Evaluating the Effects on Monte Carlo Tree Search of Predicting Co-operative Agent Behaviour

Joseph Walton-Rivers

Computer Science and Electronic Engineering

University of Essex

This thesis submitted for the degree of

Doctor of Philosophy

January 2022

Acknowledgements

I would like to take this opportunity to thank those who has made this thesis possible.

My supervisor, Prof. Richard Bartle, who's guidance has been invaluable though out my PhD. My frequent collaborator, Piers for his assistance with code and ideas. I would also like to thank the CSEE department and the IGGI CDT for their support over this thesis.

Finally, I'd like to thank my family for their support during my PhD.

Abstract

This thesis explores the effects of including an agent-modelling strategy into Monte-Carlo Tree Search. This is to explore how the effects of such modelling might be used to increase the performance of agents in co-operative environments such as games.

The research is conducted using two applications. The first is a co-operative 2-player puzzle game in which a perfect model outperforms an agent that makes the assumption the other agent plays randomly. The second application is the partially observable co-operative card game *Hanabi*, in which the predictor variant is able to outperform both a standard variant of Monte Carlo Tree Search (MCTS) and version that assumes a fixed-strategy for the paired agents. This thesis also investigates a technique for learning player strategies off-line based on saved game logs for use in modelling.

Table of contents

List of figures

List of tables

Part I

Context

Chapter 1

Introduction

Games often make use of Artifical Intelligence (AI) techniques to populate their worlds with characters with which the player can interact. These can take on the role of background characters (Non-Player Characters (NPCs)) or character that interact directly as if they were another player (bots) (Yannakakis and Togelius, 2018). AI plays an important role in player enjoyment of the game. A good AI system can help immerse the player in the world, while a bad one can yank the player out of the experience.

One important aspect of how the players perceive the AI in the game is how the AI interacts with the player and each other. While a great deal of research has gone into examining the role of AI systems that compete with the player, comparatively little work has researched how to develop AI systems that can co-operate with the player and each other.

From the player's perspective, the desire for socially capable AI for games has been well documented within the academic literature. In role-playing games, a common complaint among players surveyed is that the AI party members do the 'wrong'

thing in combat and cannot be trusted to support the player (Mäkelä and Schmidt, 2020). Researchers also note that the games industry has expressed a desire to create characters that players can form emotional connections with by creating more believable characters (Bopp *et al.*, 2019).

Compared to academic research, where state of the art AI techniques are being used to play to a grand master level, most game AI is comparatively simple. This thesis's motivation comes from the desire to see the state of the art AI techniques integrated into games. The ultimate, long-term objective is to provide intelligent, socially-capable characters that can fulfil the role of inhabitants in the game world and support players as companions. Having such AIs would allow for more complex emergent experiences to take place without the need to hand-script complex character behaviours. Traditionally, start of the art AI algorithms have shown great strength within single and non-cooperative multi-player games. Human/AI collaboration is seen as an emergent area within the field of AI research[1] (Risi and Preuss, 2020). To meet these long term goals, the agents also need to be shown to work well on co-operative or partially co-operative games. This thesis will not aim to create these fully socially-capable AI agents - this goal is far too large for a single thesis. Instead, it explores if it is possible to use modelling techniques in co-operative games to allow AIs to work co-operatively and explore if such agents are viable in simplistic environments and what limitations are present.

Within recent decades, the development of AI algorithms for game-playing has shifted from classical combinatorial games such as *Go* to video games. This background will be discussed in section 2.5. There has also been a focus on development of algorithms capable of coping with the problems that video games present via test-beds and ai competitions, such as section 2.6. One algorithm that has showed

[1]This survey lists Hanabi as one of the next large areas of research, one of the topics in this thesis

particular promise in this field of research is Monte Carlo Tree Search (MCTS) and the modifications to make it suitable for different kinds of games. This algorithm is a combination of classical tree-search approaches that proved very effective in early games research and monte-carlo simulations. Another promising avenue for research is the inclusion of generic algorithm-based approaches. These approaches are nature-inspired and generate new candidate solutions by modifying existing ones. These are presented in more detail in section 2.4.

Although this research has shown a great deal of promise, there is still significant work to be done in this area. Most notably, work on co-operative games. There have been some attempt to incorporate such games into academic competitions, for example, the *Geometry Friends* competition which includes two-player co-operative puzzles, but also features real-time physics. Games which are closer to the General Video Game AI (GVG-AI) approach include the two-player version of the competition, which was developed during the time this research was conducted, and the two games presented in chapter 3.

It might be worth considering why co-operative games are a worthy area of study for AI research. After all, most of the existing work focuses on adversarial games. The requirements of modern video game AI provide one such motivation, and the desire of players to see better more capable AI in their games. To that end, this thesis will explore one small part of this area of research - focusing on the effects of modelling in simplistic games with the two algorithms previously mentioned.

1.1 AI with agent modelling

For humans, the ability to reason about how our behaviour will affect others and incorporate them into our plans comes naturally. Humans can easily reason about

others - they have *theory of mind* Premack and Woodruff, 1978. Within game playing research, there has been a large amount of work on adapting strategies to deal with opponents (see section 2.7). The term opponent modelling is often used for this style of task, however, as the games that will be examined in this thesis are not competitive, the term 'opponent' is misleading. Instead, the term 'agent modelling' would be more accurate. Recently, work has focused on creating agents that can also work in co-operative environments (Bard *et al.*, 2019). Both co-operative and competitive modelling will be discussed in section 2.7.

Although this is clearly not the same goal as creating socially believable AI, it is none-the-less an important consideration for creating such AI. As a result, this thesis will explore the effects of introducing such modelling into AI-based agents, with a long term view to the incorporation of such strategies into socially capable AI. This should be viewed as 'first-step' towards the goal of creating more socially believable agents, but not the creation of such agents.

Therefore, this thesis will aim to evaluate if giving an agent knowledge of other agents will provide a benefit for two co-operative games (one with simultaneous moves and one with partial observability), and if so, how does such modelling affect agent performance. Creating such a model can be tricky. There are many ways in which modelling has been applied in the past. One classical way is to assume the other player will play optimally. This can be assumption cannot be relied on for co-operative play, as game theoretic approaches will not always work. Indeed, one of the games used in this thesis is noted for having this property Bard *et al.*, 2019. For example, when multiple Nash Equilibria exist and its not clear which the other player will choose. Instead, for this thesis, paired agent models are assumed to be 'black box' policies which can be used to request actions from a given state (ie, agents are assumed to be allowed to sample from the paired agent policy for a given

state). For the first chapters of this thesis, the correct policy will be used and a full evaluation of the paired policy will be performed when the agent samples from the model. This limitation will be addressed chapter 8, where a learnt model based on game logs will be presented and evaluated.

1.1.1 Motivation

The beginning on this chapter presented an academic argument for why this topic is important. The idea of creating AI that can interact with the game world to create living, breathing experiences is not new. Although well outside the scope of this thesis, it provides a long term motivation for the work conducted within it. This has been expressed most succinctly by Bartle, 2004:

> *Imagine it!* Imagine a virtual world with thousands of virtual people living virtual lives — each with their own goals, their own relationships, the own existence. A living, breathing, self-sustaining creation! Doesn't that feel you with awe? Don't you want to *go* there? Don't you want to see what they'll do, and do it *with* them? Wouldn't that *truly* be a virtual world? (Bartle, 2004)

Players feel an emotional attachment to characters within games. Characters in games can help players feel a sense of connectedness (Kleinman *et al.*, 2021). There genres of games devoted to creating immersive game environments and interacting with the characters within them. In games such as *The Sims*, the gameplay revolves around the interactions with other characters. In games like *Rimworld* and *Prison Architect* gameplay and emergent stories result from the interaction between characters in the world, and there are whole gaming communities devoted to celebrating emergent properties of complex simulations such as those of *Dwarf Fortress*. In

games such as *Minecraft* the villagers help to give life to the world. There is a requirement for creating better characters for these worlds.

1.1.2 Choosing suitable games

To choose suitable games, it is necessary to consider ways in which players can co-operate within games. For this thesis, three types of games that feature co-operation were considered: team *vs* team, team *vs* environment (where the game provides opposition) and team *vs* system (where the game is passive, and the complexity comes from solving the puzzle).

In team *vs* team games, the game pits teams of players against each other (or an individual). In these games, players must co-operate and adapt their strategies to overcome the strategies of their opponents. Often within these games, players are given asymmetric abilities; for example, in *Team Fortress* players may choose a range of possible classes, each with unique abilities. Some of these provide support for the player's team (for example, a medic being able to heal players or an engineer providing the ability to create teleporters to allow their team-mates to return to the fight quicker), others create a disadvantage for the opposing team (for example, a spy's ability to hide amongst the enemy team) and others provide counters to the advantages mentioned in the previous point (for example, the ability to set fire to hidden spies, but not to your team-mates). The construction of the team and exploiting these features is a core part of these games' gameplay. Social deduction games would also fall into this category.

In team *vs* environment games, the team must overcome some threat provided by the game itself. In *Forbidden Desert* and *Pandemic* the players must adapt to changing conditions as the game develops. The players are playing 'against the

deck', which complicates their strategy in unforeseen ways. The players may have ways to counter these effects, but the primary source of complexity comes from the game's opposition itself. This opposition differs from *team* vs *team* games, as the game itself does not provide intelligent opposition to the players. The game's responses are often based on chance or fixed based on a set of predefined rules and do not adapt over time to the player's actions. As the game's behaviour is more predictable, there is no need to create complex models of the opposing teams behaviour. In addition, the behaviour is often set out within the rules of the game. Agents that act using a world model could therefore reason about the behaviour as part of the game world itself. This style of games emphasises strategic play as well as co-operation.

In team *vs* system games, the behaviour of the game is mostly fixed. The game is more of a puzzle, and the players are co-operating to come up with a strategy to defeat the puzzle. This differs from single-player puzzle games as the players will need to synchronise their behaviours and ensure their plans to not interfere with each other. Most classical multi-agent planning domains would fit into this category. The games presented in this thesis also fall into this category. In *Tiny Co-Op*, the world is reactive to the players' behaviour, and the complexity comes from co-coordinating the actions of the players. *Hanabi* presents a more interesting challenge for the agents, the game features limited uncertainty which must be planned around, but the game itself presents to an active challenge to the player - it could be argued that the nature of the game itself (game length limited by the deck) could move it into the *vs* environment category, but the gameplay largely focuses around the other players rather than the game's 'opposition'. This game style focuses on developing a strategy that can work given the game's rules but mainly on the players rather than the game.

There are reasons why all three of these types of games pose interesting problems for AI research. In recent years, team vs team games have shown to be of particular interest to academic researchersBaker *et al.*, 2019. This thesis will mostly focus on the third kind mentioned, team *vs* system. This is because, while both of the other categories present interesting research opportunities (and would make excellent areas for further study), given available time and scope, only one area can be researched. The third category is 'purer' because the complicating factors mentioned above are not present (no opponents to model and simplistic mechanics). Since this work was started, other researchers have started to explore the game in the other two categories, which presents exciting opportunities for future avenues of AI research.

1.2 Overview

This thesis will aim to answer the following question:

Does the introduction of prediction improve the performance of MCTS in co-operative games?

One of the most popular algorithms for creating game playing agents is MCTS (which will be further discussed in the next chapter). In this thesis, it is shown that by introducing the notion of a predictor agent (that is, an agent which accounts for the observed behaviour of other agents), not only can MCTS be outperformed in co-operative games, but the agent is also more believable when judged by its actions. At present, there is little research into how an incorrect or inaccurate model can affect agent performance Albrecht and Stone, 2018

The first experiment performed (chapter 5) provides a baseline for agent performance within a co-operative puzzle game. The game is extended in chapter 6 to feature structured communication action. Two different styles of communications are investigated. The results of this investigation are used to explore the effects of introducing modelling into the search tree for a modified MCTS agent.

Although the results presented for this agent are promising in the sample game, it only supports two players. Its suitability limits it as it is mostly puzzle-based (there are limited 'correct' solutions). A more free-form environment which a larger variety of possible strategies is needed to evaluate the effectiveness of modelling.

The card game *Hanabi* shows promise as a possible environment for testing the modelling of other agents by game-playing agents, for example, it is being used by Google Deepmind as a suitable test environment (Bard *et al.*, 2019) in work that references experiments undertaken in this thesis (Walton-Rivers *et al.*, 2017). Section 3.4 describes an implementation of the game that will be used for the rest of the thesis's experiments. Chapter 7 investigates the modified agent's effectiveness as part of the game. The results presented in section 7.6.1 show that the agent has good performance compared to the implementation of MCTS that does not use modelling.

The possible reasons for this increase are examined in the rest of the chapter. Firstly, different values for the exploration constant are examined. These show that the agent performs better with much lower values for the exploration constant than those used in the original experiment, but this cannot explain the difference in performance between Infomation Set Monte Carlo Tree Search (IS-MCTS) (Cowling *et al.*, 2012a) and the predictor variant. Secondly, replacing the random roll-outs used by IS-MCTS with a policy-based roll-out is examined in section 7.8. This

shows that using an intelligent roll-out policy can improve the performance of IS-MCTS, but the predictor agent still outperforms it. In addition, the best policy also exhibits similar behaviour to the rule-based agents. Finally, a variant of the game presented in section 7.9 which removes the partial observability and the need for communication shows that the IS-MCTS agent is capable of playing the game near perfectly when the partial observability (and thus the need for communication) is removed.

Finally, a limitation of the previous chapters' models is addressed – the assumption that a perfect model of the paired agent's behaviour is available. Firstly, the effect of noise on agent performance is explored in section 8.1. This experiment is used as a target value for the accuracy of a model created by using a genetic algorithm from previously played games (section 8.2). The models are then evaluated by comparing agent performance when using the perfect model and the trained models (section 8.3). This shows a slight drop in performance when using the trained models, but the agent still performs well.

1.3 Contributions

This section outlines the contributions this thesis makes. The most important of these contributions is the second: creating a game-playing agent capable of taking into account its paired agent's decision-making processes.

1. The creation of a co-operative puzzle game with communication actions, to evaluate the performance of co-operative agents before the availability of the 2-player general video game AI track. This is combined with an analysis of the effects of communication actions on MCTS in this game.

2. The evaluation of a modification of the MCTS algorithm which takes into account the agent decision-making process as well as the current value of the state. The need for such research was referenced as an important area of study within the general video game playing 2-player competition literature.

3. The design and development of a framework for the creation of *Hanabi*-playing agents, which was also used for this research, was also used for the *Hanabi* competition held at CIG 2018 and COG 2019. This competition helped lay the groundwork for additional research in this area and the work relating to the framework has been used and replicated in additional studies on *Hanabi*.

4. The creation of learnt models and the evaluation of their effect on agent performance in the card game *Hanabi*. This provides the basis for an approach for evaluating if a learnt model.

Table 1.1 Where each of the contributions are addressed in this thesis

	1	**2**	**3**	**4**
Chapter 5	X			
Chapter 6	X	X		
Chapter 7		X	X	
Chapter 8				X

The co-operative puzzle game *Tiny Co-Op* presents a grid world-like scenario for testing the performance of MCTS in a simple co-operative game. The analysis of the agent's performance in this game helps to highlight some possible limitations that can be explored further. One of these limitations, the lack of ability to reason about non-optimal behaviour, is the basis of the second contribution. This limitation can be overcome by modifying the agent to take into account its partner's behaviour.

Thirdly, the creation of the *Hanabi* framework (and its subsequent public release under a Free Software licence, GNU/GPL v3) has allowed other researchers to

extend the work presented in this thesis (see section 2.8). It has also allowed the *Hanabi* competition to run, which has resulted in researchers and students developing agents to play the game and conduct research.

Finally, the techniques presented in chapter 8 show it is possible to use game traces to generate player behaviour models. This allows for faster execution than the versions in chapter 7, and allows for modelling of more complex agents, which would be too costly to compute otherwise. While this is not a perfect solution, and other approaches now exist for accomplishing this for *Hanabi*, it is none-the-less an important consideration.

1.4 Organisation

This thesis is comprised of 3 parts.

Part 1

Part I provides background information on this thesis. Chapter 1 sets out motivation and contributions. Chapter 2 provides an overview of the existing research and algorithms which are used in the later chapters. This part concludes with the games (chapter 3) and algorithms (chapter 4) that will be used for the rest of this thesis.

Part 2

The main contributions of the thesis are presented in part II. Chapter 5 outlines the work presented as part of the *Tiny Co-Op* environment and provides some analysis of the agents which were evaluated as part of the work. Chapter 6 outlines how this work was extended to include communication actions and their effects on the agent's performance. The chapter also introduces a variant of MCTS which is

capable of reasoning about the consequences of the other agent's responses to its actions and its effect on the game state. Chapter 7 presents work relating to the card game *Hanabi* and present agent performance. chapter 6 explores in more detail, which looks at the agents' parameters and variants. Lastly, chapter 8 explores the concept of learning models of agent behaviour based on game logs.

Part 3

Finally, part III presents conclusions and outlines future work from this thesis.

Chapter 2

Background

This chapter will provide an overview of the existing research into computational intelligence techniques used to create Artifical Intelligence (AI) in games. It will also expand on the long-term motivations driving this work, which while out of scope for this thesis, provide context for how this thesis came to exist. Following on from this, an overview of the inspirations and existing work that influenced the design of the experiments and choice of algorithms will be presented. Namely, academic competitions such as the general video game competition and the algorithms which have been used within it. The algorithms which will be used for the remainder of the thesis will be presented in more detail in chapter 4. Following on from this, a brief overview of algorithms used for AI research will be presented. Finally, existing research into one of the games used in this thesis, *Hanabi* will be presented.

2.1 Artificial Intelligence in Games

There are many ways in which AI techniques can be applied to games (Rabin, 2013; Rabin, 2015; Rabin, 2017). These techniques are pervasive within the games sector,

ranging from data mining of player behaviour to techniques to generate content for the players to explore (Yannakakis, 2012). The role of AI in games can give rise to new an interesting player experiences (Treanor *et al.*, 2015). However, one the most visible ways in which AI is used within games is to control the characters that inhabit the game world. These can be grouped into two broad categories: AIs *pretending* to be players (bots) and AIs *representing* characters in the game world (Non-Player Characters (NPCs)).

2.1.1 Non-Player characters

NPCs represent characters in the game world which are not meant to be perceived as human-controlled. NPCs in games take on a number of roles (Bartle, 2004):

- Buy, sell and make stuff
- Provide Services
- Guard Places
- Get killed for loot
- Dispense Quests (or clues for other NPCs' quests)
- Supply background information (history, lore, cultural attitudes)
- Do stuff for players
- Make the place look busy

These types were expanded and arranged into a hierarchy by (Warpefelt, 2016):

Table 2.1 The organisation of NPC types, created by Warpefelt, 2016

Metatype	Type	Subtype
Functions	Vendor Services Questgiver	
Adversies	Enemy Opponent	Boss Manipulator
Friends	Sidekick Ally Companion Pet Minion	
Providers	Storyteller Loot provider	

The purpose of these two groups is very different. Whereas bots are primarily concerned with the control of characters which are active in the foreground, NPC take on more of a background role.

2.2 Believability in games

Believability is a somewhat overloaded term. Within AI literature, the term is used to refer to creating agents which are good at mimicking human behaviours and of agents which are realistic portrayals of the entities which they are meant to represent. Researchers have attempted to disambiguate the term by splitting the two definitions into distinct categories – (Livingstone, 2006) proposes 'Player AI' and 'NPC AI' for the two groups respectively. This distinction is also present in (Togelius *et al.*, 2012) which refers to the two groups as character and player believability.

Within the first category, the creation of agents which are better imitators of human players, the focus has mostly been to create agents for human players to play against. Most of the early work in this domain was focused on the creation of bots for first person shooters (Waveren, 2001; Choi *et al.*, 2007) and learning techniques (Nashed and Davis, 2011). For players, these bots are mostly there to 'fill up numbers' and players find them 'predictable, easy to defeat and boring to play against' (Sweetser *et al.*, 2003).

The second category, focuses on approaches where the objective is to create characters which more accurately reflect the characters in the world that they are trying to portray. (Bartle, 2004) argues that creating a world inhabited by believable NPCs can help make the world more immersive.

2.2.1 Believable Agents

The study of creating believable characters in games is the study of creating believable agents. This section will present existing work in this area. As the motivation for this work is the creation of socially-believable agents, this section presents existing work in the creation of such agents.

Early work on creating believable characters for games comes from the Oz project. This project provides both a model for integrating emotion into believable characters (Bates, 1994), and provides an early definition of believable agents (Loyall, 1997). The *Façade* project (Mateas and Stern, 2003) created agents which use an extension of the reactive planning language developed by the Oz project as the basis for creating believable agents in their interactive drama. In both of these projects, the behaviour of the agents is defined ahead of time by the author.

2.2.2 Requirements for believable agents

There have been multiple attempts to create definitions of believable characters, (Loyall, 1997) draws on character arts and AI work to create his definition, combining both fields. His criteria for creating believable agents are:

- Personality
- Emotion
- Self Motivation
- Change
- Social Relationships
- Consistency of Expression
- The illusion of life
  - Appearance of goals
  - Concurrent pursuit of goals and parallel action
  - Reactive and Responsive
  - Situated
  - Resource Bounded
  - Exist in a social context
  - Broadly Capable
  - Well Integrated

Another approach is the PAR AI Believability criteria(Livingstone, 2006), based on how the agents act with the world and each other (table 2.2).

Table 2.2 The PAR AI Believability Criteria

Plan	Demonstrate some degree of strategic/tactical planning Be able to co-ordinate actions with player/other AI Not attempt a previous, failed, plan or action
Act	Act with human-like reaction times and abilities
React	React to players' presence and actions appropriately React to changes in their local environment React to presence of foes and allies

Both of these definitions contain both components which relate to the way in which the agents relate to each other, and their ability to solve problems in the game world. Game AI research (which will be presented in sections 2.3 and 2.4) has largely focused on techniques for creating better game-playing agents but doesn't have much focus on the interaction between agents. This focus of this thesis will be to attempt to integrate social capability into the existing state-of-the-art game playing algorithms to create more socially-capable agents.

It is worth noting that creating intelligent agents is not necessarily required for creating believable characters. (Reilly, 1996) argues that creating believable characters does not neccerialy require creating intelligent characters. For example, if the character is meant to be unintelligent, then giving it superhuman problem-solving capabilities would harm the believability of the character. However, creating characters that are largely incapable is also undesirable. The 'eliza effect' (Weizenbaum, 1966) shows that characters will attribute intelligence to relatively simple processes providing the character is not actively working to destroy the illusion. Common game-playing techniques feature parameters which can be tuned to limit or enhance

the problem-solving capabilities of the agents. This provides a relatively easy way to tune the capabilities of the agents.

A number of researchers have attempted to explore player perceptions of believability in games. Interviews conducted with players indicate that they feel that NPCs feel too scripted and lacked decision-making capabilities (Johansson and Verhagen, 2014). A survey of Role Playing Games (RPG) players showed that players found the social aspects of believability in RPGs lacking (Afonso and Prada, 2009). These results show that there is a demand from players for increasing the social capabilities of game-playing agents. This lacking social capability was the motivation for the work undertaken in this thesis. The social capabilities of the AI in the games should be taken into account when designing such agents. This thesis will aim to explore two co-operative games and incorporate an agent model into the algorithms.

2.3 Existing AI Algorithms

This section describes some common algorithms used within game AI research; the algorithms which will be used in this thesis will be presented in more depth in chapter 4.

2.3.1 Behaviour Trees

Behaviour trees are a technique for creating AI for games. These have been used in multiple games, most notably in *Halo 2* (Isla, 2005). These are fairly simplistic systems which are rule-based and are popular within commercial games as they are fast to execute.

The technique involves executing nodes which consist of behaviours and queries for the game state. These are combined using a number of operators, including sequence (which succeeds if all child nodes succeed) and select (which succeeds when any child node is successful). These allow for the creation of simplistic AI behaviours, but they will be quite repetitive. To combat this, the order of actions attempted can be randomised to offer variety to the AI behaviour.

Manually creating behaviour trees can be very time-consuming. There have been attempts to evolve trees automatically for the commercial game *Defcon* (Lim *et al.*, 2010) and the Mario AI competition (Pérez-Liébana *et al.*, 2011).

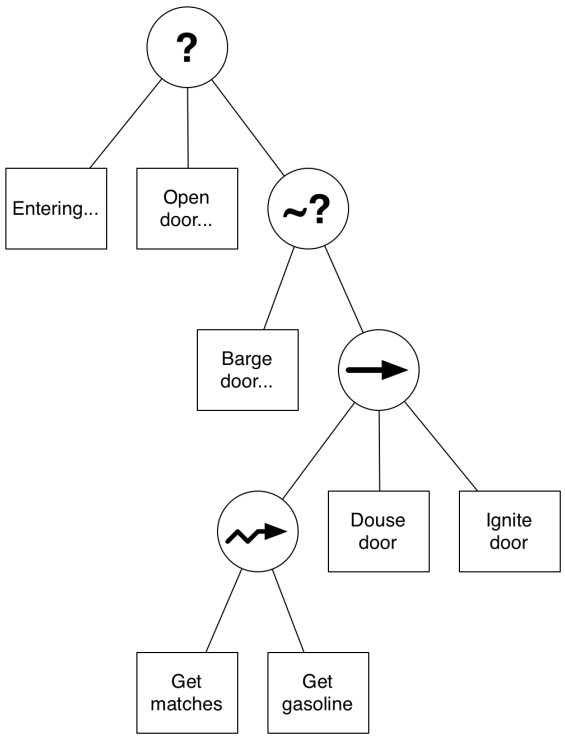


Fig. 2.1 A simple behaviour tree for a character within a game (Rabin, 2016)

A behaviour tree (fig. 2.1) can consist of many types of nodes. As well as deterministic select and sequence operators, randomised sequence and selection operators that allow the evaluation of child nodes in any order are used to help avoid the

repetitiveness of agent behaviours. In this example, the order in which 'barge door' and the routine for burning down the door is unimportant so a random selection node is used. Likewise a random sequence node can be used for getting matches and a fuel source. This is not possible with the douse door and ignite door actions which is why both a combination of randomised and deterministic versions of these operators is important.

Although behaviour trees have found widespread commercial success, they are unable to deal with unexpected scenarios as they are created off-line and don't have any on-line problem-solving capacity. Using search-based techniques such as Monte Carlo Tree Search (MCTS) allows for agents to plan dynamically at runtime; this means that the agent can deal with scenarios that were not foreseen in advance. If on-line adaptation to behaviours is required, then an approach that is based on goal-directed behaviours such as planning-based approaches is better suited.

2.3.2 Planning Techniques

Classical planning approaches have been modified for use within games. There have been a number of different approaches to planning within game environments (Neufeld *et al.*, 2017b). Two of the most prevalent techniques are Goal Orentied Action Planning (GOAP) and Hierarchical Task Network (HTN).

GOAP (Orkin, 2006) was first developed for use in *F.E.A.R*. It is based on The Standford Research Institute Problem Solver (STRIPS), an early planning technique. The implementation differs from STRIPS in the representation used for the planning domain. STRIPS uses two lists to represent the effects of actions, one of facts (atoms) that become true after an action is executed (an add list) and another of things that become false after the action has executed (a delete list). Instead, GOAP uses a

fixed-size vector to represent the world state, allowing parts of the vector to be modified as results of the actions.

HTN is an extension of classical planning (Nau *et al.*, 1999). As well as using *primitive* actions, similar to those in GOAP it also features *compound* tasks. These are tasks which describe how they can be broken down into sub-tasks and ultimately to primitive tasks. There may be multiple decompositions, with different decompositions being restricted by conditions. Instead of a goal state, the planner is provided with a high-level task that should be decomposed (for example, one such task might be 'guard building'). This is recursively broken down into subtasks, until all tasks are primitive tasks.

Within games research, there have been a number of attempts to adapt HTNs for use in games. One major limitation of game domains is the lack of time that is available per frame to plan. Squadsmart (Gorniak and Davis, 2007) overcomes this limitation by modifying the planner to be able to pause planning if it is consuming too much of the available system resources in a single frame. (Kelly *et al.*, 2008) use HTN to generate plans off-line, then store the resulting plans as game scripts to avoid the computational requirements associated with real-time planning; they applied these to the video game *Elder Scrolls: Oblivion* and were successfully able to generate plans. The authors claim that the results seemed more realistic because they had more variety in their behaviours. One possible limitation of this approach is that the agents will be unable to deal with unexpected scenarios (those which do not possess a pre-generated script). Soemers and Winands, 2016 applies HTN to Unreal Tournament 2004 and notes that in order to deal with the actions of other players full replanning is often used. They integrate plan reuse into HTN to speed up this replanning step.

Neufeld *et al.*, 2017a used HTN planning to create a controller for a fighting game, a highly dynamic environment which requires quick decisions on the part of the agent. The agent was able to beat the four agents it was evaluated against, beating MCTS in 165 out of 300 games.

2.3.3 Monte Carlo Tree Search

MCTS is a tree-search technique which was first proposed by Coulom, 2006. The algorithm, and multiple variations thereof, have been applied extensively within the game-playing research community (Browne *et al.*, 2012). The algorithm asymmetrically expands its search tree in iterations, attempting to minimise regret. The algorithm is *anytime* which means that at any given point it can be stopped and the current best solution can be returned.

The algorithm makes use of a *forward model*. This provides access to a reward for the agent at a given time step t. The agent has access to a set of actions $A_1..A_n$ which can be used with the state transition function (a function mapping state s to a successor state s' when applying an action a). This allows the agent to simulate the effect of actions without needing to perform them in the game directly. Access to this model is the primary difference between planning problems and reinforcement learning problems.

The algorithm has seen widespread adoption in the general game playing community and is one of two techniques which have worked extremely well for General Video Game AI (GVG-AI) (which will be discussed later).The details of the algorithm will be presented in section 4.2.

2.4 Games in Academia

This section outlines what a game is from an AI research perspective. It will then provide an overview of existing work in the area, with a few to outlining some key characteristics of games from an AI perspective and why they are relevant for this thesis.

2.4.1 Markov Decision Process

A Markov decision process (MDP) (Puterman, 1990) is a 4-tuple $\langle S, A, P, R \rangle$ that can be used to model decision processes. Although a game-theoretical approach might be more useful for two and multi-player games, MDPs feature prominently within the machine learning (and therefore monte-carlo) literature.

- S is the set of states
- A is the set of possible actions
- $P^a_{ss'}$ Is the probably of action a in state s will lead to state s'
- R^a_s is the immediate reward from performing action a in state s

For a state to be considered Markov, it should be independent of the history (it should only depend on the current state). A policy (π) is a mapping from states to actions. Specifying which action will be taken within a given state. The objective is to find the policy that will generate the highest expected reward. If the state transition function and rewards are known, then the optimal policy can be calculated. However, often this is not the case. Instead, the MDP can be sampled in order to estimate the value of a given state.

Partially Observable Markov Decision Processes

In many problems, the world is not fully observable. Instead the agent receives an observation instead of observing the new state (s‘) directly, as in an MDP. This observation is dependent on the prior state (s) and the action taken (a). The distribution of these observations are described by, O a function that describes the probability of receiving an observation for a given state-action combination ($O(s,o)$).

As well as many real-world problems such as robotics, partial observability is a feature of many games. For example, card games like *Poker* and *Hanabi*. This is also a feature of many co-operative board games such as *Forbidden Desert*.

2.4.2 Game Definition

Another consideration is for the case of multiple interacting agents. Within the mathematical literature, this is the area of *game theory*. A game in this context is a set of rules that can be described as an n-tuple $\langle S, S_T, n, A, F, R, p \rangle$.

Where:

S	Is the state of states, with S_0 being the initial state.
S_T	Is the set of terminal states (states which result in the end of a game).
$n \in \mathbb{N}$	is the number of players
f	is the state transition function
R	is the utility (reward) function
p	is the player about to act in each state

Within the context of a game, the game starts in an initial state (s_0) and progresses over time (t) until some terminal state is reached. Players choose actions from the possible action set, according to some policy, π. The state transition function maps states and actions to a successor state s‘. The state transition function can

be deterministic (the same same state-action pair will always result in the same successor state), or stochastic (the successor state is chosen from a set of possible successor states with probability P). The game continues until it reaches a terminal state (S_T).

Deterministic vs Stochastic Games

In deterministic games, the executing the same action in the same state always generates the same result (ie, $f(s,a,s') = 1$). This is the case for many classical board games such as Chess and Go. In these games the effect of an action in a given state is known the first time it is evaluated.

In contrast, a stochastic game, there is more than one possible outcome when executing a given action in a given state. If the probabilities are known, it is possible to calculate the expected reward by multiplying the obtained reward by the probability of ending up in that state. If the probability distribution is not known, it can be calculated based on observations.

If the game is fully observable, the observations will be of the full state, and therefore can be used to calculate the probabilities of the state transition function. However, in partially observable games, information is hidden from the agents. They instead are only given observations about the state rather than the whole state. This is the same case described above in the POMDP section, in a multi-player environment. For the purposes of this thesis, lack of knowledge of the other agent's policies is not considered partially observable. This will be discussed separately later. Instead, partial observability is not having knowledge of the game state itself.

Utility Function

The utility function allows the agent to evaluate how valuable a state is from its current perspective. The reward structure can very greatly between games. In games where the reward for all players sum to 1, the game is referred to as zero sum. In this case, the players are in competition (what is good for one player is bad for another). This is the case for most classical board games such as Go, and Chess. The goal for a given rational agent is to attempt to maximise it's cumulative reward. Rewards can be immediate (the reward is received when an action is taken) or delayed (a reward is received at a later point).

Players

A game can have one or more players, and the order in which these players act varies according to the game rules.

Sequential vs Simultaneous Moves

In alternating or sequential move games players take turns one after another. Each move can be modelled as a branch on a tree structure originating from the initial state. Each node in the tree represents a game state that occurs after a player has made their move. For fully observable games, a player therefore knows what move their opponent has made before responding to that move. Most classical board games such as Go and Chess are sequential move games.

In contrast, simultaneous move games require both players to make moves at the same time before the state can be forwarded. The player therefore does not have knowledge of the opponent's move before making their decision. Games such as rock-paper-scissors and many classical game theoretic games such as iiterated prisoner's dilemma fall into this category. In these games, it is possible to simplify

the game into an alternating move game, and use the techniques developed for alternating move games. This is the approach taken by Samothrakis *et al.*, 2010 in applying UCT to the game *Tron*.

2.4.3 Policies

A player's actions are defined by it's *policy*. This is the action or (or possible action for stochastic policies) it will take in any given state. This policy can be deterministic (for any given state, return the same value), or non-deterministic (return different actions with different probabilities).

If no player can receive a better reward for switching states, then the combination forms a Nash equilibrium. For some problems, such as generalised divide the dollar (Ashlock and Greenwood, 2016), are infinite Nash equlibira. Computing the Nash equilibria for real games is generally intractable (Browne *et al.*, 2012).

2.5 Games

Games have historically been a test-bed for AI research. In the 1950s and 60s, the focus was on classical board games (Russel and Norvig, 1995). These games are comparatively simple to modern video games. In this section an overview of the current state of the art with regards to classical games will be presented. Attention has moved on from this classical board games to video games. This shift will be presented in the following section.

2.5.1 Classical Board Games

The term 'classical board games' will be used in this section to describe two-player, perfect information, zero-sum (with ties), alternating-move games which are deter-

ministic. Examples of such games include, *Chess*, *Shogi*, *Go*, and *Checkers*. Within the literature, these games are referred to as *combinatorial games* (Browne *et al.*, 2012). Combinatorial games are sequential (turn-taking) games with perfect information.

Historically these games has proved challenging for AI due to their large state spaces. However, many of these classical games are playable to a human-competitive level. Checkers AI dates back to 1950s, was narrowly beaten in 1992 by a world champion, and was stronger than all human players by 1996 (Schaeffer *et al.*, 2007).

A lot of early game AI work was conducted on chess, it proving to be a suitable benchmark for tree search techniques (Feldmann, 1997). In 1997, the then raining chess world champion was defeated by IBM's deep blue Campbell *et al.*, 2002, however required the use specialised dedicated hardware. Shogi, an eastern combinatorial game which is similar to chess but which has a significantly larger state-space complexity was not as well researched, but good process was being made by 2001, and it was predicted that AI would be human competitive by 2010 (Iida *et al.*, 2002).

More recently, in 2015, Deepmind's alpha go defeated the raining three-time European *Go* champion 5-0, followed by beating Lee Sedol 4-1(Silver *et al.*, 2016). The algorithm used data derived from expert human players. Following on from this, a version based purely on self-play has proven to outperform this approach (Silver *et al.*, 2017). This was followed by a generalised version of this approach that could play a range of combinatorial games to a competitive level, including Go, Chess and Shogi (Silver *et al.*, 2018).

Games that include partial information, such as *Hanabi* would not fall into this category, nor would *Tiny Co-Op*, as it features simultaneous moves. Interestingly, some of the seminal work in this area, such as Berlekamp *et al.*, 2018, the games

considered are more restrictive as many of the games considered fail one of the conditions presented early in the text (for example, in chess a player a player that cannot move does not lose as it may be a stalemate).

In such games, players are expected to play until the game is either won, drawn or lost. Assuming this is used as the reward function (classical board games do not usually include a 'points' system), these games possess very sparse rewards, as a reward is only obtained at a terminal state.

The number of states in each ply of the search increases according to the number of possible moves that can be made in one 'turn'. This is known as the branching factor. The exact number of legal moves may very with the current game state, so an approximation based on the average branching factor is often used. An average branching factor of two means that on average, each state will generate two successor states. A branching factor of ten means 10 successor states per state on average and so on. The branching factor and number of states in these games also mean that an exhaustive search of the state space is often not feasible.

Generally, games with a larger branching factor and large state space result are more complex for AI-based agents to play. This can be observed in the history described above. Of the games presented, checkers has the smallest state space, followed by chess, then Shogi and finally Go. This limitation is especially notable for search-based approaches such as those described in section 2.7.1. As exploring the whole game tree is not possible, a number of techniques have been developed which help to mitigate this problem. The search can be guided by using heuristics.

One problem with this approach is that the heuristics developed are usually game-specific and as such good performance in one game does not equate to good performance in another (Levine *et al.*, 2013). In addition, because these agents make

use of game-specific information the approaches do not generalise well either to other games or to areas outside of games. General game playing (Genesereth *et al.*, 2005) is the concept of creating an agent that can play well in a range of games. As a result, the designer cannot rely on solutions that do not generalise well or are too specific to one game. The agent must show some aspect of general intelligence.

There have been attempts to create frameworks to allow the evaluation of agents on a range of classical board games. The general game playing (GGP) competition is one such framework, (Genesereth *et al.*, 2005). Games within the GGP competition are described using Game Description Langauge (GDL), a domain specific language which describes the game rules. The agents are presented with a game description to work from. The game rules are not known in advance but are presented to the player when the game starts. The agents are able to either analyse and act on this directly, or generate a model that can be used to make decisions. Although a useful resource for planning-based game problems, there are some limitations with this. Firstly, most current AI research utilizes agents written in Java (with GVG-AI, *ms Pacman vs Ghosts* Williams *et al.*, 2016b, *Hanabi* Walton-Rivers *et al.*, 2017 and Ludii Piette *et al.*, 2019 utilising Java), this means to make use of code from these projects, a Java implementation is preferable.

2.5.2 Video Games

Video games are often much more complex than classical board games and often have much stricter time requirements for AI than their classical board-game counterparts (Levine *et al.*, 2013). In addition, a full description of the game might not be available or too complicated to analyse. As a result approaches based on analysing the formal game definition are often not possible.

Just as in research into combinatorial games and classic board games. One early example of an environment intended for general video game playing is the Arcade learning environment (Bellemare *et al.*, 2013). This featured games which were available on the Atari 2600. Deep Q-Learning, an approach based on combining Q-Learning with neural networks was able to play many of these games to a human-competitive level (Mnih *et al.*, 2013). Other generalized environments have also been used for machine learning algorithms such as OpenAI Gym (Brockman *et al.*, 2016) provide a standardised interface to interact with many different environments, including games.

Although the Atari 2600 collection proved to be a useful benchmark for learning-based agents, it is not suitable for planning-based agents. The GVG-AI framework Pérez-Liébana *et al.*, 2016. This is the environment that the GVG-AI competition. The competition consists of many tracks, but one of the longest running ones is the single player planning track, which has been heavily used within research as a benchmark for testing enhancements to general video game playing algorithms Soemers *et al.*, 2016; Gaina *et al.*, 2017a; Kunanusont *et al.*, 2017. The competition and it's relevance to this thesis will be discussed later in section 2.6.

Although the GVG-AI framework is an extremely useful test-bed for game-playing ai research, for the objectives of this thesis it has limitations. Namely, many 'real-world' games are not only one or two player. Many are also not purely adversarial and require co-operation between the players. For an agent to work in these games it must not only understand how to *compete* with other players but also how to *co-operate* with them. This property is also desirable for large-scale simulations such as simulating a collection of NPCs within a game.

This need to co-operate with team members is also present in games that have gained attention from researchers. Games such as *Dota 2* which feature a mix of competitive and co-operative play has been proposed as a test-bed for novel AI research (Font and Mahlmann, 2018).

Modern board games also share many of the features which make video games challenging for AI research. They often feature partial observability and complex game rules in comparison to games like go and chess. A number of modern board games have also been proposed as suitable environments to test new AI techniques.

From a research perspective board games also feature the advantage that their rules are accessible and in a format that makes implementing them possible. This allows for multiple implementations and reproducibility. This is much more complicated with modern video games, as these are often 'living' games which are altered and adapted over time.

2.5.3 Partially observable games

A complicating factor which is not present in classical board games but is present in both modern video games and board games is partial observability. This means the agent cannot fully observe the state of the world and as a result will be making decisions in an environment that features uncertainty.

One famous card game that is heavily based around partial observability is poker. In poker the cards in the deck are hidden as are the cards present in the other players' hands. The success of the agent is not only based on the information available to it, but also the information that it cannot observe and the strategies of the other players in the game. There have been many modifications for MCTS for poker playing.

Ponsen *et al.*, 2010 integrated opponent models into MCTS which were a mixture of information gathered during the game and learnt prior information.

There has also been work on adapting MCTS to work correctly in partially observable environments more generally. Determinization is the act of making a partially observable state into a fully observable one. This is done multiple times to sample over possible states and recombining the statistics to allow the agent to act in the partially observable game. Other approaches include working over information sets rather than states and using a determination for action selection (Cowling *et al.*, 2012b).

2.5.4 Academic Competitions

Within academia, a common way to benchmark the performance of different algorithms is competitions (Togelius, 2014). Using competitions as benchmarks for AI algorithm performance is not something exclusive to games. These are also common in other AI related fields such as robotics (Anderson *et al.*, 2011), machine learning (Yang *et al.*, 2018) and classical planning Vallati *et al.*, 2015. There are many, many competitions that are used within academic research as benchmarks for games. This section will not provide an exhaustive list, but will provide some relevant examples from the recent past.

Most academic competitions are either aimed at creating AI for a specific genre or subset of games, or creating agents that can a wide range of games within a set of constraints (for example, all turn based, or all single player).

Commercial Games

Some competitions make use of commercial games (those which were originally designed for human players, are usually closed-source and offered for sale). These provide a 'real world' environment for the agents, but do have some potential pitfalls.

The StarCraft AI competition (Farooq *et al.*, 2016), presents a real-time challenge for AI agents. Although Starcraft supports games with more than two players, the competitions have focused on 1 vs 1 games (one bot vs another). As a result, the agents are in competition. The bots take on the role of an overarching commander who must issue orders to many individual units during the game. The competition has seen entrants from commercial AI research companies such as Google Deepmind, Facebook AI Research and Microsoft (Čertickỳ and Churchill, 2017). At present, the state of the art for this game is AlphaStar, which can play to a grandmaster level in the game (Vinyals *et al.*, 2019). This work shows great promise for Real-Time statergy games (RTS) research, but as the games are based around *Centralized planning* are out of scope for this thesis.

Hearthstone is a partially observable competitive card game. The competition is focused around creating agents for the collectable card game (CCG) genre (Dockhorn and Mostaghim, 2019). The competition is currently focusing on two tracks, one which players can use pre-made decks and the other which lets players use user generated decks. The competition features partial observability, and an opponent but does not offer any cooperative play.

The using an Application Programmable Interface (API) for (closed source) commercial games may be a limiting factor to running competitions. The Dota 2 competition pits controllers against the inbuilt AI in a one *vs* one match. The competition states

that it is desirable to see full team (5 *vs* 5) games played in the competition but this is not possible due to the limited API provided by the game. They also indicate that the competition would ideally be run using only the information that is present to the real game players (the communication wheel and map pings), but the framework also does not support communication to or from the built-in bots (Font and Mahlmann, 2018). This competition shows that for team vs team games it is desirable that the agents should be able to communicate and co-operate with each other and human players.

One of the longest running competitions is the *Ms Pacman competition* Lucas, 2007; Williams *et al.*, 2016b. Unlike the other competitions mentioned so far, although based on a commercially successful game the competition uses a re-implementation of the game. This avoids the pitfalls of relying on a closed-souse API, but does require more effort on the part of the competition organisers - they must rewrite a commercial game from scratch (which for most modern commercial games is infeasible).

Academic-First Competitions

As indicated by the *Dota2* competition. A commercial game is not always a suitable environment for running an academic competition. Another approach is to use a simplified version of the game genre which has been designed for AI research.

The MicroRTS framework (Ontañón *et al.*, 2018) is designed to be a simplified version of an RTS game. It aims to keep the 'interesting' parts of the problem while avoiding the potential engineering problems that come with interfacing with a commercial game engine. The game features a simplified version of an RTS game where players must collect resources, build units and fight their opponents. The game requires one controller to plan simultaneously for many units. This is an example of a centralized

planning problem where one controller is responsible for issuing orders to multiple agents.

Another competition which has taken this approach is the fighting AI competition (Lu *et al.*, 2013), which provides a simplified version of a 2D fighting game. The game complicates the usual planning process by delaying the execution of actions for a fixed number of frames before they are executed. The competition provides a sample MCTS controller.

With the exceptions of Pacman Vs Ghosts (which is player vs team), and (to a limited extent) the two player GVG-AI competition (which features a mixture of competitive and co-operative games) almost all of the current roster of academic games are competitive. A few competitions have aimed to tackle this lack of co-operative play. Firstly, the geometry friends competition Prada *et al.*, 2015 features co-operative puzzle solving in a continuous physics environment and the *Hanabi* competition (based off the work presented in chapter 7) provided the opportunity to explore co-operative behaviours in a partially observable turn-based card game.

2.6 General Video Game Competition

As discussed previously, competitions offer a good test-bed for creating AI controllers for games. One of the most prolific competitions of recent games research is the GVG-AI competition. The games within the framework are turn-based games which feature arcade-style controls (directional and an optional use/fire key). The two best performing approaches in both the single and two-player tracks are Genetic Algorithm (GA) and MCTS. This has influenced the choice of algorithms used in this thesis (and that will be presented in chapter 4).

The first iteration of the GVG-AI competition ran in 2014 (Pérez-Liébana *et al.*, 2016). The competition itself defines games using a domain-specific language known as Video Game Description Language (VGDL), which was first proposed in (Ebner *et al.*, 2013). VGDL defines games in terms of objects and interactions between them. The game rules are defined in terms of interactions between game objects (for example, when the player interacts with the block, the block moves one position).

The implementation used for the competition is written Java and provides a number of sample agents for each of the tracks within the competition.

2.6.1 GVG-AI Tracks

The GVG-AI competition has a number of 'tracks' each of which explore a different facet of game AI research (Perez-Liebana *et al.*, 2019). As of 2019, the tracks used within the competition are:

single-player planning	Agents play single-player games with access to a forward model
two-player planning	Agents play two-player games with access to a forward model
single-player learning	Agents play single-player games, but without access to a forward model
level generation	Generate a level for a predefined set of game-rules
rule generation	Generate a new set of games rules
real-world physics games	Games that have real-time physics rather than turn-based physics

All the tracks in the competition follow a similar structure. There are three groupings of games. Each of these consist of 10 games with 5 levels each. The *public* game sets

are known to researchers, and are meant to be used for evaluation of the agents before submission to the competition. There are two sets of games which are not made public. The validation set is kept on the competition server and allows agents to validate their performance on an unknown set of games. The final set of games, the test set, is kept private and are the games which the agents are evaluated against to declare competition winners.

The competition awards points based on how well each agent performs across the test set. With the top ten agents per game being awarded points based on how well they performed. The points awarded are allocated according to an 'f1-style ranking system' the points being awarded per position following a decreasing reward structure: 25, 18, 15, 12, 10, 8, 6, 4, 2, and 1. The agent's final score is the sum of all of the points awarded across the 10 games evaluated (for example, if an agent obtained first place in game 1, second place in game 2, and 11th place in the other 8 games its' final score would be $25 + 18 + 0 + ... + 0$).

For this thesis, the two most relevant tracks are the single-player planning track (which is the oldest track) and the two-player planning track (which was created while this research was taking place).

Just as the games presented in this thesis, the agents playing games in these tracks have access to a *forward model*. The model allows the agents to simulate the effects of actions without access to a full (inspectable) description of the game. This is different from traditional planning competitions such as those held at ICAPS and the GGP competition which provide a complete problem description to the agents. The design of this competition has heavily influenced the choices made within this thesis, both in terms of the design of the games presented in chapter 3 and the choice of algorithms used (and presented in chapter 4).

In addition to the planning tracks, a number of other 'off-shoot' tracks are part of the GVG-AI competition. These include a single player learning track, where the players are not provided with a forward model but are given a chance to explore the environment before scoring takes place and tracks focused on procedural generation both in terms of levels for existing games and new games.

Single Player planning track

The single-player planning track provides the controller with a description of the state, a set of legal actions and the current score. Although the game rules are not given to the agents, the agents have the ability to simulate the effects of actions on the game state using a forward model. This is in contrast to the learning track where the forward model is not available and the agents are instead allowed 'training time' explore the environment before evaluation. During the competition, agents are only given a short time-period (40ms) to return their decision.

Table 2.3 Top 10 final competition results from GVG-AI 2014

Agent	Approach	Total Points
adrienctx	OLETS	158
JinJerry	MCTS	148
SampleMCTS	MCTS	99
Shmokin	A* / MCTS	77
Normal MCTS	MCTS	68
culim	Q Learning	61
MMbot	MCTS	59
TESTGAG	GA	52
Yraid	GA	49
T2Thompson	A* / hill climbing	47

In the results from the first year of the competition (table 2.3), the tree-search based agents performed well and showed promise for general video game playing. One particular approach, MCTS and its variants performed well overall in the

Table 2.4 Winners across all single-player planning track legs

Agent	Approach	Type
CIG-14	OLETS	Tree Search
GECCO-15	YOLOBOT	Hyper-heuristic
CIG-15	Return42	Hyper-heuristic
CEEC-15	YBCriber	Hybrid
GECCO-16	YOLOBOT	Hyper-heuristic
CIG-16	MaastCTS2	Tree Search
GECCO-17	YOLOBOT	Hyper-heuristic
WCCI-18	YOLOBOT	Hyper-heuristic

competition. This algorithm will be presented in more detail later in this thesis (section 4.2).

The across all of the iterations of the track, two popular approaches have been MCTS (or similar tree-search based approaches) and agents based on evolutionary computation. The evolution-based approach which has seen the most success is Rolling Horizon Evolutionary Algorithms (RHEAs). These agents evolve a sequence of moves which are then executed using the forward model. The performance of difference sequences are compared and the first move from the sequence that shows the best performance is returned at the agent's move for this turn. Just like in MCTS, the search is recomputed every move.

Table 2.3 contains a reproduction of the results presented in Perez-Liebana *et al.*, 2019, filtered for the single-player version of the competition. The table shows the winners have largely been dominated by agents that attempt to select a suitable algorithm based on some feature of the game being played. These have been referred to as, 'hyper-heuristic' approaches by the competition organisers.

The first winner of the competition, Open Loop Expectimax Search (OLETS), was created by Couetoux. Although the agent was not described in its own research paper, the algorithm is defined in the paper describing the results of the first competition (Pérez-Liébana *et al.*, 2016). The agent utilises open-loop search (states visited are not stored in the associated tree-node). The approach is based on hierarchical open-loop optimistic planning. Rather than using roll-outs the agent uses the scores at the leaf-nodes of the tree and replaces Upper Confidence Bound applied to trees (UCT) with open-loop expectimax.

Both YOLOBOT and Return42, two agents which have won competition legs fall into this category. They both switched to breadth-first search after analysing the game type. YOLOBOT (Joppen *et al.*, 2017) performs a heuristic breadth-first search and switches to MCTS if heuristic search fails (as determined by the game being detected as stochastic, or taking too long). Return42 determines if the game is stochastic, and if so uses A*, otherwise it agent makes use of random walks. Ashlock *et al.* notes this kind of approach shows great potential, but note that there is fairly clear differentiation in the games between those that MCTS performs well on, but there are also games it performs very poorly on. They explore approaches for classifying the games to decide amongst a set of portfolio agents based around MCTS variants (Ashlock *et al.*, 2017).

There have been many attempts to modify MCTS to improve its performance in GVG-AI. A summary of these approaches will be presented in section 2.6.2.

GVG-AI Two Player Track

In 2016, the competition launched a multi-agent track (Gaina *et al.*, 2016). In this track, two agents are placed into a game, and they are required to solve the level. The framework supports both competitive and co-operative games but the majority

of the games within the framework are competitive games. In the paper which sets out the competition track, the two sample MCTS controllers achieved the best scores.

Table 2.5 Top 10 final competition results from GVG-AI 2 player competition (CIG 2017)

Agent	Approach	Total Points
ToVo2	MCTS	161
essex_acwebb	MCTS	108
not2048[1]	MCTS	104
ehauckdo	MCTS	99
Number27	GA	74
SampleRS	GA	74
SampleOLMCTS	MCTS	71
adienctx	Tree Search	63
SpaceJohn_Team	MCTS/GA	57

The results from the most recent edition of the 2-player GVG-AI competition (table 2.5) show that a majority of the top 10 agents were using MCTS. The other technique which was represented within the sample set is GAs. The GA-based techniques use an on-line variant of a generic algorithm. This technique is used to evolve action sequences based on state observations (see section 4.4 for a description of the technique).

The two-player competition winners table 2.6 are largely hybrid based approaches, filtered from the same source as the single-player planning competition winners Perez-

Table 2.6 Winners across all two-player planning track legs

Agent	Approach	Type
WCCI-16	Tovo2	Hybrid
CIG-16	Number27	Hybrid
CEC-17	Tovo2	Hybrid
FDG-18	OLETS	Tree Search

Liebana *et al.*, 2019. These are all MCTS-based hybrids and the competition authors note that the competition has not seen as marked improvement over the years as the single-player competition variant Perez-Liebana *et al.*, 2019. Two of the agents which performed well in this competition (MaasCTS2 and YOLOBOT) are MCTS agents which switch to breadth-first search after initial analysis of the game type. There were two agents based on Evolutionary Algorithms (EA) techniques.

The competition organisers noted that most competition entrants employed a random opponent model. This has been noted at an area of interest for future research by the competition organisers (Gaina *et al.*, 2016). One entry to the competition did make use of a more advanced opponent model, using Q-values for the paired agent. However, this does not seem to have resulted in a consistent competitive advantage over the random-model agents. This indicates that integrating opponent-models into these kinds of agents is an area worthy of future exploration.

Ashlock *et al.* used a fixed opponent model when trying to classify games for use with hyper-heuristic approaches to two-player games.

The sample controllers provided by the framework are adaptations of the existing, longer-running single-player versions of the competition. The one-step look ahead agent assumes random-play from the opponent a random opponent model.

2.6.2 Algorithms used within GVG-AI

From the both the single and two player tracks presented above, there are two main groups of algorithms that have performed well consistently over the lifetimes of the planning tracks. These are MCTS and GA based approaches. This section will explain both of these approaches as they have been applied to GVG-AI and some of

the research around improving their performance in these competitions. A more detailed description of these algorithms can be found in chapter 4.

MCTS in GVG-AI

MCTS is one of the algorithms which has showed the most promise within GVG-AI. The algorithm does not rely on game-specific knowledge and so is well suited to general game playing.

Pérez-Liébana *et al.*, 2015 presents the difference between closed loop and open loop search. In closed loop search states are retained between iterations as part of the tree information. The paper argues that while this works well for deterministic scenarios, it is poorly suited for stochastic games. Instead, an open-loop approach which stores statistics but not states should be used to permit re-sampling of states during the search.

Soemers *et al.*, 2016 researched the effect of a range of enhancements to MCTS both from existing literature and novel approaches within GVG-AI. Among these is tree reuse (not discarding the tree after each game tick) and performing a 1-ply breadth first search before performing MCTS to deal with cases were the agent cannot perform enough iterations to explore the tree sufficiently. To deal with stochastic games the moves are executed multiple times and the average rewards are used. They also look at knowledge-based evaluations to reduce the problem of games which feature a sparse reward space. The research shows that tree-reuse with a decay factor shows an improvement over not-reusing the tree between ticks.

One of the most successful players across the competitions run is YOLOBOT which uses a heuristic-based breadth first search, unless it determines the game is stochastic, or the level in not solved within a certain number of steps in which case it falls

back to MCTS. The approach of combining a classical search approach for deterministic games can also be seen in Return42, which employs A* for deterministic games and uses random walks for stochastic games.

Within the multi-player version of the competition, there is less variation in the agents - with the competition organisers noting that this indicates the need for further research into the area of multi-player games (Perez-Liebana *et al.*, 2019).

As in the single player version agents analysed the game state to choose a suitable algorithm, with both MassCTS2 and YOLOBot switching to breadth first search. MaastCTS2 performs a one-ply breadth-first search to prune actions. ToVo2 made adjustments to roll-outs such as limiting the depth of the roll out or weighting action section.

Most of the agents for the 2-player competition utilise random opponent models (Perez-Liebana *et al.*, 2019). This indicates that further research into using suitable models for game-playing agents is needed. For the 2016 competition, one agent attempted to assume 'always co-operate' (maximise paired agent's reward) in GVG-AI (based on the work presented in chapter 5) named *webpigeon*[2] (Perez-Liebana *et al.*, 2019). One notable exception which utilized an intelligent opponent model is MaasCTS2, which used a random opponent model in the first 2016 competition, but switched to a Q-value based approach for the second 2016 competition (Gaina *et al.*, 2017d).

There have been attempts to add opponent models for 2-player *GVG-AI*. Including a study of nine different approaches, however only two of the nine approaches examined (probabilistic and unlimited buffer) outperformed a random opponent model (Gonzalez-Castro and Pérez-Liébana, 2017).

[2]This was my agent

To work in two-player environments, the sample OLMCTS agent is modified so that it assumes that the other agent makes a random move.

Rolling-Horizon Evolution

There are a number of parameters which can be 'tuned' for RHEA. Gaina *et al.*, 2017a explores the effect of a number of parameters in the 'stock' (supplied with the framework) RHEA implementation. These included population size and length of the action sequence. The research also used a fixed number of forward model calls to restrict the performance of the agents to be similar to that of the MCTS agent in a similar time budget. They showed that for lower population counts the winning rate increased with individual's length.

The algorithm is also sensitive to its initial population (the individuals generated on the first iteration). Gaina *et al.*, 2017b analysed the effect of this by comparing random initialization (the default for the 'stock' agent), an exhaustive one-step look ahead approach to seed one member of the initial population (with the rest being random) and an approach which used half of it's budget to use MCTS to seed the first individual in the population. They found that both approaches improve the performance of the RHEA agent when the population size was small.

As RHEA consists of evolving an action sequence and the evaluation terminates at the end of the action sequence this limits the agent's ability to see long-term rewards. Some of the techniques discussed already help to counter this (i.e. increasing the action length). There have also been attempts to incorporate roll-outs to the end of the evaluation Gaina *et al.*, 2017c and incorporate learned policy and value networks Tong *et al.*, 2019 to aid in evaluation (however this second approach was not evaluated using GVG-AI).

An extreme case of small population counts is the Random mutation hill climber (RMHC). This is an example of an Evolutionary Statergy (ES), where mutation is the primary (only) driver of improvement, rather than using crossover. On each iteration, the current individual is mutated and the mutated version is evaluated. If the agent performs better, then the mutated individual replaces it's 'parent'. If its performance is worse, then it is discarded. As an individual with worse fitness will never replace it's parent, this guarantees that performance on each iteration will either remain constant or improve.

RHEA has been expanded into the two-player domain in the form of Rolling Horizon Evolutionary Algorithm (RHCA) Liu *et al.*, 2016. As with *Tiny Co-Op*, Unfortunately, the two-player track was not available for evaluation so instead spacewar was used. This version keeps two pools of agents, one representing the player and the other representing the opponent. The also compared it to the stock two player Open Loop Monte Carlo Tree Search (OLMCTS) using the assumption that the other player moved randomly (as discussed above). The results from a full round-robin league show no clear difference in performance between OLMCTS and the evolutionary-based approaches.

Other approaches for deciding the opponents behaviours have been considered. Another attempt at co-evolution for GVG-AI can been seen in Ringer *et al.*, n.d. which represents the opponents policy using RMHC. On each evaluation the current opponent policy is mutated and both are evaluated against the best player policy so far. This is an attempt to balance time spent evaluating opponent policies and allow more time evaluating the agent's own policy without assuming a random model.

Rolling-Horizon Evolution / MCTS hybrids

Following on from the work presented in the previous sections, there have been attempts to combine MCTS and RHEA to get the 'best of both worlds'. As well as the population seeding approaches mentioned above, there have been attempts to incorporate elements of MCTS and bandit-based approaches more generally into RHEA. Gaina *et al.*, 2017c compares a number of hybridisation techniques across different population sizes and lengths. These include modifying mutation to use bandit-based approaches to select which gene to mutate, keeping a tree based on the performance of the evolved agents to allow for selecting agents based on their highest UCB values and not discarding information between ticks (similar to the concept of tree reuse within MCTS) and using rollouts at the end of the action sequence to provide a longer-term view. Again, this work used a limited forward-model call budget to limit the agents. This work showed that the biggest performance gain was based on keeping data between moves and the bandit hybrid approaches performed poorly.

2.7 Multi-agent environments

This section will look at the work in multi-agent environments, both competitive and co-operative. This section will outline what work has already been conducted within the field on agent modelling, some of the approaches taken and possible limitations to these approaches. For the purposes of this section, 'player' refers to the agent who's perspective we are acting from and opponent (or paired agent) refers to the other agent(s) in the environment.

2.7.1 Tree Search-Based approaches

For small state spaces, an exhaustive search all possible states and their actions is possible. Exhaustive breadth-first search which stops execution when a solution is found is optimal (is guaranteed to find the best solution) and will find the solution if one exists.

Modelling Opponent Behaviour

Some of the earliest work in player modelling for games comes from work on adversarial zero-sum games such as chess. In these games the game states are modelled as nodes on a tree and the actions the connections between them. One of the most famous early examples of attempting to outwit an opponent's strategy is that of the minimax algorithm Du and Pardalos, 2013. This algorithm attempts to minimise the opponent's rewards while at the same time maximising the players reward. Although this strategy is optimal it requires an exhaustive search of the game tree, which is often not possible given the availability of compute resources.

As we can assume that a rational opponent will attempt to maximise their own reward, moves that lead to a suboptimal states can be ignored. This allows branches of the tree which correspond to actions that the opponent will never select to be 'pruned' from the search space. One such pruning optimisation that has worked well for classical board games is Alpha-Beta pruning (Knuth and Moore, 1975). This stores two values, the alpha value is the minimum score of the maximising player (ie, the agent) and beta, the maximum score of the minimising player (ie, the opponent). When beta becomes less than alpha, the unexpanded children of this node need not be expanded (as an optimal opponent will always choose another, better, action). This is demonstrated in fig. 2.2.

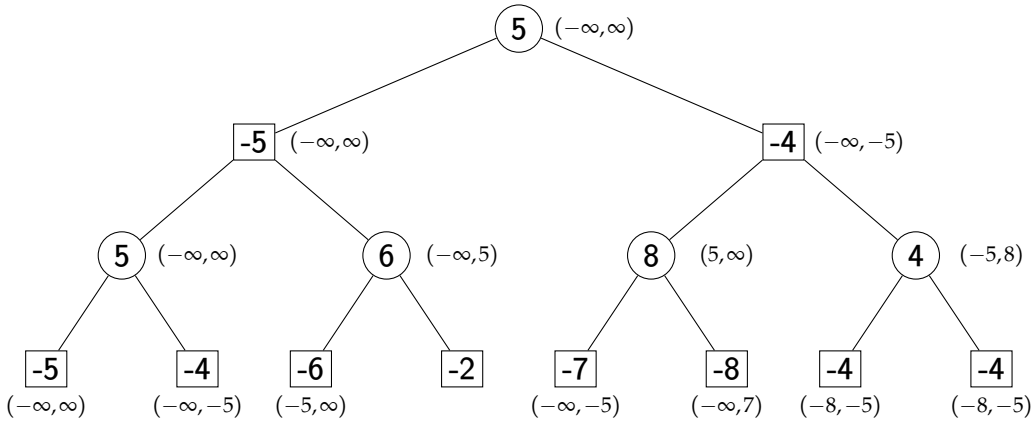


Fig. 2.2 An example of alpha-beta pruning

If an opponent does not play optimally, then it's possible that victory may be achieved sooner by exploiting errors that the opponent makes. Assuming knowledge of the opponents evaluation function (or a suitable approximation) would allow better performance that not modelling them. This is formalized as opponent model search (Iida *et al.*, 1993). This can not yield results worse than minimax search on the assumption that the agent has complete access to the opponent's strategy and that this can be expressed through their evaluation function. This approach works by computing the evaluation function for all terminal (or at the depth limit) nodes from both players perspectives and allowing this to select nodes rather than choosing optimal minimax nodes. It is also only suitable for use when the evaluation function can be executed quicker than the nodes that can be skipped as a result of the search. If the evaluation function is inaccurate, the agent may choose incorrectly and as a result may result in a worse score or possibly a loss.

This approach is not without its pitfalls, if the player does not have perfect information of the opponent's strategy, or incorrectly assumes the depth that the opponent is operating it, it may inadvertently result in a worse score, or even a loss (Iida *et al.*, 1994). OM search assumes a fixed opponent model. This is replaced with a probabilistic opponent model for probabilistic opponent model search (PrOM)

(Donkers *et al.*, 2001). This assumes multiple opponent models each of which feature a probability distribution over them.

In situations where opponents may themselves be taking into account modelling strategies of their opponents (for example player one alters their behaviour on the assumption that player one is modelling them). A term borrowed from Psychology can be used, Theory of Mind (Premack and Woodruff, 1978; Rabinowitz *et al.*, 2018). It concerns the ability of a human to be able to work out what other humans know and believe. Incorporating models of player behaviour has been successful in simple games, such as *Rock, Paper, Scissors* and limited betting. For such games, agents which have higher-order theory of mind are able to outperform those agents which in turn are able to outperform lower-order theory of mind, but beyond this they do not show an increase in performance (De Weerd *et al.*, 2013).

2.7.2 Co-operative Approaches

Modelling of behaviours of agents may not only be useful in competitive games. The field of multi-agent planning adapts the ideas of classical planning (planning assuming a deterministic single-agent environment) to environments where more than one agent are present. The presence of multiple-agents in a classical planning scenario causes issues as it can cause assumptions that are inherent in the classical planning space to be invalidated.

Within this field of research, there is a distinction between how the agents in the environment are controlled. When one controller is permitted to plan for all agents within the domain, the problem is regarded as a centralised planning problem (Kovács, 2012). This is analogous to the approaches describes in the games competition research where a single controller is able to co-ordinate for a complete

team of agents (such as in the original Pacman vs Ghosts competition). Centralized planning problems are often regarded as more efficient than requiring agents to plan independently and co-ordinate their actions. This centralized planning approach does have a fairly major drawback in that it requires the agents to be able to share information to communicate effectively - often assuming complete knowledge of the agent's internal state and goals (Kovács, 2012). This approach of allowing a single overarching controller to make team based decisions is also present in the machine learning literature, where the term *Team learning* is used (Panait and Luke, 2005).

However, in teams where the agents are expected to co-ordinate with human beings, or agents that are following a strategy that is unknown, this centralized approach will not work. In the games and competitions already discussed, such as the GVG-AI 2-player planning track. As noted when discussing the 2-player planning track, the agents that were present in the first years of the competition largely assumed they were paired with an agent making random moves. Agents in this game are not given access to the internal state of the paired agent, nor is this feasible, and considering that the track consists of both co-operative and competitive games, may not even be desirable. There is another case that is worth considering, and that is when such a model simply does not exist, as would be the case if the agent is paired with a human participant. We also cannot assume that the paired agent would be willing to co-operate in whatever strategy was devised. It may also be the case that explicit communication to co-ordinate actions is not possible (as is the case in the GVG-AI and original Tiny Co-Op games).

When agents are unable or unwilling to engage in centralized planning, the agents must plan independently and co-ordinate their actions (*distributed planning* or *concurrent learning*). This approach is often needed in real-world scenarios, as agents are

either unable or unwilling to share the information necessary to enable centralised planning to take place (Weerdt and Clement, 2009). They also argue that multi-agent co-ordination is different from a set of single-agent planning problems due to dependencies between agent tasks. This requires agents to co-ordinate with each other (Jennings, 2000). The multi-agent planning community also offers potentially useful insights into co-operative problems. (Brafman and Domshlak, 2008) attempt to establish an upper bound on the complexity of multi-agent planning problems. They found that the complexity is largely based on the coupling of the domain (how much the agents need to work together) and the number of commitments made per agent to solve a given problem.

Multi-agent pathfinding

The need for multiple-agents to be able to find paths within the same environment is a problem which spans multiple application domains, with multiple variants of the problem depending on application constraints Stern *et al.*, 2019. There is the same distinction between *centralized* and *distributed* environments present within the multi-agent planning literature described above (Felner *et al.*, 2017). For the same reasons outlined within that section, the focus of this work falls within the *distributed* setting.

Solutions to the problem assume full knowledge of the agent and it's planned path Silver, 2005. This is similar to solutions to the classical multi-agent planning problems mentioned above, and isn't suitable for the same reasons. Distributed pathfinding agents can be described as either *cooperative* or *self-interested*.

In environments where multiple agents need to find paths but avoid collisions, *cooperative pathfinding* can be used Silver, 2005. This operates under the same assumptions as some multi-agent planning approaches, in that it assumes full

knowledge of the other agents and their routes. The same arguments as to why distributed planning is important for this thesis apply here - access to the player's actions may be incomplete or inaccessible. In the *non-cooperative* case, agents can assumed to be self-interested and will act to minimise their own cost (Bnaya *et al.*, 2013).

In the fully co-operative case, agents are assumed to be willing to follow paths planned for them or obey restrictions on their paths, or self-interested and seek only to minimise their own cost. Within game-like environments, the behaviour of players cannot be assumed to be fully co-operative.

Multi-agent path-finding is still an active an active area of research, there is recent work in applying MCTS to variants of the problem Kiarostami *et al.*, 2019; Kiarostami *et al.*, 2020. One of the games explored within this thesis could be thought as a variant of this problem. The tiny co-op game, presented in sections 3.2 and 3.3 is a puzzle game which features a path-finding element, but also requires more complicated co-ordination between the agents (as they must also alter the environment to make paths become available). Two versions of the game will be used, one where communication between the agents is non-existent and one where it is very limited.

2.8 Hanabi

Hanabi is a co-operative card game in which every player knows the cards of other players but not their own. The game's rules will be fully presented later in this thesis (see section 3.4),. To briefly summarise the game for the purposes of this section, the game is a co-operative partially observable card game where players must co-ordinate to ensure that information about cards is provided to players, and

that they can play cards in the correct sequence. The communication between the players is extremely limited, and requires spending an in game resource.

The game has drawn the attention of academics, game-players and hobbyist programmers alike. Two open-source implementations of the game have been made available for academic research. The first, which was developed for the research presented in this thesis, was used for the *Hanabi* competition which took place at the IEEE Symposium on Computational Intelligence and Games (CIG), and it's successor conference, the IEEE Conference on Games (CoG). The competition is described in Walton-Rivers *et al.*, 2019). This implementation is designed to follow the format of the GVG-AI competitions and therefore provides access to a forward model. This implementation has been used for research into *Hanabi* playing agents, most notably Goodman, 2019, Goodman and Lucas, 2020, and Canaan *et al.*, 2018.

The other major implementation is that provided by Google Deepmind, is the Hanabi Learning Environment. This tackles the game from the same perspective as AI Gym Bard *et al.*, 2019. In this version, the agents are not given access to a forward model and instead are expected to learn the game model along with co-operating with the agents. As a result, this version of the framework focuses more on machine learning than planning. The framework is also implemented in C, with python bindings. The work using this framework has been impressive, with it being used as a test bed for the Bayesian action decoder (BAD) algorithm(Foerster *et al.*, 2018), which is closely related to theory of mind.

Hanabi presents an opportunity to study interaction with human players. This includes incorporating communication theory and psychology research to create agents that interact directly with human participants Eger *et al.*, 2017, and has been proposed as a test-bed for human-computer co-operation (Canaan, 2018). There

has also been research into evaluation of both the subjective (player perceptions) and objective (obtained score) performance of both rule-based and learning-based agents when paired with human participants (Siu *et al.*, 2021).

As well as research into human-ai co-operation, and approaches for working in co-operative applications. There has been research into making the most efficient use of the limited availability of communication in the game. Most notably, developing protocols to communicate information outside of what is explicitly intended. Cox *et al.*, 2015 developed two strategies around, 'hat guessing' and encoding the solution to that problem as a *Hanabi* moves. Their strategies only work for the 4 and 5-player variants as the rules don't allow giving hints that don't identify any cards. His agents were able to play games near-optimally when paired with agents using the same strategy. (Bouzy, 2017) explored variants of *Hanabi* which relaxed the rules. Hat guessing variants achieve high-scores but only when the teams are all following (or at least are aware of) the strategy. This approach is a good way to obtain a high score, but does not meet the requirement of being able to deal with ad-hoc teams.

Another interesting (and somewhat subversive) approach is to work around the lack of communication though other means. Eger and Gruss, 2019 uses a 'covert channel' in that they encode information in the time taken to perform an action. Human beings make use of timing information for communication and use subtle queues when communicating with each other such as body language. However, for the purposes of the work for this thesis, this is out of scope. The researchers which proposed this approach even recognise this, by mentioning it would not be within the spirit of the COG competition. In a similar vain, Gottwald *et al.*, 2018 integrated eye-tracking into *Hanabi*, with a view to using this to inform agent decisions. One of the reasons *Hanabi* is so suitable for this research is the limited communication channel and these approaches make use of information outside of that channel.

Although outside of the scope for this thesis, this approaches pose some avenues of research that are extremely interesting and something that should be explored further.

Human players strategies are often based around rules. This approach has been adopted by the academic literature in the form of crafting rule-based agents which express some predefined strategy. Indeed, the on-line Hanabi community have a very detailed set of rules describing subtle communication strategies. However, as this was after the vast bulk of the work in this thesis was conducted, they have not been integrated into this work. Researchers have also proposed various approaches that use conventions or rules that the AI will follow. (Osawa, 2015) used rule-based agents to explore reflective intelligence. He implemented a number of different strategies, including: a bias random agent; an agent which is capable of telling the co-operating agent about useful cards; and an agent that is capable of remembering what the other agent knows to avoid repeating information. He argues that for games like *Hanabi*, considering the other players' position is important to ensure the agent plays well. Other approaches that make use of 'rules' or conventions include the work of van den Bergh *et al.*, 2016, which used a combination of rule-based agents and Monte Carlo techniques. This work focused around tuning parameters for a series of rules and then using this as a policy for a single-layer Monte-Carlo based agent.

For the *Hanabi* 2018 competition, two approaches were considered. Canaan *et al.*, 2018 evolved a rule-based agent using a genetic algorithm. The work used both rules that were provided with the *Hanabi* framework, which were augmented by their own rules. The competition winner used an MCTS-hybrid technique which incorporated the improvements proposed by *AlphaGo* into the sample controller. The agent was also able to perform reasonably well in the two-player version of

the game by incorporating heuristics and rule-based play for specific situations. For the second iteration of the competition, a new track focused on collaborating with a set of unknown but labelled agents was introduced. This 'learning' track focused on how well agents could adapt to a set of players over a series of games and observations during those games. There has also been research into how potential strategies for evaluation could be generated. Canaan *et al.*, 2019 proposed a technique for generating a pool of agents based of the existing approaches to allow a more diverse set of strategies for comparison. Although this was used for their internal testing, sadly it has not yet been incorporated into the competition itself has it has not run for the past 2 years.

More recently, (Bard *et al.*, 2019) have proposed *Hanabi* as a benchmark for learning algorithms due to its co-operative, partially-observable nature and need to reason about other players' strategies. They performed experiments on agents located in Github repositories as well as state-of-the-art machine learning techniques. They found that the strategies did not work well when placed in mixed strategy games. Following on from this work, (Foerster *et al.*, 2018) has proposed a learning agent capable of learning communication strategies based on agent play. The technique was able to obtain a mean score of 24.174 points on two-player games, but assumes that other players can understand the conventions it learns (similar to hat-guessing agents).

2.9 Conclusions

This chapter outlined the existing work within the academic community relating to existing work around games competitions and approaches for dealing with both competitive and co-operative modelling. This chapter also provided a basis for

the deign of one of the games which will be presented in the next section. Namely the need (at time of writing), for a GVG-AI-like 2-player environment to test co-operative play. The design of this environment (and the experiments that take place in it) are designed along the same principles of the contemporary research around the general video game playing framework. Namely, a grid-based environment which features 2-players and largely assumes a random opponent model. This has been shown in the research to not be an optimal choice, but given that this assumption is made by a large majority of the agents submitted for the first few years of the competition it is not an unreasonable one.

This chapter also provided some basis for ideas behind agent/opponent modelling and why it may be beneficial for co-operative domains. Namely, that centralized approaches are not feasible and therefore approaches which incorporate some model of the opponent are needed. The motivation for this is clear - competitions such as the two-player GVG-AI competition make reference to this work being an important avenue of research. Although integrating opponent/paired agent models is not a new concept, this thesis will aim to evaluate the effectiveness of doing so in two non-competitive games as a basis for future research.

Although well outside the scope of what is possible over the time-scale of the experiments presented in this thesis. This chapter also provided some of the motivating examples as to what this research might one day lead to. Namely, it has the potential to create characters which could enhance the believability of the characters and thus improve the player's experience. Although a motivating factor for the work presented in this thesis, this will not be the primary focus of it as this is far too large of a project for this thesis to address.

Instead, this work focuses in incorporating such a model into an algorithm that has shown potential within the game-playing research. The approach which will be taken within this thesis is to compare these modifications to something similar to the sample agents within the GVG-AI framework. Clearly, as described above, a great deal of research has gone into optimising the performance of different MCTS variants within the competitions. The focus of this thesis is the modelling component, to compare against every one of these variations is out of scope and would require far more computational time than is possible. The way in which these algorithms will be used will be discussed in chapter 4.

This chapter also looked at a range of possible games which might be used for the work. The two environments used will be presented in chapter 3. The first of these is *Tiny Co-Op*, a co-operative puzzle game, which is similar in design to both GVG-AI and multi-agent path-finding problems. After work in this game, a second game with more complex agent behaviours will be used, *Hanabi*. This game offers the need for agents to communicate and co-operate but adds partial observability and provides the opportunity to extend the ideas explored in Tiny Co-Op to games with more than two players. These games will be presented in chapter 3.

Chapter 3

Games

This chapter outlines the games used by this thesis.

The first game presented is *Tiny Co-Op*, described in section 3.2. This two player game is used as a test-case for the performance of Monte Carlo Tree Search (MCTS) and Genetic Algorithms (GAs) (the experiment presented in chapter 5). The game was re-written to include communication actions, for the experiments presented in chapter 6 (the evaluation of the effect of adding communication actions to the game on MCTS - section 6.2 and the effect of prediction in the game section 6.3).

The second game, *Hanabi* is presented in section 3.4. The game allows for 2 to 5 players, which permits the evaluation of the performance of prediction in games with more than two players. The game also features a well-structured communication mechanism which similar to *Tiny Co-Op* takes the form of an action that can be performed on the player's turn. This game forms the basis for the rest of the experimental data in the thesis (chapters 7 and 8). This is a co-operative, partially-observable game which has been the topic of recent research, presented in section 2.8.

3.1 Introduction

This thesis uses two games for the research presented in the later chapters.

The General Video Game AI (GVG-AI) competition, as described in section 2.6 had plans to release a two-player version of the competition. For this work, a multi-agent environment is needed and so the single player version was not suitable. The two player track was not ready at the time the initial work into *Tiny Co-Op* was conducted. Instead, a game *similar* in nature to GVG-AI was needed. The game presented in sections 3.2 and 3.3 is similar in design to *GVG-AI*, the world consists of a grid which contains objects that the player an interact with, the game operates in turns and just like the planned 2 player track, actions are simultaneous for both players. The game also shares some features with the work on multiagent pathfinding presented in section 2.7.2. Although this game proved suitable for the early work, the simplicity of the puzzles meant that a more complex domain was needed to explore the topic further.

The first version of the game, presented in section 3.2 does not feature communication and so agents can only co-ordinate their behaviour by predicting how the players will behave. A version of the game which has been altered to feature an explicit but limited communication is presented in section 3.3.

The second game used within this thesis is *Hanabi*. The game features more complex interactions between players than *Tiny Co-Op* and there are many strategies which can result in high scores, many of which require co-ordination between the players to execute correctly. For this reason, simply computing optimal moves is not suitable Bard *et al.*, 2019.

In *Hanabi*, players need to co-operate to share information about the current state of the game but the game does not feature simultaneous moves (unlike *Tiny Co-Op*). The game has been the subject of a number of recent studies (see section 2.8) and a small but useful selection of algorithms for playing the game have been proposed. This provides a way to evaluate the effects of prediction with a set of agents that are independent and act intelligently but still require co-ordination in order to be successful. *Hanabi* also has another benefit: the game supports between two and five players. As games often feature a great many players, looking at a game which can support larger numbers of players allows the evaluation of more complex situations which feature multiple players. Note that two-player variants and more-than-two-player variants are distinct. Indeed, in analysing the characteristics of games, the difference between the dynamics of two-player and multi-player forms are so great that it's arguable whether they're even the same game (Elias *et al.*, 2012).

3.2 *Tiny Co-Op*

At the time of the creation of *Tiny Co-Op*, the GVG-AI two player track was not available for use. The *Tiny Co-Op* game was first presented in (Williams *et al.*, 2015). The game, implemented in Java, was created in order to provide a test-bed for co-operative game Artifical Intelligence (AI). The game is based around solving co-operative puzzles and is loosely based on the co-operative mode from *Portal 2* in which two players must work together to open doors and reach predefined locations (Goals). A door will only open if a player is standing on its corresponding button.

The game is a simultaneous-move game which has discrete actions and is played in a grid world. Most of the levels require the players to take turns to let each other

though doors in order to reach the goals. It has two players (agents). Each agent must visit each goal once, with no reward being offered for repeated visits. Levels consist of a number of objects; these objects are fixed in location and cannot be altered by the players during the game (doors, however can be opened).

The game serves two purposes. Firstly, it serves as a means to evaluate the ability of MCTS to solve simple, two-player co-operative puzzles. This has been explored in more detail by (Williams *et al.*, 2015). Secondly, and more importantly for this thesis, it presents a multi-agent game in which players need to interact to accomplish goals. The game provides a suitable test-bed for evaluating how effective modelling can be in the simplest case.

Although this game was created before the availability of the GVG-AI 2 player framework, the game follows roughly the same conventions (a two-player grid world, which features simultaneous moves for the two players and actions that are based on interactions between the objects and the world). This meant that work could be more easily adapted to suit the conditions of that environment when it became available. Unfortunately, once it was ready this work had already been completed and the work on *Hanabi* had begun.

3.2.1 Requirements

Tiny Co-Op represents a basic test case for the exploration of co-operative, game-playing agents. The game is designed to be simple, fast to evaluate and to be easy to write levels for (levels are stored as simple text files similar in format to those used by GVG-AI (2.6)).

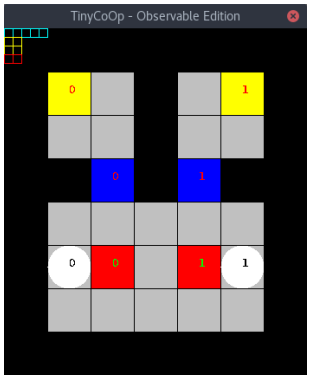
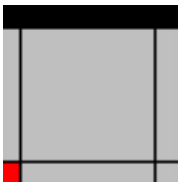


Fig. 3.1 A sample Tiny Co-Op Level

3.2.2 Level Features

The levels in *Tiny Co-Op* consist of a grid containing game objects which the player can interact with or which can hinder the player's progress though the level (see fig. 3.1 for an example).

Floor

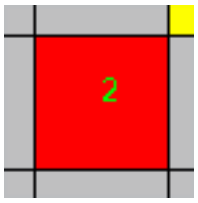


An grid cell that an agent can walk on.

Wall

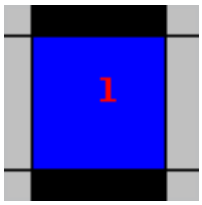
Walls are tiles which the agents cannot walk though. These are used to delimit the edges of the level and divide the level into a set of 'rooms' which the agents can navigate. If the agent attempts to walk into a wall, they will remain in their current square.

Button



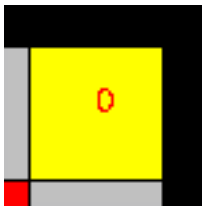
When an agent stands on a button, the button activates the doors which have the same number on them. The door will remain open as long as the agent is on the button.

Door



A door is a tile which is can only be walked upon if an agent is standing on the corresponding button. If an agent attempts to move into a square containing a door when it is 'closed' the agent will remain in their current square.

Goal



Goals are the objectives of the levels. To win the level, both agents must step into every goal square. The order in which goals are visited does not matter. Visiting the goal for the first time for each agent will result in the agents being granted a reward in the form of a score increase. Repeated visits to the same goal have no effect.

The score obtainable by the players is shared and capped at one. Both players are responsible for getting exactly half of a point (for successfully visiting each goal). When all goals have been visited by both players, the game ends and the time taken is recorded.

3.2.3 Game Interface

The agents have very limited access to the game state. The game provides a basic forward model that allows the applying of actions for both players and the creation of copies of the current state. For evaluating the current state of game, the agents

are given access to the game's scoring function and access to a method for detecting if the game state is terminal.

```
1     public interface GameState {
2
3         // methods for getting the legal actions
4         int getActionLength();
5         List<Action> getLegalActions(int playerID);
6
7         //required for the forward model
8         void update(Action p1, Action p2);
9         GameState getClone();
10
11        int getWidth();
12        int getHeight();
13
14        // methods for evaluating the state
15        boolean hasWon();
16        double getScore();
17
18    }
```

Fig. 3.2 The information available to an agent at any given tick of *Tiny Co-Op* (Java)

3.2.4 Action Space

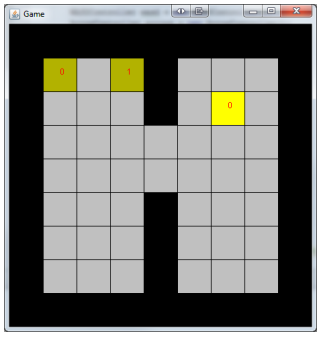
The action space included in the original *Tiny Co-Op* featured movement in the four cardinal directions. The agent was also provided with a no-op action. This permitted it to stand still on its current space, which enables the agents to stand on buttons in order to allow their partner to go through open doors.

Left	(-1, 0)	Move one grid space left
Right	(1, 0)	Move one grid space right
Up	(0, -1)	Move one grid space up
Down	(0, 1)	Move one grid space down
No-Op	(0, 0)	Stay at the current position

3.2.5 Level Descriptions

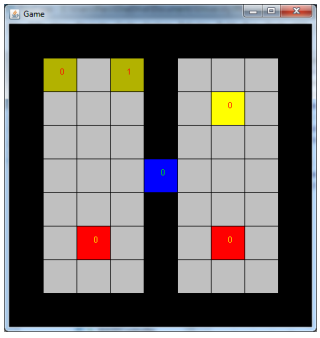
This section outlines each of the levels used in the initial version of the game. The agents (dark yellow/green squares) are in their starting positions. Other level features are as described in the above level features section. Table 3.1 shows the number of goals and size of each level. Levels with more goals require the the agent to visit more locations.

Pathfinding



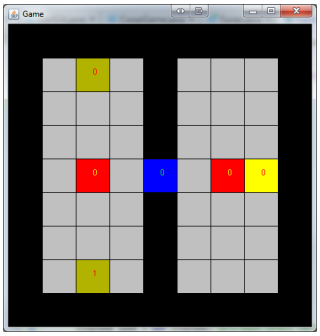
The 'pathfinding' level contains no obstacles. The agents must simply navigate to the goal. This level should be fairly trivial for an intelligent algorithm such as MCTS.

Single-door



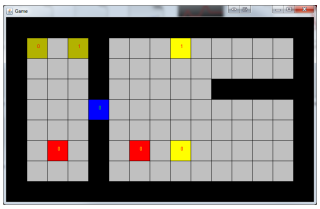
The 'single door' level contains single door that the agents need to navigate.

Symmetric Single Door



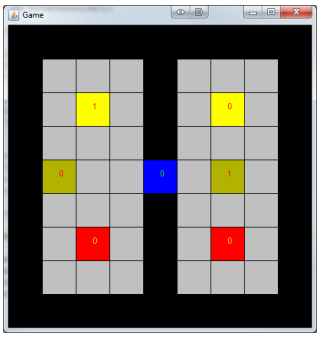
The 'single door' level is a room which is separated by door. A button in each room activates the door and both agents start in the same room. The agents must get from the first room into the second room to reach the goal.

Extended Side



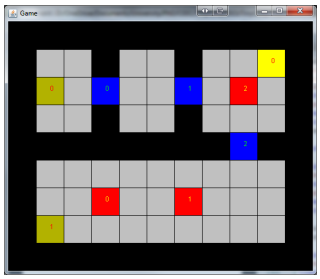
Extended side features a similar puzzle to the symmetric single door, but increases the state space. This allows comparisons of the effect of artificially increasing the state space.

Side by Side



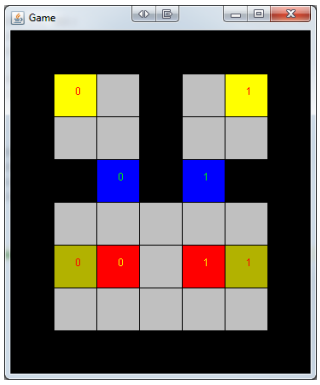
This map features two rooms, both of which contain a button and a goal. To solve this level optimally, the agents need to reach the goal in their own area, be let though the door into their partner's area and then the reach the goal in that area.

Airlock



The 'airlock' level consists of two separated spaces in which the first agent must be let though a door into another room. The actions the two agents must take are asymmetric. The first agent must navigate a series of doors which are controlled by the second agent. Once in the third and final chamber, the first agent must let the second agent into the room containing the goal.

Butterfly



This level features two small rooms which the agents must be let in and out of. The agents must collaborate with each other to be allowed into and out of the rooms. Each agent must take turns standing on the buttons and entering the rooms.

Table 3.1 Details about *Tiny Co-Op* levels

Name	Width	Height	Goals
Pathfinding	9	9	1
Single-door	9	9	1
Symmetric Single Door	9	9	1
Extended Side	15	9	2
Side by Side	9	9	2
Airlock	11	9	1
Butterfly	7	8	2

3.3 *Tiny Co-Op* (Second Edition)

Following on from the first edition of the *Tiny Co-Op*, the second edition is a backwards-compatible re-write of the game focused around the ability to add new features to the engine and explore questions central to this thesis relating to social interaction between agents. This was created specifically by the author of this thesis for the work presented in it

The game varies from the original implementation in a number of subtle but important ways. Firstly, this variant of the game features communication actions which permit agents explicitly to pass information to one another. Secondly, the information available to the agent is increased, to permit the agent to include more information about the current state in its deliberation process. Finally, the engine was designed to decrease its memory footprint and permit faster cloning by representing the state differently.

3.3.1 Communication Actions

The second edition of the game includes an added feature – communication actions. These permit the agents to spend their turn signalling the other player rather than moving. The action has no effect on the game state, other than informing the other player that it has happened.

The communication action, 'flare' (fig. 3.3) is named after its counterpart from real-time strategy games. The agent indicates a grid location for the other agent which they will receive during the next game tick (the next time they are prompted to make a move). The 'flare' is received as part of the state update and can be accessed like any other state property. This comes in two forms, relative and full.

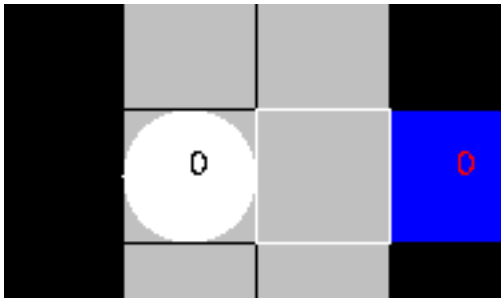


Fig. 3.3 Agent 0 being given the 'FLARE_EAST' instruction by agent 1.

Relative Communication

The agent is permitted to point out a cardinal direction to the other agent (north, east, south or west). This action set is relatively balanced in terms of branching factor: the number of communication actions will equal the number of legal movement actions, resulting in a branching factor of nine (four communication actions, four movement actions and one 'no-op' action).

By permitting the agent to communicate a direction, the agents can still show some limited explicit co-ordination between each other. They cannot communicate an arbitrary position so an agent that wanted to communicate a path would need to communicate each step individually, which is inefficient in terms of time taken.

This trades the expressiveness of full communication set for a smaller action set (allowing for a more limited branching factor) while still providing the ability for the agent to provide basic directions. This is similar to some Real-Time statergy games (RTS) or Multiplayer Online Battle Arena (MOBA) style games where players can communicate via a limited number of 'flare' or 'ping' actions. Portal 2 (the game the original Tiny Co-Op was inspired by) also features the ability for one player to communicate limited information.

Full Communication

Full communication allows the agent to identify any square on the grid. The number of communication actions will vary with the level size ($W \times H$ actions per level). This is the more expressive of the two communication actions but also introduces a much larger branching factor.

In theory, this allows the agents to be much more expressive in their communication. An agent that has access to this action set would be able to communicate a more complicated instruction such as 'move to this place'. It could also allow for arbitrary communication between two agents that had the same understanding of the action (for example, 'communicating 0,0 means I will go to the door'). If viable this also could have been used for more complex environments and puzzles which require a greater level of co-ordination between the players (for example, by adding more complicated game objects into the game).

3.3.2 State Observations

The amount of information available to each agent varies significantly in the two implementations. The second version (fig. 3.4) of the game exposes much more of the game state to the agent than the first (fig. 3.2). This is because in the original version of the game no agent required specific details for the level (they only required a forward model). The second edition was focused round the objectives of this thesis and so the requirements were relaxed to permit the agent to query state information. This additional information also allows for the creation of simple rule-based agents which can act on the information presented.

```
1  public interface ObservableGameState extends GameState {
2
3      //methods for querying the current signal state
4      boolean isSignalHigh(int signal);
5      int getSignalState(int signal);
6
7      // methods for querying about goals
8      boolean hasVisited(int agent, int goalID);
9      int getGoalsCount();
10
11     // methods for getting objects and floor types
12     GameObject getObject(int x, int y);
13     int getFloor(int x, int y);
14
15     // method for getting the player's location
16     Point getPos(int agent);
17
18     // communication
19     Point getFlare(int agent);
20 }
```

Fig. 3.4 The extended game interface available to agents in the second edition of *Tiny Co-Op*

3.3.3 Engine Differences

The second edition is built around *flyweights* and *signals*. The engine keeps a single copy of the level in memory. This is possible because the level definition is immutable in this version of the game. The game states each contain a reference to this same level definition, rather than copying the whole level whenever the agent wishes to clone the state. This is made possible by the second change made to the engine, to re-formalise the game in terms of *signals*.

In the first edition, whenever an agent stood upon a button, the map would be scanned for any door object whose value corresponds to the value of the button and its state would be changed to 'open'. In the second edition, standing on a button increases a 'signal' variable associated with the value by one. When the agent steps off the button, the value is decreased by one. When the agent attempts to step into a

space containing a door, the door is queried for its behaviour; the door then checks the value of its associated signal and either allows the agent to move onto its space or rejects the movement request. The current signal state is exposed to the agents in the interface shown in fig. 3.4.

Although not explored in the levels created to date, this allows the creation of more complex puzzle elements such as buttons which only one agent can stand on or doors which always permit one of the players though.

3.4 *Hanabi*

Hanabi is a turn-based, partially-observable co-operative card game for 2 to 5 players. The game, designed by Antoine Bauza, won the prestigious Spiel de Jahres award in 2013. Players can see other players' cards but cannot see their own and can only find information out about their cards from communication actions performed by other players and by playing them.

The game ends when the players run out of cards in the deck and all players have taken a final turn. The players start with 8 information tokens. As each action either draws a card, converts a card into an information token or does not draw a card but costs an information token the game has a finite maximum length (equivalent to the deck size plus 8).

3.4.1 Game Rules

The game consists of five suits (blue, white, yellow, green and red). Each suit contains cards which are ranked from one to five. Higher-ranked cards are present in each suit less frequently than their lower-ranked counterparts.

Rank	1	2	3	4	5
Count	3	2	2	2	1

Each player has a hand of cards which consists of either four or five cards depending on the number of players (table 3.2). Any cards not used to form these hands are used to form the deck at the start of the game.

Table 3.2 Hand sizes for *Hanabi*

Players	**Hand Size**	**Cards in deck**
2	5	40
3	5	35
4	4	34
5	4	30

Cards which are discarded during play are placed into the discard pile, which is fully observable. Cards which are played correctly in sequence are placed on the table. The highest-ranked card in each suit present on the table at the end of the game is counted towards the group's final score.

The group starts with 8 information tokens and 3 life tokens. The information tokens are used to tell other players information about the cards in their hands. Life tokens are used to keep track of errors made. If the group loses all of their life tokens, the game is over and the score obtained up to that point is the final score of the game.

Tell Spend an information token to tell another player about all the cards of a given rank or suit in their hand. All cards of this rank or suit must be pointed out and there must be at least one card present with the specified feature.

Play Play a card from the player's hand. If the card's rank is one higher than the card on the table from that suit, the card is placed on the table,

otherwise, the card is discarded and the group loses one life token. The player takes a card from the deck to replace the card from their hand.

Discard Discard a card from the player's hand, placing it in the discard pile. The group gains one information token. If the group already have 8 information tokens, this action cannot be taken. The player then takes a card from the deck to replace the card from their hand.

Players may also replenish an information token by playing the last card (the card with rank 5) of a suit correctly. If they already have 8 information tokens, then they do not receive the extra token. If all suits are completed, the game is over and the players have obtained the best possible score (25). If the deck is depleted of cards, each player gets one last turn. After this turn, the final score of the game is recorded.

Players are forbidden from looking at their own cards. The only way for the players to gain information about the cards in their hands is by being told about them by other players.

Note that the player must always point out all cards of a particular number or suit when executing a tell action. The players also receive negative information when making moves, for example if they are told that the cards in positions 1 and 3 are red cards, they are also being told that the cards in positions 0, 2 and 4 are not red cards.

3.4.2 Considerations for the Framework

The implementation of *Hanabi* used in this thesis is based on a client-server architecture. This is to allow the game to be played over network connections, across Java

Virtual Machines (JVMs) and to provide a way to isolate the agents from the engine in order to avoid accidental information leakage.

Agents send game actions (such as, 'Play the card in slot 1') to the engine and receive a set of events in return (The card played was a RED 1, it was played correctly, player 1 has picked up a BLUE 5). Events that are hidden from that player's point of view (the player 1 has picked up a BLUE 5) are never transmitted to that agent.

The agents are permitted to know:

- The discarded cards
- The cards in all other player's hands
- What each player has been *told* about their hands
- The number of live and information token counters currently active
- The current state of the table

The agents are not permitted to know:

- the current deck ordering
- the current cards in their hand

As the number of cards in the deck decreases over time and the information which is told to the player increases, in general agents get more knowledge about the game as moves are made. It is also possible to prove that a game will always end - as an action either takes a card out of the deck (possibly granting an information token) or spends an information token. The game can end before this if the players win the game or make errors and run out of life tokens before the deck is exhausted.

Information Leakage

It is important to distinguish what the player has been *told* about their hands and what the player *knows* about their cards. It is possible that a player knows more than what they have been told using what they can see, logic and deduction. This difference is important as the engine must **never** reveal what a player knows to another player, as doing so may reveal additional information to that player.

It is possible to leak information about the game state accidentally if the game itself is providing reflective capabilities, (for example, if there are only two cards of rank 5 left in play and both players are in possession of one of their fives). If the game engine can be prompted to provide what a given player knows about their hand, a naive implementation may simply enumerate the possible cards based on that player's perspective.

If it does this however, the game engine will return that only the possible card for that slot, say, is a white 5), rather than what we can see from our perspective – that the card is either a white 5 or a red 5. Even if we know nothing about the red 5 in our hand, the engine has just 'leaked' the fact that the red 5 is visible to that player and in doing so, as it was not information that we could have deduced for ourselves, the red 5 must therefore be in our hand. Because in our framework, the agent is never in a position to query the game engine directly, the engine will not leak information in this way.

Deck ordering

At any given point, it is possible to know what cards are either in the deck or in our hand but not the exact positions of these cards. The framework therefore treats cards in the player's hand as in the deck until they are played.

The game features a structured form of communication which is integrated into its rules. This makes it better suited for this kind of research than other social deduction games such as *Warewolf* or *The Resistance*, which make use of natural language as part of gameplay. In fact, *Hanabi* explicitly forbids communication outside of the well-structured communication channel imposed by the game rules. This puts the AIs and human players on a more level playing field.

Chapter 4

Algorithms in This Thesis

This chapter outlines the algorithms which are used for the experiments presented in the rest of the thesis.

Section 4.1 describes the rule-based agent framework used for *Hanabi* in chapter 7 and chapter 8.

Section 4.2 describes Monte Carlo Tree Search (MCTS), a tree-search algorithm that has found success in general game playing.

Section 4.3 describes a modification of MCTS, Predictor MCTS, that makes use of a paired agent model to see the effect this has on the games. It s used in the experiments presented in this thesis for evaluating the effectiveness of prediction in co-operative domains.

Section 4.4 describes genetic algorithms. Variants of this technique are used in chapter 5 and chapter 6 as agents for evaluation. It is combined with rule-based agents in chapter 8 for learning agent models.

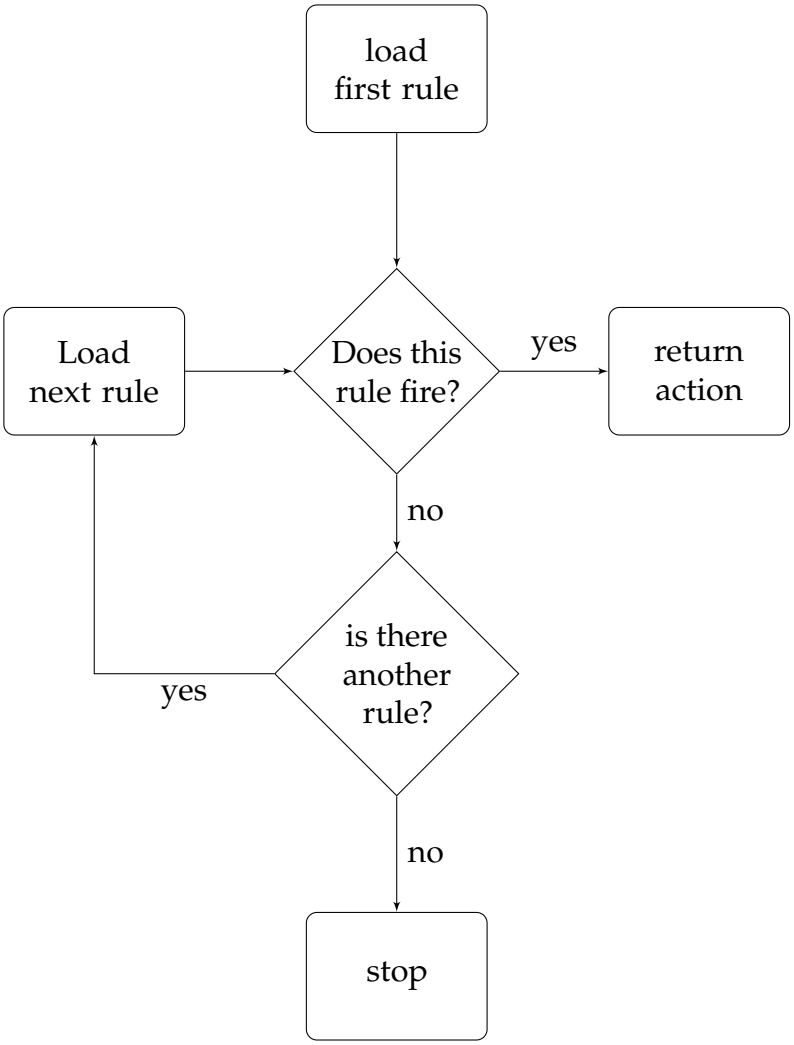


Fig. 4.1 A flow chart of the behaviour of the rule-based agents

4.1 Production-Rule Agents

Rule-Based approaches are easy to create. They are analogous to the techniques currently used for game Artifical Intelligence (AI). These can conceptually be thought of as a *production system* – a series of IF ... THEN ... rules which are evaluated until one matches. When a rule matches, the rule is executed and then the appropriate move is returned.

These are well suited to representing human *Hanabi* strategies as these are often described by players as a series of steps to follow. The framework also contains implementations of particular rule-based strategies described in other literature on *Hanabi*-playing agents (Osawa, 2015; van den Bergh *et al.*, 2016).

The rules are combined into a grouping which constitutes a production-rule agent. If all rules present in the agent are evaluated and no rule has fired, the agent is classified as *incomplete*.

The *Hanabi* framework provides a 'library' of rules which can be accessed using the 'Ruleset' class. These can be used to evolve game playing agents or used in the on-line designer[1].

4.1.1 Rule Classifications

Rules within the framework are categorised along two dimensions. The first dimension relates to the type of the action the rule is associated with. These correspond to the types of move possible in *Hanabi* (Tell, Play and Discard). Compound rules can take one of multiple game actions depending on the game state, although within the rule set only one such rule exists: Try to Unblock. This rule will attempt to ensure that the next player has a move to make, so will either tell that player about a playable card or discard a card in order to ensure they have information.

The second axis used to describe the rules is whether they are *deterministic* or *stochastic*. The condition of the rule must always be *deterministic*, however its effects may contain a random component. Most rules implemented within the framework have deterministic effects, but rules such as 'Tell Randomly' permit some randomness within their outcomes.

[1] http://hanabi.fosslab.uk/builder.html

Rules within the framework may also be *parametrised*. These rules have some arguments which control how the rule works. For example, the rule, 'play probably safe card' takes an floating point value which corresponds to a threshold below which the rule will not fire. In order to standardise usage of these rules, multiple copies of the rules with fixed parameters are present in the RuleSet, (in the case 'play probably safe card' these are 0.1 increments between 0 and 1).

4.1.2 Conditional Rules

The framework also features a 'special case' rule which accepts a predicate, which it will consult before evaluating the rule's condition. This was required in order to represent some of the agents described in the literature which take risks until only one life remains and then adopt a more cautious strategy.

4.2 Monte Carlo Tree Search

MCTS is a tree-search technique first proposed by Coulom, 2006. The algorithm is a combination of Monte-Carlo search and an incremental tree structure. The algorithm, and multiple variations thereof, have been applied extensively within the game-playing research community (Browne *et al.*, 2012). The algorithm asymmetrically expands its search tree in iterations, attempting to minimise regret. The algorithm is *anytime* which means that at any given point it can be stopped and the current best solution can be returned.

The algorithm has found notable success general video game playing (Browne *et al.*, 2012) and within the General Video Game AI (GVG-AI) competition (as described in section 2.6).

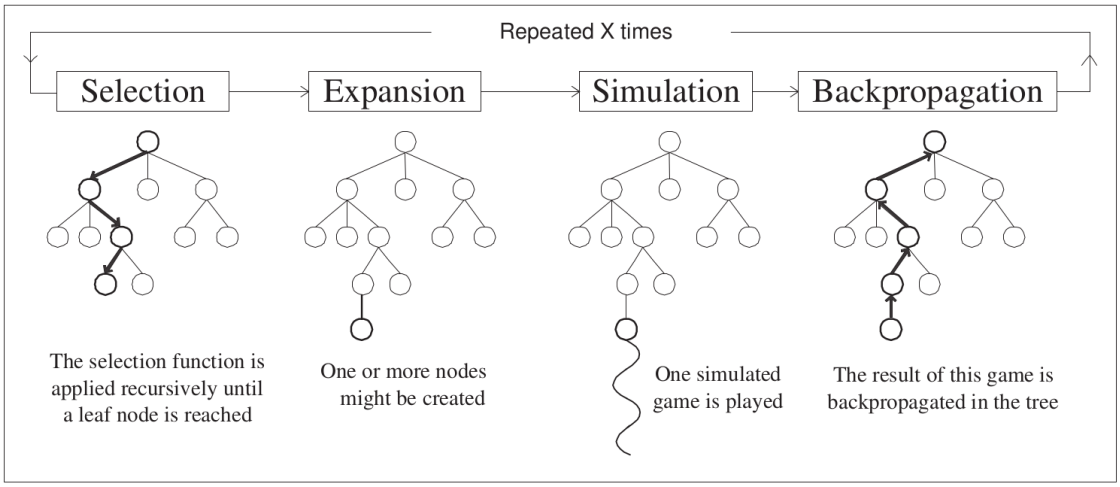


Fig. 4.2 MCTS stages, taken from Chaslot *et al.*, 2008

MCTS has been hybridised to work in a number of domains; *AlphaGo* is a hybrid between MCTS and neural networks (Silver *et al.*, 2016).

4.2.1 The Stages of MCTS

The algorithm can be divided into four phases, each of which is executed sequentially, once per iteration. These four phases are: selection, expansion, roll-out and back propagation. After each iteration, the termination criteria are checked and, if reached, no more iterations are performed. This could be a time limit or after a fixed iteration count. Once the limit has been reached, the best move from the tree is returned. This is usually selected by examining the statistics at the root node of the tree.

Selection

(Auer *et al.*, 2002) proposed the *Upper confidence bound* policy for selecting nodes for bandit selection. (Kocsis and Szepesvári, 2006) applied this formula as the tree policy in MCTS.

$$Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{4.1}$$

Where:

Q(s,a) the score obtained by applying action a in state s

C the exploration constant (balances exploration and exploitation)

N(s) the number of times this state has been visited

N(s,a) the number of times action a has been applied to state s

Although Upper Confidence Bound applied to trees (UCT) is the most popular selection algorithm used, there have been others considered. A stochastic selection strategy which has been used in simultaneous move games is Exp3. This algorithm selects moved according to a probability distribution based on the observed rewards.

Expansion

In order for Upper Confidence Bound (UCB) to be applied to a node, each action much have been explored at least once. If the selection phase reaches a node which is not fully expanded, then that node will be chosen for expansion. During expansion a new child node for an unexplored action is added to the tree. After the node has been added, the roll-out stage is entered.

Roll-Out

During the roll-out stage, actions are applied until a terminal state has been reached. In some games, the game may be infinite or the end of the game may be too far away to reach during a roll-out and allow sufficient time for exploration of other branches. To combat this, a depth-limit can be applied which will end the roll-out after a certain number of moves have been made.

Back propagation

The final stage, back-propagation, uses the results obtained from the roll-out to update the value of the node chosen in the expansion phase. The statistics for all nodes leading to this node are also updated by recursively backtracking along the visited nodes, adding the obtained score to the node and updating the visit count.

4.2.2 Modifications of MCTS

MCTS has been adapted and changed to apply to specific requirements of different kinds of environments. For this thesis, one of the more relevant set of modifications are those that are used to apply to simultaneous move games and approaches that integrate models of the opponent (or paired agents for co-operative games).

Simultaneous Move

As described in section 2.5, these are games where multiple players are expected to perform a move at the same time. One of the simplest approaches is to treat the game as a simultaneous move game. However, this is not the only approach for such games Lanctot *et al.*, 2013 compared different methods for applying MCTS to the simultaneous move game *tron*. This comparison included sequential UCT, decoupled UCT, and regret matching, as well as exploring both UCT and exp33 tree policies. They found that the UCB-tuned variants preformed the best and that deterministic strategies outperformed stochastic strategies.

Agent modelling

As mentioned in section 2.7, modelling opponents for tree-search based approaches is a fairly common technique. DECT could be described as a form of opponent modelling, as it is assuming the other agent will make decisions according to UCT

(which attempts to approximate optimal play). As discussed, this approach is common within the literature on tree-search agents.

Given infinite time, UCT will converge to minimax Kocsis and Szepesvári, 2006. Rapid action value estimate, introduced in Browne, 2012 is a domain-dependent heuristic based on the Go all-moves-as-first heuristic. The approach is to update not only the notes that are explored by the tree policy, but also sibling nodes from the tree, allowing value updates for rarely visited nodes. This approach has shown to have a negative effect on agent performance, for example in some positions in board games such as hex (Browne, 2012).

Opponent-modelling has been applied to MCTS in a number of games. Ponsen *et al.*, 2010 applied the technique to Poker using an opponent-model based on learned prior information, and found it performed well against state of the art poker-playing agents. In 2017, Kim and Kim, 2017 noted that the default behaviour for the MCTS agent in the fighting game AI competition assumed random actions and integrated a simplistic opponent model for this real-time competition. Their model consisted of partitioning the possible opponent positions and using observations of the opponent's behaviour in each position to inform the search.

There are a number of team vs individual games which have MCTS applied within games. In other games such as Ms Pac-Man, the game is one (ms. pacman) vs team (the ghosts). In the 2016 version of the competition, agents were provided with a partially observable version of the game and a limited communication channel Williams *et al.*, 2016a. In this game, the ghost team know the strategy of all members of the team as they are submitted as a single controller. This enables a prior degree of collaboration which means that deducing the behaviours of the paired agents and co-operative agent modelling is not needed. Another game, Scotland

Yard, is a board game that has this feature, Nijssen and Winands, 2012 explored integrating modifications for MCTS for imperfect information games, combined with coalition reduction. The approach taken for this game treats the game as a two-player competitive game (treating the team and the opponent as 2 competitive individuals).

4.3 Predictor MCTS

Predictor MCTS is a variant of MCTS which was first presented in (Walton-Rivers *et al.*, 2017). This modification of the algorithm is designed for two or more player games where the agent needs to model the 'social' effects of other agents' behaviour as well as the effects the agent has on the score. As well as a forward model for the game state, the agent assumes it has access to a *policy* for each other agent that describes the behaviour of that agent (the mapping of states to actions).

During the selection and expansion phases, UCB is used for selecting the agent's own moves, but agent *policy* is used for selecting moves made by other agents during the game. As the agent is no longer in control of when a duplicate move is returned, the line between selection and expansion is blurred. When the policy is prompted for a move, if the move has been seen before, the node is treated as previously visited and selection continues. If this move has not been seen before, the agent uses the node for expansion, creating a roll-out from the node. During the roll-out phase the policies are also used for other agents' moves. The moves made by the evaluating agent are still randomly sampled from the legal moves.

The rationale for this is that these moves are not (directly) under the control of the agent. This modification has two effects. Firstly, only actions that will be made by the policy will be evaluated which, for deterministic agents can significantly reduce

the search space. Secondly, it allows the agent to see the results of previous actions on agent behaviour. This allows the agent to explore the influences it has over the other agent's behaviours during the search.

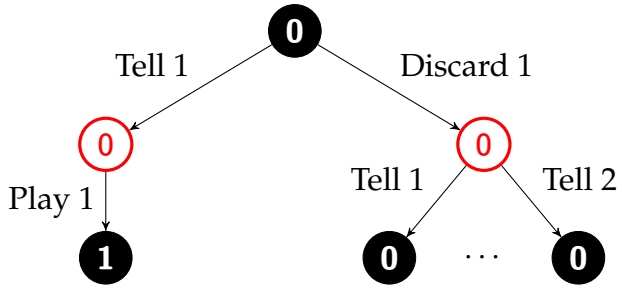


Fig. 4.3 Expansions of Predictor IS-MCTS's tree

.

An example of this behaviour can be seen in fig. 4.3. In this scenario, the agent policy has the behaviour, 'If I have been told about a one, then play it, else, if I have information tokens available, then tell the next player about a card'. The exact specifics of information tokens in *Hanabi*, were discussed in chapter 3; recall that that discarding a card generates an information token. During the agent's turn, it considers two possible moves. Firstly, telling the other agent about the 1 will result in the agent playing the one. This is not necessarily a good move (for example, if playing the one would result in a loss of a life); likewise, discarding a card during the agent's turn will result in the policy telling another player about their cards. This in turn could have knock-on effects for future turns (for example, in a 3-player game where the 3rd agent was following the same strategy, and would therefore blindly play a 1 if told about it).

4.3.1 Agent Policies

The predictor treats the policy like a 'black box', It does not ever know the internal state of the policies it is using, just a possible move that they will make when

prompted to do so with a given state. Note that the rule-based agents presented in this thesis do not have any internal state (they are entirely reactive).

Providing that the policy can be cloned (and starts the game with a predictable internal state) it is possible to extend the forward model to these agents. The process to do this is relatively straightforward: whenever a move is made in the real game, apply the same move to the model. During an iteration, create a clone of the policies, applying moves made during the iteration to the clone. When the iteration is complete, discard the clone. This is the same process followed for the forward-model of the game. It makes the assumption that the moves made by an agent are observable, which is a reasonable assumption for most games (and in fact, features quite heavily in human-centric *Hanabi* strategies). Even if the actions taken are not observable, it may be possible to re-construct them based on observations, however this is outside the scope of this thesis.

4.4 Genetic Algorithms

Genetic algorithms are group of techniques which have found widespread adoption in a range of domains. In these algorithms, a *population* of candidate solutions are re-combined to create new solutions based on two or more individuals.

Within games, their use has been varied, they have been used both for controlling AI in games both directly Pérez-Liébana *et al.*, 2013, and indirectly by evolving other controllers Pérez-Liébana *et al.*, 2011. They have also been used for purposes other than controlling characters, for procedural content generation (Togelius *et al.*, 2011).

Evolutionary computation has a very long history in computer science, and the idea of simulating evolution on computers was already being evaluated as early as the

1950s Fogel, 2006. This technique is inspired by evolution present in nature. With 'survival of the fittest' approaches being adapted based on the understanding of these ideas. The algorithm works by creating a population of candidates, which are then evaluated using a fitness function and the some proportion of these are selected to be recombined into new candidates.

The initial population can be created either at random or based on some pre-defined system. The population is then evaluated using a domain-dependant fitness function, which assigns a score to each individual. These are then selected to be recombined using genetic operators to form new solutions. After each generation, the steps are repeated. The algorithm terminates when a termination condition is met (for example, number of generations without any progress, when an individual reaches a pre-defined fitness, time limit, or after a fixed number of generations).

4.4.1 Selection Methods

There are numerous ways to select individuals for recombination (Blickle and Thiele, 1996). Some common methods are *tournament selection* and *fitness proportional selection*.

In tournament selection, (Miller and Goldberg, 1995) a subset of the population is selected at random; they are then evaluated and ranked. The winner is the individual with the highest fitness. Tournament selection doesn't require that the whole population be evaluated for fitness (only those chosen to take part in the tournament are evaluated). The size of the tournament can be varied depending on domain requirements. The importance is the rank order of the fitnesses rather than their explicit values. One tournament is run per required (new) member of the population.

In fitness proportionate selection (Goldberg and Holland, 1988) (also referred to as roulette wheel selection), the solutions are given a probability of selection proportional to their fitness, equally fit individuals are equally likely to be selected. This gives each individual a chance to be selected for reproduction. The importance is the fitness values themselves, with two individuals with similar fitnesses being given similar probabilities of selection.

For simple generic algorithms, tournament selection is better than roulette wheel selection (Zhong *et al.*, 2005).

4.4.2 Genetic Operators

Recombination of the material is performed using *crossover*, which can be either single-point or multi-point crossover. n-point crossover dates back to at least 1957 work of Fraser (Fogel, 2006). In single-point crossover, a point in the two sample solutions is chosen at random, everything before this point is copied from one parent and everything after this point is copied from the other. In multi-point crossover, multiple points are selected at random and used for recombination.

	s_1	s_2	s_3	s_4	s_5
Parent 1	1	2	3	4	5
Parent 2	A	B	C	D	E
Child	1	2	3	D	E

Table 4.1 An illustration of single point crossover

In order to ensure diversity, mutation may also be used. This consists of modifying the values which make up a candidate solution and changing it to new value. Again this can either be done to a single point in the genome (selected at random) or multi-point (going through the genome and mutating each value with some probability).

To ensure that good solutions are not lost, *elitism* can be used, which is the practice of copying good individuals into the new population without recombination or mutation.

There are modifications of these strategies to work in constrained domains, such as the limitation that part of the genetic sequence cannot be used twice. This is useful in solving problems such as the travelling salesman problem (Abdoun and Abouchabaka, 2012).

4.4.3 Rolling Horizon Evolutionary Algorithms

In addition, techniques based on genetic algorithms have been used on-line, such as Rolling Horizon Evolutionary Algorithms (RHEAs) (Pérez-Liébana *et al.*, 2013). These have seen limited success in GVG-AI (see section 2.6), but recent work on modifications to them looks promising (Gaina *et al.*, 2017a; Gaina *et al.*, 2017c).

These variants work by evolving a sequence that represents moves to be made by the agent. This is then used to advance a forward model, similar to roll-outs performed by MCTS. The algorithm maintains a population of candidate solutions, which are recombined and mutated as described in the above sections. When some termination criterion has been reached, the first move from the best candidate seen so far is used.

Macro Actions

To give the agent more, 'thinking time'. The process can be modified to use *macro actions* (Pérez-Liébana *et al.*, 2013). Each action is executed some predetermined number of times, and during macro-action execution, the search is continued from the expected state after completion of the macro action.

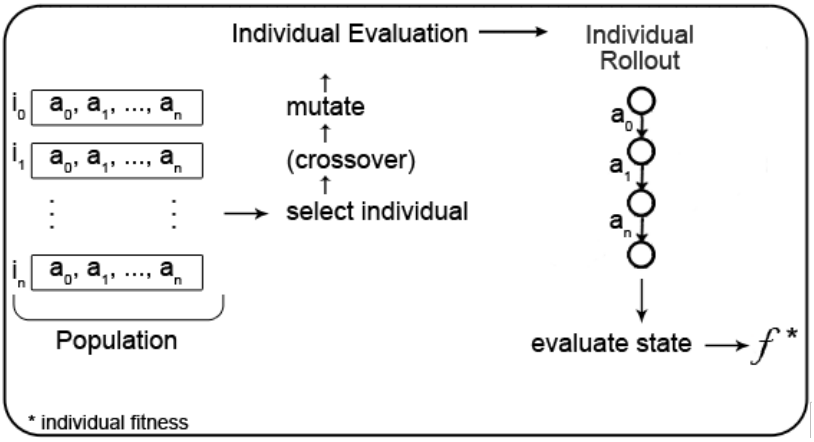


Fig. 4.4 The operation of RHEA, Gaina *et al.*, 2017a

Part II

Contributions

Chapter 5

MCTS applied to Co-operative problems

This chapter presents background work relating to the performance of Monte Carlo Tree Search (MCTS) in the co-operative puzzle domain *Tiny Co-Op* (described in section 3.2) that forms the basis of the work presented in chapter 6.

The experiment described provides a baseline for agent performance when no modelling is used and the agents cannot explicitly communicate with each other. It shows the relative performance of the two approaches common in general video game playing (MCTS and evolutionary algorithms, both described in chapter 2) which informs the choice to use MCTS as the basis of the prediction modifications that are presented in later chapters.

This work is draws largely on the work presented in Piers R. Williams, Joseph Walton-Rivers, Diego Perez-Liebana, and Simon M. Lucas (Sept. 2015). "Monte Carlo Tree Search Applied to Co-operative Problems". In: *CEEC2015 - IEEE Conference*

on Computer Science and Electronic Engineering. IEEE CEEC. IEEE Computer Society, pp. 219–224.

5.1 Introduction

This chapter deals with a third kind of co-operative domain, here, ?? ?? of the first two. In the first kind, -; in the second kind -; in this third kind, the agents have a symmetric action set but need to co-operate in order to overcome some environmental obstacles. This does not require partial observability, complex communication or the creation of opponents for the characters to face and as a result presents a comparatively simple domain for studying the effects of modelling on the agent's performance.

This chapter introduces experiments conducted in the *Tiny Co-Op* domain, described in section 3.2. This acts as a benchmark to assess agent performance in a domain which features co-operation but does not include player modelling nor explicit communication actions. This helps to establish whether these algorithms are capable of solving simple co-operative puzzles without the need for complex modelling of agent behaviours.

5.2 Experimental Set-up

To evaluate the agent's performance, the agents were given a set of puzzles which require co-operation. To evaluate how effectively the agents could perform in the domain without co-operation being required, a version of the first level with the co-operative component removed was included. This level is known as Pathfinding within the map set. The full map set used was described in section 3.2.

During each game tick, each agent is permitted to make a single move. If the agents moves conflict (for example, if they both try to move into the same square) then the agent whose ID is the lowest (player 0) will occupy the space and the second agent will be prevented from moving. Agents are permitted to move in one of the four cardinal directions, or to remain stationary.

The agents are provided with access to a forward model, which allows them to simulate the effects of actions from the current time step onwards. To use the model, moves for both agents must be provided. The model provides very limited access to internal game state, permitting the agents only to see the score that they will achieve after a given move has been made. All the agents implemented for this experiment assume the other agent makes random moves when using the forward model.

The experiment consisted of a round robin in which, each agent was evaluated over all 7 maps. The agents played the greatest possible number of games given the computing resources available at the time of running the experiments to ensure accuracy and consistency in the results, resulting in 47 repeats. This number was chosen for pragmatic reasons based on machine availability. For each game played, the agents, level, score obtained and number of game ticks required to solve the level were saved into a text file and then processed after the experimental run had finished. The scores for both games played with the same strategy and games played with all strategies were then reported. The results per level were also summarized – to show the different skill levels required for the levels to be solved.

The agents in this experiment did not have a limited time budget in which to make moves. Instead, all agents were given a fixed iteration count. This was to allow the different parameters of MCTS to be evaluated. The game could theoretically last forever, as the agents could either stay in one location or move in circles within an

area without ever reaching the goals. To prevent this, the game engine imposed a 2,000 tick time limit, after which time the game was terminated and the current score of the agents was used. If this happened, the time limit was stored as the time taken (2,000).

5.2.1 Controllers

The experiment consisted of two approaches which have shown promise within game playing research: Genetic Algorithm (GA) and MCTS. Different parameters for the MCTS-based agent were presented in order to gauge how these affect agent performance. The GA controller created for the Phyiscal Travelling Salesman Problem (PTSP) competition was included for comparison. There were six controllers used; GA controller, VariGA, MCTS (three variants) and Random. These are discussed below.

GAController

The GA controller is a modified version of the GA controller from the PTSP competition (Pérez-Liébana *et al.*, 2012). The controller uses a genetic algorithm to evolve move sequences which can be used by the controller. The sequences feature 'macro actions' which means each evolved action is repeated 3 times before the next action is executed. During evolution the controller creates action sequences which are 15 actions long; this gives the agent a look-ahead of approximately 45 actions (15×3).

The GA uses the macro actions to increase its 'thinking time', as it can effectively spend 3 turns worth of computation to evaluate generations before needing to return a decision. The agent uses tournament selection to choose individuals for crossover, with a tournament size of three. The algorithm also includes elitism and will copy the best two individuals into the next generation.

VariGA

The VariGA controller attempts to evolve action sequences, but rather than use a fixed-length sequence it uses a variable-length sequence. This agent has the advantage of being able to make small movements when needed to avoid overshooting.

MCTS

This is a fixed-iteration variant of MCTS which treats the co-operative domain as a single-player domain. As the domain is a simultaneous-move game, the actions of the paired agent within the search will be made randomly.

The implementation uses fixed iteration counts rather than time budgets so that different machine speeds will not affect the evaluation result.

In order to see how different values affect the performance of the MCTS agent in this domain, three variants of the agent were used. These correspond to different roll-out depths and iteration budgets used by the agents; these values are presented in table 5.1.

The exploration constant used for the agent was $\sqrt{2}$.

Table 5.1 Parameters for the MCTS agents

Budget	Iterations	UCT search limit	Rollout border
low	75	3	15
medium	200	5	30
high	500	10	45

VariGA

Random Controller

The random controller uniformly selects from the list of possible actions and returns one for evaluation. Note that the agent does not check that a movement action would result in the agent's staying still, so if the agent walks into a wall the move is the equivalent of a no-op move. This controller represents a poor player and can be used to benchmark the performance of the other agents.

5.3 Results

The results are presented in two different ways: the full round-robin first, followed by the subset of the games where the agents were paired with copies of themselves. An analysis of the results is provided after the results have been presented.

The scoring system for the game is described in section 3.2. Scores are bounded between zero and one. A score of zero in a game indicates that the agent did not manage to visit any goals during the game, a score of one indicates that every agent visited every goal (at which point the level terminates). The agents are given 2,000 ticks (game move pairs) to complete the level. If they fail to solve the level they will be awarded 2,000 ticks. Lower tick counts indicates the agents solved the levels on average faster.

Three types of result will be presented. Firstly, the average scores obtained, secondly the number of moves required to solve the level and finally the number of games where the agents successfully solved a level. Both the average number of moves and average game score have been rounded to two decimal places.

5.3.1 Round-Robin

First, results for the full round-robin dataset will be presented. Table 5.2 shows all combinations of agents across all maps. This table shows both the average score and the average moves taken. The $\pm$ columns indicate 95% confidence intervals.

Table 5.2 shows that the best performing agent overall was the MCTS high agent. This managed to achieve an almost perfect score when paired with itself (m=1.00 $\pm$ 0.01, std=0.27) - successfully completing the level (99% of the time), an extremely high score when paired with the medium budget player, both as the first player (m=0.84 $\pm$ 0.04, std=0.34) and as the second player (m=0.90 $\pm$ 0.03, std=0.82). It also managed to obtain the best scores of any agent when paired with it (except in the case of MCTS low where it was tied with MCTS medium and results do no show a significant difference between the agents). The GA and VariGA agents performed poorly when paired with each other obtaining scores scores to that of random play. The results also show that performance of the MCTS agents is in order of the budgets provided to them (high performed best, followed by medium with low in last place). The random agent's performance is largely dominated by the performance of the paired agent.

5.3.2 Mirror Games

The second set of results presented are the subset of games where agents are paired with themselves (ie, $\pi_1 = \pi_2$). This indicates whether the agent is capable of solving the puzzles within the turn limit without being limited by the other agent's performance. The results can be used as an estimation of the relative strength of the algorithms presented. Again the results shown for score are the average score across all games, with error bars indicating 95% confidence intervals.

The graphs presented in this section will be subsets of the games, to illiterate findings from the experiment. Performance aggregated across all maps can be see in the round-robin table (table 5.2).

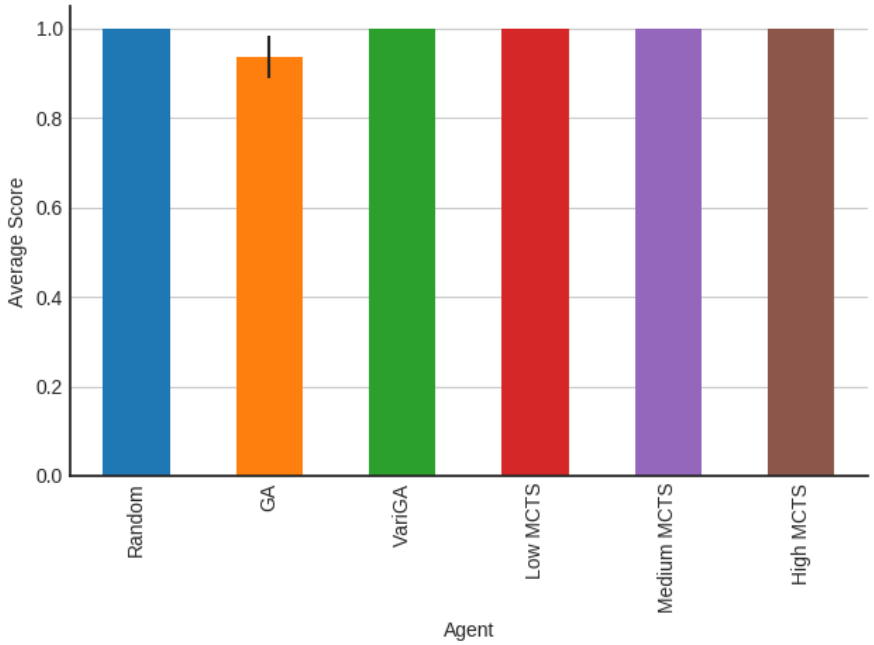


Fig. 5.1 Average scores obtained in the *pathfinding* level when paired with identical agents

Figure 5.1 shows that the only agent that was not able to obtain a perfect score when paired with itself on Pathfinding (the level which features no doors) was GAController. Figure 5.2 show the average number of moves required to solve the level. All agents which obtained a perfect score were able to solve the level in less than half the time allocated (ie, sub 1,000 game moves). The random agent performed significantly worse than all agents that obtained a perfect score. The MCTS variants were quicker than VariGA, but the results between the MCTS and it are very small, the difference between the VariGA and the low budget MCTS agent is significant ($t=-2.235$, $p<0.05$). The error bars indicate that there is a significant difference between the agents' performance.

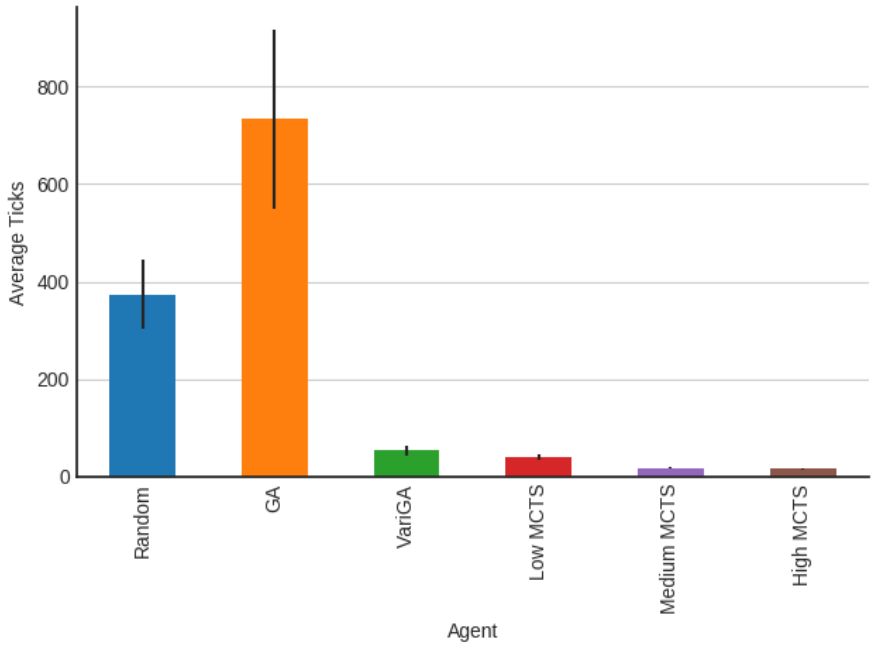


Fig. 5.2 Average turns taken to complete the *pathfinding* level when paired with identical agents

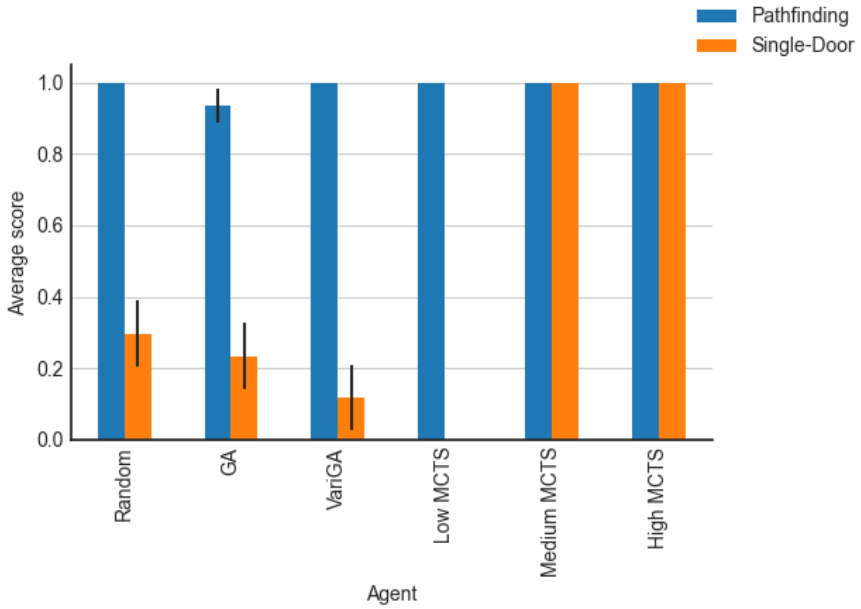


Fig. 5.3 Average scores obtained comparing pathfinding and single-door

Figure 5.3 shows a comparison between the Pathfinding and Single-door levels. Although the results presented above show that that most agents were able to solve the Pathfinding level without the need for co-operation, agent performance for the agents decreases when the door is added. Both the medium and high budget MCTS agents maintained their perfect score (solved the level 47 times out of 47 repeats), but the low budget agent did not manage to obtain a score during the Single-door level (obtained a score of zero over all 47 repeats). The VariGA managed to solve the level 5 times out of the 47 repeats played. Random managed to solve the level 4 times out of the 47 repeats played.

Table 5.2 Round-Robin mean scores for agents, a score of 1 indicates the level was solved perfectly every time.

Agent 1	**Agent 2**	**Score**	±	**Moves**	±
GAController	GAController	0.46	0.04	1783.94	56.73
	MCTS Medium	0.77	0.04	1228.77	79.14
	MCTS High	0.93	0.02	808.60	64.01
	MCTS Low	0.50	0.04	1612.52	72.55
	Random	0.50	0.04	1713.45	65.18
	VariGA	0.48	0.04	1615.57	73.82
MCTS Medium	GAController	0.67	0.04	1573.19	68.44
	MCTS Medium	0.82	0.04	849.67	86.64
	MCTS High	0.90	0.03	669.10	79.26
	MCTS Low	0.51	0.04	1488.56	86.54
	Random	0.77	0.03	1350.95	78.63
	VariGA	0.75	0.04	1090.14	85.17
MCTS High	GAController	0.81	0.03	1339.84	73.73
	MCTS Medium	0.84	0.04	707.96	83.71
	MCTS High	1.00	0.01	233.19	33.87
	MCTS Low	0.53	0.04	1499.09	85.02
	Random	0.93	0.02	938.44	68.70
	VariGA	0.87	0.03	808.29	78.55
MCTS Low	GAController	0.39	0.04	1775.84	55.97
	MCTS Medium	0.47	0.05	1494.78	84.81
	MCTS High	0.47	0.05	1495.80	85.37
	MCTS Low	0.38	0.04	1632.95	77.07
	Random	0.42	0.04	1694.67	67.99
	VariGA	0.42	0.05	1592.50	78.93
Random	GAController	0.39	0.04	1796.17	56.02
	MCTS Medium	0.76	0.04	1276.14	78.15
	MCTS High	0.92	0.02	868.13	64.71
	MCTS Low	0.48	0.04	1675.43	69.51
	Random	0.43	0.04	1740.28	63.31
	VariGA	0.46	0.04	1661.77	71.27
VariGA	GAController	0.37	0.04	1784.37	56.89
	MCTS Medium	0.74	0.04	1098.75	86.51
	MCTS High	0.85	0.03	868.40	80.38
	MCTS Low	0.47	0.04	1586.21	79.10
	Random	0.43	0.04	1705.85	66.57
	VariGA	0.34	0.05	1624.38	77.88

Split by levels

When the mirror games are separated by levels, the difficulty of the levels can be examined in more detail.

Table 5.3 Average scores obtained from mirror games across different maps, ± indicates 95% confidence intervals

	GA		MCTS			
	GA	**VariGA**	**High**	**Medium**	**Low**	**Random**
Airlock	0.17	0.00	0.97	0.11	0.00	0.09
±	0.08	0.00	0.04	0.09	0.00	0.05
Butterfly	0.39	0.13	1.00	0.79	0.25	0.39
±	0.05	0.04	0.00	0.05	0.00	0.04
Extended Side	0.31	0.10	1.00	0.88	0.00	0.30
±	0.07	0.14	0.00	0.06	0.00	0.08
Pathfinding	0.94	1.00	1.00	1.00	1.00	1.00
±	0.05	0.00	0.00	0.00	0.00	0.00
Side by Side	0.61	0.62	1.00	1.00	0.58	0.64
±	0.04	0.06	0.00	0.00	0.03	0.10
Single-door	0.23	0.12	1.00	1.00	0.00	0.30
±	0.09	0.09	0.00	0.00	0.00	0.09
Sym. Single-Door	0.55	0.40	1.00	1.00	0.86	0.32
±	0.09	0.14	0.00	0.00	0.08	0.10

As previously presented, Pathfinding proved to be solvable by both groups of algorithms (GA and MCTS). Of the levels that required co-operation between the agents, Airlock proved to be the most problematic for the MCTS-based agents. In this level, the lowest budget MCTS agent did not manage to obtain a score at all, and the MCTS medium agent performed extremely poorly, returning an average score of 0.11 ± 0.09. This was also the only level where the high-budget MCTS failed to perfectly solve the level. Although the high-budget agent was able to perfectly solve Butterfly and Extended Side, the medium agent showed a decrease in performance on these levels.

The low-budget variant was unable to perfectly solve Symmetric Single Door and Side by Side, but still was able to obtain a partial score; both of these levels were solved correctly by the medium-budget variant. The two GA-based agents follow the same trend as each other, finding the Pathfinding level easiest, followed by Side by Side. On most maps, GAController performed better than VariGA. With VariGA only having a significant advantage on Pathfinding.

Table 5.4 Moves taken by agents in mirror games across different maps

	GA		**MCTS**			
	GA	**VariGA**	**High**	**Medium**	**Low**	**Random**
Airlock	1953.60	2000.00	714.57	1958.47	2000.00	2000.00
±	73.86	0.00	146.13	41.82	0.00	0.00
Butterfly	2000.00	2000.00	409.23	1786.40	2000.00	2000.00
±	0.00	0.00	47.42	106.35	0.00	0.00
Extended Side	1984.17	2000.00	234.70	1254.77	2000.00	1990.51
±	30.70	0.00	36.15	166.83	0.00	13.97
Pathfinding	732.98	53.77	16.06	17.49	39.92	374.21
±	182.76	70.45	1.22	1.30	6.62	10.03
Side by Side	2000.00	1820.32	107.30	366.02	2000.00	1969.15
±	0.00	41.88	12.14	63.93	0.00	115.50
Single-door	1950.21	1902.53	115.79	445.15	2000.00	1959.11
±	52.77	42.03	12.39	64.63	0.00	89.82
Sym. Single-Door	1866.61	1594.06	34.70	119.38	1390.75	1888.98
±	93.55	95.10	3.63	17.47	153.98	173.00

The results in table 5.4 show that across all levels that feature co-operation the high budget MCTS agent performed the fastest. On levels where medium-budget MCTS also obtained a perfect score (Side by Side, Single-door, Pathfinding and sym. single-door) the high-budget agent solved the levels faster. Often significantly faster, with the exception of Pathfinding where there is a very small difference between the scores (which, based on the confidence intervals, is not significant). The low budget agent obtained the worst possible score (2000) on most maps. This indicates that it

was not able to successfully solve the maps. The GA-based agents also performed poorly, with scores close to the limit of 2,000 indicating that the levels were often unsolved. The results for random show that even though it managed to obtain a partial score on the levels, it did not manage to solve the levels once.

Table 5.5 Number of perfect score games across maps (n=47)

	GA		MCTS			
	GA	**VariGA**	**High**	**Medium**	**Low**	**Random**
Airlock	2	0	45	5	0	0
Butterfly	0	0	47	17	0	0
Extended Side	1	0	47	35	0	2
Pathfinding	41	47	47	47	47	47
Side by Side	0	11	47	47	0	2
Single-door	4	5	47	47	0	4
Sym. Single-Door	12	19	47	47	37	6

Table 5.5 shows the the total number of perfect score games each mirror pairing managed to obtain across the levels. There are no confidence intervals for this data because it is simply a count of the number of perfect score games in the dataset. As the score is largely based on the win rates, this data closely matches the data presented in table 5.3. The table shows that the low budget agent was only able to solve games in two of the levels (with the rest of its score coming from partial solutions).

5.4 Discussion

The results show that the MCTS agent is better suited to solving the levels than the GA agent. With the GA agent being outperformed in the by the high budget MCTS agents consistently. The low budget agent struggled with most of the levels. This indicates that it was probably not exploring enough of the tree to see rewards and so was essentially acting randomly. This is a known problem with MCTS if it does

not have enough time to search the state space. With the two MCTS agents that had higher iteration counts and longer roll-outs most of the levels were solvable.

The GA-based agent showed some difficulty with the task; this is likely due to the macro actions resulting in the agent overshooting the goal. (fig. 5.2 shows the GA performed poorly). The average result is affected by the games in which it did not manage to complete the level (as the agent is noted as having taken the maximum number of moves in these cases). These account for 6 of the 47 games played. If they are removed, the average obtained by the agent is 547.56. This is still worse performance than Random.

The results for MCTS show that providing higher budgets meant that the agent was capable of solving the levels in fewer moves and more reliably. This is to be expected as the higher-budget agents can see further into the future and explore more moves. The two levels designed to require the highest level of co-operation, Airlock and Butterfly, both proved challenging for the agent. This implies that maps that require more co-operation between the agents are harder to solve. The other level which proved challenging for the medium-budget agent is Extended Side, this is the largest level ($15x9$) and therefore has the largest number of possible states. However, size alone cannot explain the performance of the MCTS agent, as although Butterfly requires complex co-ordination between the agents, it is also the smallest level tested ($7x8$). These two factors combined (large size and the need for co-operation) may help explain why Airlock is so challenging.

The performance of the agents in the four $9x9$ levels is interesting. 'Single-door' on page 74 and Pathfinding follow the same design but the latter has the door and buttons removed. As a result, this level gives an indication of the complexity added by the co-operative mechanics. The optimal path is longer for Single-door as it

requires one of the agents to visit the button before going through the door (and relying on the paired agent to open the door for it on the other side). Extended Side has the same features, but arranges them differently so that the doors and goals are closer together and therefore requires less look-ahead to achieve a score. This is reflected in agent performance, as the low-budget agent is able to obtain a score in Extended Side but not in Single-door. This implies that one of the reasons why the co-operative levels are more difficult is because they require longer-term planning. In Side by Side, each agent starts in a room with access to a goal, because of this, it is possible to obtain a score of 0.5 without interaction of the other player. The agents also start extremely close to the goal, which means a look-ahead of only a few moves should result in a score. This is reflected in the results presented in table 5.3, in which all agents were able to obtain at least 0.5 on average. However, only the High and Medium MCTS agents were able to obtain perfect scores in this level, which indicates that the other controllers did not reliably open the door for the other agent.

As most of the agents were able to solve the level without the need for co-operation (Pathfinding), it can reasonably be concluded that the need for co-operation is driving factor for the difficulty of the environment. The two levels which feature the largest amount of co-operation between the two agents (Airlock and Butterfly) proved to be the hardest for the medium MCTS agent to solve in comparison to the high budget version. Both these levels feature multiple points where the agents must interact with each other, in Butterfly the agents must be let in and out of the rooms in turn, (4 points at which the agents must co-ordinate) and in Airlock the top agent must be let through two doors before letting the other agent into it's area to get the goal. This level also has a sparser reward space, in Butterfly letting the

agent into each room generates a reward (as there is a goal in each of the smaller rooms), in Airlock there is only a single goal located at the end of the level.

A possible reason for this is the assumption that the paired agent is moving randomly in the simulations. Table 5.4 shows that even though a random agent will eventually reach a goal, on average for the path-finding level this took 347.21 moves (when both agents are random). This implies that a relatively long lookahead is needed in order for the agent to reach the goal (for most of the co-operative levels, this should be less because the area in which the agent is located is smaller and only one agent is moving randomly).

This also will cause problems when the agent needs to go through a door to obtain a reward. The accessibility of the reward (goal) depends on the location of the paired agent (they must be standing on the button). An intelligent paired agent might stand on the button, keeping the door open and resulting in the problem being similar to that of the Pathfinding level (navigate to the goal). However, is the paired agent is assumed to be random, it is unlikely that they will remain standing on the button, resulting in fewer states where the goal is obtainable. This will also make the state evaluation noisy: standing by the door and moving into the space the closed door occupies will sometimes lead a reward (if the door is open), and sometimes will not. This makes the already quite sparse rewards even sparser.

Therefore, obtaining a reward for the Random agent's behaviour in the roll-outs is unlikely and having longer look-aheads (higher roll-out border) and exploring more moves (higher iteration count) should be beneficial. Using a more intelligent strategy than random for paired agent performance should also result in better performance as rewards will be more frequent and predictable (this is explored in chapter 6).

5.5 Conclusions

This chapter showed that adding the need for co-operation to the levels increased the complexity of the task for the game tested.

Even without doing any intelligent modelling of the agent, the high-budget MCTS agent was able to complete most of the co-operative levels correctly. The agents are completely driven by the utility of their (and their opponent's) actions on the resulting state, they had no knowledge that the domain featured another agent, only that the evaluation was noisy.

Although this technique is the same as the stock 2-player MCTS agent provided with the General Video Game AI (GVG-AI) two player competition, there are approaches which are specifically designed working in simultaneous move games (Lanctot *et al.*, 2013). Although the work presented in this section is promising, further analysis of the performance of MCTS for simultaneous move games should be conducted to ensure these findings are generalizable. In addition, GA-based agents are sensitive to parameters (Gaina *et al.*, 2017a) and so a more detailed study of the effects of the parameters on the GA-based agents in Tiny Co-Op should be conducted. Although very important future work, both of these experiments are not required for comparing the effects of modelling *vs* assuming random play.

The MCTS agents performed well in the domain when paired with the other agents. Even though these agents were not taking into account the behaviour of the other agents (and were assuming they moved randomly) the agents were able to complete most of the level. The agents are largely unaware they are playing a two player game – they make a move and the paired agent's move is made for them. This may pose a problem if the agents must co-operate with each other in a more complex

way such as integrating communication (as required by team games such as *Dota 2*, Font and Mahlmann, 2018).

Both GA agents performed poorly on the co-operative tasks and were not capable of solving the puzzles as reliably as the MCTS agent. This could be attributed to the macro-actions making fine control difficult for the agents. This indicates that the MCTS agent is a better choice for modification as it already has good performance.

This chapter has shown that *Tiny Co-Op* is a suitable game for co-operative experiments. The levels designed represent a range of skill, from 'no co-operation required' to 'not solvable with a reasonably large iteration budget'. The performance of the MCTS agent shows that there is skill involved in solving the levels. This can be seen as increasing the planning horizon results in the levels becoming more solvable (table 5.5), with the three MCTS agents representing a low, medium and highly skilled player.

The non-random agents in this experiment are all actively trying to seek out rewards. This is a reasonable assumption when playing with Artifical Intelligence (AI) players, but human players will often communicate about strategies during game-play. For the agent to be effective in group scenarios the agent should be able to cope with agents which rely on communication. In Chapter 6 the effects of introducing communication and a form of modelling that takes into account the effects of communication actions of behaviour of the paired agent is presented. Chapter 6 answers two questions: firstly, does including an intelligent modelling technique enhance agent performance in *Tiny Co-Op*?; and secondly, for agents that rely on communication, is this modelling necessary?

Chapter 6

Applying Communication Actions to *Tiny-Coop*

This chapter explores the effectiveness of modelling agent behaviour as part of a modified version of the best-performing algorithm in the experiment presented in chapter 5.

To allow the agents to communicate, and to provide the opportunity to create more complex strategies, section 6.2 explores two different kinds of communication: the richer, more expressive *full* action set and the more limited *relative* action set. The experiments show that the full action set is not suitable due to the increased branching factor it introduces. The results of this experiment resulted in the choice to use the *relative* action set for the prediction experiment.

For the prediction experiment (section 6.3), a new agent is introduced that will only move if told to do so. This agent should present a problem for agents that do not reason about the behaviour of other agents. This is a simple form of strategy that requires explicit co-ordination between the agents. The experiment indicates that

prediction has an effect on the agent's performance – the predictor variants achieve better scores in less time than the non-predicting version with the same parameters.

This chapter introduces a modification of the *Tiny Co-Op* domain presented in the previous chapter. It introduces a modification of the best-performing algorithm, Monte Carlo Tree Search (MCTS), that includes a modelling technique to reason about the actions of the other player. Introducing an opponent model is a common technique within competitive games (see section 2.7). This work will aim to see if such a technique is useful in an environment such as *Tiny Co-Op*

As none of the existing agents make use of communication, a new agent which explicitly relies on communication to co-ordinate its actions is created. This chapter shows that introducing agent modelling increases the performance of the MCTS agent. It also highlights one limitation of the strategy used for doing so – it is not possible to use the technique with computationally-intensive agents. This limitation is addressed in chapter 8.

6.1 Introduction

Chapter 5 showed that even without agent modelling, the MCTS agent could solve simple co-operative problems presented to it. In the experiments presented in chapter 5, both the MCTS and Genetic Algorithm (GA) agents assumed random moves were being made by the paired agent. This chapter explores the effects of using a more accurate prediction of paired agent behaviour.

From the results presented in chapter 5, random play performs poorly but is still able to obtain a score for most levels. The agent represents an non-optimal player which is still able to obtain a score if directed towards high-value areas of the level.

As the scores for both agents is shared, once MCTS has visited all available goals then the only way to increase its score is for the paired agent to reach the goals. If the goal and random agent are in different rooms (*i.e.* the goal is behind a closed door), MCTS needs to ensure the door is open. When performing a search any state in which MCTS is standing on the button has the possibility of the random agent moving onto the goal: all states where MCTS is not on the button will generate no change in score. If the random agent is in the same room as the goal and MCTS is standing on the button, there is a possibility that the random agent moves out of the room with the goal in, decreasing the likelihood that the random agent will come into contact with the goal (and therefore resulting in fewer rewards for MCTS). As a result, the best policy for MCTS is to move off the button to prevent the agent from leaving the room. These two behaviours combined create the emergent behaviour that MCTS will open doors to let agents into rooms with unvisited goals, but shut the door behind them. This 'herding' behaviour is beneficial for MCTS when paired with a poor player, but could result in the agent wasting time when paired with an intelligent player, or paired with an agent following a set strategy.

This poses an important question, is modelling a co-operative player as a poor player enough, or is an *accurate* model of the paired player required? To answer this question a player which is capable of playing the game, but does not act like the random player is needed. As the original *tiny co-op* is limited in scope, there is not much room for writing such an agent. It would be possible to model the player as an *intelligent* player (such as performing a search on the other player's turn), but doing so would only provide data about similar strategies. In games where there are multiple valid strategies and require agent co-ordination (this is explored further in section 2.8).

This chapter explores the effect of adding communication actions to the game. These are actions that have no real affect on the state itself, but will allow the agents to co-ordinate their behaviour. As the approaches presented in chapter 5 do not make use of anything but the score, this experiment required the creation of new agents that use the communication actions as part of their decision-making process. This chapter also introduces a modified version of the MCTS player which is specifically designed to take the effects of the communication actions into account when making decisions (providing a comparison between a poor but functional agent model and an accurate agent model).

In order to facilitate the modification of game and the creation of agents which use communication actions, a new variant of the *Tiny Co-Op* domain was created which was designed to allow exposure of the game state. As the work presented in chapter 5 did not use any heuristics or have agents that needed to examine the current state in any detail, the agents only had access to the current score and the ability to forward the game state. The full details of these modifications were discussed in section 3.3.

6.2 Exploring the Effect of Communication Actions

To explore the effect of the increased branching factors on the agents, a preliminary experiment was run which evaluated the three action sets. A smaller set of maps are used for the evaluation in order to speed up evaluation. The maps (Pathfinding, Single-Door, Airlock and Butterfly) represent the range of map complexity that was present in the original experiment.

The agent set consisted of the three MCTS variants, the GA controller and the random controller. The VariGA controller was excluded as the implementation was

too tightly coupled to the original framework and action set. In order to make use of it in the new environment it would need to be rewritten and the performance shown in the co-operative levels in chapter 5 did not justify this given the focus of this work. The games were executed as a round-robin tournament which consisted of each agent playing 25 repeats as both the first and second player. This was chosen as an estimate of how many repeats would be reasonable to balance the computing time required to run the experiment with obtaining an acceptable number of repeats.

The agents were not time-limited in this experiment, they were limited by iteration count. The same values were used as the original *Tiny Co-Op* experiment presented in chapter 5. As in the original experiment, the agents were given 2,000 turns to complete each game: when this limit was reached the agents were recorded as having failed to complete the level. The time taken was recorded as 2,000 ticks and the score obtained so far by the agents was reported as their final score. This results in a score of less than one indicating that the agents failed to complete the level in 2,000 ticks or less.

All experiments were run on the Intelligent Games & Games Intelligence (IGGI) compute cluster, which contains 8 Xeon E5-2680 v3 CPUs and 126GB of RAM. As each 'job' is singled-threaded, the cluster can execute a maximum of 192 jobs simultaneously. Queues jobs are executed after another has finished.

The full result set consisted of 15,000 games on the new version of the engine and took approximately a week of compute time. The final score obtained by the agents, the total turns required to complete the level, and the CPU and WALL times are recorded by the engine and saved to CSV files for processing.

6.2.1 Communication Strategies

The two communication strategies outlined in section 3.3 (full and relative) were added to the movement actions outlined in the original paper (up, down, left, right and no-op). In total this provides three different action sets. The full action set allows the agents to ping any square in the level (similar to the 'flare' or 'map ping' features on Real-Time statergy games (RTS) and Multiplayer Online Battle Arena (MOBA) games). The relative action set allows the agents to communicate a direction (left, right, up or down) to the other agent and the no-communication action set is the same set used in the original *Tiny Co-Op* experiments, providing a baseline. These offer a way to evaluate the trade-off between the increased expressiveness of the action set and the corresponding increase in branching factor. These were described in detail section 3.3; below is a brief overview of each of the action sets.

Full Communication

This action allows an agent to communicate any square in the level as a position. This is the most expressive set; the number of communication actions is much larger than the number of movement actions. The number of communication actions added to the game is equal to the number of squares in the level ($a_{com} = widthxheight$, the sizes of the levels are provided along with their descriptions in section 3.2).

Relative Communication

This action set allows agents to communicate a direction to the other agent. This provides a balanced number of communication and movement actions (4 of each), with the addition of a no-operation move. This brings the total number of actions for this action set to 9 (4 communication, 4 movement, 1 no-op).

No Communication

This is the default action set, and is provided for comparison with the other two sets. Including this set allows for fair comparisons as it removes any differences as between the two versions of the engine.

6.2.2 Evaluated Agents

The agents used for this experiment are:

- MCTS High
- MCTS Medium
- MCTS Small
- GAController
- Random Controller

The budgets used for the MCTS agents and the parameters for the GA agent remain unchanged from the original *Tiny Co-Op experiment*.

6.2.3 Results

In this section, the results are shown for the preliminary co-operation experiment. Table rows that feature $\pm$ are describing 95% confidence intervals. The results presented are the 'mirror' subset (games where agents are paired with themselves).

The results presented in table 6.1 show that the agents performed worse on the full and relative action sets than on the simple set. For the medium agent, the agent was not able to obtain a score at all. For the MCTS medium agent, there is a significant

Table 6.1 The results of agents paired with themselves over all maps, (n=200), ± indicates 95% confidence intervals

Controller	Full	Relative	Simple
MCTS High	0.11	0.77	0.85
±	0.03	0.05	0.05
MCTS Medium	0.00	0.68	0.79
±	0.00	0.06	0.05
MCTS Low	0.00	0.54	0.73
±	0.00	0.06	0.05
GAController	0.17	0.55	0.61
±	0.00	0.06	0.05
Random Controller	0.09	0.36	0.45
±	0.03	0.06	0.05

Table 6.2 The number of times each agent managed to complete the level when paired with itself across the three action sets, (n=200)

Controller	Full	Relative	Simple
MCTS High	2	138	169
MCTS Medium	0	111	150
MCTS Low	0	74	119
GAController	17	78	86
Random Controller	5	49	53

difference between the relative and full action sets (t=-2.814, $p < 0.05$). There is clearly a significant difference between the full action set and the other action sets for this agent. This is also the case for the MCTS high agent, which is very close, but still below the significance threshold (t=-2.012, $p < 0.05$).

Across all agents, there is a clear decrease in agent performance when using the full action set. Performance across the board almost drops to zero.

Table 6.2 shows the win rates over all 200 mirror games played. From this data, the difference in the agent's performance in the full and relative action sets is clearly evident. By looking at the number of times the agents were able to solve the levels

('wins'), it's clear that the full action set is extremely detrimental to performance. Adding the communication actions brings the win rate for the High budget MCTS agent from 138 to 17. For the medium MCTS agent the same effect can be seen, where the number of wins drops from 111 to 0 and the GA agent, which drops from 78 to 17. Even random is effected, dropping from 49 wins to 5.

Breakdown by Maps

Table 6.3 Average scores of mirror games (agents paired with themselves) broken down by map, No Communication actions

Controller	Airlock	Butterfly	Pathfinding	Single-Door
MCTS High	0.38	1.00	1.00	1.00
±	0.10	0.00	0.00	0.00
MCTS Medium	0.23	0.93	1.00	1.00
±	0.11	0.04	0.00	0.00
MCTS Low	0.24	0.67	1.00	1.00
±	0.12	0.06	0.00	0.00
GAController	0.38	0.51	1.00	0.56
±	0.13	0.05	0.00	0.10
Random Controller	0.12	0.40	1.00	0.29
±	0.07	0.05	0.00	0.07

When the games are broken down by level and action set, table 6.3 shows that the agents are able to solve the Pathfinding level without issue. 'Airlock' on page 76 proves challenging for the agents, and Butterfly provides to be a good differentiator or the MCTS agents' performance. For the two action sets that feature commutation, table 6.5 proved to be the most challenging, with the agents not being able to reliably solve the Pathfinding level. No agent managed to obtain a score on the Airlock level and agents scored very poorly on the Butterfly level. The MCTS low agent did not manage to obtain a score across any level, and as a result obtained a score of zero.

Table 6.4 Average scores of mirror games (agents paired with themselves) broken down by map, Relative Communication actions

Controller	Airlock	Butterfly	Pathfinding	Single-Door
MCTS High	0.21	0.87	1.00	1.00
±	0.11	0.05	0.00	0.00
MCTS Medium	0.06	0.65	1.00	1.00
±	0.07	0.06	0.00	0.00
MCTS Low	0.00	0.46	1.00	0.70
±	0.00	0.06	0.00	0.08
GAController	0.24	0.40	1.00	0.57
±	0.12	0.05	0.00	0.10
Random Controller	0.06	0.22	0.99	0.19
±	0.05	0.04	0.02	0.07

The table 6.4 action set showed a decrease in performance, but the agents were still able to solve the Pathfinding level.

When these results are broken down by level, it can be seen that agents which used the full action set were unable to solve most of the levels. The airlock level became unsolvable within the time limit when using the full action set for all agents (table 6.5). The MCTS high and medium agents were able to solve the Single-door and Pathfinding levels reliably using the relative set (table 6.4) but performed worse on the harder Airlock and Butterfly levels.

Table 6.5 Average scores of mirror games (agents paired with themselves) broken down by map, Full Communication actions

Controller	Airlock	Butterfly	Pathfinding	Single-Door
MCTS High	0.00	0.03	0.41	0.00
±	0.00	0.02	0.07	0.00
MCTS Medium	0.00	0.00	0.01	0.00
±	0.00	0.00	0.02	0.00
MCTS Low	0.00	0.00	0.00	0.00
±	0.00	0.00	0.00	0.00
GAController	0.00	0.11	0.56	0.00
±	0.00	0.03	0.10	0.00
Random Controller	0.00	0.07	0.30	0.00
±	0.00	0.03	0.09	0.00

6.2.4 Discussion

The results show that for mirror games, the agents performed worse in the largest of the three action sets (full), table 6.5 and a smaller decrease in performance in the relative action set (table 6.4) when compared to the action set that featured no communication (table 6.3). On the full action set the low-budget MCTS was unable to obtain a score. Adding communication actions to the game increases the branching factor for each state in the tree, which means that the agent is not able to explore as much of the tree in the same number of iterations. As MCTS biases the search towards promising avenues and no path leads to a reward, the agent will essentially act randomly. As the action set is largely biased towards communication actions (which in this scenario are functionally equivalent to no-op actions) the agent is likely to stand still executing communication actions or randomly moving around the area. This affects both the MCTS variants and the GA as it makes the rewards more sparse, meaning it is harder to find moves which lead to rewards. As none of the agents make effective use of the communication actions, for this

experiment they can be thought of as equivalent to adding several copies of the no-op action to the list of possible actions at a given state.

Comparison of Action Sets

table 6.2 shows the agent performance across the three action sets, expressed as the number of times the pair solved the level. This shows a large drop in performance between the full and relative action set. The MCTS agent only being able to solve 2 levels out of the 200 played. This is in stark contrast to its performance in the relative and full action sets, where it 'won' 138 and 169 levels respectively. This trend is followed by all agents, with the GA controller being the least affected. This will be discussed in the next subsection when the levels are compared. Clearly, the full action set is unsuitable for use in the prediction experiment. The relative action set is still usable, as the agents still perform reasonably well within it (with a much shallower drop-off in performance).

Even the high-budget MCTS agent (the best performing agent so far) struggles to solve the pathfinding level in the full action set (resulting in an average score of 0.41 ± 0.07). The GA agent is able to cope with the larger action set better, resulting in the smallest decrease in performance (1.0 ± 0.00 to 0.56 ± 0.10). This is likely due to macro actions resulting in higher chance of it moving around the level (executing one movement action will result in it moving multiple spaces during the search, and so it is more likely to find the goal).

The levels where the agents were able to obtain a partial score still showed a significant decrease in performance against the two other communication modes. The Pathfinding level, which features no co-ordination requirements also showed a marked decrease in performance, with the low-budget MCTS agent having an average score of zero and the medium-budget MCTS agent having a score of 0.01.

The Butterfly level, which has the smallest dimensions, showed the least decrease in performance. This indicates that although the 'full' action set would allow for the creation of some interesting paired agent strategies, the levels become too difficult to solve and therefore would provide little experimental insight.

The reason for this decrease is that the agents have a much larger branching factor to deal with. The random controller shows a drop in performance as the agent will remain stationary when they are executing communication actions; when they remain stationary there is less chance they will collide with the goals and therefore the agents will achieve a lower score.

The 'relative' action set (where the agents could communicate a cardinal direction) shows a decrease in performance in comparison to the simple action set, however the agents were still able to obtain reasonable scores. All the non-random agents were able to obtain an average score of 1.0 on the Pathfinding level. The high- and medium-budget MCTS agents were able to retain their perfect scores on Single-door.

The simple action set (where the agents had no communication actions) shows comparable performance to the results presented in chapter 5. This implies that the two engine implementations are largely compatible. The 'relative' action set, while less useful for agents that can make effective use of communication, shows the least impact on the agent's performance. The 'full' action set has numerous communication actions in relation to movement actions, which results in the agents being unable to expand enough layers of the tree to reach the goals. As a result, the agents are behaving largely randomly.

Comparison of Maps

In the full action set, the agents are unable to solve any of the levels reliability. The Single-door and Airlock levels are completely unsolvable, with the agent unable to obtain a score at all. The agents were able to obtain a score on the Butterfly level than Single-door. This result is surprising when the complexity of the levels considered - Butterfly requires more interaction between the agents than Single-door and so therefore should be harder to solve.

This discrepancy may be explained by the relative size of the levels (Butterfly is 7×8, (56 communication actions), compared to Single-door at 9×9 (81 communication actions) and Airlock at 11×9, 99 communication actions). As the full action set creates one extra action per square on the level, a larger level has a larger branching factor, making rewards more sparse within the tree search (as these extra actions are functionally equivalent to the no-op action). This means that there will be fewer communication actions in the full set for Butterfly than Single-door. The agents were able to obtain a partial score on the Pathfinding level; this level is the same layout as Single-door but without the need for co-ordination between the agents, resulting in a shorter 'path' to the goal.

The simple and relative action sets show a similar pattern: the agents found Airlock the hardest level to solve, followed by Butterfly, these levels have the most co-ordination required between the agents. The relative action set had made the Single-door level harder for the low-budget MCTS agent to solve, but the medium- and high-budget MCTS variants were still able to obtain perfect scores. The Airlock level shows a similar pattern, becoming unsolvable for the low budget variant.

From the results, it shows that the four levels chosen represent a good distribution of the complexity of the created levels as they can differentiate between agent performance.

Comparison of Agents

A large branching factor is known to cause issues for MCTS, and as the versions used in these experiments have comparatively few iterations in comparison to the size of the action space its likely the agents encounter very few non-communication actions during the search. Another issue caused by the large branching factor is that the tree created by MCTS will not be very deep (as the agents are iteration-limited). There are approaches which help to deal with large branching factors, such as pruning the tree (Schadd *et al.*, 2012). There has also been research into developing MCTS hybrids for games with large branching factors such as RTS games (Baier and Cowling, 2018).

In a level like airlock, which has 99 communication actions, and 5 non-communication actions, the 75 iteration cap of MCTS low will result in it being unable to fully explore a single node within the tree. Even the high-budget MCTS agent (with an iteration count of 500) will be unlikely to see beyond a few nodes in the tree. The GA-based agent will perform many deep searches (as there is no tree to generate) and will be able to explore more of the map, giving it a greater chance of finding a reward.

6.2.5 Conclusions

The results of this experiment show that the full communication action set is not suitable for use as it makes the levels unsolvable. This is due to the increased branching factor. The results also show that the relative action set is usable, but the

low-budget MCTS variant may struggle to solve the levels. As both the high-and medium-budget variants don't show a large difference in performance, this should prove suitable for the experiments. Therefore for the communication experiment, the *relative* action set should be used.

This experiment indicates that unless measures are taken to combat the increase in the branching factor caused by communication actions, large amounts of action sets are likely to make level unplayable for the agents. As a result, it makes sense to constrain the action set to the *relative* action set (4 directional communication actions, 4 movement actions and the no-op action) as a good balance between these. A possible avenue for future work is to explore the trade-off between the expressiveness of the communication actions and the effect on the branching factor of the game.

6.3 Agent Prediction

The results presented in section 6.2 show that both of the two communication systems had a negative impact on performance, but the relative action set showed the lesser impact. As a result, this is the action set that was chosen for the agent prediction experiment. The experiment described in this section determines how much of an effect prediction has on MCTS performance in the *Tiny Co-Op* environment.

There are two different scenarios which were considered for this experiment. Firstly, the effect of prediction on a goal-directed agent that exhibits intelligent behaviour: prediction should result in more directed searches for this agent, as the search will be less noisy. The second scenario that will be considered that of an agent that is dependent on explicit communication. For this agent, assuming perfect play will not be as effective, and random play is not a good model of the agent's behaviour

either (as the agent waits on the player to make choices). By evaluating the effects of prediction with such a agent, it is possible to identify the effectiveness of using an accurate model for a non-optimal player.

As none of the agents make effective use of the communication actions in the game (the existing agents treat the action as a no-op), a new agent that takes communication into account when deciding its move is needed. The simplest agent which responds to communication actions is a reactive agent that treats the flares as suggestions for the direction of movement from the paired agent and blindly executes them. This behaviour will not be reflected in the state itself and therefore an agent which is not reasoning about the other agent's behaviour will not be aware of the communication actions' effects.

This represents a possible case when some forms of modelling, such as using an evaluation function based on the players own decision making process may not be sufficient. This idea is not only important in the abstract, there are games which feature co-ordination over a set of equally valid strategies, each of which is optimal (Ashlock and Greenwood, 2016; Bard *et al.*, 2019).

6.3.1 Agent Descriptions

To facilitate this experiment, three new agents were created: an MCTS variant which makes use of a very simplistic modelling technique; an agent which implements a basic strategy dependent on communication; and a random agent that is biased towards movement actions. In addition, an MCTS variant specialised for simultaneous move games is also compared against.

Predictor MCTS

The *Predictor MCTS* variant is the same as that described in section 5.2.1, however rather than modelling the paired agent as making random moves within the tree and roll-out phases, the agent instead uses a copy of the actual policy of the paired agent. This allows it to 'model' accurately the result of the other agent's actions from any given state. Such a modelling strategy has been used before for adversarial games (Goodman and Lucas, 2020).

This approach makes the assumption that a perfect model of the paired agent's behaviour is available. This is the 'best case' scenario for the agent, but for this experiment it helps establish a baseline on how effective such modelling is within the game. A model which is inaccurate (for example, a learnt model based on previous behaviour) would result in the agent making incorrect assumptions in its decision-making. Therefore, a 'perfect' model should provide the best possible performance. Once the effectiveness of modelling has been established, the model could also be substituted with a learnt model in the future, either trained off-line from game trace information (as will be presented for *Hanabi* in chapter 8) or on-line based on the agent's previous behaviour.

Decoupled UCT

For this experiment, an agent using decoupled Upper Confidence Bound applied to trees (UCT) (duct) is introduced. This approach has shown promise for use with simultaneous move games in existing literature, and has been discussed in section 4.2. This agent represents an alternative to assuming a random model for the paired agent, instead, the agent will use an $n \times m$ matrix representing the moves made by each player. This will effectively result in the agent choosing to model the paired agent as if it was following the same strategy as MCTS.

Table 6.6 A tabular listing of the flare-following agent's policy

Paired agent's last move	**My Move**
FLARE_LEFT	MOVE_LEFT
FLARE_RIGHT	MOVE_RIGHT
FLARE_SOUTH	MOVE_SOUTH
FLARE_NORTH	MOVE_NORTH
No-Op	No-Op
LEFT	No-Op
RIGHT	No-Op
SOUTH	No-Op
NORTH	No-Op

One possible downside to this approach is that as a fixed iteration budget is being used, the agent will explore less of the search space than the agent assuming a random model (as the node will also contain opponent moves).The implementation used is adapted from the Ludii framework (Piette *et al.*, 2019).

Follow the Flare

This agent is a simple reactive agent which uses the communication action generated by the paired agent in the previous time-step and performs the associated move (see table 6.6). If the paired agent did not execute a communication action in the last time-step the agent issues a No-Op move.

Bias Random

This agent divides the legal moves provided into two sets of actions using filters provided by the framework: communication actions and non-communication actions. It then tosses a biased coin (75% towards movement, 25% towards communication) and then randomly selects an action from the corresponding action list. As the agent can only obtain scores by visiting goals, the bias encourages the random agent to move rather than to execute communication actions.

6.3.2 Method

This experiment used the same four maps used in the preliminary experiment. It is not possible to model a predictor agent accurately, because doing so is endlessly recursive. The predictor agent itself requires a model, if that model were to be accurate, it would also need to be a predictor, which in turn would require a model, and so on. It is possible to limit this recursion by imposing a limit on the depth of nesting of the predictors, but given the desire to see if an accurate model is required for co-operative planning, this could distort the findings. Instead, the agents were split into two groups, the 'test' and 'paired' sets. The 'test' set contains the agents that are being evaluated. This set consists of the predictor and MCTS variants for all three budgets. It also includes Random for comparison purposes. The 'paired' set contains the Follow the Flare agent, both random agents, the GA-based agent and the high-budget MCTS agent.

For each map being evaluated, each agent in the 'test' set was evaluated with each agent in the 'paired' set. This was done over the two action sets (relative and simple). Each agent played 50 repeats per level. This was possible as the full communication set is not used, and the time previously used for these could be allocated to running more repeats. The agents were not time-limited as they were limited based on iteration count. As the paired set contains an MCTS variant, some games took a very long time to execute.

The same turn limits and information mentioned in section 6.2 were used. The same parameters were used for the predictor, duct and standard MCTS agents.

Test Agents

- MCTS (High, Medium and Low variants)
- Predictor MCTS (High, Medium and Low variants)
- Duct (High, Medium, Low variants)
- Random Controller

Paired Agents

- Follow the Flare
- Random Controller
- Bias Random
- GAController
- High MCTS

6.3.3 Results

This section outlines the results of the experiment. The results presented are once again showing the average across the games played. With a score of one indicating perfect play, and a score of 0 indicating that no agent visited any goal. In tables that contain averages, $\pm$ indicates the 95% confidence interval.

The results presented below show the relative performance of the MCTS, duct and Predictor MCTS agents paired with both the high-budget MCTS and the Follow the Flare agents. The analysis of the results is presented after all results have shown.

Games with High Budget MCTS

This sub-section breaks the results down in the case when the agents are paired with a copy of the high-budget MCTS agent. This demonstrates the difference between assuming a random model and a 'perfect' model of an intelligent player.

Table 6.7 Comparison of the scores obtained by Predictor and Standard MCTS when paired with the standard high-budget MCTS

	Relative			**Simple**		
	High	**Medium**	**Low**	**High**	**Medium**	**Low**
MCTS	0.75	0.73	0.71	0.86	0.84	0.82
$\pm$	0.04	0.04	0.04	0.03	0.04	0.04
Predictor	1.00	1.00	0.73	1.00	0.98	0.86
$\pm$	0.00	0.00	0.04	0.00	0.01	0.03
Duct	0.82	0.76	0.75	0.92	0.89	0.87
$\pm$	0.04	0.04	0.04	0.03	0.03	0.03
Random		0.71			0.86	
$\pm$		0.04			0.03	

The results presented in table 6.7 show a difference in performance with the MCTS agent and its predictor variant. When evaluated across both action sets, there is a significant difference between the high-budget predictor and the high-budget random model ($t=14.17$, $p < 0.05$). This is also the case for the medium version ($t=13.964$, $p < 0.05$), but it is not the case for the low budget agents. The duct agent performed better than assuming a random model. This is to be expected as it will be able to exploit the goal-directed nature of the paired agent. The lower scores compared to the predictor agent is probably a result of the smaller number of games played by the predictor agent (as discussed below). As the Duct agent does not need to fully evaluate the MCTS agent for each move, it is significantly faster to run than Predictor, and so was able to complete the games in a suitable amount of time.

The results also show that the medium and high budget predictors where also much faster to solve the levels than their random counterparts, as seen in table 6.8.

Table 6.8 Comparison of the moves taken of Predictor and Standard MCTS when paired with the standard high-budget MCTS

	Relative			**Simple**		
	High	**Medium**	**Low**	**High**	**Medium**	**Low**
MCTS	902.24	991.79	1139.84	511.89	629.34	75.28
±	82.65	82.74	80.38	68.75	73.19	77.34
Predictor	90.23	175.93	1013.13	59.43	139.06	574.46
±	14.01	31.71	83.32	6.72	30.60	72.55
Duct	804.51	924.74	1005.24	361.15	470.15	608.56
±	78.32	83.78	83.22	54.78	64.57	70.92
Random		1304.49			72.12	
±		898.81			68.35	

Table 6.9 Perfect Score vs Completed for games paired with standard high-budget MCTS, for each agent the top number indicates how many games were solved perfectly, the bottom number indicates the number of games played

	Relative			**Simple**		
	High	**Medium**	**Low**	**High**	**Medium**	**Low**
MCTS	277	257	243	345	335	318
completed	400	400	400	400	400	400
Predictor	198	231	251	321	358	335
completed	198	231	397	321	365	400
Duct	312	270	259	368	353	337
completed	400	400	400	400	400	400
Random		251			327	
completed		400			400	

Although these results are very promising, there is one limitation which must be stated. When using MCTS in this way, a full MCTS iteration must be conducted to compute the move for the paired player whenever a move is made (*i.e.* the state is forwarded). This requires a considerable amount of computation. Some games scheduled for the cluster were therefore killed before returning results as

they exceeded their compute time. The number of games played with a perfect score and the total number of games played are listed in table 6.9. The 'completed' row shows the number of games that are in the set, with the rest being killed by the cluster before execution could complete.

These results are incomplete however. Due to the immense amount of time required to compute a nested search, some games did not complete as a result of being forcibly killed by the compute cluster. The total number of games completed is shown in table 6.9, where a maximum of 400 is possible (50 repeats of 4 levels, both as first and second player). This is discussed further in section 6.3.4.

Follow the Flare

These are the results when paired with Follow the Flare. As this agent only works correctly with communications, only results for the relative action set are displayed. There are no missing games for this subset.

Table 6.10 Comparison of the average scores obtained of random and accurate models when paired with follow the flare

	Scores			**Moves**		
	High	**Medium**	**Low**	**High**	**Medium**	**Low**
MCTS	0.69	0.59	0.43	1296.04	1483.45	1562.61
±	0.04	0.04	0.04	72.22	69.65	72.01
Predictor	0.75	0.70	0.52	881.78	1148.09	1456.53
±	0.04	0.04	0.04	81.34	82.32	81.40
Duct	0.76	0.65	0.52	1291.86	1475.86	1608.42
±	0.03	0.04	0.04	70.16	69.40	65.60
Random		0.37			1654.12	
±		0.04			61.35	

The results present in table 6.10 show the average score obtained across all maps when using the random and the accurate predictor. All games in this set completed.

For these games, only the relative action set is considered (as FollowTheFlare requires these to function). For this dataset, all three agent combinations show significance - high ($t=2.02$, $p < 0.05$), medium ($t=3.93$, $p < 0.05$) and low ($t=3.10$, $p < 0.05$). The table also shows the average moves required to solve the levels.

Table 6.11 Comparison of the scores of random and accurate models when paired with Follow the Flare, split by the game level being played

	MCTS			**PredictorMCTS**			**Random**
	High	**Medium**	**Low**	**High**	**Medium**	**Low**	
Pathfinding	1.00	1.00	1.00	1.00	1.00	1.00	1.00
±	0.00	0.01	0.00	0.00	0.00	0.00	0.01
Single-door	0.94	0.67	0.28	1.00	0.99	0.54	0.18
±	0.04	0.07	0.07	0.00	0.02	0.07	0.05
Airlock	0.15	0.12	0.03	0.10	0.13	0.05	0.06
±	0.07	0.06	0.03	0.06	0.06	0.04	0.03
Butterfly	0.70	0.56	0.43	0.91	0.69	0.51	0.27
±	0.05	0.04	0.03	0.03	0.04	0.04	0.03

	Duct		
	High	**Medium**	**Low**
Pathfinding	1.00	1.00	1.00
±	0.00	0.01	0.01
Single-door	0.91	0.74	0.44
±	0.04	0.05	0.06
Airlock	0.40	0.26	0.17
±	0.09	0.08	0.06
Butterfly	0.76	0.61	0.47
±	0.04	0.04	0.03

Table 6.12 shows the win rates (levels with a perfect score). From this, it is clear that both agents were able to solve path-finding with follow the flare. This level requires no explicit co-operation between the agents, but does require the agent to tell the other were to go. Predictor managed to 'win' more games when on single-door. The medium MCTS agent managed to win exactly half the games on that level. Duct

Table 6.12 Number of perfect score games, when paired with Follow the Flare (n=100)

	MCTS			PredictorMCTS			Random
	High	Medium	Low	High	Medium	Low	
Pathfinding	100	99.0	100	100	100	100	99
Single-door	88	50	14	100	99	30	1
Airlock	14	10	1	10	12	4	0
Butterfly	26	10	0	70	22	7	0

	Duct		
	High	Medium	Low
Pathfinding	99	99	98
Single-door	81	48	15
Airlock	32	19	6
Butterfly	25	10	0

achieved similar completion rates to MCTS with a random model. With a notable exception of airlock, where it performed much better than both of the other agents for the high-budget variant.

Table 6.10 shows that overall, the scores are quite close when averaged over all 4 maps. However, Predictor has better scores and fewer moves required than assuming random movement. Duct showed better performance than the random opponent model, but still took longer than the accurate model to solve the levels. There is no significant difference in scores between Duct and Predictor. When split by level, the scores (table 6.11) for the predictor variants are higher on most levels. Airlock posed a problem for the agents, with no agent managing to obtain high scores.

6.3.4 Analysis

The tables 6.7 and 6.10 show that prediction has a significant impact on performance when paired with both an intelligent agent and an agent which relies on commu-

nication. When using a predictive model of the paired agent there is a trade-off between time spent evaluating the model and the performance benefit it provides (Iida *et al.*, 1994). Although this implementation is iteration-based and so wall-time is not a factor the prediction, the agents still need to be able to execute quickly. In the case where MCTS took too long, the computer cluster it was running on intervened and killed the jobs. For this to be practical for real world applications clearly a fast model is needed. Duct, which uses UCT as its model showed no such performance problem. However, when it was paired with the flare following agent, took as long as the agent that assumed the random model to achieve the same scores.

It is possible to cap the evaluations so the model executes faster, but doing so will result in a less capable model of the paired agent. Throughout the *Tiny Co-Op* work, the iteration counts have had a large impact on performance. By reducing the iteration count the agent will not be an accurate prediction of the paired agent's behaviour.

When paired with the random agent, there is very little difference between the Predictor's scores and MCTS. This is expected because the two agents are essentially equivalent. The Bias Random agent does not show much difference between the two agents either. The performance difference between the two high-budget versions of the agent when using the relative action set is not significant ($t = 1.74$, $p > 0.05$). As Follow the Flare will always perform a no-op when not provided with a communication action on the previous turn, table 6.10 does not show the 'simple' action set.

Using the Predictor with two of the paired agents caused problems: the high-budget MCTS and the GA agent proved to be too computationally expensive to execute as a model. The details of which games were terminated for the MCTS agent can be

found in table 6.9. For the GA agent, only the high-budget predictor was affected, playing 219 of the expected 400 games for the relative action set and 295 of 400 games for the simple action set (medium and low budget predictors played all games with the GA agent).

Performance when Paired with High Budget MCTS

When the agents are paired with high-budget MCTS, the scores indicate a significant improvement in performance when assuming the other player is the MCTS agent, as opposed to playing randomly. All games played by the high-and medium-budget agents (table 6.7). There is not much variation in performance across budgets for the MCTS agent assuming random play. This indicates that the score is largely dictated by the paired agent. The agents show a decrease in performance when including communication actions; this is to be expected as the paired MCTS agent does not make use of them, which results in the agent wasting iterations exploring actions which have no effect on the game.

Table 6.9 indicates one major drawback with using a nested MCTS policy – doing so creates an extreme increase in the time required for a game as every move considered requires a full search by the nested policy to return the paired agent's move. This is clearly not feasible for a real-time game, however, the results show that when the search did complete in time, the algorithm works well. A possible solution for this limitation is addressed in chapter 8.

The moves required to complete the levels are significantly lower than when assuming random play. This is also the case if only perfect scoring games are considered (387.32 moves on average for MCTS, vs 98.95 for the predictor) for the highest budget versions using the relative action set.

Performance when Paired with Follow the Flare

When paired with Follow the Flare, the scores between the types of MCTS agent are much closer (table 6.10). This appears to be largely due to the poor performance of the agents in the Airlock level (table 6.11). As the agent is not only responsible for moving itself, but also for directing the actions of the paired agent, the required number of actions is larger. As a result the agent must be able to see further to receive rewards. This could result in the Predictor agent being starved of rewards during the search, and largely acting randomly. By assuming random behaviour, MCTS may have an advantage in this scenario, as it is more likely to 'see' rewards during the search.

In the other, easier, levels the required number of moves is likely to be larger than Random, but still within the planning horizon of the Predictor agent. In these cases, prediction appears to be beneficial. The high-budget Predictor was able to obtain a high score on both the Single-door and Butterfly levels, with the medium-budget agent also obtaining a higher average score than the standard MCTS counterpart (0.99 ± 0.02 vs 0.67 ± 0.7) on the Single-door level. These are also reflected in the number of games that achieved a perfect score (table 6.12). This experiment shows that performance with follow the flare is improved by prediction.

6.4 Conclusions

From these experiments, it is clear that using an accurate model is better than assuming the paired agent is making random moves. This is true when paired with both intelligent agents (such as the high-budget MCTS) and an agent that is reliant on communication actions (Follow the Flare). These results show that prediction in *Tiny Co-Op* seems to have a beneficial effect on agent performance. However,

due to the limited number of strategies available, it is difficult to draw any strong conclusions about the overall benefit of prediction.

The experiments in this chapter also looked at two different communication systems. Although the richer of the two allowed for a wider set of possible agents, the branching factor meant that it was not feasible to use it in experiments. The more limited, 'relative' action set does not degrade agent performance as much as the full action set. When paired with MCTS, the predictor did not suffer as strong of a performance problem when using the relative action set. This implies that the benefit gained from having a strong model (and therefore a less noisy scoring function) helps to counteract the increase in branching factor. The model cannot itself be a predictor, as doing so would require a endlessly recursive evaluation. It may be possible to factor in a second-order theory of mind agent (a model that assumes the other agent is performing modelling), however doing so is likely to be computationally expensive and is outside the scope of this thesis.

Another technique for simultaneous moves is to store a matrix in each tree node providing $m \times n$ action space (see Lanctot *et al.*, 2013). A possible future experiment could be to combine a predictive model with this approach and evaluate the effect. The ability of the model to prune the actions and thus reduce the combinatorial explosion associated with such as search could provide beneficial.

Using a model in this way creates problems if the agent being used as the model is computationally expensive. This would not be suitable for real-time applications. Access to such a model might also not be available (for example, when playing with humans). The technique is viable if the model exists and can be executed quickly. For chapter 7 this assumption will be made; in chapter 8 a way of computing a

fast-to-execute model of a computationally-intensive agent is presented to overcome this limitation.

As the Follow the Flare agent is purely reactive and is not capable of acting intelligently independently, the experiments cannot be used to conclude that an agent that is both intelligent and responds to communication will work with the predictor. This is one of the objectives for the work presented in chapter 7.

As the levels are puzzles, there are only a limited number of strategies that can be employed. In addition, the game as presented is only capable of supporting two-player games. To test if prediction works with a wider array of strategies would require extensive modifications to the game; in addition there are no third party strategies available. To overcome these limitations, a new game is needed – one that already has existing strategies and has a communication system built into it that does not introduce large branching factors. The game chosen for this purpose is *Hanabi*, a co-operative card game with multiple strategies and a well-defined communication mechanism that makes it well suited for evaluating the effects of prediction.

Chapter 7 presents a variant of the Predictor that is capable of working within *Hanabi*. This is quite a different domain from *Tiny Co-Op*, as the game is turn-based, partially observable and does not support simultaneous moves. This further allows the testing of the robustness of the effects of prediction.

Chapter 7

Effects of Prediction in *Hanabi*

The results of chapter 6 show that agent capable of prediction showed better results than the non-predicting variant. However, the domain used is designed for two-player games. In addition, the domain has no third-party strategies and as the domain is largely puzzle-based there is limited variety in possible solutions.

The card game *Hanabi* has a much larger set of possible agents and human players often reason about different strategies for playing the game. In addition, there has been some formal research in the domain (see section 2.8), providing a source of strategies from third parties. This chapter presents experiments using a variant of the prediction strategy presented in chapter 6, applied to *Hanabi*.

As *Hanabi* features 3-,4-and-5-player variants, games with more than two players can be evaluated. In many games, it is desirable to be able to interact with more than one other player. Many co-operative games require the ability to interact either with a team of players. In games like *The resistance* play only works with a minimum of 5 people, and many games such as *pandemic* require a team of players working

together. This is also important when considering larger-scale environments such as controlling large numbers of characters in games.

This chapter is based on the work presented in Walton-Rivers *et al.*, 2017.

7.1 Introduction

The results presented in chapter 6 showed that in *Tiny Co-Op* agents were able to work better when they were predicting the behaviour of the paired agent. The puzzle domains and agents presented in *Tiny Co-Op* were quite limited and the agent being evaluated did not show any intelligent behaviour beyond what the agent was being directed to do.

The range of possible puzzles present in *Tiny Co-Op* is quite restricted, In addition there are no independently-produced strategies for playing the game. Note intelligent strategies that were available were costly to execute and so was unsuitable for examining the effects of prediction.

Tiny Co-Op prediction evaluations are also costly to execute, as the only competent agents we have for the domain are based on Monte Carlo Tree Search (MCTS) and Genetic Algorithms (GAs), and so running a predictor game takes a significant period of time. In order to test how effective prediction can be, fast-executing agents are required which can be queried quickly.

Hanabi offers good test-bed for such research as it possesses a number of useful properties. Firstly, the game has a communication mechanism built into its mechanics. This is useful as it means that any agent that is able to play the game will be able to communicate with other agents. Secondly, the game rules disallow communication outside of the structured communication mechanism. This means that agents don't

need to be able to have any external functionality such as having to parse natural language chat messages in order to collaborate with other agents. Thirdly, the game is fully co-operative with a well-defined shared objective. This means that the agents will not need to worry about conflicting or unknown goals, such as in games with defectors like *The Resistance* and *Dead of Winter*.

7.2 Motivation

The game has a number of different strategies that human players employ. These can often be expressed as lists of rules. This property gives a large amount of variety in how people choose to play the game and any underlying messages that are being communicated outside of the game's core tell rules.

7.2.1 Social rules

Players also often create conventions, such as, "Discard the left most card" or "Discard the card which you've had the longest and not been told about". These are not enshrined in the game rules but are developed by players as they play the game (or over multiple games). Agents that are not capable of picking up on these social rules should not perform as well as agents which can. This makes it a perfect test-bed for socially-capable agents.

For example, if a player points out a card it is more likely than not that the card is playable. This is due to the fact that it costs a resource to tell a card, and discarding a card pulls another card from the deck but also gives one information token. The act of telling about a single card is therefore often not useful for discarding but is used for human players to indicate that a card should be played.

In order to perform well, the agent will need to understand these social rules, as well as the game rules. These social rules are encoded as part of the players' strategies.

7.3 Method

To test whether introducing these social rules to the game-playing agents made them better players, variant of MCTS called Infomation Set Monte Carlo Tree Search (IS-MCTS) was created, designed for partially-observable environments and modified to include the prediction ideas from the work presented in chapter 5. This experiment compares this agent against rule-based agents and the unmodified IS-MCTS agent.

Each agent that is being evaluated is placed in a game which consists of a set of other, 'paired' agents all using the same pre-determined strategy as each other. The paired agents used are described in section 7.4. These represent a range of abilities from random play to a strong rule-based agent.

To avoid problems arising from agent orderings, each game was played twice, with the agent being evaluated being placed in different positions for each game. The agents were evaluated over 200 randomly-selected deck orderings and their scores, total moves made, information remaining at the end of the game, and total remaining lives at the end of the game were recorded.

Hanabi allows for games of between two and five players. For completeness, the agents play all of these different player counts for each deck ordering and paired agent. This results in each agent playing 12,800 games (200 orderings x 2 repeats x 7 paired agents x 4 player counts).

Agents are given a one-second time budget per move. If they violate this the agent will be disqualified. The agent will also be disqualified if they make an illegal move.

Neither of these should occur during ordinary execution as the agents are written not to violate these constraints.

The games were played on the Intelligent Games & Games Intelligence (IGGI) cluster. During execution, the game engine performed a 'warm up' game to ensure that the Java Virtual Machine (JVM) does not interfere in the agent timings by attempting to just-in-time compile the agents. Each 'task' (evaluated agent, paired agent and deck ordering) was executed in a separate process; this was so the tasks can be run in parallel.

For all games, the full game history (deck ordering, moves made by each agent and debug information obtained from the agents) were recorded in a separate file. From this information the full game state can be reconstructed if needed. This can aid in analysis of agent behaviour and is used in chapter 8.

7.4 Agents

7.4.1 Rule-Based Agents

These are rule based agents that are implemented using the technique described in section 4.1. Based on existing research, human play and strategy discussions, a number of different rules are prevalent. These are often expressed in terms of production rules by human players: "If the next player has a playable card then tell them about the card". These rules are relatively easy to convert into production rules which can be ordered and combined to make game-playing agents.

As well as rules adapted from observed play, there are rules that emulate the strategies discussed in existing literature. The full descriptions of all implemented rules and how they behave can be found in Appendix A.

Internal

This agent is an implementation of the agent of the same name created by Osawa. It features information about its own hand but does not remember information about what other players have been told.

- Play Safe Card
- Osawa Discard
- Tell Playable Card
- Tell Randomly
- Discard Randomly

The OsawaDiscard rule implements the discard rule described in (Osawa, 2015), namely, discard cards that are no longer required. This can occur either because the card is a duplicate of one that has already been played or because it is no longer possible to play the card (for example, a red 4 if both red 3s have been discarded). This rule describes a relatively intelligent discard strategy and so the rule is used for most of the rule-based agents.

Outer

This agent is an implementation of the agent of the same name created by Osawa. The implementation of this agent is largely the same as Internal, but the tell rules are replaced with more intelligent rules. It features information about what the other agents have been told already, to avoid repeating information they are already aware of.

- Play Safe Card

- Osawa Discard
- Tell Playable Card Outer
- Tell Unknown Cards
- Discard Randomly

Cautious

This agent is derived from novice human-play. The agent will play cautiously and will avoid losing lives. It shares the same discard and play rules as Internal and Outer. It will only ever identify cards which are required, and will only discard cards if there is nothing to play or tell.

- Play If Certain
- Play Safe Card
- Tell Anyone About Useful Card
- Osawa Discard
- Discard Randomly

IGGI

This agent is a modification of the cautious agent using deterministic discard rules. This makes the agent easier for use with prediction as the card which will be thrown away is known. It allows a playable card which is about to be discarded to be identified by the predictor to ensure the card is played rather than discarded.

- Play If Certain

- Play Safe Card
- Tell Anyone About Useful Card
- Osawa Discard
- Discard Oldest First

Piers

Piers is an agent which makes use of some richer features of the agent framework, including *if* rules. The agent expands on IGGI. If the game is nearing its end, the agent will play the card most likely to be valid. During the game, the agent will take a more risky approach than IGGI and will risk playing a card even if it is more certain than not that the card is safe to play. It will only perform this behaviour if there is at least one life remaining.

- if lives > 1 & !cardsLeftInDeck then Play Safe Card (> 0)
- Play safe card
- If lives > 1 then play save card (> 0.6)
- Tell Anyone about useful card
- If information < 4 then tell disposable
- Osawa Discard
- Discard Oldest First
- Tell Randomly
- Discard Randomly

The value passed to the PlayProbablySafeCard rule determines what the rule considers an acceptable risk. The risk is calculated by taking all possible cards that could occupy a given position and calculating the subset of these which are playable. This is then represented as a value between 0 and 1 (playable cards divided by possible cards). If this value is greater than or equal to the threshold, the card is played.

Flawed

This agent is designed to be intelligent, but has some deliberate flaws added to its strategy. This makes it a weaker player than the others. The agent will only tell randomly and so will not intentionally tell players about cards which could be useful to them. It also has a very risky version of the play rule, which will play cards if there is at least a 25% chance that the card is playable.

An intelligent agent paired with this agent should be able to improve its score by pointing out cards which are useful to play. If it does not do so the agent will be reliant on being randomly told information.

- Play Safe Card
- Play Probably Safe Card (0.25)
- Tell Randomly
- Osawa Discard
- Discard Oldest First
- Discard Randomly

Van Den Bergh Rule

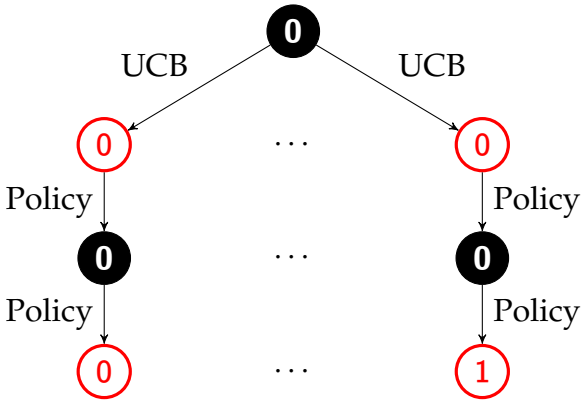
This is the best rule-based agent proposed by van den Bergh *et al.* in (van den Bergh *et al.*, 2016). From observing human play there are four main tasks:

- If I'm certain enough a card is playable, play it
- If I'm certain enough a card is useless, discard it
- Give a hint if possible
- Discard a card

van den Bergh *et al.* used a GA to evolve the best options for each of these four tasks, resulting in the following agent:

- If lives > 1 then Play Probably safe card 0.6 else play safe card
- Discard Probably Useless card (1.0)
- Tell anyone about useful card
- Tell anyone about useless card
- Tell most information
- Discard probably useless card 0.0

The DiscardProbablyUselessCard rule works under the same principle as PlayProbablySafeCard, but rather than calculating the playable cards, instead it calculates the cards that can be safely discarded.



7.4.2 Other Agents

Legal Random

This agent makes a random move selected from the set of legal moves available to it at this game state.

Monte Carlo Search

This agent is a simple Monte Carlo Search (MCS) agent that uses a single Upper Confidence Bound (UCB) layer followed by a roll-out phase using a policy. This helps the agent pick a suitable move by allowing UCB to guide the move selection, followed by a state evaluation making use of domain knowledge.

IS-MCTS

This agent is an implementation of the IS-MCTS agent presented by (Cowling *et al.*, 2012a). This agent uses the standard MCTS approach but determinsises the state before each iteration.

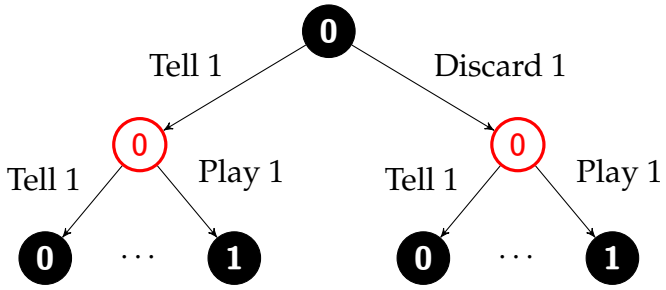


Fig. 7.1 An expansion of IS-MCTS's tree

Determinsation Strategy The agent may attempt to use the information about the cards in their hand to find the likelihood that a given card is present in their hand. The process is implemented as follows:

1. Order the slots in our hand from most to least restrictive
2. Filter cards contained in the deck to match these restrictions
3. Bind one card from the list of legal cards to a given position in our hand and remove it from the deck
4. continue until we run out of slots in our hand

As there is always at least one correct combination – the real hand – this should always result in a valid determinisation of the deck.

The remaining cards in the deck are simply shuffled in order to randomise the order.

Predictor IS-MCTS

This is an IS-MCTS variant which rather than using tree nodes for other player's moves, it instead asks a model for them. The model then returns a possible move that the predicted agent could make in that state. This modelling strategy allows the agent to explore nodes which the paired agent would actually consider. Allowing possible but unlikely branches to remain unexplored. This is designed to help the

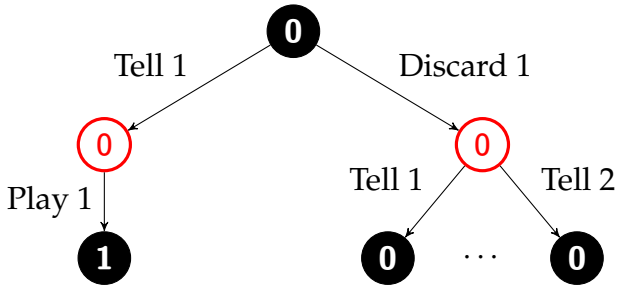


Fig. 7.2 An expansion of Predictor IS-MCTS's tree

.

agent avoid moves which look promising within the search but ultimately would never occur within the real game.

There are two main ways in which this can affect play. First, the agent can speculate about how the model will react to a given change in the world state. This allows the agent to consider not only the effects of its actions on the world, but also how those actions will affect the other agents in the environment. This is a key skill to human *hanabi* players, who develop complex strategies. This can be observed in the literature on *Hanabi* playing agents presented in section 2.8.

During the roll-outs, the model is used for decisions made by the paired agents but random moves will be used for moves made by this agent. A full description of the technique is available in section 4.3.

7.5 Validation

As the engine contained re-implementations of agents presented in other research, it is necessary to validate that the re-implementations matched what the original papers suggested they were capable of. To verify that re-implementations match the algorithms and results described in their papers, the experiments described in the original papers were recreated using the re-implementations.

7.5.1 Results

Table 7.1 Comparison of our implementations vs the reported scores from the original authors

Agent	Re-implementation	Original	N Games	N Players
Internal	10.12 (1.98)	10.97 (1.94)	10^2	2
Outer	10.12 (1.98)	10.97 (1.94)	10^2	2
Van den Bergh Rule	16.95	15.4	10^4	3

Table 7.1 shows that the two agents based on the work of Osawa have similar scores to those reported by the original paper. The re-implementation of van den Bergh *et al.*, 2016 shows returns higher results than those reported in the paper. The implementation of this agent has been validated against the information provided in their paper and has found to match the rules set out in the paper. The game is stochastic and so this could be an error relating to game itself. The paper lacks sufficient statistical information to check if the agents vary significantly.

7.6 Experiment

In order to evaluate how much of an effect player-modelling had on game-play, the results of IS-MCTS and Predictor IS-MCTS were compared. The agents played a set of different deck orderings (determined by a seed value), a range of different opponents and different positions (to prevent first-player advantage).

For each agent being evaluated, the agent is paired with each of a battery of 7 agents in turn, representing a spread of player abilities. The agent plays each deck ordering twice, with the location of the evaluated agent differing between each repeat. The agent plays the 2, 3, 4 and 5 player variants of each deck ordering and saves the final state (score, lives, moves made, information tokens) achieved at the end of the

game. The agent also logs all moves made to a separate file to allow the state to be reconstructed afterwards if needed (this functionality will be used for chapter 8).

The battery of agents used for the experiment were:

- IGGI
- Internal
- Outer
- Legal Random
- Flawed
- Piers
- Van Den Bergh Rule

7.6.1 Results

The results are presented, first averaged over all paired agents and number of players in the game (table 7.2). This shows good performance overall for the predictor agent, but the strong rule-based agents appear to beat it. When the results are broken down by the number of players in the game (table 7.3) they show that for all game sizes other than two players, the predictor agent is in fact the best-scoring agent for the 3,-4-and-5 player variants of the game.

Both overall, and when broken down by player count, the Predictor agent outperforms the standard variant which assumes random play on the part of the paired agents.

Table 7.2 Results with score, standard error of the mean for each agent, sorted by score (N=11200)

Agent	Score	SEM
Piers	**11.18**	0.06
MCS-IGGI	10.97	0.06
IGGI	10.96	0.06
Van Den Bergh Rule	10.88	0.06
Predictor IS-MCTS	10.74	0.06
Outer	10.20	0.05
IS-MCTS	5.90	0.04
MCS-Legal Random	5.45	0.04
MCS-Flawed	5.06	0.04
Flawed	5.02	0.04
Legal Random	4.59	0.04

The difference between MCTS and Predictor IS-MCTS is significant when the scores across all game sizes is compared ($t = -68.4591793141112$, $p < 0.05$). This indicates that there is a significant difference in ability between the two agents. This shows that prediction has a beneficial impact in *Hanabi*.

Table 7.3 Results of the Predictor agent paired with other agents

Agent	2	3	4	5
Flawed	3.52	4.69	5.43	6.45
IGGI	11.76	11.29	10.71	10.09
IS-MCTS	4.80	5.44	6.24	7.14
Legal Random	1.68	4.30	5.83	6.53
MCS-Flawed	3.61	4.72	5.43	6.48
MCS-IGGI	11.79	11.34	10.68	10.09
MCS-Legal Random	3.84	5.14	5.87	6.95
Outer	10.55	10.64	9.99	9.62
Piers	**11.91**	11.67	10.89	10.26
Predictor IS-MCTS	8.36	**12.14**	**11.43**	**11.02**
Van Den Bergh Rule	10.55	11.76	10.91	10.29

The best-performing agent for 3, 4 and 5 player variants of the game was the predictor agent. For the 2-player games, one of the rule-based agents, Piers, performs the best with the Predictor agent performing significantly worse than it does on the

other variants. The Predictor IS-MCTS agent outperforms the standard IS-MCTS variant in all game sizes.

7.6.2 Discussion

The results in table 7.2 show that the predictor variant of the IS-MCTS agent outperformed the version of the agent which did not use prediction. This is due to the fact that the predictor agent is better able to make use of effects of its actions than the standard variant of the IS-MCTS agent. As the predictor variant only considers moves that the paired agents will consider performing when simulating their turns, it has a more accurate picture of the possible final score of the state.

This effect is most evident when evaluating scores for the predictor agent when paired with the Flawed agent. The agent is better able to work around the weaknesses present in this agent and as a result able to obtain a better score.

The 2-player variant of the game seems to pose the largest problem for the Predictor agent, given that it obtained the highest score for the 3, 4 and 5 player variants of the game. It obtained one of the lowest scores for the 2-player variant.

The results show that Predictor IS-MCTS outperforms IS-MCTS in the game of *Hanabi*. This indicates that prediction is beneficial in this domain. However, there are a number of possible objections to this, the most significant being: firstly, the exploration constant of $\sqrt{2}$ could be better suited for Predictor IS-MCTS than IS-MCTS in this domain; Secondly, the poor performance of random play in *Hanabi* will result in very short low-score roll-outs, which may impact the ability of the agent to distinguish promising branches within the tree. That said, as the Predictor IS-MCTS agent makes random moves for its actions in roll-outs it will likely also experience these problems. The rest of this chapter will explore these objections and will also

show that the agent's performance is largely hampered by the partially-observable nature of the domain, and as a result the need to communicate to overcome this limitation.

7.7 Exploration constant

In section 7.6.1, both the IS-MCTS and Predictor IS-MCTS agents used $\sqrt{2}$ as their exploration constant. This section will explore the possible performance gain that can be attributed to altering the exploration constant.

7.7.1 Method

The agent was modified in order to allow specifying the exploration constant on creation. The paired agents used in previous experiments were used for the evaluation. Values of exploration constant between 0 and 2 in increments of 0.25 were used.

The results were grouped both by number of players, to see if different player counts warranted different exploration constants, and analysed by paired agent to see if different paired agents worked better with different exploration constants.

7.7.2 Results

In the graphs presented below, the shading represents the 95% confidence intervals.

Broken Down By Player Count (PredictorMCTS)

When broken down by player counts, the 3, 4 and 5 player games don't seem to show significant differences in performance. The 2-player games show the largest change, with an increase in of average of a point when the exploration constant is set close to zero.

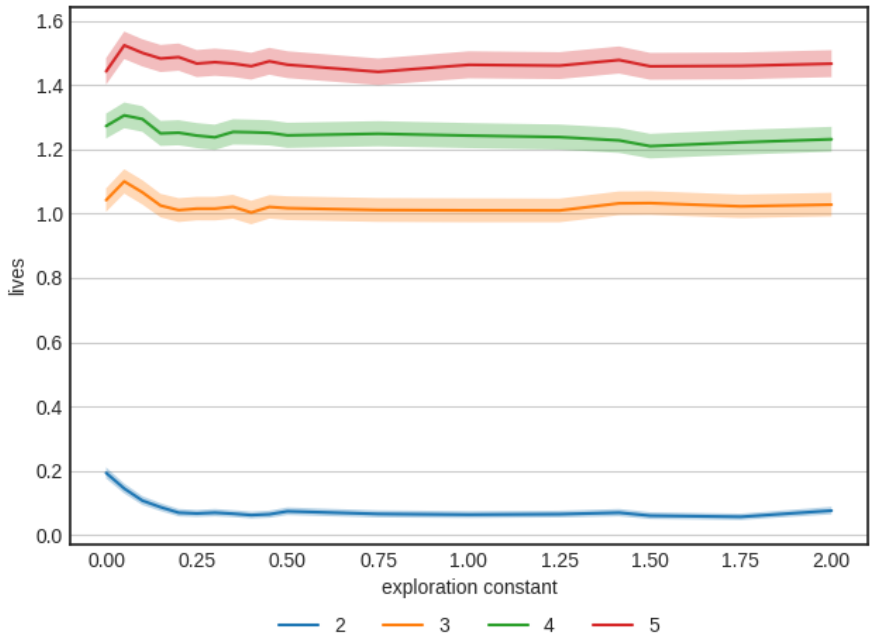


Fig. 7.3 Lives remaining at the end of the match for Predictor MCTS Games.

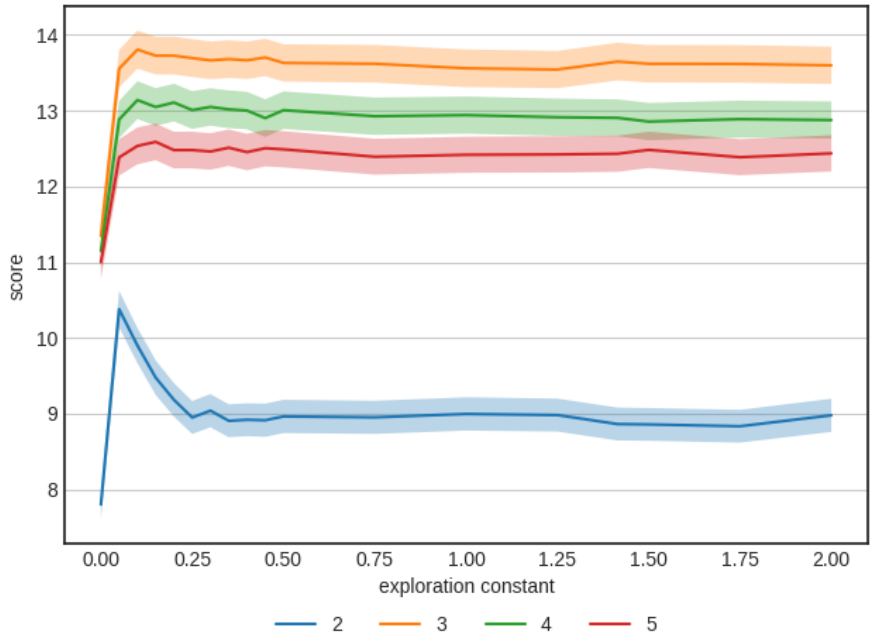


Fig. 7.4 Scores obtained at the end of the match for Predictor MCTS Games

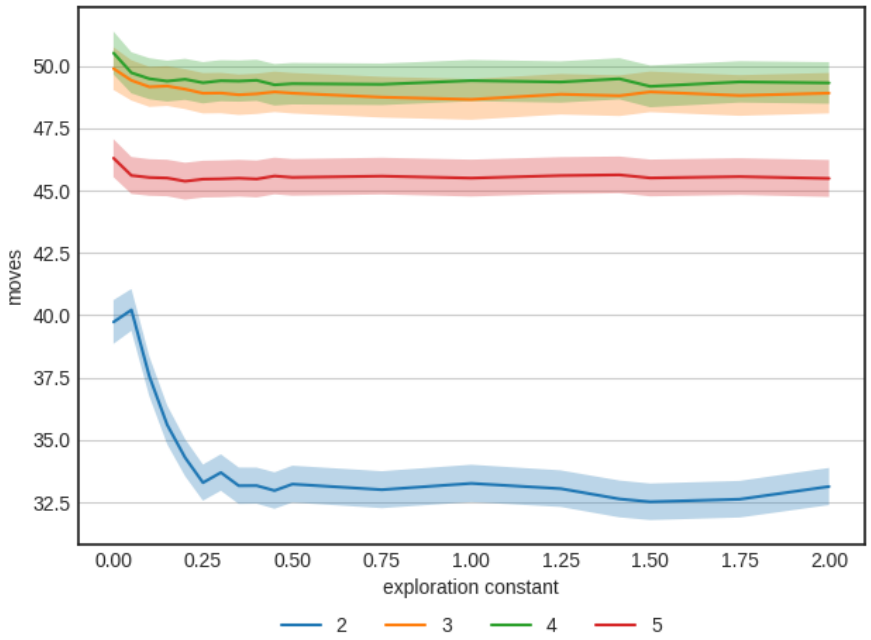


Fig. 7.5 Moves made at the end of the match for Predictor MCTS Games

Broken Down by Paired Agent (PredictorMCTS)

When broken down by agent, the graph seems to follow the curve for most agents (all those except flawed and random).

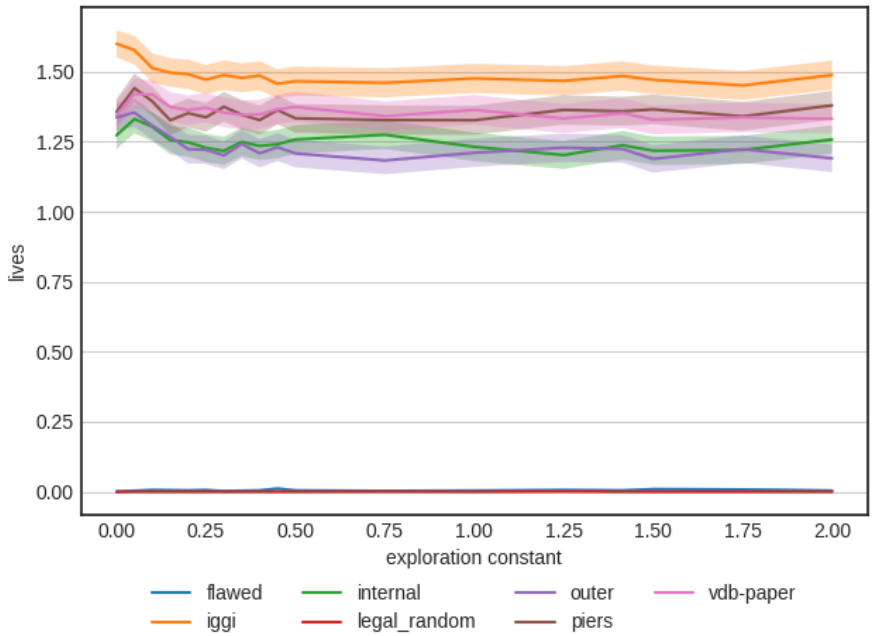


Fig. 7.6 Lives remaining at the end of the match for Predictor MCTS Games

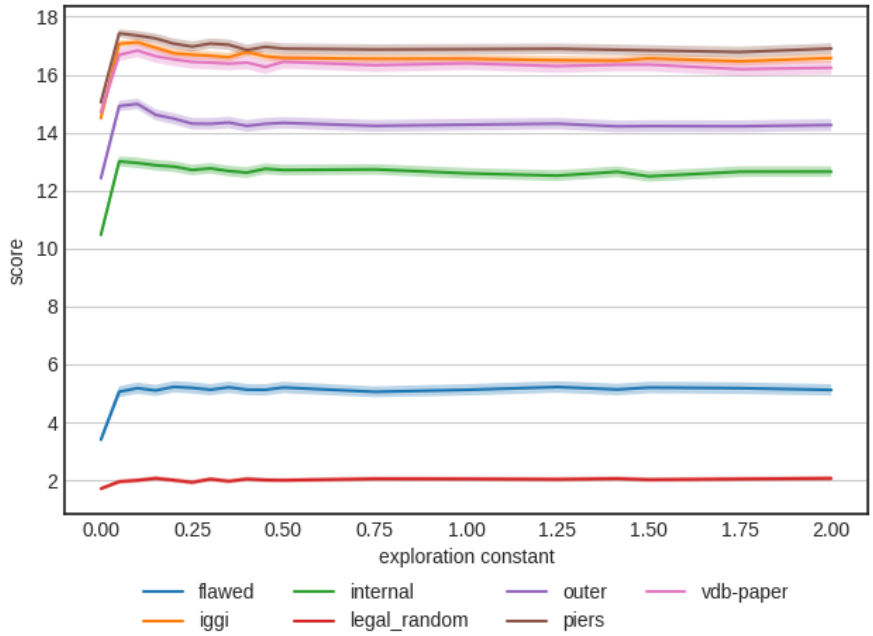


Fig. 7.7 Scores obtained at the end of the match for Predictor MCTS Games

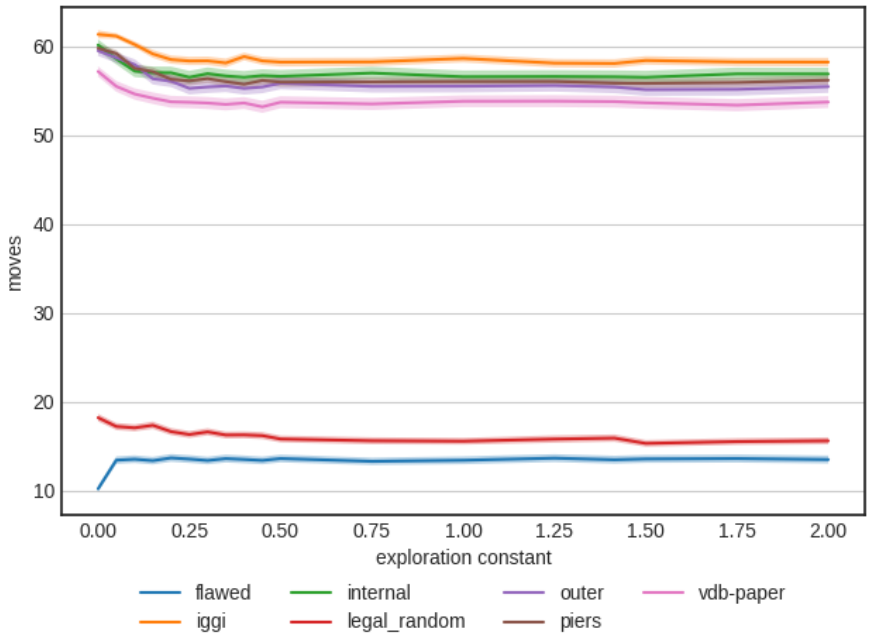


Fig. 7.8 Moves made at the end of the match for Predictor MCTS Games

Broken Down By Player Count (MCTS)

Adjusting the exploration constant for 4-and-5-players seems to be largely consistent. For two players, the games last longer and end with more lives remaining when the exploration constant is set to 0.5. This pattern is not reflected in the scores obtained by the agent.

When looking at the graphs for scores, (fig. 7.10) the same effect cannot be seen. The results follow the same general trend regardless of number of players. For all game sizes, no exploration results in very poor performance in relation to the other values tested. Beyond this value, the graphs show good performance with low values of exploration constant, with performance increasing as the exploration constant increases.

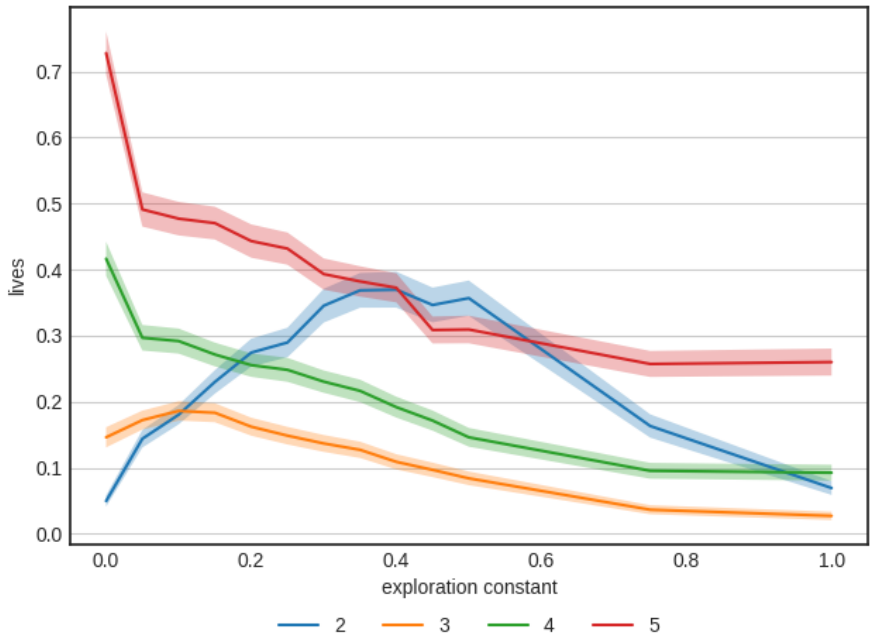


Fig. 7.9 Lives remaining at the end of the match for MCTS games

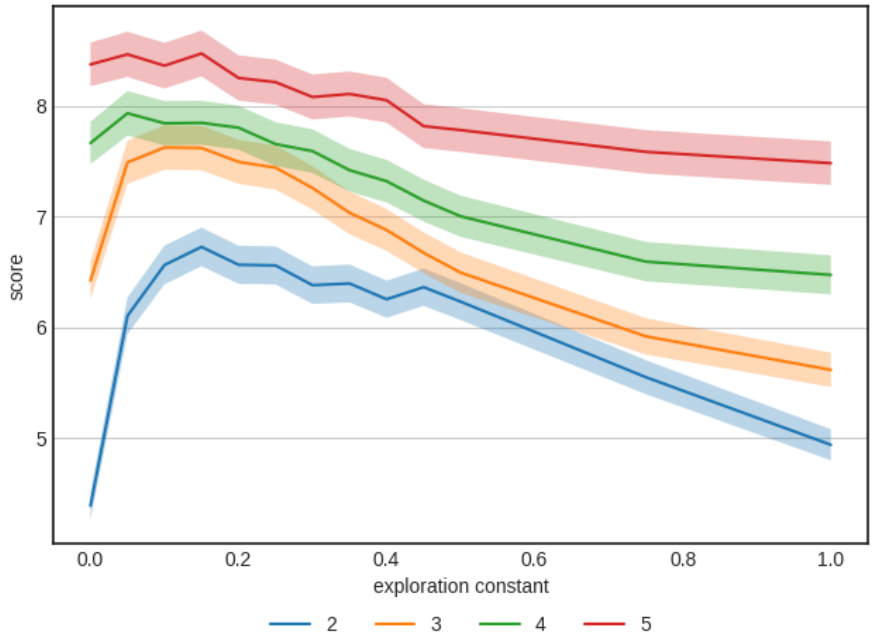


Fig. 7.10 Scores obtained at the end of the match for MCTS games

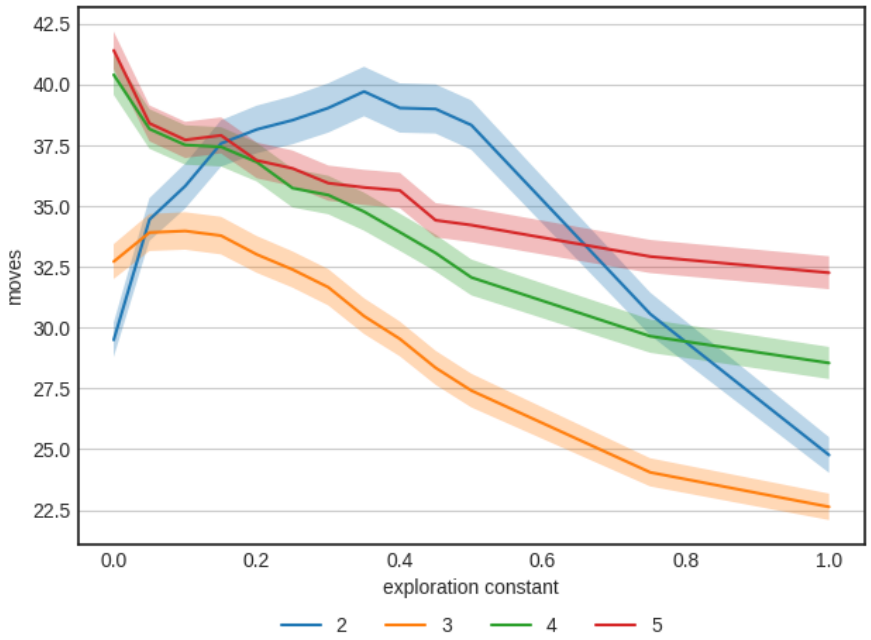


Fig. 7.11 Moves made at the end of the match for MCTS games

Broken Down by Paired Agent (MCTS)

When split by paired agent for Predictor, the optimal value for obtained scores is somewhere around 0.2.

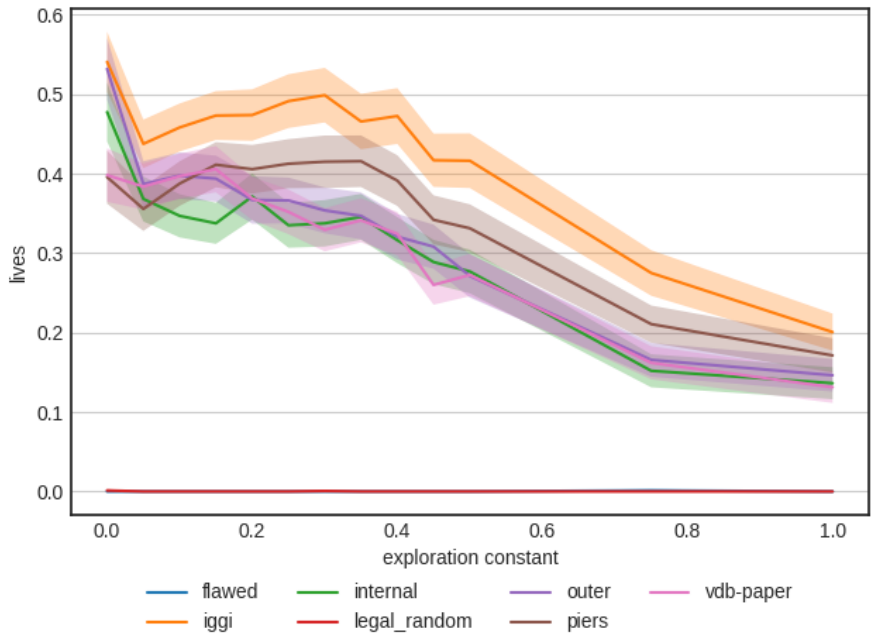


Fig. 7.12 Lives remaining at the end of the match for MCTS games

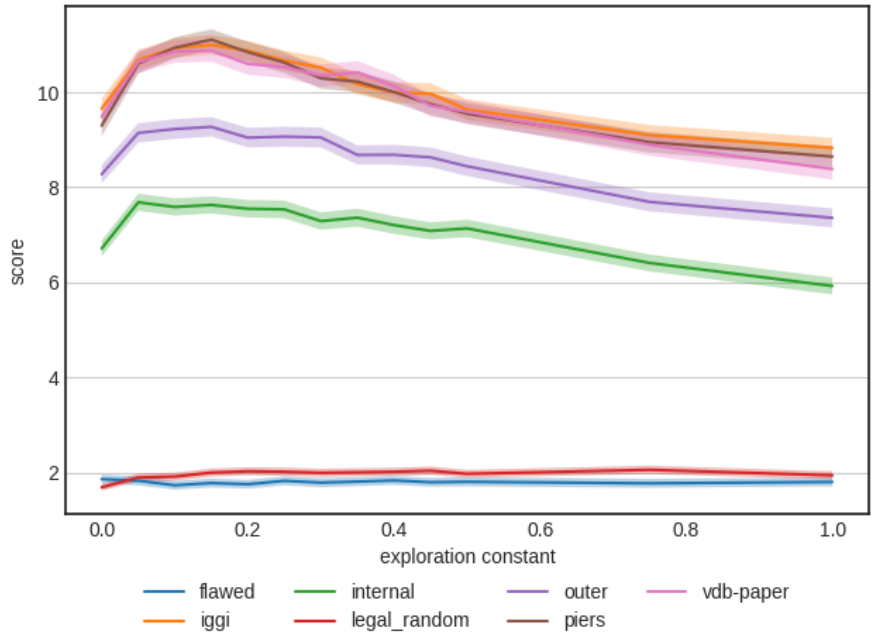


Fig. 7.13 Scores obtained at the end of the match for MCTS games

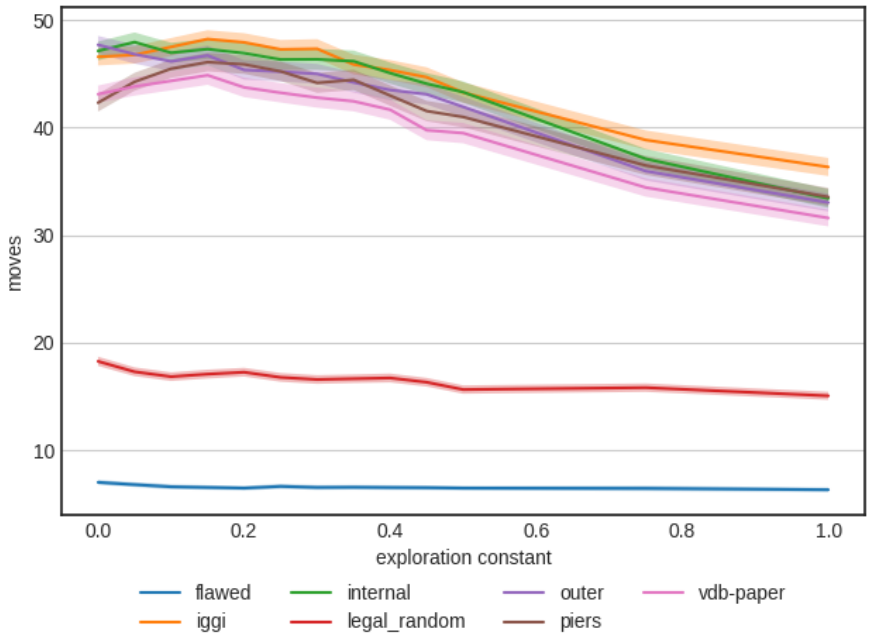


Fig. 7.14 Moves made at the end of the match for MCTS games

7.7.3 Analysis

The results show that both agents experience a drop in performance when the exploration constant is set to zero. This indicates that greedy behaviour is not beneficial in *Hanabi*. Both agents show better performance when using lower values than $\sqrt{2}$ for the exploration constant. Figures 7.6 and 7.12 shows that the exploration constant has no effect on lives remaining for either agent when paired with Flawed and Random. This is to be expected as these agents will almost always result in the game ending by running out of lives. Both of the agents show an increase in moves made when the exploration constant is 0; this indicates that they are not playing enough cards and so the scores for these games are quite low.

The Random and Flawed agents have a large impact on player scores; this is due to the fact they score so poorly. This drags down the average scores when games are

split by number of players. This effect can be seen for both agents, figs. 7.10 and 7.13 for IS-MCTS, and figs. 7.4 and 7.7 for Predictor IS-MCTS.

The scores obtained by IS-MCTS when using a low value for the exploration constant indicate that a value closer to 0.2 will generate an increase in performance in comparison to using $\sqrt{2}$, but the scores obtained are still much lower than those obtained by Predictor IS-MCTS.

The results show that the two-player variant of the game is the most sensitive to changes in the exploration constant. For MCTS, values close to 0.4 show games last longer for two-player games and tend on average to have more lives remaining at the end of the game. This does however not translate to an increase in score for two-player games. For the Predictor agent, choosing a low exploration constant seems also to generate a increase in performance for two-player games, but the result is still lower than the corresponding values for the other game sizes.

For both agents, the 3, 4 and 5 player variants show similar behaviours, following the same general trend for the agents. However, for both agents the two-player variant behaves differently. This reflects anecdotal reports that the two-player variant of the game is quite different in terms of strategy and game experience than the other variants, with one player falling to a 'tell' role and the other acting on the information. In addition, in the two-player variant the agents have access to the least amount of information, as most of the cards are still contained within the deck and are revealed very slowly over the lifetime of the game (only one additional card from the deck can be revealed between moves by the player). This is supported by the experiment that will be presented in section 7.9.

7.8 Policy-Based MCTS

As presented in the results present in table 7.2, an agent which plays randomly performs very poorly. The IS-MCTS agent relies on roll-outs for predicting the value of states and consequently the moves made. As the predictor variant incorporates a model into its roll-out phase, the roll-outs will last longer and therefore might provide it an advantage over random roll-outs. Predictor still makes random moves in roll-outs for its own moves.

This experiment aims to gauge how much of an impact using an intelligent roll-out strategy provides in *Hanabi*. To accomplish this, the policy-based approach used by predictor is incorporated into the roll-out phase of the agent, without modifying the tree. Unlike Predictor IS-MCTS the policy is *fixed* (it does not depend on the paired agents). This experiment can also provide an indication of the difference between using a strong (high-scoring) policy and an accurate policy.

7.8.1 Method

To check the possible performance gain from the agent from the use of policy-based roll-outs, the experiment presented in the above section can be performed on a variant of the agent which has been modified.

The new agent, IS-MCTS Policy, replaces the random roll-outs with the same agent-based system used by the Predictor variant. As the deck orderings can determine the scores obtained by the agent, the MCTS, IGGI and Predictor agents were re-run.

The Policy used for the variant was the IGGI agent, which represents a strong agent within the game framework. The Policy and Predictor agents are very similar, but the Predictor variant uses an *accurate* model, which is also used during the tree

search; the policy-based variant uses a *fixed* model which is only used in the roll-out phase.

The experimental set-up used is the same as that presented section 7.6.

7.8.2 Results

Table 7.4 Results of games using the policy augmented MCTS variant compared with the predictor variants

Agent	2	3	4	5
IGGI	**12.39**	12.05	11.39	10.85
IS-MCTS	4.55	5.50	6.11	7.37
MCTS Policy[IGGI]	10.12	11.06	10.67	10.32
Predictor IS-MCTS	8.93	**13.38**	**12.72**	**12.27**

The results show that the policy-based MCTS agent outperformed the standard MCTS agent. The predictor variant performed better than the fixed-policy variant in the three, four and five player games but performed worse in the two-player games. All the MCTS variants were unable to beat the scores obtained by IGGI for two-player games, only the predictor outperformed it on 3, 4 and 5 player games.

7.8.3 Discussion

The policy-based IS-MCTS variant performs much better than its random-policy based counter part. The Predictor agent still out-performs the agent in 3-,4-and-5 player games. This is consistent with the results of the earlier experiments in this chapter, and the findings of the *Tiny Co-Op* experiments presented in chapter 6. The increase in performance observed by the fixed-policy agent in comparison to the Random policy variant indicates that using a high-scoring policy does have a significant impact on the performance of the agent.

It should also be noted that IGGI is quite similar in strategy to Osawa's agents and that all of the rule-based agents will play cards they know about – therefore in the policy-based roll-outs, games which feature more tell actions before the roll-out phase are likely to obtain higher scores than those that do not.

When paired with the Flawed agent (the rule-based agent which has the least in common with IGGI), the performance shows the least improvement. The poorer performance in two-player games can still be seen in the agent. However, the difference is much less pronounced. The partially-observable nature of the domain poses a problem for search-based techniques as they must reason over the possible states they can be in. In the two player version, the agents have access to the least amount of information. To see how much of an effect this has, the same deck orderings and agents were re-run in a version of the game where the agents are told about their cards (section 7.9).

7.9 'Open Hand' Variant

To investigate if the cause of the poor IS-MCTS agent's performance in two player games could be attributed to the partially-observable nature of the game, the experiment presented above was also run with a 'cheating' variant of the game. In this variant, agents were informed about the cards they picked up. The agents will always know what cards they have in their possession and so no longer need to use tell actions to communicate with each other. The order of the cards remaining in the deck remains unknown.

Table 7.5 Results from fully observable games

Agent	2	3	4	5
IGGI	21.49	21.96	21.76	21.76
IS-MCTS	20.73	21.74	21.85	21.66
mctsPolicyND[iggi]	20.77	21.46	21.04	20.92
Predictor IS-MCTS	20.97	21.82	21.65	21.64

Table 7.6 Results from fully observable games, excluding random and flawed

Agent	2	3	4	5
IGGI	24.11	24.86	24.81	24.81
IS-MCTS	23.30	24.43	24.46	24.63
mctsPolicyND[iggi]	23.55	24.52	24.15	24.06
Predictor IS-MCTS	23.39	24.46	24.54	24.61

7.9.1 Results

Agents Split by Number of Players

The results in table 7.5 shows that there is not much difference between the agents when the hidden information component of the game is removed. When the two agents which do not exhibit intelligent play are removed from the statistics, (table 7.6) the remaining agents obtain almost perfect scores (a perfect score being 25). In addition, the scores for the two-player and three-or-more variations of the games are much closer when playing with perfect information.

Games Split by Paired Agent

The results in table 7.7 show that the agents were able to play with all agents except for random.

Table 7.7 Agents split by paired agent

Evaluated Agent	**Paired Agent**	**Score**	*plusminus*
IGGI	flawed	24.52	0.03
	iggi	24.63	0.02
	internal	24.68	0.02
	legal_random	4.46	0.08
	outer	24.68	0.02
	piers	24.58	0.03
	vdb-paper	24.66	0.02
IS-MCTS	flawed	23.95	0.03
	iggi	24.10	0.04
	internal	24.30	0.03
	legal_random	5.01	0.08
	outer	24.28	0.03
	piers	24.06	0.04
	vdb-paper	24.29	0.03
mctsPolicyND[iggi]	flawed	23.94	0.03
	iggi	24.07	0.03
	internal	24.07	0.03
	legal_random	3.04	0.07
	outer	24.05	0.03
	piers	24.10	0.03
	vdb-paper	24.06	0.03
Predictor IS-MCTS	flawed	24.15	0.03
	iggi	24.17	0.03
	internal	24.32	0.03
	legal_random	5.26	0.09
	outer	24.32	0.03
	piers	24.17	0.03
	vdb-paper	24.27	0.03

7.9.2 Discussion

From these results, it is clear that the difficulty from the game *Hanabi* comes from its hidden-information mechanic. This increase in performance is consistent with the findings of (Whitehouse *et al.*, 2011), who saw a similar increase in performance comparing a 'cheating' version in other games. The agents can simply play the cards in their hand in the correct order, using information and discard moves to skip their turn if they do not have a required card.

Revealing the cards to the players also removes the need for the agents to communicate information with each other. As there is little benefit from predicting player behaviour, the difference between performance of the IS-MCTS and Predictor IS-MCTS agents is removed. Likewise, the increase in performance for the fixed-policy variant is also reduced.

Interestingly, the agents seem to still perform slightly worse in the 2-player variant of the game when playing with perfect information. However, the difference that can be observed in the ordinary results is much less pronounced. This supports the assertion that the lack of available information is the primary cause of the poor agent performance. As the game does not require tell actions to communicate information, the need to co-operate beyond playing the next-best card in your hand is removed from the game and there is little benefit to predicting the agents behaviours.

As the game is co-operative, the agent's performance is still affected by the players that they are paired with. The agents that exhibit poor strategies will still have worse scores than agents that do not. The poorest agents in the framework (Legal Random and Flawed) cause drops in the average scores obtained by the agents as they tend to end the game quickly.

There does not seem to be a difference in performance between using IGGI for 'heavy' roll-outs and using random roll-outs. The scores for both agents are very similar. Even though the agents they are paired with were not designed to work in the 'open hand' version of the game, as they all played the same deck orderings results in this table are directly comparable. As the roll-outs will typically be quite short, the score will be dominated by the score obtained in the tree. For the fully-observable variant of the game, the agent is likely to have at least one playable card (which will generate a guaranteed increase in score); as very little long-term planning is needed, this will probably be enough to allow it to obtain a good score. This implies that the partially-observable nature and the need to co-operate with other agents is the source of the complexity for the agent.

The rule-based agents were not designed to work in a fully-observable version of the game, and so will make sub-optimal moves (they will not consider which playable card will allow the next agent to play a card, and so will play the first card in their hand which is playable). This potential flaw in their strategy may occasionally prevent perfect scores from being achieved by the group. The agents will also be unable to discard cards without a tell-action being performed.

7.10 Conclusion

This chapter outlined the benefits of the Predictor agent in the card game *Hanabi*. The agent performs better in 3-,-4-and-5 player variants of the game. The chapter also demonstrated that the main cause of poor performance for the IS-MCTS agent is its lack of ability to tell other players about their cards.

The Predictor IS-MCTS agent significantly outperformed the standard IS-MCTS agent in all game sizes. The agent was also the best-scoring agent in 3, 4 and 5

player games. This seems to match the findings presented by (Barrett *et al.*, 2011). This indicates that the addition of prediction to the MCTS agent proves beneficial for the card game *Hanabi*.

The poor performance demonstrated by IS-MCTS in section 7.6.1 can be attributed to two causes. Firstly, the use of random roll-outs: replacing these with a strong policy, most of the performance gap between itself and Predictor IS-MCTS disappears. As most of the agents are based on a variant of IGGI, the agent may also get some benefits from an accurate strategy being used as part of the roll-outs.

Secondly, the partially-observable co-operative nature of the domain. When removing the partial-observability and the need for communication (section 7.9), the IS-MCTS agent is capable of playing almost perfectly. The difference in score observed for two player games is also significantly decreased. This indicates that the communication and partial-observability aspects of the domain are the problematic aspects of the domain. This mirrors the findings of (Whitehouse *et al.*, 2011) which found that 'cheating' agents outperformed the IS-MCTS variants tested.

This work assumes that a perfect model of the modelled agent's behaviour is available. Even if it is available, then it may be computationally infeasible to execute as part of the search. The next chapter presents work related to learning agent strategies using game traces to create rule-based approximations of agent performance.

Chapter 8

Learning models for *Hanabi*-playing agents

In chapters 6 and 7, the agent that had been modified to use a 'model' of the other agent's behaviour was able to obtain better scores than a corresponding agent that assumed random play or used a fixed policy. The technique makes use of copies of the paired agent's strategy. This is not always possible, either because it is too computationally expensive (e.g. when playing with Monte Carlo Tree Search (MCTS)) or because a copy of the strategy is not available (e.g. when playing with humans).

This chapter investigates a method for generating rule-based models that approximate the behaviour of an observed agent. This is accomplished by learning individual strategies from game traces. If the approximation is sufficiently close, the model can then be used in place of the expensive evaluation function presented in earlier chapters.

This is done in three stages: firstly, a tolerance for the accuracy of the model is obtained (section 8.1); next, a model is created using a genetic algorithm (section 8.2); finally, the effectiveness of the model is evaluated by comparing the 'perfect' models to the 'learnt' ones (section 8.3).

8.1 Noise Tolerance

Many of the early agents in the 2 player General Video Game AI (GVG-AI) competition used a completely random paired agent model. From the work presented in chapter 7 in chapter 6 the evidence is that a model may prove helpful in the games explored in this thesis.

Using a 'perfect' model based on delegating to a copy of the agent has limitations. Namely, that this might not always be available. Instead, some learnt model may be preferable. To obtain a target for the accuracy required for that model this section proposes a preliminary experiment. By adding noise to the 'perfect' model of the paired agent, the level of accuracy (correct predictions) required can be predicted.

To simulate the effects of an inaccurate model, a new model which 'wraps' around an existing agent was created. This is shown in algorithm 1. The model wraps an agent (π_w) and samples from it with some probability ϵ, else it chooses a random legal move from the set of possible moves. As the number of times the model delegates to the real agent decreases, the accuracy of the model will also decrease.

This provides a basic approximation of the effect of noise on the models which the predictor agent is using. As making random moves represents a very poor player, this should represent a 'worse case' scenario. This model does not factor in if the

Algorithm 1 The Noisy Model

```
1: function DoMove(S)
2:     p ← random()
3:     if p > ε then
4:         return π_w(S)                          ▷ Choose from the wrapped policy
5:     else
6:         A_legal ← filter(A, S)                 ▷ Calculate moves that are legal for this state
7:         return random(A_legal)                 ▷ Return a random (legal) move
8:     end if
9: end function
```

agent is actively making informed but incorrect choices, but even these should perform better than random play.

8.1.1 Method

The same basic methodology that was used in chapter 7 is used for this experiment. However, rather than testing with different algorithms, the agent used is always the predictor agent. The parameter being varied is the 'accuracy' of the model.

The agent was paired with the same agent battery presented in section 7.4. The values for 'accuracy' used varied from 0 (completely random moves) to 1 (always consult the 'real' model) in increments of 0.1.

8.1.2 Results

The graphs presented in this section show the 95% confidence intervals as a shaded area round each of the lines.

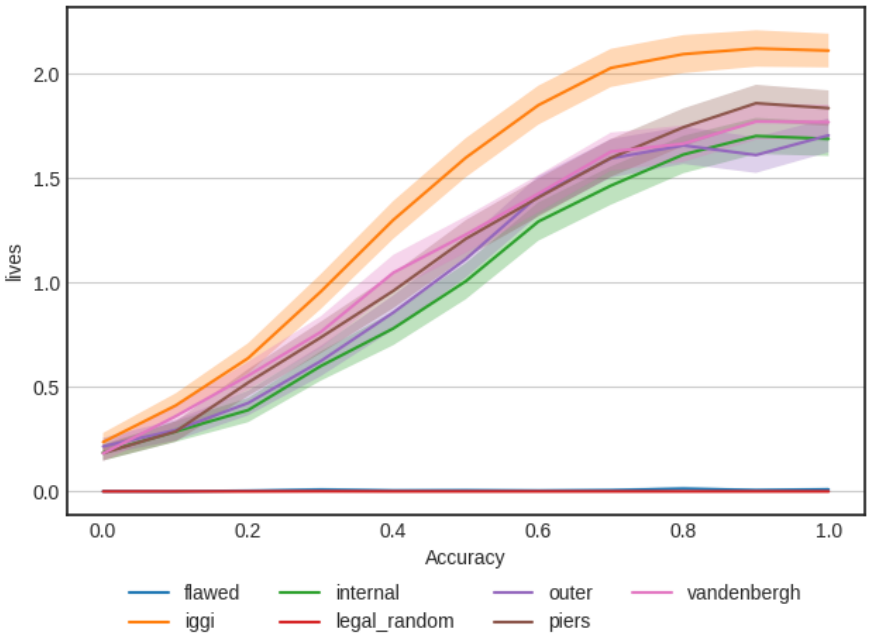


Fig. 8.1 Lives remaining using noisy predictions

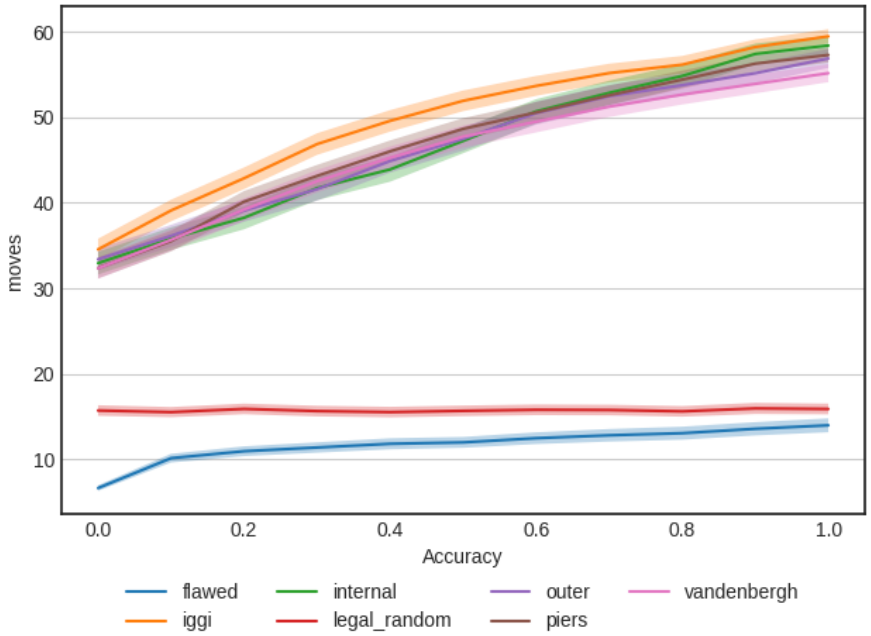


Fig. 8.2 Moves made using noisy predictions

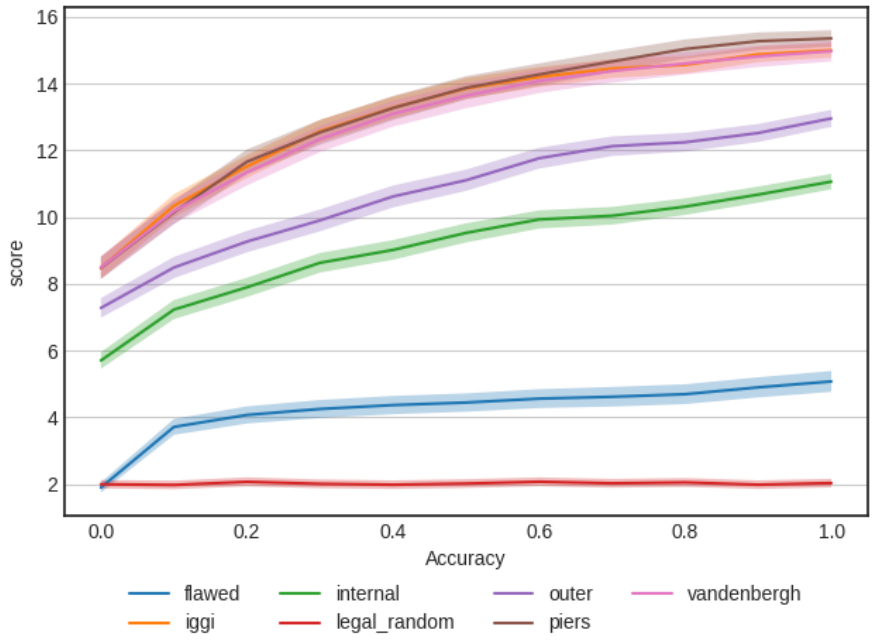


Fig. 8.3 Final scores using noisy predictions

8.1.3 Discussion

From these graphs, reducing the accuracy of the model clearly reduces agent performance. When the 'accuracy' is 0, the scores obtain results similar to those exhibited by the standard Infomation Set Monte Carlo Tree Search (IS-MCTS) agent. This is to be expected as the IS-MCTS agent uses random moves as part of its roll-out process. As accuracy increases, the scores of the agent approach those obtained in the earlier experiments.

Most rule-based agents tested seem to follow the same performance curve. Increasing accuracy when playing games with the random agent has no effect, as both sides of the graph represent random play. The same curve is shown for lives remaining, moves and final score obtained.

The lower number of moves is a result of the game ending prematurely: the agents have made more mistakes, and so lost more lives. This is shown in fig. 8.1, that indicates the number of lives remaining at the end of the game is on average lower. As the other agents have not been altered, we can conclude that it is the poor performance of the model which is having this effect.

The graphs show that a value of between 0.7 and 0.8 represents a reasonable level of accuracy for the model. Below this level, the number of lives remaining at the end of the game fig. 8.1 starts to decrease quite sharply. The effect in both the lengths of the game fig. 8.2 and the final score obtained fig. 8.3 also shows this decrease, however in both of these graphs the effect is less pronounced.

Therefore, when training the model, an accuracy of the predicted moves of 0.7 (70% of the predictions made being correct) should be considered acceptable. Below this value, the agent would exhibit the same drop in performance as observed in this

experiment. With actual learnt models, this may be less pronounced as it is likely they will provide a better approximation than random play.

8.2 Model Learning

As a result of the preliminary experiment described above, the target accuracy required was set at 0.7 (the model should make accurate predictions at least 70% of the time). To produce a model, the rule-based agent system which is part of the framework was used.

The method used to create the models was a Genetic Algorithm (GA). This was because a GA has the ability to produce a human-readable representation of the model and the resultant model would match the output of the rule builder. As a bonus, this meant that the models were easy to import into the games without much additional work.

The representation used for the models was a fixed-size integer array. Each number in the array represented a different production rule in a 'master list' (See chapter A) of the rules used to build agents. Some rules were parametrised, for example, playSafeCard; in these cases, multiple copies of the rules with different parameters appeared in the list. A value of -1 was used to indicate that the agent should stop processing rules. This allowed for variable-length models to be represented.

If two rules are mutually exclusive, they can be re-ordered without any effect. For example, a rule which will only fire if the player has at least one information token and a rule which will only fire if the player has no information tokens will be considered for the same state, so their order in the rule-set does not matter. Some rules, such as the parameterised rules, can 'eclipse' rules below them in the list as

they are more general versions of later rules. For example, if a rule such as 'play a card if you are 80% certain that the card is playable' is placed after a rule which states, 'play a card if you are 60% certain that the card is playable' then the 60% rule will always fire if the 80% one could fire and so the 80% rule will never execute.

These situations can be detected by including domain-specific rules about when rules would or would not fire, but they are difficult to detect when applying operations to the genome. In order to speed up evaluation, the resulting agent is 'trimmed' to remove useless rules, which results in a greater number of evaluations being possible in the same time period.

As rules in the framework are deterministic (but can have non-deterministic effects) and the first rule which reports that it can execute will be used, it is not necessary to include duplicate rules in the agents. As a result, the models do not need to include duplicate rules either.

8.2.1 Method

Genetic Algorithm

For this work, the Jenetics (*Jenetics* n.d.) library was used. This is a library that has found adoption with the academic literature. The algorithm used for this work had a population size of 30 individuals, with an offspring fraction (amount of the population regenerated each generation) of 0.7. The rest of the population (0.3) is retained from the previous generation (elitism). Roulette wheel selection is used to choose the survivors for the next generation. Tournament selection (tournament size of 2) is used to select parents for generating the offspring.

These parameters are close to the defaults used by the library selected for generation. The choice of mutators for generating offspring has been selected based on the way

in which agents are defined. As agents are defined as an non-repeating integer sequence (rule indices) standard selection and mutation operators are not well suited to the problem (as these may introduce duplicates). Instead, specialist variants of crossover and mutation which were originally designed for dealing with the travelling salesperson problem have been used. The crossover variant used (Partially Matched Crossover) guarantees that no duplicates are present within the solution. It does this by performing a 2 point crossover and replacing duplicates outside of the crossover region if there are duplicates.

Model Training

To evolve a model which is capable of emulating the behaviour of an agent, the game logs for games containing that agent are extracted from the set of game logs created from previous experiments.

A candidate model is evaluated by testing its 'agreement' with the observed action in the game log. The game state is reconstructed from the log data and the model is passed the game from the agent's point of view. The resulting action is compared against the recorded action in the game log. The agent is awarded a point if the score obtained matched.

This causes problems for actions in the game log which were determined using a non-deterministic rule. The agent will produce a different action to the observed action even if the rule chosen was correct. This could be avoided by keeping track of the rule used to generate the move in the game logs, but this would defeat the purpose of this research: it would be trivial to reconstruct the agent if the rules chosen are known.

Instead, the model is asked 5 times for an action. Every time the action predicted agrees with the chosen move, the agent is awarded $^1/_5$ of a point. This gives moves that are truly random more of a chance to return the correct answer. The move from the game log is fixed, so if the agent made a move that looked intelligent a deterministic rule which returned that action would of course achieve a score of 1 for this state. However, when given another state either from the same game or a different game it would obtain a score of 0, whereas the rule that features random moves would be awarded a partial score if any of the moves matched. More specific, non-deterministic rules such as “play a random card" as opposed to “perform any random legal action” are likely to choose the ‘correct’ move more often as their pool of possible moves is smaller.

8.2.2 Results

Table 8.1 Results of evolution of the agents

Agent	Best Model Accuracy
Internal	0.41
Outer	0.74
Van Den Bergh Rule	0.79
IGGI	0.95
Piers	0.90
Flawed	0.75
IS-MCTS	0.47

The results presented in table 8.1 show the accuracy obtained for the best model for each set of games. The generated models that had the highest accuracy are IGGI and Piers. Most of the other agents showed accuracies of between 0.7 and 0.8. This is within the range predicted by the preliminary experiments. The poorest models were Internal and IS-MCTS, both of which achieved less than 50% accuracy.

8.2.3 Discussion

For most agents, the target accuracy could be obtained, with the notable exceptions of Internal and IS-MCTS. The poor accuracy of the Internal model is due to a quirk of the agent – this agent will tell information which is already known. In ordinary play, this behaviour often serves no purposes (doing so is a waste of information tokens) as result, this behaviour is poorly represented in the ruleset (only the tell rule created especially for internal will exhibit the behaviour). In order for this rule to be used, it must be the first tell rule in the generated model, making it unlikely to be selected. It is possible that increasing the population size or number of generations could fix this issue.

The agents Piers and IS-MCTS cannot be accurately represented by the rule-based architecture, as Piers includes the 'if' construct that was not available to the GA and IS-MCTS is not based on a rule-based technique. As a result of the 'if' statement, the behaviour of the Piers agent would be expected to be lower than 90%. The 'if' rules act as special cases and therefore this level of accuracy could indicate that these rules do not result in a large number of unpredictable moves. These rules *do* improve score, as the are largely aimed at risky play during the endgame (when prediction is less important).

This shows that it is possible to recover a model of the agent's behaviour based on the game history. The model representation is such that a textual description of the generated agent can be provided. This indicates it may be possible (with a suitable amount of data and broad enough rule set) to describe human strategies by using this recovery method. Doing so is well outside the scope of this thesis, but could prove to be useful future work in allowing games with Artifical Intelligence (AI) and human players.

The ability to learn a model of agent behaviour based on previous performance is not a novel idea. Since this work was done, learnt models of *Hanabi* players behaviour have been used and have shown a good level of performance (Goodman, 2019 within this framework for the *Hanabi* competition and Foerster *et al.*, 2018 for the Hanabi learning environment). The main advantages of this approach is that it has the potential to generate human-readable models from game logs and its comparatively simple to implement.

8.3 Evaluating Model Performance

Once the model had been obtained, it was now possible to evaluate their use with the predictor agent. To do this, the results of games played using the model are compared with the results with those obtained by the a perfect model.

The IS-MCTS and Monte Carlo Search agent variants were also used as models for comparison purposes. In general, these agents are poorly-suited for use as models, due to their computational requirements as demonstrated in chapter 6.

8.3.1 Results

Table 8.2 shows the performance of the learnt models and the 'perfect' ones. The performance between the agents is largely similar. The internal model shows the largest decrease in performance between the learnt model and the perfect one. The MCS-based agents and the MCTS agent show a larger increase in performance with the model in comparison to their perfect versions. Most of the rule-based agents show no significant difference in performance between the models.

Table 8.2 Comparison of results between perfect models (copies) and learnt models

Agent	Model	Perfect
Monte Carlo Search-Flawed	5.44	1.81
Monte Carlo Search-IGGI	15.12	6.83
Monte Carlo Search-Legal Random	4.92	2.51
Flawed	4.84	5.10
IGGI	15.06	15.06
Internal	9.65	11.00
IS-MCTS	5.74	2.74
Outer	12.51	12.93
Piers	15.61	15.49
Van Den Bergh Rule	14.71	14.99

8.3.2 Discussion

The results show that the model-based approaches closely match the scores obtained by the perfect models for IGGI, Piers, Outer, Flawed and Van Den Bergh Rule. These models obtained good accuracy results in the first state of the investigation and so this is to be expected.

IS-MCTS and the Monte Carlo Search variants performed worse than their models; this is due to the agents consuming the host agent's time budget and therefore starving it of iterations. When using the models, the host agent benefits from increased iteration counts as the models are less computationally-intensive to compute.

8.4 Conclusion

This chapter outlines a way in which the effectiveness of a model can be assessed and shows that models which feature incorrect decisions reduce the overall effectiveness of the Predictor agent.

From this experiment, it can be concluded that the performance of the predictor is dependant on the accuracy of the model: the more accurate the model, the better the agent will perform. An 'accuracy' of no less than 0.7 should be used for the model.

Using a GA, it was possible to recover the strategies used by the agents within game traces to this accuracy level. It was also possible to use the rule-based framework to represent non-rule based agents within the framework. When the models were used, the predictor agent was able to obtain reasonable scores.

From the data, it can be observed that we were able to obtain reasonable accuracy from some of the agents. The fully-deterministic agents created more accurate models than the non-deterministic ones. The technique was able to identify a model for the non-rule based agent IS-MCTS with a reasonable degree of accuracy. This was because the GA learned that the IS-MCTS agent was capable of playing cards correctly if it knew about them, but it did not seem to tell reasonably. This is consistent with the predictions about how this agent performs in chapter 7.

These results show that Predictor IS-MCTS is capable of using a learnt model and therefore can be used when a perfect model of the agent is not available, provided that game traces are available. This technique would not be suitable for on-line learning of a model, as it requires quite a large amount of previous game data.

Part III

Conclusions

Chapter 9

Conclusions

This chapter concludes the thesis, as well as discusses some future work.

9.1 Conclusions

This thesis aimed to address if prediction for Monte Carlo Tree Search (MCTS) provided to be beneficial for co-operative domains. Based on the experiments conducted in two co-operative games, a grid-based simultaneous move game (presented in chapter 6) and a partially observable card game (presented in chapter 7), it can be concluded that prediction is beneficial. The results show an increase in performance in the two games when paired with other agents.

In *Tiny Co-Op*, introducing a model of paired agents' behaviour proved beneficial to the performance of the agent. This resulted in a significant increase in score when paired with both an intelligent agent and an agent which relied on communication to make decisions. In both these cases, the agent performed better than assuming a random paired agent model. There are many variants and adaptations to MCTS

which could enhance the performance of the agents. However, to demonstrate if prediction makes a difference in co-operative domains using the sample open-loop approach from General Video Game AI (GVG-AI) has shown an improvement in performance.

Existing work in opponent modelling for competitive games shows that agent modelling tends to work well across algorithms and approaches. Therefore it is likely that co-operative player modelling will show similar benefits to other variants of MCTS. This is outside the scope of this thesis, but none-the-less would be a good candidate for future work. Work in integrating opponent models into MCTS have shown that it performs well Ponsen *et al.*, 2010 and Recent work which has compared opponent modelling in Real-Time statergy games (RTS) games has hinted that MCTS may be less robust to a poor opponent model than Genetic Algorithm (GA) based approaches (Goodman and Lucas, 2020).

The experiments with *Tiny Co-Op* presented in chapter 5 showed that MCTS was correctly able to simple co-operative puzzles. In chapter 6 two different kinds of communication actions were explored. The 'full' action set proved to be too complex for use in experiments, due to its large branching factor. When comparing an agent which takes into account the decisions made by a partner to a version which does not take into account modelling, the agent which modelled the paired agent's behaviour performed better. However, the technique was not suitable for use with computationally-intensive agents.

Chapter 7 showed an extension of this agent to work in a partially-observable domain. When paired with rule-based strategies both novel and from existing literature, the agent with prediction showed a significant performance boost in comparison to the agent that did not use prediction. When exploring objections to

this approach, it is shown that the factor which has the largest impact on the agent performance is the partially-observable nature of the domain. However, removing this also removes the need for agents to communicate and therefore negates the advantage that modelling provides.

Using a 'perfect model' isn't always possible, either because of the computational requirements of evaluating the model, or because such as model is not available. To address the limitation of requiring a perfect model of agent behaviour, chapter 8 presented a technique for learning agent behaviours based on game traces. The initial experiments indicated that a model accuracy of $70\% - 80\%$ is needed for good performance. For most agents evaluated, it was possible to achieve this accuracy. It was also possible to learn a model of the MCTS agent, which is not composed of a fixed strategy representable by the rules. Even though the accuracy of the MCTS model was poor, it still out-performed using the raw MCTS agent as a predictor, as using the latter starves the host agent of its budget. For most other agents, the model obtained slightly lower performance than the perfect model, but the results are quite close.

The work presented in chapter 7 focused on replacing one member of a team of agents with MCTS. Since this work was conducted, competition entries to the *Hanabi* have expanded the work presented in this thesis and have shown promising results Goodman, 2019. In addition, there has been work into using GAs to generate strong rule-based agents. This has also included adding new rules to the framework. In addition, from the *Hanabi* learning environment work, a good deal of progress has been made in learning strategies for the agents.

Together, these results show that overall, prediction of paired agent behaviour in co-operative domains does improve the performance of MCTS. This is true both

when using predictive models that are perfect appropriators of agent behaviour and when using ones built from logs of previous games.

As a basis for the work presented in the motivation, this work has showed that from a performance perspective there is a performance increase for implementing modelling for these simple games. This indicates that agent-models are worth-while integrating into game-playing agents within co-operative components. This work shows potential for modelling-based approaches. There are other approaches that should also be considered such as (Guckelsberger *et al.*, 2018), which has also been evaluated in a similar grid-based environment for companion/adversarial Artifical Intelligence (AI).

9.2 Limitations

The work presented in this thesis outlines some important considerations for co-operative games, with a long term view to seeing this work adapted for video game AIs. Given available time constraints and the scope of the work conducted, there is a still a very long way to go before this work can be integrated into real-world games. However, this work does suggest that it is possible to use player models to assist in games that feature co-operation.

The research conducted in chapter 5 showed that MCTS could work in a purely co-operative game without much modification. The agent presented is similar to the initial design of the sample MCTS controller for the 2 player GVG-AI competition (as described by Liu *et al.*, 2016) and indeed many of the initial entries in the GVG-AI competition used a similar approach - assuming a random opponent model (Perez-Liebana *et al.*, 2019).

There is significant research in how MCTS can be adapted for simultaneous move games which should be examined to allow a broader evaluation of this work. At present, this comparison cannot be conducted due to the lack of computing resources available. This none-the-less is an important topic which needs to be addressed. This limitation is also present in the work conducted in chapter 6, which adapts the random-move model of the work presented in chapter 5.

The *Hanabi* work could be expanded to games which feature more than two types of agents. The framework already supports this, as it was used for the COG 2019 *Hanabi* learning track. At present, a full evaluation of this work is still in its infancy as the competition featured the requirement to learn the models on-line, but this is an exciting future area of research.

9.3 Future Work

While the results of this work are promising, there are a number of areas for future research. Firstly, this thesis only looks at fully co-operative domains. Extending the work to multi-player games where the agents may not be fully-cooperative or which feature hidden objectives would be a logical next step. However modelling behaviours in these games is more complex. In addition, fully co-operative games where agents take on different roles, such as *Forbidden Desert* or *Pandemic* (as proposed by Chacón and Eger, 2019), would also be an interesting area of study, as the agents' roles may provide hints that could aid in modelling. Another area of future work which would be promising is team-vs-team games (where one of the teams might be an individual). Suitable games for this work include Scotland Yard Nijssen and Winands, 2012, or possibly MOBAs.

Another possible area of research is learning models without access to a previous set of game logs. This would require learning about agent strategies on-line based on observed patterns. The 'learning' track of the *Hanabi* competition at COG 2019 aims to provide a scenario for exploring this.

References

Abdoun, Otman and Jaafar Abouchabaka (2012). "A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem". In: *arXiv preprint arXiv:1203.3097*.

Afonso, Nuno and Rui Prada (2009). "Agents that relate: Improving the social believability of non-player characters in role-playing games". In: *Entertainment Computing-ICEC 2008*. Springer, pp. 34–45.

Albrecht, Stefano V and Peter Stone (2018). "Autonomous agents modelling other agents: A comprehensive survey and open problems". In: *Artificial Intelligence* 258, pp. 66–95.

Anderson, John, Jacky Baltes, and Chi Tai Cheng (2011). "Robotics competitions as benchmarks for AI research". In: *The Knowledge Engineering Review* 26.1, pp. 11–17.

Ashlock, Daniel and Garrison Greenwood (2016). "Generalized divide the dollar". In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 343–350.

Ashlock, Daniel, Diego Perez-Liebana, and Amanda Saunders (2017). "General video game playing escapes the no free lunch theorem". In: *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 17–24.

Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer (2002). "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2-3, pp. 235–256.

Baier, Hendrik and Peter I Cowling (2018). "Evolutionary mcts for multi-action adversarial games". In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 1–8.

Baker, Bowen *et al.* (2019). "Emergent tool use from multi-agent autocurricula". In: *arXiv preprint arXiv:1909.07528*.

Bard, Nolan *et al.* (2019). *The Hanabi Challenge: A New Frontier for AI Research*. arXiv: 1902.00506 [cs.LG].

Barrett, Samuel, Peter Stone, and Sarit Kraus (2011). "Empirical evaluation of ad hoc teamwork in the pursuit domain". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 567–574.

Bartle, Richard A. (2004). *Designing virtual worlds*. Indianapolis, Ind: New Riders Pub. ISBN: 978-0-13-101816-7.

Bates, Joseph (1994). "The role of emotion in believable agents". In: *Communications of the ACM* 37.7, pp. 122–125.

Bellemare, Marc G, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Berlekamp, Elwyn R, John H Conway, and Richard K Guy (2018). *Winning ways for your mathematical plays*. AK Peters/CRC Press.

Blickle, Tobias and Lothar Thiele (Dec. 1996). "A Comparison of Selection Schemes Used in Evolutionary Algorithms". In: *Evol. Comput.* 4.4, pp. 361–394. ISSN: 1063-6560. DOI: 10.1162/evco.1996.4.4.361. URL: http://dx.doi.org/10.1162/evco.1996.4.4.361.

Bnaya, Zahy, Roni Stern, Ariel Felner, Roie Zivan, and Steven Okamoto (2013). "Multi-Agent Path Finding for Self Interested Agents". In: *Sixth Annual Symposium on Combinatorial Search*.

Bopp, Julia Ayumi, Livia J Müller, Lena Fanya Aeschbach, Klaus Opwis, and Elisa D Mekler (2019). "Exploring emotional attachment to game characters". In: *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, pp. 313–324.

Bouzy, Bruno (2017). "Playing Hanabi Near-Optimally". In: *Advances in Computer Games*. Springer, pp. 51–62.

Brafman, Ronen I. and Carmel Domshlak (2008). "From One to Many: Planning for Loosely Coupled Multi-Agent Systems." In: *ICAPS*, pp. 28–35. URL: http://www.aaai.org/Papers/ICAPS/2008/ICAPS08-004.pdf (visited on 09/20/2015).

Brockman, Greg *et al.* (2016). "Openai gym". In: *arXiv preprint arXiv:1606.01540*.

Browne, Cameron (2012). "A problem case for UCT". In: *IEEE Transactions on Computational Intelligence and AI in Games* 5.1, pp. 69–74.

Browne, Cameron B *et al.* (2012). "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1, pp. 1–43.

Campbell, Murray, A Joseph Hoane Jr, and Feng-hsiung Hsu (2002). "Deep blue". In: *Artificial intelligence* 134.1-2, pp. 57–83.

Canaan, R. *et al.* (Aug. 2018). "Evolving Agents for the Hanabi 2018 CIG Competition". In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8. DOI: 10.1109/CIG.2018.8490449.

Canaan, Rodrigo (2018). "Games as Co-Creative Cooperative Systems". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 14. 1.

Canaan, Rodrigo, Julian Togelius, Andy Nealen, and Stefan Menzel (2019). "Diverse agents for ad-hoc cooperation in hanabi". In: *2019 IEEE Conference on Games (CoG)*. IEEE, pp. 1–8.

Čertickỳ, Michal and David Churchill (2017). "The current state of StarCraft AI competitions and bots". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 13. 1.

Chacón, Pablo Sauma and Markus Eger (2019). "Pandemic as a challenge for human-AI cooperation". In: *Proceedings of the AIIDE workshop on Experimental AI in Games*.

Chaslot, Guillaume, Mark H. M. Winands, H. Jaap van den Herik, Jos Uiterwijk, and Bruno Bouzy (2008). "Progressive Strategies for Monte-Carlo Tree Search". In: *New Mathematics and Natural Computation* 04, pp. 343–357.

Choi, Dongkyu, Tolga Könik, Negin Nejati, Chunki Park, and Pat Langley (2007). "A Believable Agent for First-Person Shooter Games." In: *AIIDE*, pp. 71–73. URL: http://www.aaai.org/Papers/AIIDE/2007/AIIDE07-013.pdf (visited on 09/20/2015).

Coulom, Rémi (2006). "Efficient selectivity and backup operators in Monte-Carlo tree search". In: *International conference on computers and games*. Springer, pp. 72–83.

Cowling, Peter I, Edward J Powley, and Daniel Whitehouse (2012a). "Information set monte carlo tree search". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2, pp. 120–143.

– (2012b). "Information set monte carlo tree search". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2, pp. 120–143.

Cox, Christopher *et al.* (2015). "How to make the perfect fireworks display: Two strategies for Hanabi". In: *Mathematics Magazine* 88.5, pp. 323–336.

De Weerd, Harmen, Rineke Verbrugge, and Bart Verheij (2013). "How much does it help to know what she knows you know? An agent-based simulation study". In: *Artificial Intelligence* 199, pp. 67–92.

Dockhorn, Alexander and Sanaz Mostaghim (2019). "Introducing the hearthstone-ai competition". In: *arXiv preprint arXiv:1906.04238*.

Donkers, HHLM, Jos W. H. M. Uiterwijk, and H Jaap van den Herik (2001). "Probabilistic opponent-model search". In: *Information Sciences* 135.3-4, pp. 123–149.

Du, Ding-Zhu and Panos M Pardalos (2013). *Minimax and applications*. Vol. 4. Springer Science & Business Media.

Ebner, Marc *et al.* (2013). *Towards a video game description language*.

Eger, Markus and Daniel Gruss (2019). "Wait a second: playing hanabi without giving hints". In: *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pp. 1–7.

Eger, Markus, Chris Martens, and Marcela Alfaro Córdoba (2017). "An intentional AI for hanabi". In: *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, pp. 68–75.

Elias, George Skaff, Richard Garfield, and K Robert Gutschera (2012). *Characteristics of games*. MIT Press.

Farooq, Sehar Shahzad, In-Suk Oh, Man-Jae Kim, and Kyung Joong Kim (2016). "StarCraft AI competition report". In: *AI Magazine* 37.2, pp. 102–107.

Feldmann, Rainer (1997). "Computer chess: Algorithms and Heuristics for a Deep Look into the future". In: *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, pp. 1–18.

Felner, Ariel *et al.* (2017). "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges". In: *Tenth Annual Symposium on Combinatorial Search*.

Finesse, Bluff, Reverse Finesse - explained (n.d.). URL: https://boardgamegeek.com/thread/1309490/finesse-bluff-reverse-finesse-explained.

Foerster, Jakob N *et al.* (2018). "Bayesian action decoder for deep multi-agent reinforcement learning". In: *arXiv preprint arXiv:1811.01458*.

Fogel, David B (2006). "Foundations of evolutionary computation". In: *Modeling and Simulation for Military Applications*. Vol. 6228. International Society for Optics and Photonics, p. 622801.

Font, Jose M and Tobias Mahlmann (2018). "Dota 2 bot competition". In: *IEEE Transactions on Games* 11.3, pp. 285–289.

Gaina, Raluca D, Jialin Liu, Simon M Lucas, and Diego Pérez-Liébana (2017a). "Analysis of vanilla rolling horizon evolution parameters in general video game playing". In: *European Conference on the Applications of Evolutionary Computation*. Springer, pp. 418–434.

Gaina, Raluca D, Simon M Lucas, and Diego Pérez-Liébana (2017b). "Population seeding techniques for rolling horizon evolution in general video game playing". In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1956–1963.

Gaina, Raluca D, Simon M Lucas, and Diego Perez-Liebana (2017c). "Rolling horizon evolution enhancements in general video game playing". In: *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 88–95.

Gaina, Raluca D, Diego Pérez-Liébana, and Simon M Lucas (Sept. 2016). "General Video Game for 2 players: Framework and competition". In: *2016 8th Computer Science and Electronic Engineering (CEEC)*, pp. 186–191. DOI: 10.1109/CEEC.2016.7835911.

Gaina, Raluca D *et al.* (2017d). "The 2016 two-player gvgai competition". In: *IEEE Transactions on Games* 10.2, pp. 209–220.

Genesereth, Michael, Nathaniel Love, and Barney Pell (2005). "General game playing: Overview of the AAAI competition". In: *AI magazine* 26.2, pp. 62–62.

Goldberg, David E and John Henry Holland (1988). "Genetic algorithms and machine learning". In.

Gonzalez-Castro, Jose Manuel and Diego Pérez-Liébana (2017). "Opponent models comparison for 2 players in GVGAI competitions". In: *Computer Science and Electronic Engineering (CEEC), 2017*. IEEE, pp. 151–156.

Goodman, James (2019). "Re-determinizing Information Set Monte Carlo Tree Search in Hanabi". In: *arXiv preprint arXiv:1902.06075*.

Goodman, James and Simon Lucas (2020). "Does it matter how well I know what youre thinking? Opponent Modelling in an RTS game". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1–8.

Gorniak, Peter and Ian Davis (2007). "SquadSmart Hierarchical Planning and Coordinated Plan Execution for Squads of Characters". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 3.1, pp. 14–19. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/18775.

Gottwald, Eva Tallula, Markus Eger, and Chris Martens (2018). "I See What You See: Integrating Eye Tracking into Hanabi Playing Agents." In: *AIIDE Workshops*.

Guckelsberger, Christian, Christoph Salge, and Julian Togelius (2018). "New and surprising ways to be mean". In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 1–8.

Iida, H, Jos WHM Uiterwijk, HJ Van den Herik, and IS Herschberg (1994). "Potential applications of opponent-model search (Part 2: Risks and strategies)". In: *ICCA Journal* 17.1, pp. 10–14.

Iida, Hiroyuki, Makoto Sakuta, and Jeff Rollason (2002). "Computer shogi". In: *Artificial Intelligence* 134.1-2, pp. 121–144.

Iida, Hiroyuki, Jos WHM Uiterwijk, H Jaap van den Herik, and IS Herschberg (1993). "Potential applications of opponent-model search". In: *ICGA Journal* 16.4, pp. 201–208.

Isla, Damian (Jan. 2005). *Managing Complexity in the Halo 2 AI System*.

Jenetics (n.d.). https://jenetics.io/.

Jennings, Nicholas R (2000). "On agent-based software engineering". In: *Artificial intelligence* 117.2, pp. 277–296.

Johansson, Magnus and Harko Verhagen (2014). "Social believability in games the early years". In: *FDG 2014*.

Joppen, Tobias, Miriam Ulrike Moneke, Nils Schröder, Christian Wirth, and Johannes Fürnkranz (2017). "Informed hybrid game tree search for general video game playing". In: *IEEE Transactions on Games* 10.1, pp. 78–90.

Kelly, John Paul, Adi Botea, Sven Koenig, *et al.* (2008). "Offline Planning with Hierarchical Task Networks in Video Games." In: *AIIDE*.

Kiarostami, Mohammad Sina, Mohammad Reza Daneshvaramoli, Saleh Khalaj Monfared, Dara Rahmati, and Saeid Gorgin (2019). "Multi-agent non-overlapping pathfinding with monte-carlo tree search". In: *2019 IEEE Conference on Games (CoG)*. IEEE, pp. 1–4.

Kiarostami, Mohammad Sina *et al.* (2020). "Unlucky Explorer: A Complete non-Overlapping Map Exploration". In: *arXiv preprint arXiv:2005.14156*.

Kim, Man-Je and Kyung-Joong Kim (2017). "Opponent modeling based on action table for MCTS-based fighting game AI". In: *2017 IEEE conference on computational intelligence and games (CIG)*. IEEE, pp. 178–180.

Kleinman, Erica, Sara Chojnacki, and Magy Seif El-Nasr (2021). "The Gangs All Here: How People Used Games to Cope with COVID19 Quarantine". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI '21. Yokohama, Japan: Association for Computing Machinery. ISBN: 9781450380966. DOI: 10.1145/3411764.3445072. URL: https://doi.org/10.1145/3411764.3445072.

Knuth, Donald E and Ronald W Moore (1975). "An analysis of alpha-beta pruning". In: *Artificial intelligence* 6.4, pp. 293–326.

Kocsis, Levente and Csaba Szepesvári (2006). "Bandit Based Monte-Carlo Planning". In: *Machine Learning: ECML 2006*. Springer, pp. 282–293.

Kovács, Dániel László (2012). "A multi-agent extension of PDDL3. 1". In: *International Conference on Automated Planning and Scheduling*.

Kunanusont, Kamolwan, Simon M Lucas, and Diego Pérez-Liébana (2017). "General video game ai: Learning from screen capture". In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 2078–2085.

Lanctot, Marc, Christopher Wittlinger, Mark HM Winands, and Niek GP Den Teuling (2013). "Monte Carlo tree search for simultaneous move games: A case study in the game of Tron". In: *BNAIC*. Vol. 2013. Citeseer, pp. 104–111.

Levine, John *et al.* (2013). "General video game playing". In: *Dagstuhl Follow-Ups*.

Lim, Chong-U, Robin Baumgarten, and Simon Colton (2010). "Evolving behaviour trees for the commercial game DEFCON". In: *European Conference on the Applications of Evolutionary Computation*. Springer, pp. 100–110.

Liu, Jialin, Diego Pérez-Liébana, and Simon M Lucas (2016). "Rolling horizon coevolutionary planning for two-player video games". In: *2016 8th Computer Science and Electronic Engineering (CEEC)*. IEEE, pp. 174–179.

Livingstone, Daniel (Jan. 2006). "Turing's Test and Believable AI in Games". In: *Comput. Entertain.* 4.1. ISSN: 1544-3574. DOI: 10.1145/1111293.1111303. URL: http://doi.acm.org/10.1145/1111293.1111303.

Loyall, A Bryan (1997). "Believable agents: building interactive personalities". PhD thesis. Mitsubishi Electric Research Laboratories.

Lu, Feiyu *et al.* (2013). "Fighting game artificial intelligence competition platform". In: *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*. IEEE, pp. 320–323.

Lucas, Simon M (2007). "Ms pac-man competition". In: *ACM SIGEVOlution* 2.4, pp. 37–38.

Mäkelä, Ville and Albrecht Schmidt (2020). "I Don't Care as Long as It's Good: Player Preferences for Real-Time and Turn-Based Combat Systems in Computer RPGs". In: *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, pp. 231–240.

Mateas, Michael and Andrew Stern (2003). "Integrating plot, character and natural language processing in the interactive drama Façade". In: *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE-03)*. Vol. 2. URL: https://users.soe.ucsc.edu/~michaelm/tenurereview/publications/mateas-tidse2003.pdf (visited on 09/20/2015).

Miller, Brad L and Goldberg (1995). "Genetic algorithms, tournament selection, and the effects of noise". In: *Complex systems* 9.3, pp. 193–212.

Mnih, Volodymyr *et al.* (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*.

Nashed, Youssef S. G. and Darryl N. Davis (2011). "Fuzzy Q-Learning for First Person Shooters". In: URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.386.2416&rep=rep1&type=pdf (visited on 09/27/2015).

Nau, Dana, Yue Cao, Amnon Lotem, and Hector Munoz-Avila (1999). "SHOP: Simple Hierarchical Ordered Planner". In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 968–973. URL: http://dl.acm.org/citation.cfm?id=1624312.1624357 (visited on 12/10/2015).

Neufeld, Xenija, Sanaz Mostaghim, and Diego Pérez-Liébana (2017a). "HTN fighter: Planning in a highly-dynamic game". In: *Computer Science and Electronic Engineering (CEEC), 2017*. IEEE, pp. 189–194.

Neufeld, Xenija, Sanaz Mostaghim, Dario Sancho-Pradel, and Sandy Brand (2017b). "Building a Planner: A Survey of Planning Systems Used in Commercial Video Games". In: *IEEE Transactions on Games*.

Nijssen, Pim and Mark HM Winands (2012). "Monte carlo tree search for the hide-and-seek game scotland yard". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.4, pp. 282–294.

Ontañón, Santiago, Nicolas A Barriga, Cleyton R Silva, Rubens O Moraes, and Levi HS Lelis (2018). "The first microrts artificial intelligence competition". In: *AI Magazine* 39.1, pp. 75–83.

Orkin, Jeff (2006). "Three states and a plan: the AI of FEAR". In: *Game Developers Conference*. Vol. 2006, p. 4.

Osawa, Hirotaka (2015). "Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information". In: *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Panait, Liviu and Sean Luke (2005). "Cooperative multi-agent learning: The state of the art". In: *Autonomous agents and multi-agent systems* 11.3, pp. 387–434.

Perez-Liebana, Diego *et al.* (2019). "General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms". In: *IEEE Transactions on Games* 11.3, pp. 195–214.

Piette, Eric *et al.* (2019). "Ludii–The Ludemic General Game System". In: *arXiv preprint arXiv:1905.05013*.

Ponsen, Marc JV, Geert Gerritsen, and Guillaume Chaslot (2010). "Integrating Opponent Models with Monte-Carlo Tree Search in Poker." In: *Interactive Decision Theory and Game Theory* 82.

Prada, Rui, Phil Lopes, Joao Catarino, Joao Quitério, and Francisco S Melo (2015). "The geometry friends game AI competition". In: *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, pp. 431–438.

Premack, David and Guy Woodruff (1978). "Does the chimpanzee have a theory of mind?" In: *Behavioral and brain sciences* 1.4, pp. 515–526.

Puterman, Martin L (1990). "Markov decision processes". In: *Handbooks in operations research and management science* 2, pp. 331–434.

Pérez-Liébana, Diego, Jens Dieskau, Martin Hunermund, Sanaz Mostaghim, and Simon Lucas (2015). "Open loop search for general video game playing". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, pp. 337–344.

Pérez-Liébana, Diego, Miguel Nicolau, Michael ONeill, and Anthony Brabazon (2011). "Evolving behaviour trees for the mario ai competition using grammatical evolution". In: *European Conference on the Applications of Evolutionary Computation*. Springer, pp. 123–132.

Pérez-Liébana, Diego, Philipp Rohlfshagen, and Simon M Lucas (2012). "The physical travelling salesman problem: WCCI 2012 competition". In: *Evolutionary Computation (CEC), 2012 IEEE Congress on*. IEEE, pp. 1–8.

Pérez-Liébana, Diego, Spyridon Samothrakis, Simon Lucas, and Philipp Rohlfshagen (2013). "Rolling horizon evolution versus tree search for navigation in single-player real-time games". In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, pp. 351–358.

Pérez-Liébana, Diego *et al.* (2016). "The 2014 general video game playing competition". In: *IEEE Transactions on Computational Intelligence and AI in Games* 8.3, pp. 229–243.

Rabin, Steve (2016). "Artificial Intelligence in Games, week 3". In: URL: https://player.slideplayer.com/25/7690957/.

Rabin, Steven (2013). *Game AI Pro: Collected Wisdom of Game AI Professionals*. AK Peters, Ltd. ISBN: 1466565969.

– (2015). *Game AI Pro 2: Collected Wisdom of Game AI Professionals*. AK Peters/CRC Press. ISBN: 1482254794.

– (2017). *Game AI Pro 3: Collected Wisdom of Game AI Professionals*. AK Peters/CRC Press. ISBN: 1498742580.

Rabinowitz, Neil C. *et al.* (2018). *Machine Theory of Mind*. arXiv: 1802.07740 [cs.AI].

Reilly, W. Scott (1996). *Believable Social and Emotional Agents.* Tech. rep. DTIC Document. URL: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA310766 (visited on 09/20/2015).

Ringer, Charles, Cristiana Pacheco, Georgiana Cristina Dobre, and Diego Perez-Liebana (n.d.). "Rolling Horizon Co-evolution in Two-player General Video Game Playing". In: ().

Risi, Sebastian and Mike Preuss (2020). "From chess and atari to starcraft and beyond: How game AI is driving the world of AI". In: *KI-Künstliche Intelligenz* 34.1, pp. 7–17.

Russel, Stuart and Peter Norvig (1995). *Artificial Intelligence: A Modern Approach*. 2nd ed.

Samothrakis, Spyridon, David Robles, and Simon M Lucas (2010). "A UCT agent for Tron: Initial investigations". In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE, pp. 365–371.

Schadd, Maarten PD, Mark HM Winands, Mandy JW Tak, and Jos WHM Uiterwijk (2012). "Single-player Monte-Carlo tree search for SameGame". In: *Knowledge-Based Systems* 34, pp. 3–11.

Schaeffer, Jonathan *et al.* (2007). "Checkers is solved". In: *science* 317.5844, pp. 1518–1522.

Silver, David (2005). "Cooperative Pathfinding". In: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AIIDE'05. Marina del Rey, California: AAAI Press, pp. 117122.

Silver, David *et al.* (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Silver, David *et al.* (2017). "Mastering the game of go without human knowledge". In: *nature* 550.7676, pp. 354–359.

Silver, David *et al.* (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144.

Siu, Ho Chit *et al.* (2021). "Evaluation of Human-AI Teams for Learned and Rule-Based Agents in Hanabi". In: *Advances in Neural Information Processing Systems* 34.

Soemers, Dennis JNJ, Chiara F Sironi, Torsten Schuster, and Mark HM Winands (2016). "Enhancements for real-time monte-carlo tree search in general video game playing". In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 1–8.

Soemers, Dennis JNJ and Mark HM Winands (2016). "Hierarchical task network plan reuse for video games". In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 1–8.

Stern, Roni *et al.* (2019). "Multi-agent pathfinding: Definitions, variants, and benchmarks". In: *arXiv preprint arXiv:1906.08291*.

Sweetser, Penelope, Daniel Johnson, Jane Sweetser, and Janet Wiles (2003). "Creating engaging artificial characters for games". In: *Proceedings of the second international*

conference on Entertainment computing. Carnegie Mellon University, pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=958734 (visited on 09/26/2015).

Togelius, Julian (2014). "How to run a successful game-based AI competition". In: *IEEE Transactions on Computational Intelligence and AI in Games* 8.1, pp. 95–100.

Togelius, Julian, Georgios N. Yannakakis, Sergey Karakovskiy, and Noor Shaker (2012). "Assessing believability". In: *Believable bots*. Springer, pp. 215–230. URL: http://link.springer.com/chapter/10.1007/978-3-642-32323-2_9 (visited on 09/27/2015).

Togelius, Julian, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne (2011). "Search-based procedural content generation: A taxonomy and survey". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3, pp. 172–186.

Tong, Xin, Weiming Liu, and Bin Li (2019). "Enhancing Rolling Horizon Evolution with Policy and Value Networks". In: *2019 IEEE Conference on Games (CoG)*. IEEE, pp. 1–8.

Treanor, Mike *et al.* (2015). "AI-Based Game Design Patterns". In: *In Proceedings of the 10 International Conference on Foundations of Digital Games, FDG 2015. Society for the Advancement of the Science of Digital Games*, pp. 5–6.

Vallati, Mauro *et al.* (2015). "The 2014 international planning competition: Progress and trends". In: *Ai Magazine* 36.3, pp. 90–98.

van den Bergh, Mark JH, Anne Hommelberg, Walter A Kosters, and Flora M Spieksma (2016). "Aspects of the cooperative card game Hanabi". In: *Benelux Conference on Artificial Intelligence*. Springer, pp. 93–105.

Vinyals, Oriol *et al.* (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782, pp. 350–354.

Walton-Rivers, Joseph, Piers R Williams, and Richard Bartle (2019). "The 2018 hanabi competition". In: *2019 IEEE Conference on Games (CoG)*. IEEE, pp. 1–8.

Walton-Rivers, Joseph, Piers R. Williams, Richard Bartle, Diego Pérez-Liébana, and S. M. Lucas (June 2017). "Evaluating and modelling Hanabi-playing agents". In: *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1382–1389. DOI: 10.1109/CEC.2017.7969465.

Warpefelt, Henrik (2016). "The Non-Player Character: Exploring the believability of NPC presentation and behavior". PhD thesis. Department of Computer and Systems Sciences, Stockholm University.

Waveren, J. M. P. van (2001). "The quake III arena bot". PhD thesis. URL: http://192.5.53.208/~brown/242/docs/QuakeIII.pdf (visited on 09/27/2015).

Weerdt, M. D. and Bradley J. Clement (2009). "Introduction to planning in multiagent systems". In: *Multiagent and Grid Systems* 5, pp. 345–355. DOI: 10.3233/MGS-2009-0133.

Weizenbaum, Joseph (1966). "ELIZAa computer program for the study of natural language communication between man and machine". In: *Communications of the ACM* 9.1, pp. 36–45.

Whitehouse, Daniel, Edward J Powley, and Peter I Cowling (2011). "Determinization and information set Monte Carlo tree search for the card game Dou Di Zhu". In: *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. IEEE, pp. 87–94.

Williams, Piers R, Diego Perez-Liebana, and Simon M Lucas (2016a). "Ms. Pac-Man versus ghost team CIG 2016 competition". In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 1–8.

Williams, Piers R., Diego Pérez-Liébana, and S. M. Lucas (Sept. 2016b). "Ms. Pac-Man Versus Ghost Team CIG 2016 competition". In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8. DOI: 10.1109/CIG.2016.7860446.

Williams, Piers R., Joseph Walton-Rivers, Diego Perez-Liebana, and Simon M. Lucas (Sept. 2015). "Monte Carlo Tree Search Applied to Co-operative Problems". In: *CEEC2015 - IEEE Conference on Computer Science and Electronic Engineering*. IEEE CEEC. IEEE Computer Society, pp. 219–224.

Yang, Xulei *et al.* (2018). "Deep learning for practical image recognition: Case study on kaggle competitions". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 923–931.

Yannakakis, Geogios N. (2012). "Game AI revisited". In: *Proceedings of the 9th conference on Computing Frontiers*. ACM, pp. 285–292. URL: http://dl.acm.org/citation.cfm?id=2212954 (visited on 09/30/2015).

Yannakakis, Georgios N and Julian Togelius (2018). *Artificial Intelligence and Games*. Springer.

Zhong, Jinghui, Xiaomin Hu, Jun Zhang, and Min Gu (2005). "Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms". In: *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Tech-*

nologies and Internet Commerce (CIMCA-IAWTIC'06). Vol. 2, pp. 1115–1121. DOI: 10.1109/CIMCA.2005.1631619.

Appendix A

Hanabi agent rules

This appendix lists the rule-set available in the *Hanabi* implementation. These are the production rules that are provided with the engine and which can be used to create agents for the game. They are inspired by both existing literature, discussions on the boardgamegeek forums*Finesse, Bluff, Reverse Finesse - explained* n.d. and derived from the strategies observed being used by human players.

A.1 Discard Rules

Discard rules cause the player to discard one of the cards in their hand. The rules of *Hanabi* don't allow discards if the team already has the maximum number of information tokens (eight) and so these rules will never fire if this is the case.

Discard rules are used to generate information tokens by removing one of the cards in the player's hand from play and drawing a new one.

Discard Randomly

Choose a card at random from the player's hand and discard it. This is the simplest but also one of the riskiest of the discard rules in the framework as it can throw away cards which are needed for the game.

Discard If Certain

This rule gets the known colour and value of the card and checks if the current card in that suit on the table is higher or equal to that value. This means the rule will throw away any card which has been completely identified by another player and is not needed to complete the game.

This rule is deliberately very basic and does not make inferences based on the game rules or the current state. It will not discard cards which have become unplayable due to errors the team has made. For example, if both red threes have been discarded the red four is no longer playable, but this rule would not discard the red four as the highest card on the table could is a two or less.

If multiple cards match this rule, the rule will choose the card with the lowest position in the hand.

Discard Safe Card

This discard rule expands on Discard If Certain by allowing the discarding of cards for which the player only has partial information using inference.

This rule considers a card discardable if any of the following are true:

1. The suit is known, and the five of that suit has already been played.

2. The suit and value are known and the table is already higher.

3. The value is known and the value is lower than all top cards on the table.

4. The suit and value are known but all copies of a lower value card in this suit have already been discarded.

These rules act as a good benchmark for a discard rule used by a relatively intelligent player.

Discard Oldest First

When cards are placed into the player's hand by the game, the player automatically tracks the order in which they are inserted. This allows players to know the relative 'age' of cards in their hand.

This rule uses that information to find the oldest card in the player's hand which it then discards. The rule allows players to avoid spending information tokens to identify discardable cards but does potentially mean that useful cards can be thrown away.

This rule was derived from observed human play from colleagues.

Discard Oldest No Info First

This follows the same basic strategy as Discard Oldest First but rather than discard the oldest card blindly, the rule will discard the oldest card for which the suit or value are not known.

This was again derived from observing human play, in which players stopped their team-mates following the above rule by pointing out information about cards. This

rule gives the ability for players to 'veto' a card about to be discarded by executing a tell action to save it.

There is a possibility for useless cards to have some information co-coincidently pointed out during the game, however, which may limit this rule's effectiveness for AI players.

Discard Useless Card

This rule acts as a 'damage control' rule. It is a more complex version of the third criterion from Discard Safe Card. It deals with the situation where the players have already made a mistake which makes a perfect score unobtainable and attempts to use that to its advantage by identifying and eliminating cards which are no longer useful from its hand.

Firstly, the rule searches though the discard pile for cards which match the suit of the card being examined. By comparing the amount of each rank discarded to the amount which are in the game, the rule can calculate the highest obtainable score for this suit.

If the value is known, and is higher than the maximum obtainable value, then the card is discarded. If, however, the value is not known, the rule calculates the possible values which the card could be, using the information available to it. It does this by looking though the cards whose locations are not currently known, as these will either be in the player's hand or the deck. From this information, the rule can calculate the minimum value this card can be. If this is higher than the lowest value obtainable then the card is discarded.

Osawa Discard

This rule is simply the concatenation of Discard Safe Card and Discard Useless Card as these constitute the equivalent the discard rule discussed in (Osawa, 2015).

Discard Least Likely to be Necessary

A necessary card is a card which is now unique within the game and is still needed to complete a suit. This rule calculates the likelihood of a given card being necessary. It does this by finding all cards which could occupy the slot that are still in the game and then calculates $\frac{n}{N}$ where n is the number of necessary cards and N is the total number of possible cards.

This value is calculated for all slots in the player's hand and the smallest score is used as the recommendation. If all cards in the player's hand are known to be indispensable, then the rule will not fire.

Discard Probably useless card

This rule detects when cards are unlikely to be playable, with some threahold passed into the rule. If multiple slots match the rule, the most likely slot is discarded.

Discard Highest

Discard the card with the highest known value. This rule works on the rationale that these cards are less important than the cards which have lower values, as in order to play the higher-value card the lower-value one of the same suit must for it to be played.

Discard Unidentified Card

This rule discards the first slot it finds which has no information. This is similar to 'Discard Oldest No Info First', but rather than discarding the oldest card, it discards the card with the lowest slot ID.

A.2 Tell Rules

Tell rules communicate information to other players. Executing a tell rule requires that the team has at least one information token available.

Tell Playable Card

Tell playable card indicates to another player a subset of their cards which are playable.

Tell Playable Card Outer

This rule is derived from (Osawa, 2015). The rule will point out any card which is next in sequence to the next player. The rule makes use of information the player already has and therefore will avoid telling things the player already knows.

Tell Randomly

This rule points out a random legal card attribute in a players hand.

Tell About Ones

This rule fires if the next player has at least one card of rank one which they do not know about.

Tell Anyone About Useful Card

This rule searches though all other player's hands until it finds a card which is needed to complete the game.

Tell Unknown

This rule tells a player about cards they do not currently have information about.

Tell Anyone about Useless Card

This rule tells player's about cards in their hand which are no longer needed and can be discarded.

Tell Dispensable

This rule calculates if a card in the players hand is no longer needed and identifies it to allow it to be discarded.

Tell Most Information

This rule is derived from (van den Bergh *et al.*, 2016). It will execute the tell rule for a given player, prioritising the rule which gives the player the most information. For example, if the player had three red cards and two blue cards, the rule would tell the player about their red cards.

This rule has two variants: the first only tells new information (it ignores the information the player already knows); the other uses total information.

Tell Ill Informed

This rule tells a player about a useful card in their hand if they currently do not know about it.

Tell Fives

This rule simply points out all fives that are presently in a players hand if they don't know they are already fives. This rule will consider all other players, starting with the player who has their turn after this player and moving round the board in turn order.

Tell Anyone about Oldest Useful Card

This rule is designed to work with Discard Oldest No Info First. It will tell players about the oldest card in their hand which is immediately useful (the next card in sequence). This rule will consider all others players, starting with the player who has their turn after this player and moving round the board in turn order.

Tell To Save

This rule is designed to work with Discard Oldest First. It will tell a player about a card which is still needed, to avoid the player's removing it. It will only consider unique cards left in play.

Tell To Save Partial Only

This rule behaves the same as Tell To Save, but will ignore any card which the player already has partial information about.

Complete Tell Useful Card

This card completes a tell action for a useful card: if the player only has partial information about the card then this rule will let them know the missing piece.

A.3 Play Rules

It is always legal to execute a play rule. They are one of the most risky moves in the game as making three incorrect play moves will lose the team the game.

Play If Certain

This is a very simplistic play rule. If both the rank and suit of the card are known and the card is next in sequence for its suit, the card is played.

Play Safe Card

This rule iterates though the slots in the player's hand and calculates the possible cards that can be present in that slot based on the information available to the agent. If all possible cards are playable (e.g. we have been told that the card is a 1 and no 1s have been played) then choose to play this slot from our hand.

Play Unique Possible Card

This rule is unusual as it makes use of the game history. The agent searches though the history of the game (from its perspective) until it finds a tell action directed at it that points out only one card. If that card is still in its hand it blindly plays it.

This is based of a human strategy in which a player points out a single card to another player indicating that the player should play the card.

A.4 Finesse

The 'finesse' rules are a set of three rules designed to be used together. This represents an advanced human tactic which has been mentioned by other players.

Tell Finesse

This is the first stage in a finesse, and the most complex. Firstly, the rule checks if the current game contains at least 3 players, as it is not possible to finesse in a 2-player game. Next, it calculates the newest card in the next player's hand and checks if it is playable.

If it is, the rule initiates a search though the third player's hand (the player after the next turn). This attempts to find a card in the player's hand which meets the requirements. The rule then calculates what tell rules (if any) would be unique. If both a tell value and colour would be unique then it prioritises unknown information, else it will use the unique attribute. If neither attribute is unique, the finesse rule fails.

Play Finesse

This is the mid state of a finesse. If the player immediately before us tells the player immediately after us about a card which was one away from being playable (for example, the red 1 has been played and they were told about a red 3) then detect this as a finesse move. If a finesse move is detected, this rule will blindly play the last card added to their hand.

Play Finesse Told

This is the last stage in a finesse rule. If the previous player played a card and the move before this the player was told about a single card, then blindly play the card.

A.5 Other

There is one rule which does not fit into one of the basic categories mentioned above. This rule can perform more than one type of game action depending on the situation.

Try to Unblock

Firstly, this rules checks whether there is at least one information token available. If there is, the next player cannot be blocked and therefore the rule will not fire.

If there is no information token available, it checks if it the current player knows about a card they can discard or if they have a playable 5. To generate information, the rule will then either discard a card or play the five.