, T. Vodopivec, F. Kirchgeßner, J. Liu, S. M. Lucas, and D. Perez-Liebana, “The 2016 two-player gvgai competition,” *IEEE Transactions on Games*, vol. 10, no. 2, pp. 209–220, 2017.
- [41] J. Liu, D. Perez-Liebana, and S. M. Lucas, “The single-player gvgai learning framework-technical manual,” in *Technical report*. Queen Mary University of London, 2017.
- [42] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, “Deep reinforcement learning for general video game AI,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.
- [43] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, “Monte-carlo tree search: A new framework for game ai.” in *AIIDE*, 2008.
- [44] N. Metropolis and S. Ulam, “The monte carlo method,” *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.
- [45] B. D. Abramson, “The expected-outcome model of two-player games,” Ph.D. dissertation, Columbia University, New York, NY, USA, 1987, aAI8827528.
- [46] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *European conference on machine learning*. Springer, 2006, pp. 282–293.
- [47] H. Finnsson and Y. Björnsson, “Simulation-based approach to general game playing.” in *Aaai*, vol. 8, 2008, pp. 259–264.

## Bibliography

- [48] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius, “Investigating mcts modifications in general video game playing,” in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 107–113.
- [49] T. Schuster, “Mcts based agent for general video games,” Ph.D. dissertation, Masters thesis, Department of Knowledge Engineering, Maastricht University , 2015.
- [50] D. J. Soemers, C. F. Sironi, T. Schuster, and M. H. Winands, “Enhancements for real-time monte-carlo tree search in general video game playing,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.
- [51] D. Perez, S. Samothrakis, and S. Lucas, “Knowledge-based fast evolutionary mcts for general video game playing,” in *2014 IEEE Conference on Computational Intelligence and Games*. IEEE, 2014, pp. 1–8.
- [52] E. J. Jacobsen, R. Greve, and J. Togelius, “Monte mario: platforming with mcts,” in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2014, pp. 293–300.
- [53] C. F. Sironi, J. Liu, D. Perez-Liebana, R. D. Gaina, I. Bravi, S. M. Lucas, and M. H. Winands, “Self-adaptive mcts for general video game playing,” in *International Conference on the Applications of Evolutionary Computation*. Springer, 2018, pp. 358–375.
- [54] C. F. Sironi and M. H. Winands, “Analysis of self-adaptive monte carlo tree search in general video game playing,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–4.
- [55] E. H. dos Santos and H. S. Bernardino, “Redundant action avoidance and non-defeat policy in the monte carlo tree search algorithm for general video game playing,” *Proceedings do XVI Simpsio Brasileiro de Jogos e Entretenimento Digital*, 2017.

## Bibliography

- [56] I. Bravi, A. Khalifa, C. Holmgård, and J. Togelius, “Evolving game-specific ucb alternatives for general video game playing,” in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 393–406.
- [57] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2, no. 4.
- [58] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A Survey of Monte Carlo Tree Search Methods,” *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4:1, pp. 1–43, 2012.
- [59] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, “Rolling horizon evolution versus tree search for navigation in single-player real-time games,” in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 351–358.
- [60] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, “Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing,” in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 418–434.
- [61] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, “Rolling horizon evolution enhancements in general video game playing,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 88–95.
- [62] R. D. Gaina, S. M. Lucas, and D. Pérez-Liébana, “Population seeding techniques for rolling horizon evolution in general video game playing,” in *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2017, pp. 1956–1963.
- [63] B. Santos, H. Bernardino, and E. Hauck, “An improved rolling horizon evolution algorithm with shift buffer for general game playing,” in *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2018, pp. 31–316.

## Bibliography

- [64] B. Jia, M. Ebner, and C. Schack, “A gp-based video game player,” in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 1047–1053.
- [65] B. Jia and M. Ebner, “A strongly typed gp-based video game player,” in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 299–305.
- [66] D. Pérez-Liébana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, “Analyzing the robustness of general video game playing agents,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.
- [67] T. Joppen, M. U. Moneke, N. Schröder, C. Wirth, and J. Fürnkranz, “Informed hybrid game tree search for general video game playing,” *IEEE Transactions on Games*, vol. 10, no. 1, pp. 78–90, 2018.
- [68] A. Mendes, J. Togelius, and A. Nealen, “Hyper-heuristic general video game playing,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.
- [69] D. Ashlock, D. Perez-Liebana, and A. Saunders, “General video game playing escapes the no free lunch theorem,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 17–24.
- [70] T. Vodopivec, S. Samothrakis, and B. Ster, “On monte carlo tree search and reinforcement learning,” *Journal of Artificial Intelligence Research*, vol. 60, pp. 881–936, 2017.
- [71] B. Ross, “General video game playing with goal orientation,” *Diss. Masters Thesis, University of Strathclyde*, 2014.
- [72] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

## Bibliography

- [73] Y. Bengio, “Learning Deep Architectures for AI,” *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
- [74] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [75] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, “Unifying count-based exploration and intrinsic motivation,” in *Advances in Neural Information Processing Systems*, 2016, pp. 1471–1479.
- [76] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, “Go-explore: a new approach for hard-exploration problems,” *arXiv preprint arXiv:1901.10995*, 2019.
- [77] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, p. 354, 2017.
- [78] B. V. Dasarathy and B. V. Sheela, “A composite classifier system design: Concepts and methodology,” *Proceedings of the IEEE*, vol. 67, no. 5, pp. 708–713, 1979.
- [79] L. K. Hansen and P. Salamon, “Neural network ensembles,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 10, pp. 993–1001, 1990.
- [80] M. Woźniak, M. Graña, and E. Corchado, “A survey of multiple classifier systems as hybrid systems,” *Information Fusion*, vol. 16, pp. 3–17, 2014.
- [81] O. G. Selfridge, “Pandemonium: A paradigm for learning,” *the Mechanisation of Thought Processes, 1958*, 1958.
- [82] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager *et al.*, “Building watson: An overview of the deepqa project,” *AI magazine*, vol. 31, no. 3, pp. 59–79, 2010.



## Bibliography

- [83] Y. Chen, J. E. Argentinis, and G. Weber, “Ibm watson: how cognitive computing can be applied to big data challenges in life sciences research,” *Clinical therapeutics*, vol. 38, no. 4, pp. 688–701, 2016.
- [84] P. Rodgers, J. Levine, and D. Anderson, “Ensemble decision making in real-time games,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.
- [85] J. Togelius, personal communication.
- [86] G. J. Sussman, “A computational model of skill acquisition,” 1973.
- [87] A. Tversky and D. Kahneman, “Judgment under uncertainty: Heuristics and biases,” *science*, vol. 185, no. 4157, pp. 1124–1131, 1974.
- [88] G. Gigerenzer and D. G. Goldstein, “Reasoning the fast and frugal way: models of bounded rationality.” *Psychological review*, vol. 103, no. 4, p. 650, 1996.
- [89] “Bioshock,” [Steam], 2K Games, 2007.
- [90] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, “Deep learning for video game playing,” *IEEE Transactions on Games*, 2019.
- [91] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, “Openai baselines,” *GitHub, GitHub repository*, 2017.
- [92] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*, 2016, pp. 1928–1937.
- [93] W. Mischel, E. B. Ebbesen, and A. Raskoff Zeiss, “Cognitive and attentional mechanisms in delay of gratification.” *Journal of personality and social psychology*, vol. 21, no. 2, p. 204, 1972.

## Bibliography

- [94] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. New York, NY, USA: Wiley-Interscience, 2006.
- [95] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, “Matching games and algorithms for general video game playing,” in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
- [96] D. Balduzzi, K. Tuyls, J. Perolat, and T. Graepel, “Re-evaluating evaluation,” in *Advances in Neural Information Processing Systems*, 2018, pp. 3268–3279.
- [97] D. Anderson, “Infogain algorithm.” [Online]. Available: <https://github.com/Damorin/InfoGain/>
- [98] C. Guerrero-Romero, A. Louis, and D. Perez-Liebana, “Beyond playing to win: Diversifying heuristics for gvgai,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 118–125.
- [99] D. Anderson, “Generalized video game playing,” 2015.
- [100] D. Perez-Liebana, S. Mostaghim, and S. M. Lucas, “Multi-objective tree search approaches for general video game playing,” in *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2016, pp. 624–631.
- [101] S. Bubeck and R. Munos, “Open loop optimistic planning.” in *COLT*, 2010, pp. 477–489.
- [102] D. Anderson, “Eds experiment code,” 2019. [Online]. Available: <https://github.com/Damorin/PhD-Ensemble-GVGAI>

## Appendix A

# Deceptive Games

# Deceptive Games

Damien Anderson<sup>1</sup>, Matthew Stephenson<sup>2</sup>, Julian Togelius<sup>3</sup>, Christoph Salge<sup>3</sup>,  
John Levine<sup>1</sup>, and Jochen Renz<sup>2</sup>

<sup>1</sup> Computer and Information Science Department, University of Strathclyde,  
Glasgow, UK,

`Damien.Anderson@strath.ac.uk`

<sup>2</sup> Research School of Computer Science, Australian National University, Canberra,  
Australia

<sup>3</sup> NYU Game Innovation Lab, Tandon School of Engineering, New York University,  
New York, USA

**Abstract.** Deceptive games are games where the reward structure or other aspects of the game are designed to lead the agent away from a globally optimal policy. While many games are already deceptive to some extent, we designed a series of games in the Video Game Description Language (VGDL) implementing specific types of deception, classified by the cognitive biases they exploit. VGDL games can be run in the General Video Game Artificial Intelligence (GVGAI) Framework, making it possible to test a variety of existing AI agents that have been submitted to the GVGAI Competition on these deceptive games. Our results show that all tested agents are vulnerable to several kinds of deception, but that different agents have different weaknesses. This suggests that we can use deception to understand the capabilities of a game-playing algorithm, and game-playing algorithms to characterize the deception displayed by a game.

**Keywords:** Games, Tree Search, Reinforcement Learning, Deception

## 1 Introduction

### 1.1 Motivation

What makes a game difficult for an Artificial Intelligence (AI) agent? Or, more precisely, how can we design a game that is difficult for an agent, and what can we learn from doing so?

Early AI and games research focused on games with known rules and full information, such as Chess [1] or Go. The game-theoretic approaches [2] to these games, such as min-max, are constrained by high branching factors and large computational complexity. When Deep Blue surpassed the top humans in Chess [3], the game Go was still considered very hard, partly due to its much larger branching factor. Also, the design of Arimaa [4], built to be deliberately difficult for AI agents, relies heavily on an even higher branching factor than Go.

But increasing the game complexity is not the only way to make games more difficult. To demonstrate this we will here focus on old arcade games, such as

Sokoban, Dig Dug or Space invaders, which can be implemented in VGDL. Part of the motivation for the development of VGDL and GVGAI was the desire to create a generic interface that would allow the same AIs to play a range of different games. GVGAI competitions have been held annually since 2013, resulting in an openly accessible corpus of games and AI agents that can play them (with varying proficiency).

VGDL games have relatively similar game complexity: the branching factor is identical (there are six possible actions) and the game state space is not too different between games because of the similar-sized levels. Yet, if we look at how well different agents do on different games we can see that complexity is not the only factor for game difficulty. Certain games seem to be very easy, while others are nearly impossible to master for all existing agents. These effects are still present if the agents are given considerably more time which could compensate for complexity [5]. Further analyses also shows that games cannot easily be ordered by difficulty, as agents based on different types of algorithms seem to have problems with different games—there is a distinct non-transitivity in performance rankings [6]. This raises the question of what makes a game difficult for a specific agent but not for others?

One way to explain this is to consider that there are several methods for constructing agents to play games. One can train a function approximator to map from a state observation to an action using reinforcement learning algorithms based on approximate dynamic programming (the temporal difference family of methods), policy gradients or artificial evolution; alternatively, and complementary, if you have a forward model of the game you can use tree search or evolution to search for action sequences that maximize some utility [7]. Additionally, there are hybrid algorithms combining elements from several of these methods, such as the very successful AlphaGo[8] system which combines supervised learning, approximate dynamic programming and Monte Carlo Tree Search.

A commonality between these game-playing methods is that they rely on rewards to guide their search and/or learning. Policies are learned to maximize the expected reward, and when a model is available, action sequences are selected for the same criterion. Fortunately, rewards are typically well-defined in games: gaining score is good, losing lives or getting hurt is bad. Indeed, one of the reasons for the popularity of games as AI testbeds is that many of them have well-defined rewards (they can also be simulated cheaply, safely and speedily). But it's not enough for there to be rewards; the rewards can be structured in different ways. For example, one of the key problems in reinforcement learning research, credit allocation, is how to assign reward to the correct action given that the reward frequently occurs long after the action was taken.

Recently, much work has gone into devising reinforcement learning algorithms that can learn to play simple arcade games, and they generally have good performance on games that have short time lags between actions and rewards. For comparison, a game such as *Montezuma's Revenge* on the Atari 2600, where there is a long time lag between actions and rewards, provides a very hard challenge for all known reinforcement learning algorithms.

It is not only a matter of the time elapsed between action and reward; rewards can be more or less helpful. The reward structure of a game can be such that taking the actions that lead to the highest rewards in the short-to-medium term leads to lower overall rewards, i.e. playing badly. For example, if you spend all your time collecting coins in *Super Mario Bros*, you will likely run out of time. This is not too unlike the situation in real life where if you optimize your eating policy for fat and sugar you are likely to achieve suboptimal global nutritional reward. Designing a reward structure that leads an AI away from the optimal policy can be seen as a form of deception, one that makes the game harder, regardless of the underlying game complexity. If we see the reward function as a heuristic function approximating the (inverse) distance from a globally optimal policy, a deceptive reward function is an *inadmissible heuristic*.

## 1.2 Biases, deception and optimization

In order to understand why certain types of agents are weak against certain kinds of deceptions it is helpful to consider different types of deception through the lens of cognitive biases. Deceptive games can be seen as exploiting a specific cognitive bias<sup>4</sup> of the (human or AI) player to trick them into making a suboptimal decision. Withholding or providing false information is a form of deception, and can be very effective at sabotaging a player's performance. In this paper though, we want to focus on games where the player or AI has full access to both the current game state and the rules (forward model). Is it still possible to design a game with these constraints that tricks an artificial agent? If we were facing a game with unlimited resources, the answer would be no, as unbounded computational resources makes deception impossible: an exhaustive search that considers all possible action sequences and rates them by their fully modeled probabilistic expected outcome will find the optimal strategy. Writing down what a unbounded rational agent should do is not difficult. In reality, both humans and AI agents have bounded rationality in that they are limited in terms of computational resources, time, memory, etc.

To compensate for this, artificial intelligence techniques rely on approximations or heuristics that are easier to compute and still return a better answer than random. In a naive interpretation, this seems to violate the free lunch theorem. This is still a viable approach though if one only deals with a subset of all possible problems. These assumptions about the problems one encounters can be turned into helpful cognitive biases. In general, and in the right context, this is a viable cognitive strategy - one that has been shown to be effective for both humans and AI agents [9,10]. But reasoning based on these assumptions also makes one susceptible to deceptions - problems that violate this assumption and are designed in a way so that the, now mistaken, assumption leads the player to a suboptimal answer. Counter-intuitively, this means that the more sophisticated an AI agent becomes, the better it is at exploiting typical properties of

---

<sup>4</sup> To simplify the text we talk about the game as if it has agency and intentions; in truth the intentions and agency lies with the game's designer, and all text should be understood in this regard.

the environment, the more susceptible it becomes to specific deceptions based on those cognitive biases.

This phenomenon can be related to the No Free Lunch theorem for search and optimization, which implies that, given limited time, making an agent perform better on a particular class of search problems will make it perform worse on others (because over all possible search problems, all agents will perform the same) [11]. Of course, some search algorithms are in practice better than others, because many “naturally occurring” problems tend to fall in a relatively restricted class where deception is limited. Within evolutionary computation, the phenomenon of deceptive optimization problems is well-defined and relatively well-studied, and it has been claimed that the only hard optimization problems are the deceptive ones [12,13].

For humans, the list of cognitive biases is quite extensive, and subsequently, there are many different deception strategies for tricking humans. Here we focus on agent which have their own specific sets of biases. Identifying those cognitive biases via deceptive games can help us to both categorize those agents, and help us to figure out what they are good at, and on what problem they should be used. Making the link to human biases could also help us to understand the underlying assumptions humans use, enabling us to learn from human mistakes what shortcuts humans take to be more efficient than AIs.

### 1.3 Overview

The rest of this paper is structured as follows. We first outline some AI-specific deceptions based on our understanding of current game-playing algorithms. We present a non-exhaustive list of those, based on their assumptions and vulnerabilities. We then introduce several new VGDL games, designed to specifically deceive the existing AI algorithms. We test a range of existing agents from the GVGAI framework on our new deceptive games and discuss the results.

## 2 Background

### 2.1 Categories of Deception

By linking specific cognitive biases to types of deception we can categorize different deceptive games and try to predict which agents would perform well on them. We can also construct deceptive games aimed at exploiting a specific weakness. The following is a non-exhaustive list of possible AI biases and their associated traps, exemplified with some of the games we present here.

**Greed Trap:** A common problem simplification is to only consider the effect of our actions for a limited future. These greedy algorithms usually aim to maximize some immediate reward and rely on the assumption that the local reward gradient will guide them to a global maximum. One way to specifically exploit this bias (a greedy trap) is to design a game with an accumulated reward and

then use some initial small reward to trick the player into an action that will make a later, larger reward unattainable. The later mentioned *DeceptiCoins* and *Sister Saviour* are examples of this. Delayed rewards, such as seen in *Invest* and *Flower*, are a subtype. In that case, an action has a positive reward that is only awarded much later. This can be used to construct a greedy trap by combining it with a smaller, more immediate reward. This also challenges algorithms that want to attach specific rewards to actions, such as reinforcement learning.

**Smoothness Trap:** Several AI techniques also rely on the assumption that good solutions are “close” to other good solutions. Genetic Algorithms, for example, assume a certain smoothness of the fitness landscape and MCTS algorithms outperform uninformed random tree search because they bias their exploration towards branches with more promising results. This assumption can be exploited by deliberately hiding the optimal solutions close to a many really bad solutions. In the example of *DeceptiZelda* the player has two paths to the goal. One is a direct, safe, low reward route to the exit which can be easily found. The other is a long route, passing by several deadly hazards but incurring a high reward if the successful route is found. Since many of the solutions along the dangerous part lead to losses, an agent operating with the smoothness bias might be disinclined to investigate this direction further, and would therefore not find the much better solution. This trap is different from the greedy trap, as it aims at agents that limit their evaluation not by a temporal horizon, but by only sampling a subset of all possible futures.

**Generality Trap:** Another way to make decision-making in games more manageable, both for humans and AI agents, is to generalize from particular situations. Rather than learning or determining how to interact with a certain object in every possible context, an AI can be more efficient by developing a generalized rule. For example, if there is a sprite that kills the avatar, avoiding that sprite as a general rule might be sensible. A generality trap can exploit this by providing a game environment in which such a rule is sensible, but for few critical exceptions. *WafterThinMints* aims to realize this, as eating mints gives the AI points unless too many are eaten. So the agent has to figure out that it should eat a lot of them, but then stop, and change its behavior towards the mints. Agents that would evaluate the gain in reward greedily might not have a problem here, but agents that try to develop sophisticated behavioral rules should be weak to this deception.

## 2.2 Other deceptions

As pointed out, this list is non-exhaustive. We deliberately excluded games with hidden or noisy information. Earlier GVGAI studies have looked at the question of robustness [14], where the forward model sometimes gives false information. But this random noise is still different from a deliberate withholding of game



information, or even from adding noise in a way to maximize the problems for the AI.

We should also note that most of the deceptions implemented here are focused on exploiting the reward structure given by the game to trick AIs that are optimized for actual rewards. Consider though, that recent developments in intrinsically motivated AIs have introduced ideas such as curiosity-driven AIs to play games such as *Montezuma’s Revenge* [15] or *Super Mario* [16]. The internal curiosity reward enhances the AI’s gameplay, by providing a gradient in a flat extrinsic reward landscape, but in itself makes the AI susceptible to deception. One could design a game that specifically punished players for exploration.

### 3 Experimental Setup

#### 3.1 The GVGAI Framework

The General Video Game AI competition is a competition focused on developing AI agents that can play real-time video games; agents are tested on unseen games, to make sure that the developer of the agent cannot tailor it to a particular game [17]. All current GVGAI games are created in VGDL, which was developed particularly to make rapid and even automated game development possible [18]. The competition began with a single planning track which provided agents with a forward model to simulate future states but has since expanded to include other areas, such as a learning track, a rule generation track, and a level generation track [19].

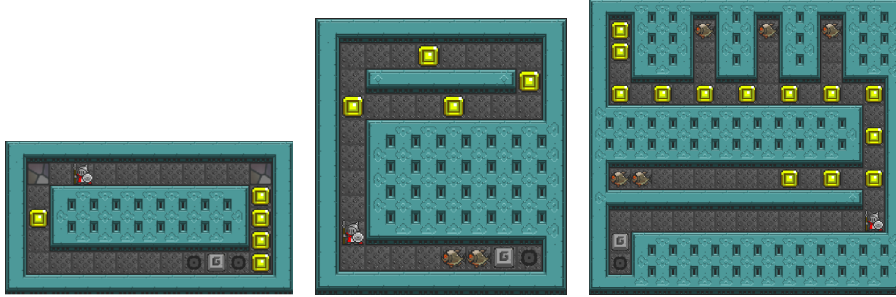
In order to analyze the effects of game deception on GVGAI agent performance, a number of games were created (in VGDL) that implemented various types of deception in a relatively “pure” form. This section briefly explains the goal of each game and the reasons for its inclusion. In order to determine whether an agent had selected the rational path or not, requirements were set based on the agent’s performance, which is detailed in this section also.

#### 3.2 DeceptiCoins (DC)

The idea behind *DeceptiCoins* is to offer agents two options for which path to take. The first path has some immediate rewards and leads to a win condition. The second path similarly leads to a win condition but has a higher cumulative reward along its path, which is not immediately visible to a short-sighted agent. Once a path is selected by the agent, a wall closes behind them and they are no longer able to choose the alternative path.

In order for the performance of an agent to be considered rational in this game, the agent must choose the path with the greatest overall reward. In figure 1, this rational path is achieved by taking the path to the right of the agent, as it will lead to the highest amount of score.

Two alternative levels were created for this game. These levels are similar in how the rules of the game work, but attempt to model situations where an agent







**Fig. 1.** The first level of DeceptiCoins

**Fig. 2.** The second level of DeceptiCoins

**Fig. 3.** The third level of DeceptiCoins

may get stuck on a suboptimal path by not planning correctly. Level 2, shown in figure 2, adds some enemies to the game which will chase the agent. The agents need to carefully plan out their moves in order to avoid being trapped and losing the game. Level 3, shown in figure 3 has a simple path which leads to the win condition, and a risky path that leads to large rewards. Should the agent be too greedy and take too much reward, the enemies in the level will close off the path to the win condition and the agent will lose.

The sprites used are as follows:

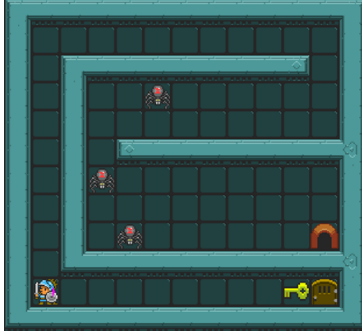
-  Avatar - Represents the player/agent in the game.
-  Gold Coin - Awards a point if collected.
-  G Square - Leads to winning the game when interacted with.
-  Piranha - Enemies, if the avatar interacts with these, the game is lost.

The rational paths for level 2 and 3 are defined as reaching the win condition of the level, while also collecting a minimum amount of reward (5 for level 2 and 10 for level 3).

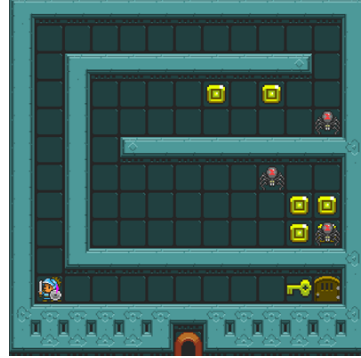
### 3.3 DeceptiZelda (DZ)

*DeceptiZelda* looks at the risk vs reward behavior of the GVGAI agents. As in *DeceptiCoins*, two paths are presented to the agent, with one leading to a quick victory and the other leading to a large reward, if the hazards are overcome. The hazards in this game are represented as moving enemies which must either be defeated or avoided.

Two levels for this game were created as shown in figure 5 and figure 4. The first level presents the agent with a choice of going to the right, collecting the key and exiting the level immediately without tackling any of the enemies. The second path leading up takes the agent through a hazardous corridor where they must pass the enemies to reach the alternative goal. The second level uses the same layout but instead of offering a win condition, a lot of collectible rewards are offered to the agent, who must collect these and then return to the exit.









**Fig. 4.** The first level of Deceptizelda



**Fig. 5.** The second level of Deceptizelda

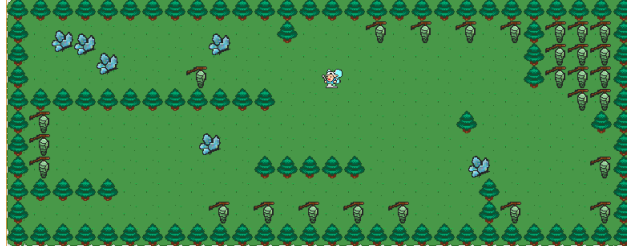
The sprites used are as follows:

-  Avatar: Represents the player/agent in the game.
-  Spider: The enemies to overcome. If defeated awards 2 points.
-  Key: Used to unlock the first exit. Awards a point if collected.
-  Gold Coin: Awards a point to the agent if collected.
-  Closed Door: The low value exit. Awards a point if moved into.
-  Open Door: The high value exit. Awards 10 points if moved into.

The rational path for this game is defined as successfully completing the path with the most risk. In the first level, this is defined as achieving at least 10 points and winning the game. This can be done by taking the path leading up and reaching the exit beyond the enemies. The second level of *DeceptiZelda* is played on the same map, but instead of offering a higher reward win condition, a large amount of reward is available, and the agent has to then backtrack to the single exit in the level. This level can be seen in figure 5.

### 3.4 Butterflies (BF)




*Butterflies* is one of the original games for the GVGAI that prompted the beginning of this work. This game presents a situation where if the agent aims for the win condition too quickly, they will lower their maximum potential score for the level. The goal of the game is simple; collect all of the butterflies before they reach their cocoons, which in turn creates more butterflies. To solve the game all that is required is that every butterfly is collected. Each collected butterfly grants a small reward to the agent. If the agent is able to defend a single cocoon and wait until all other cocoons have been spawned, there will be the maximum number of butterflies available to gain reward from. So long as the last cocoon



**Fig. 6.** The first level of Butterflies

is not touched by a butterfly, the game can still be won, but now a significantly higher score is possible. The level used is shown in figure 6.

The sprites used are as follows:




-  Avatar: Represents the player/agent in the game.
-  Butterfly: Awards 2 points if collected.
-  Cocoon: If a butterfly interacts with these, more butterflies are created.

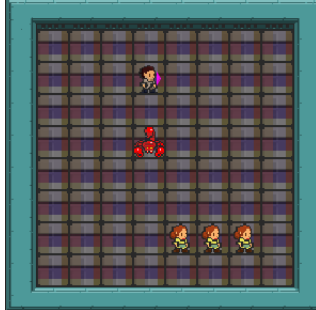
The rational path for *Butterflies* is defined as any win condition with a final score over 30. This is achieved by allowing more than half of the cocoons to be spawned and then winning the level.

### 3.5 SisterSaviour (SS)

The concept of *SisterSaviour* was to present a moral choice to the agent. There are 3 hostages to rescue in each level, and a number of enemies guarding them, as shown in figure 7. It is not possible for the agent to defeat these enemies immediately. The agent is given a choice of either rescuing the hostages or killing them. If the agent chooses to rescue the hostages they receive a small reward and will be able to defeat the enemies, which grants a large point reward. On the other hand, if the agent chooses to kill the hostages, they are granted a larger reward immediately, but now lack the power to defeat the enemies and will lose the game.

The sprites used are as follows:

-  Avatar: Represents the player/agent in the game.
-  Scorpion: An enemy which chases the avatar. Immune to attacks from the avatar, unless all of the hostages have been rescued. Awards 14 points if defeated.
-  Hostage: Can be either killed, by attacking them or rescued by moving into their space. Awards 2 points if killed, and 1 point if rescued. If all are rescued then the avatar can kill the enemy.



**Fig. 7.** The first level of SisterSaviour








**Fig. 8.** The first level of Invest

The rational path for *SisterSaviour* is defined as reaching a score of 20. This involves rescuing all of the hostages, by moving the avatar onto their space, and then defeating the enemy.

### 3.6 Invest (Inv)

*Invest* looks at the ability of a GVGAI agent to spend their accumulated reward, with the possibility of receiving a larger reward in the future. This game is shown in figure 8. The agent begins with a set number of points which need to be collected from the level, which can then be spent on investment options. This is done by moving onto one of the 3 human characters to the north of the level. Investing will deduct an amount from their current score, acting as an immediate penalty, and will trigger an event to occur at a random point in the future where the agent will receive a large score reward. Should the agent invest too much, and go into a negative score, then the game is lost, otherwise, they will eventually win. The interesting point of this game was how much reward they accumulate over the time period that they have, and would they overcome any loss adversity in order to gain higher overall rewards?

The sprites used are as follows:



-  Avatar: Represents the player/agent in the game.
-  Gold Coin: Awards a point when collected.
-  Green Investment: Takes 3 points when moved onto, returns 8.
-  Red Investment: Takes 7 points when moved onto, returns 15.
-  Blue Investment: Takes 5 points when moved onto, returns 10.

The rational path in *Invest* is defined as investing any amount of score successfully without suffering a loss.

### 3.7 Flower (Flow)

*Flower* is a game which was designed to offer small immediate rewards, and progressively larger rewards if some time is allowed to pass for the reward to grow. As shown in figure 9, a single seed is available for the agent to collect, which is worth 0 points. As time passes the value of the seed increases as it grows into a full flower, from 0 up to 10. Once collected, the seed will begin to regrow, starting from 0 again. The rational solution for this game is to wait for a seed to grow into a full flower, worth 10 points, and then collecting it.

The sprites used are as follows:

-  Avatar: Represents the player/agent in the game.
-  Seed: Awards 0 points initially, but this increases up to 10.




The rational path in *Flower* is defined as achieving a score of at least 30. This can only be done by allowing the flower to grow to at least the second stage and consistently collecting at that level.

### 3.8 WaferThinMints (Mints)

*WaferThinMints* introduces the idea that gathering too much reward can lead to a loss condition. The agent has to gather resources in order to increase their reward, but if they collect too many they will die and lose the game.

Two variants of this game were created. One which includes an exit from the level, shown in figure 11, and one that does not, shown in figure 10. These variants were created in order to provide a comparison of the effect that the deception in the level has on overall agent performance.

The sprites used are as follows:

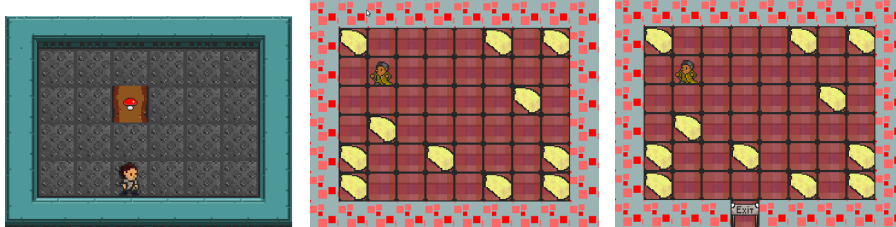
-  Avatar: Represents the player/agent in the game.
-  Cheese: Awards a point when collected. If 9 have been collected already, then the 10th will kill the avatar causing a loss.
-  Exit: Leads to a win condition when moved into.

The rational path for both versions of the game is defined as collecting a score of 9, and then either waiting for the timeout, in level 1 or exiting the game, in level 2.

## 4 Experiments and Results

The agents used were collected from the GVGAI competitions. Criteria for selection were the uniqueness of the algorithm used and competition ranking in the past. The hardware used for all of the experiments was a Ubuntu 14.04 desktop PC with an i7-4790 CPU and 16GB Ram.

Each agent was run 10 times on each level of the deceptive games outlined in section 3. If an agent was disqualified for any reason it was given another



**Fig. 9.** The first level of Flower **Fig. 10.** The first level of WaferThinMints **Fig. 11.** The second level of WaferThinMints

run to collect 10 successful results for each game and agent. In addition to comparing these performance statistics, observations were made on the choices that the agents made when faced with potentially deceptive choices. Each game’s rational path is defined in section 3. The results of these experiments are shown in table 12. Each game was played a total of 360 times. The totals at the bottom of the table show how many of those games were completed using the defined rational path. The results are ranked in descending order by their number of rational trials, and then the number of games where they managed to play with 100% rationality.

Noticeably from the initial results is that no single algorithm was able to solve all the games, with *DeceptiZelda* and *SisterSaviour* being particularly challenging. Furthermore, no single algorithm dominated all others in all games. For Example, IceLab, the top agent in overall results, only has 2 rational trials in *Butterflies*, compared to 9 for Greedy Search, which is in the 33rd place. In general, the results for *Butterflies* are interesting, as top agents perform poorly compared to some of the lower ranking agents.

*Butterflies* also has a good spread of results, with all but 4 of the algorithms being able to find the rational path at least once. While many of the algorithms are able to make some progress with the game, only 2 are able to achieve 100% rationality.

There is an interesting difference in the performance of agents between *DeceptiCoins* level 1 and 2. The agents that performed well in *DeceptiCoins* 1 seemed to perform significantly worse in level 2. The requirements of the levels are quite different which appears to have a significant effect on the agents. If a ranking was done with only the performance of *DeceptiCoins* level 2 then IceLab, the 1st ranked in this experiment, would be in the bottom half of the results table.

The hardest games for the agents to solve were *DeceptiZelda* levels 1 and 2, and *SisterSaviour*. *DeceptiZeldas* levels had only 4 and 13 runs solved respectively, and *SisterSaviour* having 14. These games present interesting challenges to the agents, with the rational solution requiring a combination of long-range planning and sacrificing apparent reward for the superior, long-range goal.

Another interesting case here is Mints, the only game in our set with a generalization trap. Most algorithms do well in Mints, suggesting that they do

not generalize. This is to be expected, as a tree search algorithm does not in itself generalize from one state to another. But bladerunner, AtheneAI, and SJA86 completely fail at these games, even though they perform reasonably well otherwise. This suggests that they perform some kind of surrogate modeling of game states, relying on a generality assumption that this game breaks. The inclusion of an accessible win condition in *Mints 2* also dramatically reduced the number of algorithms that achieved the maximum amount of score, from 26 to 8. This seems to be due to also introducing a specific greed trap that most algorithms seem to be susceptible too - namely preferring to win the game outright, over accumulating more score.

Note that, the final rankings of this experiment differ quite significantly from the official rankings on the GVGAI competition. It is important to note that a different ranking algorithm is used in the competition, which may account for some of the differences observed. Many of the agents have a vastly different level of performance in these results compared to the official rankings. First of all, IceLab and MH2015 have historically appeared low in the official rankings, with their highest ranks being 10th place. The typical high ranking algorithms in the official competition seem to have been hit a bit harder by the new set of games. Yolobot, Return42, maastCTS2, YBCriber, adriencx and number27 tend to feature in the top 5 positions of the official rankings, and have now finished in positions 2, 4, 15, 8, 9, and 7. For them to lose their positions in this new set of games could show how the games can be constructed to alter the performance of agents [17,19].

In order to look at the effect of deception on specific types of algorithms, such as genetic algorithms (GA) or Tree Search techniques, a second set of experiments were performed. A selection of algorithms were ran an additional 10 times on each of the games, and each algorithm was investigated to identify the core component of its operation. It should be noted that these classifications are simple, and an in-depth analysis of the specifics used by the algorithms might reveal some further insights. The results for these experiments are shown in figure 13.

These results show a number of interesting observations. First of all, for *DeceptiZelda1* and 2 it appears that agents using a genetic algorithm perform better than most other approaches, but do poorly compared to tree search techniques at *SisterSaviour*. Portfolio search agents, which employ different algorithms for different games or situations, take the top two positions of the table and place quite highly overall compared to single algorithm solutions.

## 5 Discussion and Future Work

The results suggest that the types of deception presented in the games have differing effects on the performance of different algorithms. The fact that algorithms, that are more sophisticated and usually perform well in the regular competition are not on top of the rankings is also in line with our argument, that they employ sophisticated assumptions and heuristics, and are subsequently



Agent Name	DC1	DC 2	DC 3	DZ 1	DZ 2	SS	BF	Flow	Inv	Mints 1	Mints 2	Rational
1. IceLab	10	2	10	0	0	0	2	10	10	10	10	8
2. Return42	10	1	7	0	0	2	1	10	10	10	0	8
3. MH2015	2	4	5	1	3	0	5	0	1	10	0	8
4. YoloBot	10	3	10	0	0	0	6	6	0	10	10	7
5. jaydee	9	6	9	0	0	0	1	10	0	10	3	7
6. NovTea	3	4	10	0	0	6	3	2	0	10	0	7
7. number27	0	5	4	0	1	0	4	6	3	10	0	7
8. YBCriber	10	5	10	0	0	0	0	0	10	10	10	6
9. adrienctx	0	3	4	0	0	0	1	10	0	10	10	6
10. TeamTopBug	9	5	10	0	0	0	1	10	0	10	0	6
11. Catlinux	0	7	10	0	4	0	6	7	0	10	0	6
12. muzzle	6	2	1	0	0	0	3	0	10	10	0	6
13. novelTS	1	4	10	0	0	0	3	0	4	10	0	6
14. bladerunner	10	4	8	0	3	0	1	0	10	0	0	6
15. maastCTS2	0	2	0	2	0	0	3	3	1	10	0	6
16. SJA86	1	1	0	0	1	0	9	6	0	1	0	6
17. Catlinux3	0	3	10	0	0	0	10	8	0	10	0	5
18. aStar	2	1	0	0	0	0	0	10	10	10	0	5
19. AtheneAI	10	6	0	1	0	0	10	0	10	0	0	5
20. Root	0	1	7	0	0	0	7	10	0	10	0	5
21. SJA862	3	0	7	0	0	0	3	0	10	10	0	5
22. roskvist	0	2	0	0	0	0	5	2	0	10	3	5
23. EvolutionStrategies	0	1	1	0	0	0	3	5	0	10	0	5
24. AlJim	0	1	0	0	0	0	9	1	5	5	0	5
25. HillClimber	2	0	0	0	0	0	7	3	1	0	1	5
26. MnMCTS	0	1	0	0	1	0	6	0	8	6	0	5
27. mrtndwrđ	0	0	0	0	0	4	3	4	0	10	0	4
28. simulatedAnnealing	4	0	0	0	0	0	8	4	0	0	1	4
29. TomVodo	3	0	0	0	0	0	5	5	0	4	0	4
30. ToVo1	1	0	0	0	0	0	9	1	0	9	0	4
31. Thorbjrn	0	4	5	0	0	0	9	0	1	0	0	4
32. BFS	0	1	0	0	0	0	5	0	10	0	0	3
33. Greedy Search	10	0	0	0	0	0	9	0	0	0	0	2
34. IterativeDeepening	0	0	0	0	0	2	1	0	0	0	0	2
35. Catlinux4	0	0	0	0	0	0	0	0	10	0	0	1
36. DFS	0	0	0	0	0	0	0	0	0	0	0	0
Totals	116	79	138	4	13	14	158	133	124	235	48	

Fig. 12. The results of the first experiment

susceptible to deception. Based on the data we have now it would be possible to build a game to defeat any of the agents on the list, and it seems possible to design a specific set of games that would put any specific AI at the bottom of the table. The difficulty of a game is, therefore, a property that is, at least in part, only well defined in regards to a specific AI.

In regards to categorization, it seems there is a certain degree of similarity between groups of games and groups of AIs that perform similarly, but a more in-depth analysis would be needed to determine what exact weakness each AI has. The games in this corpus already contain, like *Mints 2*, a mixture of different deceptions. Similarly, the more sophisticated agents also employ hybrid strategies

Agent Name	Algorithm	DC 1	DC 2	DC 3	DZ 1	DZ 2	SS	BF	Flow	Inv	Mints	Mints 2	Rational
1. IceLab	Portfolio	20	3	19	0	0	0	9	20	2	20	20	8
2. Return42	Portfolio	20	3	13	0	0	4	5	20	20	20	0	8
3. MH2015	GA	2	10	10	2	3	0	11	0	6	20	0	8
4. SJA86	MCTS	1	2	0	2	1	0	17	10	16	8	0	8
5. YBCriber	Portfolio	20	9	20	0	0	0	2	0	20	20	20	7
6. YoloBot	Portfolio	20	8	20	0	0	0	12	13	0	20	20	7
7. Catlinux	GA	0	10	20	2	4	0	15	15	0	20	0	7
8. muzzle	GA	12	3	2	0	0	1	10	0	20	20	0	7
9. NovTea	Tree	8	7	20	0	0	12	9	3	0	20	0	7
10. SJA862	MinMax	8	3	13	0	0	0	8	2	20	20	0	7
11. number27	Portfolio	0	9	6	0	1	0	11	12	5	20	0	7
12. adrienctx	MCTS	0	5	10	0	0	0	7	20	0	20	20	6
13. TeamTopBug	GA	19	9	20	0	0	0	3	20	0	20	0	6
14. bladerunner	Portfolio	20	6	12	4	5	0	3	0	0	0	0	6
15. EvolutionStrategies	GA	0	1	1	0	0	0	3	8	0	20	1	6
16. HillClimber	Hill	3	0	0	0	0	0	13	6	8	1	1	6
17. aStar	A*	4	1	0	0	0	0	0	20	20	20	0	5
18. novelTS	Tree	2	5	20	0	0	0	8	0	0	20	0	5
19. TomVodo	MCTS	5	0	0	0	0	0	14	8	3	8	0	5
20. mrtdwrdr	MCTS/A*	0	0	0	0	0	0	9	9	7	0	20	4
21. simulatedAnnealing	SA	8	0	0	0	0	0	14	10	0	0	1	4
22. Greedy Search	Tree	20	0	0	0	0	0	10	0	0	0	0	2
23. BFS	Best First	0	1	0	0	0	0	8	0	0	0	0	2
24. IterativeDeepening	ID	0	0	0	0	0	3	1	0	0	0	0	2
25. DFS	Depth	0	0	0	0	0	3	0	0	0	0	0	1
<b>Total Clever</b>		192	95	206	10	14	32	202	194	140	337	83	

Fig. 13. The results of the second experiment.

and some, like YoloBot, switch between different AI approaches based on the kind of game they detect [20]. One way to explore this further would be to use a genetic algorithm to create new VGDL games, with a fitness function rewarding a set of games that can maximally discriminate between the existing algorithms.

There are also further possibilities for deception that we did not explore here. Limiting access to the game state, or even requiring agents to actually learn how the game mechanics work open up a whole new range of deception possibilities. This would also allow us to extend this approach to other games, which might not provide the agent with a forward model, or might require the agent to deal with incomplete or noisy sensor information about the world.

Another way to deepen this approach would be to extend the metaphor about human cognitive biases. Humans have a long list of cognitive biases - most of them connected to some reasonable assumption about the world, or more specifically, typical games. By analyzing what biases humans have in this kind of games we could try to develop agents that use similar simplification assumptions to humans and thereby make better agents.

## 6 Acknowledgements

Damien Anderson is funded by the Carnegie Trust for the Universities of Scotland as a PhD Scholar. Christoph Salge is funded by the EU Horizon 2020 programme under the Marie Skłodowska-Curie grant 705643.

## References

1. Turing, A.M.: Chess. In: Bowden, B.V. (ed.) *Faster than Thought*, pp. 286–295. Pitnam, London (1953)
2. Von Neumann, J., Morgenstern, O.: *Theory of games and economic behavior*. Princeton University Press Princeton, NJ (1945)
3. Campbell, M., Hoane, A.J., Hsu, F.h.: Deep blue. *Artificial intelligence* 134(1-2), 57–83 (2002)
4. Syed, O., Syed, A.: Arimaa-a new game designed to be difficult for computers. *ICGA JOURNAL* 26(2), 138–139 (2003)
5. Nelson, M.J.: Investigating vanilla mcts scaling on the gvg-ai game corpus. In: *Computational Intelligence and Games (CIG)*, 2016 IEEE Conference on. pp. 1–7. IEEE (2016)
6. Bontrager, P., Khalifa, A., Mendes, A., Togelius, J.: Matching games and algorithms for general video game playing. In: *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*. pp. 122–128 (2016)
7. Yannakakis, G.N., Togelius, J.: *Artificial Intelligence and Games*. Springer (2018), <http://gameaibook.org>
8. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Driessche, G.V.D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K.: Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7585), 484–489 (2016), <http://dx.doi.org/10.1038/nature16961>
9. Tversky, A., Kahneman, D.: Judgment under uncertainty: Heuristics and biases. *Science* 185(4157), 1124–1131 (1974)
10. Gigerenzer, G., Goldstein, D.G.: Reasoning the fast and frugal way: models of bounded rationality. *Psychological review* 103(4), 650 (1996)
11. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* 1(1), 67–82 (1997)
12. Whitley, L.D.: Fundamental principles of deception in genetic search. In: *Foundations of Genetic Algorithms* (1991)
13. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions
14. Pérez-Liébana, D., Samothrakis, S., Togelius, J., Schaul, T., Lucas, S.M.: Analyzing the robustness of general video game playing agents. In: *Computational Intelligence and Games (CIG)*, 2016 IEEE Conference on. pp. 1–8. IEEE (2016)
15. Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R.: Unifying count-based exploration and intrinsic motivation. In: *Advances in Neural Information Processing Systems*. pp. 1471–1479 (2016)
16. Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363* (2017)
17. Perez-Liebana, D., Samothrakis, S., Togelius, J., Schaul, T., Lucas, S.M., Couëtoux, A., Lee, J., Lim, C.U., Thompson, T.: The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games* 8(3), 229–243 (2016)
18. Ebner, M., Levine, J., Lucas, S.M., Schaul, T., Thompson, T., Togelius, J.: Towards a video game description language. In: *Dagstuhl Follow-Ups*. vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2013)
19. Perez-Liebana, D., Samothrakis, S., Togelius, J., Lucas, S.M., Schaul, T.: General video game ai: Competition, challenges and opportunities. In: *Thirtieth AAAI Conference on Artificial Intelligence* (2016)

20. Mendes, A., Togelius, J., Nealen, A.: Hyper-heuristic general video game playing. In: Computational Intelligence and Games (CIG), 2016 IEEE Conference on. pp. 1–8. IEEE (2016)

## Appendix B

# Continuous Information Gain

# A Continuous Information Gain Measure to Find the Most Discriminatory Problems for AI Benchmarking

Matthew Stephenson<sup>1</sup>, Damien Anderson<sup>2</sup>, Ahmed Khalifa<sup>3</sup>,  
John Levine<sup>2</sup>, Jochen Renz<sup>1</sup>, Julian Togelius<sup>3</sup>, Christoph Salge<sup>3</sup>

<sup>1</sup>Research School of Computer Science, Australian National University, Canberra, Australia

<sup>2</sup>Computer and Information Science Department, University of Strathclyde, Glasgow, UK

<sup>3</sup>NYU Game Innovation Lab, Tandon School of Engineering, New York University, New York, USA

## Abstract

This paper introduces an information-theoretic method for selecting a small subset of problems which gives us the most information about a group of problem-solving algorithms. This method was tested on the games in the General Video Game AI (GVGAI) framework, allowing us to identify a smaller set of games that still gives a large amount of information about the game-playing agents. This approach can be used to make agent testing more efficient in the future. We can achieve almost as good discriminatory accuracy when testing on only a handful of games as when testing on more than a hundred games, something which is often computationally infeasible. Furthermore, this method can be extended to study the dimensions of effective variance in game design between these games, allowing us to identify which games differentiate between agents in the most complementary ways. As a side effect of this investigation, we provide an up-to-date comparison on agent performance for all GVGAI games, and an analysis of correlations between scores and win-rates across both games and agents.

## Introduction

Competitions and challenges are regularly used within AI as a way of evaluating algorithms, and also for promoting interest into specific problems. However, if the challenge poses a large set of possible problems it can often be impractical or even impossible to evaluate a new algorithm on every problem within this set. Comparing a new algorithm with the state of the art on the full set of problems can require immense computational resources, which are not available to many researchers. Therefore, a smaller set of problems is usually selected that intends to be representative of the entire problem space. This leads to the fundamental question: how should we select this subset of problems? This question is critical, as selecting a poorly representative subset of the problems available might leave out key aspects of the challenge, resulting in an unintentional bias that leads to specific solutions performing better than they would have on the entire problem set. If you have good knowledge of the domain, you could choose a set of problems with interestingly difficult design features, but there is no guarantee that these differences in design translate to meaningfully different challenges. Also in many cases, you do not have deep knowledge about the design of the different problems.

This issue is prevalent in any situation where it may be computationally prohibitive to test a new algorithm on all sub-problems presented. Examples of competitions or challenges where this is the case include the GVGAI (Perez-Liebana et al. 2016b), ALE (Bellemare et al. 2013) and Kaggle competitions (Carpenter 2011), each of which have hundreds of separate problems. There are also other sets of machine learning benchmarks that contain a multitude of disparate tasks, such as the OpenAI Gym (Brockman et al. 2016) or the UCI repository of supervised learning tasks (Dheeru and Karra Taniskidou 2017). Many of the participants in these challenges cherry-pick benchmarks where their new algorithm performs well. This will continue to happen as long as benchmark sets are so large that it is computationally infeasible to test on all benchmarks available. However, we should not simply reject good papers simply because the authors lack the resources to perform full evaluations on every benchmark possible. The solution to this dilemma is to test new algorithms on problems that are most relevant given the current set of well-performing algorithms, not simply those where the new algorithm performs best. This paper therefore proposes an approach for selecting a small number of problems out of a larger set for accurately testing an algorithm, and has great potential to revolutionize AI benchmarking across many different AI challenges.

As a first approach to the task of selecting which problems to test an algorithm on, we look at the correlations between different algorithm’s performance for different problems. After testing an algorithm on one problem, another problem could be selected with an anti-correlated performance profile (i.e. one where other algorithms perform differently). This is an incomplete solution however, most importantly because it does not tell us which problem to test on first and does not factor in the potential variability in agent performance. We instead propose an information-theoretic measure for determining which problems are best at telling a given set of algorithms apart; a measure that also takes into account the concept of noise when analyzing performance measures. By recursively applying this measure, we can find problems that are maximally informative considering previously selected problems, meaning that we can identify problems that discriminate among a set of algorithms in different ways (Martinez-Plumed and Hernández-Orallo 2016).

We use the General Video Game AI (GVGAI) frame-

work as a testbed for our method. The GVGAI library includes more than a hundred mini video games (Bontrager et al. 2016), and several dozen different agents that can play these games (Soemers et al. 2016; Gaina, Lucas, and Perez-Liebana 2017; Weinstein and Littman 2012; Pérez-Liébana et al. 2016c; Mendes, Togelius, and Nealen 2016) have been submitted to the associated GVGAI competition (Perez-Liebana et al. 2016a). We present a formalised analysis on the correlations between agent performances across different games, and use our information-theoretic measure to select a subset of the GVGAI game library that accurately represents the full discriminative scope when testing on all games available (i.e. provides a diverse range of problems that best distinguishes between agents).

## Background

General Video Game Playing (GVGP) is an area of research that looks to expand upon the success of the General Game Playing (GGP) competition. The GGP competition offers a platform for researchers to create agents that can play a wide range of board games (Genesereth, Love, and Pell 2005). While board games offer an interesting variety of problems, they do not offer real-time situations where rapid decision making is key, which is what the GVGP competition does by providing a library of video games as a research platform (Levine et al. 2013).

The GVGAI competition has been running annually since 2014 and provides a Video Game Description Language (VGDL) with which to quickly design games, and a common API for agents to access those games (Ebner et al. 2013). Each year ten games are selected to evaluate the submitted agents, which often covers a wide range of game types from role-playing to puzzle games (Perez-Liebana et al. 2018a). One of the key elements of this competition is that the games being played by the agents for each year’s competition are unknown to both the developers and agents beforehand. Many of the GVGAI games and most of the agents include some form of stochasticity, meaning that performance evaluation is inherently noisy. Playing a GVGAI game will also give two signals of performance, whether an agent won the game or not and what score was obtained (Perez-Liebana et al. 2016b). The competition currently offers multiple tracks, including a single and multi-player planning track (Gaina, Prez-Libana, and Lucas 2016), which provides a forward model for analyzing future game states, and a learning track which removes the forward model but allocates a training time to agents before submission (Pérez-Liébana et al. 2018b). For this paper, we only consider the games and agents used in the single-player planning track.

One of the issues that we set out to tackle with this work is that the games within the GVGAI framework are currently not well documented. A few previous papers attempted to evaluate how certain agents perform on different GVGAI games (Bontrager et al. 2016; Nelson 2016), but none have investigated the different discrimination profiles presented by the full game corpus, or how this information could be used to help design better agents and games in the future. The fact that different GVGAI games pose different types of problems to agents, may lead to biases towards a particular

type of algorithm when selecting game subsets. Understanding what bias may exist in a given set of games, and being able to select ten games which minimize any particular bias, is desirable for ensuring that the competition is genuinely evaluating general problem-solving capabilities.

## Data Collection

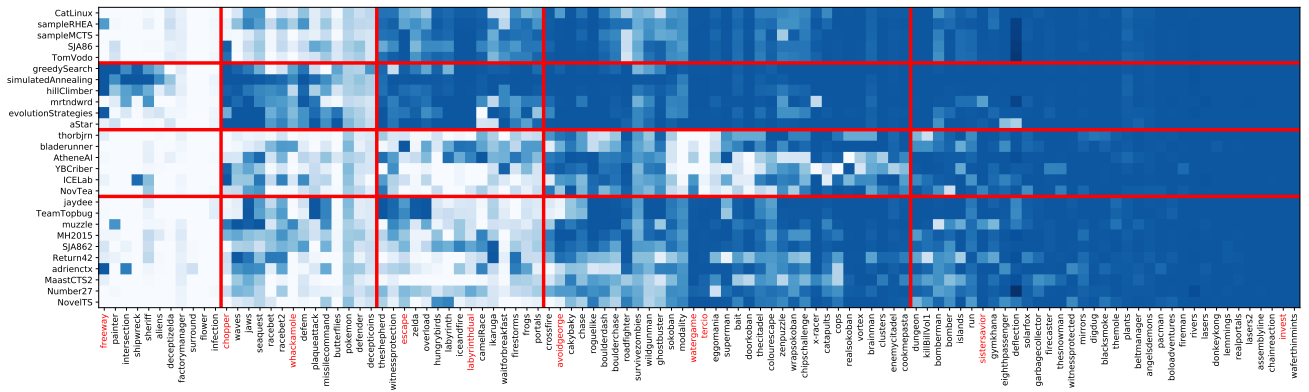
The first step towards analyzing different GVGAI games is to collect data from various playthroughs using a collection of agents. We used twenty-seven commonly available agents, which were some of the top performing entries in the previous GVGAI competitions over the last five years. The games that were used consist of the full corpus of 102 GVGAI games that are currently available (at the time of writing), plus an additional six deceptive GVGAI games introduced by Anderson et al. (Anderson et al. 2018), making the total number of games equal to 108.

Similar to the GVGAI competition format, each agent has 40 milliseconds to perform each action and runs for at most 2000 time steps. To replicate the competition environment, we ran the agents using 243 CPU cores with 2.6 GHz and 8 GB of memory. Each successful playthrough of a game that resulted in either a win or loss without any crashes produces one unit of data, containing the information [*agent, game, score, win/lose*]. Unfortunately, some of the agents occasionally crashed on certain games due to changes that the GVGAI framework has received through over the years, so not all the agents have the same amount of the generated data. A total of 3,990,760 successful playthroughs were recorded across all agents and games, with an average of 1,368.6 data samples per game-agent pair.

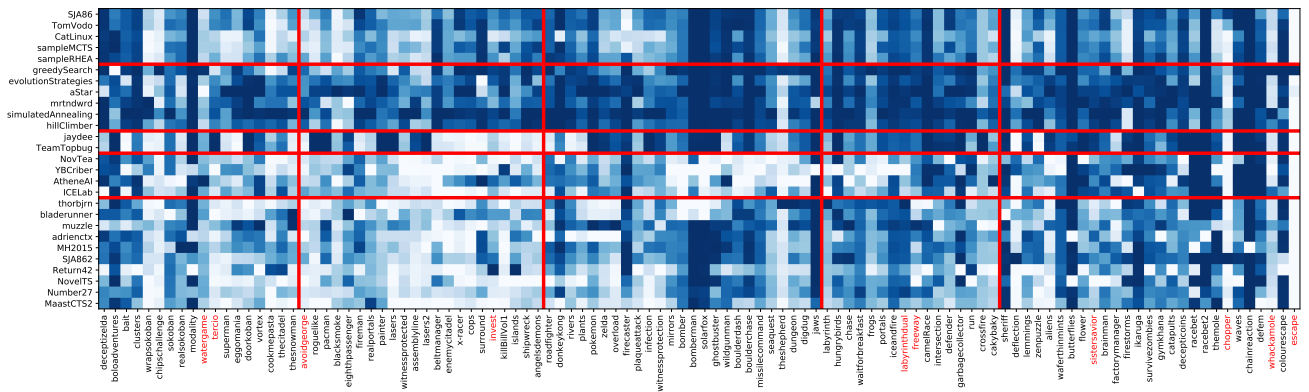
## Algorithm Performance

As a basis for further analysis, we compute both the average win-rate and score for each game-agent pair, with the results visualized in Figure 1. Looking at the win-rate in Figure 1a we can already see that there are some games where nearly all agents either win or lose, in which case the score seen in Figure 1b would be the deciding value. Conceptually, it seems that games which offer a large spread of performance values would be best at discriminating between good and bad agents in a competition, but we can also see in Figure 1 that not all games are necessarily won by the same agents.

We investigated this further by performing a principal component analysis, where the win-rate and score of the games are the data points, and the initial dimensions are the games. We found that the first ten principal components account for 80.1% of the variance. We also checked which agents did well along which axes, the results of which can be seen in Table 1. The fact that different agents perform well along different dimensions reinforces the fact that it matters which subset of games is picked for a competition. By choosing games aligned with any of these principle dimensions, one could design a competition that would almost certainly be won by the top performer for that component.



(a) The average win-rate of each agent for each game.



(b) The average normalized score of each agent for each game.

Figure 1: The average performance of each agent for each game. Figure 1a shows the average win-rate, while Figure 1b shows the average score (normalised based on the highest score achieved by any agent for each game). The games and agents are sorted based using a hierarchical clustering algorithm.

Rank	D1 Agents	D2 Agents	D3 Agents	D4 Agents	D5 Agents	D6 Agents	D7 Agents	D8 Agents	D9 Agents	D10 Agents
#1	MaastCTS2	AtheneAI	NovelTS	bladerunner	TomVodo	adrienctx	MaastCTS2	MaastCTS2	aStar	MaastCTS2
#2	thorbjrn	YBCriber	aStar	Return42	sampleMCTS	TeamTopbug	YBCriber	ICELab	adrienctx	Number27
#3	NovTea	NovTea	TeamTopbug	TomVodo	SJA86	muzzle	AtheneAI	AtheneAI	muzzle	NovTea
#4	Return42	MaastCTS2	jaydee	sampleMCTS	CatLinux	MaastCTS2	NovTea	thorbjrn	NovTea	adrienctx
#5	AtheneAI	ICELab	muzzle	muzzle	Number27	MH2015	Return42	Number27	bladerunner	muzzle

Table 1: Top five performing agents for each dimension using principal component analysis.

## Correlation Analysis

In this section, we analyze the correlations between games in terms of agent performance, as well as between using win-rate and score as performance measures, as games which have similar performance patterns should have similar problem characteristics. The resulting correlation matrices are then used for clustering, and these clusters are analyzed for meaningful similarities between games.

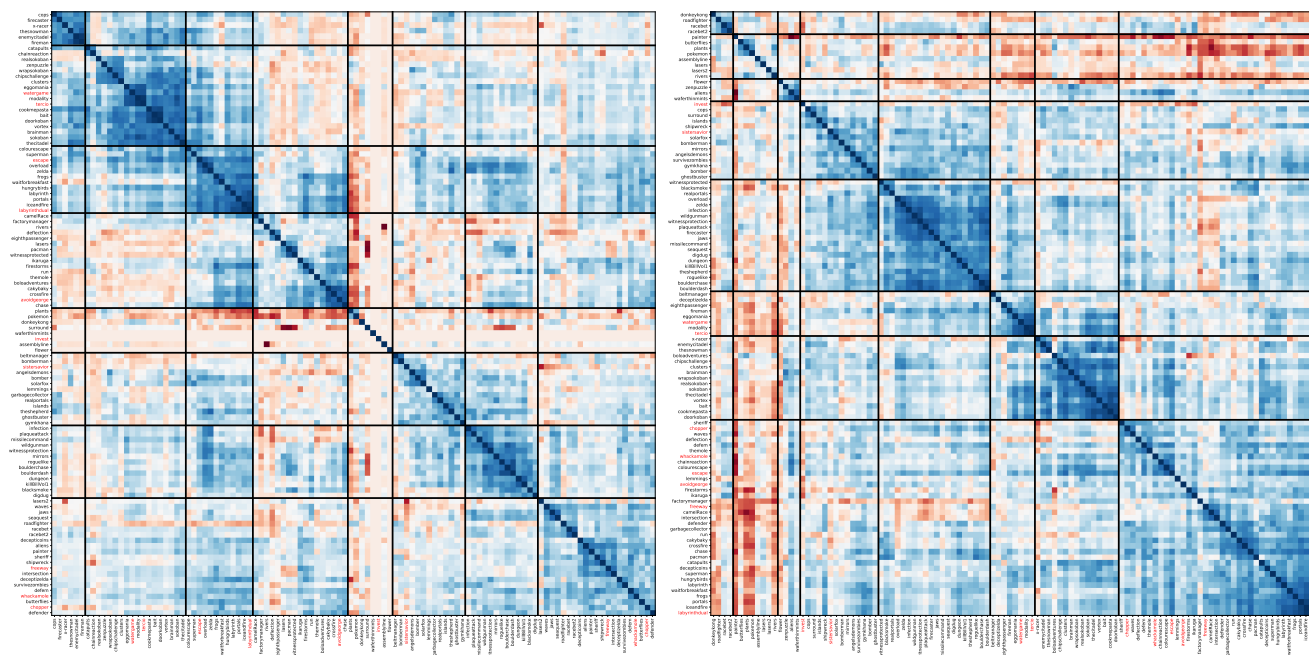
### Game / game correlation

In the video game industry, similar video games are usually grouped under a specific category that is defined by common gameplay characteristics, referred to as a game genre.

Games in the GVGAI framework are mostly ports of known video games, meaning that we can often find genre relations between them. However, attempting to group games by their genres does not necessarily indicate that similar problem-solving capabilities are required to solve them. We, therefore, took a more formal and robust approach for identifying correlations between games based on agent performance.

For this analysis, we calculated the correlation matrix between all 108 games in our sample using either the agents' win-rates or scores. Figure 2 shows these correlation matrices where blue means high correlation, red means high anti-correlation, and white means no correlation. To simplify the task of analyzing such a large matrix, we clustered





(a) Win-rate correlation matrix.

(b) Score correlation matrix.

Figure 2: The correlation matrix between every game in the framework. Figure 2a is based on the agents’ win-rates, while Figure 2b is based on the agents’ scores. The games are sorted based on the result of a hierarchical clustering algorithm.

their values using a hierarchical clustering algorithm and selected clusters that minimize the variance between the games within each cluster. The different clusters are represented by the black vertical or horizontal lines and are ordered (and subsequently referred to) in terms of their location from the left/top of the matrix.

Figure 2a shows the correlation matrix using the agents’ win-rates. Using this matrix, we can see that games in the fifth cluster have a low anti-correlation to the rest of the games in the framework. These games are characterized by either being very hard to beat (plants) or not having a winning condition (invest). By analyzing the clusters row by row, we can see that the win-rates of most games are not highly correlated except for the first three clusters. Most of the games within these first three clusters appear to be puzzle games (zenpuzzle, sokoban, cookmepasta are some examples). These types of games are typically characterized by the need for long-term planning to solve them, which likely causes their win-rates to be highly similar.

Figure 2b shows the correlation matrix using the agents’ scores. By looking closely, we can see that the score distribution between most of the games are similar (the matrix is mostly blue). This was not surprising as we know that most of the games in the framework are designed to have a score distribution that reflects the progress of the agents in the game (good states have high scores, while bad states have low scores). The only exception to this is the first three clusters, which are highly anti-correlated with every game in the framework except for those within its cluster. These games appear to be characterized by a delayed score distri-

bution (score only received near the end of the game) which makes them very different from the other games that provide rewards for incremental steps closer to the solution.

### Win-rate / score correlation

Since many GVGAI games are designed so that the score heavily indicates progression towards the win condition, most of the 108 games within our sample had very high correlations between agent win-rates and score. In fact, 13 of the games had perfect correlation values of one (frogs, pokemon, racebet2, roadfighter, run, waitforbreakfast, wattergame, x-racer, modality, portals, racebet, tercio and witnessprotected). However, some of the games had a very low or even negative correlation, with the ten lowest correlation games shown in Table 2. Note that some games such as flower and invest always result in the agent either losing or winning regardless of the actions they perform, meaning that these games do not provide any correlation measure.

From these results we can see that Painter has a very high negative correlation between win-rate and score, far more so than any other game. The likely reason for this is that the objective of Painter is to change the color of all of the tiles in the game to the same color, in as few steps as possible. Each time the color of a tile is changed the agent receives a score reward, so solutions that win the game quickly will often have a lower score. While this is a rather counter-intuitive idea, several other GVGAI games are also known to have this property within their design. This strong negative correlation between win-rate and score appears to be indicative of a particular type of deception in the game, the greed trap.

Game name	correlation coefficient
painter	-0.90488026
rivers	-0.62030330
surround	-0.49545454
lemmings	-0.37996316
chainreaction	-0.35905795
donkeykong	-0.11941969
lasers2	-0.04154697
boloadventures	-0.02221504
beltmanager	0.01585323
deflection	0.05087994

Table 2: Games with the lowest correlation between win-rate and score.

This deceptive design element exploits the fact that most agents assume the reward structure for a game leads towards a goal state, by creating levels where this is not strictly the case (Anderson et al. 2018).

While our presented correlation matrices could be used to roughly identify a collection of games with decent discriminatory performance by selecting a game from each cluster, this approach has several limitations. Not only is it difficult to tell which games in each cluster would provide the most information, but neither the fact that certain agent’s performance on the same game can vary dramatically between attempts, nor that two distinct performance measures are available, are taken into account. However, these correlation results can certainly be useful in other areas, such as for allowing game designers to understand which games are similar in terms of agent performance. Identifying which games present unique performance distributions could help in designing additional games that fit entirely new or under-represented clusters. Accomplishing this would increase the overall discrimination potential of our total game set, and thus also increase the total amount of information that could be achieved from a subset of games (i.e. allows our proposed information-theoretic measure to be even more effective).

## Information Gain Analysis

In this section, we analyze the information provided by each of our 108 sample games. Information here is used in the sense of Shannon Information Theory (Cover and Thomas 2006), and the information gain of a game is the average reduction in uncertainty regarding what algorithm we are testing, given the score and/or win-rate performance of that algorithm. This information gain measure can then be used to identify a benchmark set of games that provides us with the maximum information about our agents.

While it is possible to compute information gain on discretized data by first binning the mean performances of the different agents, this is problematic for two reasons. First, as long as all the agents’ results are at least somewhat different, it would be theoretically possible to obtain all information from just a single game. This situation would make calculating the information gain highly redundant, as nearly all of the games would give us the same maximal amount of information. Second, this approach disregards any noise

within the measuring process. As an example, if we assume that the average results for two agents are .49 and .50 when playing a specific game, then a discretized information gain analysis would give these as two separate outcomes (assuming we binned to the nearest .01 value). This approach does not take into account the fact that repeated measurements would likely produce slightly different results, varied by some noise. Consequently, a game that gives us average results for two agents of .1 and .9, rather than our previous example of .49 and .50, would be much better suited to tell two agents apart, as the scores are likely to be significantly different even when taking noise into account. In essence, games with agent results that are furthest apart and with the lowest noise, provide the most information.

The following information gain formalism is an attempt to accurately measure this difference by modelling the noise within the agents’ performances as a Gaussian distribution. This approach calculates the information gain for a specific game  $g$ . Let us first define a few terms:

- $\mathcal{A}$ : The set of all algorithms,  $a$
- $a_1$ : A specific algorithm, having an average performance of  $\mu_1$ , with a variance of  $\sigma_1$ , for the game in question.

This allows us to approximate the conditional probability  $p(a_2|a_1)$ . This probability expresses how likely it is that we are observing the result of algorithm  $a_2$ , if we are in fact observing the average performance for algorithm  $a_1$ . In other words, how well does  $a_2$  work as an explanation for what we see from  $a_1$ .

Equation 1 approximates this probability. It assumes that observations of performance are normally distributed, parameterized by the means and variances from their actual results. The upper part of the equation is the probability density function for a normal distribution based on  $a_2$ , computing how likely a result equal to the mean of  $a_1$  is. The denominator is a normalization sum over all possible algorithms, ensuring that the overall probabilities sum to one.

$$p(a_2|a_1) \approx \frac{\exp\left(-\frac{(\mu_1 - \mu_2)^2}{2(\sigma_2 + \sigma_1)^2}\right)}{\sqrt{2\pi(\sigma_2 + \sigma_1)^2}} \sum_{a \in \mathcal{A}} \left( \frac{\exp\left(-\frac{(\mu_1 - \mu_a)^2}{2(\sigma_a + \sigma_1)^2}\right)}{\sqrt{2\pi(\sigma_a + \sigma_1)^2}} \right) \quad (1)$$

Computing the probabilities for all pairwise combinations of algorithms allows us to define a confusion matrix  $C$  between  $n$  different algorithms as:

$$C = \begin{pmatrix} p(a_1|a_1) & p(a_2|a_1) & \cdots & p(a_n|a_1) \\ p(a_1|a_2) & p(a_2|a_2) & \cdots & p(a_n|a_2) \\ \vdots & \vdots & \ddots & \vdots \\ p(a_1|a_n) & p(a_2|a_n) & \cdots & p(a_n|a_n) \end{pmatrix} \quad (2)$$

Each row of the matrix sums to 1, and each entry in the first row indicates our best guess for the actual algorithm given that we observed the mean of algorithm  $a_1$ . The matrix of conditional probabilities can then be seen as an error matrix for a channel defined by using the game in question as a measurement device. This allows us to compute the mutual information for this channel, under the assumption that

the input distribution is an equal distribution. This value is equivalent to the amount of information we get about what algorithm is used from observing the average performance result. Formally, we can define this as the mutual information between your belief distribution  $\hat{A}$  and the distribution  $A$  of the actual algorithm  $a \in \mathcal{A}$ , expressed in Equation 3.

$$\begin{aligned} I(\hat{A}; A) &= H(\hat{A}) - H(\hat{A}|A) \\ &= \log_2(|\hat{A}|) - \sum_{a \in \mathcal{A}} p(a) \sum_{\hat{a} \in \mathcal{A}} -p(\hat{a}|a) \log_2 p(\hat{a}|a) \end{aligned} \quad (3)$$

The a priori distribution of our beliefs  $\hat{A}$ , is an equal distribution. If we observe the average performance of the algorithm  $a$  we get a distribution of  $\hat{A}|a$ , as defined by the confusion matrix. The average information gain of observing these results is the average difference in the entropy before observation  $H(\hat{A})$  and after observation,  $H(\hat{A}|A)$ . The equal distribution reduces to the log of the states, so we only need to compute the conditional entropy. A higher value here is more desirable, as the best games should provide us with the most information.

### Information gain for multiple games

The previous formalism allows us to quantify how much information a single specific game can provide us with about what algorithm is being used, but the information gained from looking at two games is always less than or equal to the sum of the information gain from both games individually. To address this, we can directly compute a confusion matrix for a pair, or any higher number, of games  $g \in G$  by extending the definition of the conditional probability to that presented in Equation 4.

$$p(a_2|a_1) \approx \frac{\exp\left(-\sum_{g \in G} \left(\frac{(\mu_{1,g} - \mu_{2,g})^2}{2(\sigma_{2,g} + \sigma_{1,g})^2}\right)\right)}{\prod_{g \in G} \left(\sqrt{2\pi(\sigma_{2,g} + \sigma_{1,g})^2}\right)} \sum_{a \in \mathcal{A}} \left( \frac{\exp\left(-\sum_{g \in G} \left(\frac{(\mu_{1,g} - \mu_{a,g})^2}{2(\sigma_{a,g} + \sigma_{1,g})^2}\right)\right)}{\prod_{g \in G} \left(\sqrt{2\pi(\sigma_{a,g} + \sigma_{1,g})^2}\right)} \right) \quad (4)$$

Using this new conditional probability definition allows us to compute the information gain for any subset of games, by just picking a suitable set  $G$ . The mutual information for the resulting confusion matrix is computed as usual. This means that the theoretical maximum information gain that any set of games could give is equal to  $(\log_2(|\mathcal{A}|))$ . In general, those games that offer different kind of information lose less information due to redundancy.

### Combine win-rate and score together

Using the previous equations, we can calculate the information gain for a particular game or set of games, using either the win-rate or score as the measure of performance. However, it is also possible to calculate the total information gain based on both win-rate and score combined. To do this, we treat each of these cases as a separate game (i.e., for a particular game  $g_i$  there are two variants, one where the win-rate is

used as the measure of performance  $g_{i,w}$  and one where the score is used  $g_{i,s}$ ). Since the distance is scaled by the variance, both win-rate and score can be translated in the same way as information. We can then use Equation 4 to calculate the total combined information gain of the game  $g_i$  by setting  $G = [g_{i,w}, g_{i,s}]$ . This means we can create a single confusion matrix for each game that encompasses both the win-rates and scores of all agents. The first six columns of Table 3 show the 10 games with the highest information gain when using either the win-rate, score, or both of these combined as the measure of performance. In general, this approach allows for the combination of any scalar values expressed by the game, and can, therefore, be applied to a range of different gaming benchmarks, even those where games have entirely different performance measures.

Note, though, that the approximation used here operates under the assumptions that the performance measures are distributed independently. This is true for combining the same performance measure across different games, but not necessarily true for different performance measures, such as win-rate and score, for the same game. A more faithful, but also more complex, approximation could be achieved by using the Mahalanobis distance (Mahalanobis 1936) instead of the sum of variances.

### Top ten games (of 2018)

By initially selecting the game that provides the largest information gain (based on both win-rate and score combined) and then recursively selecting the game that adds the most information to the already selected games, we can create a set of 10 games that provide the most information possible. The rightmost two columns of Table 3 provide the 10 games that were chosen for this set in the order they were selected, along with the total cumulative information gain of the set after each game was added. These 10 games are also highlighted in red in Figures 1 and 2.

### Discussion

The theoretical maximum information gain that any set of games could give is roughly 4.75 ( $\log_2(27)$ ), so we can see from these results that after selecting only 3 or 4 games we can already get the majority of information about which agent is playing. It is worth noting that this set of games is not simply the ten games that individually provide the most information, as some of these games likely provide the same ‘‘kind’’ of information. For example, the game intersection had the third highest information gain when looking at each game individually but was not selected for our top 10 games set. This is likely because it provides the same information as one of the previously selected games. By looking at how this game is played and our correlation matrices in Figure 2, it would appear that this game is very close to that of the game freeway and would likely give similar information.

We can also compare the information gain provided by using just the win-rate or score for certain games, versus the combined information gain from using both. When looking at each of these performance measures separately it appears that Invest has the highest information gain, which is likely due to the large variation in possible scores that agents could

Game Name (win-rate)	Information gain	Game Name (score)	Information gain	Game Name (combined)	Information gain	Game Name (top 10)	Information gain (cumulative)
freeway	1.17484168	invest	1.62405816	freeway	1.89430152	freeway	1.89430152
labyrinth	1.10088062	intersection	1.13955416	invest	1.62405816	invest	3.08236771
tercio	1.10018133	freeway	1.13619392	intersection	1.59362941	labyrinthdual	3.81992620
labyrinthdual	1.08531707	tercio	1.10018133	chopper	1.48524965	tercio	4.22563462
iceandfire	1.07275305	watergame	0.89206793	tercio	1.44693431	sistersavior	4.40856274
chopper	1.06542656	cops	0.88658183	labyrinthdual	1.42090667	avoidgeorge	4.54036694
doorkoban	0.98911214	flower	0.86746818	iceandfire	1.32455879	escape	4.60252506
hungrybirds	0.91886839	waitforbreakfast	0.80128373	hungrybirds	1.32100004	whackamole	4.64444512
watergame	0.89206793	labyrinth	0.78021437	waitforbreakfast	1.28983481	chopper	4.67138328
escape	0.87721725	realportals	0.73246317	doorkoban	1.28593860	watergame	4.68457480

Table 3: The games with the highest information gain (using win-rate, score or both combined as measure of performance), as well as the top 10 games which collectively provide the highest information gain.

achieve. However, as agents will always lose this game, either by spending too much money or the time limit expiring, the win-rate provides no information gain at all. Freeway, on the other hand, has a high information gain when using either win-rate or score, allowing it to have a combined information gain that is higher than Invest. It is worth reiterating that the combined information gain for a game is not simply the sum of its individual parameters, as some information may be shared between the different performance measures (calculation for combined information gain is subadditive).

## Conclusions and Future Work

In this paper, we have proposed an information-theoretic method for selecting which problems to test a given algorithm on. This is particularly useful for the many cases where it is computationally infeasible to test a new algorithm on all benchmark problems. Our method is generally applicable to any situation where algorithms need to be tested on many problems, and is especially useful when the problems are noisy and/or have multiple performance metrics.

As part of developing this method we performed an in-depth analysis into the discriminatory capabilities of the games in the GVGAI framework, as well as the correlations between games in terms of agent performance. Our correlation analysis shows that there are substantial variations in agent performance between different GVGAI games, and the resulting correlation matrices can be used to cluster certain games together. Developing new GVGAI games that do not fit within these identified clusters would present an entirely new challenge for the current selection of agents, making them highly desirable. Games that have different discriminatory profiles from those that already exist would likely be more useful for investigating agent performance than those with similar profiles to previous games.

Our proposed information theory-based method provides a more principled approach to finding discriminatory games. We extend the notion of information gain to handle noisy feedback and to combine two feedback signals (win-rate and score). We also show how this measure can be applied recursively to find a small set of games that give us almost as much information about an agent as the full set of games would have. Analyzing this set of games reveals that sev-

eral of them have “deceptive” qualities, where the score is not strongly correlated with win-rate. Future work could involve expanding the evaluation criteria to include additional data from agent playthroughs, such as the time required to solve a level or the number of moves used, which may help us to better differentiate between agents.

This research will hopefully allow future developers and researchers to accurately compare the performance of their new agent against the current set of evaluated agents without the need for exhaustive testing on the full GVGAI corpus. New agents can be tested on a set of exploratory experiments to help gauge how well the agent may perform on more detailed experiments that consider the entire GVGAI game set. This will be especially important in the future as more and more games are added to the GVGAI game library, resulting in significantly increased benchmark evaluation times. The proposed approach can also be used to evaluate games that are selected for future GVGAI competitions, to ensure that they present a diverse range of problems.

One thing to note here is that the specific results of our analysis are based on the combined game-agent ecosystem. Having a different set of agents could mean different games, that might previously have been too hard, would suddenly be more discriminatory. Similarly, adding additional games can affect which games provide us with redundant information. Because of this, while the specific games identified here are interesting today, they might very well change in the future. However, we believe the more important contribution presented in this paper is our proposed methodology that was used to select these games. Furthermore, this method, while used here on GVGAI games, could also easily be applied to other sets of problems and algorithms. Our approach can be generalized from just win-rates and scores to include any number of different outcome measures from other domains. This could even include problems outside of the traditional game space, as long as it is possible to obtain the mean and variance of each algorithm’s performance. Some obvious future application would be to analyze the performance of multiple deep reinforcement learning algorithms on the Atari games in the Arcade Learning Environment (ALE) framework, and supervised learning algorithms tested on datasets associated with Kaggle competitions.

## Acknowledgments

Damien Anderson is funded by the Carnegie Trust for the Universities of Scotland as a PhD Scholar. Christoph Salge is funded by the EU Horizon 2020 programme under the Marie Skłodowska-Curie grant 705643. Ahmed Khalifa acknowledges the financial support from NSF grant (Award number 1717324 - "RI: Small: General Intelligence through Algorithm Invention and Selection.").

## References

- [Anderson et al. 2018] Anderson, D.; Stephenson, M.; Togelius, J.; Salge, C.; Levine, J.; and Renz, J. 2018. Deceptive games. In *21st International Conference on the Applications of Evolutionary Computation*, 1–16.
- [Bellemare et al. 2013] Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.
- [Bontrager et al. 2016] Bontrager, P.; Khalifa, A.; Mendes, A.; and Togelius, J. 2016. Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 122–128.
- [Brockman et al. 2016] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *CoRR* abs/1606.01540.
- [Carpenter 2011] Carpenter, J. 2011. May the best analyst win. *Science (New York, N.Y.)* 331:698–699.
- [Cover and Thomas 2006] Cover, T. M., and Thomas, J. A. 2006. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. New York, NY, USA: Wiley-Interscience.
- [Dheeru and Karra Taniskidou 2017] Dheeru, D., and Karra Taniskidou, E. 2017. UCI machine learning repository.
- [Ebner et al. 2013] Ebner, M.; Levine, J.; Lucas, S.; Schaul, T.; Thompson, T.; and Togelius, J. 2013. Towards a video game description language. *Artificial and Computational Intelligence in Games* 1–17.
- [Gaina, Lucas, and Perez-Liebana 2017] Gaina, R. D.; Lucas, S. M.; and Perez-Liebana, D. 2017. Rolling horizon evolution enhancements in general video game playing. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*, 88–95.
- [Gaina, Prez-Libana, and Lucas 2016] Gaina, R. D.; Prez-Libana, D.; and Lucas, S. M. 2016. General video game for 2 players: Framework and competition. In *2016 8th Computer Science and Electronic Engineering (CEECE)*, 186–191.
- [Genesereth, Love, and Pell 2005] Genesereth, M.; Love, N.; and Pell, B. 2005. General game playing: Overview of the AAI competition. *AI Magazine* 26(2):62–72.
- [Levine et al. 2013] Levine, J.; Congdon, C.; Ebner, M.; Kendall, G.; Lucas, S.; Miikkulainen, R.; Schaul, T.; and Thompson, T. 2013. General video game playing. *Artificial and Computational Intelligence in Games* 6:77–83.
- [Mahalanobis 1936] Mahalanobis, P. C. 1936. On the generalized distance in statistics. *National Institute of Science of India* 2(1):49–55.
- [Martinez-Plumed and Hernández-Orallo 2016] Martinez-Plumed, F., and Hernández-Orallo, J. 2016. Ai results for the atari 2600 games: difficulty and discrimination using irt. *EGPAI, Evaluating General-Purpose Artificial Intelligence* 33.
- [Mendes, Togelius, and Nealen 2016] Mendes, A.; Togelius, J.; and Nealen, A. 2016. Hyper-heuristic general video game playing. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 1–8.
- [Nelson 2016] Nelson, M. J. 2016. Investigating vanilla mcts scaling on the gvg-ai game corpus. *2016 IEEE Conference on Computational Intelligence and Games (CIG)* 1–7.
- [Perez-Liebana et al. 2016a] Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Lucas, S. M.; and Schaul, T. 2016a. General video game ai: Competition, challenges and opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*, 4335–4337.
- [Perez-Liebana et al. 2016b] Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S. M.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2016b. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games* 8(3):229–243.
- [Pérez-Liéñana et al. 2016c] Pérez-Liéñana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; and Lucas, S. M. 2016c. Analyzing the robustness of general video game playing agents. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 1–8.
- [Perez-Liebana et al. 2018a] Perez-Liebana, D.; Liu, J.; Khalifa, A.; Gaina, R. D.; Togelius, J.; and Lucas, S. M. 2018a. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms. *arXiv preprint arXiv:1802.10363*.
- [Pérez-Liéñana et al. 2018b] Pérez-Liéñana, D.; Liu, J.; Khalifa, A.; Gaina, R. D.; Togelius, J.; and Lucas, S. M. 2018b. General video game AI: a multi-track framework for evaluating agents, games and content generation algorithms. *CoRR* abs/1802.10363.
- [Soemers et al. 2016] Soemers, D. J.; Sironi, C. F.; Schuster, T.; and Winands, M. H. 2016. Enhancements for real-time monte-carlo tree search in general video game playing. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 1–8.
- [Weinstein and Littman 2012] Weinstein, A., and Littman, M. L. 2012. Bandit-based planning and learning in continuous-action markov decision processes. In *ICAPS*.

## Appendix C

# Superstition in the Network

# “Superstition” in the Network: Deep Reinforcement Learning Plays Deceptive Games

Philip Bontrager,<sup>1</sup> Ahmed Khalifa,<sup>1</sup> Damien Anderson,<sup>2</sup> Matthew Stephenson,<sup>3</sup>  
Christoph Salge,<sup>4</sup> Julian Togelius<sup>1</sup>

<sup>1</sup>New York University, <sup>2</sup>University of Strathclyde, <sup>3</sup>Maastricht University, <sup>4</sup>University of Hertfordshire  
{philipjb, ahmed.khalifa}@nyu.edu, damien.anderson@strath.ac.uk, matthew.stephenson@maastrichtuniversity.nl,  
ChristophSalge@gmail.com, julian@togelius.com

## Abstract

Deep reinforcement learning has learned to play many games well, but failed on others. To better characterize the modes and reasons of failure of deep reinforcement learners, we test the widely used Asynchronous Actor-Critic (A2C) algorithm on four deceptive games, which are specially designed to provide challenges to game-playing agents. These games are implemented in the General Video Game AI framework, which allows us to compare the behavior of reinforcement learning-based agents with planning agents based on tree search. We find that several of these games reliably deceive deep reinforcement learners, and that the resulting behavior highlights the shortcomings of the learning algorithm. The particular ways in which agents fail differ from how planning-based agents fail, further illuminating the character of these algorithms. We propose an initial typology of deceptions which could help us better understand pitfalls and failure modes of (deep) reinforcement learning.

## Introduction

In reinforcement learning (RL) (Sutton and Barto 1998) an agent is tasked with learning a policy that maximizes expected reward based only on its interactions with the environment. In general, there is no guarantee that any such procedure will lead to an optimal policy; while convergence proofs exist, they only apply to a tiny and rather uninteresting class of environments. Reinforcement learning still performs well for a wide range of scenarios not covered by those convergence proofs. However, while recent successes in game-playing with deep reinforcement learning (Justesen et al. 2017) have led to a high degree of confidence in the deep RL approach, there are still scenarios or games where deep RL fails. Some oft-mentioned reasons why RL algorithms fail are partial observability and long time spans between actions and rewards. But are there other causes?

In this paper, we want to address these questions by looking at games that are designed to be deliberately deceptive. Deceptive games are defined as those where the reward structure is designed to lead away from an optimal policy. For example, games where learning to take the action which produces early rewards curtails further exploration. Deception does not include outright lying (or presenting false in-

formation). More generally speaking, deception is the exploitation of cognitive biases. Better and faster AIs have to make some assumptions to improve their performance or generalize over their observation (as per the no free lunch theorem, an algorithm needs to be tailored to a class of problems in order to improve performance on those problems (Wolpert and Macready 1997)). These assumptions in turn make them susceptible to deceptions that subvert these very assumptions. For example, evolutionary optimization approaches assume locality, i.e., that solutions that are close in genome space have a similar fitness - but if very bad solutions surround a very good solution, then an evolutionary algorithm would be less likely to find it than random search.

While we are specifically looking at digital games here, the ideas we discuss are related to the question of optimization and decision making in a broader context. Many real-world problems involve some form of deception; for example, while eating sugar brings momentary satisfaction, a long-term policy of eating as much sugar as possible is not optimal in terms of health outcomes.

In a recent paper, a handful of *deceptive games* were proposed, and the performance of a number of planning algorithms were tested on them (Anderson et al. 2018). It was shown that many otherwise competent game-playing agents succumbed to these deceptions and that different types of deceptions affected different kinds of planning algorithms; for example, agents that build up a model of the effects of in-game objects are vulnerable to deceptions based on changing those effects. In this paper, we want to see how well deep reinforcement learning performs on these games. This approach aims to gain a better understanding of the vulnerabilities of deep reinforcement learning.

## Background

Reinforcement learning algorithms learn through interacting with an environment and receiving rewards (Sutton and Barto 1998). There are different types of algorithms that fit this bill. A core distinction between the types are between ontogenetic algorithms, that learn within episodes from the reward that they encounter, and phylogenetic algorithms, that learn between episodes based on the aggregate reward at the end of each episode (Togelius et al. 2009).

For some time, reinforcement learning had few clear successes. However, in the last five years, the combination of

ontogenetic RL algorithms with deep neural networks have seen significant successes, in particular in playing video games (Justesen et al. 2017) such as simple 2D arcade games (Mnih et al. 2015) to more advanced games like Dota 2 and Starcraft (OpenAI 2018; Vinyals et al. 2019). This combination, generally referred to as deep reinforcement learning, is the focus of much research.

The deceptive games presented in this paper were developed for the GVGAI (General Video Game Artificial Intelligence (Perez-Liebana et al. 2016)) framework. The GVGAI framework itself is based on VGDL (Video Game Description Language (Ebner et al. 2013; Schaul 2013)) which is a language that was developed to express a range of arcade games, like Sokoban and Space Invaders. VGDL was developed to encourage research into more general video game playing (Levine et al. 2013) by providing a language and an interface to a range of arcade games. Currently the GVGAI corpus has over 150 games. The deceptive games discussed in this paper are fully compatible with the framework.

## Methods

To empirically test the effectiveness of the deception in every game, we train a reinforcement learning algorithm and run six planning algorithms on each game. The benefit of working in GVGAI is that we are able to evaluate the same game implementations with algorithms that require an available forward model and with learning agents. GVGAI has a Java interface for planning agents as well as an OpenAI Gym interface for learning agents (Perez-Liebana et al. 2016; Rodriguez Torrado et al. 2018; Brockman et al. 2016).

All algorithms were evaluated on each game 150 times. The agent’s scores are evaluated along with play through videos. The qualitative analyses of the videos provide key insights into the causes behind certain scores and into what an agent is actually learning. The quantitative and qualitative results are then used for the final analysis.

## Reinforcement Learning

To test if these games are capable of deceiving an agent trained via reinforcement learning, we use Advantage Actor-Critic (A2C) to learn to play the games (Mnih et al. 2016). A2C is a good benchmark algorithm and has been shown to be capable of playing GVGAI games with some success (Rodriguez Torrado et al. 2018; Justesen et al. 2018). A2C is a model-free, extrinsically driven algorithm that allows for examining the effects of different reward patterns. A2C is also relevant due to the popularity of model-free agents.

Due to the arcade nature of GVGAI games, we train on pixels with the same setup developed for the Atari Learning Environment framework (Bellemare et al. 2013). The atari configuration has been shown to work well for GVGAI and allows a consistent baseline with which to compare all the games (Rodriguez Torrado et al. 2018). Instead of tuning the algorithms for the games, we designed the games for the algorithms. We use the OpenAI Baselines implementation of A2C (Dhariwal et al. 2017). The neural network architecture is the same as the original designed by Mnih et al. (Mnih et al. 2016). The hyper-parameters are the default from the

original paper as implemented by OpenAI: step size of 5, no frame skipping, constant learning rate of 0.007, RMS, and we used 12 workers.

For each environment, we trained five different A2C agents to play, each starting from random seeds. In initial testing, we tried training for twenty million frames, and we found that the agents converged very quickly, normally within two million frames of training. We therefore standardized the experiments to all train for five million frames. One stochastic environment, WaferThinMints, did not converge and might have benefited from more training time.

## Planning Agents

For comparison with previous work and better insight into the universality of the deceptive problems posed here, we compare our results to planning algorithms. What we mean by planning agents are algorithms that utilize a forward model to search for an ideal game state. In the GVGAI planning track, each algorithm is provided with the current state and a forward model and it has to return the next action in a small time frame (40 milliseconds). This time frame doesn’t give the algorithm enough time to find the best action. This limitation forces traditional planning algorithms to be somewhat greedy which, for most of these games, is a trap.

In this paper, we are using six different planning algorithms. Three of them (aStar, greedySearch, and sampleMCTS) are directly from the GVGAI framework, while the rest (NovelTS, Return42, and YBCriber) are collected from the previous GVGAI competitions. Two of these algorithms, Return42, and YBCriber, are hybrid algorithms. They use one approach for deterministic games, such as A\* or Iterative Width, and a different one for stochastic games, such as random walk or MCTS. Both algorithms use hand designed heuristics to judge game states. These hybrid algorithms also use online learning to bypass the small time per frame. The online learning agents try to understand the game rules, from the forward model during each time step, and then use that knowledge to improve the search algorithm.

## Deceptive Games

In our previous work, a suite of deceptive games was created in order to take a look at the effects that these deceptive mechanics would have on agents (Anderson et al. 2018). These deceptive games were designed in order to deceive different types of agents in different ways.

From a game design perspective, the category of deceptive games partially overlaps with “abusive games”, as defined by Wilson and Sicart (Wilson and Sicart 2010). In particular, the abuse modalities of “unfair design” can be said to apply to some of the games we describe below. Wilson and Sicart note that these modalities are present in many commercial games, even successful and beloved games, especially those from the 8-bit era.

This section describes some of these games in detail, and defines optimal play for an agent playing each game. We focus on four key categories of deception that these games exploit. We believe these categories represent general problems that learning agents face and these simple games allow



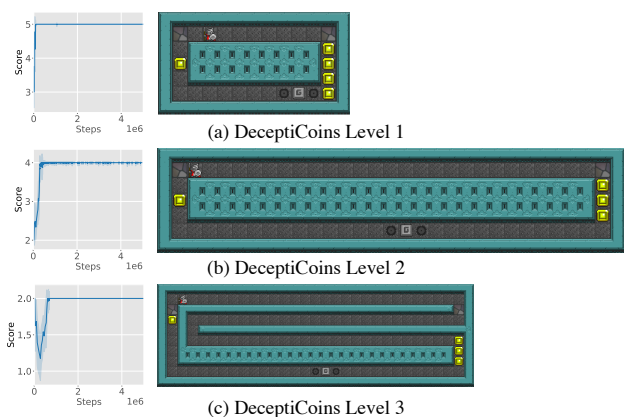


Figure 1: DeceptiCoins Levels

us to shine a spotlight on weaknesses that model-free, deep reinforcement learning agents still face. For a more comprehensive list of types of deceptions and deceptive games see Deceptive Games (Anderson et al. 2018).

The following four different categories of deception will be discussed further in the discussion section: Lack of Hierarchical Understanding, Subverted Generalization, Delayed Gratification, and Delayed Reward.

### DeceptiCoins (DC)

**Game** DeceptiCoins, Figure 1, offers an agent two paths which both lead to the win condition. The first path presents immediate points to the agent, in the form of gold coins. The second path contains more gold coins, but they are further away and may not be immediately visible to a short-sighted agent. Once the agent selects a path, they become trapped within their chosen path and can only continue to the bottom exit. The levels used here are increasingly larger versions of the same challenge, but remain relatively small overall.

The optimal strategy for DeceptiCoins is to select the path with the highest overall number of points. For the levels shown in Figure 1, this is achieved by taking the right side path, as it leads to the highest total score (i.e., more gold coins can be collected before completing the level).

**Goal** The game offers a simple form of deception that targets the exploration versus exploitation problem that learning algorithms face. The only way for the learning agent to discover the higher reward is for it to forgo the natural reward it discovers early on completely. By designing different sized levels, we can see how quickly the exploration space becomes too large. At the same time, an agent that correctly learns, on the short route, about coins and navigation could then see that going right is superior.

**Results** The first two levels of DeceptiCoins are very small, and the agent fairly quickly learns the optimal strategy. However, in level two the agent took several times longer to discover the optimal strategy, as expected from an agent that can only look at the rewards of individual moves. Level 3 proves to be too hard, and the agent converges on the

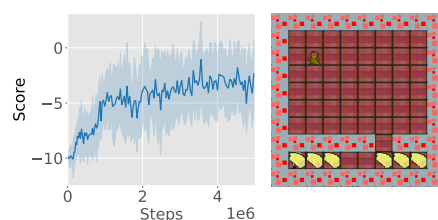


Figure 2: The first level of WaferThinMints

suboptimal strategy. By comparison, a randomly initialized agent is very likely to select the easy path, since it starts next to it, before being forced to move toward the exit.

The training curve for level 3 shows a significant drop in performance at the beginning of training. The video footage suggests that the agent learns the concept of the gold coins and is attempting to collect them all, but fails to understand that once it takes the easy coin it will become trapped in the left path. The agent will also move back and forth between the paths at the beginning of the game, trying to decide.

### WaferThinMints (Mints)

**Game** WaferThinMints is inspired by a scene in Monty Python’s *The Meaning of Life*. The game presents the agent with easily obtainable points, but if the agent collects too many it will lead to a loss condition. The idea of this game is to model a situation where a repeated action does not always lead to the same outcome or has a diminishing return over time. The levels for this game feature mints which each award a point when collected and also fill up a resource gauge on the agent. The level used is shown in figure 2. If the avatar’s resource gauge (green bar on avatar) is filled, defined in this case as nine mints, and the agent attempts to collect an additional mint, then the agent is killed and a loss condition is reached. Losing the game also causes the agent to lose 20 points. A waiter (not seen in Figure 2) moves around the board distributing mints at random. This means it is possible for an agent to get trapped while the waiter places mint on the agent’s square, forcing the agent to eat it. The agent must, therefore, try to avoid getting trapped.

The optimal strategy is to collect as many mints as possible without collecting too many, which is currently set as nine. The player should avoid mints early on and try to avoid getting trapped. Near the end of the game, the agent should then eat the remaining mints to get to 9.

**Goal** WaferThinMints is our primary example of the changing heuristic deception. The mint goes from providing a positive reward to giving a substantial negative reward with the only visual indication being a green bar on the avatar that represents how full the character is. The agent must learn that the value of the mints is dependent on that green bar. Since the bar moves with the Avatar, it cannot just memorize a fixed state in which to stop eating the mints. The mint is distributed by a chef and left around the board at random. For the agent to play optimally, it should also learn that it is not good to get full early on because it might get trapped in

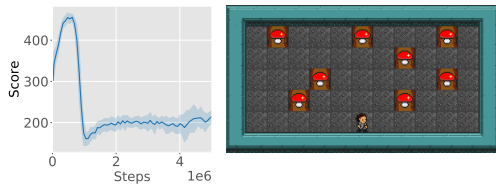


Figure 3: Flower level 1

and forced to eat another mint at some point.

**Results** As can be seen from the graph, this agent did not have enough time to converge completely. This points to the difficulty of learning in the noisy environment where even a good strategy could result in a bad reward if the agent is unlucky. This is necessary though, as in a simpler environment with a fixed mint layout, the agent would learn to memorize a path that results in a perfect score. The agent shows some improvement over time but still plays very poorly.

By observing the agent, we see that the agent uses location to solve this problem. At the beginning of the episode, the agent rushes to the room where the initial mints are placed. This is a guaranteed source of rewards. The agent will mostly stay in the room, a safe place, unless chased out by the chef's mint placement. After the initial mints, the agent attempts to avoid mints until it's trapped by them.

It is not clear whether the agent understands its fullness bar or uses the amount of mints placed in the game to assess the risk of eating more mints. The agent seems to have learned that the mints become dangerous, but it seems to use strange state and location information to help it know when to eat mints. This is related to the behavior we see in the game *Invest*. It also is incapable of reasoning about waiting until the end of the game to eat mints when it is safer to eat, an instance of the delayed gratification deception.

### Flower (Flow)

**Game** Flower is a game which rewards patient agents by offering the opportunity to collect a small number of points immediately, but which will grow larger over time the longer it is not collected. As shown in figure 3, a few seeds are available for the agent to collect, which are worth zero points. The seeds will eventually grow into full flowers and their point values grow along with them up to ten points. Once a flower is collected, another will begin to grow as soon as the agent leaves the space from which it was collected.

The optimal strategy for *Flower* is to let the flowers grow to their final stage of development before collecting them.

**Goal** In Flower, an agent is rewarded every time it collects a flower. To get maximum points the agent should collect each flower the moment it matures to 10 points. This will provide a better score than constantly collecting seedlings.

**Results** The training graph for this game shows the agent falling for the specific deception with the sudden drop-off in performance. As the agent gets better at knowing where the flowers are, the score starts to improve. Then the agent gets

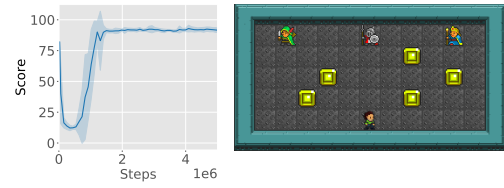


Figure 4: Invest level 1

too good at collecting the flowers, and they no longer have a chance to grow, lowering the score. Watching agent replays further confirms this, the agent finds a circuit through all the flowers and then gets better at quickly moving through this circuit. The agent perfectly falls for the deceit and has no way back unless it ignores the immediate rewards.

### Invest (Inv)

**Game** Invest is a game where agents can forgo a portion of their already accumulated reward, for the benefit of receiving a larger reward in the future. The level used is shown in figure 4. The agent begins with no points but can collect a small number of coins around the level to get some initial amount. These points can then be “spent” on certain investment options. Doing this will deduct a certain number of points from the agent's current score, acting as an immediate penalty, but will reward them with a greater number of points after some time has passed. The agent has several different options on what they can invest in, represented by the three human characters (referred to as bankers) in the top half of the level. Each banker has different rules: Green banker turns 3 into 5 after 30 ticks, Red turns 7 into 15 after 60 ticks, and Blue turns 5 into 10 after 90 ticks. The agent can decide to invest in any of these bankers by simply moving onto them, after which the chosen banker will take some of the agent's points and disappear, returning a specific number of timesteps later with the agent's reward. The agent will win the game once the time limit for the level expires.

The optimal strategy for *Invest* is defined as successfully investing with everyone as often as possible.

**Goal** Invest is a game where the agent has to intentionally seek some negative reward to get a positive reward, and then wait for a certain amount of time to get the positive reward. This delayed reward makes it very difficult for the reinforcement learning algorithm to assign credit to a specific assignment. The initial investment will only be assigned a negative reward, and the agent then has to figure out that the reward that happens later should also be assigned to this action. In this case, the reward is deterministic, and the challenge could be increased further by making the delay stochastic.

**Results** The agent learns a very particular strategy for all five instances of training. The agent first collects all the coins and then invests with the Green Banker. From there it runs to the far right corner and waits, some agents always choose the top while others choose the bottom. As soon as the Green banker returns, the agent runs back over and reinvests only to run back to its corner and wait. This at first seems like

Agent	DC 1	DC 2	DC 3	Inv	Flow	Mints
aStar	3.36	3.54	1.33	17.53	604.99	1.92
greedySearch	5.0	3.0	1.23	1.0	6.83	-5.15
sampleMCTS	2.0	2.0	1.99	3.5	392.73	5.73
NovelTS	2.1	2.0	2.0	4.8	298.51	8.75
Return42	5.0	2.0	2.0	190.12	329.73	-2.66
YBCriber	5.0	4.0	4.0	10.91	300.73	5.2
A2C	5.0	3.79	2.0	69.6	228.86	-6.21

Table 1: Average score for different games using different agents. Darker blue entries have higher positive score values for that game between all the agents, while darker red entries have higher negative score values.

puzzling behavior as a better strategy would be to sit next to the Green Banker and be able to reinvest faster and collect more points. On closer inspection, it becomes apparent that the time it takes the agent to reach the far corner correlates with the arrival of the delayed reward. It appears that the agent learned that investing in the Green Banker and then touching the far tile resulted in a large positive reward.

The size of the game board allowed the agent to embody the delay through movement and predict the arrival of the reward through how long it takes to walk across the board. It is possible that the agent would have learned to invest with the other bankers if the board was larger so the agent could have found a location associated with the delayed reward.

The training graph shows an interesting story too. The initial random agent would accidentally invest with all three bankers and get a fairly high score despite not consistently investing with anyone. The agent quickly learns to avoid the negative reward associated with the bankers and its score drops. It stops investing with the Blue Banker first, then the Red, and finally the Green. After it discovers how to predict the delayed reward for the Green Banker, it starts doing this more regularly until its performance converges.

## Comparison with planning algorithms

In this section we want to compare the results from some of the planning agents in the previous paper (Anderson et al. 2018) with the deep RL results in this paper. Table 1, shows the average score respectively for all the games using six different planning agents and the trained reinforcement learning agents. Every agent plays each game around 150 times, and the average score is recorded. These are drastically different algorithms from A2C, but they provide context for how different algorithms are affected by our deceptions.

While the planning agents perform slightly better on average, this depends highly on what exact planning algorithm we are examining. The planning algorithms have an advantage over the reinforcement learning algorithm as they have a running forward model that can predict the results of each action. On the other hand, the small time frame (40 milliseconds), for deciding the next action, doesn't give the algorithm enough time to find the best action.

In an important way, both RL and planning are facing a similar problem here. In both cases, the algorithms can only query the game environment a limited amount of times. This makes it impossible to look at all possible futures and forces

the algorithms to prioritize. While most planning agents entirely rely on the given forward model, some, such as Return42, also use online learning. These agents initially play with the forward model but will try to learn and generalize the game rules while playing. As the game progresses, they rely more and more on those learned abstractions. In general, this is an efficient and smart strategy but makes them vulnerable to deceptions where the game rules changed in the middle of the game, such as in *Wafer Thin Mints*. Here the agents might get deceived if they do not verify the result using the forward model. This is very similar to the problem that A2C encounters since the network representation is tries to generalize the states of the game.

In summary, while the best planning agents seem to be stronger than A2C, they also are subject to different forms of deceptions, dependent on how they are implemented.

## Discussion

In summary, while the A2C deep reinforcement learning (Mnih et al. 2016) approach performs somewhat well, it rarely achieves the optimal performance in our games and is vulnerable to most deceptions discussed here. In contrast, the A2C algorithm performs quite well across the board for different AI benchmarks and can be considered competitive (Arulkumaran et al. 2017; Justesen et al. 2017). It should also be noted that the fast-moving field of deep reinforcement learning has already produced numerous modifications that could potentially solve the games discussed here (Arulkumaran et al. 2017). However, instead of discussing possible modifications to overcome any particular challenge presented here, we want to take a step back and refocus back on the point of this exercise. We are interested in deceptions to gain a better understanding of the general vulnerabilities of AI approaches, and try to gain a more systematic understanding of the ways deep learning in particular, and AI, in general, might fail. With the previous games as concrete examples in mind, we now want to discuss four, non-exhaustive, categories for deception.

## Types of Deception

**Lack of Hierarchical Understanding** The DeceptiCoin games are relatively easy to solve if one thinks about them at the right level of abstractions. DeceptiCoins can be seen as a single binary decision between one path and another. Once this is clear, one can quickly evaluate the utility of choosing the correct one and pick the correct path. The deceptive element here is the fact that this is presented to the AI as an incredibly large search space, as it takes many steps to complete the overall meta-action. Humans are usually quite good at finding these higher levels of abstraction, and hence this problem might not look like much of a deception to us - but it is pretty hard for an AI. The large search space, paired with the assumptions that all actions along the path of the larger action matter, makes it very hard to explore all possible steps until a possible reward is reached. This is a similar problem to the famous problem in Montezuma's Revenge, where the AI could not reach the goal, and its random exploration did not even get close. This problem was only recently solved with forced exploration (Ecoffet et al. 2019).

Finding a good hierarchical abstraction can actually solve the problem. For example, in DeceptiCoins we can look at the path from one point to another as one action - something that has been explored in GVGAI playing agents before.

**Subverted Generalization** Waferthinmints is a game specifically designed to trick agents that generalize. Agents that simply use a forward model to plan their next step perform quite well here, as they realize that their next action will kill them. But in general, we do not have access to a forward model, so there is a need to generalize from past experience and use induction. The fact that each mint up to the 9th gives a positive reward reinforces the idea that eating a mint will be good. The 10th mint then kills you. This is not only a problem for reinforcement learning, but has been discussed in both epistemology (Hume 1739; Russell 1912) and philosophy of AI - with the consensus that induction in general does not work, and that there is not really a way to avoid this problem. The subverted generalization is also a really good example of how more advanced AIs become more vulnerable to certain deceptions. On average, generalization is a good skill to have and can make an AI much faster, up to the point where it fails.

**Delayed Reward** The big challenge in reinforcement learning is to associate what actions lead to the reward (Sutton 1992). One way to complicate this is to delay the payment of this reward, as we did in the example of invest. The player first has to incur a negative reward to invest, and then, after a certain amount of time steps gets a larger positive reward. The RL agent had two problems with Invest. First, it only ever invests with the investor with the shortest repayment time. The Red Banker would, overall, offer the best payout, but the RL agent either does not realize this relationship, or does not associate the reward correctly.

Furthermore, the RL agents also seem to be learning “superstitions”. When we examined the behaviour of the evolved RL agent, we see that the agent invests with the Green Banker and then runs to a specific spot in the level, waiting there for the reward payout. This behaviour is then repeated, the agent runs to the banker and then back to the spot to wait for its reward. We reran the training for the RL agent and saw the same behaviour, albeit with a different spot that the agent runs to. We assume that this superstition arose because the agent initially wandered off after investing in the Green Banker, and then received the reward when it was in that spot. It seems to have learned that it needs to invest in the banker - as varying this behaviour would result in no payout. But there is little pressure to move it away from its superstition of waiting for the result in a specific spot, even though this has no impact on the payout. In fact, it makes the behaviour, even with just the Green Banker sub-optimal, as it delays the time until it can invest again, as it has to run back to the green banker.

What was exciting about this behavior, was the fact that similar behavior was also observed in early reinforcement learning studies with animals (Skinner 1948). Pigeons that were regularly fed by an automatic mechanism (regardless of their behaviour) developed different superstitious behaviours, like elaborate dance and motions, which Skinner

hypothesized were assumed (by the pigeon) to causally influence the food delivery. In our game, the agent seems to develop similar superstitions.

**Delayed Gratification** There is a famous experiment (Mischel, Ebbesen, and Raskoff Zeiss 1972) about delayed gratification that confronts 4 year old children with a marshmallow, and asks them not to eat it while the experimenter leaves the room. They are told that they will get another marshmallow, if they can just hold off eating the first marshmallow now. This task proves difficult for some children, and it is also difficult for our agent. Flower is a game where the agent actually gets worse over time. This is because it initially is not very good at collecting the flowers, which allows the flowers time to mature. The optimal strategy would be to wait for the flowers to grow fully, and then go around and collect them. The agent learns the expected reward of collecting seeds early on but does not realize that this reward changes with faster collection. When it updates its expected reward based on its new speed, it forgets that it could get higher rewards when it was slower. While some of the planning algorithms perform better here, it is likely that they did not actually “understand” this problem, but are simply much worse at collecting the flowers (like the untrained RL agent). This example demonstrates that we can design a problem where the AI gets worse over time by “learning” to play.

## Conclusion

It appears that deep reinforcement learners are easily deceived. We have devised a set of games specifically to showcase different forms of deception, and tested one of the most widely used RL algorithms, Advantage Actor-Critic (A2C), on them. In all games, the reinforcement learners failed to find the optimal policy (with the exception that it found the optimal policy on one level of one game), as it evidently fell for the various traps laid in the levels.

As the games were implemented in the GVGAI framework, it was also possible for us to compare with tree search-based planning agents, including those based on MCTS. (This is very much a comparison of apples and oranges, as the planning agents have access to a forward model and direct object representation but are not given any kind of training time.) We can see that for every game, there is a planning agent which performs better than the A2C agent, but that there is in most cases also a planning agent that performs worse. It is clear that some kinds of deception affect our reinforcement learning algorithm much more severely than it affects the planning algorithms; in particular, the subverted generalization of WaferThinMints. On the other hand, it performed better than most planning algorithms given the delayed reward in Invest, even though the policy it arrived at is bizarre to a human observer and suggests a warped association between cause and effect.

We look forward to testing other kinds of algorithms on these games, including phylogenetic reinforcement learning methods such as neuroevolution. We also hope that other researchers will use these games to test the susceptibility of their agents to specific deceptions.

## References

- [Anderson et al. 2018] Anderson, D.; Stephenson, M.; Togelius, J.; Salge, C.; Levine, J.; and Renz, J. 2018. Deceptive games. In *International Conference on the Applications of Evolutionary Computation*, 376–391. Springer.
- [Arulkumaran et al. 2017] Arulkumaran, K.; Deisenroth, M. P.; Brundage, M.; and Bharath, A. A. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34(6):26–38.
- [Bellemare et al. 2013] Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.
- [Brockman et al. 2016] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.
- [Dhariwal et al. 2017] Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; and Wu, Y. 2017. Openai baselines. <https://github.com/openai/baselines>.
- [Ebner et al. 2013] Ebner, M.; Levine, J.; Lucas, S. M.; Schaul, T.; Thompson, T.; and Togelius, J. 2013. Towards a video game description language. In *Dagstuhl Follow-Ups*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- [Ecoffet et al. 2019] Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.
- [Hume 1739] Hume, D. 1739. *A Treatise of Human Nature*. Oxford University Press.
- [Justesen et al. 2017] Justesen, N.; Bontrager, P.; Togelius, J.; and Risi, S. 2017. Deep learning for video game playing. *arXiv preprint arXiv:1708.07902*.
- [Justesen et al. 2018] Justesen, N.; Torrado, R. R.; Bontrager, P.; Khalifa, A.; Togelius, J.; and Risi, S. 2018. Procedural level generation improves generality of deep reinforcement learning. *arXiv preprint arXiv:1806.10729*.
- [Levine et al. 2013] Levine, J.; Bates Congdon, C.; Ebner, M.; Kendall, G.; Lucas, S. M.; Mikkulainen, R.; Schaul, T.; and Thompson, T. 2013. General video game playing. *Artificial and Computational Intelligence in Games*.
- [Mischel, Ebbesen, and Raskoff Zeiss 1972] Mischel, W.; Ebbesen, E. B.; and Raskoff Zeiss, A. 1972. Cognitive and attentional mechanisms in delay of gratification. *Journal of personality and social psychology* 21(2):204.
- [Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.
- [Mnih et al. 2016] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.
- [OpenAI 2018] OpenAI. 2018. Openai five. <https://blog.openai.com/openai-five/>.
- [Perez-Liebana et al. 2016] Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Lucas, S. M.; and Schaul, T. 2016. General video game ai: Competition, challenges and opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*.
- [Rodriguez Torrado et al. 2018] Rodriguez Torrado, R.; Bontrager, P.; Togelius, J.; Liu, J.; and Perez-Liebana, D. 2018. Deep reinforcement learning for general video game ai. In *Computational Intelligence and Games (CIG), 2018 IEEE Conference on*. IEEE.
- [Russell 1912] Russell, B. 1912. *The Problems of Philosophy*. Williams and Norgate. chapter On Induction.
- [Schaul 2013] Schaul, T. 2013. A video game description language for model-based or interactive learning. In *IEEE Conference on Computational Intelligence and Games, CIG*.
- [Skinner 1948] Skinner, B. F. 1948. 'superstition' in the pigeon. *Journal of experimental psychology* 38(2):168.
- [Sutton and Barto 1998] Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT press.
- [Sutton 1992] Sutton, R. S. 1992. *Introduction: The Challenge of Reinforcement Learning*. Boston, MA: Springer US. 1–3.
- [Togelius et al. 2009] Togelius, J.; Schaul, T.; Wierstra, D.; Igel, C.; Gomez, F.; and Schmidhuber, J. 2009. Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz* 23(3):30–33.
- [Vinyals et al. 2019] Vinyals, O.; Babuschkin, I.; Chung, J.; Mathieu, M.; Jaderberg, M.; Czarnecki, W. M.; Dudzik, A.; Huang, A.; Georgiev, P.; Powell, R.; Ewalds, T.; Horgan, D.; Kroiss, M.; Danihelka, I.; Agapiou, J.; Oh, J.; Dalibard, V.; Choi, D.; Sifre, L.; Sulsky, Y.; Vezhnevets, S.; Molloy, J.; Cai, T.; Budden, D.; Paine, T.; Gulcehre, C.; Wang, Z.; Pfaff, T.; Pohlen, T.; Wu, Y.; Yogatama, D.; Cohen, J.; McKinney, K.; Smith, O.; Schaul, T.; Lillicrap, T.; Apps, C.; Kavukcuoglu, K.; Hassabis, D.; and Silver, D. 2019. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>.
- [Wilson and Sicart 2010] Wilson, D., and Sicart, M. 2010. Now it's personal: on abusive game design. In *Proceedings of the International Academic Conference on the Future of Game Design and Technology*, 40–47. ACM.
- [Wolpert and Macready 1997] Wolpert, D. H., and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* 1(1):67–82.

## Appendix D

# Ensemble Decision Systems for General Video Game Playing



# Ensemble Decision Systems for General Video Game Playing

Damien Anderson, Philip Rodgers and John Levine  
*Computer and Information Sciences*  
*University of Strathclyde*  
Glasgow, United Kingdom  
{damien.anderson, philip.rodgers, john.levine}@strath.ac.uk

Cristina Guerrero-Romero and Diego Perez-Liebana  
*School of Electronic Engineering and Computer Science*  
*Queen Mary University of London*  
London, United Kingdom  
{c.guerreroromero, diego.perez}@qmul.ac.uk

**Abstract**—Ensemble Decision Systems offer a unique form of decision making that allows a collection of algorithms to reason together about a problem. Each individual algorithm has its own inherent strengths and weaknesses, and often it is difficult to overcome the weaknesses, while retaining the strengths. Instead of altering the properties of the algorithm, the Ensemble Decision System augments the performance with other algorithms that have complementing strengths. This work outlines different options for building an Ensemble Decision System as well as providing analysis on its performance compared to the individual components of the system with interesting results, showing an increase in the generality of the algorithms without significantly impeding performance.

**Index Terms**—GVGAI, GVGP, Ensemble Decision Systems, Game AI

## I. INTRODUCTION

When developing agents to play a game, characteristics of the game under consideration can be included in the logic to guide them to the objective and play optimally, but that agent will not be able to perform well in other games or when a rule is updated. General Video Game Playing (GVGP) aims to tackle the challenge of building agents capable of performing well in different games, so these agents' value functions need to be as general as possible.

Most of the heuristics present in planning algorithms developed for GVGP focuses on specific goals, which usually are winning or guiding the agent following the maximization of the score. Others also include information about the proximity of different elements of the game, which is combined with the previous ones. The agents follow these goals and they perform well in certain types of games, while the reward structure is well defined, or the goals are in a reachable distance. However, what happens when the game is built in a way where the reward structure is not clear or is designed to guide the agents away from the optimal solution? What happens in games with large maps where the agent needs to move in a particular path to reach the goal? The agents who focus on just the goal and score will have a poor performance; being unable to solve the games. A solution would be building an agent with new goals that overcomes the weakness of that heuristics (e.g. focused on exploring the level). Nevertheless, even when this new agent could solve the games that the other ones were unable to, it

may become worse at games that it previously did well at, which is not the outcome we are looking for.

The approach taken by some authors to tackle this problem is using systems that combine different agents based on the type of game they believe they are facing, but their performance relies entirely on the correct prediction of the game. The solution we present in this paper is using an Ensemble Decision System (EDS), which is built to make the most of different agents by focusing on their strengths instead of weakness. We carried out an experiment where we ran a series of EDS with different configurations and compared their performance in contrast with known sample agents provided in the General Video Game AI framework. The results show the flexibility of this approach and open an interesting GVGP line of research to develop its full potential.

## II. BACKGROUND

The field of General Video Game Playing (GVGP) began in 2013 as a way of promoting interest towards developing game playing agents that could play a wide variety of video games [1]. The initial interest in this field originally sprang from work in General Game Playing and a desire to explore game playing agents in real-time situations [2].

The GVGP field is supported through an online competition known as the General Video Game AI competition<sup>1</sup> (GVGAI) where entrants can submit GVGP agents. The GVGAI competition offers a framework that provides a large library of games, up to 122 currently. Further games can be quickly created and added to the library through the Video Game Description Language (VGDL) that simplifies the game creation process [3]. The competition also offers a variety of tracks for different types of agents. These tracks focus on two main areas, planning and learning [4] [5]. The planning track offers both a single player and multi-player variation, which provide agents with a forward model in order to facilitate search-based algorithms. Agents can use the forward model to search through and evaluate future states of the game. The learning track, on the other hand, removes the forward model and replaces it with a training period before official evaluation occurs which allows agents to develop a policy for playing the

<sup>1</sup><http://www.gvgai.net/>

game. The work carried out for this research focuses on the single player planning track.

Over the years the competition has received a large number of entrants [6], though only a few have attempted to incorporate multiple types of algorithms into one system. These agents typically take on a portfolio approach, however, determining the type of game being played and applying a single algorithm to that game. One such example is *YOLOBOT*, which uses the game dynamics observed to determine whether a game is deterministic or stochastic, running, respectively, a heuristic Best First Search or Monte Carlo Tree Search (MCTS) [7]. Another agent, *Return42*, uses a similar approach but instead uses an A\* algorithm for stochastic games, and random walks for deterministic games [8]. These approaches have enjoyed a great deal of success in the GVGAI competition, yet, rely largely on the correct identification of the type of game to fully leverage the strength of the algorithms. Mendes *et al.* used a portfolio hyper-agent approach that predicted which of the seven controllers they included should be used based on the features present in the game; outperforming the winners of the 2014 and 2015 competitions [9].

The concept of tackling complex problems with an array of algorithms has found success in other areas of AI research. Most notably, Google Deepmind's *AlphaGo* makes use of multiple systems to create complex behaviour that is capable of defeating professional human players at the game of Go [10]. By using a combination of neural networks and tree search algorithms, the system is capable of succeeding where no single algorithm has been able to so far. Similarly, an Ensemble Decision System (EDS) has achieved a world record for an AI playing the game of *Ms. Pacman*. The EDS achieves this by allowing complex behaviour to emerge from the combining of simple algorithms that focus on specific tasks, such as collecting pills or dodging ghosts [11].

One of the main challenges to overcome is the notion of games, or environments, that are purposefully designed to lead agents away from an optimal outcome [12]. Perez *et al.* took a look at the robustness of agents to disruptive changes in the environment, such as a forward model that would return incorrect information, or a system that would occasionally apply a random action instead of the agent's intended action for a given state [13].

### III. CONTROLLERS

In this section we present, and briefly describe, the algorithms and heuristics that have been either included in the EDS implemented (Section IV) or executed for performance comparison.

#### A. Algorithms

The controllers used in these experiments belong to the sample pool of the GVGAI framework.

1) *sampleRandom*: The action is chosen randomly between the options available.

2) *One Step Look Ahead (OSLA)*: It estimates the reward gained for each of the possible actions on the next step of the game and chooses the action that returns the highest value. The *sampleOneStepLookAhead* agent uses by default the *SimpleStateHeuristic*, provided in the framework.

3) *Open-Loop Monte-Carlo Tree Search (OLMCTS)*: It is a variant of Monte-Carlo Tree Search (MCTS) [14] that is designed to work better in stochastic environments. It uses the forward model to reevaluate the actions instead of keeping the states of the game in the nodes of the tree. The *sampleMCTS* version provided in the framework has a rollout length of 10 and a C-value of  $\sqrt{2}$ . The default value function takes into consideration the winning condition and the raw score.

4) *Open-Loop Expectimax Tree Search (OLETS)*: It is based on the Hierarchical Open-Loop Optimistic Planning (HOLOP), improved to work better in stochastic environments. Its details and the differences between both algorithms are described in [4]. The version provided by the framework has a rollout length of 5 and the default value function takes into consideration the winning condition and the raw score.

5) *Rolling Horizon Evolutionary Algorithm (RHEA)*: It is an evolutionary algorithm that uses a population of individuals, which represents a sequence of actions to execute in a specific order. The plan of actions of each individual is evaluated using a forward model and the agent takes the first action of the individual with the best fitness [15]. The *sampleRHEA* provided in the framework uses the *WinScoreHeuristic*, evolves the individuals keeping 10 at a time and has a mutation rate of 1. It applies mutation and crossover to get the next generation until the time runs out. Because it produces as many sequences as possible in this time, the number of individuals is dynamic.

6) *Random Search (RS)*: It is similar to RHEA, but it randomly generates the individuals instead of evolving them. The *sampleRS* uses lengths of 10, and it produces as many sequences as possible in the given time.

The algorithms that we executed in the experiments as a point of comparison were not modified in any way. However, those that we included as part of the EDS required some modifications that do not affect their core implementation but allow them to fit the needs of the system. These include those related to the abstraction of the heuristics [16], and the modifications needed to make the algorithms return Opinions instead of the action to take. As detailed in the next section, an Opinion is a simple data structure which holds the action to take, a value computed for that action and the name of the algorithm which suggested it.

#### B. Game Heuristics

The EDS combines different algorithms and game heuristics. The heuristics that we utilize go further than merely winning the game by following score, and are based on the work of Guerrero-Romero *et al.* [16].

1) *Winning Maximization Heuristic (WMH)*: Its goal is winning the game, maximizing the score when reaching the winning status is not possible.



2) *Exploration Maximization Heuristic (EMH)*: Its goal is maximizing the exploration of the level, prioritizing visiting those locations that were not visited before, or have been visited less often. For these experiments, this heuristic has been updated regarding the one in [16], so winning the game is always rewarded instead of penalized.

3) *Knowledge Discovery Heuristic (KDH)*: Its goal is interacting with the game as much as possible to trigger interactions, and new sprite spawns. For these experiments, this heuristic has been updated regarding the one in [16], so winning the game is always rewarded instead of penalized.

4) *Knowledge Estimation Heuristic (KEH)*: Its goal is interacting with the game to predict the outcomes of the interactions between the different elements of the game, related to both the victory status and score modifications.

#### IV. ENSEMBLE DECISION SYSTEMS

Each of the algorithms described have their own strengths and weaknesses. Each algorithm can solve a different set of problems, but none can currently solve all of them.

Ensemble Decision Systems are a flexible system for combining the decisions of multiple algorithms into a single action. Instead of trying to develop complex problem solving behaviour in a single algorithm, complexity is built up with simpler layers of behaviours that each focus on different aspects of a problem, or different types of problems altogether.

Expanding the capabilities of an algorithm whilst maintaining the strengths it already has can be challenging. EDS's offer a potential solution to this by allowing each algorithm to focus on its strengths, and addressing their weaknesses through other algorithms. Intuitively, if an algorithm is good at finding paths through an environment, but bad at identifying long term goals, then combining that initial algorithm with a long term planner may give the overall system the best of both worlds.

##### A. Architecture

The EDS is comprised of two components. The algorithms, known as *voices*, which evaluate the current state, and the *arbitrator* (Section IV-C), which makes the final decision of which action to take. When a game begins, the arbitrator is given the state which is then passed to each voice. The voices perform their own analysis and then return their *opinion* to the arbitrator. An opinion is a data structure which holds the action selected, as well as a value assigned to that action.

Once every voice has returned their opinion, the arbitrator will use a final action selection policy to decide which action to take, based on the information from the voices.

This architecture can be adjusted in a variety of ways. First of all, the number of voices, and which are used, can be altered at implementation or at run-time. For example, if a voice does not seem to be performing well in a given game, it can be disabled or swapped with another voice to improve overall system performance. The action selection policy is also a parameter which can be adjusted. Some examples would be using a bandit selection algorithm to decide which action to take from the opinions, or a diplomatic option which selects

the action that most voices have selected. A neural network could also be trained to identify which voices do well in certain states, as an action selection policy. Each voice could potentially assign a value to all possible actions, and the action selection policy then decides based on the highest value, and could potentially decide actions that none of the voices had primarily suggested. There are a wide number of possibilities available for adjusting the EDS.

##### B. Action Selection Policies

The policy used to select the action to take has a large impact on the overall behaviour of the EDS. The variations that were used for the experiments are the following:

1) *Highest Value*: This policy simply selects the action which returns the highest value, based on the analysis of its voice. The possible range from this is not currently limited, though future work would look at normalizing the output from the heuristics.

2) *Diplomatic*: This policy is only useful when using more than two voices in the system, and essentially selects the action that has the most votes from the voices. The action that is returned with each Opinion is counted as a vote in favour of that action. In the case that no majority exists, then an alternative action selection policy is used. For the experiments in this work, this was the random action selection policy.

3) *Random*: This policy simply selects an opinion at random and uses the action assigned to it.

##### C. Arbitrator

The arbitrator can be implemented in a number of different ways, depending on the constraints of the problem set. In particular, the GVGAI has a 40ms per action time limit that needs to be respected, which has a significant impact on the analysis that algorithms can do. This has a large impact on an EDS because each algorithm has to be allocated a slice of that 40ms. Further impacting this, the GVGAI does not allow multi-threading, which is where an EDS would be able to leverage the voices simultaneously.

In order to deal with these limitations, two different arbitrators, described in this section, were created and tested.

1) *Central Arbitrator*: This arbitrator splits the 40ms evenly between all of the voices within the system. As an example, if there are  $N$  voices, then each voice receives  $(40/N) - 1$  ms per time-step to return its opinion.

2) *Asynchronous Arbitrator*: This arbitrator gives each of the voices within the system the full 40ms decision time, but does so by skipping actions until each voice has returned an opinion. If there are  $N$  voices within the system, then for  $N-1$  time-steps a Nil action is returned and a voice performs its analysis. At the  $N$ th time-step, after all voices have returned their Opinions, then an action is selected based on the current action selection policy. The decision of taking no-op actions comes from the will to affect the game the least, so the voices analyze the most similar state of the game as possible.

#### D. Variations

Some of the variations that were tested for this work are outlined in the following sections.

1) *BestFour*: This variation makes use of prior work, which measured the performance differences in playing GVGAI games with different heuristics [16]. The heuristics created for that work with the algorithms identified as performing best for each heuristic were used here in an EDS. There are four voices in this variation (OLETS with WMH, RS with EMH, RS with KDH and OLETS with KEH), the Central Arbitrator is used, and the action selection policy is Highest Value.

2) *BestFourDiplo*: This variation uses the same four voices as BestFour (OLETS with WMH, RS with EMH, RS with KDH and OLETS with KEH) and the Central Arbitrator, but it uses Diplomatic as the action selection policy instead.

3) *BestExpSc*: There are two voices in this variation: OLETS with WMH and RS with EMH. The Central Arbitrator is used, and the action selection policy is Highest Value.

4) *MCTSExpSc*: There are two voices used in this variation: MCTS with WMH and MCTS with EMH. This variation was tested to give a more direct comparison to the sampleMCTS controller. The Central Arbitrator is used and the action selection policy is Highest Value.

5) *OLETSExpSc*: There are two voices used in this variation: OLETS with WMH and OLETS with EMH. This variation was tested to give a more direct comparison to the OLETS controller. The Central Arbitrator is used, and the action selection policy is Highest Value.

6) *OLETSExpScAsync*: There are two voices used in this variation: OLETS with WMH and OLETS with EMH. It uses the the Asynchronous Arbitrator and Highest Value as the selection policy. This variation was also tested to give a more direct comparison to the OLETS controller.

#### V. GAME SELECTION

There are 122 available single-player games in the GVGAI Framework at the time of writing<sup>2</sup> and running the experiments using all of them is prohibitively time consuming. Therefore, a subset of these was selected, with enough diversity to ensure that the games selected represent the full variety of options available from the GVGAI set.

The choice of games for the experiment was based on previous work. Twenty games were based on Gaina *et al.* selection in [17], where they used a balanced set of 20 stochastic and deterministic games. We also included the ten games which collectively provided the highest information gain in Stephenson *et al.* work in [18]. Because 2 of the games are present in both sets (*Chopper* and *Escape*), the total number of games used in the experiment was 28 (Table I).

The selection includes a series of deceptive games, which are designed to lead the agent away from the most optimal solution [12]:

<sup>2</sup>[https://github.com/GAIGResearch/GVGAI/blob/master/examples/all\\_games\\_sp.csv](https://github.com/GAIGResearch/GVGAI/blob/master/examples/all_games_sp.csv)

TABLE I  
THE GAMES SELECTED FROM THE GVGAI FRAMEWORK (GAMES IN BOLD ARE DECEPTIVE)

<i>Aliens</i>	<i>Avoidgeorge</i>	<i>Bait</i>	<b><i>Butterflies</i></b>
<i>CamelRace</i>	<i>Chase</i>	<i>Chopper</i>	<i>Crossfire</i>
<i>Digdug</i>	<i>Escape</i>	<i>Freeway</i>	<i>Hungrybirds</i>
<i>Infection</i>	<i>Intersection</i>	<b><i>Invest</i></b>	<i>Labyrinthdual</i>
<b><i>Lemmings</i></b>	<i>Missilecommand</i>	<i>Modality</i>	<i>Plaqueattack</i>
<i>Roguelike</i>	<i>Seaquest</i>	<b><i>Sistersavior</i></b>	<i>Survivezombies</i>
<i>Tercio</i>	<i>Waitforbreakfast</i>	<i>Watergame</i>	<i>Whackamole</i>

a) *Butterflies*: The goal is capturing all of the butterflies before the time runs out or all the cocoons open. A cocoon opens when a butterfly collides with it, spawning a new butterfly. Each butterfly captured increases the score +2. Therefore, the higher the number of cocoons opened, the higher the final score can be. Trying to win quickly leads to a less optimal solution, as the score will not be as high as it could.

b) *Invest*: The goal is maximizing the score by collecting gold coins and investing them with the investors. Investing too much and getting into debt results in a loss. Once an investment is made, that investor will disappear for a set period, before returning and awarding a higher amount of points to the player. Each gold coin awards 1 points when collected; the Green investor takes 3 points and returns 5 after 30 timesteps; the Red investor takes 7 points and returns 15 after 60 timesteps; the Blue investor takes 5 and returns 10 after 90 timesteps.

c) *Lemmings*: Lemmings are spawned from one door and try to get to the exit of the level. There are obstacles on the way, so the goal is destroying these so the lemmings can reach the exit. For every lemming that reaches the exit +2 is given, but -1 is subtracted from every piece of wall destroyed. If a lemming falls into a trap, the score is reduced -2, and if it is the avatar who falls, the score is reduced -5.

d) *Sistersavior*: The goal is rescuing all of the hostages to be able to defeat the scorpion and win the game. There is the option to kill the hostages, receiving a moderate immediate reward (+2 points if shot vs. +1 if rescued). The scorpion provides 14 points if defeated, but it can only be defeated if all 3 hostages are rescued.

#### VI. EXPERIMENTAL WORK

This section describes the experiments that were carried out and provides instructions for replication.

##### A. Experimental Setup and Specifications

The code used for the experiments is in a Github repository<sup>3</sup>. All of the experiments were performed using a laptop running Linux Mint with an Intel i7-8565U CPU with 16GB of RAM.

The GVGAI framework has up to five levels available for each of the games. Each level was run 30 times in order to build up confidence in the results being gathered, having 150

<sup>3</sup><https://github.com/Damorin/PhD-Ensemble-GVGAI/tree/master/src/COGPaper>

runs per game for each controller. Each successful iteration of a game that resulted in either a win or loss without any crashes produces one unit of data, containing the information [agent, game, score, win/lose, time]. In total, 48,600 units of data were generated from these experiments, with each game generating 1800 units of data across all of the agents.

The controllers described in Sections III and IV were used for these experiments. The final list of controllers can be found in Table II.

TABLE II  
FULL LIST OF CONTROLLERS USED

EDS Agents	Sample Agents
BestExpSc	OLETS
BestFourDiplo	sampleMCTS
BestFour	sampleOneStepLookAhead
MCTSExpSc	sampleRandom
OLETSExpSc	sampleRHEA
OLETSExpScAsync	sampleRS

### B. Results

To thoroughly analyze the performance of the GVGAI agents, both wins and scores are taken into account. A number of visualizations and a ranking of performance based on those metrics have been created. These are detailed below:

1) *Percentage of wins per game:* Fig. 1 shows the percentage of wins that each agent achieved in each game. It has been ordered to show the win rates per game, with the easier games to win at the top and the harder at the bottom.

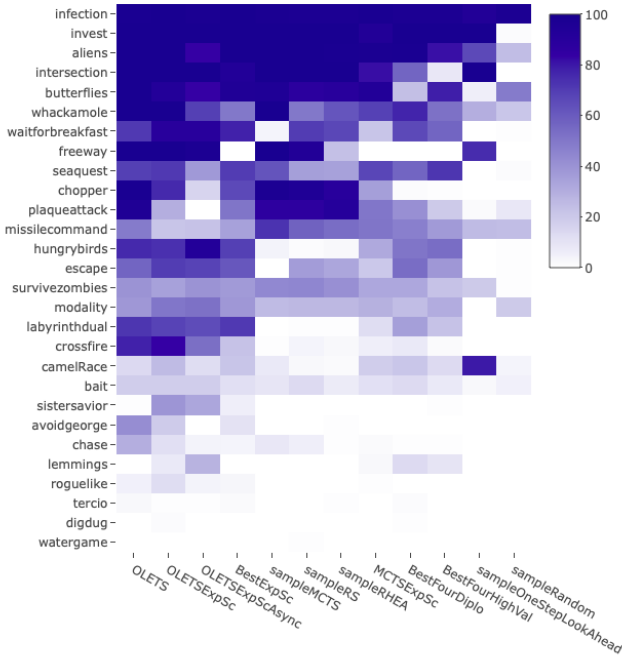


Fig. 1. Total percentage of wins (0 – 100%) of each controller by game.

2) *Score average per game:* Fig. 2 shows the average score that each agent achieved per game. The score depends on the game considered, so to have all the results on the same scale to be able to be displayed together, the average of the scores per game were normalized. This normalization was done by taking the maximum and minimum score average obtained by the agents in a game, and used those values for the normalization, so that for every game, there is an agent whose score is represented by 0 and another represented by 1, and all others in between. This approach visualizes the difference in scores between the agents, by game, in an informative manner. It is possible to distinguish between those games were agents manage to get a score easily, and those were they struggle, and the contrast between the higher and lower average obtained.

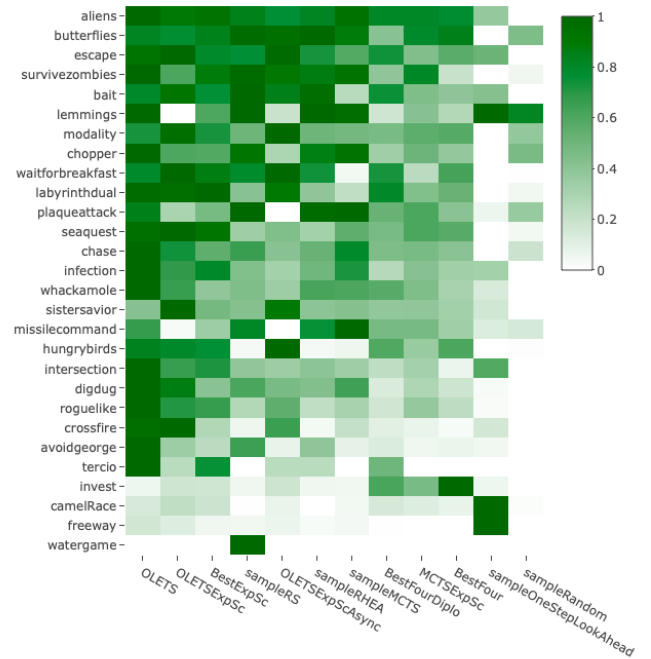


Fig. 2. Average scores per game. To be able to make a comparison between the different games, the scores have been normalized, taking the maximum and minimum average scores obtained in each game as limits for that game.

3) *F1 Ranking:* In order to have an overall performance of the agents through every game, we include a Formula 1 (F1) ranking, like the one used in the GVGAI Competition [4]. Each agent receives 25, 18, 15, 12, 10, 8, 6, 4, 2, 1 or 0 points per game based on their performance regarding the rest of the algorithms in that game. The performance is measured with the data that was gathered at the end of each run (Section VI-A), considering the total number of wins, the average of the score and the average of timesteps per game. In general, the agent with a higher number of wins is better in that game. In case there is a tie, the highest average score between the agents breaks the tie. In the case that there is a tie in both measures, the lowest average of timesteps is considered. The highest ranked agent receives 25 points in

that game, the second one 18, and so on, until all agents have been assigned a score. If there is a tie in the three measures, those agents involved in it will receive the same number of points. Table III shows the final ranking overall the 28 games used in the experiment.

TABLE III  
CONTROLLERS RANKED BY THE TOTAL NUMBER OF F1-POINTS GAINED OVERALL THE GAMES. IT INCLUDES THE TOTAL PERCENTAGE OF WINS (%WINS) AND THE NUMBER OF DIFFERENT UNIQUE GAMES WON (#GAMES) BY EACH OF THEM.

	Controller	F-1	% Wins	#Games
1	OLETS	501	55.69%	25
2	OLETSExpSc	463	53.59%	27
3	BestExpSc	318	42.86%	24
4	OLETSExpScAsync	316	45.52%	24
5	sampleMCTS	253	40.00%	19
6	sampleRS	237	39.76%	22
7	BestFourDiplo	213	33.02%	23
8	sampleRHEA	194	36.71%	23
9	BestFour	182	28.93%	22
10	MCTSExpSc	181	33.29%	22
11	sampleOneStepLookAhead	88	21.48%	12
12	sampleRandom	29	9.29%	15

### C. Discussion

The first finding is that overall the best performing agent is the OLETS agent, which gets a final score of 501 points and wins the most amount of games out of all of the agents (55.69%). The second highest, OLETSExpSc, has a similar level of performance (53.59%), though the games that it wins are quite different from OLETS. In particular, it is able to outperform OLETS on a variety of games, such as *Waitforbreakfast*, *Crossfire* and *Escape*. OLETSExpSc also appears to make progress towards winning over a wider range of games than OLETS, with OLETS winning at least once 25 unique games, as opposed to OLETSExpSc, winning 27.

In regards to deceptive games, which are currently not solved by the sample agents, such as *SisterSavior* and *Lemmings*, EDS agents manage to achieve significant results. Interestingly, the best-performing agents on these levels are EDS agents which use OLETS as their voices, but the standard OLETS algorithm is not able to progress on these games. This improvement in the OLETS performance suggests that combining the traditional OLETS algorithm with a dedicated search algorithm provides enough of an advantage to allow OLETS to solve more problems beyond its original capability.

OLETSExpSc gets 59 of 150 wins in *SisterSavior*, OLETSExpScAsync, 50, and BestExpSc, 9, when OLETS gets 1 and the rest of the algorithms, none. In particular, the EDS algorithms are able to consider compromising their immediate score reward (saving the hosts instead of killing them), having the chance to reach the most challenging goal and win the game.

In *Lemmings*, the EDS agents that manage to win the game are MCTSExpSc, OLETSExpSc, BestFour, BestFourDiplo and OLETSExpScAsync, with 4, 12, 15, 20 and 43 of 150 wins respectively. As Fig 3 shows, they also have the lowest score

average, as they use heuristics with further considerations other than trying to maximize the score, not being afraid of losing points while exploring the level and destroying the walls. Even when the solutions might not be the most optimal ones, and their levels could be solved getting a higher score average, they are able to win the game when the other controllers fail.

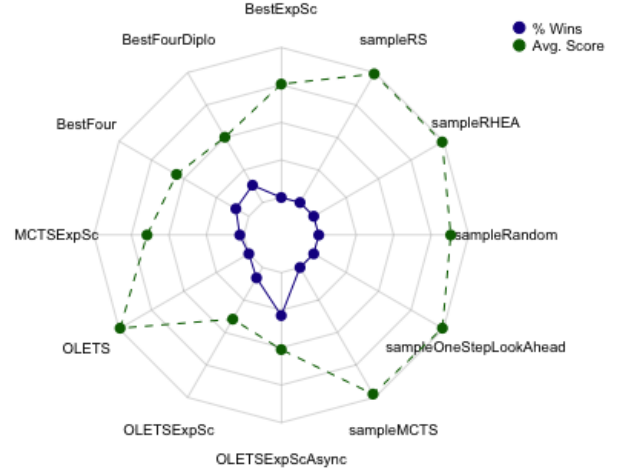


Fig. 3. *Lemmings* stats per agent. The pct. of wins is in range [0, 100%] and the avg. of score is relative to the maximum and minimum scores achieved in the game (range: [-143, 1]). Values have been normalized in this figure.

Looking further at the deceptive games, *Invest* has some interesting results (Fig. 4). All of the sample agents score an average of five or less across their games of *Invest*, while the EDS agents manage to score significantly higher. This is an interesting result as the deception in *Invest* was designed to trick search based agents that would typically be reluctant to surrender their current score in order to progress. The EDS agents are able to perform well in this game, with BestFour scoring highest with an average of 123. Interestingly, one of the EDS agents, MCTSExpSc, loses some of the games. As the sampleMCTS agent does not lose any games of *Invest* in the experiments, it seems that the exploration combined with MCTS causes it to take some risks, while increasing its overall performance.

Games where OLETS still performs notably better than its EDS versions, are discussed next. In *Missilecommand*, OLETSExpSc gets 22.00% of wins and OLETSExpScAsync 22.67%, in contrast with OLETS's 48.67%. We believe that this decrease is due to the exploratory behaviour added to these agents, which brings them to explore the level instead of focusing on achieving the goal of the game. When the agents finally realize they are about to lose, they are too far away to be able to prevent themselves from losing the game. In *Chopper*, OLETS manages to win most of the games (99.33% of wins) with an average of score of 17.03. In contrast, OLETSExpSc achieves a 76.00% of wins and an average of score of 4.5, and OLETSExpScAsync gets a 16.00% of wins averaging -5.55 points. In *Plaqueattack*, the

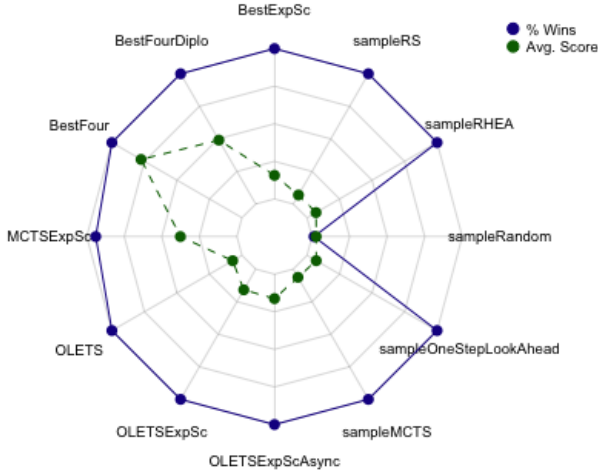


Fig. 4. *Invest* stats per agent. The percentage of wins is in range [0, 100%] and the avg. of score is relative to the maximum and minimum scores achieved in the game (range: [-7, 161]). Values have been normalized in this figure.

decrease of performance is even higher: OLETS manages to get 96.00% of wins when OLETSEpSc only wins a 30.00% of the games and OLETSEpScAsync does not win any of them. Looking at the behaviour of the agents in both of these games, during observation the Async controller spends most of the time exploring rather than learning to gain points. The central controller moves much faster, reaching new positions sooner and, therefore, lessening the value of the exploration heuristic sooner. It allows it to focus on the score and win the game. This may be due to the exploration heuristic providing excessively high evaluations in the decision process of these agents, causing them to ignore the actual goal. Hence, the value returned by the EDS agents' heuristics should be tuned in future work to have a more balanced input from all of the heuristics in levels with a large state space, particularly where the rewards are focused in a small area of the level.

*CamelRace* is a racing game where the avatar must get to the finish line before any other camel does. It is a compelling case to comment on, as it is quite a simple game that the complex algorithms struggle to solve but OSLA, one of the simplest controllers in the sample batch, gets 120 of 150 wins. Some EDS agents perform better than OLETS (21 wins) in this game (MCTSEpSc, BestFourDiplo, BestExpSc and OLETSEpSc, with 28, 32, 33 and 38 wins respectively) but they are far behind OSLA. The reason for the good performance of OSLA comes from how fast the game is, the well defined path that should be followed and the lack of score rewards until the end. Because the win state is not reached unless the agent heads to the goal, and there is no score information to use, the agents that just use these values as a reference will behave randomly until their movement heads them to the exit. If those agents are lucky enough to choose those actions that get them closer to it quickly, then they may win. Exploring in this game is also not beneficial, because the path to the exit should be followed quickly to avoid losing. OSLA's

heuristic (SimpleStateHeuristic) considers, apart from winning condition and score, those actions where the distance between the avatar and the closest portal sprite (which is the category of the goal sprite) is small. In the first four levels, where this information is enough to reach the goal, it manages to get there promptly. However, in the last level, where there is a wall between the agent and the goal, it gets stuck not being able to rectify its path.

Finally, an example where the exploration heuristic is profitable is *Hungrybirds*. In this game, the player is a bird that needs to exit a maze but needs to find food and eat it while doing so to avoid dying. As observed in Fig. 5, the only sample agent with a significant number of wins is OLETS, while the rest of the sample controllers achieve less than a 4.60% winning rate. In contrast, all EDS agents based on sample controllers achieve between 32.00% and 74.00% win rate, and OLETSEpScAsync improves OLETS (92.67% vs 76.00%). Something similar happens in other games such as *Escape*, *Labyrinthual* and *Lemmings*. These results show how using agents with different policies can help to improve the win rates in a game, showing the potential of the approach developed.

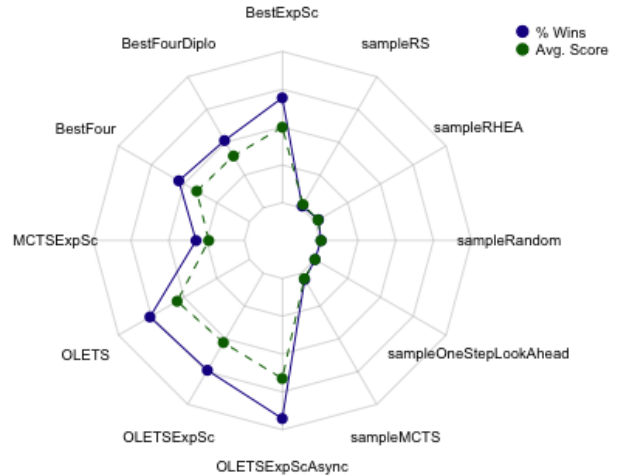


Fig. 5. *Hungrybirds* stats per agent. The pct. of wins is in range [0, 100%] and the avg. of score is relative to the maximum and minimum scores achieved in the game (range: [0, 140]). Values have been normalized in this figure.

## VII. CONCLUSIONS AND FUTURE WORK

This work presents a novel approach to designing GVGP agents through the use of an Ensemble Decision System (EDS) and showcases a number of potential variations of how such a system may work. While the EDS does not currently outperform the individual algorithms, it maintains a significant portion of their strengths and adds to the generality of the agent by allowing it to succeed at more games. The flexibility of the EDS is the main strength that it offers, and the potential of this approach has not yet been fully explored.

The results show the potential of combining different algorithms with different goals using this kind of system. OLETS' EDS variant is able to win two more games than the original

agent without compromising its overall performance, simply by providing two different intentions to take into consideration. One of these games, *Lemmings*, is a deceptive game that none of the sample controllers are able to solve but five different EDS systems win, one of them 43 times. This is an impressive result that shows what this approach is capable of achieving, having noted potential that could be developed further. A similar generality improvement is seen between the sampleMCTS and MCTSExpSc, winning in 19, vs 22 games.

Future work for this project will look at developing further variations of the system, which could look at different action selection policies or types of arbitration. Possibilities in this area could include developing a bandit style selection algorithm for deciding which voice should be selected, or the arbitrator itself could be a neural network that is trained to identify which voices are best suited to deal with a given state. It will also compare EDS to portfolio approaches as they have distinct strengths and weaknesses that would be interesting to analyze in detail.

At the moment the EDS variations that have been implemented are quite naive. There has not been any optimization of which voices work best together, or development of the more sophisticated arbitrators and action selection policies. Each voice is treated as an expert, and their evaluation is trusted. This creates some performance issues for certain games, that were highlighted earlier in Section VI-C. In particular, it seems that there are cases where the exploration heuristic might be providing more weight in the decision process of the agents than it should. This has been noticed especially in games that present significant exploratory areas but where the primary objective of the game is focused on a particular area. This problem could be improved by not only tuning the values returned by the heuristics, but by adding in voices that are designed to fulfill a specific purpose. As an example, a survival voice could be added to an EDS whose role would either be to veto actions that lead to a losing state or remove an action from consideration. Or perhaps all voices could be able to veto actions that they evaluate as being dangerous. A fitness function could be built into the arbitrator that is able to evaluate how well a voice is performing in a particular situation, and could adjust the confidence of that voice or swap it with another voice. Voices themselves could be required to evaluate all actions available, and the system could then make selections based on evaluations from all perspectives. Lastly, a voice does not need to be a singular algorithm, but could instead be a collection of algorithms working together to return a single opinion. An EDS of multiple EDSs.

In short, the EDS offers a considerable amount of flexibility and consolidates the strengths of current GVGP approaches to overcome their weaknesses, making the system more robust overall. It is also worth noting that while a range of variations were built for these experiments, they are still naive in their implementation. Further sophistication in their decision making process, and a more intelligent selection process regarding which voices to pair together will likely improve performance further. With the recent trend of high performing solutions

coming not from singular algorithms, but from algorithms working together, the interesting results of this paper show that this is a fruitful area for further research.

#### ACKNOWLEDGMENT

This work was partially funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1.

#### REFERENCES

- [1] J. Levine, C. Bates Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General Video Game Playing," 2013.
- [2] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI magazine*, vol. 26, no. 2, pp. 62–62, 2005.
- [3] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a Video Game Description Language," 2013.
- [4] D. Pérez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.
- [5] J. Liu, D. Pérez-Liebana, and S. M. Lucas, "The Single-Player VGAI Learning Framework-Technical Manual," in *Technical report*. Queen Mary University of London, 2017.
- [6] D. Pérez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms," *arXiv preprint arXiv:1802.10363*, 2018.
- [7] T. Joppen, M. U. Moneke, N. Schröder, C. Wirth, and J. Fürnkranz, "Informed Hybrid Game Tree Search for General Video Game Playing," *IEEE Transactions on Games*, vol. 10, no. 1, pp. 78–90, 2018.
- [8] D. Ashlock, D. Pérez-Liebana, and A. Saunders, "General Video Game Playing Escapes the No Free Lunch Theorem," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 17–24.
- [9] A. Mendes, J. Togelius, and A. Nealen, "Hyper-Heuristic General Video Game Playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.
- [10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the Game of Go without Human Knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.
- [11] P. Rodgers, J. Levine, and D. Anderson, "Ensemble Decision Making in Real-Time Games," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.
- [12] D. Anderson, M. Stephenson, J. Togelius, C. Salge, J. Levine, and J. Renz, "Deceptive Games," in *International Conference on the Applications of Evolutionary Computation*. Springer, 2018, pp. 376–391.
- [13] D. Pérez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "Analyzing the Robustness of General Video Game Playing Agents," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.
- [14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4:1, pp. 1–43, 2012.
- [15] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liebana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 418–434.
- [16] C. Guerrero-Romero, A. Louis, and D. Pérez-Liebana, "Beyond Playing to Win: Diversifying Heuristics for VGAI," in *Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 118–125.
- [17] R. D. Gaina, S. M. Lucas, and D. Pérez-Liebana, "Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing," in *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2017, pp. 1956–1963.
- [18] M. Stephenson, D. Anderson, A. Khalifa, J. Levine, J. Renz, J. Togelius, and C. Salge, "A Continuous Information Gain Measure to Find the Most Discriminatory Problems for AI Benchmarking," *arXiv preprint arXiv:1809.02904*, 2018.