University of Strathclyde

Doctoral Thesis

# Generalising Plans to Influence Landscapes for Robust Agent Execution in Virtual Worlds

*Author:*
Luke Dicken

*Supervisor:*
Dr. John Levine

*A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy*

*in the*

Strathclyde Artificial Intelligence and Games Research Group
Department of Computer and Information Sciences

July 4, 2016

# Declaration of Authorship

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:

Luke Dicken
July 4, 2016

UNIVERSITY OF STRATHCLYDE

# *Abstract*

Department of Computer and Information Sciences

Doctor of Philosophy

**Generalising Plans to Influence Landscapes for Robust Agent Execution in Virtual Worlds**

by Luke Dicken

Artificial Intelligence is one of the most promising areas of modern technology, with great potential for changing the face of the modern world. However almost universally we use one of two paradigms in AI – either techniques that are very efficient but which lack good long-term reasoning, or techniques that are exceptionally good at providing long-term solutions but which are slow to execute and reasonably inflexible. This is a problem because the natural world does not decompose so neatly into one or other box, many situations require fast problem-solving that is also cognisant of long-term objectives and motivations.

A great example of this kind of problem is frequently encountered in the video game industry, where the computational load is predominantly tied up simulating an environment and rendering this graphically to the player. As a consequence there is very limited processing power available for the AI systems that power the activity within that environment. However, there is also a need for the actions being taken by the agents in the game to at least appear to be intelligent, and for many games that intelligence needs to be exhibited in a dynamic, rapidly changing world. This is a clear example of an area where efficient long-term reasoning is necessitated, and cannot adequately be solved with either of the two existing families of algorithms.

The core hypothesis of the work is derived from the need to bridge the gap between the two paradigms and states that an architecture that operates in this way "will be demonstrably more robust and efficient than those that are either purely Reactive or purely Deliberative." Additionally, as a technique intrinsically bound with an industrial need, it is also essential that any proposed architecture be viable in an industry setting.

The work firstly presents an extensive literature review focusing on techniques for both Reactive and Deliberative reasoning, with a particular emphasis on those that are in use in the video game industry. By understanding contemporary techniques, along with their strengths and weaknesses, a context for the problem is provided from which a new framework will be created that draws strengths from both paradigms and mitigates the potential weaknesses. The result is the Integrated Influence Architecture (I2A) which uses a combination of techniques from both types of reasoning and leveraging the nature of video games to make use of the large amount of resources available during their development, rather than relying on the comparatively small amount available at runtime. The I2A functions primarily by creating a "Common Representation" that is generated from a symbolic description of the game world, formatted in such a way that it lends itself to mathematical manipulation (as opposed to more traditional symbolic representations that are better suited to search). The premise is that by using such a representation, information that has been generated from either Reactive or Deliberative types of reasoning can be combined to provide a holistic view of the situation, informed by both paradigms

This bulk of the work sets out the overall methodology for the I2A, specifically the manner in which a problem can be described using a symbolic description language (PDDL) and from this the Common Representation can be calculated. During execution, the CR is used much like an Influence Map with sources of influence coming from the different reasoning systems. Additionally time is spent on evaluating the proposed I2A by the criteria established. This is done by demonstrating that the processes underpinning it are sound, that the technique can be used in an industry setting, that it is capable of reacting to threats within the context of long term reasoning and that the I2A as a whole is an efficient process. Potential future directions that the work could be taken in in order to have wider applicability and better results are also discussed.

# *Acknowledgements*

This work would not have been possible without the input and involvement of more people than I can reliably name. I owe a huge debt to all the practitioners around the world who have taken time out to help me to understand the issues and challenges facing Game AI. To Chris Preston at Ubisoft Reflections who first looked at my work, gave me insights and helped me to realise that industry engagement didn't need to be scary. To Alex Champandard of AIGameDev.com, who not only runs one of the best conferences in the field but also a very welcoming community for those aspiring to learn and apply AI in games. To Dave Mark and Steve Rabin whose AI Game Programmers Guild remains a clubhouse I'm not sure I deserve to be a part of, and to Neil Kirby without whom I wouldn't have been accepted. To Kevin Dill at Lockheed Martin's Advanced Simulation Center for the support and encouragement, and for prompting me to discuss this work at GDC. I'm also grateful to so many members of AIGameDev.com, the AIGPG and the International Game Developers Association who I've interacted with over the years - too numerous to list by name. You know who you are.

On the academic side, I'm grateful to old hands such as Michael Mateas, Mark Riedl and Julian Togelius, and to the newcomers, Michael Cook, Josh McCoy, Gillian Smith, Ben Sunshine-Hill and Ben Weber who have all had wisdom and insight to share.

Closer to home, putting all of this together wouldn't be possible without a great support system backing me up. To my family and friends, I couldn't have done this without you. My parents in particular deserve an honourable mention for all they've done to support me throughout this. Years of proof reading and insufferable conversations have finally come to an end! Brian McDonald also deserves recognition for making my final submission possible, acting as my tele-presence from 5,000 miles away.

To my friends and family at Zynga who have supported, teased, cajoled and otherwise motivated me, and then allowed me space to complete my corrections and submit them alongside a demanding project, thank you. Special thanks to Alex Ntoulas, Moises Goldszmidt and Arash Nia for all their assistance in the past 18 months.

And finally thanks of course to my supervisor Dr John Levine. It might have taken a couple of years for me to explain this well enough for you to see the whole picture John, but thank you for believing in it anyway, and helping me to reach the finish line!

I'm honoured to have shared this thing of ours with so many wonderful people. Thank you all for letting me be a part of it.

# Contents


**Declaration of Authorship** i

**Abstract** ii

**Acknowledgements** iv

**Contents** v

**List of Figures** ix

**1 Introduction** 1
- 1.1 A History of Artificial Intelligence in Video Games . . . 2
  - 1.1.1 The First Artificially Intelligent Games . . . 3
  - 1.1.2 Artificial Intelligence in Contemporary Video Games . . . 4
- 1.2 Motivation . . . 8
  - 1.2.1 Reactive Systems . . . 8
  - 1.2.2 Deliberative Systems . . . 10
  - 1.2.3 The Necessity to Bridge the Gap . . . 11
- 1.3 Research Statement . . . 12
  - 1.3.1 Concept . . . 12
  - 1.3.2 Research Process . . . 13
  - 1.3.3 Applicability . . . 14
  - 1.3.4 Evaluation . . . 14
  - 1.3.5 Summary . . . 15
- 1.4 Papers and Presentations Used . . . 16
- 1.5 Thesis Layout . . . 17

**2 Related Work** 18
- 2.1 Automated Planning . . . 19
  - 2.1.1 Problem Representation . . . 19
    - 2.1.1.1 PDDL . . . 20
    - 2.1.1.2 SAS+ . . . 23
      - Generating SAS+ From PDDL . . . 24
  - 2.1.2 Classical Planning . . . 25
    - 2.1.2.1 Heuristic Search and A* . . . 26
  - 2.1.3 Contemporary Planning - Planning Graphs and DTGs . . . 27
    - 2.1.3.1 Fast Forwards . . . 29

The Relaxed Planning Graph . . . 30
Helpful Actions . . . 30
2.1.3.2 Macro Actions . . . 31
2.1.3.3 Landmark Analysis . . . 32
2.1.4 Planning For Uncertainty . . . 34
2.1.4.1 Probabilistic Planning . . . 34
Markov Decision Processes . . . 35
2.1.4.2 Plan Repair . . . 35
2.1.4.3 Partial Observability . . . 36
2.1.5 Automated Planning Summary . . . 36
2.2 AI Techniques in Games . . . 37
2.2.1 State-based Techniques . . . 38
2.2.1.1 Basic Finite State Machines . . . 38
2.2.1.2 Hierarchical Finite State Machines . . . 39
2.2.1.3 Hierarchical Concurrent State Machines . . . 40
2.2.1.4 Behaviour Trees . . . 42
Behaviour Trees and FSMs . . . 44
2.2.2 Mathematical Techniques . . . 45
2.2.2.1 Influence Maps . . . 45
2.2.2.2 Utility-based Architectures . . . 47
Reasonable vs Optimal Play . . . 49
2.2.2.3 Neural Networks . . . 50
2.2.3 Deliberative Techniques . . . 53
2.2.3.1 Goal Oriented Action Planning . . . 53
2.2.3.2 HTN . . . 56
2.2.3.3 Monte Carlo Tree Search . . . 57
Monte Carlo in Games . . . 59
POMCoP . . . 61
2.2.4 Game AI Summary . . . 62
2.3 Other Relevant Architectures for AI Agents . . . 63
2.3.1 Subsumption . . . 63
2.3.2 Three Layer Architecture . . . 65
2.3.3 T-REX . . . 66
2.4 Summary of Existing Systems . . . 67
2.4.1 I2A and GOAP . . . 68
2.4.2 I2A and Influence Maps . . . 68
2.4.3 I2A and Utility . . . 69
2.4.4 I2A and Markov Decision Processes . . . 69

**3 Method** **71**
3.1 Concept . . . 71
3.2 Overview . . . 73
3.2.1 Components . . . 73
3.2.1.1 Develop-time . . . 74
3.2.1.2 Run Time . . . 75
3.2.2 The I2A In Practice . . . 77
3.2.2.1 Develop Time . . . 78

3.2.2.2 Run Time . . . 80
3.3 The Common Representation - Compiled PDDL . . . 82
3.4 Abstraction of States . . . 87
3.4.1 Abstraction Through Clustering . . . 88
3.4.2 The Adapted Fuzzy c-Means Algorithm . . . 89
3.4.2.1 Consequences of Operation in a Discrete Space . . . 91
3.5 Types of Graph Nodes . . . 92
3.5.1 Goal Nodes . . . 92
3.5.2 Focal Nodes . . . 92
3.5.3 Super Nodes . . . 93
3.6 Sources of Influence . . . 94
3.6.1 Deliberative Influence . . . 94
3.6.1.1 “Active” Focal Nodes . . . 95
3.6.1.2 Goal Nodes . . . 96
3.6.2 Sources of Reactive Influence . . . 96
3.7 Influence Propagation . . . 97
3.7.1 Propagation Techniques . . . 97
3.7.1.1 Reward Sharing Propagation . . . 98
3.7.1.2 Example . . . 99
3.7.1.3 Influence Propagation Through Super Nodes . . . 101
3.8 The Integrated Influence Landscape . . . 104
3.8.1 Mechanics for Combining Landscapes . . . 104
3.9 Implicit Contingency Planning . . . 105
3.9.1 The Nature of Contingencies . . . 105
3.9.2 Finding Contingencies . . . 106
3.9.3 Functional Equivalence of Focal Nodes . . . 107
3.10 The I2A Executive . . . 108
3.10.1 Action Choice . . . 108
3.10.1.1 Localised Expansion-bound A* Search . . . 109
3.10.2 Acting . . . 110
3.11 Summary . . . 110

**4 Evaluation - Functionality and Viability 112**
4.1 Results of Processing . . . 113
4.1.1 Logistics+ Problems . . . 113
4.1.1.1 Decoupling Worlds and Problems . . . 114
4.1.1.2 The Logistics+ Worlds . . . 115
4.1.2 Domain Preprocessing . . . 116
4.1.3 Clustering Analysis . . . 118
4.1.4 Time to Execution Analysis . . . 121
4.2 Unity Case Study . . . 122
4.2.1 Unity Overview . . . 123
4.2.1.1 GameObjects . . . 124
4.2.1.2 Component-based Architectures . . . 125
4.2.1.3 MonoBehaviours . . . 126
4.2.1.4 Editor Extensions . . . 126
4.2.2 Developing an Agent in Unity with I2A . . . 127

4.2.3 Creating a PDDL Game Representation . . . 129
4.2.3.1 The PDDLManager GameObject . . . 129
4.2.3.2 The PDDLObject Component . . . 130
4.2.3.3 Automated PDDL Generation . . . 131
4.2.4 Executing Actions . . . 134
4.2.5 Retrieving and Updating the Plan . . . 137
4.2.6 Execution Monitoring . . . 139
4.3 Summary . . . 140

**5 Evaluation - Robustness and Efficiency 142**
5.1 Worked Example . . . 143
5.1.1 An Example From Industry . . . 143
5.1.1.1 Movement in MGM . . . 144
5.1.1.2 Hazards and Items in MGM . . . 145
5.1.2 Algorithmic Example . . . 149
5.1.2.1 During Game Development . . . 149
5.1.2.2 During Runtime . . . 154
5.1.3 Summary . . . 158
5.2 Complexity Analysis . . . 160
5.2.1 Introduction to Computational Complexity . . . 161
5.2.2 Complexity Analysis of the I2A System . . . 162
5.2.2.1 Complexity During Development . . . 162
5.2.2.2 Complexity During Runtime . . . 167
5.2.3 Complexity Summary for Mountain Goat Mountain . . . 171
5.2.3.1 Generalising to Larger Problems . . . 173
5.2.3.2 Comparative Complexity of Alternate Techniques . . . 174
5.3 Overview of Algorithmic Analysis . . . 175

**6 Discussion 177**
6.1 Immediate Improvements . . . 177
6.1.1 Influence Propagation as a Vector Operation . . . 178
6.1.1.1 Worked Example . . . 178
6.1.2 Memory Constraints . . . 190
6.1.3 Extensibility . . . 191
6.2 Future Considerations . . . 192
6.2.1 Black Swans . . . 192
6.2.2 Procedural Content Generation . . . 196
6.2.3 Generation of Entities . . . 198
6.3 Parameters and Tuning . . . 199
6.4 Summary . . . 201

**7 Conclusions 202**
7.1 Final Thoughts . . . 205

**Bibliography 206**

# List of Figures


2.1 An example world to which A* path planning is well suited . . . . . . . . 26
2.2 An example planning graph . . . . . . . . . . . . . . . . . . . . . . . . . . 29
2.3 Demonstrating how landmarks can manage the combinatorial explosion of the search space . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
2.4 A Simple FSM for Guard Behaviour . . . . . . . . . . . . . . . . . . . . . 38
2.5 An example HFSM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
2.6 An Example of an Influence Map in Ms. Pac-Man . . . . . . . . . . . . . 47
2.7 Overview of a Perceptron . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
2.8 Example of a Game Tree in practice . . . . . . . . . . . . . . . . . . . . . 58
2.9 Diagram of a Subsumption Architecture . . . . . . . . . . . . . . . . . . . 64

3.1 Overview of the Integrated Influence Architecture Components . . . . . . 73
3.2 Layout of the Example Scenario . . . . . . . . . . . . . . . . . . . . . . . . 78
3.3 The Obvious Solution - Computed Upfront Forming the Initial Plan . . . 79
3.4 The Contingency Plan Provided by I2A . . . . . . . . . . . . . . . . . . . 80
3.5 An Alternative Contingency . . . . . . . . . . . . . . . . . . . . . . . . . . 82
3.6 A simple example of a DTG for a package within one city . . . . . . . . . 84
3.7 An example DTG for the truck . . . . . . . . . . . . . . . . . . . . . . . . 86
3.8 The Cartesian Product of the two DTGs shown . . . . . . . . . . . . . . . 87
3.9 A Basic Scenario for Reward Propagation . . . . . . . . . . . . . . . . . . 99
3.10 Influence Applies Directly to the Goal Node . . . . . . . . . . . . . . . . . 100
3.11 Results of a Single Propagation Step . . . . . . . . . . . . . . . . . . . . . 101
3.12 Completed Propagation of Influence . . . . . . . . . . . . . . . . . . . . . 102

4.1 The *Package1* DTG from World 1 . . . . . . . . . . . . . . . . . . . . . . . 117
4.2 An in-application visualisation of the *Package1* DTG from World 1 . . . . 118
4.3 The *Package1* DTG from World 2 . . . . . . . . . . . . . . . . . . . . . . . 120
4.4 The PDDLManager Component . . . . . . . . . . . . . . . . . . . . . . . . 130
4.5 An example of a PDDLObject-enabled GameObject . . . . . . . . . . . . . 131
4.6 An Example of a Walkable NavMesh . . . . . . . . . . . . . . . . . . . . . 133
4.7 Tool for PDDL Generation from NavMesh . . . . . . . . . . . . . . . . . . 134
4.8 Results of NavMesh extraction as shown using debug tools ingame . . . . 135
4.9 PDDLObjects registered with the PDDLManager . . . . . . . . . . . . . . 136
4.10 An example plan loaded into the PDDLManager . . . . . . . . . . . . . . 137

5.1 The Mountain Goat Mountain world, with tile locations picked out in purple. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146
5.2 Uphill connections between tiles in MGM. . . . . . . . . . . . . . . . . . . 147

5.3 Additional traversable connections when using the “Super Goat” ability. . 148
5.4 Downhill connections between tiles . . . . . . . . . . . . . . . . . . . . . . 149
5.5 Recap of the Integrated Influence Architecture Components . . . . . . . . 150
5.6 Data Flow of the Develop-time Components of the I2A . . . . . . . . . . . 151
5.7 The MGM world shown from a top-down perspective, with height annotations for each tile. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 152
5.8 Data Flow of the Runtime Components of the I2A . . . . . . . . . . . . . 155

6.1 The Basic Propagation Example - Node 1 is a Goal Node . . . . . . . . . . 179
6.2 An example from Stampede Run, each section of path is laid out procedurally and many are reused . . . . . . . . . . . . . . . . . . . . . . . . . 197

*To my parents,*

*the finest role models I could have asked for.*

*And to Heather,*
*my producer and my heart.*

# Chapter 1

# Introduction

Artificial Intelligence (AI) is the name given to the field of scientific study that deals with automated decision making. Whenever a system makes a decision on its own, independent of human oversight, it is acting with Artificial Intelligence. This doesn't reflect the quality of the decision being made, as AI systems are still capable of making bad choices - the fact that they made a choice at all is what makes them AI. Typically we refer to such a system as an "Agent", and the standard definition calls for it to be able to perceive its environment using sensors, and make changes to that environment using effectors[1]. These sensors and effectors could be as simple as a small piece of code that can read the contents of a file, and a small piece of code that can create files; the agent here would be able to make the new files dependent on the contents detected, which although rudimentary qualifies as an intelligent, automated system. This definition holds for much more complicated systems as well, for example the Mars Rover which is capable of sensing a great deal of information about the environment it is in and then using effectors both to move around that environment and also to interact with it to undertake scientific experiments.

The promise of Artificial Intelligence is frequently shown in Science Fiction - an automated workforce and sophisticated personal assistants, able to adequately adapt and perform duties in changing circumstances with only high-level instruction, but today one of the most prevalent application areas for this kind of technology is in the world of play, and specifically Video Games.

## 1.1 A History of Artificial Intelligence in Video Games

For as long as there have been games, people have had a desire to play them with fewer people than intended. A relatively modern example of this can be found in Bridge, the earliest reference to which seems to be from 1886 (although as with many such things, the context makes it clear that the game was played extensively as a variant of Whist). Traditionally a four-player game, by 1922 three player variants of the game were becoming common enough for their popularity to be commented on in newspapers[2]. More classically, consider Chess. This game began in the 6th century and has since evolved into the game we know today, with a broadly recognisable (by today's standards) version emerging in the 15th century. Of significance however is Wolfgang von Kempelen's "Automaton Chess Player", more commonly referred to today as the "Turk".

The Turk was promoted as a fully automated chess player, built in 1770 to widespread acclaim. Such was the interest in the device that it was taken on regular tours around Europe and played exhibition matches against world leaders and dignitaries. It would be fifty years before the device was revealed to be a hoax, being operated by a person hidden inside the cosmetic mechanisms within. Although not in any way artificially intelligent, the Turk remains an important part of Artificial Intelligence in the area of games, since it highlights clearly how much, even over three centuries ago, people wanted

to have automated players to compete against. Unfortunately it would be a long time before this could be realised, with the creation of "El Ajedrecista" by Leonardo Torres y Quevedo in 1912[3].

### 1.1.1 The First Artificially Intelligent Games

El Ajedrecista is referred to as the first "computer game", but more than anything it is a milestone in Artificial Intelligence for games. It was an automated player capable of playing "KRK" games, which is to say games in which one player controlled a King and a Rook, whilst the other player had a single King. Torres made a number of assumptions and simplifications that reduced the complexity even further, but regardless, this machine was still the first of its kind, capable of playing games independent of human control and providing people with the first glimpse of the single player games that were to follow, using algorithms to replace human ingenuity and demonstrating the first true instance of Artificial Intelligence in games.

As El Ajedrecista was being demonstrated, elsewhere the "arcade" was beginning to catch the public's imagination. Initially an evolution of carnival entertainment, what were previously sideshows manned by attendants began to be replaced by machines that allowed players to take part in skill-based contests. Ball-tossing and shooting galleries are two prime examples of this style of game, where the mechanisms controlled the enforcement of the rules of the game, resetting the field at the start of a new session but were not implicitly making decisions, and therefore not artificially intelligent. These machines too evolved, giving way to pinball machines and other "electro-mechanical" devices that allowed more and more diverse scenarios and gameplay elements, but still in a constrained manner. Pong (Atari, 1972) and Space Invaders (Taito, 1978) were the earliest successes from the area of video games, but even these were still run according to

rules that didn't alter dependent on the player. Arguably the first video game to make use of AI techniques was Pac-Man (Namco, 1980). Although a deterministic game, each of the enemies in the game is imbued with its own personality, and the reactions they make to the player's actions reflect that there is a level of decision making[4]. If not the very first video game to exhibit this kind of feature, it certainly was the first commercial hit to do so.

### 1.1.2 Artificial Intelligence in Contemporary Video Games

In the games industry, it is worth highlighting how little processing power is available for AI systems. Most modern games are designed to run at 60 frames per second[5], which is to say that the screen will be updated 60 times each second. Typically this means that the state of the game world will be updated as required 60 times a second as well, although there are certain techniques that allow this to occur at a different frequency. In general then, each frame update can take only 16 milliseconds, and in that time a significant portion of the processing power available will be dedicated to other tasks such a graphics rendering, physics simulations and user interface. As a result, it is estimated that AI routines might get a maximum of 10%[1] of the available processing power, or in real terms about 1.6 milliseconds per frame update. This is in stark contrast to what is considered typical in the academic AI discipline, where for example, the bi-annual Automated Planning Competition gives participants 7 gigabytes of RAM and a half hour per problem[6].

As a result of this necessity for light-weight approaches, many industrial applications of AI in games tend to be somewhat rudimentary. This has been generally accepted as "good enough" by the industry, because they are able to work around it to some extent

[1]Personal communication with Chris Preston, then Engineering Manager at Ubisoft Reflections

since they are creating the simulated world that the agents inhabit. It is less computationally expensive to skew the rules of that world than it is to create a "scientifically competent" AI system capable of acting truly intelligently in these worlds.

It is also worth noting that in many cases, the "smoke and mirrors" approach to AI in games is seen by the industry as acceptable. The objective for these companies is to make an entertaining product for the players, and a scientific approach to creating the AI systems does not always respect this. Consider even a simple game such as Chess, which has had significant research targeted at it. Currently IBM's Deep Blue is arguably the most high profile AI system for Chess having a significant ability to play at a high level, as proven when it defeated Garry Kasparov, a Grandmaster of the game and at the time World Champion[7]. For all this success, Deep Blue has not gone on to be a successful product for IBM, and there does not seem to a widespread consumer desire to play against this type of exceptionally strong "intelligent" chess system.

Although it is important to recognise that the games industry is much more sales-oriented than academia, and consequently is much less reliant on advances in the state of AI, there are arguments to be made in favour of being able to incorporate better AI into games than is currently available. Foremost among these is that different circumstances require different types of reasoning. In a presentation at the Paris Game AI Conference DICE's Mikael Hedberg stated that the average lifespan of their AI controlled, or Non-Player Characters (NPC) was just five seconds, and in that timeframe there is not much scope for a player to see past the illusion of intelligence[8] - but that is just one game in one genre. In fact it could be argued that the level of AI in use in these kind of games becomes a cycle of reinforcement, where the fact that many game developers choose to rely on such fragile approaches to their character AI, creates a situation where these same developers aren't able to push beyond this to develop characters more fully. This

could explain the large number of games that employ the "NPC as canon fodder" cliché, since conventional wisdom and the current state of the art does not allow for much richer decision making and instead relies on tight control over the AI systems and agents in order to create specific cinematic experiences, often involving set-piece elements over which the developers have strong directorial control.

That of course is not to say that all games approach their AI this way, and one notable exception to this is F.E.A.R. (Monolith Productions, 2005) which used Goal Oriented Action Planning (GOAP) (a technique which will be explained in greater detail in the next chapter). The broad premise of GOAP is that it allows NPCs to perform some basic reasoning about their environments and make plans[9]. As one of the first mainstream industry applications of more advanced reasoning, it was a major step forwards but noticeably this alternative approach to AI struck a chord with many players, and is still often cited by players as being an example of good Game AI[10]. More recently Just Cause 2 (Avalanche Studios, 2010), which had a much more of a sandbox and open-world approach to its gameplay, also included an AI system derived from the GOAP technique, and again it resonated very well with players as the NPCs in the game world behaved in interesting and in many cases rational ways despite the fact that with such a varied number of game play approaches, the exact behaviour of the player could not have been predicted. This highlights that there is a demand for games with much stronger AI where the player can feel that they are truly interacting with characters able to make rational decisions, rather than ones whose sole purpose is to provide target practice.

Nowhere is this more evident than in large-scale Role Playing Games (RPG) such as Elder Scrolls V: Skyrim (Bethesda Game Studios, 2011) or Mass Effect (Bioware, 2007), in which there are a selection of enemies to be killed, but also a range of characters that will exist throughout the duration of the game, which in some cases can extend beyond

80 hours of content to explore. These may be the inhabitants of villages that the player will encounter, who are going about their daily lives - or some abstract representation of it - but by far the most important are the companion characters who travel with the player character. Many of these RPG games adopt a party-style approach where the player gets AI controlled teammates who not only contribute during combat but also are an integral part of the story being told and the decisions that the player is making[11]. These characters are often driven by the same basic architectures powering the enemies in combat - meaning that they are often heavily driven by designers, and considered good enough to exhibit intelligence for a short period of time. They are notoriously perceived as stupid by players, and this is a frequent sticking point in reviews and anecdotes. Skyrim in particular exhibited a number of problems with the way that NPCs were controlled. During combat, they would perceive certain spells cast by the player, notably those that caused damage to anything within a certain radius, as a hostile action - even when the clear target was a common foe. This would often result in the NPCs ignoring this foe and attacking the player - an irrational response. Another source of criticism of Skyrim focused on an emergent gameplay issue with the merchants; the game allows players to steal items from shops, and uses a line of sight check from the merchant to determine if you are "caught", and if so the guards in the town are alerted. The game also allows for physics-based interaction with the world, picking up objects and placing them down. Players soon realised that by using this method of interaction, they could use buckets and place them over the merchants' heads, thus effectively blindfolding them and allowing thievery to occur unchecked[12].

This example of player innovation highlights that any attempt to design an NPC's responses to player actions is by its nature doomed – the development team cannot hope to exhaustively enumerate all the ways that the players will interact with the

world and the NPC, so the reliance on the accepted smoke and mirrors wisdom rapidly breaks down in the face of NPCs having to do anything more rigorous than run onto a battlefield, get shot and fall over. It's clear that this can have a major impact on public perception of games – Skyrim's faults have been expounded on at length in gaming media, whilst games such as F.E.A.R. that allow for novel solutions to be derived by the agent on the fly have been praised.

Despite many industry veterans' claims that the current state of Game AI is sufficient, there is clearly scope for broad improvements to be made, as well as a strong desire for those improvements in the audience for the products. As such, more robust and capable AI systems can be developed that will have relevance and impact on the industry if adopted.

## 1.2 Motivation

When we speak about possible improvements in Game AI, we have two conflicting problems. Firstly there is the need for better reasoning systems and secondly there is the need for the reasoning systems to be computationally simplistic to retain the speed of operation needed in the context of games. Broadly, prior approaches to AI, both in the games domain and more generally, have treated each of these as very distinct problems which have been tackled by two different AI paradigms, Reactive and Deliberative Systems respectively.

### 1.2.1 Reactive Systems

Reactive Systems is a catch-all term used to describe a family of AI techniques that are designed for very quick, responsive decision making. The biological model for this

kind of system can be thought of as the instinctive responses we have to certain stimuli, such as flinching away from sources of pain. This happens below any level of conscious thought. In much the same way, Reactive AI approaches are designed to quickly map a stimulus to a reaction, without involving any sort of reasoning. A common example of this is a Subsumption Architecture[13], which will be examined in detail in Section 2.3.1, but resembles a prioritised mapping of responses to certain conditions. Coupled with a default behaviour, it is easy to use this kind of system to create "intelligent" autonomous behaviour. Imagine a small robot with a default action of moving forward, but a response mapping that when it detects a collision, it should back away and turn. With this very simple combination, it is easy to see that the robot is now capable of ensuring that it is always in motion, and of avoiding obstacles.

What is less clear is the way that this system can achieve objectives that require long term reasoning. The response mappings are not stateful (meaning they have no history of prior states that the agent has been in), and often when concepts of state are introduced, the advantages of the Reactive System paradigm are lost - most frequently the speed with which sense can be translated into action[14]. As a result, architectures based solely on this model tend to be very good at taking actions that ensure certain conditions are maintained or avoided provided that this can be solved by a single action, rather than requiring a sequence of steps be executed over time.

Consider a puzzle in which it is necessary to slide small squares around in order to re-assemble a picture. This sort of problem can be solved using a strategy that understands where each piece needs to end up, and then works backwards from there to establish the process by which the current state of the puzzle can be transformed into the desired state. A Reactive System is ill suited for this, as the only thing that can be sensed by the system is that the pieces are not in the correct position, and the only response

possible is to move a piece into a new position - probably still incorrect. As a result, the Reactive System cannot "see" the solution as it requires multiple steps, but would be capable of executing such a solution that has been provided by some other system as a "recipe" to follow.

### 1.2.2 Deliberative Systems

In contrast, Deliberative Systems are much better suited to this kind of long term reasoning about sequences of actions. The broad philosophy of Deliberation is to draw in as much information about the state of the world as possible, and perform analysis in order to find a method of achieving a desired result. This is often a computationally expensive process, and as such is much slower than a Reactive System, but the benefit is that it can provide very intricate sequences of steps required to find the result. A Deliberative System is very well equipped for solving the sliding picture puzzle described above. Providing it with a description of the initial state of the puzzle and the desired state at the end, along with a description of how the pieces can be manipulated (in other words, the actions that can be performed in this problem), a Deliberative System can use this information to simulate performing those actions, and understand how they have affected the world. As a result, this kind of architecture is able - whether through trial and error, or more sophisticated means - to produce a solution for the puzzle.

Deliberative Systems are used extensively in AI for board games, such as Chess and Go[15] to provide state of the art AI approaches to game playing. Early Deliberative Systems such as those built on the MiniMax algorithm[16] relied on exhaustive analysis of the state space to discover optimal strategies of play, whilst other techniques such as Monte Carlo Tree Search[17] use intelligent sampling to get a sense for the state space, exploring new areas whilst trying to find optima in areas previously found to be

promising. Outside of games, techniques like Automated Planning rely on guided search and heuristic estimation to shape a path through the state space towards solutions, but in all these cases, the emphasis is on assessing all the options available at a given choice point and finding the one that provides the best outcome in the long-term view[18].

### 1.2.3 The Necessity to Bridge the Gap

These two contrasting paradigms provide the two ends of the AI application spectrum; domains that require very rapid decision-making but which do not have long-term components that require sequences of actions to achieve. At the other extreme, we have a more logic-centric approach that is not concerned about any kind of time constraints. These contrasting characteristics make for a very good theoretical demarcation, but unfortunately, not all real world problems fall neatly into one or other classification. In fact, a range of applications for AI techniques fall in between the two - requiring both long term reasoning in order to satisfy some goals and short term reaction to account for aspects of the world that cannot be reasoned about.

What is needed to deal with this classification of problem is something that takes the best elements of both Reactive and Deliberative systems and combines them in some way to create a system that is able to provide strong long-term reasoning quickly enough that it can be of use alongside a game engine, where the majority of the computational power is being dedicated to managing the game and rendering it to the screen for the player.

## 1.3 Research Statement

### 1.3.1 Concept

The work described here seeks to address the disconnect between Reactive and Deliberative Systems in a more subtle way than has previously been attempted, creating an efficient way to build agents that can make use of the benefits of both styles of reasoning.

The contrast between the the two has previously been summed up by the following phrase:

"When someone accidentally puts their hand on a hot stove, the hand is removed before conscious consideration of whether this action is a good idea".

In this formulation the Reaction and the Deliberation are seen as two wholly independent processes and there is no need to undertake a complicated and computationally complex approach to address the fact that the hand is burning.

This arbitrary divide between the two approaches neglects to consider that there is actually value in guiding reaction in the context of prior deliberation. To put this another way, when a quick reaction is required, there are likely several different forms that reaction might take, some of which might better serve the longer term goals that a Deliberative system has set. In this way, the prior phrase might be altered in the following way, which serves as the conceptual framing for the research presented here:

"When someone accidentally puts their hand on a hot stove, the hand is removed before conscious consideration of whether this action is a good idea, but it is preferable to move it in the direction of the first aid kit."

It is the core hypothesis of this work that an architecture derived from this conceptual model will be demonstrably more robust and efficient than those that are either purely Reactive or purely Deliberative. Domains that benefit from this type of reasoning include the specific use-case of providing decision logic for characters in virtual worlds such as those found in video games. Due to the focus on this application area it is also essential that any architecture proposed be viable for use in an industry setting.

### 1.3.2 Research Process

In order to prove the hypothesis, a new architecture must be developed that adequately combines key elements of both the Reactive and Deliberative paradigms. This requires a deep understanding of the current state of the art in order to appropriately build on past techniques. As a result of this understanding, the research has been broadly guided by three fundamental principles as follows:

- It is relatively well accepted that algorithms which use search as a technique (such as many Deliberative systems) are much more computationally expensive than algorithms used for function evaluation and validation. As a consequence, minimising reliance on search strategies would seem prudent.
- The area of path finding in games is relatively well explored in literature, and techniques exist to diminish reliance on search in this type of problem which may have applicability.
- Games present an interesting opportunity in that there are significant resources available during the development of the game when compared to the execution of the game.

These key insights provide the framing for the literature review and the basis for the development of a proposed architecture capable of testing the hypothesis.

### 1.3.3 Applicability

It should be noted that such an architecture will not be applicable for every game. Above, it was noted that reactive and deliberative solutions address two specific classes of problem, and equally the converse will be true in that the proposed architecture will only be suitable for use in scenarios that combine elements of both. There is no expectation that the proposed solution be able to outperform a purely deliberative approach being applied to a purely deliberative problem, or equally a reactive solution to a reactive problem. The key to the applicability of a game to the kind of solution being proposed is that it should combine elements of deliberation and reaction; it must have some degree of long term objective that can be reasoned about whilst at the same time having dynamic elements that will force the agent to deviate from its planned approach to the objective or otherwise reassess its reasoning. Games without this combination our considered beyond the scope of the architecture proposed.

### 1.3.4 Evaluation

It is necessary to conduct a thorough evaluation of the proposed architecture in order to validate the hypothesis that is superior to prior techniques. There are a number of criteria that must be satisfied in order to demonstrate this.

- **Functionality** - It must be functional, which is to say that it must be able to actually make decisions informed by both Reaction and Deliberation in dynamic environments.

- **Viability** - It must be applicable to industry. One of the core motivations of the work is addressing a shortcoming in contemporary solutions in use by industry, so any proposed solution must in turn be able to address that shortcoming in the industry directly.
- **Robustness** - It must be able to handle dynamic environments by reacting to threats and opportunities in a timely manner
- **Efficiency**- It must demonstrate computational efficiency against deliberative solutions

The first of these will be proven by demonstrating its applicability in a range of scenarios. To support the second requirement, it will be shown that the technique can be integrated into a modern game development toolkit. Finally, the third and fourth requirements will be fulfilled by presenting a worked example of a game closely related to a real project from industry and demonstrating the capabilities of the architecture for this style of game.

These four criteria will sufficiently support the hypothesis, although it is outside the scope of the research to exhaustively demonstrate the architecture's superiority. Most notably to this point, the intended approach is sufficiently modular and adaptable, and a thorough exploration of configurations of the architecture and its performance under differing circumstances will not be tackled as part of this work.

### 1.3.5 Summary

To summarise, this work postulates that there is value in thinking about Reaction and Deliberation differently, and that it is possible to create an architecture that is informed

by both methods simultaneously. Such a system should blend the best of both approaches, taking the computational efficiency of the Reactive System and pairing it with the robust long term reasoning of a Deliberative System. In order to prove this theory, an architecture has been created, referred to as the Integrated Influence Architecture (I2A), which is built on these principles. The research presented in this thesis will describe the manner in which it works, as well as demonstrating that such a system provides a notable improvement over contemporary techniques by providing efficient but strong decision making.

## 1.4 Papers and Presentations Used

This thesis includes an extensive amount of work that has already been published, both in academic conferences and journals, as well as through more industry-facing channels.

- At the 11th International Conference on Intelligent Data Engineering and Automated Learning Conference 2010, work from this project was published that explained in detail the clustering algorithm used to create abstract representations of state spaces[19].
- An overview of the creation of Influence Landscapes was presented in a paper at the Society for Artificial Intelligence and Simulated Behaviour's Symposium on Artificial Intelligence and Games 2011[20].
- In 2012, the savings obtained by using a clustered abstraction approach to planning was demonstrated in an article published in the International Journal of Data Mining, Modeling and Management[21].

- The work was featured as an invited seminar given at Lockheed Martin's Burlington MA campus in July 2012, and was presented at the AltDevConf held in 2012[22]. It was featured as part of the the Game Developers Conference in the Artificial Intelligence Summit in March of 2013[23].

## 1.5 Thesis Layout

The remainder of the thesis is structured as follows. Chapter 2 will provide an overview of relevant techniques and approaches to AI systems that have been in use in the video game field as well as elsewhere. In Chapter 3, the specific method proposed for the I2A will be shown and explained thoroughly. To address the evaluation criteria outlined above, results of experimentation proving the functionality of this approach will be presented in Chapter 4 along with a proof of viability by implementation in the popular game engine Unity 3D. Chapter 5 will demonstrate robustness by outlinging an example game from industry to showcase the manner in which the components interoperate and the flow of information between them and will also address efficiency by presenting an analysis of the computational complexity involved in each component. Discussion of the architecture, possible extensions and future enhancements will be presented in Chapter 6, whilst Chapter 7 will give a summary and draw conclusions from the work.

# Chapter 2

# Related Work

The Integrated Influence Architecture draws inspiration from a number of different techniques in use both within the games industry currently as well as those that are more academic or that have previously not been seen within games. In this chapter, a review of relevant previous work has been undertaken to fully contextualise the I2A, not only in terms of its components but also the shortcomings of contemporary approaches that necessitate a system like the I2A. As the majority of the system is motivated by, and built on top of, Automated Planning, Section 2.1 takes an intensive look at this discipline. Section 2.2 discusses techniques within the games industry, with specific emphasis on those techniques in use currently and their perceived shortcoming. In Section 2.3, other techniques that are of relevance but not specifically related to Automated Planning or Games are discussed. These are predominantly executive systems that translate a decision from a logic system into an action in the world. The chapter closes by providing a comparative analysis between the I2A and several key works that were introduced in earlier parts of the chapter. This analysis can be found in Section 2.4.

## 2.1 Automated Planning

Automated Planning (or "planning") is the name given to a branch of AI in which a problem is attempted to be solved by creating a concrete list of actions that must be executed in sequence (a "plan") in order to alter the situation that an agent finds itself in, in order to achieve some desire outcome.[1]

Planning is a deliberative AI technique, which is to say that it performs reasoning, taking all the information it knows about the environment into consideration in order to produce a coherent approach to solving the problem.[18]

### 2.1.1 Problem Representation

A planning problem is typically specified in terms of three things.

- A complete, fully enumerated description of the current state of the world.
- A description of the possible actions that can be taken within that world.
- A partial description outlining what the final objective of the plan should be.

The description of the goal is typically partial because only certain elements are important - consider this in terms of any real world task. Rarely is the goal state exhaustively enumerated, but instead it is generally considered in terms of a small number of conditions being met. Any state that meets these is a potential goal state, and the remainder of the world can be disregarded. In a package delivery problem, the goal is that the package is in the right place - the configuration of the rest of the world is irrelevant, therefore any state where the package is in the right place is a goal state. The same holds for most planning problems, where there are only a small number of salient things

describing a goal. The objective of planning is to find a sequence of actions that reach a goal state, and typically an optimal solution will be that one that finds the goal state using the shortest path.

On the other hand, the initial world state specification needs to be fully exhaustive. This is used by the planning agent to determine the resources available to it, as well as the configuration that all objects within the world are in when the resulting plan begins execution. Note that in general the assumption is that these descriptions are accurate, and the world behaves in a deterministic and predictable manner. An overview of ways that this assumption can be overcome is detailed in Section 2.1.4.1.

In order to be easily parsed by a planning agent, the problem representation needs to be described formally in a machine readable format. Generally, one of two standard formalisms are used for this, either the Planning Domain Description Language (PDDL) or Simple Action Structures + (SAS+), although there is no requirement for this and other languages can be used to describe these problems.

#### 2.1.1.1 PDDL

PDDL is the de facto standard for the Automated Planning community, and was first defined in 1998 ahead of the AIPS Planning Systems Competition as a standard representation that could be used in order to allow planning algorithms to work on the same problem set, and thereby be comparable.[24]

As a language based on first order predicate calculus, PDDL provides a rigorous grammar by which the nature of a planning domain can be represented. PDDL uses a two-component approach, decomposing a problem into a Domain specification, which describes the kind of environment that the agent will be acting in, including the kinds

of objects that it might encounter, and a Problem specification which explicitly sets out what objects exist in the world, their types and how they are initially arranged, as well as the goal of the planning task being described.

PDDL represents the world as a set of facts. Facts that are not asserted are assumed to be false. PDDL solves the “Frame Problem”[25] by making use of Reiter’s “Successor State Axiom”, which is to say that facts that are asserted true at one time point will remain true at all subsequent time points unless explicitly negated.[26]

PDDL utilises a predicate-based or “lifted” approach to formalism, meaning that it is defined in context of variables that can later be quantified. This means that the world can be described using a framework such as $at(var1, var2)$ with the variables being substituted in later. This is in contrast to a “grounded” representation, in which every permutation of the $at(var1, var2)$ predicate must be enumerated, leading to exceptionally large world descriptions. This lifted approach means that the representation can be abstracted, and PDDL supports variable typing to constrain this even further. However, it should be noted that this approach is largely a convenience for the human creating the specification - when solving the problem many planners will perform variable substitution in order to create a grounded representation internally.

Since its inception, PDDL has undergone a number of revisions that have added additional features. By far the most adopted is PDDL 2.1, which was introduced in 2002. The primary contributions of this revision to the standard were “durative actions”, which allowed actions to be represented as having a temporal length associated with them, where previously they were assumed to be instantaneous. This allowed planners to search for shortest-time plans, and distinguish that although a plan might have more steps involved, it would complete quicker. PDDL 2.1 also brought in “numeric fluents”

which allowed numeric values to be tracked such as fuel consumed by performing actions, and "plan metrics" which allowed a planning problem to try to find a plan that minimised the value of these fluents.[27] Subsequent revisions introduced "timed initial literals", which is to say actions that occur at specific times independent of plan execution (PDDL 2.2 [28]), "preferences" which allow a problem to specify soft constraints that planners should try to conform to if possible (PDDL 3.0 [29]). PDDL 3.1 would have introduced "object fluents", allowing fluents to not only take numerical values as before, but also an object-type, however this work does not appear to have ever been adopted as part of the official PDDL specification.[30]

PDDL has also spawned a number of successor languages that have attempted to tackle either a much broader set of planning problems than the PDDL specifications allow for, or have seen the concepts of PDDL adapted for a more niche application. Two examples of the latter are the New Domain Definition Language (NDDL)[31] and the Multi-Agent Planning Language (MAPL)[32], which fall outwith the scope of this work. Of more relevance are those languages that extend the PDDL specification to be more expressive. One prime example of this is PDDL+[33] which uses autonomous "processes" and "events" to attempt to produce a model of the world that was more accurate and correctly portrayed actions as affecting continuous change on the world (as opposed to PDDL 2.1's model in which actions did not affect change except at their initiation and termination). This meant that for example a bucket being filled over time would experience a constant change in volume over time, rather than a single discrete update when the filling action completes. Although this enhanced the accuracy of the model, it also increased the computational complexity of algorithms tackling problems specified in this way. As a result, PDDL+ failed to gather much traction within the planning community, as evidenced by it never being adopted by the International Planning Competition.

#### 2.1.1.2 SAS+

The approach that PDDL takes to its representation is motivated by having a lot of statements about the world and each one being either true or false. This means that the PDDL formalism has a “high dimensionality” but since each dimension can only take one of two values, it is a “low value” representation. This is an approach that is well suited for certain search strategies - there are a lot of switches to change, but as they can only be on or off it is easy to see whether changing one has had a positive or negative effect. SAS+ is conceptually very different in that its approach is “low dimension / high value” - almost opposite to PDDL. This means that there are a smaller number of switches, but rather than on/off, they are selectors across a number of choices. Changing one value here is less informative - it might have changed to a worse value, but that doesn’t mean that the previous value was the optimal. This alternative representation is less well matched to some of the earlier search algorithms used in planning, but when paired with an appropriate algorithm it can be an effective strategy.

More formally, SAS+ makes use of a sequence of multi-valued variables to represent what in PDDL would be a set of implicitly linked facts.[34] A good example of this alternative approach is to consider a scenario in which a package can be in one of several locations. In PDDL, this would be represented as a set of facts, and perhaps a move action that simultaneously deletes the proposition that the package is at its current location whilst also asserting the fact reflecting its new location. In this way, PDDL will ensure that only one of these facts is true at any one time, although from a short inspection of the facts themselves, there is no indication that they are linked.

Under the SAS+ formalism, such things that are implicitly represented in PDDL become explicit. A SAS+ representation of the example would make use of a single variable,

whose value could represent any of the locations that the package could be at. SAS+ not only captures the specific values that the variable can take, but also the manner in which it transitions between those values. Consider if the package can only be moved to adjacent locations, then from Location 2, it could move to Location 1 or Location 3. This is represented in SAS+ as a "Domain Transition Graph" (DTG) and plays a significant role in the I2A as shall be explained in Chapter 3.

SAS+ also represents the interdependency of types of object within the world, by making use of the "Causal Graph" (CG). This shows which objects rely on others, so captures the manner in which the respective DTGs for each variable might be interleaved.

**Generating SAS+ From PDDL** It has been shown to be possible to automatically generate SAS+ from PDDL.[35] Naively it is possible to treat each proposition in the PDDL as a binary valued variable, but this does not allow for any of the alternative formalism to be leveraged. By analysis, it is possible to collect the mutually exclusive propositions in a PDDL domain and turn these into variables with associated DTGs. The CG can be found by analysis of PDDL actions; the types of object that appear in the effects of an action are dependent on the objects that appear in the preconditions for that action.

It should be noted that this automatic generation process does not necessarily retain a semantic meaning that is easily observed for non-trivial domains. In particular, the naming of variables and values loses any sense of the orginal meaning of these in the PDDL.

### 2.1.2 Classical Planning

So called "Classical Planning" first came to the fore in the early 70s with techniques such as the Stanford Research Institute Problem Solver (STRIPS). For their time these were ground breaking and represented a new way of looking at problem solving in real world scenarios, or as the title of the original STRIPS paper indicates "A New Approach to the Application of Theorem Proving to Problem Solving".[36]

The STRIPS language was a very simple arrangement of logical operators, structured such that an action was defined in terms of the preconditions that needed to be satisfied, and the effect of applying the action. In this it is clear that later formalisms such as PDDL were heavily influenced by STRIPS, and its position as the initial precursor to the Automated Planning field is unquestionable, as evidenced by its longevity in literature and courses over forty years later.

Naively, it should be noted that a planning problem can be solved by taking the initial state and applying all the actions whose preconditions are currently satisfied to generate a set of successor world states, and continue with this process down a tree of states until a state is found that matches the goal. This is an example of a "forwards-chain" search, but also one that is unguided, meaning that no information is being provided to focus the algorithm on promising areas.

The STRIPS solver works in a more robust manner, using "backwards-chain" or "regression" planning.[1] This means that the solver starts with the goals that must be satisfied and finds actions that would achieve them as part of the action's effects. The relevant effect is replaced by the unsatisfied preconditions of that action, the action is added to the front of the plan and the process recurses until the initial state is reached.

#### 2.1.2.1 Heuristic Search and A*

In contrast to unguided search, a "Heuristic Search" or "Heuristic-guided Search" uses the notion of a heuristic (or estimate) to guide it. The A* algorithm is perhaps one of the best known heuristic search algorithms[37] and is typically used for pathfinding in well-behaved worlds.[38] Imagine a tile-based world with obstructions. The aim is to find the shortest path from point A to point B, moving around the obstacles. An example of this world is shown in Figure 2.1, where the goal and starting point are represented by the diamond and triangle.

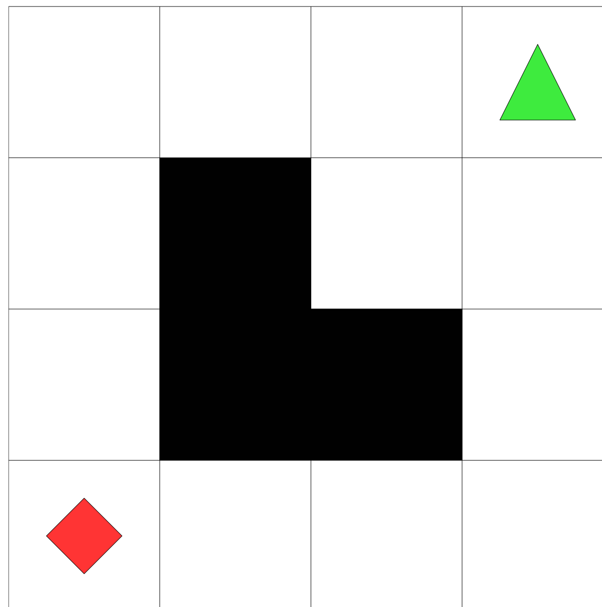

FIGURE 2.1: An example world to which A* path planning is well suited

The algorithm starts from point A and works towards point B by considering the states that can be reached from the current point. Each one is scored in such a way that an estimate of the distance of the path traversing that point is obtained. At all times, the next node to consider will be the node that is currently the lowest scoring that has not been expanded.

In order to calculate this score there are two components, the distance the agent has

travelled so far, and the remaining distance. The cost of the path to reach the current node from Point A is known - this is the number of layers traversed down the tree so far, however the remaining length to the goal is unknown, and calculating it exactly would be computationally complex. Instead the remaining cost is estimated using a heuristic, which is generally analogous to a "good rule of thumb". For A* in the kind of scenario shown, this is often the Manhattan Distance between the current location being considered and point B, meaning the summation of the absolute difference in the x and y coordinates. A good heuristic should be relatively accurate so as to properly inform the search, but it also must be computationally efficient. A good heuristic will also often be what is termed "admissible", which is to say that it will never generate a heuristic value larger than the correct value. Put another way, it will never overestimate the work remaining, although will often underestimate, and here the Manhattan Distance is demonstrably admissible as a location in a grid system cannot ever be closer than the difference in its x and y coordinates from the current position.

In this way, heuristics are used to guide a search and suggest areas of the tree that seem promising in a strategy often referred to as "Best-First Search" (contrasting with Breadth- and Depth-First techniques). This is used to good effect in planners, with the Fast Forwards system being particularly effective and discussed in Section 2.1.3.1. An alternative approach to heuristic guided search, based on state space sampling, will be introduced in Section 2.2.3.3.

### 2.1.3 Contemporary Planning - Planning Graphs and DTGs

As planning evolved and emerged as a standalone discipline within Artificial Intelligence, it became clear that there were cases for both backwards and forwards chained search. STRIPS had originally used backwards chaining in order to control the size of the state

space, but it was evident that this had limited value, and that an alternative approach was required. Forwards-chained search became a viable alternative as more research was undertaken, and the nature of planning problems was better understood.

One such advance was the representation of the problem as a “planning graph”. When considering a planning problem previously, the conventional wisdom was to visualise it as a series of states connected by actions. The actions would change what facts were true, and the states were effectively an encapsulation of the facts true at any given point. This meant that the problem was being perceived as one of identifying a set of state transitions which would traverse this representation. This was true regardless of formalism - both PDDL and SAS+ (and others) used this as a conceptual starting point. The Graphplan system turned this on its head and instead used the “planning graph” as its conceptual start point.[39]

The planning graph represents both actions and facts as nodes within the graph, with these being structured as alternating layers of facts and actions. The initial layer of the graph represents the facts true in the initial state, whilst the subsequent layer includes all those actions that can be applied from this state. Edges within the graph represent causality, so actions are connected to the facts in the subsequent layer that are altered as a consequence of the action, and also to the facts in the preceding layer that they are contingent on as part of their pre-conditions. An example of this structure is shown in Figure 2.2.

Using this structure, Graphplan is able to extend the planning graph until a layer is reached in which the goal facts are concurrently true. Then it is a relatively simple process to work backwards from those facts to find the actions which cause them to be true, and so on until a plan is formed.

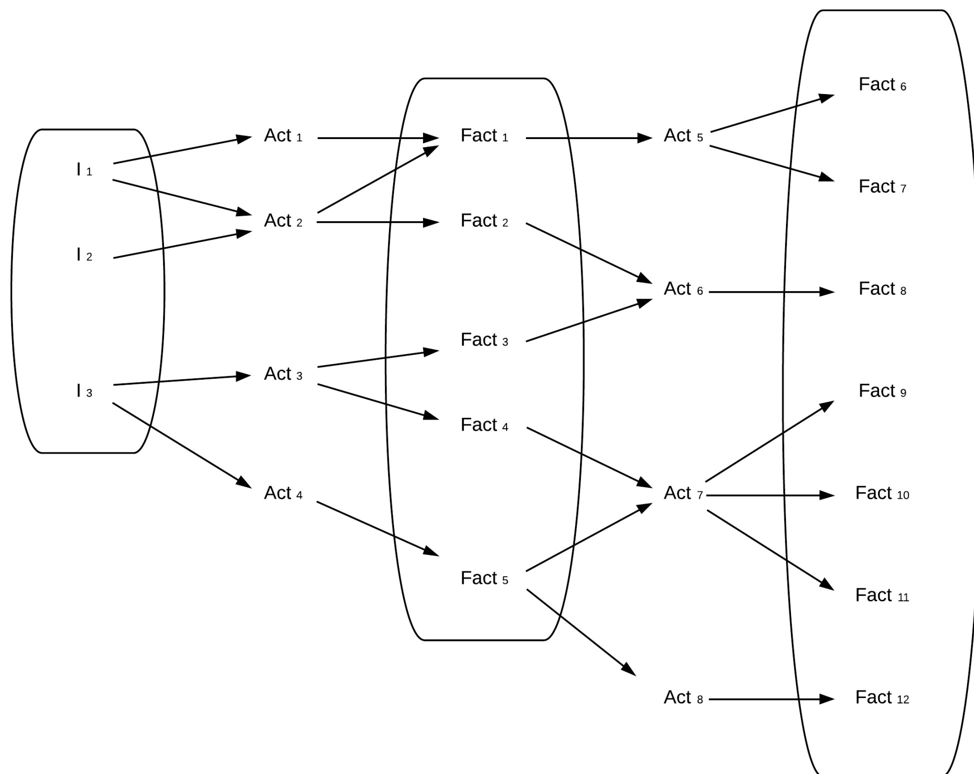


FIGURE 2.2: An example planning graph

Unfortunately Graphplan predates the International Planning Competition, therefore a rigorous analysis of its performance was never conducted, however based on the work it led to, such as Fast Forwards[40], Fast Downwards[41], LAMA[42] and others, it can be thought of as a predecessor to an entire family of planning algorithms. As such, its lasting significance is much more related to its conceptual contribution than the improvement this directly had on its performance on planning problems.

#### 2.1.3.1 Fast Forwards

Of the successors to Graphplan, one of the most celebrated remains Jorg Hoffman's Fast Forwards system. It uses a variant of the planning graph in order to provide efficient heuristic guidance for a forwards-chained search, alongside a novel approach to action choice.[40] FF proved its dominance against a range of planners as part of

the Second IPC, in which it was recognised for “Exceptional Performance” - the only domain-independent planner to achieve this that year.[43]

**The Relaxed Planning Graph** FF’s most significant contribution is the “Relaxed Planning Graph” (RPG), a variant of Graphplan’s planning graph in which only the positive outcomes of actions are considered. Put another way, when a fact is asserted as true, it is never falsified in the RPG. This gives a simplified view of the problem, and the plans generated will be misleading - in effect, the RPG is providing an idealised view of the world, but this simplification drastically reduces the computation required. Consequently this makes the RPG very useful for estimating the amount of work remaining in a problem, using this idealised plan length as a heuristic estimate. It should be noted that due its nature, the RPG will underestimate the amount of work required (although it will also never overestimate) making it an admissible heuristic as defined in Section 2.1.2.1.

By its nature, it is important to recognise that the accuracy of the estimate is dependent on the nature of the domain, since it is poorly suited to problems that feature structures in which a goal fact must be falsified in order to achieve another fact and then reachieve the first, a scenario famously encountered in the Sussman Anomaly.[44] The core issue with this is that a plan derived using the RPG will not account for reachieving the first goal, since it does not negate the goal being achieved when it is undone. This can lead to vastly underestimated heuristics in these situations, however since it is only guiding the search this inaccuracy will be handled by the search algorithm when it is detected.

**Helpful Actions** FF also introduced a system of pruning for the search space based on the potential actions under consideration. Specifically it restricts its attention to

"helpful actions" which are those that will add at least one of the goals of the planning system in the next step. This allows the search to focus solely on actions that contribute directly to the plan.

On the face of it, this filtering of actions may seem excessively constraining, but it is important to be aware that both the RPG and the helpful actions filter are only used during one particular search technique that FF uses, known as "Enforced Hill Climbing" - when this strategy fails to provide information to the algorithm, it defaults to a traditional breadth-first approach, which allows FF to maintain a robust success rate on a range of plans, since its particular techniques are bypassed in the event that they are not informing the search.

#### 2.1.3.2 Macro Actions

The concept of the "Macro Action" is relatively simple and one that has been around even since the time of STRIPS, albeit undergoing a number of iterations[36]. Put simply a macro action is a sequence of actions that occurs together in such a way that it makes sense for it to be packaged up as a single action. Consider the sequence of actions:

1. Leave the house
2. Drive to the store
3. Order coffee
4. Collect coffee
5. Drink coffee

Trivially it can be surmised that this sequence of actions will likely occur together on a regular basis, and because of this, its possible to reduce the depth to which a planner needs to search by simply encapsulating the whole sequence in a single action.

Importantly, this example hinges on having an understanding of the nature of the world, which for the purposes of the example is the domain of the planning problem. Because of this, humans can appreciate the significance of this sequence of actions. For Automated Planning, this intuition is not available, and the majority of significant improvements to macro actions have been centred on their generation.

There are broadly two methods by which macro actions can be derived, either by analysis of the domain or by analysis of plans (although some planners adopt a mixed approach using elements of both). In the first instance, analysis of actions is typically used to establish causal links between actions, which is to say an action that generates a fact that another condition uses. An example of this could be unlocking and opening a door, the first of which asserts that the door is unlocked, whilst that the second requires that the door be unlocked. Examples that utilise this approach include REFLECT[45] and more recently Macro-FF.[46]

Analysis of plans more closely resembles the coffee example above, where sequences and patterns are sought in the generated plans for a domain in order to increase efficiency when solving other, more complex problems. This approach can be seen in planners such as MARVIN[47] and MORRIS.[48]

#### 2.1.3.3 Landmark Analysis

A comparatively recent advance in planning, “Landmark Analysis” refers to discovering facts that must be true at some point during execution in order to reach a solution to a

planning problem.[49] Conceptually these are bottlenecks in the domain that all plans must go through, which can be visualised as a bridge crossing a river for plans requiring you to drive from a point on one side to a point on the other - the route taken on either side can change, but every plan must by its nature cross the bridge.

Identifying these locations allows a planning problem to be safely decomposed into separate problems, firstly from the initial state to the landmark state and secondly from the landmark to the goal. This allows the combinatorial explosion of states per layer in the search tree to be managed directly, since these can be treated as two independent problems, as shown in Figure 2.3 which shows this explosion control as the comparative width of the triangular search spaces, both under a traditional search and a Landmark-based search.

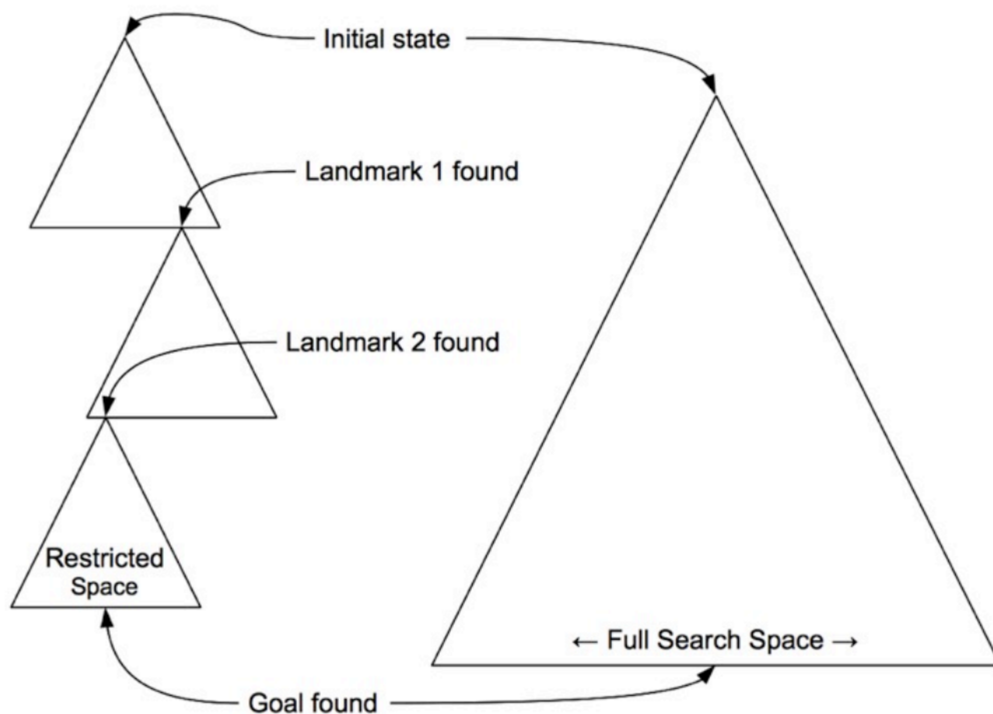


Figure 2.3: Demonstrating how landmarks can manage the combinatorial explosion of the search space

Although trivially, the initial and goal facts constitute landmarks, they are not considered interesting. The extraction of those landmarks that are interesting is a more

complicated task that utilises the properties of the Relaxed Plan Graph in order to identify states that appear as a requirement to satisfy facts that must be true down the line. A level of verification is also required since the RPG is not able to identify cyclic graphs and will falsely indicate the shortest path round the cycle as a landmark.

Landmark analysis has been shown to cause significant performance improvements in planners based on FF and Graphplan[49] and two notable planning systems that have exploited aspects of landmarks, Fast Downwards[41] and LAMA[42], have gone on to great success with their respective entries in the IPC.

### 2.1.4 Planning For Uncertainty

As mentioned previously, planning assumes a deterministic well-behaved environment, and that the agent performing the planning is the only one active in that environment. These are very idealistic assumptions that occur infrequently in real-world applications. One viable strategy is to pair a planning system with some other form of executive system, which is a model that will be discussed further in Section 2.3, however attempts have been made that are purely based on planning. The first of these that will be discussed is the attempt to model uncertainty as part of the planning problem, whilst the second approach accepts that the assumptions of planning are invalid and attempts to find ways to work around this.

#### 2.1.4.1 Probabilistic Planning

Probabilistic PDDL (PPDDL) was adopted as the official standard for the IPC's probabilistic track, held in 2004 and 2006[50]. PPDDL's primary contribution was to introduce the notion that effects of actions could be probabilistic and to allow a distribution to be

defined for the possible outcomes. For example consider a robot picking up a box in a warehouse. Under PDDL, the assumption is that the robot will always successfully pick the box up, but there could be many reasons why this action could fail, from mechanical failure to the box being misaligned. PPDDL allows this to be represented in the model, and the effects of failure to be enumerated - perhaps in the case of the robot above, there is a 5% probability of failure in which the box ends up on the floor, a 1% likelihood that the robot's arm fails and a 94% chance that the action completes successfully. Although it is true that this is a more realistic model of the world in many respects, it relies on an enumeration of all the ways that an action can affect the world. In simulated worlds and example scenarios, this less rigorous assumption about the world may hold, but it is still not accurate as all the ways in which something can go wrong is effectively unquantifiable. A more detailed discussion of this specific issue will be given in Chapter 5.

**Markov Decision Processes** Effectively, PPDDL representations are just a logical restatement of a classical mathematics device called the Markov Decision Process (MDP).[51] This is a "discrete-time stochastic control process", or more informally describes a system in which the next state that the system will be in is defined by a probability distribution based on the current state.

#### 2.1.4.2 Plan Repair

Under plan repair, the underlying concept is that since planning is inherently problematic, instead of modelling a world using further assumption about its probabilistic nature, it is better to instead monitor plan execution and when the state of the world differs from the expectation, to replan from this new state. This is intuitively quite a

naive approach, and yet when first introduced by FF-Replan[52] it proved very successful, winning the 2004 IPC probabilistic track, and had it entered the 2006 track, it would also have won that too.

However, a more nuanced approach is to attempt to find some way of "patching" the existing plan such that the computation already performed does not need to be redone. By instead planning for ways that the plan can be resumed, the CPU requirements of the end-to-end process has been shown to be reduced.[53]

#### 2.1.4.3 Partial Observability

It is also worth noting that there is another class of uncertainty, known as "Partial Observability". It is easy to see that in some cases the result of an action may be one among a set of choices, it is also possible that it is not known with full certainty what state the agent is in. There may be areas of an agent's world that it is not currently aware of, meaning that the world has some state but aspects of that state are not observable by the agent. This introduces significant complexity to the problem. Although I2A does not cater specifically to the partially observable case, it is worth bearing it in mind as this is more general than the fully observable case assumed by the majority of AI systems.

### 2.1.5 Automated Planning Summary

Automated Planning has been shown to be a valuable area of AI research. Over the decades since it first was explored more complex problems can now be tackled and although initially some of the limitations of the discipline were severe, advances have allowed for the planning tasks to be achieved faster. However, the techniques as a group

still assume that a lot of computational power is available, orders of magnitude beyond that which game engines typically allow for AI subsystems. Not only that but in order to provide tractable problems a lot of assumptions are made about the environment, most notably that the agent will be the only thing able to make changes to the environment. This does not hold in game environments.

Although the ability to create long term plans of action to solve complex problems is a vital aspect of the perception of intelligence, in its current form, Automated Planning is not an ideal solution to tackle the kinds of problems highlighted in the prior chapter.

## 2.2 AI Techniques in Games

As noted in the previous chapter, there is a long tradition of putting AI systems in games. For the purposes of this literature review, the discussion is limited to only to what might be called the "contemporary" era, which is to say what can be recognised as video or computer games. Even here, there are techniques that have withstood the test of time due to the simplicity, and a range of techniques that are more modern which provide additional refinements, increased believability or perform more optimally.

Broadly, AI techniques in games can be broken down into three families. Firstly are those that have derived from Finite State Machines, in which the AI system steps through discrete, defined states in a manner controlled by the system as it has been created by the AI developer. Another range of techniques try to model the world in such a way that mathematical processes can be used to evaluate appropriate responses for the AI system. Finally, a group of techniques rely on much more significant processing in order to produce their responses, which are based not only on the current state that the agent

is in, but also reasoning about future states and the way that the world can be changed to suit the agent's needs.

### 2.2.1 State-based Techniques

#### 2.2.1.1 Basic Finite State Machines

Perhaps the most popular "entry-level" technique for AI in games is the Finite State Machine (FSM).[54] In this paradigm, states that the agent can be in are represented as mutually exclusive, and there are discrete and explicit transitions that allow the agent to change between specific states when a certain trigger is received.

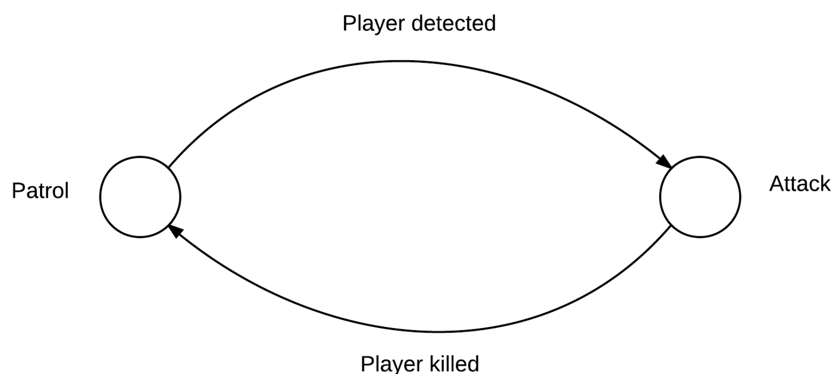


FIGURE 2.4: A Simple FSM for Guard Behaviour

For agents in game worlds this could be a reasonably simplistic system such as a guard who patrols until it detects the player at which point it transitions into an attack state. When the player is dead, the guard transitions back to patrolling. This simple system can be seen in Fig. 2.4. However immediately there are oversights and situations that have not been considered - what if the player escapes from the guard for example? At some point, the guard should return to its patrol. Perhaps it would be more realistic if the guard transitioned into an intermediate suspicious/exploratory state when it loses

sight of the player. The FSM system is very powerful, however this expressive power is matched by a very unwieldy system that suffers greatly from combinatorial explosion. This can make designing AI systems using basic Finite State Machines a very challenging problem.[55]

#### 2.2.1.2 Hierarchical Finite State Machines

The Hierarchical Finite State Machine (HFSM) attempts to overcome the complexity of the FSM by organising states into a hierarchy, grouping similar states together.[56] [57] This allows for more complex behaviour by allowing a designer to box out an entire set of behaviours as one "state" at the top level, and then internally handling them more robustly, perhaps as an iterative series of states.

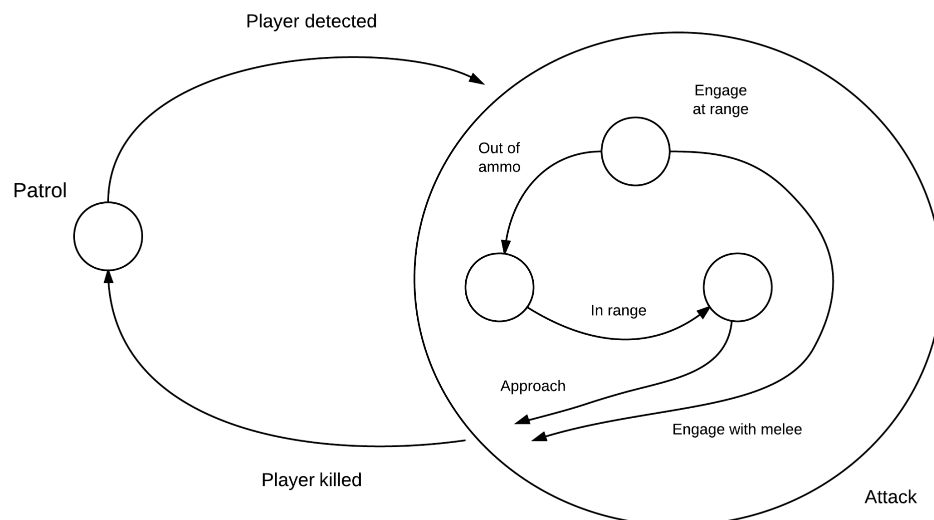


FIGURE 2.5: An example HFSM

Revisiting the guard from the previous example, it might be appropriate to think in terms of attacking the player as a single state, but in reality there are many different ways that the player could be attacked depending on circumstances. If the player is a long way away, the guard may choose to attack with a ranged weapon, but if the player

is close, a melee weapon would be more appropriate. However the process of selecting between them is kept internal to the attack state, and transitions move in and out of that state rather than having to wire each of the substates into the rest of the machine. In this way, the HFSM manages to some extent to mitigate the combinatorial explosion that results as more states and transitions are required in a typical FSM.

#### 2.2.1.3 Hierarchical Concurrent State Machines

The Hierarchical Concurrent State Machine (HCSM) is a relatively modern addition to the AI programmer's arsenal, having been developed as part of a research project at University of Iowa to provide scenario management in a driving simulator.[58] HCSMs trace a lot of their heritage back to Statecharts[59] and Finite State Machines before them.

The core building block of an HCSM is the HCSM itself, creating a nested, Russian doll style system that organises modular component. Where those internal HCSMs are linked by transitions, the parent HCSM is described as "Sequential", since only one can be active at a time, they are activated in sequence. If there are no transitions between the child HCSMs, then the parent is "Concurrent" and all the internal states are active at any given tick.

Each HCSM has an "activity function" associated with it, which is used to generate an output based on, but independent of, the active state within the HCSM. The HCSM specification defines input "wires" used to supply a continuous stream of data to the machine, as well as controls and "dials" (represented by variables in code) which are used to set parameters within the machine. It is important to note that time within the

machine is discretely modeled, and although changes to the input are visible immediately inside the machine, changes made to the controls are not reflected until the next tick.

When an HCSM is executed, firstly a “pre-activity” function is run which is used to control the flow of information into the child HCSMs that will be active this tick (without reference to which will be active). Subsequently, if the HCSM is sequential then which transitions should fire must be determined, and which of the children will become active as a result. Having determined this (or bypassed this step for concurrent HCSMs) each of the children of this HCSM that is active will have its execution function activated, generating a recursive set of calls inside the nested HCSM from the top down. Subsequently the HCSM’s “activity function”, which is to say it’s actual output value, is evaluated. This is an important ordering to understand, since it means that whilst children of an HCSM are able to influence their parents, parents cannot have any bearing on a child’s output this tick, and need to wait to impart this as part of the pre-activity next tick to set the controls of the child HCSMs.

HCSMs provide a way of managing the combinatorial complexity of large sets of states by removing the need for states to transition between each other in the case of those that can be interrupted, for example a set of states that represent a “patrol” behaviour don’t all need to be able to transition into an “attack” when certain conditions are met, instead the patrol behaviour states are unified into a “superstate”. Their modular nature also means that different aspects of the world can be handled separately, or jointly. The HCSM gives a significant degree of modularity whilst also being relatively easy to implement due to its recursive nature.

The HCSM is seen as significant in the Game AI community as it is the driving structure behind the Left 4 Dead AI Director developed by Valve.[60] This was a system designed

to alter the game in subtle ways both to ensure added replayability since the layout of enemies and pickups in each level would change, but also to control the flow of the game and enforce a certain pacing. The motivation for this was drawn from the horror movie genre in which it is accepted that there is a certain pacing, in which intensity grows gradually to a peak, stays at this peak and then tails off leading to a rest period, which is seen as very engaging by audience members. Left 4 Dead attempted to replicate this pacing, but in an interactive setting it is very difficult to attain this kind of experience on a per-player basis. Traditionally the aim of level design has been to provide this for an approximately "average" player, but by being able to tailor the experience through the HCSM system, specifically designed for scenario control, a number of factors of the level design can be subtly influenced such as the number and location of the enemy characters in the level. More recent implementations have also taken to putting a wider range of aspects of the environment under the control of the AI Director in order to create a more engaging experience for the player.

#### 2.2.1.4 Behaviour Trees

By far the most common contemporary AI algorithm for driving characters in a game world currently is the Behaviour Tree (BT).[61] BTs are derived in part from Finite State Machines, and owe at least part of their ancestry to the State Charts widely used in the Embedded Systems community, and first came to the fore in the video game industry after their successful implementation by Damian Isla during the development of Halo 2 (Bungie, 2004).[62]

A Behaviour Tree can be thought of at its most basic as a tree, although more technically it is a Directed Acyclic Graph since a node can have multiple parents. Leaf nodes within the DAG represent some action or set of actions that the agent can express as well as

conditions that must be true, whilst nodes higher in the DAG structure are concerned with the logic of traversing the DAG.[63]

More specifically, these higher nodes are grouped into "Selectors" and "Decorators". Selectors choose which of the children of the node will be traversed, and in what order.

- A "Sequence Selector" ensures that all the children of this node will be visited in a left-to-right manner.
- A "Concurrent Selector" is used when the children of the node should be visited together.
- A "Random Selector" is used to determine at random the order in which the nodes of the sub-tree should be visited.
- A "Priority Selector" allows for a run-time evaluation of which nodes should be visited in what order.

There is only one type of Decorator, which is used to ensure that certain conditions hold true or for example to allow a timer to lockout an entire subtree if it has been visited too recently, thus controlling the frequency with which certain behaviours can be triggered.

Leaf nodes can either be "Conditions", which will evaluate as true or false dependent on whether the condition has been satisfied, or "Actions" which can be false if the action's preconditions are not met, or "running" if the behaviour begins executing.

When an action or condition return false, that blocks all siblings of that node in the DAG from being considered. In a concurrent or sequence selector, that causes the selector itself to evaluate as false, but in random and priority selectors, it means that they re-evaluate and choose a new subtree to work with, either at random or the subtree

with the next highest priority. When an action and selectors returns a "running" result, this is passed back up the DAG to the parent node, effectively meaning that the DAG is traversed until a behaviour begins running, or the DAG has been exhausted.

At the next iteration, traversal will resume at the root node and continue as normal until a selector which is still set to "running" is found. At this point, rather than begin processing the selector as normal, the BT will begin from the behaviour that was running at the previous iteration. This allows a minimal amount of state information to be passed between ticks implicitly, since using this, complex sequences of behaviours can be executed whilst still intelligently overridden as circumstances change over time.

One of the key strengths that game developers like about the Behaviour Tree architecture is its extensibility. New behaviours can be added easily without requiring any additional changes be made to existing behaviours. It's also easy to arrange and prioritise behaviours that have been crafted in order to tune the overall feel of the agent's actions. Because the architecture is inherently stateless (beyond a memory of the prior running behaviour) it is significantly easier to manage complex behaviour systems - in particular from a design point of view - since for example the concept of high priority behaviours intervening and preventing lower priority behaviours from executing does not need to be considered as a specific set of state changes as it would in a more traditional architecture such as a FSM.

**Behaviour Trees and FSMs** It is worth noting that in many cases the Behaviour Tree is used as a device to limit the expressive power exposed to a systems designer. This artificial limit allows the designer to focus on creating realistic behaviours in a manageable manner, but at runtime many Behaviour Tree systems will take the tree that the designer has created and compile it into a FSM for execution. This exploits the

fact that all Behaviour Trees can be represented as an equivalent FSM (although the reverse is not true), a point made by Sunshine-Hill at GDC 2014.[64]

### 2.2.2 Mathematical Techniques

#### 2.2.2.1 Influence Maps

One of the early approaches to AI in strategy games was the Influence Map, which evolved from approaches that were used by research in the game Go[65] and has been used to great effect in specific works in the games industry.[66] In many ways it can be seen as a kind of abstract representation of the world in which objects within that world exert positive or negative influence. That influence can then radiate out throughout the world and serve as an attractive or repellent force to guide the movement of an agent.[67]

Influence Maps are spatial representations of the world - typically a two-dimensional, discrete, tile-based world. Influence radiates to adjacent tiles from a source. In a game context, powerful enemy soldiers may be a source of negative influence as the agent wants to avoid combat with them. Weaker enemies may be a source of negative influence too (generating much less influence than their stronger counterparts) to avoid potential losses, however in a game where the units can gain additional attributes through combat, these weak enemies may generate positive influence since the agent can defeat them. Items and resources that the agent can use could be a source of positive influence attracting the agent to them.

An Influence Map relies on a formula in order to radiate the influence a source creates in the world around other locations by propagating the influence out. There are a range of different approaches that could be used here, but the most common are typically linear and inverse square.[67] In either case, the amount of influence an agent experiences due

to one specific source decays as the distance from that source increases. This parallels a number of physical forces such as magnetism and gravity, which suggests why an alternative term for this approach is the Artificial Potential Field.

All the sources of influence in a world have their respective influence propagated throughout the world. In areas where one or more source exerts an influence, some method of combining them is used, which could be summation, taking the maximal value or some other approach. There are arguments for many different techniques to be used here, and each will give an agent using an Influence Map a different behaviour. A summation of influence allows a positive source to cancel out a negative source but two positive or negative sources will reinforce each other. An agent operating under this system would avoid areas that were clearly bad, but might be tempted to explore areas that had positive and negative elements. A maximal value approach might give the agent a very positive view of the world, where it overestimates its likelihood of success, whereas a minimal value approach, where the agent uses the least value of the combined influence sources would generate a very pessimistic view of the world, and consequently an agent that acts quite timidly.

What is important to observe is that whereas the previously mentioned systems required a designer to architect a set of behaviours, an Influence Map uses the interplay of mathematical formulae to evoke that sense of behaviour and intent in the agent. Although it is used solely for choosing areas of the world to travel to (and it should be noted is not implicitly a navigation system per se, although an A* algorithm can be used in conjunction with the influence values), it is still possible to generate different tones to the agent's actions.

In Figure 2.6 an example of a calculated Influence Map is shown. This is drawn from

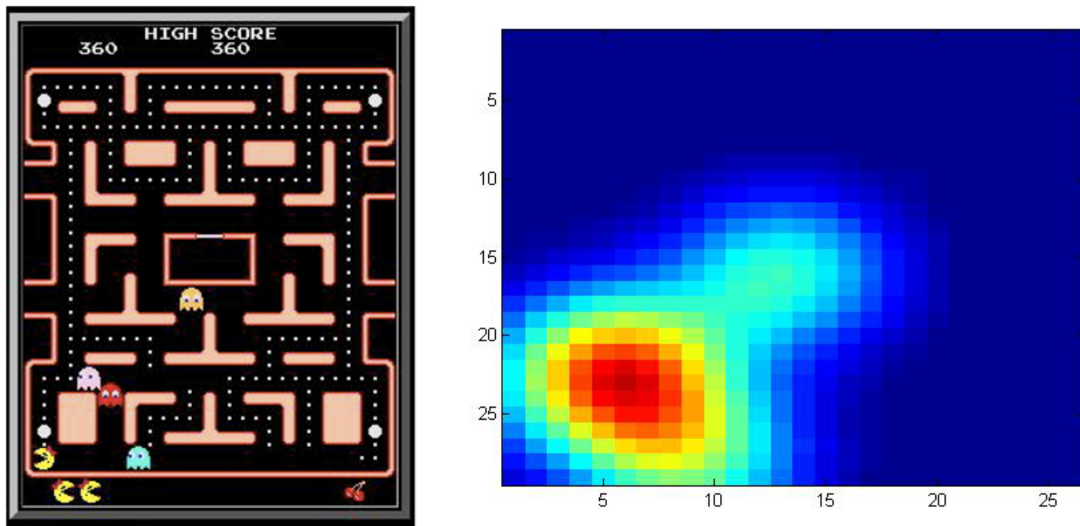


FIGURE 2.6: An Example of an Influence Map in Ms. Pac-Man

previous work with Ms. Pac-Man (Midway, 1982).[68] In the left of the image, the state of the game is shown, the agent is trapped and enemy ghosts are closing in. On the right, an Influence Map for this situation has been calculated, with a high value reflecting a strong negative influence. The cumulative effect of the close grouping of ghosts in the bottom right reinforces to produce a high amount of influence in these areas (represented by the red “heat”), whilst the ghost in the centre of the screen produces comparatively low amount of influence (as shown by the cyan colour).

#### 2.2.2.2 Utility-based Architectures

Utility architectures are so called because they assess the “utility” or “value” of the options available and make their choices based on what seems to be the most appropriate course of action at any given moment. Utility systems are set up in such a way that the relative value of different objects can be described and combined with other stimuli. This means that a large number of factors can contribute to a decision, and because the evaluation is a mathematical operation, the outcome can be found quickly. Utility lends

itself to situations where there isn't a specific "right" action to take in a given situation, but there are a number of actions that would be acceptable. [63] [69]

Each potential action can be assessed to determine its suitability in a given situation based on a response curve. For a very simple action, there may be only a single factor being considered, and consequently a single response curve, an example of this might be a naive "Attack Enemy" action, which becomes more valuable the closer the enemy is. A simple function for this might start with a high value when the enemy is 0 units away, and decay linearly to a distance where the value of the attack becomes 0. However, this is not an especially sophisticated implementation as other factors would need to be considered such as the ammunition level of the NPC and their health - with low ammunition, or low health, a better response would be to avoid the combat and retreat. These would be represented with their own response curves which may be linear or any other function. The output from each one is then combined to produce an overall rating of the action. This combination function can take a number of different forms, such as taking the minimal or maximal value, or multiplying or adding the values together. The result is then compared to the rating of all the other actions available, putting them into a ranked order. From this, the most appropriate action can be taken based on taking a number of factors into consideration.

One interesting part of the nature of Utility systems is designing them to be extensible. Care must be taken with this to ensure that, for example, the addition of a new factor for consideration doesn't increase the potential value that an action can take and create an accidental bias. Dave Mark's Infinite Axis Utility System is an example of a system that works around this by ensuring that all the response curves are clamped between 0 and 1, so that any result from a response curve will always be in that range, and should the curve give a value outwith this, it will be adjusted to be either 0 (for values lower

than 0) or 1 (for values greater than 1). This combination technique used by the IAUS is a multiplicative process, which since the components of the function range between 0 and 1, results in a value that is guaranteed to be between 0 and 1, regardless of the number of factors and response curves that are considered.[23]

Previously, it was discussed that Behaviour Trees could make use of a "Priority Selector", and it is worth noting here that a good implementation for assessing the priority of children of such a selector is based on Utility. This was shown to good effect in Redshirt (The Tiniest Shark, 2013) which uses a single Behaviour Tree to model a number of distinct characters.[70] The nature of the game is such that each character has a personality and must take actions in keeping with that personality, and here a Priority Selector can tie into a utility system that takes that personality into account to ensure that actions chosen are in keeping with the character traits that the player has previously seen the agent exhibit.

**Reasonable vs Optimal Play** One of the aspects of Game AI that is highlighted very well by the Utility approach to decision logic is the distinction between Optimal play and Reasonable play. In traditional Artificial Intelligence, the assumption is typically that the best decision is always the one that results in some metric being optimised, such as speed, cost and so on. In a game-based domain, especially for agents within a game world, this means optimally completing their objective. When this objective is set to be an interaction with the player, and especially when it is an enemy attempting to oppose the player, making the optimal choice at every decision point can lead to a very difficult game for the player to win. Because Utility Theory scores all of the possible actions, it is possible to ignore the "optimal" highest scoring action and instead choose the next-best action. By doing this, optimal play can be avoided thereby creating a beatable

challenge for the player whilst also ensuring that the actions chosen are sufficiently realistic choices to maintain believability. It is also possible, instead of choosing the next-best action to randomly choose between N next-best actions, introducing an amount of non-determinism whilst also maintaining believability of action choice.[71]

#### 2.2.2.3 Neural Networks

The Neural Network architecture is modelled after our understanding of how a biological brain operates. Whereas the Neuron forms the core of a biological brain, a Neural Network is formed from interconnected “Perceptrons”. These are broadly the same in function, and can almost be seen as analogous to an organic transistor, albeit where a transistor is only able to output a signal at 0 or 1, making it digital, a perceptron can output any value in the range 0 to 1. This output is calculated based on an internal weighting of inputs combined with an ”activation function” which maps that input into the appropriate output signal.[1] A diagram demonstrating this structure is shown in Figure 2.7.

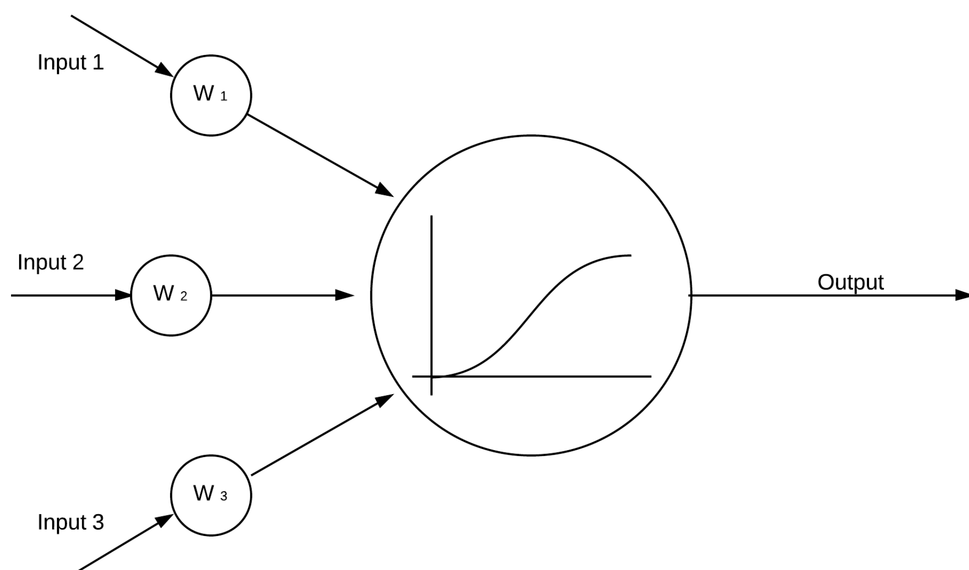


FIGURE 2.7: Overview of a Perceptron

The power of the Neural Network comes when a number of these perceptrons are wired together. At the initial layer, a range of inputs are brought in from the world, and then these are wired to a number of perceptrons. At the final layer of the network, the output from these last perceptrons is wired up to a series of actuators that will effect change in the world. When a perceptron has an input added to it, it can determine what weight to give to that input, so it can bias in favour of one or more of the inputs, or have no bias.

The nature of the Neural Network allows the weights to be "trained" in such a way that they provide the expected response out of the actuators in a set of training examples. This is an example of "supervised learning", where the weights are being tuned so that the behaviour of the system matches the outputs that are deemed to be appropriate in the situations that the training examples describe, with the notion that being able to adequately respond to the situations described in the training data will produce a Neural Network capable of reacting to a range of circumstances.

Neural Networks in their purest form are about pattern recognition, and mapping that recognition to an output signal.[72] However, the way that they are doing this is through function approximation. Training an NN can be seen as giving it datapoints, and explaining what side of a curve that point belongs to. The Neural Network then performs a regression analysis to classify unseen datapoints.[56]

Typically in games however, it is more common to be talking about using Neural Networks for agency rather than classification, although from a certain point of view it can be argued that this is still classification - a determination of "good" and "bad" actions dependent on circumstances. In this model, the classification itself is secondary to the output being generated and mapping that into action in the game world.

A simple example of this kind of system can be seen in research done with “EvoTanks”, in which small tanks are controlled by Neural Networks in order to play a game inspired by the classic game Tank (Kee Games, 1974) in which two tanks attack each other. The game is shown from a top down point of view, and each tank is able to turn left or right, move forwards or back and fire.[73] In EvoTanks, each tank is controlled by an NN whose outputs are wired to each of the potential actions within the world. The input for each NN is connected to a sensor. In this way, the perception of the world becomes a datapoint, and the resultant response a classification of what the agent did in that situation. This creates a system that can learn as it goes by trial and error, feeding the validity of its actions back to either reinforce or correct an action.

This has been used to good effect in a limited number of games directly, such as Creatures (Millenium Interactive, 1997) and Black and White (Lionhead, 2001) which provide scope for players to train agents about their world. In Creatures, these were pet-like characters, and the player was responsible for looking after them, but also giving them the necessary knowledge of what aspects of the world were good and bad such that they could help themselves. In Black and White, the player took on the role of a god with a giant animal acting as their avatar. The avatar could be trained to interact with the world on the player’s behalf, performing good deeds for their worshippers - or equally bringing down vengeful wrath upon them, depending on their play style. In both cases, a Neural Network was part of the solution that powered these systems.[74][75]

More recently, NNs have been used to power the decision making in Supreme Commander 2 (Gas Powered Games, 2010).[76] The approach taken here was to train neural networks such that they could make decisions for commanding an army in a real time strategy game. One key aspect that this work explains very well is that it is very important when working to train Neural Networks that an appropriate evaluation function be

used. Robbins relates how an oversight in the evaluation of actions in SC2 initially led to armies that were unwilling to attack the enemy commander, since his death caused a large explosion that destroyed the friendly army. However, killing the enemy commander also won the game, an aspect that wasn't exposed to the evaluation function. Because of this, initial attempts to train the system avoided a number of obvious game-winning strategies.

This annecdote is a rare situation where the actions of an NN are explainable, however in general the NN is a black box, and there is no scope for tuning it by hand. The inputs supplied to the Neural Network are complexly modified by the Perceptrons in such a way that it is not clear why specific values have been derived or how to change them without retraining the Neural Network. For video game developers, this can be an awkward proposition since the behaviour of the agent needs to be tailored to feel just right in the game. For other techniques, small issues can be corrected more directly to create the requisite behaviour, whereas this is not possible with an NN. Equally, the developer's understanding for why a specific action is being taken is diminished using Neural Networks, since there is no method for justifying an action either in terms of the situation, or in terms of previous training data. As a result, Neural Networks remain an interesting, but often unused technique in games.

### 2.2.3 Deliberative Techniques

#### 2.2.3.1 Goal Oriented Action Planning

Goal Oriented Action Planning (GOAP) is a technique pioneered by Jeff Orkin for F.E.A.R.[9] (Monolith Productions, 2005), and has been used in a number of games since, most recently for titles such as Just Cause 2 (Avalanche Studios, 2010) and Deus Ex:

Human Revolution (Eidos Montreal, 2011). GOAP is an implementation of some of the basic concepts of STRIPS-style classical planning (discussed previously in Section 2.1.2), relying heavily on backwards chained search. GOAP is designed to control Non Player Characters (NPC) within a game world, and the majority of applications that it has been used for have been First Person Shooter style games, although this is not a requirement of the technique.

GOAP plans to achieve objectives that are chosen by a high level system from a number of options, selecting what seems to be the most appropriate goal for a given situation. This goal is passed to the planning system to be achieved. One interesting aspect of GOAP that extends it beyond the traditional classical planning paradigm is the inclusion of "Context Preconditions" - these are preconditions of actions that are evaluated during execution rather than during reasoning. They reflect aspects of the world that, were they put into the representation during reasoning, would make the problem intractable. A good example of this sort of preconditions is the requirement that a target be in range before an agent can shoot at it. In order to reason about this aspect of the world, it is possible to expose the entire knowledge of the pathfinding system to the reasoner in order for it to take steps within the plan to ensure that the target is within range. However, if the target is non-static and acting with its own intelligence, even this would not be sufficient since the plan could not reliably predict the target's actions. GOAP bypasses this issue by abstracting the problem to be dealt with at execution - if the target is not in range then the agent will not shoot and will be forced to provide a new plan that can be executed - perhaps by swapping its equipped weapon to one with a longer range, or by coming up with a new approach to attacking the target.

GOAP represents a significant milestone in AI for games in that it was the first time this kind of planning system was incorporated into a mainstream title, however it is fair to say

that it is also very much the lowest hanging fruit of what Automated Planning can do in game environments. One of the most significant issues is that GOAP doesn't address the fundamental problem of applying deliberative reasoning in a game environment, namely the computational expense associated with deliberation. In order to work around this issue, a lot of GOAP implementations are restricted to reasoning at a very high level, for example about tactical options rather than specific actions within the world.

Within the industry GOAP has received a mixed response. Because techniques that derive from STRIPS are able to create novel solutions to problems through their reasoning, there are many stories related by players about their experiences and the weird and wonderful solutions that the NPCs come up with when presented with edge-case situations that the game designer would not have envisioned as part of the development process, meaning that more traditional, designer-driven approaches to NPC AI algorithms would likely not have had a suitable response. However, for many development teams, it is precisely this character-based creation of novel solutions that is their major concern with GOAP and similar techniques. Typically these teams put a higher emphasis on ensuring that a player has a specific experience when playing their game, rather than providing the tools for the NPCs and players to interact to have one among many experiences. As a development philosophy, these teams are much more about creating an interactive yet almost cinematic experience, where the NPCs do not have significant autonomy or choice and reasoning is not a desirable trait. As such, adoption of GOAP and similar planning-based techniques has not been widespread, and more sophisticated implementations have not been explored.[77]

#### 2.2.3.2 HTN

Outside of the STRIPS planning paradigm, and its successors, another approach to planning that has received attention within the game development community is Hierarchical Planning, in particular Hierarchical Task Networks (HTN). HTN planning has been used to good effect in the Transformers franchise (High Moon Studios, 2010-2012)[78], as well as the Killzone series (Guerilla Games, 2004-2013).[79]

An HTN planner works by being provided with an initial configuration of the world (just as in STRIPS) and then a "Task" that must be accomplished. This task is given as a high level objective of what must be achieved. The HTN system also contains a library of "decompositions" that break tasks either into "Compound Tasks" (which can then be further decomposed) or into "Primitive Tasks" which are a set of ground actions that will achieve the task. This means that the HTN system will start from the high level task it has been assigned and break it down into more concrete things it must do, until it reaches a sequence of actions that it can execute directly.[63]

There are typically multiple different possible decompositions for a task, representing different ways that it can be completed. The role of the HTN planner is to find a suitable sequence of decompositions such that when the task is reduced to primitives, it can change the initial state of the world into one in which the original task provided has been completed. In this way, HTN can be seen as a top-down approach to planning, where the initial statement provides a plan that is highly abstracted and the planner acts to concretise it into actions, fundamentally at odds with the STRIPS family that builds the plan from the bottom-up.

The main drawback of HTN planning is that it relies on the library of possible decompositions being created in advance. This adds significant overhead to the work involved

in creating an HTN implementation, but at the same time, it also serves to address GOAP's shortcomings in so far as creative and directorial control of the NPC behaviour is concerned, since the decompositions govern the way the NPC will interact with a game world. In effect, the decompositions give an NPC flexibility in how it approaches a problem, but they do not allow it to create a novel solution to a problem, since it will always be constrained by the available decompositions, which must have been envisioned by a member of the development team.

#### 2.2.3.3 Monte Carlo Tree Search

Previously when search has been considered, it has been visualised as a tree of states, with the successors of each state being those potential states that are reached by applying some modifier to the current state. At each layer of the tree, the agent in question is able to make a decision. In many applications however, this is not the case, and there are multiple agents to account for. Game trees attempt to take this into consideration by allowing for the possibility of other agents to take actions. Typically this is turn-based in nature, with games such as chess, tic-tac-toe and checkers being among the most frequent to be represented as game trees.

By allowing the idea of external agency, these games can be played using algorithms that attempt to infer the actions the other players might make. Perhaps the most celebrated of these is the MiniMax algorithm, which assumes that the other player will play optimally and is working to minimise the agent's potential score. By making this assumption at every node in the game tree where the opponent might make a choice, the agent will always be able to maximise the potential minimum reward.[16] In Figure 2.8 a game of tic-tac-toe is in progress and the agent is playing the role of X. There are three possible moves that it could make, and by expanding the tree of potential moves,

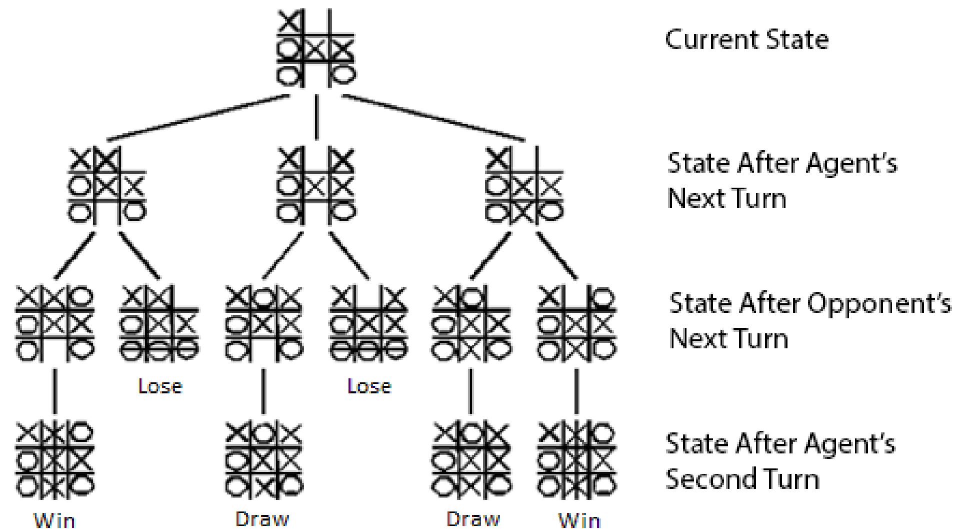


Figure 2.8: Example of a Game Tree in practice

the agent can determine its best course of action. Minimax would indicate that the best move was the rightmost of those presented, since in this tree the minimum result possible is a draw, whereas in all others, the minimum is a loss.

Monte Carlo Rollout is a technique pioneered by von Neumann and Ulam which hinges on the premise that for an arbitrarily large search space, sampling within that space can provide as useful an overview of the space as can be obtained from an exhaustive enumeration. In the context of trees, a Monte Carlo Tree Search (MCTS) involves a random walk down the tree to a terminal state and an assessment of the value of the end state. For game trees in particular, where the nodes represent moves by alternating players, a random walk can be misleading since it assumes all the moves by all the players have an equal probability of being chosen which is not the case. Equally, randomness in the walk does not allow for the information being gathered across multiple samples to bias the selection towards areas of the space that warrant further exploration due to being "good", whilst also contrasting this with the need to find areas of the search space that have yet to be explored which could contain equally promising states.

MCTS uses an approach in which a history of the nodes visited is built up over time, with each sample through the tree an additional node is added to this history, and the value of terminal state reached is recorded at the leaf node in the history, as well as propagating back up the partial tree that has already been created. This is typically stored as the average value of the leaf nodes reached from this state, along with the number of times this node has been considered in a route to a leaf node.

The balance between exploration and exploitation is handled by the selection algorithm used whilst the sample is traversing the area of the tree that has previously been expanded and is currently stored in the history. A heuristic is used here to bias the selection algorithm in some way. Most frequently this uses the the "Upper Confidence Bounds applied to Trees" heuristic (UCT).[17]

**Monte Carlo in Games** Monte Carlo was for years left as something of a dead-end after its creation by von Neumann; the number of simulations required to make it a viable approach was greatly at odds with the computation power available. In the academic games research community, it wasn't until very recently that it experienced a renaissance as a technique with the application of MCTS to the game of Go[15] and then the creation of the UCT heuristic.

Go is a traditional board game originating in China thousands of years ago. At its most basic, the rules of Go are fairly straightforward. The game is played on a grid of intersecting lines, with the points of intersection being the positions available for play. Players place a stone of their representative colour (typically White or Black) at one of these positions, taking turns. The goal of the game is to capture the opponent's stones by surrounding them either individually or in groups by your own stones, thereby removing them from the game and creating an area on the board that you control. Go is most

commonly played on a 19x19 grid (although the game can be played on smaller grids, most frequently for tuition), for a total of 361 playable "points" on the board, creating a search space that is incredibly broad, and assuming all positions remain playable (which is unlikely but provides an upper bound on game length), 361 moves can be taken, creating a potential space of approximately $2^{361}$ final configurations for the board. This is far too large a space for exhaustive search to be feasible, and with the bushy nature of the tree, most traditional search strategies would not be appropriate. Because its nature is to sample the search tree, Monte Carlo has proven especially effective at tackling this problem and evaluating what the best move in a particular situation is.

When used in a game context, initially Monte Carlo was implemented such that during the rollout phase, choices were made at random. Moving into a game such as Poker, it is clear that a random selection between the choices available is not at all reflective of the behaviour of a player, and as such the search process is not necessarily exploring areas of the tree that would realistically be in use by the opponents.

Previous work has explored the ways in which the rollout can be adapted to make use of existing knowledge about the way that Poker is played using a large corpus of data based on "hand histories" downloaded from an online casino.[80] Each history is a record of what a player at the table experienced, meaning that there is a wealth of information that remains hidden since one of the key aspects of the game is the imperfect information each player has. By taking a large number of these histories and generating an action predictor from them, Van Den Broeck et al. showed that an opponent model could be imparted into a Monte Carlo Tree Search algorithm resulting in a more accurate search, producing noticeably better decisions for the agent being driven by such an algorithm.

One of the weaknesses of the approach taken by Van Den Broeck et al was the assumption

that there is only a single archetype representing a generic "Poker Player". Fundamentally this meant that all of the data they were using to learn an opponent model from could therefore be acted on as a single group. Work at University of Strathclyde moved to address this oversight by learning a classification from the hand history data that used a clustering approach to identify a number of different archetypes into which a player could be assigned. This gave even finer grain control over the action prediction by ensuring that not only were the actions selected based on what was appropriate for a Poker player, but specifically for this type of Poker player, since it is well understood that different players have different play styles. By using an action predictor conformant with a player's archetype, a more accurate simulation could be put into the rollout.[81]

Both this and the Van Den Broeck work suffered from the weaknesses of their data however, in that the information gained was incomplete and told one side of a very complex story. The Strathclyde Poker Research Environment (SPREE) was an attempt to overcome this by reducing the reliance of research on the commercial online casinos for large datasets related to Poker, by creating an environment in which players could play the game, and researchers could access the logs from the server itself, giving complete information and vastly improving the quality of the data provided to any machine learning algorithm.[82]

**POMCoP** The Partially Observable Monte Carlo cooperative Planning (POMCoP) technique is of particular interest, since this technique deals explicitly with what they term "sidekicks", which are effectively the companion characters discussed in Chapter 1.[83] POMCoP uses a Partially Observable Markov Decision Process (POMDP) a variant of the MDP previously introduced in Section 2.1.4.1, and uses partially observable Monte Carlo methods in order to find a controller that solves the POMDP. The

work uses POMCoP to play a game called Cops and Robbers, in which a human player is supported by an NPC sidekick. Two cops are required to capture a robber, therefore its necessary for the human and the sidekick to cooperate in order to trap one of the robbers and win the game. The sidekick is able to ask the human which robber to target, but doing so costs a turn. The main source of uncertainty in this game relates to the human's intentions. Although the sidekick can know which robber is being targeted, they cannot know how the player intends to act (or even if they have changed to another target), one of the biggest findings of this work was knowing when to query the human and confirm their intentions, allowing the sidekick to adequately support the player in achieving their objectives.

### 2.2.4 Game AI Summary

It could be argued that in many ways, Game AI techniques are drastically behind the state of the art in academia. What this would overlook is the differing reality of what resources are available for execution in a game environment, where so much processing power is being diverted to other areas.

Despite this, Game AI has proven exceptionally good at finding systems that are "good enough" and using sub-standard AI techniques in ways that hide their flaws to provide a good experience to players. However, with that said, there are many areas where "good enough" fails to hold up, and these are often highlighted by critics as fundamental weaknesses, since they are so immersion breaking.

Attempts to address this such as through the use of GOAP and HTN planning in recent years have found moderate success but in many ways their implementation is still very rudimentary. It's potentially the case that the industry adopts older techniques from

academia because the processing power required in order to make them function becomes available to the AI team, and arguably there is very little true innovation in this area coming from the industry - understandable since blue sky research projects are at best hard to justify in context of the financial bottom line.

With that said, the initiative shown by the industry in adapting systems and finding ways to make things work given their constraints is remarkable, and the application of this kind of thinking to other systems is one of the core tenets of the work presented below.

## 2.3 Other Relevant Architectures for AI Agents

### 2.3.1 Subsumption

The Subsumption Architecture, developed by Brooks, is one of the most well known examples of a Reactive System. The architecture consists of an prioritised list of behaviours which can be active at any given time, with each behaviour being expressed as a mapping of a set of input stimuli to a set of output effects. Multiple behaviours can be active at once if appropriate, and where there is a conflict in the outputs, the higher priority behaviour overrides, or “subsumes”, the output from the lower priority behaviour.[13]

The primary virtue of Subsumption is its speed, since in order to ascertain which behaviours should be active at a given time it is a relatively simple case of matching the current observed input from the environment against the pre-defined patterns of the behaviours, which is a computationally inexpensive process. This means that a system

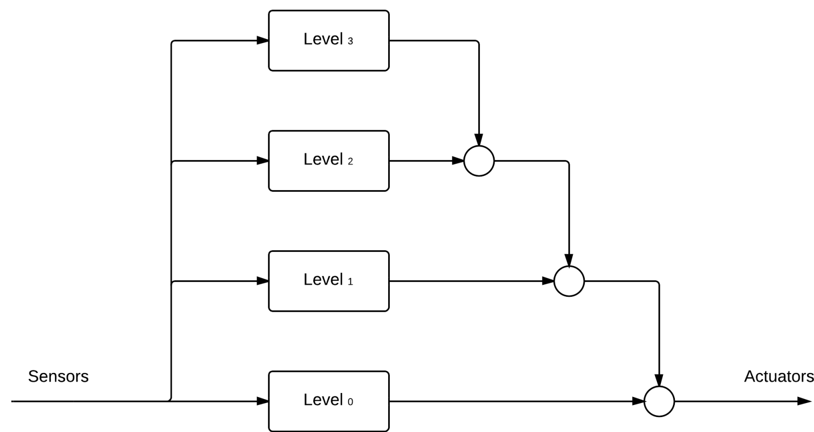


FIGURE 2.9: Diagram of a Subsumption Architecture

developed using this architecture is inherently able to quickly react to new circumstances, giving it a robust capability to resolve issues encountered.

Additionally, the Subsumption Architecture is very extensible, since it is constructed for a modular set of behaviours. To add more behaviours is simply a case of creating the behaviour by establishing the input that triggers it and the consequent output response, and then inserting this behaviour into the list of behaviours with an appropriate priority.

There are, however, a number of disadvantages to this approach. Because it is a relatively simplistic model, it is not able to represent long-term goals except implicitly within its behaviours; it does not have a stateful representation that allows for a memory of previous situations, and this can lead to an inability to achieve more complex tasks. Also, because multiple behaviours can be active at a given point, the architecture has large potential for creating an "emergent system" in which simple behaviours combine to produce more complex high-level behaviours. This can be an undesired effect and lead to problematic situations if that high-level behaviour is not anticipated.

### 2.3.2 Three Layer Architecture

The Three Layer Architecture (TLA) was a paradigm first put forward by Gat to address these shortcoming of Subsumption.[84] Prevailing wisdom before Subsumption had favoured a process based around a chain of Sense, Plan and Act phases. It was broadly felt that Subsumption eliminated the Planning phase, and although the virtue of this could be seen, as already mentioned it was not without its shortcomings. TLAs were an attempt to reintroduce the planning phase by creating a layered system, in which the speed of the reactive system could be preserved, but certain aspects of the environment could also be addressed by a deliberative reasoning system.

The canonical TLA is split into the following components: The Controller, The Sequencer and The Deliberator. The Controller can be thought of as being similar to a Subsumption Architecture inasmuch as it is designed to provide a very quick mapping from sense to actuation through a library of functions very similar to Subsumption's behaviours. The Controller typically holds a larger array of functions than are ever active at a given point, with the selection of which are active being determined by the Sequencer. This allows the Sequencer to manipulate the Controller in such a way as to ensure that the behaviours it is able to make use of are contributing to a broader task, effectively reconfiguring the lower layer in real time. The selection of high-level task to achieve is handled by the Deliberator layer, which is where the more time consuming processing takes place. This process can either be instigated by the Sequencer requesting a new task, or by the Deliberator itself generating a new task and handing it off to supercede the previous task.

The TLA approach does not stipulate the mechanics by which each layer undertakes its task, meaning that a number of different modules can be plugged together in this

fashion to achieve a result, albeit with varying degrees of success. This has made the TLA a very robust and timeless paradigm that is still seeing use today.

However, TLAs are somewhat inflexible, in as much as they rely on the task decomposition of a high-level objective being reduced to a series of shorter tasks, and then the assumption is that the emergent nature of the controller layer will allow these tasks to be achieved regardless of the environment in which the agent is acting. However, in this canonical model of the TLA, there is no scope for this to be verified and for issues encountered by the controller layer to be passed around within the architecture - only for information to be passed down from deliberator to sequencer to controller (with the exception of the sequencer polling up to request a new series of tasks to achieve).

In general terms, this is a fundamental weakness of the TLA, since at a conceptual level it makes the assumption that aspects of the world are either solvable through this emergent reaction, or by some sort of deliberation to create a schedule of emergence. This fundamentally separates the world into elements that must be reacted to, and elements that must be deliberated about, however this kind of classification is not always as clear cut in a real (or even simulated) environment.

### 2.3.3 T-REX

The Teleo-Reactive EXectuive (T-REX) generalises some of these concepts to form a more robust, fast-acting agent, specifically designed to cater to the fast-changing unpredictable environment of underwater exploration and experimentation[85].

The basic principle of T-REX is to create a set of “reactors” representing specific objectives being controlled. The reactors are able to pass goals to each other via predetermined paths, as well as passing observations about the current state, again using

pre-determined paths, providing a structure not dissimilar to a hierarchical decomposition of the task being undertaken.

The main weakness in this approach is that information from high-level deliberative modules is passed to other modules, and is not implicitly transparent to the executive. This means that although the executive is told what needs to be achieved, it has no understanding of the context, of why it needs to be achieved, and so plan repair can only occur by the same process, with observations bubbling back up to the appropriate layer for recovery and being used to update goals to be satisfied at the lower layers. This is certainly a more efficient process than requiring a full replan, but the black-box compartmentalisation of the problem by topic seems to be artificially restrictive, and still requires that deliberation be re-executed without additional instructions being passed along. Of all the executives discussed here, T-REX is the only one that comes close to resembling the approach that appears to be crucial to efficient, robust execution.

## 2.4 Summary of Existing Systems

As can be seen, although there are a number of techniques in use both within academia and industry, none adequately bridge the gap between the Reactive and Deliberative paradigms. This is problematic as it lends each technique a significant set of weaknesses and consequently, no current approach is suited to the kind of combination of scenarios envisaged that combine both long-term reasoning with short-term reaction.

In this, the Integrated Influence Architecture specifically addresses a number of weaknesses of those techniques that have been analysed above, and also inherits traits from many of them.

### 2.4.1 I2A and GOAP

By far the most obvious comparison is to GOAP due to both systems' utilisation of STRIPS-derived planning. However, it is important to reiterate that GOAP was very much the lowest hanging fruit of what is possible with planning systems in games. Planning was limited to a very high level in GOAP, reasoning only about the sequence of abstract concepts and "modes" that an NPC could use in order to limit its computational complexity. Because of this GOAP did not address replanning when a plan can no longer be executed, instead relying on the simplistic nature of computed plans to allow it to recompute a new solution to the full planning problem in a reasonable timeframe.

However, with that said, I2A still shares a lot of traits with GOAP, most notably the need to add markup to the game world so that it can be described in PDDL and reasoned about. I2A can naturally be thought of as a successor to GOAP in many ways in that it takes the concepts introduced by Orkin and expands on them to provide a more thorough implementation of STRIPS planning, without making the compromises and sacrifices that GOAP did.

### 2.4.2 I2A and Influence Maps

Of all the previously mentioned approaches to decision making, it could be argued that the I2A most closely resembles Influence Maps. In fact, a good way to visualise how I2A operates is to visualise it as an Influence Map operating across the state space, rather than being limited to the traditional constraint of only using a spatial representation of the world; in a traditional IM, the influences radiates from good and bad locations in the world, whilst in the I2A the influence originates from states that the agent should (or should not) be attempting to traverse. In this way, the I2A can be thought of as an

extension of IMs into "concepts" as well as locations, reasoning about ways of traversing the state space not just by navigating the game world but also interacting with it.

### 2.4.3 I2A and Utility

The I2A also relates closely to architectures based on Utility Theory. One of the core features of I2A is the avoidance of search as a decision-making tool, instead leveraging function evaluation as far as possible to limit the computational complexity of the approach. Like Utility, I2A is a system for ranking and evaluating a range of potential choices and like Utility, I2A evaluates all the options before selecting the one deemed most appropriate. Where Utility is evaluating potential responses to a situation and the action to take however, I2A is evaluating the potential trajectories through the state space.

### 2.4.4 I2A and Markov Decision Processes

The I2A is also closely related to the MDP system in that it is an enumeration of state transitions, which could be described in a matrix format showing adjacency of states. However, in contrast to MDPs, it is assumed that the outcome of each action is deterministic, that is to say that the action will result in exactly one resultant state. To compensate for this, the I2A allows for the state to be altered after an action is achieved, so rather than have actions with non-deterministic outcomes and have to deal with this added layer of complexity, states can be thought of as being fully connected through actions that are "secret" to the agent - they cannot be utilised by the agent explicitly, but can be applied in order to rationalise discrepancies between expected and detected state. Perhaps the most crucial distinction is that the MDP model does not allow for

changes to the world over time in terms of the value of the state transitions, which is a core concept of the I2A. This allows for the representation to be updated and links broken or added as the circumstances change, which would not be possible with a pure MDP.

# Chapter 3

# Method

## 3.1 Concept

The previous section described many of the approaches that have previously been taken to the problem of achieving agency within a game world. The core aspect of previous work has been centred around a paradigm of Reactive or Deliberative approaches to interaction with the world; architectures have predominantly tried to tackle either one or the other, and even those few that have attempted to bridge the gap have done so by decomposing the task into the two aspects, and switching between sub-systems capable of targeting one or the other.

Fundamentally, this is at odds with how we approach the world as humans. We typically react to problems within the context of the long term goals that we are trying to achieve, and we allow those goals - and our belief about the plan we need to execute in order to achieve them - to inform the decision process during our reaction. Consider a situation in which we go to start a car, and the “check engine” light comes on. The logical next step to this is to book the car in to be inspected by a mechanic. This is a valid reaction,

but it doesn't consider that tomorrow we need to get to an appointment, and driving to it is the only option. This is a reasonably trivial example, but it highlights that reacting in a vacuum does not make for optimal decision-making and can be very disruptive to what we are trying to achieve in broader terms. Having access to information about our long term plans and intentions means that we can take them into account when reacting, and react in a way that makes sense both in terms of the current decision required, as well as that big-picture view.

The Integrated Influence Architecture (I2A) attempts to bring this kind of decision making process to the AI field by acknowledging that the barrier we have placed between the Reactive and Deliberative paradigms is an artificial construct, and for good decision making, we need to be able to make choices informed by elements of both. Not only that, but the point of relying on Reactive decision-making has always been the speed with which it can act, so one of the core principles in the creation of the I2A has been retaining that speed.

At a high level, the I2A system functions by putting together a number of well understood techniques in order to solve the problem. The core of the system is a unified representation of the world that can be shared between the Deliberative and Reactive aspects. This representation is an abstract topographical model of the potential state space of the world, and the manner in which those states can be traversed. This gives a representation very similar to that used by a number of spatial reasoning systems and a problem very reminiscent of path-finding, which is a particularly well-understood area with a number of proven solutions. The "Integrated Influence" aspect of the architecture is about combining the information being provided by environmental sensing (as in a traditional Reactive system) with the information exposed by a Deliberative reasoning

system in such a way that both are influencing the pseudo path-planning around the full state space space.

## 3.2 Overview

Fundamentally, the I2A is a system for providing robust, efficient execution for agents within a game world. It does this by combining a number of components and exploiting the fact that a lot of pre-computation can be handled upfront and not be made a part of the run time process. Figure 3.1 shows the full set of components that make up the I2A and their manner of interaction.

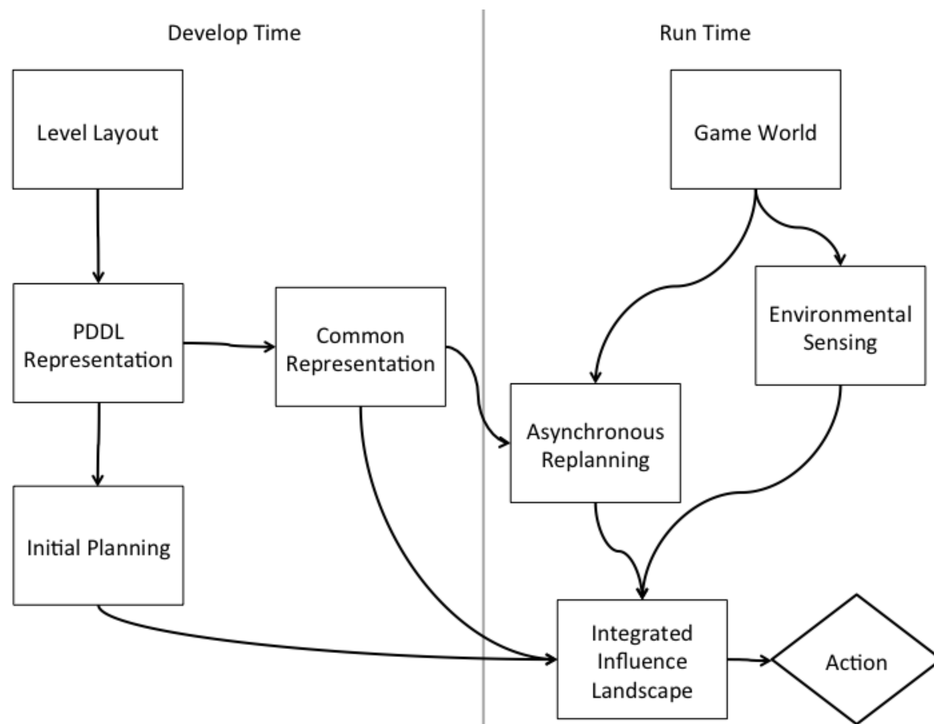


Figure 3.1: Overview of the Integrated Influence Architecture Components

### 3.2.1 Components

The components of the I2A are divided into two types, firstly those that perform preprocessing during the development of the game and utilise the time and resources available

at this point. The second type are those that are used during the execution of the game.

#### 3.2.1.1 Develop-time

The following components comprise the sub-systems of the I2A that are used during development of the game.

**Level Design**

The main component that the I2A requires is an environment to act in that has been determined in advance. That means that the game must have its levels designed during development. In order to be able to describe the level and how it works to the I2A, concurrent with the design of the level, the designer must also add descriptive markup to the level that highlights the different types of objects in the world and associates a semantic description of the world with the internal representation that the game engine will use.

**PDDL Representation**

Once the level has been designed, it needs to be transformed into a PDDL representation that will be more abstract than the level itself. This may not need to capture every aspect of the level, just those that the agent being powered by the I2A can interact with. With an appropriate component-based markup system as mentioned above, creating this could be implemented automatically, otherwise it can be generated by hand, although as a first order logic, this can quickly become a complex process to approach without automation.

**Initial Planning (IP)**

During development, a goal for the agent is determined and a plan is formed a priori using classical planning methods based on the PDDL representation generated. This

is a highly idealised plan that relies on the standard set of assumptions that planning requires such as the agent being the only active actor in the environment. As such, it will almost certainly not remain valid for the duration of execution, but it provides a starting point that will be used to inform the agent's decisions during run time.

**Common Representation**

For the I2A to function correctly, it needs to make use of a model of the world that can be shared between both the deliberative and reactive elements of the I2A during execution. In order to create such a representation, the PDDL description of the world is used and the aspects that remain constant are extracted creating a complete enumeration of the state space and the manner in which each state can transition to others. As this results in a combinatorial explosion, and an infeasibly large footprint, this enumeration then is abstracted to create a representation that is more manageable and lightweight for use at run time.

#### 3.2.1.2 Run Time

Having performed extensive computation during development, the system is then much more efficient during run time.

**Game World**

Naturally, as the I2A is designed to run inside a game, the world it operates in - the game as experienced by the player - is a very important component, and underpins the whole of the operation of the I2A. The location of entities and state of the world are very important aspects, and although a lot of the Game World component sits outside the scope of the I2A, being part of the engine powering the game, it is still important

to consider the manner in which I2A interfaces to that world, and the information is exposed back to the agent.

#### Environmental Sensing (ES)

The environmental sensing component is used to take the information being generated by the Game World about the state of certain aspects of the game world and relate that back to relevant states in the common representation.

#### Asynchronous Replanning (AR)

The point of the I2A is to alleviate a necessity to constantly replan when the outcome of a plan's execution deviates from expectation. Whilst the information presented by the IP provides an important initial source of insight into the domain, the I2A expects that it will become inaccurate at some point during execution. To compensate for this, Asynchronous Replanning takes place as a background process, allowing the I2A to update the view of the world that the deliberative reasoning is providing. Note that for expediency at run-time, the AR component does not plan in the entire of the state space, but uses the abstracted state space held by the Common Representation component. As this is asynchronous, the agent can continue to make decisions while an update is pending.

#### The Integrated Influence Landscape (IIL)

The Integrated Influence Landscape is the combination of the Environmental Sensing and Initial Planning (or later, Asynchronous Replanning) components. The IIL itself makes use of the Common Representation in order to bring in the data being exposed by both of those components and apply them to a model of the game world. The way this is achieved is by using the ES and IP/AR as sources of "influence", a heuristic estimator

of how good or bad a state is. Because this is applied to a representation common to both components, it is possible to directly combine these amounts of influence to create a coherent opinion of the current estimated value of states in the world, not only based on the state of the environment that the agent is currently perceiving, but also the objectives it is trying to accomplish.

#### Acting in the World

Finally, action in the world is achieved by evaluating the agent's current position within the IIL and performing a very localised search process to determine what the best course of action is at this time. This allows the agent to pick an action at this time that will lead to a more desirable state and ultimately to the agent's goals.

### 3.2.2 The I2A In Practice

A good way to highlight the division of these components may be to discuss a worked example of the I2A in action in a small game. In this scenario, the agent is positioned in a series of rooms - a small dungeon or perhaps a castle basement. The agent has to retrieve an object from within the level, but must be able to intelligently - and rapidly - react to changes in the environment. A simple planning system based around STRIPS can create a plan order to reach and obtain the item, however it cannot reason about the presence of monsters and traps. The agent must be able to react to their presence intelligently yet still complete the objective.

It can be seen that in the example in Figure 3.2 there are six rooms. The agent begins in the upper-left room. There is an item to retrieve in the upper-right room, but the most direct route is blocked by a yellow door. The yellow key is in the bottom-left room.

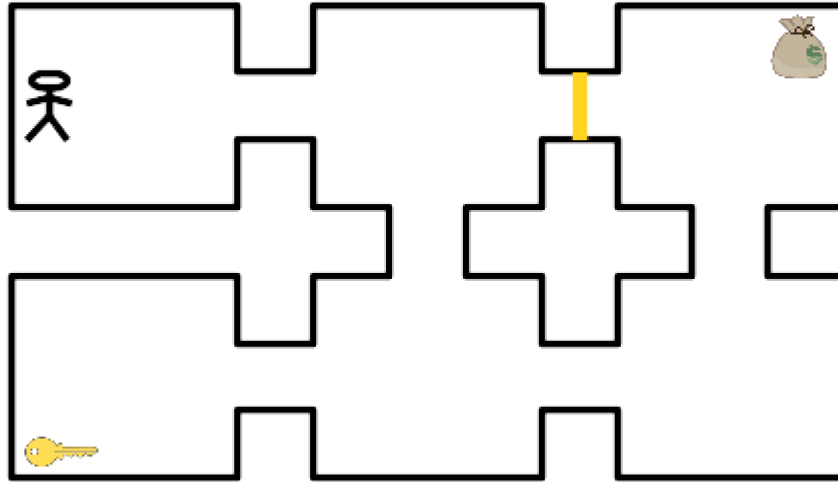

FIGURE 3.2: Layout of the Example Scenario

#### 3.2.2.1 Develop Time

As the game is developed, the objects will be placed within the world. The rooms and corridors will be laid out, and the door, goal item, key and the model for our I2A powered NPC will be positioned. As these are laid out we can semantically say that they exist in the world, that the rooms are connected to each other as shown in Figure 3.2 and so on.

It is then possible to automatically generate a PDDL representation of the world by using a system that indexes the semantic descriptions supplied by the level designer, and then undertakes some analysis to determine more implicit information such as the connections between locations in the world. The outcome of this is a PDDL Domain and Problem description pair describing the world that the agent is acting in and what the agent is trying to achieve.

Using this PDDL representation the Initial Plan component generates a plan that would allow the agent to collect the item. An optimal plan is to traverse the middle rooms

to the bottom-right and then move up, bypassing the locked door. This is shown in Figure 3.3.

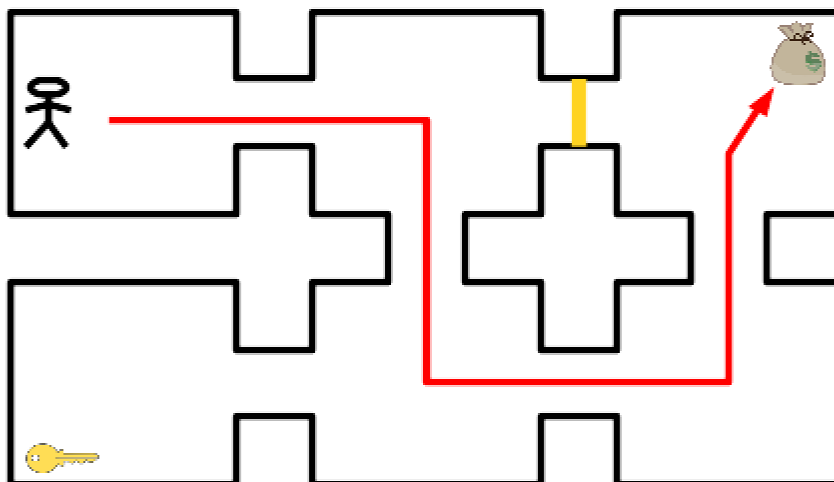

FIGURE 3.3: The Obvious Solution - Computed Upfront Forming the Initial Plan

We also need to calculate the common representation which enumerates all of the potential states that the world can be in. The agent can be in any of the rooms, it can have the key or not, the door can be open or not and it can have the item or not. Individually these form small graphs representing different aspects of the world. The common representation begins to emerge when we consider a full enumeration of all the potential combinations. The actions that cause the transitions between states become edges in what is a full state space.

For a small problem like this, the state space is manageable, but it can suffer from combinatorial explosion when this is done for larger problems, therefore as part of this process the common representation goes through a process of abstraction to create a more manageable problem space to work with.

#### 3.2.2.2 Run Time

At run-time, the game world begins to operate as expected. Effectively this means that the simulated world has come to life and the agent can begin acting in the world. To begin with the agent can conform to the Initial Plan provided during development and begin by moving into the bottom-middle room. As it attempts to move into the bottom-right room, the combination of the Game World and Environmental Sensing signal that there is a problem - the agent would have to pass through fire to continue executing the plan. As this is not a good thing for the agent, the Integrated Influence Landscape is updated to reflect that a negative area has been found.

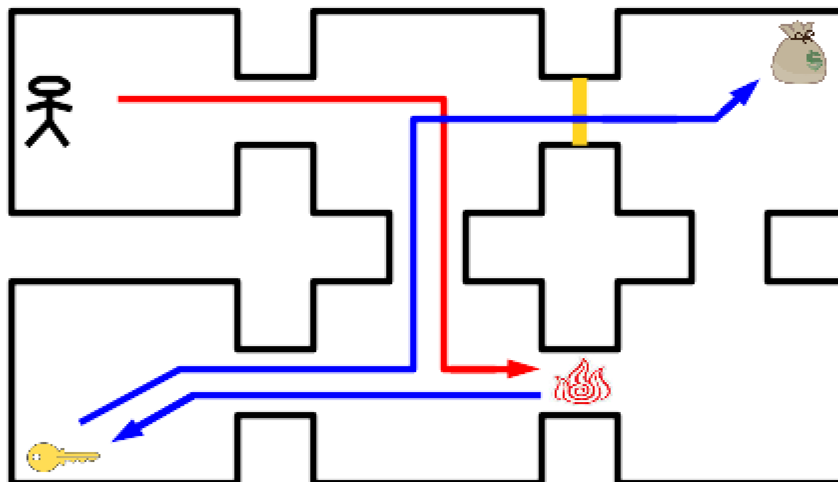

FIGURE 3.4: The Contingency Plan Provided by I2A

By applying this negative influence to the IIL, a contingency route through the state space becomes the dominant path to the goal. This is shown in Figure 3.4. The Action component will perform the analysis necessary to determine that this is the best course of action and begin implementing it.

For such a simple example, the need to replan is greatly diminished since the broad input from the plan is still respected over such a small disruption, but for larger problems, as

the contingencies become dominant and the agent deviates from the plan it should be noted that the optimal plan, and therefore the information being supplied to the IIL may become obsolete, so a new plan is generated to reflect this using the Asynchronous Replanning component. For speed, this acts on the abstract representation of the world to give a more tractable problem, but is a low priority, which is why this is treated as a side process. Over time, the AR will update the IIL, but the agent can continue to execute whilst waiting for that update.

It is also worth noting that the specific type of contingency that becomes dominant may not require the use of an alternative route, but the use of some sort of system that negates the problem. In the case of the fire in the dungeon, the agent might cast a protection spell that allowed it to pass unscathed as in Figure 3.5. This highlights a particular aspect of the IIL and underlying Common Representation components which is that although the agent does not have specific knowledge of the presence of hazards in the world, it must have knowledge of how to handle those hazards. Although on the diagram it appears as though the agent respects the original path through the dungeon, in context of the Common Representation, it is actually traversing a very different set of states - namely those where the agent experiences the effects of this protection spell.

This small example shows the way that the I2A divides the problem clearly into Develop Time and Run Time aspects in order to provide good decision making whilst also retaining speed and computational simplicity during execution, when many other factors of the game are vying for processing power.

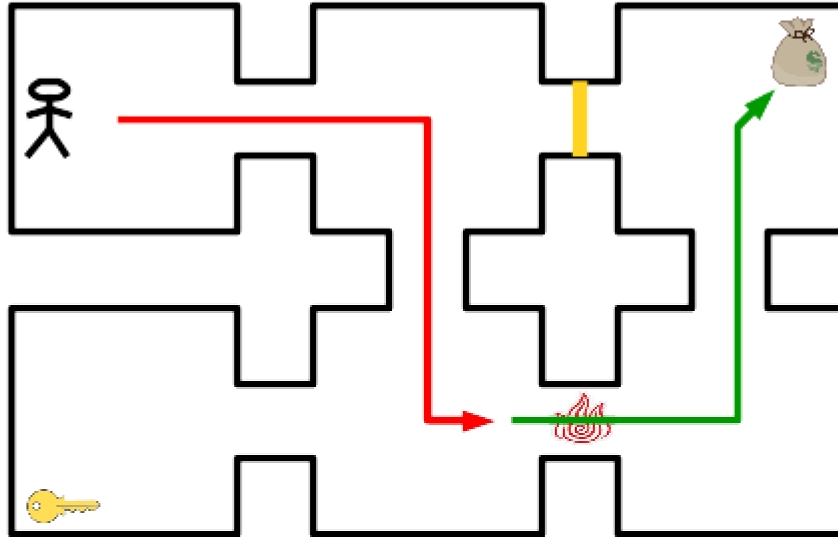

FIGURE 3.5: An Alternative Contingency

## 3.3 The Common Representation - Compiled PDDL

As mentioned one of the key aspects to bridging the gap between the Reactive and Deliberative AI paradigms is finding a way that the two can share a common representation of the world. This means that information from a reasoning system can be combined with the suggested choices offered by a deliberative system in such a way that the final arbitration and decision takes both paradigms into account.

As was noted in the previous chapter, the most frequently used representation for Automated Planning is the Planning Domain Description Language. Helmert has previously described a system for automatically compiling a PDDL representation into a SAS+ representation by using analysis of mutually exclusive facts and grouping these together to form the SAS+ variables and their associated Domain Transition Graphs[30].

The best way to explain this process is through a short example. Consider a world in which a package is moved between locations within three separate cities. There is a truck that the package is loaded into and the truck can be moved around the city it is in. To move between cities, the package must be taken by plane to another city. We

assume that the locations within the city are fully connected (from any one location, you can reach any other), and that one plane connects a specific pair of locations within two cities, which are the designated airports. This scenario is based on the classical Logistics planning domain, although tailored to highlight certain aspects of the I2A system better than the original, creating the Logistics+ variant[21]. The variation from the classical design of the domain is that traditionally a plane could fly between any pair of airports. By introducing a concept of a "flightpath" and restricting a single plane to only serve a pair of airports, this better allows the domain to express alternative arrangements and contingency plans. This domain will be discussed in greater depth in Section 4.1.1.

The PDDL representation of this world consists of a series of facts about it that are true. Some of these assertions cannot be changed and are considered to be intransitive, for example no action described would allow for a road to be removed so the road layout is intransitive. Contrastingly, some assertions are only true for this particular state of the world. For example the location that the truck is at currently will change when the appropriate action causes it to move. However, the nature of this means that it is possible to group certain facts about the world together; for each truck, it can only ever be at one location at a time, moving the truck removes one fact and asserts a new one. This is a reasonably obvious example, but a more obfuscated one can be seen by considering the location of the package. Provided there is a truck in each city, and each city is connected to another by a plane, it is possible for the package to be at any location in the world, but it is also possible for it to be at no location - when it is loaded into a truck or plane. The analysis of mutually exclusive states will group these together. Facts that have been grouped together become the values that a variable can take in the SAS+ representation. As the process is automated the meaning of each variable is not explicitly exposed. However, it is possible to deduce that a variable in a SAS+ model

does have a meaning; in the example it could represent the location of a package, or a truck.

Not only does the SAS+ representation capture the values that a variable can take, but also the way that it can change between those values. For the case of the package, the domain is constructed such that the package can never jump from one location to another, it must always be moved from one to the other by being loaded into a vehicle and then unloaded. This means that it is possible to construct a Domain Transition Graph that shows the manner in which the transitions can occur as in Figure 3.6, which shows a small example of a world with just one city and three locations.

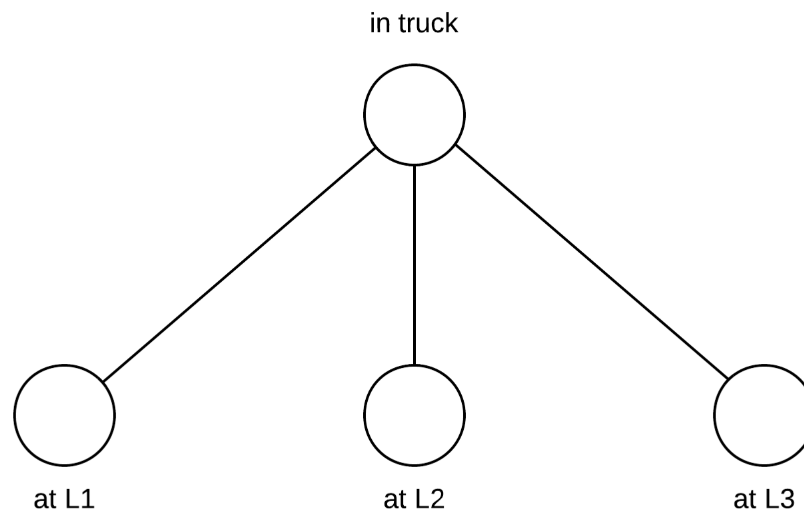


FIGURE 3.6: A simple example of a DTG for a package within one city

It is also possible to represent a graph structure as an "Adjacency Matrix" which is an $n$ x $n$ matrix of binary values (for $n$ nodes within the graph) in which the value at $(i, j)$ reflects the existence (or lack thereof) of an edge connecting node $i$ to node $j$[86]. Note that for a bi-directional graph, the adjacency matrix is required to be symmetric, but for a DTG representation, which is a directed graph, this is not the case. For the simple

| | in truck | at L1 | at L2 | at L3 |
|---|---|---|---|---|
| in truck | 0 | 1 | 1 | 1 |
| at L1 | 1 | 0 | 0 | 0 |
| at L2 | 1 | 0 | 0 | 0 |
| at L3 | 1 | 0 | 0 | 0 |

TABLE 3.1: A matrix representation of the example DTG

example DTG we could represent this using the matrix shown in Table 3.1. Note that as there are no explicit NO-OPS in the domain (although there is equally no stipulation as to the frequency at which actions must be taken in I2A), so there is no edge recorded that will link a state to itself. The adjaceny matrix representation provides a solid data structure with which to work on these problems, and also has other implications which will be discussed in Section 6.1.1.

The SAS+ formalism provides a directed graph for each of the variables it identifies. Individually, each one provides a small portion of the current world state, consequently the full state is identified by the value each of these variables takes. It is therefore possible to build a complete representation of the potential state space and the nature of the transitions within that space by evaluating the Cartesian Product (an operation denoted below by the open-square □ symbol) of all of these DTGs together as in equation 3.1.

$$DTG_1 \Box DTG_2 \Box ... \Box DTG_n \tag{3.1}$$

Following on from the simple example above, it has been shown that one of the generated DTGs captures the location of a package with relation to the locations and truck. Another would capture the location of the truck as in Figure 3.7. The product of these two provides the current world state, where the truck is and where the package is. However, there are certain restrictions on how the two variables interact - it isn't possible for example to load the package into a truck unless it is currently at the same location

as the package. By taking this into consideration, the graph shown in Figure 3.8 can be generated. It can be seen that this shares characteristics with both DTGs previously shown, in as much as the triangular pattern of the truck's DTG is replicated four times, once for each state in the package's DTG. The package can only transition when the truck is in the right location. In order to ensure that only that the Cartesian Product process has not introduced spurious edges into the graph it is necessary to validate in each that the preconditions that would need to be satisfied for the edge to be traversed in fact are satisfied in the state that the edge originates from.

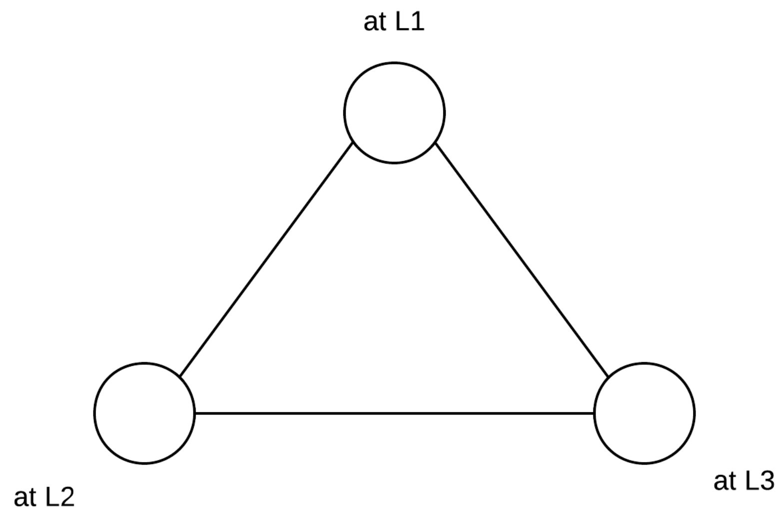


FIGURE 3.7: An example DTG for the truck

This means that from a PDDL description of a world, it is possible to automatically create an exhaustive enumeration of all the states the world may be in, and the way that the world will transition between those states.

This enumerative approach gives a consistent context that data sensed from the environment can be added to during execution, whilst data from the deliberative reasoning system - in this case a planner - can also be applied in this same context.

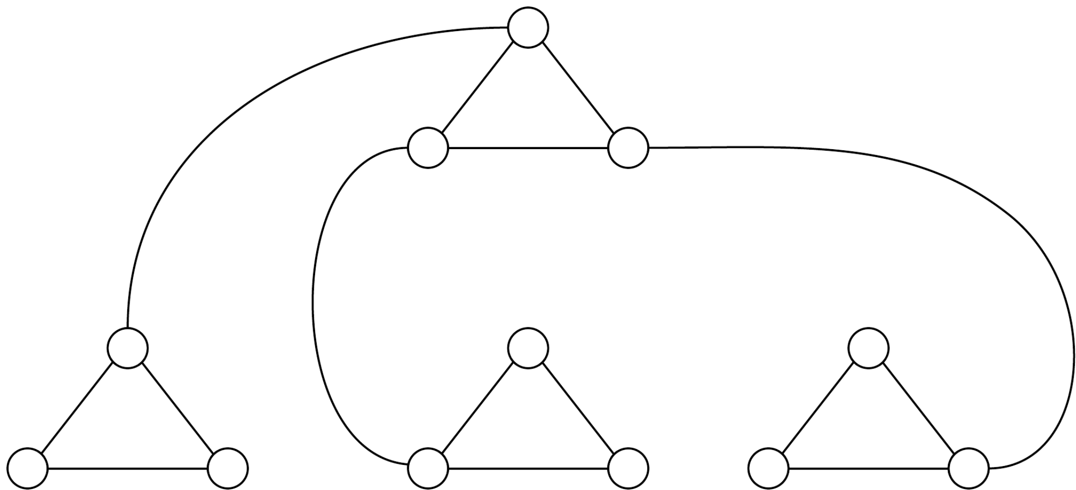

FIGURE 3.8: The Cartesian Product of the two DTGs shown

## 3.4 Abstraction of States

A fully enumerated state space will, except in trivial problems, rapidly become quite unwieldy and difficult to manipulate. However, importantly, we know that there is likely to be disruption in the long term reasoning system, due to the interactions of the agent with forces outwith its control such as the player. As a consequence there is little use in thinking in concrete terms very far into the future since circumstances are likely to change. To use a real world example, when we do any sort of reasoning about our future actions, the further away from now they are, the less concrete we are likely to be - when we go to work in a morning, we might have a plan of all the items we will need to pack for a day in the office, but our plan will then include a higher level "drive to the office" abstraction, rather than a definitive list of steps we will take when we leave our house. There are two reasons for this, one is the traditional Macro Action paradigm in which a group of actions are so often performed together that they can be grouped into a single action for expediency of planning. More interesting in the context of the I2A is the notion of flexibility, which is to say that there are a number of different routes that might

be taken in order to "drive to the office", which will change in desirability. In this way, the abstract "drive to the office" directive is actually a placeholder - it isn't possible to determine a route until factors such as traffic and time of day are considered, so in effect the placeholder acts to push a concretisation off until execution time. In many ways, this approach holds with the concepts that underpin task decomposition in Hierarchical Task Networks (discussed previously in Section 2.2.3.2). Rather than decomposing all tasks in advance however, here the emphasis is on what could be described as "just in time" task decomposition.

In order to achieve this, there is a need for an abstraction mechanism that allows for a variable granularity dependent on the distance within the state space from the current state. This allows for local states to be reasoned about concretely, whilst distant states can be thought of in the abstract.

### 3.4.1 Abstraction Through Clustering

A natural solution to this sort of problem is to consider a clustering of the state graph. Broadly such a clustering analysis allows for a systematic grouping of similar or proximate states. For this type of problem, proximity is more useful since by the nature of the underlying graph, states that are in close proximity will typically share a number of characteristics in terms of the world they represent.

There are a number of algorithms that would provide a partitioning of the graph into appropriate clusters in order to allow for the kind of abstraction sought, but for reasons that will be introduced in Section 3.5.2 a sub-family of algorithms known as "Fuzzy Clustering" is used to perform this. Whereas more traditional clustering techniques use a hard edge for each cluster, and a node is either part of one or another of the clusters,

in Fuzzy Clustering a node belongs to every cluster to a varying degree. Nodes will tend to express a strong weight towards belonging to clusters they are well within the boundary of, but those that are on the fringes will find themselves having some amount of tie to two or more clusters.

### 3.4.2 The Adapted Fuzzy c-Means Algorithm

One of the most accepted methods of performing Fuzzy Clustering is the Fuzzy c-Means algorithm which attempts to find the centre-points (or "means" - in the sense of average) of $k$ clusters within the provided data. This algorithm assumes that the distance between any pair of nodes can be calculated by treating them as spatially positioned data points and finding the length of the vector separating them.

The principal adaptation from the traditional implementation of the algorithm is altering the notion of distance from the traditional Cartesian sense to one that is applicable in context. In this case, Djikstra's algorithm[87] is used to pre-compute a lookup table of the shortest distance (in number of edges traversed) between any pair of nodes. This provides an appropriate substitute to the standard distance measure.

The Fuzzy c-Means algorithm works by randomly picking an initial placement of the centroids of the $k$ clusters being identified. Subsequently the strength with which each node belongs to each cluster is assessed based on the distance from the node to the cluster centroid, proportional to the distance that node is from the all of the other cluster centroids. With this calculated for each node, the centroids are then updated, and then the strengths are again re-evaluated, with this iterative process being repeated until the centroids and strengths stabilise between iterations.

$$\forall k, \exists C_k := C_k = argmin_a \left( \Sigma_x \left( distance(x, a) * \left( \frac{w_{xk}}{\Sigma_y(w_{yk})} \right) \right) \right) \tag{3.2}$$

$$\forall x, \forall k := w_{xk} = \Sigma_j \left( \begin{cases} \frac{distance(C_k,x)^2}{distance(C_j,x)^2} & \text{if } C_j \neq x \\ 1 & \text{otherwise} \end{cases} \right)^{-1} \tag{3.3}$$

Equation 3.2 states that for every cluster $k$, there exists a centroid for that cluster $C_k$ which is the node $a$ which minimises the sum of the distance from $a$ to every other node $x$, normalised by the strength with which node $x$ is deemed to belong to the cluster $k$, relative to the strength with which is belongs to all other clusters.

In Equation 3.3, the strength with which a node $x$ belongs to a cluster $k$, $w_{xk}$ is evaluated as the sum of the squared distance of x from the centroid of k divided by the squared distance of x from the centroid of each other cluster. This sum is then inverted. As a consequence of squaring and then inverting, the weighting used can be characterised as a normalised inverse-square relationship. Note that there are special cases when $x$ is the centroid of a cluster being assessed ($C_j$), in which case during this step of the summation, a value of 1 is used in place of the result of division. Note also that in this case, a total sum for $w_x$ may exceed 1.0, which is why in Equation 3.2 it is necessary to sum the weights of $w_x$ when determining relative weighting.

The number of clusters that are to be found can be tuned as required but a standard rule of thumb[88] is given in Equation 3.4 in which the number of clusters $k$ is given as the ceiling of the square root of half the number of nodes. This has been shown to provide reasonable solutions for the I2A, but has not been proven to be optimal, and it is possible that something more complicated would provide better results, such as an

adaptive algorithm that varies the number of clusters until the best fit of clustering is found.

$$k = \lceil \sqrt{(n/2)} \rceil \tag{3.4}$$

#### 3.4.2.1 Consequences of Operation in a Discrete Space

Due to the discrete nature of the space, the centroid of a cluster must be an existing node within the space. That creates a situation in which an imprecise clustering can be generated because the iterative change per step is insufficient to cause the centroid to change location, causing a premature stabilisation.

It is also possible for a “misclassification” to occur due to the initial random seeding, for example when two clusters start out similar they will likely converge to be identical. These cases can be detected and rejected easily, and are referred to as “Faults”.

However, it should be noted that the random seed is beneficial for overcoming the problem of discretisation in the space, since a repeated run of the algorithm will generate a potentially different result. Therefore both weaknesses of the approach can be solved trivially by a majority voting system derived from the Byzantine General problem[89] in which a number of solutions are generated and a majority consensus is taken as the true value. This has been shown to be an effective solution to the problem due in large part to the low failure rate experienced when clustering the state space.[20]

## 3.5 Types of Graph Nodes

Within the state space graph, there are a number of nodes that are important to the operation of the I2A. These nodes have significance for a number of reasons, and their effect on the state graph is distinct from that of standard nodes. For this reason, each has a specific name and is described here.

### 3.5.1 Goal Nodes

A Goal Node (GN) is a node that conforms to the goal of the task being undertaken. Note that there will generally be more than one GN in a given state space graph, since a goal is only a partial definition of those facts that must be true. This leaves a range of conditions about the world that are irrelevant, meaning a range of states when the world is fully enumerated.

A GN is obviously a state that the agent should try to attain, since when the agent reaches a GN, it has completed its task, so in effect, the whole I2A framework is designed to guide the agent to a suitable GN.

### 3.5.2 Focal Nodes

Conceptually, Focal Nodes can be thought of as states within the state space that have structural significance, or in very high-level terms they represent bottleneck states that plans will typically pass through in order to achieve meaningful activity. A good way to illustrate this is with another real-world example, this time using two islands that we want to navigate which are connected by a single bridge. We can navigate around one island relatively trivially, but if we want to go to the other island, we must pass across

the bridge, so in this example, the bridge is a Focal Node (FN). Note that conceptually this closely parallels the idea of Landmarks presented in Section 2.1.3.3, but whereas Landmarks are found by analysis of plans, FNs are found by structural analysis of the state space.

FNs are the reason that the clustering algorithm shown above must be a Fuzzy Clustering, because this allows for FNs to be found with relative ease. The clustering is effectively creating a representation of the islands mentioned above, and by determining those nodes that lie between clusters, we are implicitly identifying the "bridges" between the islands. More formally, Focal Nodes are those nodes that do not express a strong classification for one specific cluster.

Focal Nodes are used by the I2A when processing the input from a planning system, as will be explained in Section 3.6.1.

### 3.5.3 Super Nodes

A Super Node (SN) is the name given to a cluster of nodes in a partially abstracted graph of the state-space. This is an important clarification because influence must behave differently when applied to a Super Node since it is a placeholder for a neighbourhood of states rather than a single state.

It is also important for the I2A to track which states are SNs in the state graph so that it can accurately manage the abstraction as the agent traverses the graph. As distant SNs become closer to the agent, at some point they need to be replaced with a concrete representation of that area, whilst the area that the agent has moved away from can be substituted for a single SN.

## 3.6 Sources of Influence

There are two primary sources of influence in the I2A framework (although note that other sources could also be added - a detailed discussion of this is addressed in Chapter 5). The plan generated by a planning system is processed to create a source of Deliberative Influence, whilst sensing of the environment that the agent is acting in creates a source of Reactive Influence.

### 3.6.1 Deliberative Influence

Importing deliberative reasoning into the architecture involves applying a positive influence value to states in the world that the planning system thinks the agent should pass through. At a basic level, it would be sufficient to take each state that appears in the plan and give that state a positive influence score. This would create a "Royal Road" style view of the desired trajectory through the state space, an ideal path[90]. However, when circumstances of the scenario changed and an agent found itself away from this ideal, this approach would conceptually create a situation where the dominance of the deliberative plan required that the agent resume progress down this royal road as quickly as possible, without consideration for the more broader view of the state space. A good way to visualise this problem is to visualise the problem as a real-world hill climbing exercise. By bringing each state of the plan into the influence system, we would create a landscape with a pronounced ridgeline that gets us directly from the bottom to the top. However, we expect that at some point we are likely to fall off this ridge and end up somewhere else, and from that position it may not make sense to expend the effort required to climb back up the side of the ridge to get back on the Royal Road to the

summit. In fact, from where we are now, it may be that there is a better route to the top, and if we fixate on the ridgeline, we will overlook this.

This means that rather than apply influence directly to each state that the plan passes through, what is needed is to get a gist of the plan's intention at a more abstract level and use this as the source of influence. This is where the Focal Nodes become significant, since as mentioned above these are the "bridges" by which the I2A can traverse Super Node "islands". This means that the Focal Node is by its nature providing that context as to a plan's intention - any plan that passes through a specific FN intends to traverse between the islands that that FN connects.

Although the abstraction doesn't implicitly retain a sense of meaning as to what is being abstracted, it is easy to see that this approach gives a much better sense of the semantic significance of the plan. Alternative approaches not leveraging the underlying structure of the World, such as taking every Nth action in the plan as a source of influence, do not.

#### 3.6.1.1 "Active" Focal Nodes

Importantly, not every Focal Node is significant in the context of every plan, and so the I2A performs an analysis to determine which Focal Nodes the Initial Plan expects to traverse through. These are the "Active Focal Nodes" and the ones at which influence is applied in order to form the Deliberative Influence Landscape. This allows the intentions of the plan to be represented in the state space based on which of the "bridges" it will pass through.

#### 3.6.1.2 Goal Nodes

Because Goal Nodes are the target of the agent's overall execution, an amount of influence is applied to these states within the space. This provides a basic shape to the influence landscape that will generate an overarching gradient for the agent to work with and provide a steady draw from the agent to the GN.

### 3.6.2 Sources of Reactive Influence

Reactive influence is applied to the landscape based on detection from the environment and correspondingly updating the relevant states. The amount of influence a situation warrants is tunable as a parameter, allowing designers to vary the strength to create compelling reactions. Most commonly, these will apply to a specific location, therefore all states that correspond to that location under different circumstances should have an influence applied. In general, the nature of games-style domains is that these will generally be negative influence sources, which is to say, things that should be avoided. This is a broad abstraction over a number of different types of thing that could be detected in the environment such as enemies or hazards within the world. Positive influence sources might be items within the world that cannot be reasoned about deliberatively, perhaps power-ups that appear randomly or allied agents acting independently within the world.

Sources of influence detected in the environment need not necessarily be tied to location, though as noted this is the most common. Consider a scenario in which rain is detected. For an outside environment, this is effectively location-agnostic, and might apply a negative influence to all states in which the agent does not have an umbrella, biasing the agent into picking the umbrella up when rain is seen. This highlights that influence can apply to concepts as readily as to locations.

## 3.7 Influence Propagation

If Influence was only applied at the nodes it was detected, some nodes would be desirable and others not, but this wouldn't help in guiding the agent between these nodes. Instead, influence is propagated around the graph of the state space to create the landscape, which identifies not just the high and low points within the graph, but also the ascents and descents between them. This creates a unified view not just of where in the graph the agent should go, but the most appropriate methods by which it can get there. This technique has already been shown to be effective in the Influence Map approach discussed in the previous chapter. Threats are detected in a single location but the influence that threat creates is smoothed out over all nearby locations. Influence Landscapes operate on much the same principle but instead of this smoothing happening between nearby locations, it takes place across the state space.

### 3.7.1 Propagation Techniques

How this smoothing is handled is in large part specific to the implementation; many different algorithms tackle this kind of scenario and may under different circumstances generate different results. For example, it would be possible to designate each source of influence as the peak of a Gaussian distribution, and assign each node within the graph a score based on its minimum edge-distance from that influence source. When a node experiences influence from multiple sources, it may be appropriate under different circumstances for the maximal, minimal or average value to be taken, and it may in certain domains be appropriate for more exotic combination metrics to be considered, such as a summation of the various influence levels (although this is probably only useful in very specific cases since from a philosophical point of view, it would doubly-weight

a node that was halfway between two sources of influence, and give rise to that node potentially being preferred to either of the positive influence sources).

#### 3.7.1.1 Reward Sharing Propagation

Since our notion of Influence within the graph is intrinsically connected to the concept of the heuristic value of a state, it makes sense that we can apply approaches derived from those that are used in heuristic state analysis to solve the problem of propagating values around the graph. Traditionally these methods are designed for passing the value of a state upwards in a search tree so that the parent nodes that contribute to that state are also seen to be worthwhile, but since we can view a tree structure as a special type of graph, the method can be applied just as well in the context of generating the Influence Landscape.

The Reward Sharing approach to propagation updates the value of each node in the graph by starting at the sources of influence, and finding the nodes which are parents of these, which is to say those nodes having edges originating at them which terminate at the influence source. For each of these, an updated value is calculated as the reward of the successor under consideration split equally amongst all of the parents of that node as in Equation 3.5. If this value is higher than the current value that the parent node has (which is initially 0), then the value of the parent node is updated. Every node that has its value updated is added to the list of nodes that need to be considered in this manner.

$$\left\lfloor \frac{(V_n)}{Count(Parents(n))} \right\rfloor - 1 \tag{3.5}$$

In this way, the reward that a node provides, which is synonymous with its Influence value, is distributed across the graph structure.

#### 3.7.1.2 Example

Consider the following small example that will show the Reward Sharing system in practice. In this graph there are 5 nodes as shown in figure 3.9.

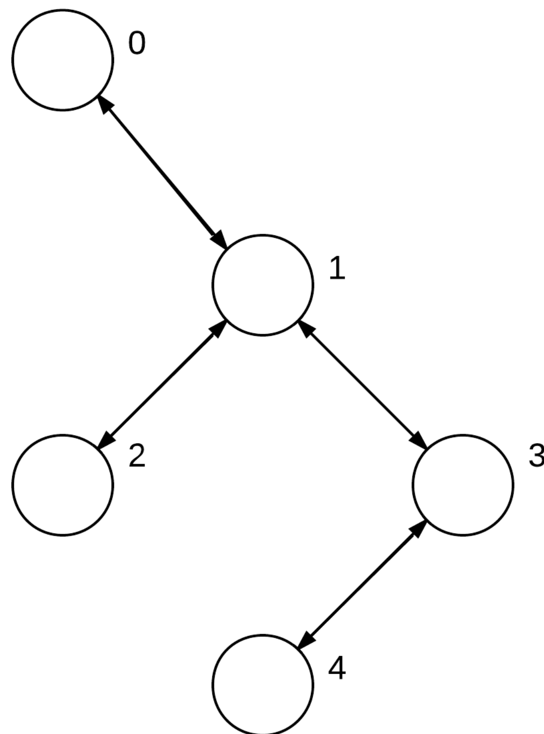


FIGURE 3.9: A Basic Scenario for Reward Propagation

This scenario describes a simple state space representation in which state 0 and state 2 connect only to state 1, 1 connects to 0, 2 and 3. 3 connects to 1 and 4, and from state 4 only state 3 is reachable. For the purposes of this example, Node 1 is the goal to be reached, making it a source of influence. It receives a positive score of 100 as a result, as shown in figure 3.10.

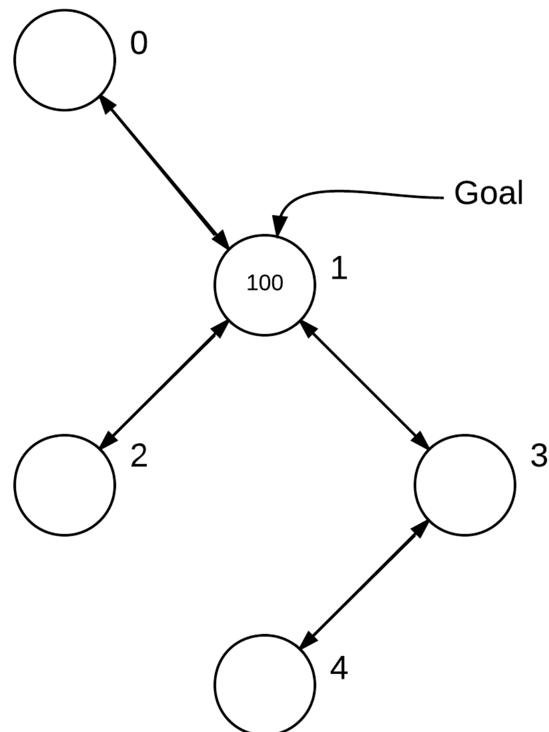


Figure 3.10: Influence Applies Directly to the Goal Node

Node 1 is the child of three nodes within the graph, Node 0, Node 2 and Node 3. By Equation 3.5 each of the parent nodes should have a reward of 32 propagated to them. As their previous score was undefined, these values of 32 get stored, and each of the three nodes is then considered for further propagation. This is shown in Figure 3.11.

For Node 0 and Node 2, their only parent is Node 1, so the equation generates a score of 31 - lower than the already recorded 100 so propagation from these nodes can be disregarded. Node 3 however also has Node 4 as a parent, and when the reward is shared, the resultant value of 15 is higher than Node 4's previously undefined value, so the new one is used. Again, for Node 1, the value of 100 is retained.

The update to Node 4 causes it to be considered for further propagation, but it will not update the value of its parent (Node 3) as the result is lower than that already recorded.

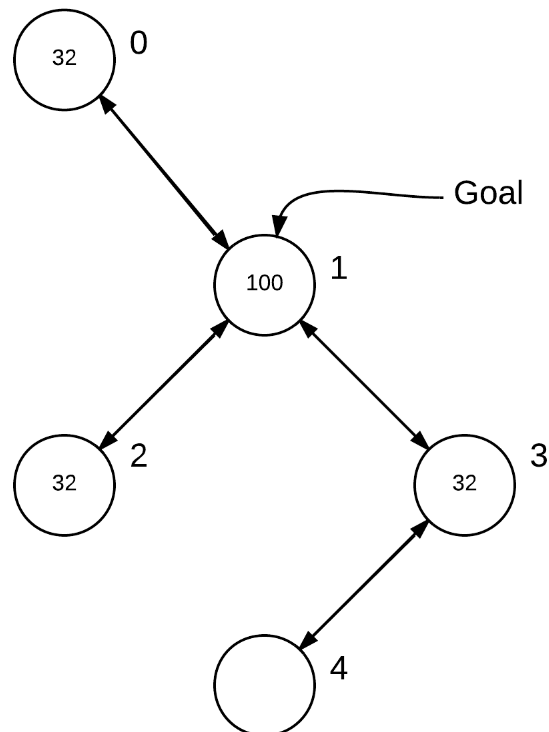


FIGURE 3.11: Results of a Single Propagation Step

At this point, there are no further candidates for propagation, and the algorithm is complete having appropriately shared the reward from Node 1 around the graph network. The results are showing in Figure 3.12. For scenarios with multiple sources of influence, they are just initialised with the appropriate number of nodes requiring consideration and the algorithm executes as normal until stability when all nodes have been considered and no updates have been made.

#### 3.7.1.3 Influence Propagation Through Super Nodes

When states are abstracted, the ability to directly apply influence to a node within the state space is lost. Influence must be applied to the whole Super Node, and the amount of influence applied must reflect the combined influence detected for that Super Node. This introduces some imprecision since even within a Super Node, there are likely to be

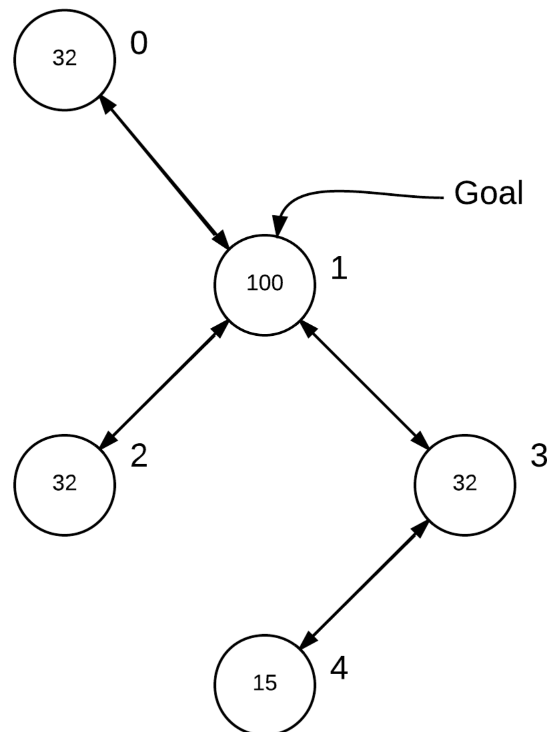


FIGURE 3.12: Completed Propagation of Influence

short-term contingencies available - even within this local group there are potentially alternative pathways through the graph structure. At an abstract level the granular representation will implicitly be lost, however some semblance of its activity can be captured if we consider the expected "per-node value" of an influence source.

The per-node value is a heuristic measure of what the expected average value of influence is likely to be, and is based on an estimated density and connectedness within the Super Node. For example, if we are using the standard cost sharing heuristic and apply a negative influence of 100, we know that firstly only one node within this cluster would have a value of -100 applied, but we also know that the amount of propagation experienced will be largely dependent on the connectedness of the sub-graph within the cluster. A fully connected subgraph will in general offer more possibilities to avoid a bad state, but the bad state will remain proximate to the states being traversed. Equally a

minimally connected subgraph - effectively a straight line of states - offers no opportunity to avoid that state. In either case, connectedness is demonstrably an important factor in how much weight should be given to an influence source.

The connectedness of a cluster can be ascertained based on its edge count. A fully connected graph will have $N^2$-N edges (recall that the state graph is a directed graph, so edge direction is salient). A minimally connected graph will have N-1 edges. Therefore for a graph with E edges, we can evaluate its connectedness using the formula below.

$$C = (E - N - 1)/(N^2 - 2N - 1) \tag{3.6}$$

This will generate a value of 1 in the case of a fully connected graph and 0 for a minimally connected graph, and a linear function between the two.

The amount of influence to be applied should naturally be related to the average of the influence applied across each of the nodes contained within the Super Node. The formula used for this, using I to refer to the amount of influence being applied to a specific node, is as follows

$$I_{s.n.} = \frac{I}{N} + \left(\left(I - \frac{I}{N}\right) \times (1 - C)\right) \tag{3.7}$$

In a fully connected graph, the impact of the influence being applied is limited solely to the average of that influence spread around the sub-graph. This is a lower value to represent that the more connected graph will have multiple paths through avoiding this node. In a minimally connected graph, at the other end of the scale, the full amount of influence is applied to the Super Node since there is no option in this case and all routes

must flow through the node being considered. Note that between these two extremes, the function is linear dependent on how connected the graph is.

Note that this formula applies both when influence is applied directly within the Super Node, and is also useful when determining the amount of influence a Focal Node should exert into the Super Node, since in this case the source of the influence doesn't matter. This formula can be used in either case to calculate the per-node value, based on either the injected influence value in the case where there is a source of influence inside a super node, or on the propagated value originating from the sharing heuristic.

## 3.8 The Integrated Influence Landscape

At this point in the process, there now exist two landscapes that have been fully propagated, one for the deliberative data that has been extracted from the plan, and the second representing the reactive data that has been sensed from the environment.

The aim of the architecture is to unify these two disparate approaches into a single view of the world, and this is the Integrated Influence Landscape (IIL), so called because it integrates the influence from different sources. Since both landscapes are structurally identical, the challenge of combining them lies solely in determining what a rational mechanic for this would be.

### 3.8.1 Mechanics for Combining Landscapes

In large part, the process by which multiple landscapes can be combined is left as a decision to be taken by implementors of the I2A - dependent on the context, different processes create different agents which may or may not behave appropriately. In the

most basic case, simply adding the two values of a node to create its value in the IIL may be an acceptable approach. It may also be that taking the larger of the two is under certain circumstances the most appropriate technique, creating an agent that overestimates the likely value of nodes within the search space. The I2A is designed to be flexible and allow for experimentation to find techniques that create agents consistent with the kind of behaviour the designer is looking for.

## 3.9 Implicit Contingency Planning

Under ideal conditions, the plan and the influence landscape generated from that plan will describe the critical path through the state space in order to reach the closest goal state from the starting point. However, as discussed, the underlying assumptions of the planner mean that the conditions encountered in a game world - or in any world beyond sterile tightly controlled environments - are rarely ideal. The interaction of the Deliberative Influence and the Reactive Influence allows for this to be handled intuitively within the single architecture. As a consequence, as the circumstances that an agent finds itself encountering deviate from that which is expected, an I2A agent is able to adapt to this and find alternative routes across the state space that are now more desirable.

### 3.9.1 The Nature of Contingencies

There are, broadly speaking, two reasons a contingency is required: because an action within the plan will fail, or because, due to changes in the environment, an alternative approach is of more value. The distinction can be best exemplified by considering a scenario in which the plan calls for air travel on a specific flight.

- "Roadblocks" occur when we cannot execute a required action in the plan for some reason. In the example scenario, consider the case that the required flight is cancelled. If this were to happen we cannot rely on that step within the plan, and so will need to reconsider our actions.

- "Preferences" occur when external elements disrupt the world such that although the planned action is still possible, it is not the best way of achieving the agent's goals. Again referring to the example scenario, assume instead that when booking the required flight, it is found to be prohibitively expensive. This new information makes this flight less desirable, and there will come a tipping point where the cost of the flight outweighs the inconvenience of other flights or other modes of transport.

In both cases, it is clear that to work around the problem will require considering the use of a different path through the state space. These different paths are contingencies, and as previously discussed, are traditionally found by halting execution and re-evaluating the deliberative component of the problem. However, in either case, the I2A is designed to overcome these quickly using simple and efficient arithmetic operations instead of resorting to re-solving a search problem, a significant advantage over other systems.

### 3.9.2 Finding Contingencies

The Integrated Influence Landscape, by its nature, implicitly contains all the potential contingencies that are available in the state space, since all possible routes through that space are contained in the graph representation. As the influence values are updated during execution, the best path through the space changes, but crucially it remains clear how that path traverses the space.

### 3.9.3 Functional Equivalence of Focal Nodes

One advantageous aspect of adopting the abstracted representation from the Fuzzy Clustering algorithm described above is that we have the classification of each Focal Node, which is a mathematical representation of which Super Nodes each FN lies between. Two FNs with a similar classification will be connected to the same SNs, and as such are described as "Functionally Equivalent" inasmuch as these FNs provide a route between the same sets of SNs. Consider another transportation example from the real world in which the domain involves traveling around and between two cities. It could be argued that each city might form its own cluster of nodes, thus making each one a Super Node, but the ways that it is possible to travel between the cities, perhaps a car journey or a plane ride, could be represented as two Focal Nodes connecting the two SNs. Here is a prime example of this functional equivalence - although there is a choice to be made, and each may have positive and negative effects to be considered when evaluating the two, in terms of the structure of the domain, the two are, at an abstract level, providing the same service broadly speaking, in that they are providing a link between these two SNs.

This is significant as again this provides another way of finding contingencies without requiring a new solution to the planning problem. In circumstances where a FN is not reachable, Functionally Equivalent FNs (FEFN) can be used in place and provide identical sources of influence as the original FN. This allows an FN that is "active" as defined above to be modified during execution, and for the deliberative influence to be modified during run time and updated in a very simple manner as it becomes clear that the plan is not going to be able to be followed. Importantly, although this is not a robust solution, since the rate of occurrence of FEFNs will vary between domains, it should be

noted that this is another computationally "free" solution that is derived as part of a process already being undertaken, namely the identification of the FNs themselves.

## 3.10 The I2A Executive

Putting all of these components together creates an analysis of the world that is powered by the Integrated Influence Architecture. Crucially, an awful lot of the processing can be done upfront during development and stored, meaning that the cost at run-time is significantly reduced. This is an important aspect because as was shown in the previous chapter, traditional methods for performing the kinds of long-term reasoning necessary are usually significantly computationally expensive and not viable to be used alongside the other intensive tasks that a computer game typically requires.

However, in order to effect decisions based on the I2A system, it must be coupled with an Executive, as was seen in the previous chapter. Whereas Gat's Three Layer Architecture arbitrated between components to determine the most appropriate choice, the I2A system uses the "integrated influence" of these components to make a choice informed by both.

### 3.10.1 Action Choice

In the most basic terms, an Executive must choose what action to take. Naively, for the I2A this is simply an evaluation of the values of the states that are immediately reachable from the current node in the IIL, which as a look-up process in an existing structure is computationally trivial. This is however quite a short-sighted process that requires significant reliance on the IIL producing a monotonic or strictly increasing landscape, which cannot be guaranteed by the functions that generate the IIL.

At the other end of the spectrum computationally would be trying to find a best path through the graph to the nearest goal node. However, this reintroduces the notion of over-reliance on search that the I2A was designed to avoid, meaning that the computational complexity reduction is lost. This is clearly not a good strategy.

As a compromise a good approach would seem to be some form of limited look ahead using a localised search.

#### 3.10.1.1 Localised Expansion-bound A* Search

A* search (introduced in Section 2.1.2.1) is typically used in order to find the shortest path between two nodes in a graph, but as a best-first strategy and with a modified heuristic, it can instead do a localised search to perform an expanding-radius best-first search looking for the node in the state space that has the highest value in the IIL.

In order to make a computationally tractable search, the number of nodes that can be expanded as part of the algorithm is bound and when this limit is reached, the node with the best value is chosen as the agent's current short-term objective. Due to the abstract nature of the IIL, the algorithm is prohibited from replacing a Super Node with its concrete equivalent as it is unrealistic to assume that a Super Node represents a single step of radius expansion (and for typical limits, it is unlikely that the radius would expand to the neighbourhood on the far side of the Super Node). This ensures that the Executive can choose to enter a SN by traversing the entry edge to the SN but may not then expect to exit the SN during this execution step. The SN will then be substituted naturally after which the Executive can traverse the nodes contained with the SN as it has now been replaced by the more detailed representation comprised of standard nodes.

A simple heuristic for the search is to take the IIL value for a node, but it is worth noting that other alternatives could be more effective or provide more appropriate behaviours under certain circumstances. The obvious modification is to discount the value of a node based on its distance from the current node, which would serve to model the potential for disruption faced by the agent as it attempts to make choices based on data that may later be updated.

### 3.10.2 Acting

Having used the information from the IIL and used some method of action choice, the immediate action required for the agent to take will be apparent, whether that is the first action that will lead eventually to the state the Executive has identified as being the short-term goal, or that is the single action that the Executive has determined as being appropriate.

This action is passed to the game world in order for execution to be taken, and the agent moved or otherwise for a change to effected.

## 3.11 Summary

This chapter has presented the proposed components of the Integrated Influence Architecture and the mechanism by which it functions. The fundamental philosophy of the I2A system is that it incorporates both deliberative and reactive paradigms concurrently. The process hinges on leveraging resources during development of the game in order to generate a representation of the world that can be used by both reactive and deliberative systems at the same time.

Because of this, the I2A is able to quickly begin directing the behaviour of a non-player character, and do so in a way that allows it react to changing circumstances in an intelligent manner, that is respectful of long-term goals. The I2A implicitly exposes contingency plans and alternative ways of completing objectives to provide a robust architecture that offers a more efficient approach to agent activity in a dynamic environment such as that found in a video game setting.

# Chapter 4

# Evaluation - Functionality and Viability

In the initial research statement presented in Section 1.3, four criteria for evaluation were presented. This chapter will address the initial two of these, namely the functionality and viability of the I2A as a novel approach to decision making in NPC characters. In order to prove the functionality, it is necessary to demonstrate that the theory is sound, which is to say that the process by which the I2A functions to turn a PDDL representation into its own Common Representation formulation, to create an abstraction of this using the clustering technique and the ability to plan on this more abstract space. Here each of these processes will be verified to highlight that I2A can achieve what has been described above. Many of the results here were presented in previously published work[21].

To establish the viability of the technique, the second section of this chapter will focus on an implementation of the I2A within the game development tool Unity, one of the most popular engines for game development[91][1]. This is done to show that not only is

[1]http://unity3d.com

the work presented sound, but also that it is relevant to current trends in industry, and could have impact on the state of the art used by practitioners.

## 4.1 Results of Processing

In order to demonstrate the viability of the I2A from a theoretical point of view, a number of experiments were performed. These broadly were designed to show that it was possible to reformulate a planning problem in the manner that the I2A requires[20]. It was also important to highlight that although moving towards an enumeration of the state space gives a combinatorial explosion in problem size, using a clustering analysis allows for this to be managed in a way that is not cost prohibitive[21].

In both cases a set of scenario problems were used that were derived from the Logistics+ domain. Note also that for the domain preprocessing phase, a range of domains were also tested in order to verify that the technique was broadly applicable and that some aspect of Logistics+ didn't make it uniquely suited to this kind of approach.

### 4.1.1 Logistics+ Problems

The Logistics+ domain is a version of classic planning problems in which packages must be delivered to locations. This is a very standard collection of problems and many variants exist such as Driverlog (in which the trucks that move the packages around also require drivers who must move around the world) and Logistics which groups locations into cities and then utilises aeroplanes which can travel between any pair of specially designated locations within cities which act as a model of an airport. In Logistics+ we identify that although in certain implementations, it makes sense for planes to operate in this manner (modelling, effectively, the ability to charter a plane

between two destinations) a more typical scenario has planes serving specific routes between cities, and this is the variant represented by Logistics+. This is useful in the context of the I2A because this highlights alternative routing and contingency planning; choices for an air route that were initially sub-optimal can become at runtime more desirable because of facets of the problem that were not exposed to the deliberative reasoning system. In addition, this kind of problem is one that is frequently encountered in games albeit at a somewhat abstract level; many tasks within game environments are effectively represented from a reasoning point of view as a sequence of sub-tasks involving bringing the correct item to the correct place, often in the context of multiple items needing to be positioned correctly. This holds both for short term, for example bringing the correct key to open a door and long term, such as bringing the One Ring to Mount Doom. The experience of the gameplay itself might be different, but many games can be boiled down to this sort of reasoning, making a domain like Logistics+ a very useful abstraction.

#### 4.1.1.1 Decoupling Worlds and Problems

In designing sample problems for the I2A within the Logistics+ domain we reasoned that more often than not we would see the same game world (or level) being used by an agent who tried to achieve multiple sets of objectives sequentially, or by two or more agents acting within the same game world. As such, it makes sense to separate the unchanging elements that describe the environment from those that make up the specific instance in order to reduce the amount of computation that gets duplicated.

This is a subtle distinction from the manner in which a planning problem is usually stated, being typically framed as a Domain and a Problem, with the former describing

possible interactions and types of object within the world, and the latter specifying how the world is put together, the state of the world initially and the goal of the task.

Although the I2A relies on a PDDL representation and a standard problem statement, it is conceptually useful to recognise this distinction since fundamentally a significant amount of the processing the I2A performs is based on these intransitives elements that form the specific layout of the problem, which is termed a World rather than the Problem in the traditional way. Consider a real-world delivery problem, although the specifics of what packages will need to be delivered where will change day to day, there will be a range of characteristics of each problem that are fixed, for example the road layout and number of trucks available. Because the layout of the World does not change between problems, a significant amount of the I2A computation can be reused, reducing the computational requirement further.

#### 4.1.1.2 The Logistics+ Worlds

For demonstrative purposes, five distinct worlds have been created for the example Logistics+ problems:

- World 1 : This world contains 3 cities, each of which comprises 3 locations. Each city contains a truck and two airports, which link respectively to the other cities. There are two packages in this world.
- World 2 : This world also contains 3 cities, but only two of these have 3 locations. The other has 7 locations, making for an imbalanced graph. The larger city is also served by a second truck for a total of 4. There remain 3 planes and 2 packages.
- World 3 : A smaller world, this version has only 2 cities, each of which is made of 3 locations. 1 plane links the cities, which each have a truck. There are 2 packages.

- World 4 : This is an imbalanced version of World 3, where one city has 4 locations and the other 5 for a total of 9.
- World 5 : This world has the same basic structure as World 4 but contains 3 packages.

There are three different goal conditions that can be applied in all five worlds, allowing for the generation of a valid set of benchmarks by combining each world with each goal. It should be noted that in order for the translation process to not eliminate redundant variables, in worlds with packages that are not required to move, we include a dummy goal that the package remain at its initial location.

- Easy goals : In this set of goal conditions, a single package must be moved to a location other than that which is starts at. For consistency, this is set to be *City2Location3*, a location that occurs in all worlds.
- Medium goals : These goal conditions require that two packages be moved, with the second package moving from *City2Location1* to *City1Location2*.
- Hard goals : The final set of goals require two packages be moved as before and also dictate final locations for one of the trucks and one of the planes, *City1Location1* and *City1Location3* respectively.

### 4.1.2 Domain Preprocessing

The first step towards making these Logistics+ problems work with the I2A is to translate them into the Common Representation. The way do this is to reframe them from PDDL into SAS+ which is a relatively trivial matter using Helmert's translation system designed for Fast Downward[41]. The output from this is loaded into a Python script

built on top of Muise's KRToolkit[2] which extracts the DTG information from the SAS+ encoding and reformulates it as a set of simple graph representations, one for each DTG. This is visualised for the DTG for the package in Figure 4.1, which has been shaded to highlight the three distinct cities emerging within the DTG structure.

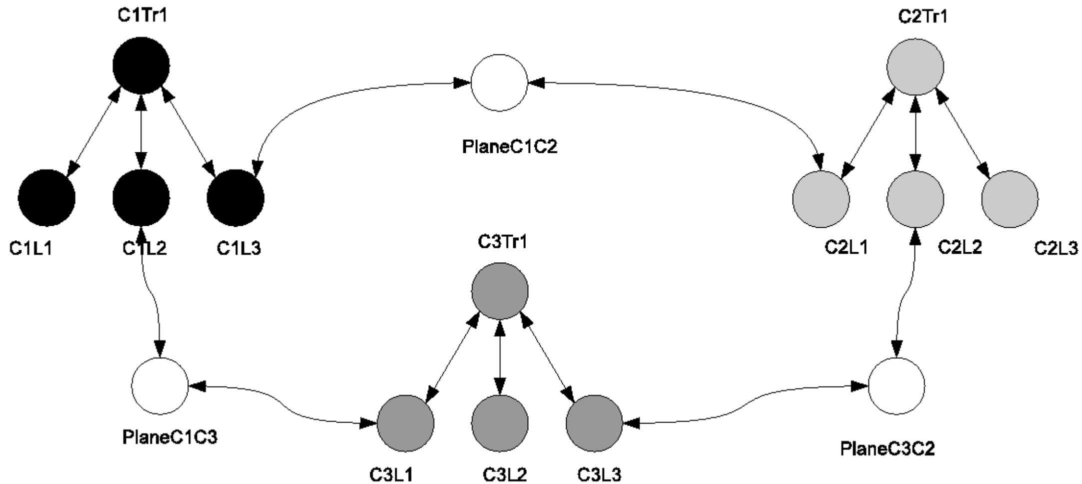


FIGURE 4.1: The *Package1* DTG from World 1

This representation can then be imported into an application and manipulated. In Figure 4.2 the DTG shown in Figure 4.1 has been loaded into a Java application. The diamond shape denotes the initial location of the package (*City1Location2*), whilst the rectangle represents the goal stat (*City2Location3*) This allows for the DTG generated to be visualised (by processing through Graphviz[3]) for verification. Note that as a holdover from the processing, each DTG has a node titled "None" which is unconnected. This refers to the state in which that variable has been assigned no value, and is not connected to the remainder of the graph since although Helmert's translation system allows for this to be a theoretical case, it is not a state that occurs in practice.

The Common Representation is the result of the Cartesian Product of each of the DTGs. In World 1 there are two packages, three trucks and three planes. Each package can take one of 15 states, each truck one of three and each plane one of two, meaning that

[2]https://bitbucket.org/haz/krtoolkit/overview
[3]http://www.graphviz.org/

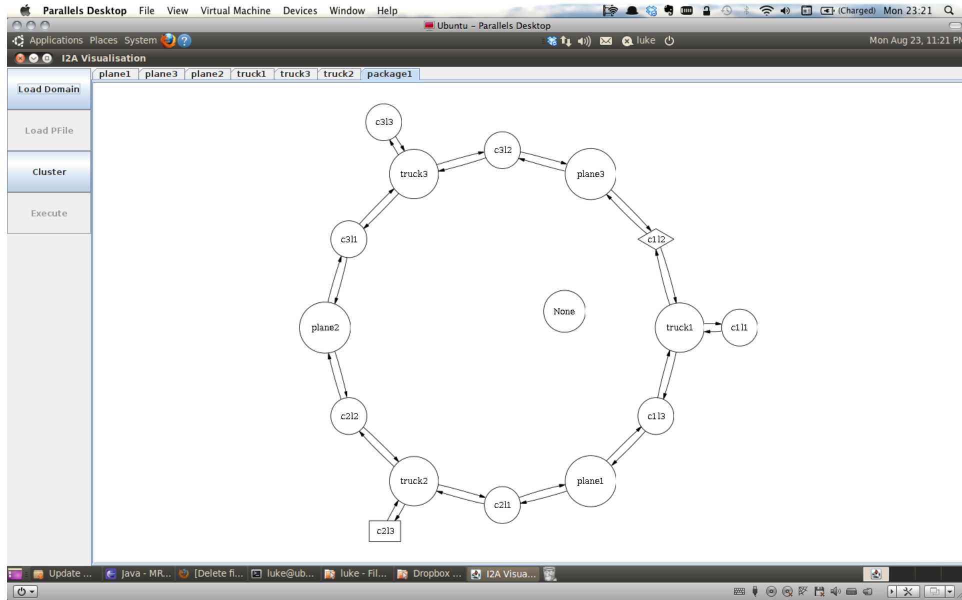


FIGURE 4.2: An in-application visualisation of the *Package1* DTG from World 1

the Common Representation has $15 * 15 * 3 * 3 * 3 * 2 * 2 * 2 = 48,600$ possible states (too large to attempt to visualise). As mentioned in Section 3.3, it is necessary after this step to perform a process of edge elimination to remove those edges introduced by the Cartesian Product operation that do not have their preconditions satisfied.

### 4.1.3 Clustering Analysis

Because of the number of states involved in the Common Representation, it becomes necessary to apply a clustering to create an abstract representation that is more easily managed. As was explained in Section 3.4 the method by which a clustering is produced is an iterative process using the Fuzzy c-Means algorithm.

The fact that clustering is possible across a graph representation is not inherently an interesting result, but as was discussed in Section 3.4.2.1, Faults can occur due to the non-deterministic nature of the initial mean choice, and the discrete state space involved. The rate at which these faults occur in each Logistics+ World is shown in Table 4.1.

| World # | Nodes | Time taken to cluster | Number of Faults Detected (Avg) | Average Time Per Cluster Attempt |
|---|---|---|---|---|
| 1 | 48,600 | 0.83 ms | 0.33 | 0.62 ms |
| 2 | 1,411,200 | 7.40 ms | 8.85 | 0.75 ms |
| 3 | 1,458 | 0.57 ms | 1.24 | 0.25 ms |
| 4 | 5,760 | 0.71 ms | 1.5 | 0.28 ms |
| 5 | 69,120 | 1.06 ms | 2.32 | 0.32 ms |

Table 4.1: Results of Clustering Algorithm (Averaged Over 500 Iterations)

This table also shows the total time taken for each world to be clustered successfully. Although the intention is for this process to be undertaken during development, it is important to point out that it is not a computationally complex process. When factoring the fault rate against the average time taken to generate a successful clustering, an indication of how rapidly the Common Representation can be manipulated is shown. Note that due to being less than a single millisecond, these numbers are unlikely to be especially precise.

What is important to note is that World 2 is a significantly larger problem, stemming from the increased size of the DTGs for each package. The relevant DTG is shown in Figure 4.3. With the addition of 5 states per Package DTG raising the total number to 20, and with two trucks with a seven state DTG, the true impact of the combinatorial explosion can be seen in this DTG, which serves to explain both the longer time taken per clustering attempt and also the larger number of faults present during attempts to cluster this World.

World 1 is quite a well-behaved world in that it is formed from quite symmetrical graphs and has a relatively small explosion. The results seem to indicate that this gives a better fault tolerance. World 5 seems to also demonstrate that the speed of clustering is not solely determined by the number of nodes, but perhaps as World 5 is generated by the

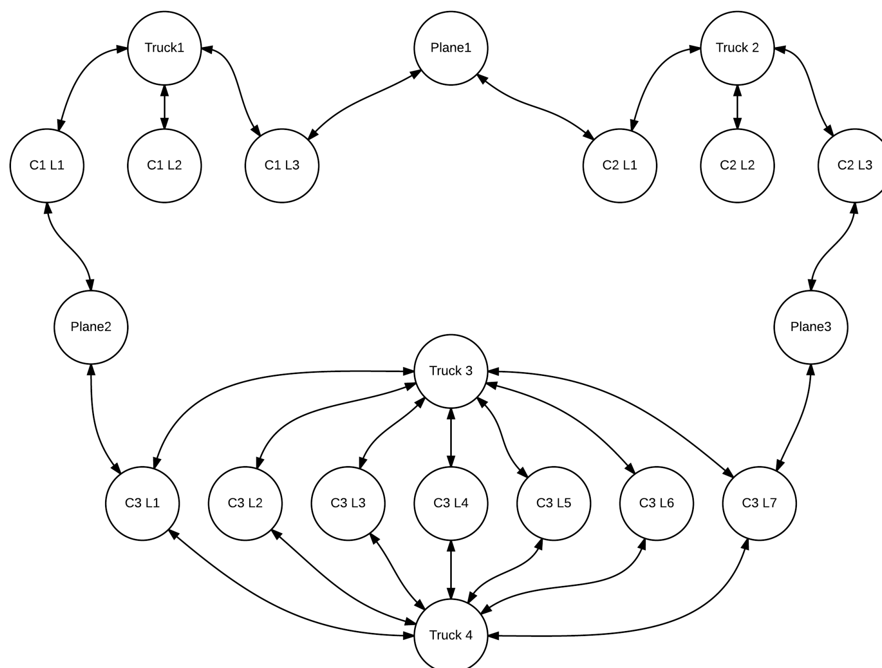


FIGURE 4.3: The *Package1* DTG from World 2

composition of more DTGs, this plays a role in the complexity required to perform the clustering.

It was also felt to be relevant to test this process on a range of other pre-existing domains from planning literature to verify that Logistics+ was not in some way uniquely suited to this type of process. The results of this are shown in Table 4.2. This shows that there are two other domains that can be clustered in this way, Satellite and Zeno. Others are significantly more challenging. The results for Freecell are perhaps the most indicative of the reasoning for this.

The Freecell domain models the popular card game of the same name. A normal deck of cards is dealt in a layout similar to traditional solitaire but face up (meaning that there is perfect information). The aim of the game is move the cards into four ordered and suited piles, by moving only the top card of each stack. The name comes from four placeholder positions that can hold a single card, these are the "free cells".

| Domain | Successful Clustering % |
|---|---|
| Freecell | 0% |
| Rovers | 20% |
| PipesNT | 52% |
| Driverlog | 70% |
| Depots | 81% |
| Satellite | 100% |
| Zeno | 100% |
| Logistics+ | 100% |

TABLE 4.2: Results of Clustering on Different Domains

This is inherently a domain that has a large number of DTGs that have fewer nodes, since each card in the deck will be represented by its own, reasonably substantial DTG. This compounds the combinatorial explosions, and an inspection of those Rovers, PipesNT and Driverlog problems where the clustering fails suggests that this is also the case in these instances as well. This is potentially a broader symptom of the combinatorial explosion, that although the intention of the I2A is to mitigate the impact of the combinatorial explosion at runtime, it seems that it may be having an impact during the preprocessing phase. Further investigation will be required to determine if this can be overcome by optimisation in the clustering algorithm to diminish the memory footprint required.

### 4.1.4 Time to Execution Analysis

Of particular interest are the potential gains that this clustering provides in terms of the computational speed-up, and the reduced complexity of the problem. This is useful because an abstract solution to the problem provides sufficient information to apply influence to relevant Focal Nodes at runtime. Recall that Focal Nodes will never be abstracted as they exist between the clusters.

The LAMA planner was used both as a benchmark acting on the original PDDL files, and in a modified form using the generated Common Representation. Because of the nature of the input and output systems for this, and specifically because the preprocessing was done inline, involving intermediate writes to the file system, it was not felt appropriate to compare the execution times directly. Instead a comparison was drawn based on the number of nodes within the search tree each planning system expanded, which provides a suitable metric for direct comparison and shows the reduction in computational complexity more clearly. The resulting plans from both systems were validated subsequently to ensure correctness.

Table 4.3 shows the results of this analysis. In all the scenarios presented, the abstraction provided the expected complexity reduction to show a significant speed increase. World 2 consistently required the most search for a solution, which is reasonable since it has the most nodes by a large factor, and consequently the most clusters to search. It also is expected for all worlds under the Easy set of problems to have reasonably similar number of expansions required since the differences between them are somewhat trivial in the context of a package traversing from City 1 to City 2 - the complexity of the domain in World 2 is actually largely irrelevant since much of this comes from City 3, which in the Easy goal does not feature in an optimal plan. This will be discussed in more theoretical terms in Section 5.2.2.2 below.

## 4.2 Unity Case Study

One of the core aspects of the I2A is that it is applicable in real scenarios, and one of the best ways to demonstrate this is to highlight its compatibility with a commonly used game engine. Unity is one of the most popular game engines currently available,

| | World # | Node Expansions to Abstracted Solution | Node Expansions to LAMA Solution |
|---|---|---|---|
| Easy | 1 | 3 | 28 |
| | 2 | 4 | 28 |
| | 3 | 2 | 12 |
| | 4 | 4 | 12 |
| | 5 | 4 | 12 |
| Medium | 1 | 6 | 46 |
| | 2 | 9 | 46 |
| | 3 | 3 | 24 |
| | 4 | 5 | 24 |
| | 5 | 3 | 25 |
| Hard | 1 | 4 | 69 |
| | 2 | 11 | 69 |
| | 3 | 9 | 44 |
| | 4 | 3 | 44 |
| | 5 | 5 | 28 |

Table 4.3: Results of Abstraction of Search Space

with numerous games being developed using it from small independent titles such as Easy Money (Robot Overlord Games, 2015) to large-scale productions such as Pillars of Eternity (Obsidian Entertainment, 2014) which raised almost $4,000,000 through Kickstarter, becoming the most widely funded game project to raise money through this platform[92]. This breadth of adoption at all levels of industry makes it a natural choice to develop a selection of components to highlight the manner in which the I2A could be implemented.

### 4.2.1 Unity Overview

Unity is designed to be relatively straightforward to begin development with. There are broadly two aspects to the system, the Unity Editor, a powerful visual tool for developing games, and the Unity Runtime, which is a series of APIs and libraries that provide a computing environment for those games to execute in. Basic shapes and elements can be added to a level relatively easily using a drag and drop interface, but

the primary power of Unity comes from its ability to execute scripts, written in either Javascript (technically a proprietary version of this language commonly referred to as UnityScript), C# or Boo (a language developed to conform to Microsoft's Common Language Infrastructure whilst using a Python-based syntax).

As a powerful contemporary game engine, Unity is quite a complicated system. The following sections will provide a high-level overview of its approach in order to give context for the work done inside Unity for the I2A.

#### 4.2.1.1 GameObjects

The core of Unity's representation of the game is known as a GameObject. All elements within the world are GameObjects, and can be moved around a 3D coordinate system. GameObjects can be organised such that they nest within another GameObject, in which case they can be moved around a coordinate system whose origin is the parent's position. GameObjects are named, and can also be "tagged". These are both primarily systems that allow the GameObject to be referenced from another element within the GameWorld. Names and tags do not have to be unique, but each GameObject can only have one.

Unity holds a number of basic GameObject types internally. This allows for very simple creation of basic 3D shapes such as cubes and spheres, and the creation of commonly used things such as lights, physics colliders, terrain objects and GUI elements.

GameObjects that will be reused multiple times, or that are to be exported between projects, can be stored as "Prefabricated Objects" or prefabs. This is a powerful technique since not only does it allow for a reduction in the amount of configuration a

GameObject requires, but it also means that GameObjects can be instantiated programmatically at runtime. For example, a bullet object can be stored as a prefab and then instantiated as a clone of the original bullet whenever a gun is fired, or an enemy soldier can be instantiated whenever a new enemy is encountered, using the prefab as a template.

#### 4.2.1.2 Component-based Architectures

Unity uses what is referred to as a Component-based Architecture, which is what allows it to use the GameObject as an all-purpose holder for anything within the world. The differentiation between a cube, a light and an audio source comes from the components that are attached to each GameObject to add to or alter its behaviour in the game world.

This is an exceptionally robust approach since it means that the elements in the game world are really the result of a composition of one or more components, which gives a lot of scope for diversity and flexibility. As an example, although both the cube and the light are primitive GameObjects, it is quite possible to quickly create a cube that is also a light by adding the appropriate components. In this case, the "box collider" component gives the GameObject the physical properties of a box, a "mesh filter" and "mesh render" give it the visual appearance of a box, and the "light" component, which allows it to emit light into the world.

This flexibility is expanded significantly by the fact that in Unity, components can be added from the online "Asset Store" as well as developed locally, meaning that GameObjects can be made to do specifically what is required within the game with

relative ease, with the emphasis being on creating that behaviour rather than developing the infrastructure to enable it.

#### 4.2.1.3 MonoBehaviours

One of the prime aspects of the Unity infrastructure, and the way that many components are created is through the use of MonoBehaviours. Technically a MonoBehaviour is a superclass from which scripts can inherit in order to be able to perform actions as components. This means that it is possible to create a script in any of the supported languages, inherit from this class and be in a position where the basic hooks for interacting with the Unity environment are already present.

Specifically, a script derived from the MonoBehaviour class can have a range of functions created that will trigger at certain points in the execution of the game's main loop. When the GameObject is first instantiated in the world, it will trigger each of its contained component's *Start()* function and each frame every component will have its *Update()* function called, and many more such functions are triggered at the appropriate point in the game's execution.

The combination of the component approach to game development, coupled with the MonoBehaviour class system provides a very powerful framework from which to build complicated yet robust behaviour of various GameObjects, and gives a great degree of control to the developer.

#### 4.2.1.4 Editor Extensions

One of Unity's most powerful features is that the Unity Editor itself is running with the Unity Runtime. This means that it is possible to create what are known as "Editor

Extensions" which allow for additional functionality to be added to the Editor itself to support new toolchains or workflows.

One of the most common of these extensions is to create a custom "Inspector", a panel which exposes a component to the game designer so they can manipulate values without having to modify the code. The default inspector is very general-purpose which is great in some situations, but often there is a need to present the information in a different way or allow different interactions. Sometimes the issue is that the designer needs to alter a complex and purpose-built datatype, and again this can be handled with a custom inspector.

Less often but still quite common is that an extension will provide programmatic functionality that is useful during development, often providing a way to automate tasks or otherwise simplify the process. Examples include providing in-editor access to cloud-based data storage such as an entire suite of extensions allowing for the development of Behaviour Trees visually such as "Behave"[93].

Editor Extensions allow for new features to be developed for Unity in such a way that they can be used by those not intimately familiar with the code underlying their operation, making them much more usable and valuable as part of the development process.

### 4.2.2 Developing an Agent in Unity with I2A

In order to create an NPC within a game developed in Unity, it is first and foremost important to make the game. The I2A is intended to function as a decision logic system for NPCs within the game, so this is perhaps the most time consuming aspect of the process. As a placeholder during development, other more simple decision systems can be used. As Maxis's Dan Kline observed during his keynote at the 2011 Paris Game AI

Conference, a great way to approach the design of a new AI system in a game is to use the rule “0, 1, Rand(), Game AI”[94], which is to say that initially the system should be off, then its components should be tested as always on to ensure each works correctly, before testing that they can be used in different sequences and finally an intelligent system determines how to trigger the components meaningfully. This is more directly applicable to systems like Behaviour Trees which can be decomposed into individual behaviours and approached in this manner, but it also applies to the overall agent system as a whole.

Developing the runtime portion of the I2A as a script to execute within Unity is reasonably straightforward. An internal representation of the Common Representation must be developed, and a method for influence updates to be applied and propagated through that representation. Its also necessary to implement a local search algorithm to choose the most appropriate action to take. The majority of this work can be undertaken with relative ease since it is almost exclusively development of a single component to achieve a specific set of criteria.

The more challenging aspect of the development of an I2A agent is connecting this logic to the broader game world, since much of this is determined by the constraints imposed by Unity. There are four specific hurdles that must be overcome; firstly, the game world must be exposed to the I2A system in such a way that it can create the Common Representation, which requires that the world be described symbolically. There must also be a way that instructions from the I2A system can be appropriately passed through the game world in order for an action to take place. It must also be possible that a plan can be made and updated using the PDDL representation the game world is generating, by making use of an external planning system. Finally, since the expectation of the I2A is that there will be some form of unintended consequences or prevention of the

successful completion of actions, it is necessary to ensure that there is a level of execution monitoring in place to ensure that these situations are detected and compensated for.

### 4.2.3 Creating a PDDL Game Representation

One of the first challenges to be overcome when trying to implement the I2A is the manner in which the game can be represented as a PDDL problem. There needs to be some way to not only convert the scenario as laid out in the Unity editor into the initial description used by the system, but also to maintain that representation as actions are taken within the game.

The system implemented in Unity to handle this relies on two aspects, firstly a GameObject called the PDDLManager, and secondly a component MonoBehaviour called a PDDLObject, which is added to all elements of the game that need to be described as part of the PDDL representation.

#### 4.2.3.1 The PDDLManager GameObject

The PDDLManager is a single GameObject with an associated PDDLManager script component. It is intended to be a unique element within the game world and as such is also tagged as a "PDDLManager" to enable other scripts to find the appropriate GameObject and reference it. There should only ever be a single GameObject that uses this tag in order to facilitate this.

The inspector window for the PDDLManager is shown in Figure 4.4. The Actions object holds a set of elements. In the Figure you can see that this world has a single action *Move* which has three parameters. Before the *Move* action can be applied, the fact *at a_crate loc1* must be true, and after this will not be true, and it will be asserted that

*at a_crate loc2* is now true. This is just a slightly different format than a traditional PDDL representation, designed to be slightly easier to edit without specific knowledge of the language.

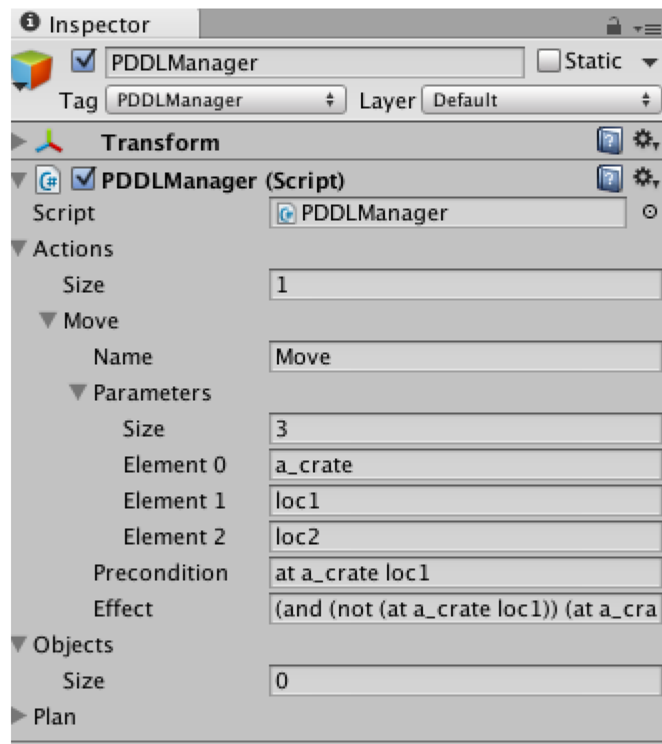


FIGURE 4.4: The PDDLManager Component

The PDDLManager is the central place where the PDDL representation is built and maintained, so it needs to be aware of every item within the game that must be represented in PDDL. This is achieved using the PDDLObject component.

#### 4.2.3.2 The PDDLObject Component

Every GameObject within the Unity environment that is to be part of the PDDL system has a PDDLObject component attached to it. This is shown in Figure 4.5. The core elements exposed to the designer are the unique name for this object in the PDDL description, and its object type.

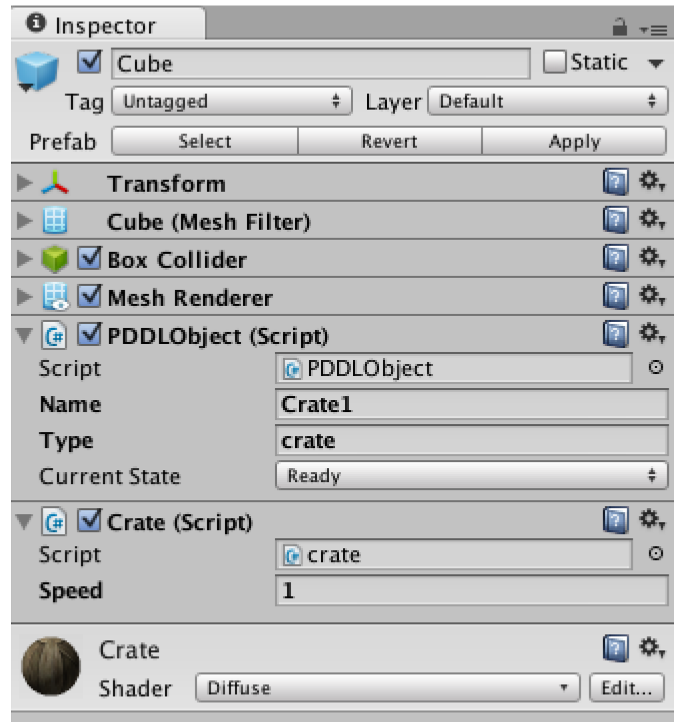


FIGURE 4.5: An example of a PDDLObject-enabled GameObject

When the game begins execution, the PDDLObject finds the instance of the PDDLManager using the Unity engine's tools for looking up a GameObject based on its tag. It then "registers" itself with the PDDLManager, passing its name, type and a reference to the parent GameObject that the PDDLObject is attached to. This is primarily the entire role of the component - to make its presence, and that of its GameObject, known to the PDDLManager.

#### 4.2.3.3 Automated PDDL Generation

For specific objects like NPCs, or those that have potential impact on gameplay, explicitly defining them in this manner is necessary. However, for a number of less explicit elements, it is quite a tedious process, most notably for locations within the world. Under a standard explicit representation using the PDDLManager/PDDLObject model,

locations would need to be defined as empty GameObjects, have the PDDLObject component attached and then have the other locations that are reachable from that one linked by hand.

A much better method is to devise a system for extracting these implicit details automatically from information available from the Unity engine. Specifically for Unity, this can be achieved using the NavMesh feature. A Navigation Mesh is a computed surface that represents the area that could be traversed by an agent with specific parameters such as width. The NavMesh is created from a set of polygons, with the vertices of these representing significant points where the walkable surface changes. In Figure 4.6 an example scenario is shown with the NavMesh highlighted in light blue. The walls in the scenario create a disruption that cannot be walked through, but also there is an amount of space given. Agents will not go closer to the walls than this, allowing them not to project their bodies into the wall (in the same way a human would not try to get their centre of mass closer than half a shoulders width to a wall). By observing this distance, the vertices can be clearly seen as the points at which the NavMesh changes, since its edges are straight line segments.

The NavMesh is used for pathfinding as the edges within the mesh describe ideal paths for traversing the space. From within the initial polygon a path can be created to a vertex and from that vertex, using the edges to any other polygon within the NavMesh. A detailed description of this process is outwith the scope of this work.

What is significant about the NavMesh is that it is calculating the location of what are considered to be important positions for navigation already in the vertices it marks. Based on analysis of the corresponding edges not only can PDDL objects for locations automatically be generated, but also the corresponding associations between links that

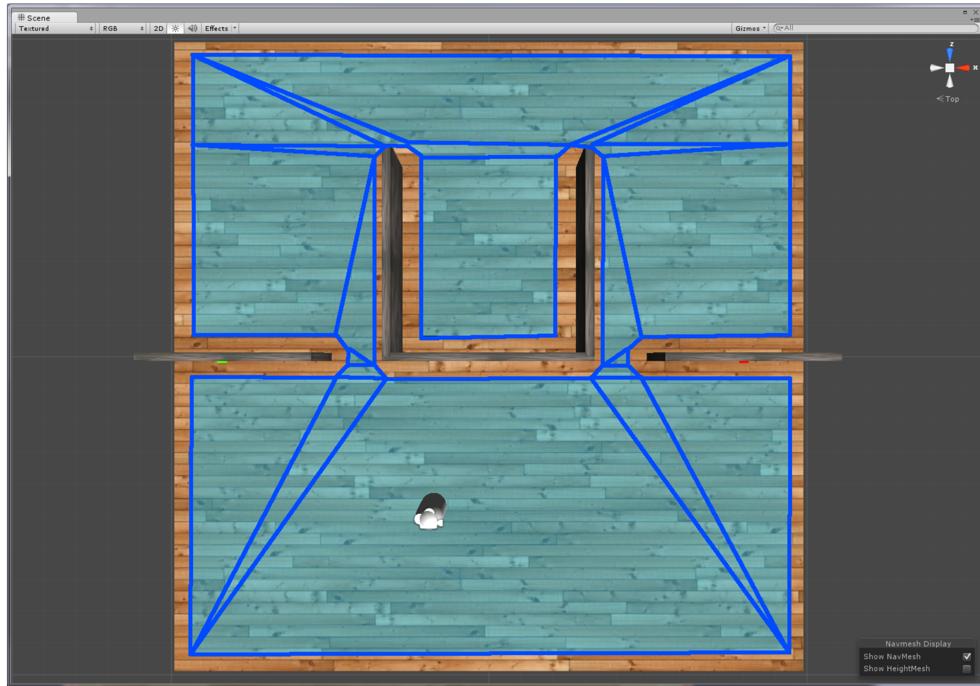


FIGURE 4.6: An Example of a Walkable NavMesh

are traversable can also be created, saving a significant amount of time for the game designer.

Figure 4.7 shows the Editor Extension created to aid with this process. Unfortunately, the vertices are not exposed directly and instead it is necessary to analyse the description of the polygons that comprise the NavMesh using Unity's *NavMesh.Triangulate()* function[4]. Duplicate vertices are eliminated by comparing relative positions of vertices. Those that appear grouped together are only counted once, since these are the vertices being duplicated as they form part of multiple polygons within the NavMesh.

With the set of unique vertices, it is possible then to perform a raycast analysis to determine which pairs of vertices can be traversed directly. Performing this from each vertex to each other vertex in the NavMesh allows for all pairs of walkable locations to be found, this can then be stored and represented directly in PDDL. The result of this analysis is shown in Figure 4.8.

[4]Since replaced by *NavMesh.CalculateTriangulation()*

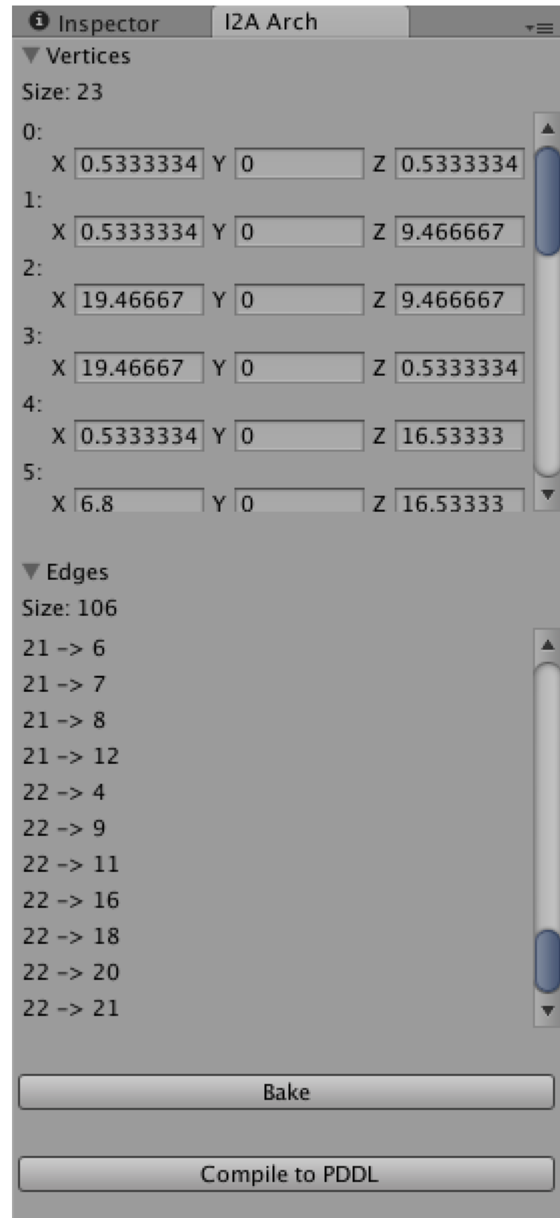


FIGURE 4.7: Tool for PDDL Generation from NavMesh

### 4.2.4 Executing Actions

The ability to take an action as chosen by the I2A executive and translate that into an action within the game world is clearly one of the major hurdles. Recall that edges within the Common Representation of the I2A are all representing actions, therefore there is a mapping of an edge within the CR into a PDDL action with associated parameters. Being able to dispatch a PDDL action inherently means that an I2A action can also be

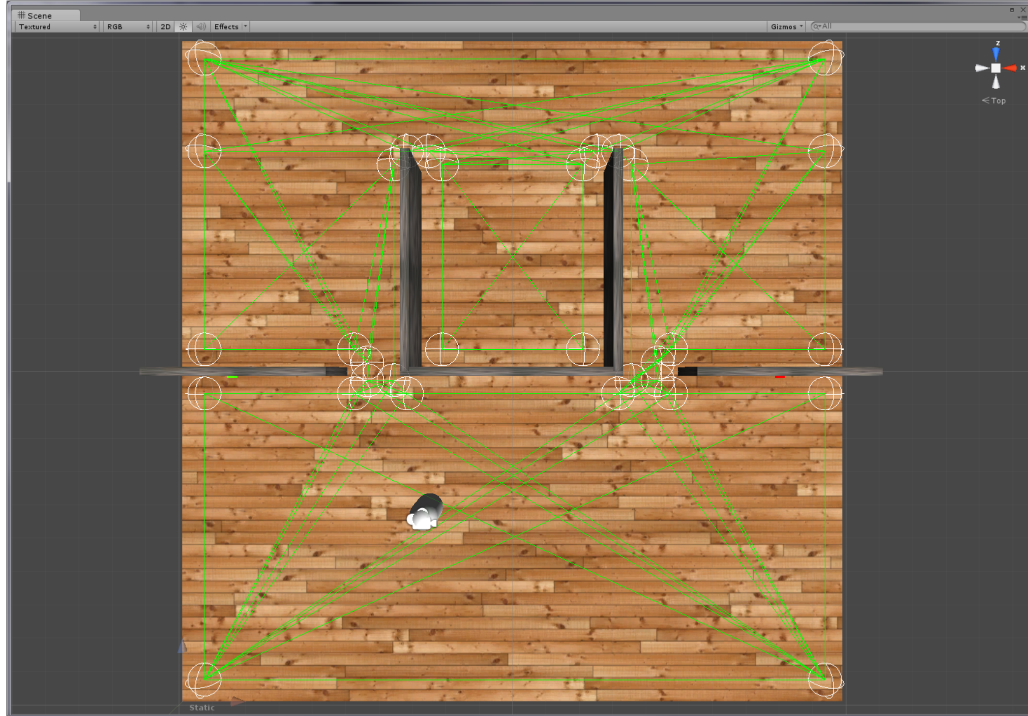

FIGURE 4.8: Results of NavMesh extraction as shown using debug tools ingame

dispatched and executed.

This relies on using the PDDLManager to interact with GameObjects that have registered themselves using the PDDLObject component. In Figure 4.9 six GameObjects have been registered with a reference to the specific GameObject, name and type.

In Figure 4.10 a plan consisting of four actions is set to be executed. This plan is stored as an array of strings, and takes a format slightly distinct but functionally similar to PDDL.

Because of the format of the action, it can easily be decomposed using a string tokeniser into its constituent parts, those being firstly the name of the action and subsequently the one or more relevant parameters. Note that there is no limit on the number of parameters that could be specified.

Each parameter is specified by the name of the PDDL object, with the registered objects list of the PDDLManager being used to translate this into a list of GameObjects. Then

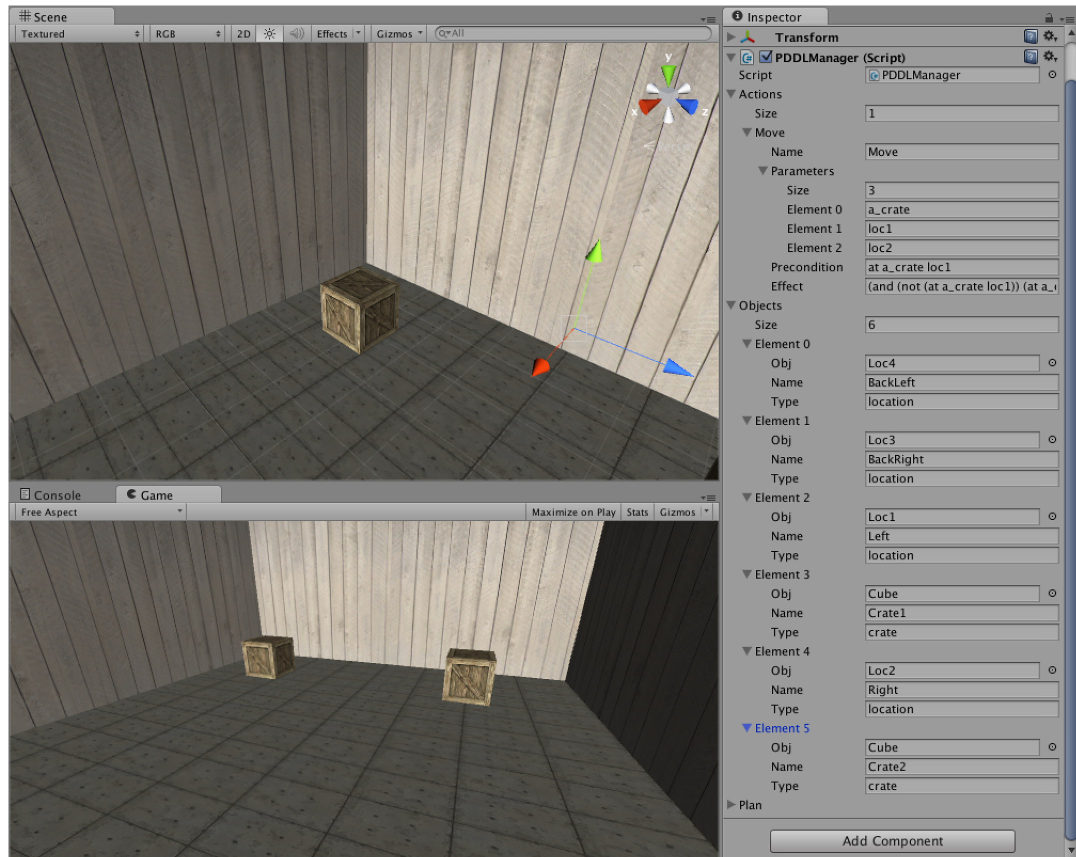


FIGURE 4.9: PDDLObjects registered with the PDDLManager

another property of the Unity engine is used in order to activate them, the *SendMessage()* function, which broadly allows for a function with a specific name to be triggered. This is called on a GameObject, but the function is actually invoked - or at least attempted to be invoked - within each component. In Figure 4.10 the first action reads MOVE CRATE1 LEFT BACKLEFT. When this is parsed, the GameObjects that are specified as having the PDDL names Crate1, Left and BackLeft are sent the Move command, and the objects involved are passed in as parameters for that command. Because Left and BackLeft are locations, although they receive this message to invoke their respective Move functions, they do not have these present. However, Crate1, whose inspector window was previously shown in Figure 4.5, also has another script attached to it as a component, the Crate MonoBehaviour, and this does provide an implementation of the Move, invoked as *Mov(GameObject[] params)*.

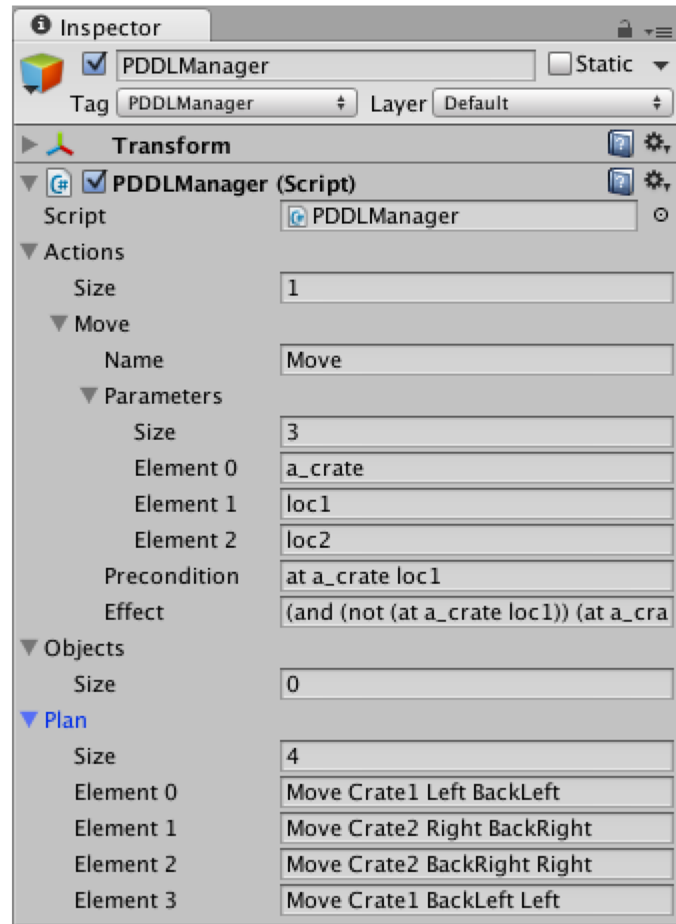


FIGURE 4.10: An example plan loaded into the PDDLManager

*SendMessage()* is a powerful way for the PDDLManager to call function implementations without having to be confident that these implementations actually exist, meaning that the PDDLManager can remain very compartmentalised and independent of the specifics of the domain being used.

### 4.2.5 Retrieving and Updating the Plan

It is very important for the I2A that it be able to update the plan as execution progresses and the nature of the world deviates from that which the planning system originally expected. Although the bulk of the I2A is designed to alleviate the reliance on this replanning, it does not remove it, so there is still a necessity to be able to update the plan held by the PDDLManager.

It was felt that implementing an entire planning system within Unity was outside the scope of the project, but it was still necessary to prove that such replanning was possible. Due to the nature of the replanning within the I2A being asynchronous, a good solution was felt to make use of network communications to interface with an external component. Unity provides a robust library within its engine for interacting with websites, which allows for a delay in processing time.

The JavaFF planning system[5] created by Coles, Coles, Fox and Long is a natural fit for development in this manner since it is an open source implementation of the FF planning system[40]. Although FF is no longer considered to be at the cutting edge of planning technology, it is still a strong workhorse, and because of the open nature of the JavaFF implementation, it was relatively simple to adapt the system to function as effectively a planning web server, taking a PDDL problem and domain as input strings, running the JavaFF system and capturing the output and presenting back as the result of the query that the Unity system instigated.

From the perspective of the PDDLManager, when a new plan is required, it can use the current PDDL description of the world to generate an HTTP request to the Network-JavaFF server running locally. This is a non-blocking function that allows other aspects of the Unity engine to continue execution while the request is served. This fulfills the requirement that the I2A be able to perform its replanning asynchronously.

The resulting text is then parsed into a format compatible with the PDDLManager, a small amount of plan repair is undertaken to compensate for the actions that have been taken while the plan was being generated, and the new plan is then imported into the PDDLManager ready to be incorporated into the I2A logic.

[5] http://www.inf.kcl.ac.uk/staff/andrew/JavaFF/

### 4.2.6 Execution Monitoring

The ability to detect the state of execution and ensure that the internal representation of the world is only updated as actions complete is also important, as is detecting when actions fail and ensuring that an appropriate recovery is attempted.

The primary way in which this is ensured is that each action being executed is called from within what Unity calls a coroutine, which is best thought of as (although technically very dissimilar to) a multi-threaded task. A single coroutine starts its execution and at a specific point during that execution uses the YIELD statement to pause its process. This can then allow other activities to continue, and at each pass through the main game loop, Unity will start processing this coroutine again until it sees the next YIELD and so on. This means that when an action is triggered, it can be monitored during each frame of the game execution, using YIELD to ensure the action can check if it has completed and if not, make the changes necessary for a single frame of the action. In the next frame, this action can be resumed, and when it has completed successfully, this can be alerted to the PDDLManager.

Detecting and compensating for an action's failure is a more complicated task, with some reasons for the failure being dependent on the specific game, whilst others are likely to be more generic and broadly applicable. An action might fail to execute because its preconditions are no longer satisfied, which would be easily detectable as that action came to be executed. Equally, it might fail because of the interaction of some other elements during the execution process. For example consider an NPC walking through an open door. If the door is closed after the NPC's action has started but before it has actually stepped through, then the NPC may find itself walking against the closed door. In this case, it is necessary to have some more robust form of fault detection.

Using the coroutine process it would be possible to build a simple timeout into the action execution that if the action does not complete within a certain time. A better approach would be to implement fault detection as part of the action execution itself. For example in the case of the NPC walking through the doorway it may be that any collision that NPC suffers would be considered a fault. In either case, this can be signaled to the coroutine which can interrupt itself, and use the same *SendMessage()* function to trigger error correction systems on the relevant objects.

In this way, the execution of the system can be monitored, and faults can be detected. Although this will always be very implementation specific, it is possible to use the tools that the Unity engine provides to ensure that the I2A is informed as to the outcome of the actions it has instigated.

## 4.3 Summary

This chapter has addressed two of the evaluation criteria laid out in Section 1.3, namely the functionality and viability of the I2A. Its functionality was demonstrated by an experimental analysis showing that the methodology of the I2A is sound, and discussing its performance under certain conditions. A comparison to the contemporary planning system LAMA was drawn to demonstrate the effectiveness of the clustering techniques that the I2A uses and finally it was shown that these techniques apply in a range of common planning domains and are not just specific to the Logistics+ domain described. These are important results since as a novel technique, the I2A is unproven and its ability to reason and act in game worlds must be shown through experimentation. The experiments performed give a clear indication that the I2A methodology is sound and is applicable in domains that are likely to be analogous to those that describe game worlds.

In the second half of the chapter, the viability of the I2A was established by demonstrating that the I2A could be implemented insde the popular Unity game engine. Four crucial obstacles to this were highlighted and solutions described that would enable the I2A to function within Unity, specifically with reference to how the I2A could be connected to the game world and the way that instructions could be coordinated within that world. This proves that the technique can be viable in an industry-facing context, making it a valid approach to solving the problem of decision-making for agents in virtual worlds.

# Chapter 5

# Evaluation - Robustness and Efficiency

The prior chapter demonstrated the feasibility of the I2A both in terms of its functionality as well as its viability for use in an industry context. However two key criteria for evaluation remain, namely demonstrating that the I2A is sufficiently robust to deal with dynamic environments and potential threats to its execution and that it is efficient enough to outperform traditional deliberative solutions. In this chapter these will be addressed by firstly showcasing an example problem, from a recent commercial game, and demonstrating that the I2A would be well suited to empowering NPC agents within this game. Secondly the question of efficiency will be addressed to demonstrate the computation time of the I2A and contrast this with contemporary alternatives.

## 5.1 Worked Example

In order to effectively showcase the potential of the I2A, this section will explore an example based on a real-world game released by a major game studio. The game will be described at a high level before being explored in more detail, with specific reference to the manner in which the components of the I2A operate and interconnect. This will then be summarised, and comparison will be drawn to alternative systems.

### 5.1.1 An Example From Industry

A great example of the kind of problem that the I2A is uniquely suited to can be seen in Mountain Goat Mountain (Zynga, 2015). Zynga released this game in the third quarter of 2015 to a strong critical[95][96] and consumer reception, garnering a 4.5 star (out of 5) rating on both the Google Play Store[97] and Apple's iOS App Store[98]. Mountain Goat Mountain was developed using the Unity framework which has been previously introduced, therefore what follows builds in many ways on the case study presented previously in Section 4.2.

The aim of Mountain Goat Mountain is to get your character (the eponymous Mountain Goat) as high as possible up a mountain, by moving forwards, backwards, left and right. The goat can jump up a single step at a time. The goat needs grass to survive, and this is placed sporadically around the mountain, meaning that the goat must continuously keep moving. The goat will die if it makes a mis-step and falls off the mountain. Along the climb, there are coins that can be collected that allow the player to unlock new types of goats (changing only the aesthetic, not the mechanics) as well as the "Super Goat" power-up that allows the goat to jump more than a single step up the mountain in one

move. Additionally as the goat climbs, it will encounter obstacles such as rolling logs and storm clouds either of which can kill the goat and end the game.

This problem lends itself well to exemplifying the strengths of the I2A because there are clearly elements of both deliberation and reaction inside the game. There is an element of planning involved in determining the goat's path up the mountain. There are also a range of elements whose presence or absence will be determined during execution, meaning that they cannot be reasoned about as part of the planning operation. These have varying utility and worth - a single coin has significantly less value than the Super Goat, and the latter may be sufficiently valuable to warrant risking an end to the game by crossing the path of one of the logs.

It should be noted that in the "real" version of Mountain Goat Mountain (MGM), the level is created procedurally. The implications of this will be discussed in detail below in Section 6.2.2, however for the purposes of this example, we will assume that the level is crafted by hand during development.

#### 5.1.1.1 Movement in MGM

As mentioned above, the goat is able to move forwards, backwards left and right (although because of the angle the camera is placed at, this appears to be more of a diagonal movement). The mountain is made from discrete building blocks, meaning that there is a uniformity to its creation that from the top down makes its construction resemble a tile-based map like that discussed in relation to the A* algorithm in Section 2.1.2.1, albeit with the addition that each tile also has an associated height. The mountain as shown ingame is visible in Figure 5.1 and the tile equivalent, viewed from the top down, is shown in Figure 5.4.

There are several rules that govern allowed movement, summarised as follows:

- Any adjacent tile that has the same height, or one higher is connected for normal movement (jumping). These connections are shown in Figure 5.2.
- Any adjacent tile that has a lower height is connected for normal movement (falling). These connections are shown in Figure 5.4.
- A tree occupying a tile makes it impassable for movement.
- The “Super Goat” is able to traverse in a straight line from a tile to the next walkable tile in any direction, regardless of height and intervening terrain. This type of movement is shown in Figure 5.3.
- Springboards provide a temporary effect similar to the Super Goat in that they launch the goat to the next walkable tile. Springboards are valid for a single direction only and when the goat lands on them will automatically launch it in that direction.

#### 5.1.1.2 Hazards and Items in MGM

One of the key elements in Mountain Goat Mountain is that the environment is exceptionally dynamic. There are a range of hazards to avoid and items to collect which appear within the world at (from the player’s perspective at least) random points within the level. The nature of these is listed below:

- Logs - These are a type of hazard that move in a single direction down the mountain. They will continue moving in that direction until they reach the end of their

Figure 5.1: The Mountain Goat Mountain world, with tile locations picked out in purple.

life, come to a missing tile or have to climb more than a single level in one step, in which case they will expire.

- Boulders - The Boulder follows more or less the same rules as the Log, with the exception that it will occasionally change the direction of movement down the mountain. This is unpredictable, and therefore makes the Boulder a more significant threat.

- Storms - Storms sit over specific tiles and often occur at bottlenecks in the mountain. A Storm is effectively a timed gate, causing a lightning strike at a regular interval. If the Goat is struck by lightning, it dies.

Figure 5.2: Uphill connections between tiles in MGM.

- Crumble Tiles - Although most tiles on the mountain have a level of safety associated with them, Crumble Tiles do not - a short time after first standing on a Crumble Tile, it will disintegrate, and if the Goat is still standing on it, it will die as a result.

- Grass - The Goat needs grass to survive. Over time the Goat's "grass meter" will empty, and it must be topped up regularly from the grass tiles which are dispersed intermittently throughout the mountain. Grass is not something to be pursued in itself, but its presence should decide which of two similar routes up the mountain the Goat will take.

- Coins - Appearing intermittently, but potentially on any traversable tile, Coins are the primary currency of MGM. When the player has collected 100 Coins, they

Figure 5.3: Additional traversable connections when using the "Super Goat" ability.

are able to purchase cosmetic items to change the look of the game (but this does not affect the gameplay). Their positioning may influence a player's path up the mountain, but their individual utility to a player is relatively low.

- Chests - Chests hold a group of coins, with the number being randomly assigned from as little as 5 to as high as 100. As such, they are more useful to the player but typically positioned in such a way that makes them very difficult to get to.
- Super Goat Spring - In order to turn into the Super Goat, the Goat must collect the Super Goat Spring. After a short period as the Super Goat it will return to its normal mode.

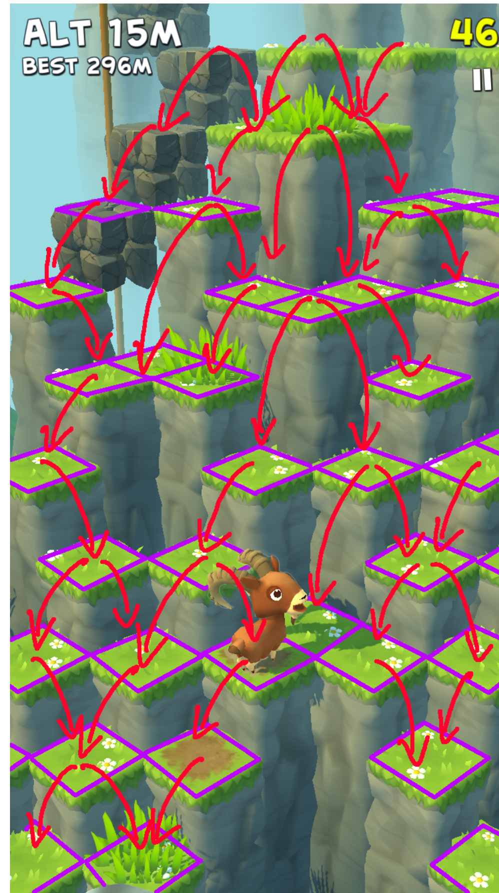


FIGURE 5.4: Downhill connections between tiles

### 5.1.2 Algorithmic Example

As was previously discussed in Section 3.2.1, there are a number of components to the I2A. Figure 5.5 revisits these and the broad flow of the I2A process. This section will present a detailed look at the way the components interact, and the data that is passed between them.

#### 5.1.2.1 During Game Development

During the game's development, a number of things must be established in the Level Layout, and the Common Representation and Initial Plan must be calculated. An overview of the data flow for this is presented in Figure 5.8.

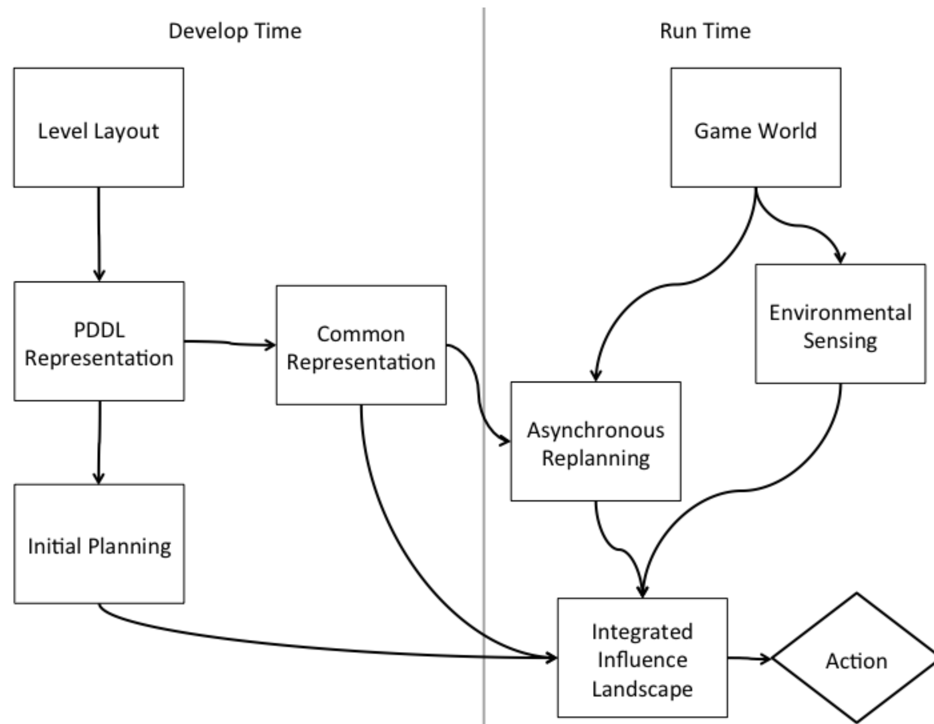


FIGURE 5.5: Recap of the Integrated Influence Architecture Components

### Level Layout

A Mountain Goat Mountain level is comprised of tiles positioned at varying heights in order to present a challenging set of paths to the player. When generating the PDDL Representation in the next phase, one of the things that is paramount is knowing the height and location of every tile and whether it is traversable by the Goat. One of the easiest ways to do this is to ensure that each tile makes use of the Unity tagging system referenced in Section 4.2.3.1, and adding a component to the tile prefab object (the concept of prefabs was introduced in Section 4.2.1.1) to allow the designer to select whether it is walkable or not.

### PDDL Representation

With each tile tagged, it is easy to retrieve a list of all the tiles in the level, and iterate through that list. For each tile, the system must take its X,Y and Z coordinates inside Unity and using the standard size of a tile, create a grid representation of height, which

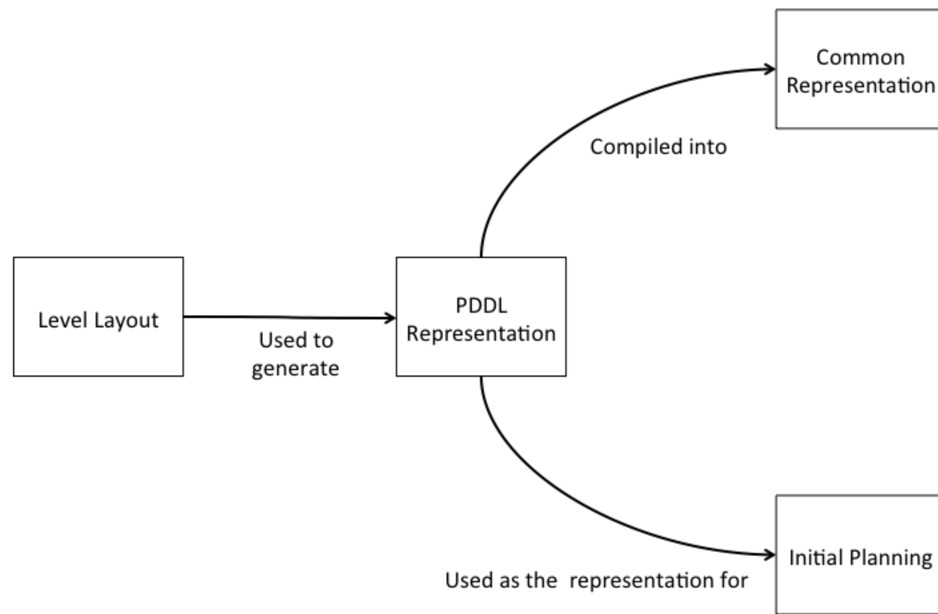


FIGURE 5.6: Data Flow of the Develop-time Components of the I2A

is best held as a two dimensional integer array. An example of this, derived from Figure 5.1 is shown in Figure 5.7. Using a representation like this, and the movement rules presented above, it is possible to then step through the grid and assess what tiles serve as origins for which destination tiles. This process must be done for each tile, in all four potential directions of movement and all three movement types (jumping, falling and Super Goat jumps). For each, if a valid movement type exists, it can be recorded as a PDDL fact in the form CONNECTED(ORIGIN, DESTINATION). For connections that require the Super Goat ability, these can be recorded as SUPER-CONNECTED(ORIGIN, DESTINATION). This will give a description of the world that the Goat is operating in.

A PDDL definition of the actions that the Goat can take is also a requirement. The MGM world is relatively simple, and the Goat is only able to take one of four actions at any time, however for the sake of the representation, the standard movement and Super Goat movement would be modeled as separate actions, the former have a precondition

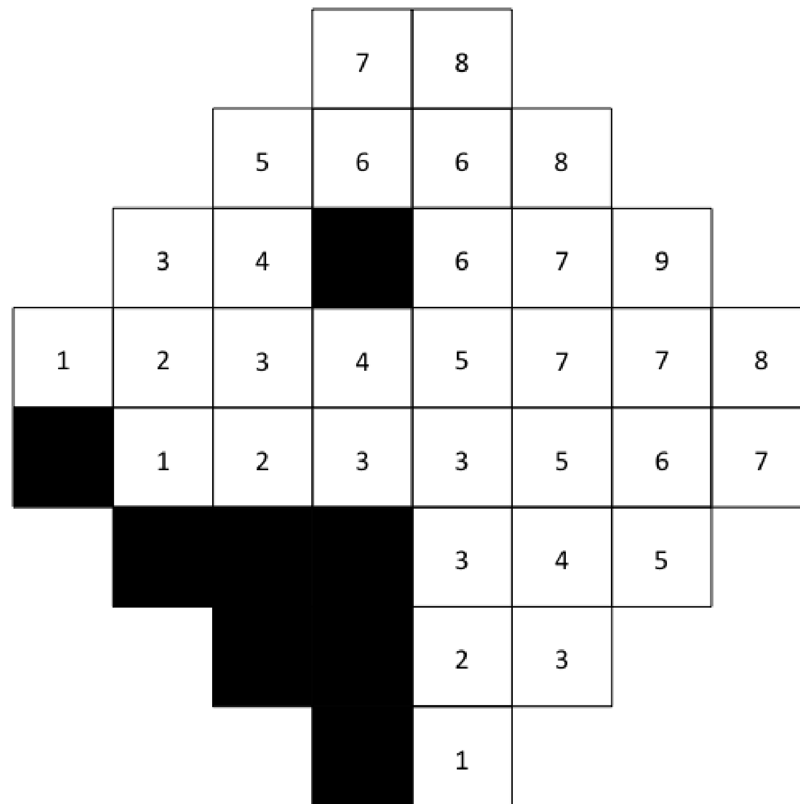


Figure 5.7: The MGM world shown from a top-down perspective, with height annotations for each tile.

that the Goat not be in Super Goat mode, and the latter having one that it is. It is also necessary to model the collection of the Super Goat Spring as that provides the mechanism by which the Goat will change modes, as well as an action to cause the Goat to revert to its normal mode when the Super Goat ability expires

Finally in order to satisfy the requirements of the PDDL formalism, there must be a goal presented. The overall goal in MGM is to climb as high as possible. Because in the real game the mountain is procedurally generated, this could be arbitrarily high, but for the sake of this example it is assumed that there is an ending height, represented by a set of tiles that, when the Goat lands on them will trigger the win condition. Because of this, the win condition can be set as the goal of the problem, and will causatively imply that reaching one of these highest tiles is the way to reach that goal to the planner. Note that

there are alternatives to this model, such as using a disjunctive set of conditions that the Goat is at one of this set of tiles, or even drawing on the use of fluents and metric planning (introduced in Section 2.1.1.1) to track the height as a number and attempt to maximise it. These alternatives introduce their own challenges however, and will not be explored here.

**Common Representation**

Now that there is a complete PDDL problem definition, it is possible to hand this off to the Common Representation component which will compute the CR. Firstly, the PDDL is reformulated into the SAS+ formalism using Helmert's translator[35] which operates primarily by mutual-exclusion analysis to determine those facts that can be grouped together. Here the output would be three DTGs, one for the location of the Goat on the mountain, another for the current Super Goat status and finally a third that captures the current value for the win condition.

The next step in the process is to apply the Cartesian Product operation to combine these three DTGs. This process will increase the number of states dramatically, but also the number of edges, and many of these states that were present in a DTG now can be eliminated as their preconditions are not satisfied when the DTGs are combined. Consider that any transition that requires the Super Goat ability will be present in the DTG, however after the Cartesian Product process is undertaken, that edge will appear twice in the Common Representation. Likewise, the edge that activates the Super Goat ability will appear multiple times, even in situations where this would not be a valid transition because the Super Goat Spring is not present. This means that there needs to be a clean-up pass to remove edges representing actions whose preconditions are not

satisfied, and then to remove those states that are now unreachable. The resulting graph structure is the foundation of the Common Representation.

With this graph structure now in place, the clustering algorithm can be used to identify the Focal Nodes and Super Nodes within the graph using the process described in Section 3.4. The Common Representation is then stored in both the full and abstract versions with a mapping between the two.

**Initial Planning**

Because the Initial Plan is created during development, when efficiency and computational time is not a concern, the full PDDL representation is used in order to create a complete listing of the actions that must be taken in order to optimally achieve the goal. This is done using a standard planning system built to operate with PDDL - the choice here is somewhat flexible since this is primarily a black box operation, and the planner in use here will be deployed onto development machines, meaning that there are no usability concerns around ensuring compatibility and availability of specific language runtimes. LAMA, Fast Downwards or JavaFF would all be suitable choices as they will all be able to adequately output the plan in an appropriate format.

#### 5.1.2.2 During Runtime

During the game's execution the I2A will need to make a choice as to what action should be taken next. This will be determined by observing the world in which the agent is acting and taking into account the information being presented by the deliberative reasoning component in order to evaluate possible action choices in the context of both types of reasoning, iteratively updating the Integrated Influence Landscape. A visual representation of this process is presented in Figure 5.8. Note that the components

represented by dashed ellipses are from the develop-time operation of the I2A. This process will be described by component below.

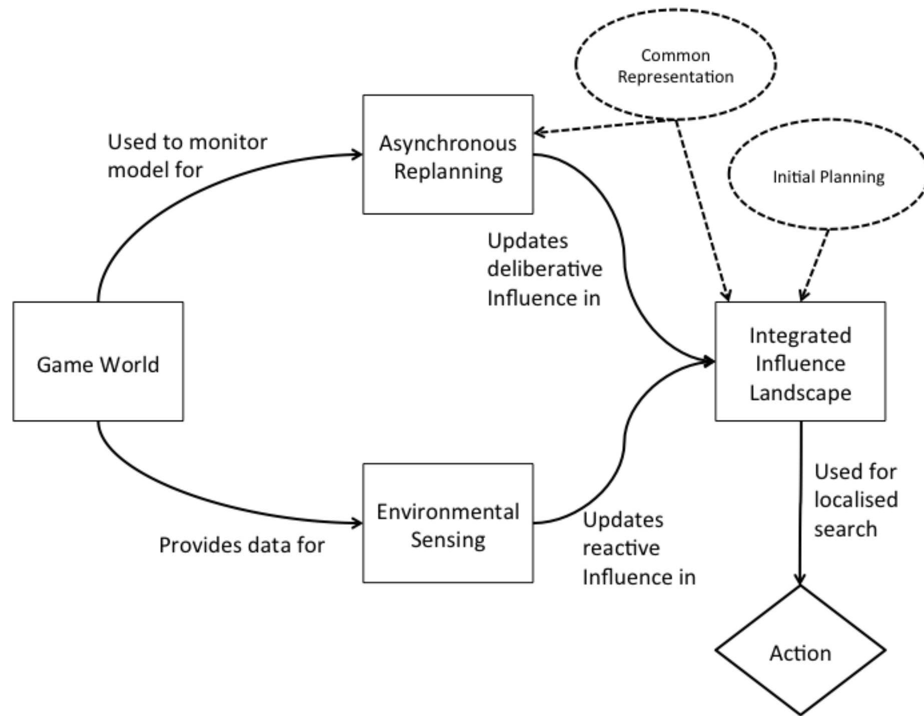


FIGURE 5.8: Data Flow of the Runtime Components of the I2A

**Creating the Initial Integrated Influence Landscape**

The Integrated Influence Landscape is built by taking the Common Representation as a core structure and then layering in the different sources of Influence. In its initial form, the IIL is constructed purely based on the Initial Plan - as the level is being loaded there is not yet an environment from which to draw additional data. As such, this IIL is created simply by taking the plan created during Initial Planning and determining which of the Focal Nodes within the Common Representation that plan would pass through. For computational simplicity, this process can be carried out on the non-abstract version of the CR rather than attempt to translate actions in the plan into their abstracted counterparts. This will identify the Active Focal Nodes. A small amount of influence is added to each of these and to any nodes that satisfy the goal of the problem, which for

this example would be any node for which the win condition is true. For each of these, the influence is propagated across the CR's graph structure.

Given that this initial seeding can be done on the full CR, it is important to also use the method described in Section 3.7.1.3 to also give a valid Influence value to each of the Super Node clusters stored in the abstract representation.

**Game World**

During execution, the Game World will need to expose a number of aspects to the I2A architecture, primarily to validate that the I2A is reflecting the current state of the game. The location of the Goat and whether it is in Super Goat mode are the main things that needs to be monitored to ensure that the model the I2A is working from is kept current. Since the map of the level is available to the I2A in advance, there are few scenarios that should cause the model to deviate from the reality, perhaps the primary one being an attempt to utilise a Super Goat path and the Super Goat ability expiring before it is possible to act on this. Effectively, this is an execution monitoring system to provide insurance and fault tolerance.

**Environmental Sensing**

The I2A system needs to be aware of potential hazards and desirable objects in its vicinity, and in order to achieve that it needs to be able to detect the current state of objects within the Game World. To do this, a new component is added to the Goat during development that will effectively act like a radar system, detecting all objects within a certain radius from the Goat. This is somewhat analogous to a radar system, detecting the obstacles and their location. Unity has built-in functionality to achieve this in the Physics.OverlapSphere function, which returns all GameObjects that have active Collider components touching or within the sphere provided as an argument

to the function. It is trivial to then iterate through this list of GameObjects to select only those relevant to the I2A system, which can then be passed to the IIL component in order to update the influence that the game world is generating.

#### Asynchronous Replanning

The high level view of the current game state as exposed through the Game World component is used by the Asynchronous Replanning system in order to detect when the Goat has deviated from the expected execution of the existing plan. At this point, a planning operation is instigated in order to determine if a new, more optimal, plan exists. This is run as a low priority background process on the abstracted Common Representation. As the planning is being performed in the abstract, it will have limited complexity, and as a background process it is non-blocking, which means that the I2A is able to continue executing to the best of its ability in the intervening time between the initial need for a replan operation being detected and the result of the replan being available.

#### Updating the IIL

As new information is provided to the IIL component from the Environmental Sensing and Asynchronous Replanning systems, the IIL is updated as necessary. This involves recalculating the propagation of influence based on this new information, with the new version of the IIL replacing the old. In the case of Environmental Sensing, this means taking the presence of hazards and items into consideration and applying a reflective amount of influence to match their perceived utility. In this way, coins would only create a small amount of positive influence, while a coin chest would produce significantly more. In Section 3.7.1 it was observed that the propagation techniques could be implementation-specific and the log hazard in Mountain Goat Mountain is a compelling

example of the necessity for this as it moves only in a single direction - the negative influence that it generates should not radiate along the path it has already traversed as it is not a threat in those states, and again the I2A is designed in such a way that this can be captured and represented as part of the algorithms within the system.

For an update to the Asynchronous Replanning information, again the Active Focal Nodes are determined and propagation is carried out as described in Section 3.7.1.3.

**Choosing an Action**

The final phase in choosing an action is to use the IIL, updated with the most recent information available at this point in the game's execution, to determine an action to take. In Section 3.10.1.1 the Localised Expansion-Bound A* Search was introduced and is a viable option in the context of Mountain Goat Mountain. This allows the I2A to do a limited look-ahead to select a promising path by which to climb the mountain. The look-ahead aspect prevents the agent from getting stuck in localised dead-ends on the mountain by ensuring not only that the immediate choice is good, but that the path it leads to is also a good option when considering all kinds of influence.

### 5.1.3 Summary

The above described how each of the components outlined in Section 3.2.1 contributes to the functioning of the I2A as a system capable of acting based on a hybrid of deliberative and reactive data. Mountain Goat Mountain is in a lot of ways a thorough encapsulation of the kind of problem that the I2A is designed for in that it combines long-term planning in the route and method of traversing the Mountain, along with highly dynamic elements such as the the Boulders, Logs and Coins that require a more reactive approach as they enter the Goat's awareness.

When contrasted with alternative techniques, it is clear to see the strengths of the I2A. Consider a technique such as relying entirely on A* search. Clearly such an approach would be quite strong in its ability to plot a route up the Mountain that was optimal, but it would not be able to cope with the dynamic elements of the game, and as such would be a very high risk approach that would not be expected to do well in the average case. The same argument could be made for a more robust Deliberative approach such as planning, which would require the majority of the design overhead of the I2A in order to create a PDDL representation on which planning could be undertaken, but would not have the capability to react to the dynamic elements either.

A more Reactive based approach, such as one utilising the principles of Subsumption[13], in which the default behaviour is to climb the mountain as best as possible with higher priority behaviours taking precedence when necessary in order to avoid the hazards and take advantage of items as they appear. This approach would have a higher survivability since in an ideal scenario it would be able to avoid danger, but this avoidance and the naive approach to pathing would mean that there would be a significant trade-off against the amount of progress the Goat was able to make up the Mountain.

Finally, with reference to prior hybrid systems, the Three Layer Architecture[84] introduced in Section 2.3.2 is a prime example of such a system that does not solve the core issues underlying either approach. In the TLA, what might be expected would be for both a planning system and something like a Subsumption system to cohabitate, with the third layer arbitrating between which one should be active at a given moment. However, without knowledge of the existing plan to guide it, the Subsumption system will make potentially sub-optimal choices in terms of the long term objective. Equally, having had its execution disrupted, the planning system would then have to perform a

replan operation, which as has been discussed is computationally expensive. Replanning will be triggered each time the agent needs to divert to overcome an obstacle, which means in the context of MGM many many times in the course of one game, which is clearly unwieldy and inefficient.

This example demonstrates not only how the I2A operates, but also provides a context for why it does so and highlights how it overcomes the challenges that other techniques face when attempting to operate within this environment.

## 5.2 Complexity Analysis

When creating a new algorithm, it is important to consider not just the end results of the process but also the time it takes to run - as has been previously discussed, one of the critical aspects that differentiates the games domain is the very strict computational constraints. An obvious method for comparing the I2A to other algorithms would be by doing a simple comparison of execution times for specific problems but this overlooks that the I2A pushes much of its computation into develop-time, and that an algorithm's implementation will have dramatic impact on its execution. As an example, consider that a specific implementation of an algorithm may use file storage in order to move information between its components - this is not a requirement of the process and is significantly slower than other methods might be, but for whatever reason was thought to be an advantageous approach. In the context of execution time though, file storage is a ponderous medium by which to exchange information and as such, the execution time would be artificially higher due to implementation details that are not a part of the algorithmic specification.

### 5.2.1 Introduction to Computational Complexity

Instead of assessing execution time, we utilise methods from theoretical computer science in order to compare algorithms in an implementation-agnostic manner. Instead of assessing execution time, the emphasis shifts to evaluating the number of steps the algorithm will take in order to reach a conclusion. Consider the following trivial example: In algorithm 1, an input value is given and it is added to itself 4 times. In algorithm 2, an input value is given and it is multiplied by 4. Intuitively, these will produce the same result, but algorithm 1 will do it after 4 instructions (the repeated addition) whilst algorithm 2 will do so after just 1, making it more efficient[1]. Crucially this assessment is completely independent of context concerns such as parity in the hardware executing the algorithms, language that the algorithms are implemented in and so forth.

In these sorts of problems, it is typical to consider the efficiency of an algorithm in relation to the size of the input that it is passed. Computational Complexity typically aims to answer the question of "what happens to the number of steps in the algorithm as this number gets larger and tends towards infinity?" or put another way, "what is the worst possible case?". This is the premise of "Big-O" notation, an accepted way of representing the computational complexity of an algorithm. The Big-O notation not only assumes that the input size (typically denoted as $n$) tends towards infinity but that for a sufficiently large $n$, the complexity can be adequately approximated by taking the "order of the function" as an upper bound on its expected growth rate. As an example, consider an algorithm with a nested For loop; the outer loop runs $n$ times, and each time performs $n$ operations meaning that the nested loop itself costs $n^2$. There is a subsequent process that requires stepping through the entire input again, and there is

[1]It is worth pointing out that many modern compilers will perform optimisations on trivial problems such as this, recognising the repeated addition and replacing it with the single multiplication step.

some overhead and additional steps that are independent of the size of the input which we can represent as a constant value $C$. It would be accurate to say that this algorithm will execute in $n^2 + n + C$ steps, but for the sake of simplicity and comparison it is important to recall that computational complexity is interested in the general case as $n$ tends towards infinity. As this occurs, higher order terms within the function become increasingly dominant, so it is fair to say that for a sufficiently large $n$, $n^2$ will be a dominating factor, and indeed this is what the Big-O notation captures, extracting the dominant term. In this case, the computational complexity of the algorithm would be represented as $\mathrm{O}(n^2)$.

### 5.2.2 Complexity Analysis of the I2A System

As a novel approach to the problem of agent execution in dynamic environments, it is important to consider the computational complexity of the I2A in order to ensure that it is satisfying its primary aim - to be a better approach to AI in a game setting. However, with that said, one of the core concepts underpinning the I2A is that during the development of a game, there is an abundance of computational power available. As such, it is important to be mindful that the complexity of those components that are part of the develop-time process is much less significant than those that will ship as part of the game itself, and these components will be treated separately below.

#### 5.2.2.1 Complexity During Development

As described previously, during development several different components are used in order to perform the necessary calculations to provide the runtime components with the

needed data structures and seed information. Below each one of these will be assessed to determine its computational complexity.

#### Level Layout

In some regards, the Level Layout process is likely to be the most time consuming since it is primarily a manual process. Ensuring that each tile is positioned appropriately and has the correct information cannot be an not automated process, and therefore does not have an associated computational complexity.

#### PDDL Representation

In order to build the PDDL Representation of the level, a list of all the tiles is needed as are the maximum width and depth that tiles can be positioned at. Finally the standard width, depth and height of a tile are required. All of these should be easily accessible from the level. By iterating across each tile, for a given $n$ number of tiles, it is possible to build the 2 dimensional height map as was shown in Figure 5.7. Since this involves dividing each of the components of the 3 dimensional position of every tile within the Unity engine by respectively the tile's width depth and height, this is an $O(n)$ operation.

To determine the predicates that will form the PDDL representation it is now necessary to establish for each of the three movement types whether it is applicable and what the resulting destination tile is, using the height map as a lookup device. This will again be an $O(n)$ operation since there are a fixed number of lookups required.

#### Common Representation

In order to build the Common Representation there are three steps required. The first is to reformulate the PDDL problem statement into SAS+, the second to perform the

Cartesian Product operation and finally the resulting state space is clustered. Each of these will be considered separately.

Translating a PDDL problem into the SAS+ formalism is undertaken using Helmert's TRANSLATE system[35]. Although Helmert does not present a complexity analysis for this system, the detailed description of the process does give a level of insight. The process broadly consists of several steps. In the first PDDL object types are compiled out, reduced to explicit predicates in the definition. Intuitively this is likely to be a linear complexity of $O(m)$ where $m$ is the number of objects in the PDDL problem. The Invariant Synthesis step considers which PDDL predicates that are true in the initial state remain true throughout the entire state space, again intuitively this is likely approximated as a linear complexity in terms of the number of initial predicates $p$ and number of states $s$ since each one must be checked in every state, so is $O(p * s)$. Every predicate is now grounded, which means being repeated with every possible combination of objects put in place of its variables. For the Mountain Goat Mountain example, the predicates predominantly take two arguments as they determine the location of an object, or that two locations are connected, so for every predicate $p$ there will need to be $m$ versions of the first object and $m$ versions of the second object, so the complexity here can be approximated as $O(p * m^2)$. The final step in translation is to build the Multi-Valued Variable representation based on the prior analysis of invariants. This appears to be a relatively simple operation with complexity $O(i)$ with $i$ representing the number of invariants detected previously. As a whole process then, this stage can be represented as $O(m^2)$ with the bounding factor being the number of objects in the PDDL problem.

The translation process will generate a number of DTGs $d$ with each DTG having $n_d$ nodes. The Cartesian Product process will take each DTG and combine them into a

single process. If the average number of nodes in a single DTG is captured as $n_{avg}$ then the number of operations that will be required will be $(n_{avg})^d$. As noted previously there is a subsequent pass ensuring that the preconditions for each edge are met, removing the edge if not. The Cartesian Product process will have introduced an additional $d$ edges into the state space for each node, which means that validating these edges will be a process based $d * (n_{avg})^d$ iterations, or $O((n_{avg})^{d+1})$.

The final step required in computing the Common Representation involves iteration of the clustering algorithm described in Section 3.4.2. On the face of it, there is an issue with regards to computational complexity here in as much as this clustering algorithm does not provably terminate - it is not hard to envisage a situation where the cluster center swaps between two points repeatedly. However here the discretisation of the state space works (as described in Section 3.4.2.1) to the algorithm's advantage in that migrating a cluster center from one state to another is a step function rather than continuous. This should drastically alleviate the circumstances under which such a scenario can happen, if not eliminate it completely. Due to the discrete nature of the space, the only way such a scenario could occur would be if the space that the cluster occupied was perfectly symmetrical and balanced against every other cluster, with the "true" center of the cluster sat perfectly equidistant between two states, creating a situation not unlike that described by the Buridan's Ass paradox[99]. The circumstances needed to create this appear vanishingly unlikely. Based on this, it holds that the clustering algorithm will determine a result within some $k$ iterations, and if the above justification does not hold true in specific cases it would be possible to modify the algorithm to resolve after $k$ iterations and return its best result, which would also overcome the challenge that this lack of provable termination introduces.

On the assumption that one way or the other, the algorithm will terminate after $k$

iterations, the actual operation of the clustering should be considered next. In this it will be assumed that there are $n$ nodes in the Common Representation, and that $c$ clusters are being found. By Equation 3.3 it can be shown that updating the weights with which each node belongs to each cluster will be $n * c$ operations, and each one requires an evaluation of $c$ distances to get a proportional measure of the distance from this node to the current cluster centers, so this process will require $n * c^2$ operations. In order to update the cluster centers, Equation 3.2 indicates that for each cluster it is necessary to find the node which minimises the distance from itself to every other node. This means that there are $c$ operations in which $n$ nodes are compared against $n$ nodes, so this process will require $c * n^2$ operations. Finally, by Equation 3.4 $n$ and $c$ can be equated, since in this it is stated that $c$ is the ceiling of the square root of $n$, meaning that $c$ is approximated by $n^{0.5}$ This means that the abstraction of the Common Representation can be done in $k(n * c^2 + c * n^2)$ which is equivalent to $k(n^2 + n^{2.5})$. Consequently the complexity of this process is $O(n^{2.5})$.

As a whole then, this phase of the develop-time process is likely to be approximated in complexity by the Cartesian Product operation as $d$ (the number of Domain Transition Graphs) will be more than 2.5 - in fact in Mountain Goat Mountain it will be 3, representing the location of the Goat, its Super Goat status and whether it has successfully achieved the win condition. Consequently, this complexity of this phase will be $O(n^3)$ for $n$ nodes average per DTG.

**Initial Planning**

In Helmert's "Understanding Planning Tasks"[35] a comprehensive study of many contemporary planning domains is undertaken in order to establish their complexity class. Helmert makes a particular point of separating what he terms "route planning and

transportation" tasks and determines that the task of finding an optimal solution to such a problem is NP-Hard. Informally this means that the complexity of the problem corresponds to being at least that of the hardest problems that can be solved in Non-Deterministic Polynomial time (from which derives the NP), however it should be noted that this is a lower bound and as such more complex instances are possible. This means that the complexity in the Big O notation is at least $O(n^k)$ for some value $k \in \mathbb{N}$.

#### 5.2.2.2 Complexity During Runtime

During the I2A's execution process at runtime, computational efficiency is exceptionally important. As previously discussed, with the complex requirements of managing a 3D game world, rendering it on the screen and all the other requirements involved in engaging a player with a simulated world experience there is limited computational power available, and so efficiency is a significant factor in ensuring that techniques developed are relevant and usable in industry. This was the entire motivation behind the I2A, and in the remainder of this section, each component of the I2A that is used during the game's execution will be assessed.

**Game World**

During runtime, the main computational power of the device is dedicated to running the Unity engine that powers the game world. Crucially however, this is the case regardless of what AI technique is in use, so will not change between different implementations. As a result, a thorough complexity analysis of Unity can be set aside. Instead, what must be considered is the process by which the state variables discussed in Section 5.1.2.2 are updated. Since this process involves simply assessing the current situation, and looking

up locations and states for specific objects it will be a linear process based on needing to update $v$ values, or O($v$)

### Environmental Sensing

The Environmental Sensing component is responsible for detecting those hazards and items within a certain range from the Goat. As noted above, the best way to achieve this is through the Physics.OverlapSphere method native to Unity, however this obfuscates its implementation inside the Unity engine. Naively though, in the worst case it is possible to imagine an implementation of this routine that steps across every GameObject present in the Unity scene and compares that object's location with the coordinate range of the sphere being described. It would be expected that there are more subtle and elegant solutions to this, but in the worst case then, this routine is O($m$), being a linear process against all $m$ of the GameObjects within the scene.

For the $n$ GameObjects that are within the sphere each one must be assessed to determine its type and for those whose type is one that will generate influence, its location should be noted. However, this too is a linear process since there are simply several steps that may be taken for each of the $n$ GameObjects, meaning that this part of the process can be captured as O($n$). For this component as a whole, the $n$ GameObjects inside the sphere reflect a subset of the $m$ GameObjects in the scene, so without better understanding of the proprietary approach taken by Unity's Physics.OverlapSphere method, it is plausible that this is the computationally dominant part of the process. As a result, the Environmental Sensing component can best be approximated as O($m$).

### Asynchronous Replanning

In terms of computation, by far the most expensive piece will be the Asynchronous Replanning component since it is solving a planning problem. However, with that said, it is

important to recall that this component is using the abstracted Common Representation as the basis for the planning problem being undertaken, and regardless of its complexity, this is a process that is expected to occur over multiple frames meaning that although its complexity may seem high, the impact of this will be ameliorated over time.

With reference again to Helmert's "Understanding Planning Tasks"[35], it is of note that although optimal planning is NP-HARD, approximation of such plans under certain constraints can be achieved much more quickly. In particular Helmert lists in his Conclusion 7.1.5: "*Optimal plans in the* TRANSPORT *and* ROUTE *domains can be approximated by some constant factor if all mobiles have the same capabilities and mobile capacities (for* TRANSPORT *are either all 1 or all unrestricted)* ". In the case of Mountain Goat Mountain, there is a single mobile element that constitutes part of the planning problem, namely the Goat, so this constraint is satisfied. As a consequence, the Asynchronous Replanning component is working with a problem whose complexity class is APX which is to say that it is a Non-Deterministic Polynomial Time task, being approximated in (deterministic) Polynomial Time. Technically speaking this means that it still has $O(n^k)$ complexity similar to the Initial Planning component, but importantly it is now no longer a non-deterministic process, which will mean that its behaviour at runtime is significantly more predictable. Additionally, because the number of nodes in the state space has been reduced through the abstraction factor. Equation 3.4 put the number of clusters as being contingent on the square root of the number of nodes in the graph, meaning that the reduction in complexity carries through to actually give $O(\sqrt{n}^k)$ or simplified, $O(n^{k/2})$.

**IIL**

Updating the IIL is undertaken as necessary, when new sources of influence are detected

or old ones require updating. When that updated information is coming from the Environmental Sensing component there will be $m$ sources of influence to propagate across $n$ nodes in the Common Representation. This process will be driven in large part by the reward propagation method in use, but intuitively each source of influence will first apply at its relevant location, and will radiate across the network. At each node, the influence being applied is contingent purely on the influence that its parent received, so there is no necessity to consider other nodes. As a result, each propagation step will be a straightforward set of calculations. As a result updating based on Environmental sensing will be linear complexity based on $m$ and $n$ or $\mathrm{O}(m * n)$.

If the influence updated is from the Asynchronous Replanning component, then the process undertaken is slightly different in that the plan generated must be analysed to determine which of its steps pass through a Focal Node so that that Focal Node can be deemed an Active Focal Node and a source of influence. The process of determining whether a plan step passes through a Focal Node will involve a linear time operation based on the number of steps in the plan, $s$. This will determine how many Active Focal Nodes there now are, which will all become sources of influence as in the previous paragraph represented by $m$. As such, the IIL update generated by Asynchronous Replanning is approximated by $\mathrm{O}(s + m * n)$.

### Action Choice

As discussed above, the choice of action for the I2A is determined using an A* search, modified to be a localised expansion-bound process of determining the best square in the nearby vicinity for the Goat to move towards. The typical A* search has its computational complexity measured in terms of the perceived imprecision in the heuristic calculations, which makes intuitive sense since a provably accurate heuristic will turn the

search problem into a very straightforward depth-first tree traversal to the goal state. Equally for a wholly inaccurate heuristic, the whole tree must be explored in order to discover the goal state. For the I2A, the process is somewhat different since the aim is to effectively leverage the best-first search strategy to find a locally optimal result within a specific horizon. This means that in the general case, the A* implementation will be limited to at most $e$ node expansions. In broad terms, the A* algorithm starts from the initial state, which for the I2A in this case is the Goat's current location as a node in the IIL. It will then consider each of the nodes reachable in one step and add each one, with an assigned heuristic value of its perceived influence value (factoring in the steps taken from the initial state to reach it) and add them to a list. From that list it will select the highest scoring node, and consider each subsequent node reachable in that list, adding them to the internal list with their respective values. The algorithm progresses, always selecting the best scoring node from the list first. After it has repeated this process $e$ times, it has determined the best candidate node for the I2A to attempt to reach and can pass the initial action required to reach it as the action choice. Because this utilisation of the A* algorithm is not about optimal searching, the efficacy of the heuristic is not a factor in the computational complexity, and the process as a whole can be approximated as $O(e)$.

### 5.2.3 Complexity Summary for Mountain Goat Mountain

To summarise the above, it has been demonstrated that during development by far the most computationally complex piece of the process is the Initial Planning required in order to pre-seed the I2A with a sense of how to achieve its objectives. This was an expected result, since a large part of the motivation for the I2A is that planning is far too costly from a computational point of view. Because this process is undertaken during

development however, the impact of it is dramatically lessened since there are significant resources available for this kind of "off-line" process.

More interesting is the complexity analysis of the runtime components of the I2A which clearly show that - with the exception of the Asynchronous Replanning component, the complexity involved is kept very manageable and in fact is linear in relation to the number of nodes and sources of influence in the environment. The Asynchronous Replanning component is the most complex but again this is as expected. The complexity and overhead of this component is the motivator for it being asynchronous and executing in such a way that the I2A is able to continue its operation while the AR is generating a new set of data to incorporate into the IIL. Utilising the abstracted CR representation as a planning formulation also allows for a considerable efficiency gain since there are far fewer nodes to consider. As noted above, this changes an $O(n^k)$ process to $O(n^{k/2})$. This mitigates the potential impact of the AR on the runtime performance of the I2A. Given these safeguards, and the low complexity of the remaining components, the I2A is provably a lightweight approach to decision making at runtime.

One final aspect of note is the complexity of the Action Choice component, which is $O(e)$. This value $e$ is the only value that is both a limiting factor on the complexity of the system and a tunable, designer led parameter. This means that by varying $e$ directly, the I2A can be tuned to have a slightly lower or higher computation time, and consequently have a longer or shorter lookahead horizon in its choice of action. This flexibility is another key strength of the I2A, and having a direct impact on runtime complexity in this manner is an exceptionally powerful tool.

#### 5.2.3.1 Generalising to Larger Problems

Mountain Goat Mountain is in some regards a well behaved game world in that it is not especially complex. There is an amount of dynamism as there are hazards to be avoided, but none of these hazards have agency - there is no malevolent intelligence attempting to confound the Goat, nor are there particularly convoluted actions within the game world. There are no deep intricate layers of logical thinking required in order to address the kind of scenario described by the Sussman Anomaly described in Section 2.1.3.1. This makes the MGM example a little simplistic in some ways compared to richer environments, however the core operation of the I2A would remain the same in more conceptually complex games. The likelihood is that such features would cause the planning and re-planning components to require more computation, but this extra complexity would not cause an undue burden on the processing time in the other components since they are not bound to the complexity of the planning task being attempted. Similarly, more complex games might have multiple mobile objects that can be moved (analogous to having multiple goats), or more varied hazards, but in either case, these would expand the linear-time components of the I2A's reasoning system so would be unlikely to cause significant impact.

One potential point of concern when considering larger problems is that typically, the issue of complexity is considered to be a trade-off. Above the emphasis has been put on so called “Time Complexity” which is to say the amount of time that the algorithm takes to run, but another factor to consider is “Space Complexity” or the memory required in order to perform the algorithmic task. In essence this concept is the key to the I2A since the Common Representation, based as it is on the Cartesian Product of multiple Domain Transition Graphs, is expanding the memory usage of the algorithm in order to

decrease its runtime. For the kinds of problems described, this has generally not proven to be a problem but the approach taken is unlikely to scale indefinitely and may in fact be a contributing factor to the results seen in Table 4.2. This will be discussed in more depth in Section 6.1.2.

#### 5.2.3.2 Comparative Complexity of Alternate Techniques

In Section 5.1.3 comparison was drawn to three techniques that could be used to power the agent in the Mountain Goat Mountain example presented. Of course, not only is the behaviour of the agent under each of these algorithms an important consideration but also the complexity of these alternatives, which will be addressed here. Automated Planning is perhaps the simplest of these to address since its complexity has already been discussed above as part of the Initial Planning component. A system based purely on this approach would be equivalent to running just this component once. This would mean that it would have complexity $O(n^k)$, but would as mentioned previously be exceptionally high risk. A Subsumption architecture as described in Section 2.3.1 has $b$ prioritised behaviours with those with a higher priority "subsuming" the output of the lower priority ones. The architecture works by iterating across the list of behaviours and determining which is active given the current world state, and each behaviour is a simplistic mapping to an output. As a consequence, the Subsumption architecture is clearly $O(b)$. This is a very clear statement on the Deliberative and Reactive paradigms, one that presents an NP-Hard algorithm that is vulnerable to dynamic threats and invalidation, the other that is a linear-time algorithm but has no concept of long-term strategy.

In these terms it's easy to see the parallels between the I2A and a Three Layer Architecture, as well as the distinctions between them. A traditional TLA approach will first solve the deliberative problem which may be through planning, an $O(n^k)$algorithm. It

will then monitor execution in order to detect a threat to the execution of the generated plan and when such a situation is detected, it will pass control over to a reactive system such as a Subsumption architecture in order to adequately deal with the threat, which is O($b$). Authority then passes back to the deliberative component which will now have to start from scratch using the resulting state after the reactive operation as the initial state in a new planning problem, another O($n^k$) operation. Having to repeatedly solve an NP-Hard problem is clearly a suboptimal approach. This serves to reinforce that solutions derived from neither Reactive nor Deliberative paradigms alone are well suited to the kinds of problems described above, and demonstrates that in the context of computational complexity, attempts at hybridisation such as the Three Layer Architecture have combined the worst of both world rather than the best.

## 5.3 Overview of Algorithmic Analysis

This chapter has presented a detailed assessment of the I2A both from an implementation and complexity point of view, in the context of an example application that was recently created in an industry setting. This allowed for the flow of the I2A to be demonstrated and for each of the components to be clearly described, with particular reference to their interdependency and interoperation. Additionally, the computational complexity of each component was assessed in order to determine the overall complexity of the I2A with comparison drawn between this and alternative approaches. This serves to highlight the strengths of the I2A, in that it is a relatively lightweight architecture that is flexible and well suited to scenarios that need to combine both reactive and deliberative forms of reasoning. It is designed to be resistant to the kinds of threats that typically invalidate a purely deliberative solution by constantly being informed by

influence from both paradigms. This lessens the reliance on the complex deliberative reasoning process since information from this component is not simply thrown out when such a threat is tackled. This means that there is a significant increase in the system's ability to operate in dynamic environments when compared to alternative approaches, but crucially due to moving much of the computationally complex calculations into the development process, this does not come with a punitive increase in computation against typical reactive solutions. As noted above, the Asynchronous Replanning component is the single largest piece of computation that must be tackled during the I2A's runtime execution, and by design this piece is intended to be used infrequently and given time to perform its task since it is asynchronous from the main execution of the core of the I2A. Aside from this component, the I2A executes in linear time, making it comparable to solutions such as Subsumption Architectures in terms of computational complexity, whilst retaining the ability to perform long term reasoning that Subsumption lacks.

With reference to the research statement presented in Section 1.3, and particularly to the evaluation criteria presented, this chapter has fulfilled the third and fourth points in that it has clearly demonstrated both the computational efficiency of the I2A against accepted deliberative solutions as well as a clear ability to handle a dynamic environment by being able to react to the changing environment that cannot be reasoned about deliberatively.

# Chapter 6

# Discussion

The I2A has the power and flexibility to offer a much more sophisticated approach to robust agent execution than previous architectures have provided. Most significantly, the idea of combining information from the Deliberative and Reactive paradigms, rather than arbitrating between the two, represents a significant step forwards.

However, with that said, what has been presented still requires further work to be fully fleshed out. This broadly falls into three categories: ways that the technique could be further improved, considerations to bear in mind as the I2A is developed further and finally a discussion of the aspects of the I2A that are parameterised and the impact of varying these parameters.

## 6.1 Immediate Improvements

The work undertaken to date has indicated that the I2A appears to be viable as a technique, but that is not to say that the design of the architecture is perfect. There are currently a number of areas that could be improved upon, such as being able to handle

a wider range of domains, a broader selection of sources of influence or reformulating the architecture to take advantage of advances in modern technology such as general purpose computation on graphics processing units (GPGPU).

### 6.1.1 Influence Propagation as a Vector Operation

One of the core advantages of the I2A approach is that because it can represent the graph of the state space as an adjacency matrix, it is possible to represent the influence propagation steps as a number of iterated instructions that act on that matrix. This is significant because this means that it can be easily adapted to operate in an inherently parallel manner and make use of systems designed for the “Single Instruction Multiple Data” (SIMD) paradigm such as modern Graphics Processing Units, which typically implement GPGPU either through NVidia’s CUDA or OpenCL.

#### 6.1.1.1 Worked Example

As an example, in this section, the Reward Sharing method of propagation shown in Chapter 3 will be shown as a sequence of vector operations, which implicitly lend themselves to being processed on these platforms.

For this example, the small example used in Section 3.7.1.1 will be revisited. The underlying graph is shown in Figure 6.1. This can be represented quite easily as what is called an ”Adjacency Matrix”, which indicates which nodes are connected together using a boolean-style representation. A value of 1 indicated an edge exists, and a 0 that it does not. Note that edges are not bidirectional in this representation.

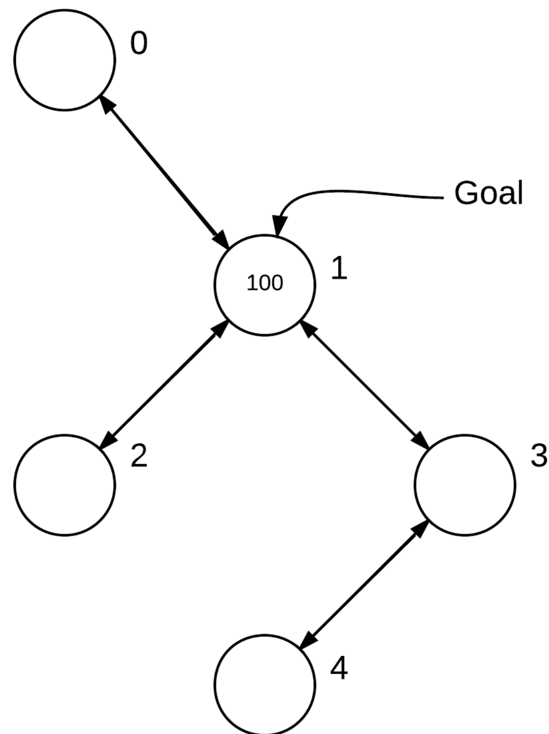


Figure 6.1: The Basic Propagation Example - Node 1 is a Goal Node

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 | 0 |

It is also possible to express the current Influence Landscape as a vector showing the influence value of each node as well as identify a specific node as a vector whose components are 0 except for nth component, to represent node N. Below, V represents the initial value of the IL and G highlights that Node 1 (the Goal Node) has been modified and is pending propagation. The adjacency matrix is represented by M.

$$V = \begin{pmatrix} 0 \\ 100 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad G = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The parent nodes, those having edges terminating at the node referenced by G can be found as N in the equation below.

$$
\begin{aligned}
N &= M \times G \\
&= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}
\end{aligned} \tag{6.1}
$$

Where I is an Identity column vector, the dot product of I with N provides the number of parents that the current node has. Similarly taking the dot product of V and G gives the amount of influence that the current node has. Subtracting 1 from the current influence at the node, and dividing by the number of parents gives the amount of influence that the parents should share in. Using this as a scalar multiple with N gives a new vector reflecting the updated influence for each parent node.

$$\mathbb{I} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$N \bullet \mathbb{I} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \bullet \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = 3 \tag{6.2}$$

$$V \bullet G = \begin{pmatrix} 0 \\ 100 \\ 0 \\ 0 \\ 0 \end{pmatrix} \bullet \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 100 \tag{6.3}$$

$$N * \left( \lfloor \frac{(V \bullet G)}{N \bullet \mathbb{I}} \rfloor - 1 \right) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} * \left( \lfloor \frac{100}{3} \rfloor - 1 \right) = \begin{pmatrix} 32 \\ 0 \\ 32 \\ 32 \\ 0 \end{pmatrix} \tag{6.4}$$

The result is then compared against V, which is the established landscape from which is being propagated, and components-wise the maximum amount of influence at each

node is selected to produce a new view of the landscape.

$$
\begin{aligned}
U &= max\left(N * \left(\lfloor\frac{(V \bullet G)}{N \bullet \mathbb{I}}\rfloor - 1\right), V\right) \\
&= max(\begin{pmatrix} 32 \\ 0 \\ 32 \\ 32 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 100 \\ 0 \\ 0 \\ 0 \end{pmatrix}) \\
&= \begin{pmatrix} 32 \\ 100 \\ 32 \\ 32 \\ 0 \end{pmatrix}
\end{aligned} \tag{6.5}
$$

As three nodes have had their values updated, the effect propagation will have on these values must be considered, so where G was initially a single vector, it can instead be thought of as a set of vectors whose initial size was one. The relevant nodes to add to $\{G\}$ can be calculated by considering the column vectors of an n x n Identity Matrix ( $\mathbb{IM}$ below) and taking each with the new influence values calculated for the parent nodes at this step, as in Equation 6.4. If the value found is 0, then influence has not been updated at this node, otherwise it has and the relevant $\mathbb{I}_n$ must be added to $\{G\}$. In the equation below this process is shown using $\{G'\}$ to represent the updated contents of $\{G\}$.

$$\mathbb{IM} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.6}$$

$$\mathbb{I}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \mathbb{I}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \mathbb{I}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \mathbb{I}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \mathbb{I}_5 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{6.7}$$

$\forall \mathbb{I}_n :$

$$\left(\left(N * \left(\lfloor \frac{(V \bullet G)}{N \bullet \mathbb{I}} \rfloor - 1\right)\right) - V\right) \bullet \mathbb{I}_n > 0 \implies \quad \{G'\} = \{G\} \cup \mathbb{I}_n \tag{6.8}$$

$$\left(\left(N * \left(\lfloor \frac{(V \bullet G)}{N \bullet \mathbb{I}} \rfloor - 1\right)\right) - V\right) \bullet \mathbb{I}_n \leq 0 \implies \quad \{G'\} = \{G\} \tag{6.9}$$

$$\{G\} = \{\}$$

$$\mathbb{I}_1$$

$$\left(\left(N * \left(\lfloor\frac{(V \bullet G)}{N \bullet \mathbb{I}}\rfloor - 1\right)\right) - V\right) \bullet \mathbb{I}_1 = \left(\begin{pmatrix}32\\0\\32\\32\\0\end{pmatrix} - \begin{pmatrix}0\\100\\0\\0\\0\end{pmatrix}\right) \bullet \begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix} = 32 \tag{6.10}$$

$$\{G'\} = \{G\} \cup \mathbb{I}_1 \tag{6.11}$$

$$\{G'\} = \left\{\begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix}\right\} \tag{6.12}$$

$$\mathbb{I}_2$$

$$\left(\left(N * \left(\lfloor\frac{(V \bullet G)}{N \bullet \mathbb{I}}\rfloor - 1\right)\right) - V\right) \bullet \mathbb{I}_2 = \left(\begin{pmatrix}32\\0\\32\\32\\0\end{pmatrix} - \begin{pmatrix}0\\100\\0\\0\\0\end{pmatrix}\right) \bullet \begin{pmatrix}0\\1\\0\\0\\0\end{pmatrix} = -100 \tag{6.13}$$

$$\{G'\} = \{G\} \tag{6.14}$$

likewise for $\mathbb{I}_3$, $\mathbb{I}_4$, $\mathbb{I}_5$

$$\{G\} = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\} \tag{6.15}$$

Continuing the example, G now contains three vectors as shown. For each one, the result of sharing the influence at the relevant node to that node's parents is computed. Note that for G1, since the influence calculated for Node 1 is lower than that which is already recorded for that node, the original value is kept and the result of the propagation for the vector G1 remains V. G2 produces identical results and is omitted.

$$G_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad V = \begin{pmatrix} 32 \\ 100 \\ 32 \\ 32 \\ 0 \end{pmatrix} \tag{6.16}$$

$$N = M \times G_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{6.17}$$

$$N \bullet \mathbb{I} = 1 \tag{6.18}$$

$$V \bullet G_1 = 32 \tag{6.19}$$

$$\begin{aligned} U &= max\left(N * \left(\lfloor\frac{(V \bullet G_1)}{N \bullet \mathbb{I}}\rfloor - 1\right), V\right) \\ &= max\left(\begin{pmatrix} 0 \\ 32 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 32 \\ 100 \\ 32 \\ 32 \\ 0 \end{pmatrix}\right) \\ &= \begin{pmatrix} 32 \\ 100 \\ 32 \\ 32 \\ 0 \end{pmatrix} = V \end{aligned} \tag{6.20}$$

When the resultant influence vector is calculated for G3, Node 4 receives an updated influence score of 15. As this is higher than the previous recorded value, this is updated by the maximum operation.

$$G_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad V = \begin{pmatrix} 32 \\ 100 \\ 32 \\ 32 \\ 0 \end{pmatrix} \tag{6.21}$$

$$N = M \times G_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{6.22}$$

$$N \bullet \mathbb{I} = 2 \tag{6.23}$$

$$V \bullet G_3 = 32 \tag{6.24}$$

$$U = max\left(N * \left(\lfloor\frac{(V \bullet G_3) - 1}{N \bullet \mathbb{I}}\rfloor - 1\right), V\right) \tag{6.25}$$

$$= max\left(\begin{pmatrix} 0 \\ 15 \\ 0 \\ 0 \\ 15 \end{pmatrix}, \begin{pmatrix} 32 \\ 100 \\ 32 \\ 32 \\ 0 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 32 \\ 100 \\ 32 \\ 32 \\ 15 \end{pmatrix}$$

In the next iteration, Node 4 shares its influence to its sole parent, giving Node 3 a value of 15. Since this is lower than the current 32, this is not recorded, and the network has reached stability.

This process is functionally identical to the Reward Sharing Propagation described above but demonstrates that because one of the fundamental structures underpinning the I2A is a matrix representation of the state-space, this means that it can relatively easily become a process very well suited to the more advanced processor architectures available now. In particular, note that the SIMD system was leveraged to evaluate an update to the whole landscape as one step, whereas a traditional model would have considered each node separately.

If the I2A were to be adapted to follow this processing paradigm, it is likely that there would be even further reductions in processing required to drive the architecture, which

would allow even more to be done with it.

### 6.1.2 Memory Constraints

The results presented have shown that in domains that resemble typical game environments the I2A system is able to reliably reformulate a PDDL description of the game world into the Common Representation. However, due to the combinatorial explosion of the reformulation and especially the Cartesian Product process, it has been shown that this can under certain domains (especially those with a high number of entities), become too large a problem for the existing preprocessing system to work with. This is the likely cause of the results shown in Table 4.2 and briefly alluded to in Section 5.2.3.1. As the number of DTGs being combined increases, the amount of space required to represent this grows exponentially meaning that effectively computational efficiency is being gained by sacrificing storage efficiency.

To some extent, solving this could be a simple matter of creating a more efficient application to undertake the preprocessing, and as such be a relatively simple software engineering challenge, but this is likely to only be a stop-gap that simply allows more domains to be processed without solving the underlying issue. More sophisticated methods would likely have to analyse the manner in which the Cartesian Product of the graphs is calculated during preprocessing alongside the clustering system and attempt to find a solution that allowed for an interleaving of clustering whilst the combinatorial explosion was occurring so that at no point could the representation being stored in system memory get too unwieldy. This is likely to be an appropriate approach since by far the majority of the edges within the full Common Representation are contained with the individual DTGs that contribute to the whole state space. By alternating between

clustering and the Cartesian product process, each DTG can be added and then reduced into the abstract representation.

This solution is likely to only be appropriate for specific domains, and further work would need to be undertaken to establish the viability of this process, and whether those domains for which it was suitable shared the characteristics of those that seem to cause the memory issues in the first place.

### 6.1.3 Extensibility

It should also be noted, that much like the previously mentioned Infinite Axis Utility system discussed in Section 2.2.2.2, the I2A is a very extensible approach - Deliberation and Reaction represent two of the most prominent information sources that might inform reasoning, however there are many other potential sources of influence. For NPC characters an example might be an influence source that notates which states represent more "human-like" behaviours, naturally biasing the I2A system towards these areas and provide more realistic action sequences for NPCs. For more traditional applications, it might be a representation of some other expert knowledge, such as an intuition based on years of experience with a manufacturing process.

Regardless of the meaning of these other sources of influence, the I2A is designed in such a way that their values can be incorporated into the reasoning system easily and without any particular bias towards one paradigm or another. The I2A, and the underlying representation of the world, provides a common robust system through which these other sources can easily be captured, converted into Influence Landscapes and factored into the results.

It is worth highlighting that some techniques for combining the landscapes will ensure that the upper and lower bounds used in the individual landscape are respected, whilst others violate these. Consider an addition method against a maximal value method in a situation where a node has a high positive score in both landscapes. Under addition, the new value will exceed the possible values in either landscape, whilst under a technique that selects the maximal value, the highest possible value in either landscape remains the boundary in the IIL. This becomes more of a significant issue when further extensibility is considered and additional sources of influence landscapes are brought into the architecture. This is largely the kind of issue that the Infinite Axis Utility System has already overcome, but care must be taken as there is a fundamental distinction between the two in that a score of 0 in an I2A landscape does not explicitly negate the input from other landscapes as it does in the IAUS. More formally, the landscapes are designed to inform the decision, but no landscape should have the ability to override the information from any other landscape, and care must be taken when combination techniques are implemented to preserve this.

## 6.2 Future Considerations

Future work on the I2A should not be undertaken in a vacuum, and as such it is important to point out several things that should be kept in mind as the technique is developed further that will help to signpost the direction that this work should be developed in.

### 6.2.1 Black Swans

In his book The Black Swan, Nassim Nicholas Taleb presents the idea of events that have extremely low probability of occurrence yet whose impact on the outcome is very

significant.[100] In general, their low probability causes these potential events to be overlooked when modeling systems and interactions, and yet their consequences are so great that this oversight can cause drastic consequences to the model. Taleb refers to the events of September 11th 2001 as a classical Black Swan - something never predicted or given credibility due to the perceived low likelihood of occurrence, and yet having occurred has had far reaching implications.

Taleb goes further and also discusses what he terms “The Ludic Fallacy”, which is best exemplified with an example taken from The Black Swan which describes a Las Vegas casino. The casino employed sophisticated systems in an attempt to detect cheating in order to manage the risk of loss that the company was exposed to. However, a simple look at their paperwork would show that by far the greatest losses in the casino came not from the gaming but instead from more mundane sources - theft by a former employee, on-stage accidents, fines due to incorrectly filed paperwork and a kidnap ransom.

Taleb postulates that the idea that a situation can be modeled and the act of modeling it create something that can be solved or a game to be played and beaten (hence the term 'Ludic' which derives from the Latin term for play). This in turn blinkers us into having something that as humans we insist on solving. The casino became fixated on the gaming aspect of their business because it was something that they could model, quantify and reason about despite the fact that objectively, it was not necessarily the most important part - the most important parts were excluded from the model altogether. In general the field of AI is very highly susceptible to the Ludic Fallacy, modeling a problem and then attempting to solve the model to optimality, with at times a flagrant disregard for the necessity that this solution be relatable to the real world.

In 2002, US Secretary of Defense Donald Rumsfeld made an often quoted statement,

at the time ridiculed for its obtuseness. "There are known knowns; there are things we know that we know. There are known unknowns; that is to say, there are things that we now know we don't know. But there are also unknown unknowns – there are things we do not know we don't know."[101] In fact, Rumsfeld is making an astute observation about the nature of the models:

- Unforeseen Consequences - This class of flaw is simply when executed actions do not have the expected effect. If a robot attempts to pick up a box and the box slips and falls, this is an unexpected effect - the box is supposed to be in the robot's gripper, instead has been left on the shelf. Unforeseen Consequences mean that the world will not necessarily transition in the ways that the system expects, but its possible to describe the resulting effects because they are still part of the enumerated state of possibilities.
- Unanticipated States - The limitations of any model are that it is bound up in the states that can be enumerated, which is only as good as the domain knowledge provided. Consider a coin being flipped, traditionally it is stipulated that there is a 50% chance of it landing on heads, and a 50% chance of it landing on tails. However, this is not strictly true, it is an abstract representation. There is a vanishingly small, but importantly non-zero, chance that a flipped coin will land on its edge, however due to its low likelihood, this is never modeled Equally for the gripper robot, if during its attempt to pick the box up it were to fall on the ground, and if the ground were to not be part of the modeled locations that the box was expected to occupy, the robot is now in an Unanticipated State.

In general, the nature of game worlds creates a natural solution to these problems inasmuch as they themselves are modeled, so for the most part the assumptions made

about the world hold true - the noise of the real world is not replicated. With that said, a lot of gameplay is inherently emergent, and the interoperation of a number different mechanics can create Unanticipated States. As an example, consider the behaviour exhibited in The Elder Scrolls V: Skyrim (introduced in Chapter1), in which the player could take any item in the world, but if the NPC who the item belonged to saw it being taken, it would cause the player to be accused of theft. At the same time, players were able to manipulate items within the world, move them and place them down. As you will recall, players quickly discovered that by interacting with buckets, carrying them to the NPCs and carefully placing them on their heads, they could break the perceived line of sight to the thefts occurring; there was no scope in the AI system driving the NPCs to reason about them having a bucket placed over their head, so the AI found itself unable to adequately act because it was in a Unanticipated State.

This notion of the Black Swan, and its potential impact, is highly relevant for the I2A as it is for all AI systems with aspirations to act in a realistic environment. Typically our models are built in a manner that is borne of the Ludic Fallacy. That is to say, simplified in such a way that they are more easily computed and more easily "beaten", and to an extent this is necessary in order to formulate an abstract representation, however it is not a "lossless compression". The I2A inherently is designed to cope with the concept of Unforeseen Consequences, however it is not currently able to handle Unanticipated States. This weakness means that models must be designed carefully to ensure that these circumstances, such as those found in Skyrim do not occur and create a situation where the agent is forced into a state outside of that which has been modeled

It should also be noted that tackling this challenge directly is likely well outwith the scope of the I2A, since it remains one of the big challenges remaining in the quest for General AI. The ability to adapt and update the internal model of the world, as well as

reason about connections within that model and states that might result, is a significant piece of work in itself.

### 6.2.2 Procedural Content Generation

An important contemporary game design paradigm being used by a number of developers now is Procedural Content Generation (PCG), with examples such as Warframe (Digital Extremes, 2013) in large productions[102] and Easy Money (Robot Overlord Games, 2015).[103]. Rather than creating content such as a level design during development, instead the developer creates a systemic way of generating this content at runtime. From the developer's perspective this allows them to create a game that has ever-changing content, enhancing the game's value to players and allowing for it to constantly feel fresh. It allows for players to play even simple games without ever being able to learn the level.

Typically (although not exclusively) PCG uses an algorithmic approach to place prefabricated pieces of a level together as a sort of jigsaw. For example endless runner games such as Stampede Run (Zynga, 2013) see the player running through a maze and having to make their character dodge around obstacles and follow the path by turning corners. The layout of the path through the maze is not fixed in advance, but it is assembled from a series of ready-made components at runtime. As was mentioned in Section 5.1.1, this method is actually the way that the commercial version of Mountain Goat Mountain functions, with the slightly simplified version described in the previous chapter being used to remove the concerns that will be laid out below.

Figure 6.2 shows two sections of maze in Stampede Run which have been positioned within the level procedurally. The individual "jigsaw" pieces are designed to fit together

FIGURE 6.2: An example from Stampede Run, each section of path is laid out procedurally and many are reused

seamlessly, meaning that the maze as a whole can be almost endlessly generated in different configurations. Many games reuse this philosophy and within one piece allow different configurations of obstacles and items to be positioned, creating a significantly large number of distinct levels. The process by which this is achieved uses a semi-random selection in order to ensure that certain constraints are met, and that the maze is both structurally sound and also fun (whereas a true random placement of the jigsaw pieces would not be able to guarantee this).

Because the level, and even the characters and objects within that level, are placed at runtime, this presents a fundamental challenge to the I2A which anticipates being able to pre-compute much of its infrastructure, such as the Common Representation, during development. As such, this is not immediately compatible with the I2A, but there are likely some techniques that could be taken that might yield interesting results.

Specifically, as the Common Representation is an exhaustive enumeration of the possible states in the game world, it may be that it is possible to build the CR piece-wise at runtime from sections that are based on the possible states within each of the pieces available for procedural combination. This would at least simplify the process of creating the CR at runtime, since it would become a problem of connecting graph segments together in a coherent way, rather than trying to calculate the CR from scratch. This may prove to be an effective technique and allow the I2A to continue to be performant even when combined with the PCG paradigm, but further work must be undertaken to explore this.

### 6.2.3 Generation of Entities

One final thing to bear in mind when considering the I2A and its applications is a fundamental limitation of PDDL that could potentially be problematic. PDDL is, as has previously been discussed, a symbolic representation of a game world, but it should be noted that it has no ability to represent the creation of entities as part of its reasoning. That is to say that the elements that occupy a game world at the start of the plan will always occupy the game world at the end (but could be in some terminal state that removes them from the world as perceived by the player), and no additional entities will be added to the world.

This has a significant impact on the way certain situations and games can be modelled. Consider a real time strategy game in which the object is to build a base and an army and then attack. In PDDL, there is no way to symbolically represent the result of building units for the army since the language cannot have entities added to it. As a consequence, abstract representations are required, perhaps representing army strength as a PDDL fluent to be increased, with the various units adding to that, however this

is clearly a level of reasoning removed from what might otherwise be used to represent the problem.

This isn't an insurmountable issue, but does highlight that there are modelling challenges inherent in the I2A due to some of the underpinning technologies, and that these must be taken into account when looking to build an I2A agent for a game environment.

## 6.3 Parameters and Tuning

One of the key elements of the I2A as it has been specified in this work is its flexibility to empower designers and developers to implement it in a way that makes sense for their specific game, rather than forcing certain decisions as part of the architecture. This does however mean that there is significant scope for tuning of parameters within the architecture in order to elicit the desired behaviours.

The three most significant aspects that can be altered are the influence propagation system, the method by which influence landscapes are combined and the localised search algorithm that is used to determine the best action for the agent to take.

- **Influence Propagation** - The method by which influence propagates across the CR is important for determining the manner in which the agent will act. The work undertaken has used a simple reward sharing heuristic, but alternatives could be used in place of this which would perhaps more quickly (or slowly) decay the perceived value of a state as it propagates, possibly based purely on edge distance rather than sharing proportional to the number of parent nodes.

- **Combining Landscapes** - As discussed previously, the way that landscapes are combined to create the Integrated Influence Landscape will have a significant impact on the behaviour of the agent as this is where traits such as caution (taking a minimal value of all landscapes) and confidence (taking a maximal value) can be introduced. It would also be possible to weight one landscape more than another, which would give behaviour that was more spontaneous or deliberate. The manner in which the IIL is calculated from its constituent parts will have a great impact on the tone of the agent.

- **Localised Search** - The mechanism by which the agent performs its search for a nearby state to reach will also have great bearing on the tone of the agent. As noted in Section 5.2.3 this parameter represents is a trade-off between lookahead search and computational complexity, and invariably a larger number of permitted expansions in which this search is conducted will slow down the process but also remove much of the flexibility of the technique, since before a far away state is reached, there is likely to be disruption to the agent's perception of the world. In general the specific horizon for how far ahead this search should look will be implementation-dependent, and will be impacted most by how much disruption there is likely to be; the more disrupted the agent's actions are likely to be, the smaller the search neighbourhood should be.

Different combinations of approaches for each of these three will generate very different behaviour in agents even within the same scenario. It is up to the game developer to experiment and determine what combination best suits the tone that is required for the behaviour of the agent.

## 6.4 Summary

This chapter has highlighted some areas where the I2A could be improved, as well as several things that must be considered when both extending or implementing the system. As mentioned previously, the work to date has emphasised showing the viability of the I2A rather than finding optimal parameters, and as a result there are a number of open questions as to how best to implement the systems that have been discussed above. Moving forwards, these will become more important now that the I2A has been demonstrated to be a robust methodology, and subsequent work can explore more thoroughly these questions and how they relate to specific instances of implementation. With that said, it should be clear that the I2A has significant potential to be developed beyond what has been accomplished in this work.

# Chapter 7

# Conclusions

The necessity for more emphasis on AI in games is inarguable. Games that have good AI typically are noticed for this, and those that have bad AI will often have this mentioned specifically as a criticism as part of reviews. Consumers want games to have good AI - not in the sense of presenting a tough challenge, but in the sense of allowing a realistic and immersive environment to be portrayed.

With that said, one of the biggest challenges facing AI in games is the computational power available. Ever increasing specifications in consumer-level hardware are alleviating this to some extent, but as has been shown, many state of the art techniques in academia require orders of magnitude more time in order to adequately perform their processing than is available during the execution of a game. Projects such as IBM's Deep Blue tackle this as primarily a hardware problem, creating a super-computer that can perform the computation necessary in an appropriate amount of time, but for a commercial industry this isn't currently viable since this technology is not in every household.

The proposed Integrated Influence Architecture is an attempt to leverage three key observations. Firstly that game characters need to have long term goals and plans

for achieving them, so the kinds of deliberative reasoning Deep Blue performs are still essential. Secondly that there are a range of AI techniques designed to act quickly, and although these are typically relatively naive in their decision-making power, they operate in a manner that is more aligned with what is needed for decision making in games. Finally, because of the manner in which games are developed, there are effectively limitless resources during this period, which provides a valuable opportunity by allowing many aspects of decision making to be pre-computed, thereby reducing the computational overhead needed during the game's execution.

The I2A aims to utilise deliberative reasoning in a lightweight architecture, performing the intensive computation upfront during development and allowing the result of this to be reused during runtime to provide guidance to the execution system.

This represents a relatively novel approach to execution, since previously the predominant method that an agent would use to incorporate these aspects would be more along the lines of arbitration, with the circumstances dictating whether the agent should act in the context of its long term reasoning or short term reactions. Instead, the I2A tries to ensure that this is not an either/or proposition and instead an agent is able to react to changing circumstances, with information about its long term goals.

The work presented has explained the method by which the I2A achieves this kind of execution, and has also shown that there is both evidence to support the nature of the I2A in that fundamentally an architecture built in this manner can achieve what is intended, as well as highlighting the core challenges to integrating this technique within a game developed using industry-standard tools and ways that these can be overcome.

In Section 1.3 four key points were outlined by which the I2A would be evaluated. To reiterate, they were as follows:

- It must be functional, which is to say that it must be able to actually make decisions informed by both Reaction and Deliberation in dynamic environments.
- It must be applicable to industry. One of the core motivations of the work is addressing a shortcoming in contemporary solutions in use by industry, so any proposed solution must in turn be able to address that shortcoming in the industry directly.
- It must be able to handle dynamic environments by reacting to threats and opportunities in a timely manner
- It must demonstrate computational efficiency against deliberative solutions

In the first case, Section 4.1 clearly demonstrated the viability of the technique and showed it to be functional in that it could create a Common Representation and Integrated Influence Landscape for a variety of problems. Its applicability to industry was demonstrated in Section 4.2 where it was demonstrated that an I2A agent could be developed based on one of the most common game development engines in the industry at the time of writing. The third criteria for evaluation was satisfied in Section 5.1 a worked example of how the I2A could be applied to a recent game that clearly demonstrated the type of environment the I2A is designed to address. This also reinforced its applicability to industry. The final point of the evaluation is the need for computational efficiency which was addressed in Section 5.2. On all points, the I2A has been demonstrated to effectively fulfill the evaluation criteria established, and so is deemed to be an appropriate solution to the initial problem described.

## 7.1 Final Thoughts

The necessity for a new approach to decision making is clear, and in this work the case for the I2A has been argued. It has the potential to be a powerful technique given that it is based on a strong theoretical grounding, leveraging elements of existing methods, and there is evidence that it will prove applicable in practice as well. There is significant scope for the technique to be developed further, tailored for specific problems and extended to include other types of reasoning, as well as be adapted to be more suited to the new types of parallel processing systems becoming increasingly available even at a consumer level.

Overall, the I2A shows significant promise, and appears to be a significant and feasible solution to a fundamental weakness in artificial intelligence, namely that of the deep disconnect between deliberative and reactive reasoning. Although this approach has been argued to be a viable strategy for bridging this gap specifically in the domains of games and simulated worlds, it is also reasonable to suspect that the I2A may have potential beyond these, in any application field where fast reaction in the context of long term reasoning is a key factor in exhibiting "intelligent" behaviour.

# Bibliography


[1] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2009.

[2] How to Play Three-handed Bridge, sep 1922. URL http://news.google.com/newspapers?nid=1314&dat=19220924&id=PKxVAAAAIBAJ&sjid=t-EDAAAAIBAJ&pg=2099,6243793.

[3] Nick Montfort. *Twisty Little Passages: An Approach to Interactive Fiction.* The MIT Press, 2005.

[4] Michael Mateas. Expressive AI: Games and Artificial Intelligence. In *Proceedings of Level Up: Digital Games Research Conference*, 2003.

[5] Frank Cifaldi. id's John Carmack Chooses Framerate Over Graphical Fidelity. *Gamasutra.* URL http://www.gamasutra.com/view/news/126649/ids_John_Carmack_Chooses_Framerate_Over_Graphical_Fidelity.php.

[6] Amanda Coles, Andrew Coles, A Olaya, S Jimenex, C L Lopex, S Sanner, and S Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2012.

[7] Fenghsiung Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion.* Princeton University Press, 2002.

[8] Mikael Hedberg. Creating the Battlefield AI Experience. In *Paris Game AI Conference*, 2010.

[9] J. Orkin. Three states and a plan: the AI of FEAR. In *Game Developers Conference*. Citeseer, 2006.

[10] Steam Forum Discussion - Is their[sic] a game with better ai?, 2013. URL `http://steamcommunity.com/app/21090/discussions/0/846957366840442743/`.

[11] Matt Barton. *Dungeons and Desktops: The History of Computer Role-Playing Games*. A K Peters/CRC Press, 2008.

[12] Jeff Cork. In Skyrim, Bucket + Head = The Perfect Crime. *Game Informer*, 2011. URL `http://www.gameinformer.com/b/news/archive/2011/11/11/in-skyrim-bucket-head-the-perfect-crime.aspx`.

[13] R Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 1986.

[14] V Braitenberg. *Vehicles, Experiments in Synthetic Psychology*. MIT Press, 1984.

[15] B Bouzy. Computer Go: An AI oriented survey. *Artificial Intelligence*, 132(1): 39–103, oct 2001.

[16] Morton D. Davis. *Game Theory: A Nontechnical Introduction (Dover Books on Mathematics)*. Dover Publications Inc., 2003. ISBN 0486296725.

[17] L Kocsis and C Szepesvari. *Bandit Based Monte-Carlo Planning*. Springer Lecture Notes in Computer Science, 2006.

[18] D. Nau, M. Ghallab, P. Traverso, and Ebooks Corporation. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers, 2004.

[19] Luke Dicken and John Levine. Applying Clustering Techniques to Reduce Complexity in Automated Planning Domains. In *Proceedings of the 11th International Conference on Intelligent Data Engineering and Automated Learning*, 2010.

[20] Luke Dicken and John Levine. Influence Landscapes - From Spatial to Conceptual Representations. In *Proc. AISB AI and Games Symposium*, 2011.

[21] Luke Dicken, Peter Gregory, and John Levine. Abstraction through clustering: complexity reduction in automated planning domains. *International Journal of Data Mining, Modelling and Management*, 4(2):123–137, 2012.

[22] Luke Dicken. The Integrated Influence Architecture - Combining Reactive and Deliberative AI for NPC Control. In *Proceedings of the 2012 AltDevConf*, 2012.

[23] Luke Dicken, Dino Dini, and Dave Mark. Architecture Tricks: Managing Behaviors in Time, Space, and Depth. In *Proceedings of the Game Developers Conference*, 2013.

[24] D McDermott, M Ghallab, A Howe, C Knoblock, A Ram, M Veloso, D Weld, and D Wilkins. PDDL — The Planning Domain Definition Language. *Technical Report, Yale Center for Computational Vision and Control*, 1998.

[25] Patrick Hayes. The Frame Problem and Related Problems in Artificial Intelligence. Technical report, Stanford University, 1971.

[26] Raymond Reiter. The Frame Problem in Situation the Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. 1991.

[27] M Fox and D Long. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 2003.

[28] Stefan Edelkamp and Jorg Hoffman. PDDL 2.2: the language for the classical part of IPC-04. *Proceedings of the International Conference on Automated Planning and Scheduling*, 2004.

[29] Alfonso E Gerevini and Derek Long. Preferences and Soft Constraints in PDDL3. In *ICAPS-2006 Workshop on Preferences and Soft Constraints in Planning*, pages 46–54, 2006.

[30] Malte Helmert. IPC-2008, Deterministic Part - Changed in PDDL 3.1, 2008. URL `http://ipc.informatik.uni-freiburg.de/PddlExtension`.

[31] Sara Bernardini and David E Smith. Developing Domain-Independent Search Control for Europa2. In *Workshop on Heuristics for Domain-independent Planning: Progress, Ideas, Limitations, Challenges at ICAPS-2007*, 2007.

[32] Michael Brenner. A Multiagent Planning Language. In *Workshop on PDDL at ICAPS-2003*, 2003.

[33] M Fox and D Long. PDDL+ level 5: An extension to PDDL2. 1 for modelling planning domains with continuous time-dependent effects. *Technical Report, U. of Durham*, 2001.

[34] C Bäckström and B Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 1995.

[35] M Helmert. *Understanding Planning Tasks : Domain Complexity and Heuristic Decomposition*. Springer LNAI, 2008.

[36] R Fikes and NJ Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 1971.

[37] PE Hart and NJ Nilsson. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4 (2):100–107, 1968.

[38] James Matthews. Basic A* Pathfinding Made Simple. In *AI Game Programming Wisdom*, pages 105–113. 2002.

[39] A Blum and M Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[40] J Hoffmann and B Nebel. The FF Planning System: Fast plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 2001.

[41] M Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 2006.

[42] Silvia Richter and Matthias Westphal. The LAMA Planner Using Landmark Counting in Heuristic Search. In *Proceedings of the IPC 2008*, 2008.

[43] Fahiem Bacchus. AIPS 2000 Planning Competition: The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems. *AI Magazine*, 22(3):47–56, 2001.

[44] GJ Sussman. A Computer Model of Skill Acquisition. 1975.

[45] Clive Dawson and Laurent Siklossy. The role of preprocessing in problem solving systems: "An ounce of reflection is worth a pound of backtracking". In *Proceedings of the 5th international joint conference on Artificial intelligence*, pages 465–471, 1977.

[46] Adi Botea, Markus Enzenberger, Martin Muller, and Jonathan Schaeffer. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005.

[47] Andrew Coles and Amanda Smith. MARVIN: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 28:119–156, 2007.

[48] Steven Minton. Selectively generalising plans for problem-solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985.

[49] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. *Proc. European Conf. on Planning*, 2001.

[50] HLS Younes and ML Littman. PPDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects. Technical report, Carnegie Mellon University, 2004. URL `http://www.tempastic.org/papers/CMU-CS-04-167.pdf`.

[51] R Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6, 1957.

[52] S Yoon, A Fern, and R Givan. FF-Replan: A baseline for probabilistic planning. *17th International Conference on Automated Planning and Scheduling*, 2007.

[53] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proceedings of International Conference on Automated Planning and Scheduling*, pages 212–221, 2006.

[54] Mat Buckland. *Programming Game AI by Example*. Wordware Publishing Inc., 2004. ISBN 1556220782.

[55] Alex Champandard. 10 Reasons the Age of Finite State Machines is Over, 2007. URL http://aigamedev.com/open/article/fsm-age-is-over/.

[56] Brian Schwab. *AI Game Engine Programming*. Delmar, 2nd edition, 2009. ISBN 1584505729.

[57] John Krajewski. Creating All Humans: A Data-Driven AI Framework for Open Game Worlds, 2009. URL http://www.gamasutra.com/view/feature/130279/creating_all_humans_a_datadriven_.php?print=1.

[58] O. Ahmad, J. Cremer, J. Kearney, P. Willemsen, and S. Hansen. Hierarchical, concurrent state machines for behavior modeling and scenario control. *Fifth Annual Conference on AI, and Planning in High Autonomy Systems*, pages 36–42, 1994.

[59] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[60] Luke Dicken. HCSM: A Framework for Behavior and Scenario Control in Virtual Environments, 2009. URL http://aigamedev.com/open/review/hcsm-concurrent-state-machine/.

[61] Bjoern Knafla. Introduction to Behavior Trees, 2011. URL http://www.altdevblogaday.com/2011/02/24/introduction-to-behavior-trees/.

[62] Damian Isla. Managing Complexity in the Halo 2 AI System. In *Game Developers Conference*, 2005.

[63] Michael Dawe, Steve Gargolinski, Luke Dicken, Troy Humphreys, and Dave Mark. Behavior Selection Algorithms. In *Game AI Pro*, pages 47–60. 2013.

[64] Brian Schwab, Alex Champandard, Luke Dicken, Rez Graham, Kevin Dill, Michael Dawe, Ben Sunshine-Hill, and Dave Mark. Turing Tantrums - AI Developers Rant. In *Game Developers Conference*, 2014.

[65] Phillipa Avery, Sushil Louis, and Benjamin Avery. Evolving Coordinated Spatial Tactics for Autonomous Entities using Influence Maps. *IEEE Symposium on Computational Intelligence and Games*, 2009.

[66] Paul Tozour. Influence Mapping. In Mark DeLoura, editor, *Game Programming Gems vol. 2*, pages 287–297. Charles River Media, 2001.

[67] Ian Millington and John Funge. *Artificial Intelligence for Games*. CRC Press, 2nd edition, 2009.

[68] Liam Kelly, Luke Dicken, and John Levine. StrathPac - An Automated Player for Ms. Pac-Man. In *IEEE Symposium on Computational Intelligence and Games, Competition Track*, 2009.

[69] Dave Mark. *Behavioral Mathematics for Game AI*. Charles River Media, 2009.

[70] Luke Dicken. Redshirt AI Developer Diary, 2012. URL `http://mitu.nu/2012/06/20/guest-post-luke-dicken-on-redshirt-ai/`.

[71] Dave Mark and Kevin Dill. The Dark Art of Mathematical Modeling. In *Game Developers Conference*, 2012.

[72] Chris Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995. ISBN 0198538642.

[73] T Thompson, J Levine, and G Hayes. EvoTanks: Co-Evolutionary Development of Game-Playing Agents. *IEEE Symposium on Computational Intelligence and Games*, 2007.

[74] Richard Evans. The Future of AI in Games: A Personal View. *Game Developer Magazine*, pages 46–49, aug 2001.

[75] Steven Woodcock. Game AI: The State of the Industry 2000 - 2001. *Game Developer Magazine*, pages 36–44, aug 2001.

[76] Michael Robbins. Using Neural Networks to Control Agent Threat Response. In *Game AI Pro*, pages 391–399. 2013.

[77] Alex Champandard. This Year in Game AI: Analysis, Trends from 2010 and Predictions for 2011 — AiGameDev.com, 2011. URL `http://aigamedev.com/open/editorial/2010-retrospective/`.

[78] Troy Humphreys. Exploring HTN Planners through Example. In *Game AI Pro*, pages 149–167. 2013.

[79] Remco Straatman, Tim Verweij, Alex Champandard, Robert Morcus, and Hylke Kleve. Hierarchical AI for Multiplayer Bots in Killzone 3. In *Game AI Pro*, pages 377–390. 2013.

[80] Guy Van Den Broeck, Kurt Driessens, and Jan Ramon. Monte-Carlo Tree Search in Poker using Expected Reward Distributions. pages 1–15.

[81] Nick Birnie. *Opponent Modelling in Poker*. PhD thesis, University of Strathclyde, 2010.

[82] Luke Dicken, Nicky Johnstone, John Levine, and Phil Rogers. SPREE : The Strathclyde Poker Research Environment. In *Proc. AISB AI and Games Symposium2*, 2011.

[83] O. Macindoe, L.P. Kaelbling, and T. Lozano-Perez. POMCoP: Belief Space Planning for Sidekicks in Cooperative Games. In *Proceedings of the 8th Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[84] E Gat. On Three-Layer Architectures. *Artificial Intelligence and Mobile Robots*, 1997.

[85] C McGann, F Py, K Rajan, H Thomas, and R .... T-rex: A model-based architecture for auv control. *Proceedings of ICAPS Workshop*, 2007.

[86] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press; 3rd Revised edition edition, 2009. ISBN 0262033844.

[87] Edsger Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerichse Mathematik*, pages 279–271, 1959.

[88] KV Mardia, JM Bibby, and JT Kent. *Multivariate Analysis.* Academic Press, 1979.

[89] L Lamport, R Shostak, and M Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 1982.

[90] Melanie Mitchell. *An Introduction to Genetic Algorithms.* 1998.

[91] Mobile game developer survey leans heavily toward iOS, Unity, 2012. URL `http://www.gamasutra.com/view/news/169846/Mobile_game_developer_survey_leans_heavily_toward_iOS_Unity.php`.

[92] Mike Rose. Obsidian's Project Eternity is the most-backed video game Kickstarter, 2012. URL `http://www.gamasutra.com/view/news/179461/Obsidians_Project_Eternity_is_the_mostbacked_video_game_Kickstarter.php`.

[93] Aung Sithu Kyaw, Clifford Peters, and Thet Naing Swe. *Unity 4.x Game AI Programming*. Packt Publishing, 2013.

[94] Daniel Kline. The AI Director in Dark Spore. In *Paris Game AI Conference*, 2011.

[95] Dean Takahashi. Zynga launches Mountain Goat Mountain, its latest game in its climb into mobile, 2015. URL `http://venturebeat.com/2015/07/16/zynga-launches-mountain-goat-mountain-mobile-game/`.

[96] Jaymes Carter. Mountain Goat Mountain by Zynga is quite enjoyable, 2015. URL `http://www.droidgamers.com/index.php/game-reviews/9535-game-review-mountain-goat-mountain-by-zynga-is-quite-enjoyable`.

[97] Mountain Goat Mountain - Google Play Store Entry, 2015. URL `https://play.google.com/store/apps/details?id=com.zynga.mountaingoat&hl=en`.

[98] Mountain Goat Mountain - iOS App Store Entry, 2015. URL `https://itunes.apple.com/us/app/mountain-goat-mountain/id979415701?mt=8`.

[99] Elizabeth Knowles, editor. *The Oxford Dictionary of Phrase and Fable*. 2 edition, 2014.

[100] Nassim Nicholas Taleb. *The Black Swan: The Impact of the Highly Improbable*. Penguin, 2010. ISBN 978-0-1410-3459-1.

[101] Donald Rumsfeld. DoD Briefing - February 12th 2002. URL `http://www.defense.gov/transcripts/transcript.aspx?transcriptid=2636`.

[102] Daniel Brewer, Alex Cheng, Aleissia Laidacker, and Richard Dumas. AI Postmortems: Assassin's Creed III, XCOM: Enemy Unknown, and Warframe. In *Game Developers Conference*, 2014.

[103] Luke Dicken and Heather Decker. Procedural Processes - Lessons Learnt From Automated Content Generation in "Easy Money". In *NoShow Conference*, 2012.