UNIVERSITY OF ESSEX

DOCTORAL THESIS

Artificial intelligence in co-operative games with partial observability

---

Author:
Piers R. Williams

Supervisors:
Dr Michael Fairbank,
Dr Simon M. Lucas

A thesis submitted for the degree of
Doctor of Philosophy
in the
School of Computer Science and Electronic Engineering
6th February 2019

UNIVERSITY OF ESSEX

Abstract

Faculty of Science and Health

School of Computer Science and Electronic Engineering

Doctor of Philosophy

Artificial intelligence in co-operative games with partial observability

by Piers R. Williams

This thesis investigates Artificial Intelligence in co-operative games that feature Partial Observability. Most video games feature a combination of both co-operation, as well as Partial Observability. Co-operative games are games that feature a team of at least two agents, that must achieve a shared goal of some kind. Partial Observability is the restriction of how much of an environment that an agent can observe.

The research performed in this thesis examines the challenge of creating Artificial Intelligence for co-operative games that feature Partial Observability. The main contributions are that Monte-Carlo Tree Search outperforms Genetic Algorithm based agents in solving co-operative problems without communication, the creation of a co-operative Partial Observability competition promoting Artificial Intelligence research as well as an investigation of the effect of varying Partial Observability to Artificial Intelligence, and finally the creation of a high performing Monte-Carlo Tree Search agent for the game *Hanabi* that uses agent modelling to rationalise about other players.

Acknowledgements

This is my opportunity to say thank you to the endless people behind the scenes supporting me through my research as well as more directly assisting and collaborating on it.

I of course need to thank my supervisors: Dr Michael Fairbank, Dr Simon Lucas, and Dr Diego Perez-Liebana. Without their guidance, time, and assistance none of this work would have been possible. Special mention to my frequent collaborator and colleague Joseph, for the daily discussions on various topics and strategies that have made this possible. I would also like to thank both the CSEE department at the University of Essex, and the IGGI CDT for their support during these past years.

On a more personal note, I would like to thank my fiancé Hannah, who has supported me throughout. When I went to start my three year undergraduate degree, extended it to a fourth year, and then bolted on a four year PhD she has been there for me. I would also like to thank both my own family and Hannah's family for all the help getting away from my research to relax.

Contents

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Introduction

The field of research into Artificial Intelligence (AI), and in particular AI in games has been growing into an important field with conferences like CIG[1] and journals like ToG[2] leading the charge. Games provide interesting benchmarks to develop AI algorithms but are increasingly an important consumer of AI with the games industry beginning to recognise the need for more intelligent Non-Player Characters (NPCs) in their virtual environments.

There are a wide variety of different types of games, each providing its own unique challenge to AI. Not all games provide full access to the environment, creating interest and difficulty by hiding particular pieces of information from the player or players. Other types of game expect teamwork from the players rather then being solely adversarial. Some games use both restrictions, and it is this type of game that this thesis concentrates on.

1.2 Organisation

This thesis is divided into two main parts. Part I introduces the reader to the main ideas behind the thesis (Chapter 1) as well as introducing the background research (Chapter 2) and explanations of various things discussed in the thesis (Chapters 3 and 4).

[1] IEEE conference on Computational Intelligence in Games
[2] IEEE Transactions on Games

Part II covers the main work done towards this thesis. Chapter 5 discusses the uses of General Game Playing (GGP) AI in a simple co-operative environment. Chapter 6 describes the re-introduction of the Ms. Pac-Man Vs Ghost Team Competition as a Partial Observability (PO) co-operative AI competition to stimulate further research in the community. Chapter 7 describes extensive evaluation of modern research as well as new techniques in the card game *Hanabi* which is a PO co-operative game gaining research interest lately. Chapter 8 details experiments performed with AI agents to investigate the effect of varying PO on difficulty in *Ms. Pac–Man* and then investigates two possible games on human participants to validate the AI results.

1.3 List of Papers

The following list of published or accepted papers contributed towards this thesis. The contributions they made are explained in **bold** font.

1. Piers R. Williams, Joseph Walton-Rivers, Diego Perez-Liebana and Simon M. Lucas (2015). 'Monte Carlo Tree Search Applied to Co-operative Problems'. In: *CEEC'2015 - IEEE Conference on Computer Science and Electronic Engineering*. IEEE CEEC. IEEE Computer Society, pp. 219–224 **Comparison of Algorithms and experimental co-operative domain. Additional authors helped with creation of algorithms, experiment running and paper writing.**

2. Piers R Williams, Diego Perez-Liebana and Simon M Lucas (2016). 'Ms. Pac-Man Versus Ghost Team CIG 2016 Competition'. In: *CIG'2016 - IEEE Conference on Computational Intelligence and Games*. IEEE CIG, pp. 420–427 **Introduction and description of competition including new features to the engine. Introduction of some basic AI and a comparison of performance between PO and Complete Observability (CO) techniques.**

3. Joseph Walton-Rivers, Piers R Williams, Richard Bartle, Diego Perez-Liebana and Simon M Lucas (2017). 'Evaluating and Modelling Hanabi-Playing Agents'. In: *Congress on Evolutionary Computation, 2017. CEC'17. IEEE Conference On*. IEEE, pp. 1382–1389 **Introduction to Hanabi, including description of open source game engine and evaluation of previous research into the game. Introduction of our own agents, including an Information-Set Monte-Carlo Tree Search (IS-MCTS) agent that used models of the other agents to predict what they would do. My main contributions were to the general development of the framework and agents as well as contributing to paper writing.**

Chapter 2

Background

This chapter provides a thorough background introduction to previous literature in several areas. Section 2.1 introduces the reader to the concept of Partial Observability (PO), Section 2.2 gives an overview of what it really means to be co-operative, while Section 2.3 discusses the effects of communication. Finally Section 2.4 details a number of Artificial Intelligence (AI) competitions that feature PO or co-operation in them.

2.1 Partial Observability

PO is the impairment of the ability of an agents to completely observe the world that it is situated within (Bertoli *et al.*, 2001). There are a great many methods through which agents can observe the world. Games typically focus on sight limitations in different ways, as discussed in Table 2.1

The focus on sight within games is largely due to the simplicity with which it can be implemented and the realism that it provides. Sight is also often the primary sense used by gamers while playing. When there is either too many restrictions to observability or too few restrictions to observability can spoil a game. Immersion would be broken if there was too few observability restrictions, such as if a player could see clearly across an entire map in a First-Person Shooter (FPS). *Poker* would be significantly less interesting with too

TABLE 2.1: Examples of PO in games.

First-Person Shooter	The field of view presented to the player naturally restricts sight. Often a shorter range "minimap" is provided with restrictions that can vary from game to game.
Real-Time Strategy	The game often calculates what all of the player's units can see, and then obscures the rest.
Platform Games	The game often only presents a viewport with the character within it, and progression through the game moves the viewport.
Role-Playing Game	The field of view is often restricted to whatever the character could potentially see.
Horror	The view is often dark and poorly lit. Enemies will purposefully hide from the player until ready and combined with ominous sound effects more can be said with the unseen than the seen.

many restrictions to observability, such as if the player could not see their own cards either, reducing it to a game of chance.

PO can apply to more than just the visual sense that players use to observe the game. Players are typically capable of hearing things within the game, and on some controllers receive limited tactile feedback as well. All these things are observations of the game and are used to enhance the experience for the player. There is also a distinct difference between PO for human players, and PO for AI agents. AI agents do not have the same senses that humans do, and often interact with the game through a defined Application Programming Interface (API) instead. Limiting the information available through the API is PO for AI agents. AI agents may have identical or different PO restrictions placed upon them by the game in comparison to human players.

If games are created that make better use of partial observability and co-operation through communication as mechanics, then there will be a higher demand for AI that can handle this shift. Many games are still sold with single player modes which will require realistic companion Non-Player Character (NPC) behaviour as well as opposition AI.

Games can be placed on a theoretical scale from No Observability to Full Observability. For example, traditional *Ms. Pac–Man* is a fully observable game, while the card game *Poker* is much nearer the no observability end of the spectrum. Figure 2.1 shows a number

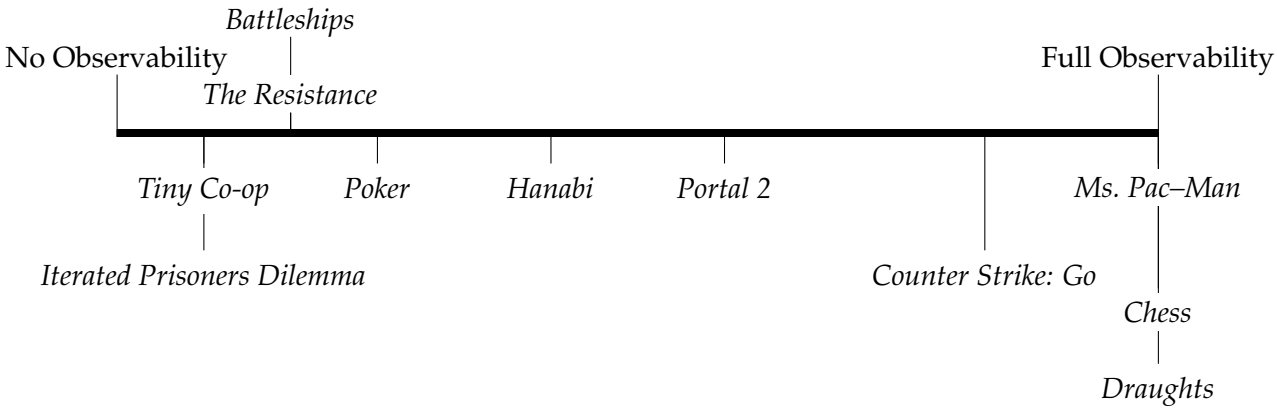


FIGURE 2.1: Games placed on a scale from no observability to fully observable.

of possible games on such a scale. The values used for these scales are arbitrary and only meant to give a rough idea.

If games can be placed on this scale, it stands to reason that the many possible variations of a single game can also be placed on this scale. The competition for Ms. Pac–Man Vs Ghost Team has a number of parameters and settings that alter the amount of information visible to an agent ranging from full observability to nearly no observability. These are described in more detail in Section 4.2. Figure 2.2 shows the various possible values for sight limitations on a scale. Line-of-Sight (LOS) refers to a restriction where sight is prevented by obstacles in the maze. Only the four cardinal directions up to either a distance limit or the next wall is visible. Forward Facing Line-of-Sight (FF-LOS) refers to a LOS restriction that only allows forward LOS. Radius refers to a restriction where anything within the distance limit is visible, causing a circle of observability centred on the player.

2.1.1 Symmetric or Asymmetric Partial Observability

Symmetric PO would be the situation where all players in the game have the same abilities to make observations as each other. It is not a requirement for this to be the case and

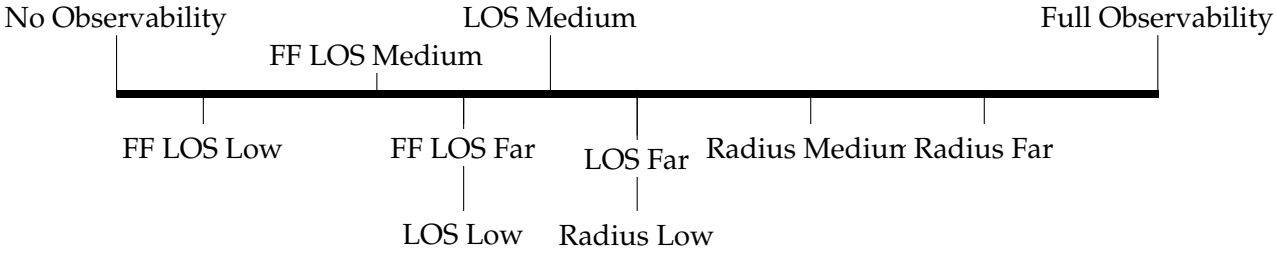


FIGURE 2.2: Combinations of PO mode and sight limit placed on a scale from no observability to fully observable.

many games alter that either permanently or temporarily. Some FPS games allow players to choose various perks or upgrades that can alter visibility. Being able to ”sneak" would remove certain players from the minimap whilst other perks could allow temporary x-ray vision to add observations.

Careful consideration of how much of an advantage PO gives one side compared to another is important, for example in *Ms. Pac–Man*, the ghosts outnumber Ms. Pac–Man four to one. The balance of PO in *Ms. Pac–Man* is investigated in more depth in Chapter 8.

2.2 Co-operation

Co-operation is an area of particular interest to a great many researchers. Competitions in the area are very popular, and some have even reached recognition with the general populace such as the Sony Robot Dog football. This section will describe what it means to be co-operative in a general sense as well as computer games and competitions that have sought to improve co-operative AI in general.

2.2.1 What does it mean to be Co-operative?

Co-operation, as defined in the Oxford English Dictionary at time of writing, is: “*The action of co-operating, i.e. of working together towards the same end, purpose, or effect; joint operation*”.

This definition includes both passive co-operation and active co-operation.

Passive Co-operation

Passive co-operation is where agents strive towards the same goal, performing actions with no respect to each other. These agents are simple to write and feature mostly in computer games.

Games such as Real-Time Strategy (RTS) typically feature top level commander agents that can be placed in alliances, though that only influences the list of legitimate targets.

FPS games typically feature a multitude of agents that are in teams opposing or fighting for the player. Those fighting for the player typically stay close to the player but essentially just shoot at the enemy players with no regard for the player's goal. Enemy agents just typically either hunt the enemy or guard a specific location, shooting at anything that opposes them. This is passive, due to the lack of communication in their behaviour. Not all FPS games do this, for example Battlefield 2142 used squads that took commands from the player.

Active Co-operation

Active co-operation is where agents communicate and use joint strategies in order to achieve a goal or set of goals. Active co-operation can be seen in more advanced computer games AI such as Empire Earth II (EEII) [1].

EEII was a RTS game that featured the ability for agents to construct battle plans on an abstracted form of the game map. These plans allowed for co-ordinated manoeuvres and importantly, the AI agents could understand these plans and act on them.

Portal 2 [2] was a computer game released in 2011 that featured simple laboratory experiments that the player had to complete as a technological lab rat. The multiplayer mode was a major introduction in this game, featuring puzzles that required two test subjects

[1] Developer: Mad Doc Software, 2005
[2] Valve, 2011

to solve. Co-operation was a requirement, the puzzles couldn't be solved alone. Communication was often needed, as timed switches forced the players to act in a co-ordinated fashion.

Active Co-operation typically features in on-line modes, where all agents are controlled by human players. These players were originally provided with textual chat within the game, though that is often now joined with voice chat to enhance the experience. This allows co-ordinated strategies, and can greatly improve the quality of play.

The Need for Co-operation

When is co-operation necessary? Co-operation is necessary in a situation where there are a number of restrictions or constraints that don't allow a single agent to perform the necessary tasks. Some situations require co-operation from the rules. Nuclear missile controls are often popularised to require two people with a key each that must be turned at the same time. This is where a synchronised action is required. Other, less sinister situations require synchronicity of action. A nut and bolt requires both the nut and bolt to be held tight, with one or both of them turned. Failure to secure either end will result in the both parts rotating and the nut not tightening on the bolt.

Games sometimes force players to co-operate, with Portal 2's multiplayer mode requiring synchronous activities frequently. One player would need to push a button, and elsewhere in the game another player would have to react to the button push. One example is where the button drops a cube that the second player has to jump off of a powered catapult at the right time to be launched over a void and catch the cube. This is only possible if the second player enters the catapult during a small window of time after the first player activates the cube.

2.2.2 When to Co-operate

Co-operation can be considered to occur prior to an event or during the event. One major example of prior co-operation is situations where multiple agents communicate and agree strategy prior to commencing a task. This communication isn't entirely necessary, as often software agents will be using identical algorithms which is a form of prior co-operation. Co-operation without communication can occur when agents have agreed a strategy between them. This allows them to react to each other's actions correctly without explicitly communicating anything between each other. Using the same algorithm can function as a form of agreed strategy, such as the exponential backoff used to prevent continued multiple transmissions of data on a network. Co-operation can also occur during an event which is likely to be more flexible than prior co-operation although arguably more complex to achieve. Co-operating during the completion of tasks can also have other negatives such as taking valuable time needed to complete the task.

2.2.3 Co-operative Games

There are a great number of games that feature co-operation with the next few sections covering some of these.

Physical Games and Sports

A huge number of physical sports games feature active co-operation, albeit it is often emergent through communication as opposed to being defined in the rules of the game itself. A full list is significantly too large to go into detail here, but some basic well known examples follow.

Footballers often shout basic chat to each other to inform intent or request help, and all members of the team share the same goal of maximising score difference.

TABLE 2.2: Table of Player types in Pandemic

Dispatcher	Has ability to move another player without using an action or can move any player a large distance as an action.
Medic	Can treat all cubes in a city in a single action, or take no action if the cure has been discovered.
Scientist	Needs less cards of the same colour to discover a cure.
Researcher	Can exchange cards with a player when in same city as them.
Operations Expert	Can store a discarded card as a back up plan for use later.

Badminton is a sport that can be played as "doubles" and often the two players will have strategies about which areas of the court they occupy or target, and will communicate to stop both or neither player going for the shuttlecock.

Board and Card Games

Board and Card games are known for having great social interactions even with completely competitive games. Some of these games are co-operative however.

Hanabi *Hanabi* is a game with custom cards that features a set of cards depicting fireworks. There are five possible colours and numbers from 1-5 for each card. The aim of the game is to play the cards in order for all 5 colours. The total score is the sum of the maximum cards played for each colour. Players are able to see each others cards, but not their own cards. The game of *Hanabi* and existing literature are described in more depth in Section 4.3 and Chapter 7 respectively

Pandemic *Pandemic* is a board game for 2-4 players based around the premise of 4 diseases breaking out across the world. The player's goal is to stop as many of the diseases as they can. Each player has a different role that is randomly selected as shown in Table 2.2. *Pandemic* differs from *Hanabi* as the player's differing capabilities mean they have to co-operate in certain ways based on their role. This is rather different to *Hanabi's* co-operation model where all players have the same capabilities as each other.

TABLE 2.3: Table of some puzzle elements from *Portal 2*

Name	Description
Door	A door that blocks passage. Can be triggered to open by a variety of items connected to it.
Heavy Duty Super Button	A button that requires something heavy to activate whatever is connected to it
Weighted Companion Cube	Something heavy that stays put when left unattended

Video Games

Video games often contain co-operation as mentioned above though it is rarely forced upon the players.

Portal 2 *Portal 2* built on the success of the original *Portal* by adding a new campaign and crucially a two player experience. It is this two player mode that is particularly interesting. The simple premise of *Portal* games are that the player is stuck in test mazes designed to test increasing numbers of puzzle elements. The puzzle elements vary in operation and a few key items are described in Table 2.3.

2.3 Communication

Communication is important when considering the world as a PO environment and more than one co-operative agent is present.

2.3.1 Communication in Completely Observable Environments

Communication can be useful, or even necessary in a Complete Observability (CO) environment. Consider two agents: A and B. A and B can see the entire environment that they are in. They cannot, however, observe each other's internal state. This information can, however, be voluntarily communicated between them. Suppose that the agents are not homogeneous in capabilities, but have different programming allowing mastery of different

tasks. This information could be vital in organising which agent performs which actions in the environment.

Internal state can be predicted from external observations, especially if you know how the other agent will behave in a given situation, or have a reasonably accurate model of the other agent.

People driving can predict what other drivers will due because we share the same strategy for driving. For example, two drivers approaching a mini-roundabout know which driver has to give way and which driver is allowed to continue un-interrupted.

In contradiction, two humans approaching each other in the street have no pre-agreed strategy for who gives way, often leading to the irritating circumstances shown in Table 2.4. In this case, we consider four possible actions for each human: turning left, turning right, stopping, and continuing straight. Humans rarely stop, it impedes progress, so typically will alter course either to the left or to the right. Optimally each human needs to choose the same direction in order to avoid each other. There is no pre-agreed direction to choose though, leaving the risk of collision at 50% with just these two options. A single human continuing, with the other human avoiding gives slightly better odds, but again no pre-agreed rules exist for this.

This problem is often solved by a third party, such as in busy pedestrian areas. The London underground is a good example of this. On most escalators there are signs indicating that people wishing to stand still are to stand on the right, allowing a space on the left for people that wish to walk up or down the escalator. Step sections in busy stations will also often have a railing across the middle and indicators for which side is up, and which side is down to reduce cross-flow.

Problems cannot always be solved with pre-arranged rules however. Sometimes agents will find themselves in situations with unknown co-operative allies and a task to solve. In these cases, it can be more advantageous to communicate with each other.

TABLE 2.4: Table showing possible interactions between two people on a collision course. When both people stop, the situation is simply paused until at least one person does something else.

First \ Second	Avoids Left	Avoids Right	Stops	Continues
Avoids Left	Avoid	Collide	Avoid	Avoid
Avoids Right	Collide	Avoid	Avoid	Avoid
Stops	Avoid	Avoid	Paused	Collide
Continues	Avoid	Avoid	Collide	Collide

2.3.2 Communication in Partially Observable Environments

Communication allows for the sharing of knowledge between two agents, increasing the view of a PO environment and reducing the restriction on them. This makes it possible for an agent to be aware of:

- Their own internal state
- The world around them
- Other agent's internal state with a communication delay
- The world around other agents with a communication delay

These agents can also obtain the same benefits that CO agents can from communication, although it may be less effective within a PO environment.

The primary benefit in a PO environment is the sharing of the world around each agent to every other agent. Sharing this information greatly increases the total information available to each agent, increasing their ability to reason about the world accurately.

2.4 Artificial Intelligence Competitions

Competitions provide a scoring technique against often current state of the art controllers for a problem domain within a strict set of constraints and often equal computational budget.

Competitions are often good catalysts for an area of research, sometimes even with nothing more than credit on offer for winning. A single large competition such as the Google AI Challenge in 2011 had over 100 competitors and 7900 submissions. This is a huge amount of people writing some high quality and well performing bots in order to take part. The wealth of information available, with many bots having articles and tutorials on their functions being written. 116 topics were made in the official forum for strategy alone in 2011.

The next sections describe individual relevant competitions that have been, or are being organised.

2.4.1 Robot Soccer World Cup

Robot Soccer World Cup, often known as RoboCup, is an annual competition (Kitano *et al.*, 1997) that aims to promote AI research through a simple mandate - to produce a team of robot players that can compete under FIFA[3] rules to beat the winner of the most recent World Cup.

The competition has grown a lot since conception, with multiple leagues now recognising both robotic and simulated modes of play. The robots have also evolved to include bipedal movement as well as quadrupedal.

2.4.2 Robot Rescue Cup

RoboCup also has a separate section for search and rescue challenges, aiming to promote the development of robots that can assist in disaster zones. In the 2001 article (Kitano and Tadokoro, 2001), the authors set out their vision for the rescue cup as a grand challenge for Multi-Agent System (MAS). The scope of the problem is vast, with difficulties in a great many areas including the poor condition of the environment that the robots are supposed to operate in. A proposition for a simulator being developed in order for the AI agents to operate in is also included. This challenge is heavily focused on massive teams of agents

[3]Fédération Internationale de Football Association

using active co-operation to solve goals. The environment is also vastly varied and unpredictable - proving a challenge to directly code logic for.

2.4.3 Ms. Pac-Man Vs Ghost Team

The Ms. Pac–Man Vs Ghost Team competition originally ran from 2007 till 2011, based around the popular 1980's arcade game *Ms. Pac–Man* (Rohlfshagen and Lucas, 2011). The competition allowed entrants to submit controllers for Ms. Pac–Man and in 2011 added the ability to submit controllers for the ghost team. The game engine mimicked the original as closely as it could, and was a CO environment.

The competition featured a number of sample controllers including one that emulated the original game's ghost team, which was written using a separate set of rules per ghost agent. This didn't involve communication, but would allow communication easily between the individually controlled agents.

The competition was revived in 2016 running at CIG[4] (Williams *et al.*, 2016) and repeating in 2017 at CIG. The competition added PO to the environment, and created a MAS structure for the ghosts as well as controlled communication. This is described in more detail in Section 4.2. The competition has gained good interest from the community, with over 40 entrants in 2017. The competition will run at CIG in 2018 for its third iteration.

2.4.4 Geometry Friends

Geometry Friends is a physics based 2D world whereby a circle and a rectangle are required to solve puzzles. The circle and the rectangle are each capable of different actions, providing heterogeneous game play between them. Geometry Friends has run from 2013 - 2017.

[4]IEEE conference on Computational Intelligence in Games

The circle was given the ability to roll left, roll right or jump. Due to the physics-based world, rolling was implemented through applications of torque and wouldn't necessarily cause a movement in the intended direction due to gravity or loss of contact with a ground.

The rectangle was given the ability to slide left, slide right or morph. Morphing maintained area, but exchanged height for width or vice versa.

The primary track for the competition was to provide two co-operating agents, one for each of the circle and square. Two additional tracks involved submitting a controller for just one of either the circle or rectangle. The results from the 2014 competition paper (Prada *et al.*, 2015) concludes that the agents submissions that were received left lots of scope for improvement, implying that the game had not been sufficiently well solved.

2.4.5 Multi-Agent Programming Competition

This competition is a competition operated annually by the AI group at Clausthal University. The problem domain is changed each year, with a number of publications from the team and entrants available. The competition has run from 2005 - 2017 and had a number of different scenarios, some of which ran for multiple years with minor tweaks (Behrens *et al.*, 2012; Ahlbrecht *et al.*, 2013; Ahlbrecht *et al.*, 2018). The scenarios used are listed in Table 2.5. The competition is currently running for the year 2018 at time of writing.

All of the scenarios focused on teams of agents with the 2011 competition including varying types of agent that needed controlling. This heterogeneous population is a particular challenge to MAS but one that is very appropriate in many types of computer games.

2.4.6 Trading Agent Competition

This competition is a MAS whereby each contestant submits an agent that takes the place of a travel agent (Wellman *et al.*, 2001). Each travel agent is given a list of 8 clients who all have individual preferences for hotels, locations and entertainment. The agent is then required

TABLE 2.5: Scenarios from the Multi-Agent Programming Competition

Food Collectors	(2005)	A simple scenario with a grid world containing either food or an agent. Each agent had partial observability of the world. Food appeared randomly, and would require searching for. Food would be returned to a depot by the agent.
Gold Miners	(2006 - 2007)	Similar to Food Collectors, but trees were added as obstacles and to form mazes, as well as the addition of the opposing team to the scenario. In 2007 agents were allowed to carry multiple pieces of gold.
Cowboys	(2008 - 2010)	A new scenario involving teams of agents herding cows into corrals. The cows were NPCs that were programmed with flocking behaviours and were scared of cowboys. From 2009, gates were added to increase the challenge.
Agents on Mars	(2011 - 2014)	This challenge consisted of agents trying to cooperate in order to occupy zones on the planet Mars.
Agents in the City	(2016 - 2018)	This challenge involves agents earning money across a realistic city. Each team has to earn as much money as possible by completing jobs that involve the acquisition, assembly, and transportation of goods. Agents are specialised to different tasks, as well as having different battery capabilities

to bid on tickets for all these things to match as best it can the client's requirements. Each agent is then scored on how well it constructs journeys for each client.

2.4.7 Google AI Challenge

The Google AI Challenge was a competition publicly run three times with the most recent competition receiving a large number of entrants of high quality.

TABLE 2.6: Times and concept for each competition

Winter 2010	Tron	Tron lightcycles, controlling a single cycle
Fall 2010	Planet Wars	Basic RTS in space, multiple planets to control
Fall 2011	Ants	RTS controlling ant colony with many units to handle

Most interesting are the Planet Wars and Ants scenarios. These featured multiple objects that needed controlling, forming a large MAS, however almost all of the entrants used a

single AI to make all of the decisions for each unit.

2.4.8 General Video Game AI Competition

The General Video Game Artificial Intelligence (GVGAI) competition provides a video game version of the Stanford General Game Playing (GGP) competition. The games in GVGAI are based around traditional arcade games such as *Frogger* and *Space Invaders*. The games are expressed in Tom Schaul's Video Game Definition Language (VGDL) (Schaul, 2013), and converted into Java objects dynamically at runtime. This allows games to be written very concisely, as much of the logic is contained within the system and available for re-use and configuration. The agents are provided with a forward model for running simulations of how an action sequence performs. The original python implementation included a first person view point that gave a PO view though the competition lacks any PO constraints.

The GVGAI competition has been run since 2014 with the first results being published including sections from some of the top scoring agents (Perez-Liebana *et al.*, 2016). The competition now consists of multiple tracks covering learning, level generation and two player games. Predominant tactics for agents are to use either tree search algorithms or Rolling Horizon Evolutionary Algorithm (RHEA).

The competition has been expanding, now featuring a multitude of tracks to enter:

- Single Player Planning
- Learning
- Level Generation
- Rule Generation
- Two-Player Planning

The two player planning track is of particular importance here, as some of the games are co-operative in nature. In these two player games, each agent has independent score and

either joined or independent win conditions. Some of the games are co-operative, some are competitive. agents don't know which type of game they are playing either.

2.4.9 Showdown AI Competition

This competition focuses on single player Pokemon fighting, where each agent is given 6 random Pokemon and has to fight another agent with another 6 random Pokemon. The competition features a reasonable amount of observability restrictions, with the opponent Pokemon and their known moves being un-observable until they are revealed, and their exact statistics being un-observable permanently. It is therefore up to the agent to take what observations it can in order to reduce uncertainty and make better decisions such as when an opponent uses a move then an agent can add that information to its knowledge (Lee and Togelius, 2017).

2.4.10 microRTS Competition

This is a competition that uses a custom Java RTS game engine that was designed to be as basic as possible while still containing all the relevant characteristics of a RTS (Ontanón, 2013). There are three tracks running this year (Table 2.7).

TABLE 2.7: The three tracks in the microRTS competition.

Track	Determinism	Observability
Large state space	Deterministic	Full Observability
Partial Observability	Deterministic	Partial Observability
Non-Deterministic	Non-Deterministic	Full Observability

The PO track is of particular interest as scouting is a difficult task in RTS games. An agent needs to balance use of resources in scouting to obtain information or building more capable units of destruction.

2.4.11 Visual Doom AI Competition

This competition (*VIZDOOM*) provides only the same visual input to AI agents as the view usually given to human players. The view is PO, as only a certain angle of view can be seen in the direction that the player agent is facing. This competition asks agents to work with a large amount of data in real time and work out how to correctly filter what is relevant and what isn't on screen. Two tracks were run in 2016 with a very restricted track using a single weapon type and a known map as well as a less restricted track with multiple weapons, items and unknown maps. The less restricted track will force agents to learn more general strategies for Doom. The competition ran again in 2017 and 2018.

2.4.12 The 2K BotPrize

This competition (Hingston, 2010) runs on the Unreal Tournament 2004 FPS game. This competition focused on human like performance instead of the absolute best performance possible. The bots had access to a rich API and were judged by humans in a blind process where the judges didn't know they were viewing a human or AI agent.

2.4.13 Hearthstone AI

The popular online collectible card game, *Hearthstone* varies in a few ways to other card games. In *Hearthstone*, each player chooses the deck of cards that they use from over 1000 possible cards. The makeup of each players deck is hidden from the other player as the first form of PO. The current cards in each players hand are also hidden from other players as the second form of PO in *Hearthstone*. The Hearthstone AI competition ran at CIG in 2018 for the first time (*Hearthstone AI Competition*).

2.4.14 Hanabi

Hanabi is a PO co-operative card game described in detail in Section 4.3. The *Hanabi* competition ran at CIG 2018 for the first time (*Hanabi Agent Competition*), and provides entrants with a full Java based framework as well as numerous sample agents from the existing literature.

2.4.15 StarCraft AI

The *StarCraft* AI competition (*StarCraft AI Competition*) has been running since 2010, and is a RTS based competition that features PO forcing players to explore and scout ahead to make intelligent decisions. The agents use the BWAPI to play *Starcraft Broodwar* games against each other, and the PO, huge state space, and small time budgets combine to make a truly difficult challenge.

2.5 General Game Playing

This section will describe a basic history of the field of GGP itself. GGP is the field of writing computer AI agent that can play unseen games before, without requiring explicit programming to handle different games. Traditional AI such as IBM **Deep Blue** was world leading at playing *Chess*, but wouldn't have been capable of making the opening move for *Checkers* or even *Tic-Tac-Toe*.

2.5.1 METAGAMER

In 1993, Barney Pell outlined the first system and AI for playing multiple previously unseen games in his doctoral thesis (Pell, 1993). It functioned by analysing the rules of the game that were provided to it, and then using a simple tree search algorithm to play the game with reasonable proficiency. Learning from self play is considered as a potential source of

improvement to the AI. Pell showed that when playing *Chess* and *Checkers*, METAGAMER derived strategies that were similar to human play.

2.5.2 2005 - Stanford University

In 2005, the first General Game Playing competition was run at the AAAI[5] Conference. This competition focuses on board games written in the Game Description Language (GDL). GDL is a variant of **Datalog** and describes the game state as a series of facts and the game rules as a series of logical rules.

TABLE 2.8: List of winners of the Stanford GGP competition and their primary technique

2005	*Cluneplayer*	Depth first search with automatically constructed evaluation functions from game GDL (Clune, 2007)
2006	*Fluxplayer*	Depth first search with automatically constructed evaluation functions from game GDL (Schiffel and Thielscher, 2006)
2007	*Cadiaplayer*	MCTS (Finnsson and Björnsson, 2008)
2008	*Cadiaplayer*	MCTS (Finnsson and Björnsson, 2008)
2009	*Ary*	MCTS (Méhat and Cazenave, 2010)
2010	*Ary*	MCTS (Méhat and Cazenave, 2010)
2011	*TurboTurtle*	Unknown
2012	*Cadiaplayer*	MCTS (Finnsson and Björnsson, 2008)
2013	*TurboTurtle*	Unknown
2014	*Sancho*	MCTS (*Sancho goes Green*)
2015	*Galvanise*	Unknown
2016	*WoodStock*	MCTS (Koriche *et al.*, 2017)

As seen in Table 2.8, starting in 2007 the GGP competition saw a prevalence of Monte-Carlo Tree Search (MCTS) based approaches. I was unable to find evidence of how TurboTurtle or Galvanise were implemented, but once MCTS arrived on the scene it became the defacto method for competing. It is also worth noting just how quickly after the early MCTS papers were published that the first MCTS agents appeared.

[5] Association for the Advancement of Artificial Intelligence

2.5.3 General Video Game AI Competition

As described in detail in Section 2.4.8, a large amount of academic research since 2014 has been conducted by entrants to the competition as well as simply by people using the framework itself.

2.5.4 Alpha Zero

Silver *et al.* (2017) improved their earlier work on Alpha Go (Silver *et al.*, 2016), generalising the algorithm to more than a single domain. Alpha Zero is able to learn to play *chess*, *shogi*, and *Go* at a superhuman level in under 24 hours. Alpha Zero is a MCTS algorithm that uses self-taught board evaluation and move evaluation Deep Neural Networks (DNN).

2.6 Game Design

Adjusting the PO in Ms. Pac–Man will arguably create different versions of the original game, and should be treated as a form of game design. Computer games are fundamentally games, and many techniques for designing games are equally applicable to computer games such as Ms. Pac–Man. AI assisted game design is a field concerned with at least partial automation of the game design process, such as using AI to tune game parameters like PO in Ms. Pac–Man. This section will highlight relevant work on game design including AI based game design, a technique that the experiments in this chapter will use.

Hunicke *et al.* define the Mechanics, Dynamics, and Aesthetics (MDA) framework and by their definition the PO restriction added into Ms. Pac–Man is a mechanic, as the restriction is a fundamental part of the control mechanism (Hunicke *et al.*, 2004). Koster states that fun comes from the pursuit of mastery of a game (Koster, 2013). If this is true then potentially a more difficult game would be more difficult to master, but also provide a longer source of fun as the player tries to master it. This is investigated in this chapter by trying to predict fun using the difficulty that AI agents experience. Koster also makes the observation that

games are often linked, changing only by as little as a single element sometimes. This is the process used to create PO Ms. Pac–Man for this chapter - the same game but with the single element of visibility altered.

Nelson and Mateas describe a method to formalize game mechanics as well as how to generate games automatically that utilize those mechanics (Nelson and Mateas, 2007). The authors make use of both WordNet and ConceptNet to reason about which verbs and nouns make sense together. A working example is provided that produces games in the style of Nintendo's *WarioWare* series. Isaksen *et al.* give a method that explores the game space of the popular mobile game *Flappy Bird* (Isaksen *et al.*, 2015a; Isaksen *et al.*, 2015b; Isaksen *et al.*, 2017). By varying multiple parameters such as jump height, tube spacing, and bird speed they locate "playable" games and try to find the most different but still playable locations in the design space. They use play testing to obtain a more subjective evaluation of the different games and find that they are significantly different to the original game in challenge, game feel, and theme. This is similar to the approach used in this chapter where we vary the PO constraints and analyse the effects before using play testing to verify the results. Khajah *et al.* used Bayesian methods to design games and evaluated them to maximise user engagement with the generated games (Khajah *et al.*, 2016). Their evaluation used participants that played the games for several minutes and then they had the option to either stop or continue to play the game with no further compensation. Total play time was used as well as a post-game survey. The results indicated that a user's self-perception of competence was critical.

There have been a number of studies on the effects that computer games have on people. A large amount of this work has been on the psychological impact of games on both adults and children, but a reasonable amount of research has been carried out on measuring enjoyment in games. Naturally there is interest in discovering why games are fun, but so far much of the research has simply focused on actually measuring fun itself. Beume *et al.*, 2008 compared algorithmic measurements (Yannakakis and Hallam, 2004) with questionnaire responses (Beume *et al.*, 2008). They found that the algorithmic measurements were

not suitable for measuring fun, and that better answers came from questionnaires. Ryan *et al.* measure game enjoyment in a number of studies and find that the self-determination theories: autonomy, competence, and relatedness predict enjoyment of games (Ryan *et al.*, 2006). Fang *et al.* developed a questionnaire that extends the work of Nabi and Krcmar (Nabi and Krcmar, 2004) to measure affective, behavioural, and cognitive reactions of respondents (Fang *et al.*, 2008). This technique was then revised with the input of expert consultants as well as exploratory and confirmatory card sorting sessions. The final version consisted of 11 questions to measure enjoyment in games.

Chapter 3

Algorithms

This chapter gives a detailed description of the more common algorithms used throughout this thesis in a single place. Section 3.1 describes the anytime tree search algorithm Monte-Carlo Tree Search (MCTS) that is frequently used in games and General Game Playing (GGP) environments. MCTS features in Chapters 5 and 7. Section 3.2 covers Genetic Algorithms (GAs) that are also often used in rolling horizon form to play games or as parameter tuners to train neural networks. A GA is used in Chapter 5

3.1 Monte-Carlo Tree Search

MCTS is a tree search algorithm originally proposed in 2006 (Coulom, 2007; Kocsis and Szepesvári, 2006; Chaslot *et al.*, 2008a).

MCTS is a tree search algorithm that in games is typically applied directly to the action space. MCTS requires the ability to simulate action sequences and retrieve at least some form of reward (or lack of) from the results of these simulations.

The absolute minimum that MCTS needs to operate is the ability to do three things. The first is for the forward model M_t at time step t to provide the observed score r_t. The second is for the forward model M_t to provide the set of actions available $S_t = \{A_0 \dots A_n\}$ at time step t. Finally the forward model M_t needs to provide the transition to $S_{t+1} = F(S_t, A_t)$.

This means that you can see the score of a state, the available actions for a state, and the resulting state obtained by applying an action to a state.

The basic steps for MCTS are:

1. *Selection* - Selection is the stage where the algorithm navigates the search tree, selecting optimal nodes (Based on the selection policy) until it reaches a leaf node.
2. *Expansion* - Expansion is the stage whereby the leaf node is expanded by adding one of the remaining child states if it is not a terminal node.
3. *Simulation* - Simulation is the stage where the algorithm forwards the model until a result is achieved. There is usually a policy that defines how the simulation is made, with the simplest being random possible moves. This policy is often called the 'default policy'
4. *Back propagation* - Back propagation is the stage where the results of the simulation are propagated up the tree, so that they can influence the selection phase.

MCTS has been applied to a wealth of domains (Browne *et al.*, 2012) and is one of the primary algorithms in use for GGP (Finnsson and Björnsson, 2008). Primary advantages are that, when given a sufficient forward model, MCTS does not require any strategic knowledge about the game itself in order to play. This non reliance on strategic knowledge is what gives MCTS the flexibility to perform well in GGP environments and is likely to be a key algorithm for my own work.

3.1.1 Selection Policy

The selection policy is an important part of any MCTS algorithm. Early algorithms used the simplest of policies, making just random choices when selecting nodes. The subsequent sections discuss various alternative tree policies that have been proposed since.

Upper Confidence Bound applied to Trees

Upper Confidence Bound applied to Trees (UCT) is one the most common selection policies applied to MCTS to the extent that it is considered implicit unless specified otherwise that MCTS uses UCT. UCT applies the Upper Confidence Bound (UCB) algorithm, presented by Auer *et al.*, to the selection phase by viewing the choice of child node as being equivalent to a Multi-Armed Bandit (MAB) problem (Auer *et al.*, 2002). UCB was a well researched algorithm for solving the MAB problem by balancing a desire to explore arms that had not been visited often (in relation to others) and the desire to exploit arms that had proven worthwhile.

UCT allows MCTS to build asymmetric trees that have been proven to converge on an optimal solution, but providing good solutions much earlier than traditional search techniques.

Sequential Halving applied to Trees

Sequential Halving applied to Trees (SHOT) takes the Sequential Halving algorithm and uses it as a replacement for UCB (Cazenave, 2014).

Sequential Halving is based on sequentially eliminating moves from consideration. The algorithm consists of rounds, and in each round all the moves that remain are sampled equally. The results of this sampling, added to any previous sampling is used to sort the moves and eliminate the worst performing half. The next round then begins - typically allocating more budget per move than the previous round.

SHOT has numerous advantages over UCT. The first is the ability to parallelise the algorithm much more efficiently, as only at the end of a round does the tree need to be updated and used to perform calculations. This reduces the amount of time that a parallel implementation requires locking and synchronising. SHOT spends significantly less time in the tree, unlike UCT that traverses the tree every iteration.

SHOT lacks a tuning constant, removing much of the work that is required of UCT in finding the correct exploration constant to use for a particular problem domain. UCT is in some circumstances very reliant on the correct exploration constant for good performance.

Last Good Reply

Last Good Reply (LGR) is a technique that keeps successful replies that occur in the playouts in memory to bias the move choices (Drake, 2009). This makes the assumption that, regardless of state - answering a move x with move y is at least usually the best choice. This is difficult to know for sure in GGP and may not prove overly useful. In a large number of games, the current state is important. Tak *et al.* (2012) apply LGR to the GGP competition and find that LGR has promise in GGP.

3.1.2 Default Policy

There are two primary methods for performing rollouts in MCTS (Browne *et al.*, 2012).

Light rollouts consist of randomly choosing the move to operate and have two primary advantages. This is the original, simpler, default policy implemented. The first advantage is that random choice is incredibly fast to calculate, leading to more rollouts per second compared to heavy rollouts. The second advantage is that random choice considers all possibilities given enough budget. This allows optimal strategies to eventually be found[1], allowing MCTS to converge to optimality.

Heavy rollouts involve performing some decision making into filtering the list of all possible moves in order to reduce the action space and better approximate a reasonable player. If a heuristic decides that a move is a bad idea, then the algorithm won't waste cycles exploring those possibilities. This focuses the statistics that are generated on sensible portions of the action space. Heavy rollouts do however rely on domain specific knowledge

[1] As t tends to infinity, which may not be useful

and aren't often suitable for GGP. Some useful knowledge can however often be extrapolated from Game Description Language (GDL) or from observations during the game that can be used instead. Heavy rollouts can lead to MCTS missing optimal branches of the tree if the heuristic is incorrect.

3.1.3 Partially Observable Games

In order to handle Partial Observability (PO) games, there are a number of possible enhancements to MCTS. Key to these techniques is the concept of determinising a PO game state into a possible completely observable game state. It is often possible to generate all the possible game states that an agent might find itself in. In Battleships, for example, we can try all valid ship arrangements that do not contradict any information we currently have such as existing hits or misses. This might show some positions as being more likely to contain a ship than others. Determinising is the act of randomly choosing one possible game state out of all the possible game states and creating a perfect information copy of it to simulate with (Cazenave, 2005).

Determinised UCT

This technique creates N individual trees, giving each tree a unique determinised state at the root (Cowling *et al.*, 2012). The trees are explored as usual for MCTS and the move chosen at the end of the algorithm is the move for which the number of visits from the root across all trees is greatest.

Information-Set Monte-Carlo Tree Search

Information-Set Monte-Carlo Tree Search (IS-MCTS) is a technique similar to determinised UCT but maintains a single tree, and changes the determinisation on each iteration of the algorithm (Cowling *et al.*, 2012). Special care is taken to ensure that moves already in the

tree that are not possible in this iteration are ignored, as well as new moves not previously considered included.

Monte Carlo Counterfactual Regret

Monte-Carlo Counterfactual Regret (MCCFR) is an enhancement to the Conterfactual Regret (CFR) algorithm from Lanctot *et al.* (2009). CFR requires the entire game tree to work, proving problematic on most games of interest in this thesis as their trees are too large to expand. MCCFR samples blocks of paths from the root to a leaf, and computes counterfactual regrets for these.

Partially Observable Monte-Carlo Planning

Silver and Veness (2010) propose the Partially Observable Monte-Carlo Planning (POMCP) algorithm to provide the intelligence of full-width planning with lower performance requirements. POMCP is the combination of monte-carlo belief state updates with PO-UCT.

3.1.4 Parallelisation

MCTS typically improves the optimality of its decision the more computational budget that is provided to it. Parallelisation provides a way to obtain more iterations in a fixed time budget than otherwise possible. MCTS typically plays more intelligently when given more iterations to decide on a move. This means that parallelisation is a good way to play games when given a fixed actual time budget on multi-core machines.

There are multiple approaches to parallelising MCTS, typically identified by which point in the tree the switch to parallel code occurs.

Single Run Parallelisation

Cazenave and Jouandeau (2007) investigated this method of developing completely separate trees, and aggregating the results. This benefits from the low amount of communications needed between nodes which both simplifies the programming and enables a low overhead to really get the most out of the available performance on tap. There is also the additional benefit that the possibility for MCTS in non-deterministic games to miss certain branches due to early poor scores can be avoided, as this is similar to multiple restarts in a GA.

Multiple Runs Parallelisation

Cazenave and Jouandeau (2007) also investigated this method that involves a master thread and worker threads whereby the master periodically aggregates the trees and re-distributes it to the workers who then continue to work on the new aggregated tree. This is more complex to code, and the authors had to divide the number of wins and games of the root moves by the number of workers to provide better results.

At the Leaves Parallelisation

Cazenave and Jouandeau (2007) also investigated this method that involves developing just one tree, traversing it on a single thread and then running multiple rollouts from the chosen leaf node. This does not require complex parallel code writing but does not make the best use of all the computing power available as there are many points where only a single thread is operating as it traverses the tree.

Tree Parallelisation

Chaslot *et al.* (2008b) investigated this method that is the process of traversing and running simulations from a single tree with multiple threads. This is complex to write, complex to

debug, and difficult to get good performance from. Most of the performance gains come from using the correct locking strategy although it can introduce large overheads.

Global Mutex This technique uses a single global mutex for the entire tree, locking access for tree traversal and back propagation. This technique is simple, but would lead to a lot of blocking with multiple threads waiting for the tree to become unlocked.

Local Mutexes This technique uses a mutex for each node of the tree that is being visited. This allows different sections of the tree to be searched, but the asymmetric nature of MCTS tree construction means that later in searches the same sections of the tree will be searched.

3.1.5 Learning Domain Knowledge

GGP agents are deprived of domain specific knowledge. Most research into Artificial Intelligence (AI) has revolved around the application of domain specific knowledge in some way being used to improve playing performance. The two primary GGP competitions provide some limited and generic information about the games being played to the agents. Méhat and Cazenave (2010) present the workings of Ary. This program used **Prolog** to generate legal moves, apply moves, detect end of game states, and determine the score for each player. These abilities allowed Ary to implement a forward model, making possible the MCTS algorithm. This use of domain knowledge was essential, as without it there would have been little chance of the MCTS algorithm working correctly. The General Video Game Artificial Intelligence (GVGAI) Competition makes this sort of analysis unnecessary with its provision of a forward model, as well as impossible with the non-provision of the game description. Finnsson and Björnsson (2010) describe a number of techniques for learning policies for MCTS node selection in GGP. Prior work that this paper draws upon is the earlier papers on CADIAPLAYER (Finnsson and Björnsson, 2008). Move-Average Sampling Technique (MAST) is a technique for biasing tree search towards more promising branches by remembering statistics about individual moves, irrespective

of the location in the tree or game state. Tree Only Move-Average Sampling Technique (TO-MAST) is similar to MAST except that statistics are only backed up the current path of the tree. Predicate-Average Sampling Technique (PAST) is a technique that instead of generalising over individual moves, it generalises over moves and some predicates p. Biasing is done towards moves with the most true predicates, and the best move when they are equal. Sharma *et al.* (2008), uses self play between random AIs to learn the value of individual moves within a new game from the GGP Competition. These values are then used to influence the UCT algorithm into exploring moves indicated as better. The paper considers allowing the AI to continue to update the learnt values during the competition. Silver *et al.* (2016) describe the techniques used in the AI agent *AlphaGo*. This AI successfully won a 5 game match (4:1) against 9-dan ranked Lee Sedol. The essential improvements are the extensive use of off-line learning through the use of Deep Neural Networks (DNN) to learn a significantly improved board evaluator that is able to assist the MCTS algorithm that is the core part of decision making.

3.2 Genetic Algorithms

GA's are a class of algorithms that are inspired by the natural process of evolution (Anderson-Cook, 2005). GA's evolve populations of candidate solutions until the search criteria is met. The best candidate at the end of the search is then chosen. A number of genetic operators can be defined to operate on the candidate solutions in order to alter them into new candidate solutions.

3.2.1 Fitness Calculation

Candidate solutions need to be evaluated to calculate their fitness. The best solution at the end is often chosen for actual use. Fitness functions can be critical to the computational cost of a GA because often hundreds or thousands of candidate solutions will be evaluated for fitness. Designers will need to balance accuracy with computational cost sometimes.

UP	LEFT	LEFT	UP	UP	RIGHT

FIGURE 3.1: Diagram of a possible solution to a simple pathfinding task.

3.2.2 Candidate Solution Representation

Candidate solutions need to be represented in a form that makes them operable upon as well as able to be evaluated for fitness. Typically the solutions is represented as a series of genes. Each gene can represent something different and be as simple or complex as needed. For a path finding algorithm, the candidate solution could be a series of moves from the starting position for example as shown in Figure 3.1.

3.2.3 Genetic Operators

Genetic operators are a class of function designed to operate on candidate solutions. There are three main types of operator that are described next.

Selection Rules

This class of operator determines which candidate solutions from the population are selected for operating upon. Some common selection rules are described.

Random Randomly selects a valid member of the population to be operated upon. Very fast and easy to implement, however it does not take the fitness of a member into account.

Roulette Wheel Randomly selects a valid member of the population, where the chance of being chosen is proportional to the fitness of the candidate solution (Zhong *et al.*, 2005). A performance issue with this technique is the requirement to evaluate every member of the population's fitness which can be computationally expensive.

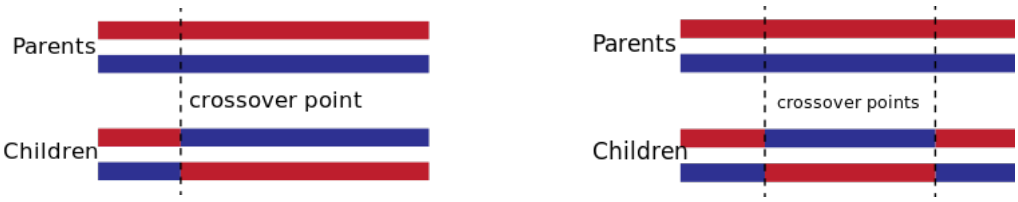


FIGURE 3.2: Left: Diagram of single point crossover (R0oland, 2013a). Right: Diagram of two point crossover (R0oland, 2013b)

Tournament Randomly selects n candidate solutions and returns the candidate solution with the best fitness (Zhong *et al.*, 2005). Not all candidates need their fitness calculated for this method to work, so can be a faster technique than Roulette Wheel.

Crossover Rules

This class of operator determines how to combine 2 or more candidate solutions to create one or more new candidate solutions. Single point crossover involves taking two parents, selecting a point in their genome and creating two children that start with the genes from one parent and switch to the genes from the other parent at the crossover point. This is shown in a diagram in Figure 3.2. Two point crossover is similar, with a second point later in the genome where the child switches back to the original parent. This can be generalised to n-point crossover where n points are used to switch parent at.

Mutation Rules

This class of operator determines how to create a new candidate solution from a parent candidate solution. The new candidate solution is typically very similar, i.e a mutation of the parent candidate solution. Often mutations are a case of choosing either a new random valid value for a gene, or in the case of numbers adding small values to the gene.

Chapter 4

Games in this Thesis

This chapter gives descriptions of the multiple game environments that are used in this thesis.

Section 4.1 describes the *Tiny Co-op* domain that is a simple domain created for the work in this thesis to test Monte-Carlo Tree Search (MCTS) and Genetic Algorithm (GA) Artificial Intelligence (AI) agents in a co-operative Partial Observability (PO) environment. To ensure that the agents do not use too much information about the world to operate, the ability to observe the environment is severely restricted with only the score being revealed to agents in the forward model.

Section 4.2 gives a description of the *Ms. Pac–Man* environment that was modified by the author to add a PO restriction. The domain then had messaging added to give the ghosts the ability to co-operate with each other in their task to find Ms. Pac–Man. An overview of prior research in the Ms. Pac–Man domain is also provided (Section 4.2.3).

Section 4.3 gives a description of the environment that was created to develop co-operative agents for the card game *Hanabi*. This game poses the unusual challenge that the PO restrictions are applied to your own cards, and that communication is strictly controlled by the game. Prior research in *Hanabi* is discussed in Section 4.3.1.

This chapter contributes to the thesis a simple co-operative environment (*Tiny Co-op*), a PO modified version of the *Ms. Pac–Man*, and a co-operative card game (*Hanabi*) that makes

communication part of the game.

The sections of this chapter are adapted from:

Section 4.1 Piers R. Williams, Joseph Walton-Rivers, Diego Perez-Liebana and Simon M. Lucas (2015). 'Monte Carlo Tree Search Applied to Co-operative Problems'. In: *CEEC'2015 - IEEE Conference on Computer Science and Electronic Engineering*. IEEE CEEC. IEEE Computer Society, pp. 219–224

Section 4.2 Piers R Williams, Diego Perez-Liebana and Simon M Lucas (2016). 'Ms. Pac-Man Versus Ghost Team CIG 2016 Competition'. In: *CIG'2016 - IEEE Conference on Computational Intelligence and Games*. IEEE CIG, pp. 420–427

Section 4.3 Joseph Walton-Rivers, Piers R Williams, Richard Bartle, Diego Perez-Liebana and Simon M Lucas (2017). 'Evaluating and Modelling Hanabi-Playing Agents'. In: *Congress on Evolutionary Computation, 2017. CEC'17. IEEE Conference On*. IEEE, pp. 1382–1389

4.1 Tiny Co-op

Tiny Co-op is a small, grid based world featuring obstacles, goals, and interactive items. Two avatars are controlled independently by either AI agents or by a human controller. Moves are polled for simultaneously from the agents, and executed sequentially in the world. The main objective in *Tiny Co-op* is for all agents to visit each goal once. The score for an agent visiting a goal for the first time is equal to one divided by the number of goals and agents. This means that when every agent has visited every goal the team will have scored exactly one point and the game can end. This score system also promotes co-operation because all scores are shared, with no benefit to being the agent that obtains the score. This should remove competition between agents.

TABLE 4.1: Possible objects in *Tiny Co-op*.

Name	Colour	Description
Floor	Grey	Passable
Wall	Black	Impassable and fixed location
Agent	Dark Yellow	Moving objects controlled by players
Door	Dark Blue	Doors can either be open or closed. Doors are impassable and visible when closed, and are passable and invisible when open. Doors are controlled by corresponding buttons. When at least one linked button is active, the door will be open
Button	Red	Buttons can be either active or inactive. Buttons are activated by an agent residing on the same location as the button
Goal	Bright Yellow	Goals are passable objects that confer all agents with a portion of the score when visited. Every agent must visit every goal at least once to obtain maximum score. Revisiting a goal gains no additional score

4.1.1 Objects

The complete list of objects in the *Tiny Co-op* environment are shown in Table 4.1. Each object will fully occupy a single location in the grid world, rendered in a unique colour per type of object (See Figure 4.1). Some objects have an ID associated - these are rendered in text at the centre of the grid location. IDs are important for linking objects together - a button with the same ID as a door will open that door when an agent is on that button.

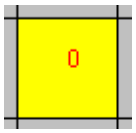


FIGURE 4.1: An example of how objects are rendered in *Tiny Co-op*

4.1.2 Movement

Agents are able to perform one movement per turn, with both agents performing a move in each turn. There are five available moves: up, down, left, right, and no-op. These are described in detail in Table 4.2. Agent moves are evaluated in order of ID, with agent 0 going first. Agents cannot occupy the same location as each other, and an attempt to do so will result in the agent with the highest ID not moving at all instead. The same rules apply

to agents that try to travel to a non-traversable area of the map such as walls or closed doors.

TABLE 4.2: Table of moves in the *Tiny Co-op* domain

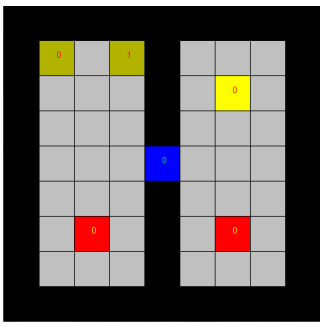
Name	Vector	Description
Left	(-1,0)	Move the agent one grid square to the left
Right	(1, 0)	Move the agent one grid square to the right
Up	(0, -1)	Move the agent one grid square up
Down	(0, 1)	Move the agent one grid square down
No-Op	(0, 0)	No movement

4.1.3 Maps

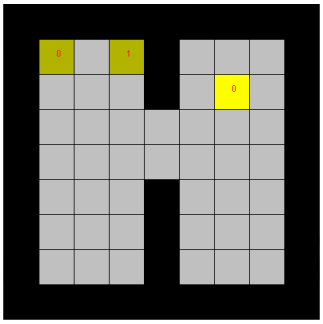
A number of test maps have been also created for this environment.

Single Door

This map is a basic map with only a single door (blue). One of the two agents (gold) must open the door for the other agent to go through. The goal (yellow) can then be collected before opening the door from the other side to let the left agent through. This problem requires both pathfinding, as all problems require, and co-operation to solve.

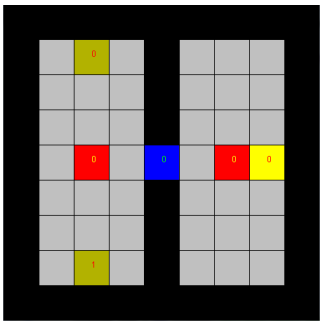


Pathfinding



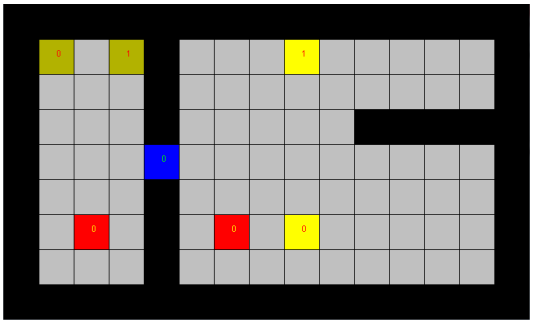
This map is Single Door without the door, and is designed to test an agent's ability to navigate the environment and locate the goal. This problem does not require co-operative behaviour, as both agents are able to reach the goal without assistance from the other.

Symmetric Single Door



This map is a modification of Single Door to place each agent closer to the button. The goal has also been moved so that the button can be opened on the way to the goal

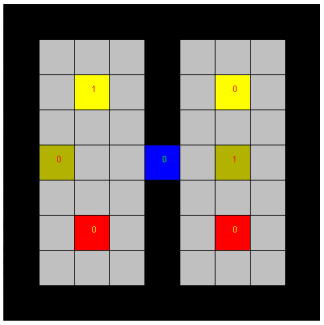
Extended Side



This map is an extension of Single Door with a second goal with more spread out rewards

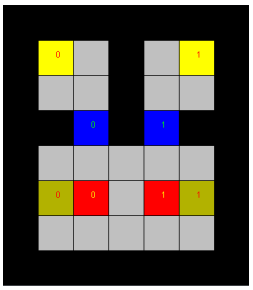
Side By Side

This map is designed to be symmetric with a goal that each agent can initially reach and one that they cannot reach without help. The symmetry can provide for clashes if both agents try the same tactic



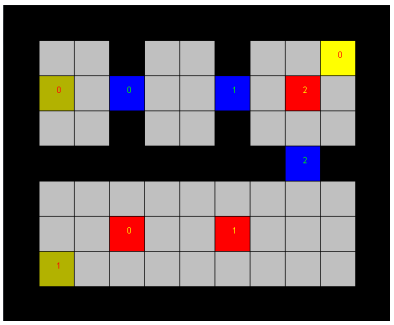
Butterfly

This map is designed to force each agent to need to be let into a room, and out of it again. Perfectly symmetrical, it is possible for both agents to decide to either head for the doors or the buttons, when ideally exactly one agent should head for the button and the other should head for the door. The left button will only open the left door, with the right button operating the right hand door



Airlock

This map is designed to provide asymmetric roles to the agents. The uppermost agent, in the first position, must travel through the two doors in order to reach the goal. The agent in the second position must open the doors for the airlock in order to allow the first agent to reach the goal and the button that will allow the second agent to reach the goal itself.



4.2 Ms. Pac-Man Vs Ghosts

This is a software environment designed to mimic the *Ms. Pac–Man* arcade game with the modification of adding PO constraints, to create a different but fun experience. The environment was originally implemented for the second iteration of the Ms. Pac–Man Vs Ghost Team competition (Rohlfshagen and Lucas, 2011). This section will describe the full implementation of the game environment including the modifications made to it.

This game consists of 5 agents: a single Ms. Pac–Man, and 4 Ghost agents. The world is a maze environment, with non-traversable walls. There is a ghost lair in the centre, where the ghosts start and also respawn after being eaten. Pills are placed in the corridors for Ms. Pac–Man to collect as well as larger power pills that, for a short period of time, allow Ms. Pac–Man to consume the ghosts and score additional points. A view of the game is shown in Figure 4.2. The various characters in the game are shown in Figure 4.3. Eating a pill earns Ms. Pac–Man 10 points and eating ghosts earn 200 points for the first ghost but doubling each time up to 1600 points for the fourth ghost. The maximum points s for a maze where n is the number of pills in the maze is $s = 10n + 4 \times (200 + 400 + 800 + 1600)$.

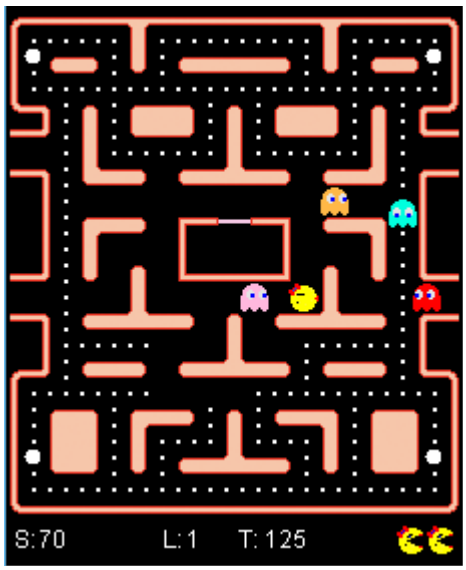


FIGURE 4.2: A view of the basic Ms. Pac–Man game

FIGURE 4.3: The various characters of the game, Left to Right: Blinky, Inky, Pinky, Sue, and Ms. Pac–Man

(a) Radius (b) Line-of-Sight (c) Forward Facing Line-of-Sight

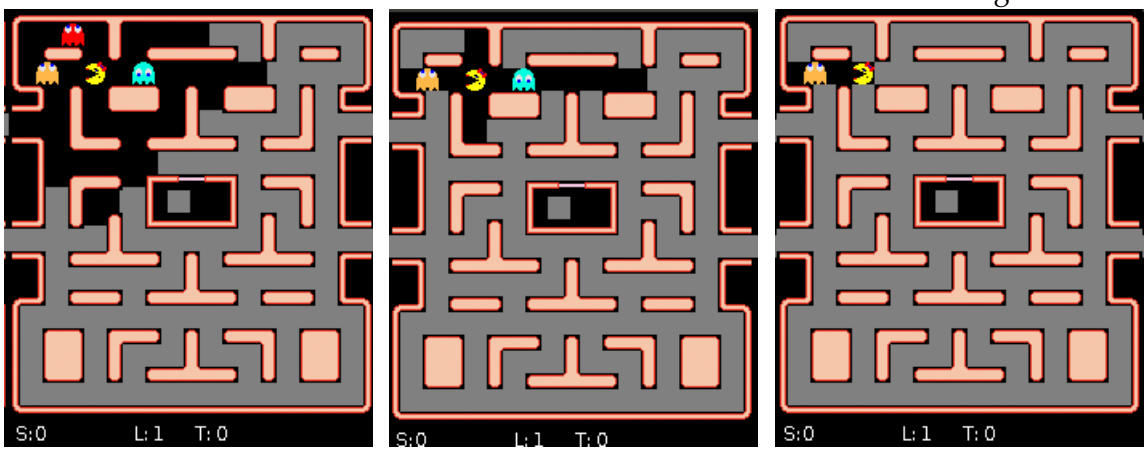


FIGURE 4.4: The three modes of Partial Observability visualised.

4.2.1 Partial Observability

PO is the impairment of the ability of an agent to completely observe the world that it is situated within. There are three different implementations of PO in the engine, shown in Figure 4.4, and are currently applicable to all agents within a game equally. PO restrictions are calculated based on the individual locations of each agent, so each ghost and Ms. Pac–Man have unique areas that they can observe, but each area is calculated with the same method.

Line-of-Sight

Line-of-Sight (LOS) is where the agents can see in straight lines up to a limit unless there is an obstacle in the way. Obstacles are considered to be the walls in the maze. Ghosts and pills do not count as obstacles. This applies to both Ms. Pac–Man and the ghosts and means that they can see both forwards, backwards, and sideways. This method is simple

to implement as well as fairly realistic, with agents not able to see around corners, like real people.

Forward Facing Line-of-Sight

This is an additional restriction on LOS where the agent can only observe in the direction that they are currently travelling.

Radius-Based Partial Observability

Radius-based PO is a simple technique where we consider that anything within a distance d from the agent is considered visible. This allows agents to view other agents that are around corners or behind walls. This is not particularly realistic, but does provide more information to the agent than LOS.

4.2.2 Messaging

Communication is the cornerstone of teamwork and vital to the creation of co-operative agents. In this game, the communication is heavily controlled by the game in order to force agents to share information rather than attempt to control the actions of each other. The communication component is composed of two main parts - the messenger and the message. The messages allowed are presented in Table 4.3.

TABLE 4.3: Table of messages allowed in Ms. Pac–Man Vs Ghost Team. All locations are represented as node indices of the node-graph, and headings correspond to the four cardinal directions possible in the game.

Message Type	Description
Pacman Seen	A message informing others that Ms. Pac–Man has been seen.
Pacman Heading	A message informing others which direction Ms. Pac–Man has been seen heading.
I Am	A message informing others where the sender is currently located.
I Am Heading	A message informing others where the sender is currently heading.
Pill Not Seen	A message informing others where there is not a pill

Messages can be either sent to a single recipient or broadcast to all ghosts on the map.

TABLE 4.4: Feature vectors for evolved neural network to play Ms. Pac–Man

Input	Description
$g_1 \dots g_4$	distance to each predatory ghost
$e_1 \dots e_4$	distance to each edible ghost
x, y	location of current node
pill	distance to nearest pill
power pill	distance to nearest power pill
junction	distance to nearest junction

The time it takes for a message to be delivered can be calculated using $t_d = t_a + \delta_c + (\delta_x \times \delta_m)$ where t_d is the tick a message will be delivered, t_a is the tick that a message arrives, δ_c is the constant delay added to all messages, δ_m is the delay added to messages of this type, and δ_x is the multiplier applied to message delays. This allows a level of configurability in how quick the messages get delivered, for example all messages can be delivered equally fast if $\delta_m = 0$.

4.2.3 Prior Research

A following discussion investigates different approaches to playing *Ms. Pac–Man*, as either Ms. Pac–Man or the ghost team.

Neural approaches

Neural techniques are inspired by theories about how neurons in the brain work together to make decisions. Artificial neural networks work in this way, with a series of inputs going into the neural net and a series of outputs coming out of it. Inside the network, both the way that the neurons are connected (topology) and the weighting of each connection has a large influence on the final outputs. Lucas (2005) explored using a simple $(N + N)$ Evolutionary Algorithm (EA) with $N = [1, 10]$ to train the weights of a neural network that was used as a location evaluator to decide the next move for Ms. Pac–Man. The neural network used a simple feature vector (Table 4.4) as input for each considered location. The location with the highest score is chosen as the next move. Burrow and Lucas (2009) compared two

different approaches to learning to play the game of *Ms. Pac–Man*. The two techniques used were Temporal Difference Learning (TDL) and EA. These techniques were each used to train a Multi-Layer Perceptron (MLP) that was then evaluated within the game. The EA was subsequently shown to be superior to TDL. Schrum and Miikkulainen (2014) investigated the use of modular neural networks to control Ms. Pac–Man. The agent was developed for the same simulator as used in the Ms. Pac–Man Versus Ghost Team. The final result was that the best performing agents were those that evolved to lure ghosts to power pills for easy eating, which is a common human tactic when playing. Wittkamp *et al.* (2008) investigated using an online learning technique, Neuro-Evolution Through Augmenting Topologies (NEAT), to evolve the controllers for the ghost team. Each ghost evolves separately, but shares the score of the team. The paper tries several different approaches, with each one being an adaptation to the features or evaluation functions to try and bring out certain behaviours. In the first experiment, the authors used a simple performance metric using the number of lives that Pac–Man has remaining for the primary reward and distance from ghost to Pac–Man (inversed when edible) for a secondary reward. This constructs a chasing and evading set of ghosts that performed significantly better than the original basic AI. In the second experiment, the authors altered the performance metric to include a tertiary score that promotes ghost dispersion. This resulted in worse performance than the results from experiment 1 despite the reasonable idea that dispersed ghosts can trap Pac–Man better. In the third experiment the metrics were largely tweaked to a new system. Rank 1 was still Pac–Man's lives, but Rank 2 and 3 aimed to reduced the count of edible ghosts and increase the count of chasing ghosts. This provided interesting behaviour with chasing ghosts moving towards vulnerable ghosts in order to deter or eat Pac–Man if he tries to chase the vulnerable ghost. The inverse of this was also observed, with vulnerable ghosts travelling towards chasing ghosts looking for protection. An interesting behaviour that was observed was that ghosts attempted to protect power pills from being eaten in the first place. All these experiments gave interesting behaviour but the best score was obtained with the first experiment.

Tree-search approaches

Tree search is a technique that involves building game trees through simulations, typically with statistics and heuristics to evaluate positions rather than fully expand the tree to the games end. These statistics and heuristics are used to make decisions about how to play the game. One particularly common tree search algorithm used in this thesis, MCTS, is covered in more detail in Section 3.1. Samothrakis *et al.* (2011) used a 5-player max^n MCTS tree with limited tree search depth. The paper experimented with both MCTS for Ms. Pac–Man and for the ghosts. In order to better guide the MCTS a target node of the maze is chosen as a "*game preferred node*" (*gpn*). This assigns a reward for reaching that node and no reward for any other node. This allows Ms. Pac–Man to reach a terminal state in the game tree without dying. The *gpn* is set when there are no pills or power pills within the tree search depth limit. This leaves a simple reward function for Ms. Pac–Man of 1.0 if Ms. Pac–Man consumes the last pill on the map, 0.8 if the preferred node is reached, 0 if Ms. Pac–Man died, or 0.6 otherwise. The ghosts are rewarded for being close to Ms. Pac–Man and a top score of 1 for Ms. Pac–Man being eaten by any of the ghosts. Pepels *et al.* (2014) described their work in creating an entrant to the Ms. Pac–Man Versus Ghost Team competition (WCCI[1]'12 and CIG[2]'12). An MCTS agent is described in detail containing a number of enhancements and alterations designed to improve performance specifically in *Ms. Pac–Man*. The first enhancement described involved not considering reversal of direction outside of the first level of the MCTS tree. This reduces the branching factor of the tree, and also guarantees some progress be made along the map. Additionally, this will enable branches of the tree to more directly reflect different game states. If Ms. Pac–Man takes 20 ticks, 15 of which travelled forward and 5 backward then 10 steps forward would have been made. The number of different combinations for order of moves would consist of a large number of leaf nodes corresponding to near identical game states - diluting the statistics. A second enhancement consisted of storing additional statistics within each MCTS node, corresponding to the scores obtained through the use of different tactics. The "ghost-score"

[1]IEEE World Congress on Computational Intelligence
[2]IEEE conference on Computational Intelligence in Games

tactic corresponds to score obtained by eating ghosts, the "pill-score" tactic corresponds to the score obtained by eating pills, and the "survival-score" tactic corresponds to a reward given for surviving. The tactic to use to bias the search is selected at each node. The "ghost-score" tactic is selected if at that node Ms. Pac–Man is currently able to consume ghosts; the "pill-score" tactic is the default tactic; and the "survival-score" tactic is applied when the survival rate of the previous or current search is below the threshold $T_{survival}$. Rewards were altered to account for time taken to achieve them. The third enhancement described consisted of reusing search trees in subsequent ticks of the game. This information is worth keeping in order to effectively expand the computational budget available to the algorithm. The simulation phase of the algorithm contains rule based approaches to both the ghosts and to Ms. Pac–Man. The paper experimented with each enhancement switched off and provided a clear analysis of the effect of each one. The agent itself came second and first place (chronologically) in the competitions that it entered. Nguyen and Thawonmas (2011) presented their winning agent team that was entered into the CEC[3] 2011 Ms. Pac–Man vs Ghost Team Competition. Pinky, Sue, and Inky were controlled with an MCTS agent while Blinky was given a rule based agent. The MCTS agents were given a k Nearest Neighbour (KNN) learned model of Ms. Pac–Man to use. Given the set of features about the current game state, they were able to predict what Ms. Pac–Man would do.

Evolutionary approaches

Evolutionary techniques use methods inspired by natures way of problem solving. Solutions are evolved through repeatedly mutating, cross-breeding, and selecting candidates in a genetic pool. Over time, the candidates improve in fitness until a viable solution is found. They are explored in more detail in Section 3.2. Handa and Isozaki (2008) used fuzzy logic tuned by a 1+1 EA. The rules were tuned with the EA and consisted of a series of predefined rules about avoidance and chasing as well as pill collecting. Alhejali and

[3]IEEE Congress on Evolutionary Computation

Lucas (2010) extended the work by Koza (1992) on Genetic Programming (GP) with *Pac–Man*. Using Koza's original functions, as well as implementing his suggested additions and a small number of new functions, the authors found that when focusing on a single maze, the controllers would typically eat the power pills and chase ghosts, but evolved little strategy for consuming the standard pills. This lead to many agents losing interest in the game and simply wandering until death. When the agents were asked to repeat the first maze infinitely, being presented with a newly populated maze each time they successfully cleared their current maze the agents evolved a different strategy that focused on maze clearing to continue to obtain as many points as possible without dying. Alhejali and Lucas (2013) studied the idea of using GP to evolve heuristic functions for a MCTS agent. The GP system was strongly-typed with a variety of non-terminal and terminal nodes designed to move Ms. Pac–Man, make decisions, or provide information about the game. With this system, a wide variety of potential calculations can be made about the current game state. The population was evaluated by running the candidates through games of *Ms. Pac–Man* and the final scores recorded. Their results showed that allowing the candidates to be tested on longer games provided poorer results about 100 game ticks. Final results provided an 18% increase in average score over using a random default policy for Ms. Pac–Man.

Nature Inspired approaches

Nature inspired algorithms such as Ant Colony Optimisation (ACO) and other Swarm Intelligence (SI) techniques work by mimicking the tactics used by other species in nature. Often, these tactics are usable in games such as *Ms. Pac–Man*. Ants spend their lives collecting food and avoiding predators, a similar task to that faced by either Pac–Man or Ms. Pac–Man. Emilio *et al.* (2010) worked with ACO to design an agent for *Ms. Pac–Man*. Two objectives are chosen to drive the agent: maximising pill collection and minimising being eaten by ghosts. This leads to two types of ants used in the system: the *collector ants* maximising pill collecting, and the *explorer ants* minimising death. *Collector ants* also eat ghosts

if possible and include those points in their paths. At each iteration of the game the agent launches an ant of each type in all adjacent positions to the current agent position. If the agent is near a ghost then the best *explorer ant* is chosen to follow. If the agent is not near a ghost then the best *collector ant* is followed. The ants themselves follow simple rules of simply travelling to the next node, recording information about that node, listing it as visited, and then updating the local pheromones. Liberatore *et al.* (2014) look into the use of SI and flocking to control the ghost team. Flocking is inspired by bird flocks and is the blending of three strategies to control an agent: Separation, Alignment, and Cohesion. These keep the birds from colliding with each other, facing the same way, and close enough to be together. The authors consider that a GA should be able to learn the correct balance of the three strategies by playing a sequence of games. Two sets of weights are learned, one for when the ghosts are edible, and another for when the ghosts are to be avoided.

4.3 Hanabi

Hanabi[4] is a card game that has co-operative and PO characteristics. It won the prestigious *Spiel des Jahres* award 2013 for best board game of the year.

Each game consists of a team of players trying to collaboratively create stacks of cards that match in colour and increase in value consecutively. There can only be one stack per colour. There are 50 cards in the standard deck with five colours and five possible values, and example of which is given in Figure 4.5. The five colours in *Hanabi* are White, Yellow, Green, Blue, and Red. The number of cards of each value is given in Table 4.5.

The team of players can contain either two, three, four, or five players. The number of cards given to each player is five cards for two and three-player games, and four cards for four and five-player games. Each player cannot see their own cards, but can instead see the cards that the other players are holding. This is the main source of the PO in *Hanabi*. Remaining cards are placed in the deck as the last source of PO in the game.

[4] Antoine Bauza, 2010

FIGURE 4.5: An example of a card from *Hanabi*.

TABLE 4.5: Number of cards of each value in each suit of *Hanabi*.

Card Value	1	2	3	4	5
Quantity	3	2	2	2	1

The team are given eight information tokens and three life tokens.

Each player takes it in turns to make a single action of their choice. The three different actions are:

Tell Select a player and point to all their cards of a given number or suit. This costs one information token. At least one card must be identified, so the lack of a suit or number cannot be told.

Play Choose a card from the player's own hand and play it. A new card is then removed where possible from the deck and placed in the hand.

Discard Choose a card from the player's own hand and add it to the discard pile. A new card is then removed where possible from the deck and placed in the hand. Discard is only possible if the team has less than 8 information tokens remaining, and discarding earns one information token back for the team.

Playing a card doesn't require the player to know exactly what it is - if it fits anywhere on the table then it is a valid play. If the card turns out to not be playable, the team loses a life token.

The game ends when either all the life tokens are spent, the score reaches 25, or the deck runs out of cards and every player has taken their last go. When the deck runs out of cards, each player including the player that took the last card receives one additional turn before

the game ends.

The game is scored by summing the top of each stack of cards, giving a maximum score of 25 for the standard game.

4.3.1 Previous Research

Osawa introduced rule-based agents focusing on the two-player version of *Hanabi* (Osawa, 2015). Some of these agents are implemented in this chapter (Section 7.2.1). Osawa found that the consideration of other agent's strategies improved performance. Cox *et al.* generalised the five player version of *Hanabi* into a mathematical game, to create a highly capable agent (Cox *et al.*, 2015). The agent does, however, require all agents to follow the same strategy or chaos ensues. Bergh *et al.* played *Hanabi* and observed common tendencies amongst players that could be expressed as rules. An intelligent exploration of parameters and rule selection resulted in some rules showing more effectiveness than others, especially that discarding when a hint is possible is not optimal (Bergh *et al.*, 2016). This agent is implemented in Section 7.2.1. Bouzy adapted the hat-guessing strategy to also use tree search, improving scores even further (Bouzy, 2017). The authors additionally explored relaxing the rule concerning telling that a player has no cards of a given rank or suit - allowing the hat-guessing strategy to extend to less than five players.

Part II

Artificial intelligence in co-operative games with partial observability

Chapter 5

Monte-Carlo Tree Search Applied to Co-operative Problems

This chapter contributes to the thesis an experiment to see how standard Monte-Carlo Tree Search (MCTS) handles a simple co-operative problem in a Partially Observable environment, that we call "*Tiny Co-op*" (Detail: Section 4.1, Recap: Section 5.2). This environment is formed from a simple grid world with obstacles and interactive elements, as well as avatars controlled by Artificial Intelligence (AI) agents. All avatars have to each reach every goal that is present. In some cases it is necessary for the agents to behave co-operatively in order to achieve these goals. All communication between the two agents is prevented, however MCTS performs well when given enough computational time.

This chapter is adapted from:

Piers R. Williams, Joseph Walton-Rivers, Diego Perez-Liebana and Simon M. Lucas (2015). 'Monte Carlo Tree Search Applied to Co-operative Problems'. In: *CEEC'2015 - IEEE Conference on Computer Science and Electronic Engineering*. IEEE CEEC. IEEE Computer Society, pp. 219–224

5.1 Introduction

The main contribution of this chapter to the thesis is an experiment on how MCTS and Genetic Algorithm (GA) agents perform when trying to solve a simple co-operative Partial Observability (PO) problem without the ability to communicate with or observe each other.

Games that feature co-operation of some form between human players and AI agents are commonplace. Most, however, feature very limited forms of co-operation that are typically scripted such as in most First-Person Shooter (FPS) games. Where FPS games typically excel at co-operation is in online modes that enable teams of humans to play against each other. Real-Time Strategy (RTS) games also often have a small number of features designed for communication in a bid to facilitate co-operation. Two games that stand out for co-operation are *Rise of Nations*[1] and *Empire Earth II*[2]. *Rise of Nations* allowed a human and an AI agent to control the same set of units and buildings, though no communication was possible at all. This allowed a form of co-operation, but the AI operated to its own agenda, often leading to the AI spending resources that the player was trying to save for a particular item. *Empire Earth II* allowed for humans and AI agents to co-operate by letting "plans" be drawn up between them that could also be followed by both the human and AI agent. These allowed a fairly complex set of instructions to be created, despite the simple interface.

A highly popular game that had an entire mode designed for co-operation between humans was *Portal 2*[3]. This featured human sized lab-test mazes with elements that required players to work together by activating buttons, moving cubes and using intra-dimensional portals to get to the end goal. Both players were required to reach the goal in order to complete the level.

[1] Big Huge Games, 2003
[2] Mad Doc Software, 2005
[3] Valve, 2011

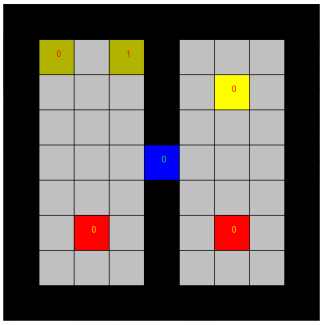


FIGURE 5.1: Example of the *Tiny Co-op* domain.

5.2 Tiny Co-op Domain

The test-game introduced in this chapter, *Tiny Co-op*, was inspired by *Portal 2*. A reminder of how the game looks is provided in Figure 5.1.

5.3 AI Agents

This section presents a set of AI agents designed to solve the problem domain. None of the agents have the ability to communicate with other agents, with no messaging protocol or the ability to observe the world either. Each avatar is controlled separately by a unique instance of an agent.

5.3.1 Random

The Random agent simply uniformly chooses one of the possible five actions (four cardinal directions and staying still). This is one of the simplest to implement and serves as a reasonable baseline for performance.

5.3.2 MCTS

This agent is a simple implementation of MCTS, with a fixed number of rollouts, tree search depth limit, and a fixed rollout depth. The rollout depth is how far in total the

forward model will be allowed to progress from the root node before the game state is evaluated. No knowledge about the game is provided and the assumption is made, both in the tree[4] and in the rollouts, that the other agent will play randomly. The fixed rollout depth makes an improvement to the number of iterations performed (from between 1-3 iterations without the fixed rollout depth to 500-600 iterations with the fixed rollout depth in 40ms [5]). This greatly improves the ability of MCTS to make informed decisions. The score at the end of a rollout is taken from the game state — so if MCTS does not see any agent reach a goal, all branches will be equal. For the experiment, three variants were created using different parameters and are listed below as well as being summarised in Table 5.1. For these agents, the total number of steps they can simulate the model forward is the addition of the depth of their tree and the depth of their rollouts.

High MCTS

This agent is designed to be fast enough to execute in real time, while having the highest budget possible. It performs 500 iterations per decision, with a tree depth of 10, and a rollout depth of 45.

Medium MCTS

This agent is designed to be more restricted than High MCTS, while still having enough budget to have some intelligence. It performs 200 iterations per decision, with a tree depth of 5, and a rollout depth of 30.

Low MCTS

This agent is designed to be a heavily restricted form of High MCTS. It performs 75 iterations per decision, with a tree depth of 3, and a rollout depth of 15.

[4] This is unusual but essentially the agent behaves as if it is playing a single player game where the other agent is a non-deterministic part of the environment.

[5] Intel Core i5-3570, 8GB RAM, Windows 7 Enterprise 64bit

TABLE 5.1: Parameters for the three MCTS agents

Budget	Rollouts	Tree Depth Limit	Rollout Depth
Small	75	3	15
Medium	200	5	30
High	500	10	45

5.3.3 Genetic Algorithms

The two GA agents are based on a Rolling Horizon Evolutionary Algorithm (RHEA) (Samothrakis and Lucas, 2010). This type of agent will evolve candidates consisting of a list of actions to execute, and use simulations to evaluate how good the list of actions is. Typically this list is of a certain length, and, therefore, presents a horizon that the agent can simulate to. The addition of macro actions is a technique used to improve the distance to the horizon without increasing the length of the candidates. This is achieved by repeating each action in the candidate solution n times (Perez *et al.*, 2013). An additional effect of this technique is that for n turns the action to make is known due to the repetitions, allowing the controller to spend n turns computing the next sequence of actions to perform. The MacroGA agent implements this behaviour.

MacroGA The MacroGA uses a population size of ten and tournament selection of size three to evolve its solutions. The population size is kept low as evaluating candidates is expensive. Each candidate consists of a string of 15 actions — with each action performed three times in a row. This means that the MacroGA can simulate 45 ticks in the future. The total length of the action sequences corresponds to a little less than the total simulation depth of High MCTS.

VariES The VariES agent is designed to solve the shortcomings[6] of the MacroGA agent. The candidate representation is extended to include individual lengths for each macro action, as well as a variable number of macro actions. This allows more complex sequences of

[6]The use of macro actions prevents MacroGA from executing single actions, leading to locations that are inaccessible. Discussed in more detail in Section 5.5

Lengths	1	4	3	2	5	1	1	2	4	3
Actions	UP	LEFT	LEFT	UP	UP	LEFT	UP	LEFT	LEFT	DOWN

Lengths	1	4	2	5	1	2	4
Actions	UP	LEFT	LEFT	UP	LEFT	LEFT	DOWN

FIGURE 5.2: Example candidate for the VariES. Candidates can vary on any of: the number of macro actions, the length of each individual macro action, and the value of each individual macro action.

TABLE 5.2: Parameters for the VariES

Parameter	**Controls**	**Value**
minNum	Number of Macro Actions	3
maxNum	Number of Macro Actions	10
minLength	Length of Macro Actions	1
maxLength	Length of Macro Actions	5
numChance	Chance of altering Number	0.25
lengthChance	Chance of altering each length	0.8
actionChance	Chance of altering each action	0.75
iterationBudget	Number of iterations of the algorithm	500

actions than the MacroGA is capable of. An example of two candidates for this algorithm are shown in Figure 5.2.

The main evolutionary technique used is also changed to a $1 + 1$ Evolutionary Strategy (ES) (Bäck *et al.*, 2000). A $1 + 1$ ES is a very simple Evolutionary Algorithm (EA) that maintains a single candidate. On each iteration, the mutation operator is applied to this individual, and it is saved as the new best candidate in case of an improvement in fitness (and discarded otherwise). This kind of ES is much simpler than a full GA, requires much less memory, and, due to a small computational cost within each iteration, is able to use more of the available time budget due to each iteration taking less time. This allows more regular checks of the time budget, and reduces the chance of overspending.

The algorithm is bounded by a number of parameters (shown in Table 5.2), and a further tuning of these is possible future work in order to explore the full potential of this algorithm. The absolute maximum simulation depth possible is 50 ticks, which is slightly lower than High MCTS and slightly higher than MacroGA. Each algorithm is fundamentally different, so maintaining exactly the same parameters across them all was impossible.

5.4 The Experiment

A round-robin tournament between the following six agents was performed:

- Random
- High MCTS
- Medium MCTS
- Low MCTS
- MacroGA
- VariES

Each agent pair played 47 games[7] on each of the seven maps[8] (Section 4.1.3). Games ended when the agents had either achieved a perfect score of 1, or reached the 2000 tick limit. Final scores and the number of ticks taken to achieve that score were recorded for analysis.

5.5 Results

The data presented in the bar-graphs comes from a single data set, *all*, and that is additionally filtered to create another dataset named *mirror*. The *mirror* dataset is formed from games where both agents were being controlled by the same type of agent. So games featuring the pair: High MCTS and Medium MCTS, would not feature in *mirror* because the two agents are different. The graphs show error bars, which are calculated as a 95% confidence interval.

Figures 5.3 and 5.4 compare the ability of each agent to solve a simple path finding problem against the same problem with the addition of the *Door* and *Buttons*. The MacroGA, with its macro actions, is hindered by its inability to make single step moves, and has trouble

[7] Number of games determined by computing budget and time available
[8] For a total of 329 games in each pair and 11,844 games across the tournament

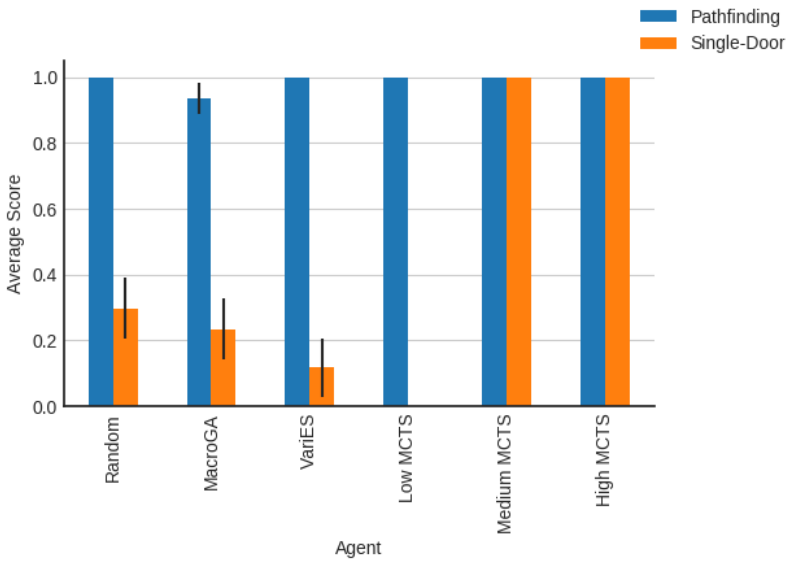


FIGURE 5.3: Average score achieved in the Pathfinding map compared with the Single Door map, on the *mirror* dataset

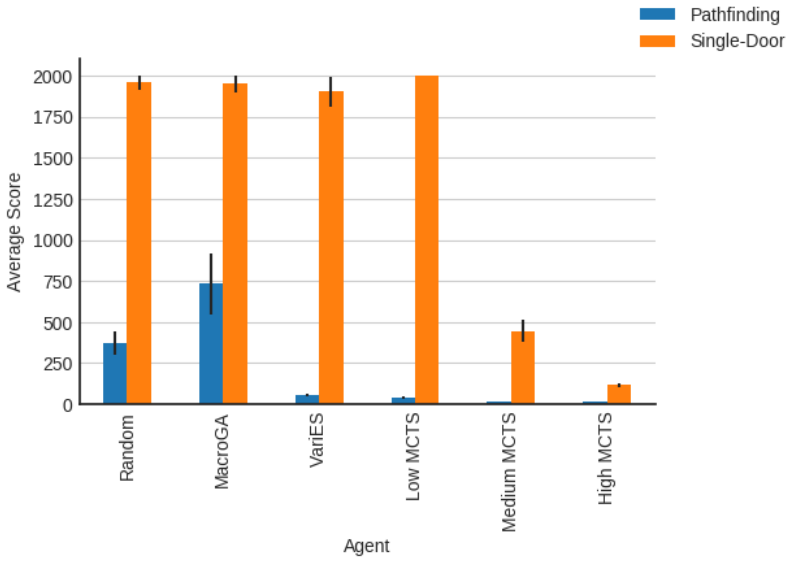


FIGURE 5.4: Average ticks taken to complete the Pathfinding map compared with the Single Door map, on the *mirror* dataset

actually travelling straight to the target. The VariES does significantly[9] better in Pathfinding, due to its ability to perform variable-length step moves, combining the advantages of macro actions with the ability to still make fine-grained moves when needed. Low MCTS is the worst performing agent in *Single Door*, however the higher budget of Medium MCTS

[9]Significance being defined as two values being outside of each others error bars

is enough to always score the maximum in *Single Door*. The High MCTS agent improves only in average ticks taken to solve the task over Medium MCTS.

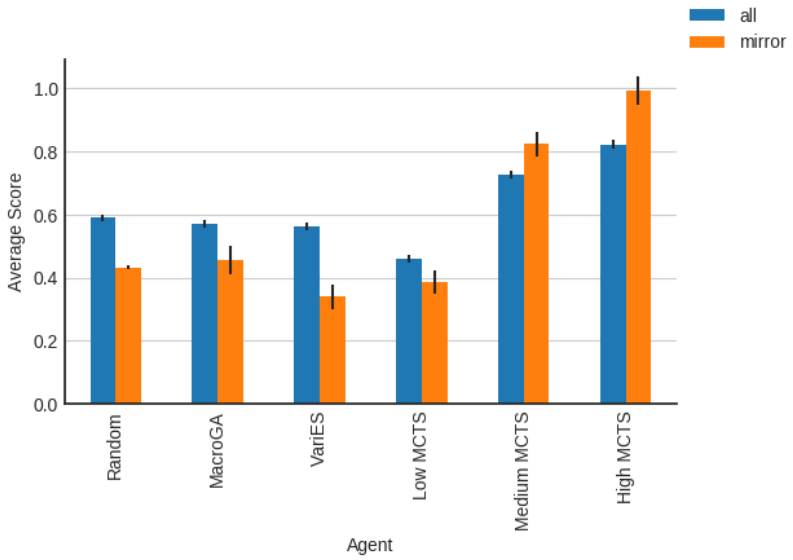


FIGURE 5.5: Average score of each AI Agent over all the maps

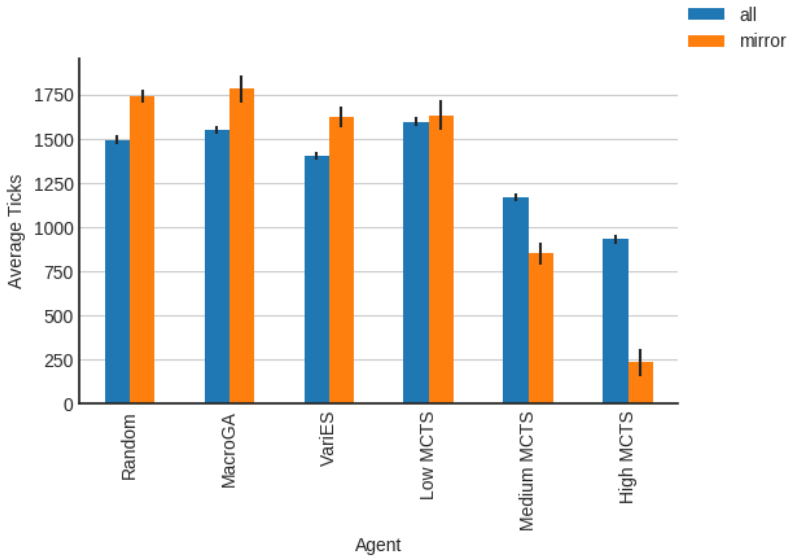


FIGURE 5.6: Average ticks taken for each AI Agent over all the maps

Figures 5.5 and 5.6 show the average score and ticks that each AI agent achieved over all the maps in both the *all* and *mirror* datasets. The two more powerful MCTS agents performed the best, doing much better than the competition. Figure 5.6 shows a similar result, with the more powerful agents typically completing the maps in fewer ticks. The

Random agent is the only deviant here — it outperformed the GA agents in score but was worse in average ticks. The good results for Random are potentially due to the fact that all three MCTS agents and both GA algorithms based their decisions on having Random as an accomplice.

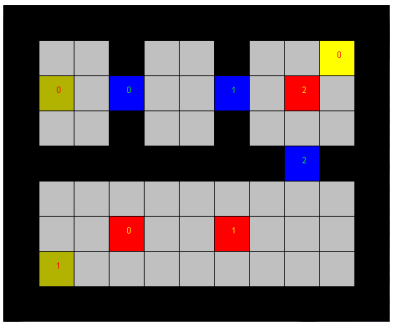


FIGURE 5.7: Recap of Airlock originally described in Section 4.1.3.

Airlock, shown in Figure 5.7, poses a particular problem for many of the agents (see Table 5.3). The asymmetric nature of the level, and the delayed reward, caused great difficulty for the group of agents. Relying on two button presses and the other agent to get through the open doors - in order - greatly reduced performance. Only the High MCTS agent scored well with an average of 0.97 on the *mirror* dataset compared to the next highest for MacroGA of 0.17.

Table 5.3 shows each agent's average score on each map with the *mirror* dataset. High MCTS shows its strengths here, with only Airlock showing a sub-perfect score. Medium MCTS matches High MCTS on 4 maps, showed some trouble with Butterfly and Extended Side, as well as doing terribly on Airlock. Low MCTS shows where its poor overall score has come from, not posting an average score at all on two maps[10].

[10]Implying the agent spent the full 2,000 ticks without reaching a single goal

TABLE 5.3: Average score for each AI Agent over each map on the *mirror* dataset

Map	Agent	Score	StdDev	Ticks	StdDev
Pathfinding					
	Random	1.00	0.00	374.21	249.09
	High MCTS	1.00	0.00	16.06	4.32
	Medium MCTS	1.00	0.00	17.49	4.62
	Low MCTS	1.00	0.00	39.91	23.41
	MacroGA	0.94	0.17	732.98	646.18
	VariES	1.00	0.00	53.77	35.45
Side By Side					
	Random	0.64	0.15	1969.15	148.05
	High MCTS	1.00	0.00	107.30	42.93
	Medium MCTS	1.00	0.00	366.02	226.03
	Low MCTS	0.58	0.12	2000.00	0.00
	MacroGA	0.61	0.12	2000.00	0.00
	VariES	0.62	0.21	1820.32	408.39
Symmetric Single Door					
	Random	0.32	0.35	1888.98	336.25
	High MCTS	1.00	0.00	34.70	12.85
	Medium MCTS	1.00	0.00	119.38	61.76
	Low MCTS	0.86	0.29	1390.74	544.40
	MacroGA	0.54	0.33	1866.60	330.76
	VariES	0.40	0.50	1594.06	611.65
Butterfly					
	Random	0.39	0.16	2000.00	0.00
	High MCTS	1.00	0.00	409.23	167.67
	Medium MCTS	0.79	0.19	1786.40	167.67
	Low MCTS	0.25	0.00	2000.00	0.00
	MacroGA	0.39	0.17	2000.00	0.00
	VariES	0.13	0.14	2000.00	0.00
Single Door					
	Random	0.30	0.32	1959.11	148.59
	High MCTS	1.00	0.00	115.79	186.58
	Medium MCTS	1.00	0.00	445.15	228.49
	Low MCTS	0.00	0.00	2000.00	0.00
	MacroGA	0.23	0.33	1950.21	186.58
	VariES	0.12	0.32	1902.53	317.58
Extended Side					
	Random	0.30	0.29	1990.51	49.38
	High MCTS	1.00	0.00	234.70	127.81
	Medium MCTS	0.88	0.21	1254.77	589.86
	Low MCTS	0.00	0.00	2000.00	0.00
	MacroGA	0.31	0.25	1984.17	108.52
	VariES	0.10	0.19	2000.00	0.00
Airlock					
	Random	0.09	0.19	2000.00	0.00
	High MCTS	0.97	0.16	714.57	516.67
	Medium MCTS	0.11	0.31	1958.47	147.85
	Low MCTS	0.00	0.00	2000.00	0.00
	MacroGA	0.17	0.28	1953.60	261.12
	VariES	0.00	0.00	2000.00	0.00

5.6 Discussion

The following subsections discuss the performance of different AI's (Sections 5.6.1 to 5.6.3) in detail. Map complexity is discussed in Section 5.6.4

5.6.1 Random

The Random agent performed poorly in this problem domain, finishing on average under the 2000 tick limit and scoring less than half the maximum.

5.6.2 MCTS

The MCTS agents performed very well in this problem domain. As seen above, in Figure 5.6, the Medium MCTS and High MCTS agents provided the two quickest completion times across all maps. The differences shown were significant, as well as being the only two agents to have completed the levels in under 1000 ticks on average. Figure 5.5 showed that the medium and high budget implementations scored significantly better than all other agents. The Low MCTS agent performed poorly, scoring worse than the Random agent - showing that there simply was not enough computational budget to perform well. This is likely due to Random at least exploring the environment with a random chance of doing the right moves, while the Low MCTS agent may simply spend a long time staying still instead of going anywhere.

One possible reason for MCTS scoring so well in this problem domain is its use of statistics over hundreds and thousands of simulations to provide it with the ability to act in such a way that it handles all eventualities. Statistically, in certain situations MCTS would only see rewards in the tree when it was situated on a *Button*. This tended to cause the MCTS agent to travel towards *Buttons*, increasing the possibility that its own simulations would cause it to be situated on the button. Eventually, most other AI agents would cross through the open door.

5.6.3 GA

The two GA algorithms did not perform very well in this problem domain. As seen in Figure 5.5 and Figure 5.6, when tasked with solving the problem domain with another identical agent, neither the MacroGA or VariES performed well at all. The VariES only performed significantly better than either Random or MacroGA in Pathfinding. In all other cases, VariES performed similarly to the standard MacroGA. The ability to mutate the lengths of individual action sequences made the VariES a more flexible pathfinder that the MacroGA but did not aid its ability to solve the co-operative problems present elsewhere in the experiment.

The GA algorithms do not have the stored tree structure of MCTS with which to gain the statistical model for what happens when they pursue certain actions. This leads to a seemingly poor performance for the GA despite GAs and MCTS typically performing equally well in other domains (Perez *et al.*, 2013).

5.6.4 The Maps

The agents performed across the maps as expected - with maps containing goals behind interactive elements proving the hardest. Airlock in particular was interesting with the asymmetric roles designed to make the second agent open and close the doors in order to allow the first agent to progress to the goal. Figure 5.8 shows each agent on the *mirror* dataset, as well as the average score when the other agent is High MCTS in either of the possible positions. It is clear that the presence of High MCTS is beneficial in both cases, but it is particularly good for team performance when High MCTS is in the crucial second position.

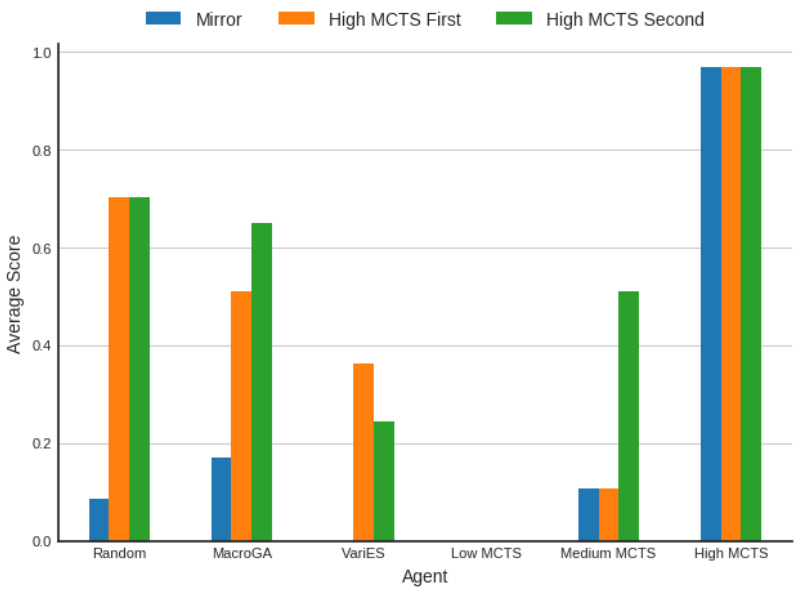


FIGURE 5.8: Analysis of Airlock, showing each agent with High MCTS in different positions as well as the *mirror* data set. The first player has a different role in this map, providing the difference between the scores. The second player has to use *buttons* to open *doors* for the first player. The first player is powerless unless this is done for them.

5.7 Conclusions

In this chapter, we found that a strong MCTS agent can solve simple cooperative PO problems without requiring communication between agents. We also found that a number of other AI techniques experienced difficulty when the problem became cooperative. We hypothesised that MCTS's use of a stored tree guiding its explorations (something that GAs lack) was a major advantage in solving problems that rely on the other agent. GAs were found to perform poorly in the experiment, despite being capable of finding the goal without the co-operative obstacles.

Chapter 6

The Ms. Pac-Man Vs Ghost Team Competition

This chapter introduces the revival of the popular Ms. Pac–ManVersus Ghost Team competition. An updated game engine with Partial Observability (PO) constraints is presented as well as a new Multi-Agent System (MAS) approach to developing ghost agents, and several sample controllers to ease the development of entries. A restricted communication protocol is provided for the ghosts, providing a more challenging environment than before but also allowing them to overcome some of the PO constraints. The addition of PO to the co-operative ghost-team environment is the primary contribution of this chapter to the thesis. Some preliminary results showing the effects of PO and the benefits of simple communication are also presented. In addition to this, results from the competition are presented.

This chapter is adapted from:

Piers R Williams, Diego Perez-Liebana and Simon M Lucas (2016). 'Ms. Pac-Man Versus Ghost Team CIG 2016 Competition'. In: *CIG'2016 - IEEE Conference on Computational Intelligence and Games*. IEEE CIG, pp. 420–427.

6.1 Introduction

Ms. Pac–Man is an arcade game that was immensely popular when released in 1982. An improvement on the original *Pac–Man* game; *Ms. Pac–Man* added better graphics, additional mazes, and new Artificial Intelligence (AI) behaviour for the ghosts. The primary difference that interests academics and researchers is the addition of non-deterministic ghost AI which vastly increased the challenge in creating an effective agent for *Ms. Pac–Man*.

Ms. Pac–Man has been the focus of two competitions in the past: The Ms. Pac–Man screen capture competition and the Ms. Pac–Man Vs Ghost Team competition. The Ms. Pac–Man screen capture competition periodically provided agents with a pixel map of the game and requested the direction of travel from the agent. This competition only allowed the entrants to submit agents for the Ms. Pac–Man character. The Ms. Pac–Man Vs Ghost Team competition was based on a simulator that mimicked the original game reasonably closely and provided agents with an Application Programming Interface (API) to interact with. Entrants had to submit a controller for either the Ms. Pac–Man agent or the ghost team. The ghost team controller would return 4 actions, one per ghost.

The new Ms. Pac–Man Vs Ghost team competition adds PO to *Ms. Pac–Man*. PO greatly increases the challenge in creating good AI controllers. Limited information about the ghosts makes it more difficult for Ms. Pac–Man to plan effectively. Limited information about Ms. Pac–Man forces the ghosts to search and communicate effectively in order to trap Ms. Pac–Man. In addition to the PO constraints, the competition now expects the ghosts to be controlled individually by a team of agents. Further review of competitions was provided in Section 2.4.

6.2 The Competition

This section will describe the implemented AI controllers for both PO and non-PO operation (Section 6.2.1), an initial experiment comparing the effect of PO on score (Sections 6.2.2

and 6.2.3), and the way the competition will be organised (Sections 6.2.4 and 6.2.5).

6.2.1 Sample Controllers for Ms. Pac–Man vs Ghosts

Having implemented PO for both Ms. Pac–Man and the ghosts, new controllers were needed as examples for people entering the competition. These new controllers, and those used in the experiments in this chapter, are described here.

StarterPacman (COP)

This is the original basic controller for the previous competition and works only in Completely Observable environments. This controller follows a very basic algorithm with some simple sequential rules as shown in Algorithm 1. The controller will avoid ghosts that are too close, chase ghosts that are edible, or travel to the nearest pill.

Algorithm 1 StarterPacman basic algorithm

```
function GETMOVE()
    limit ← 20
    nearestGhost ← GETNEARESTCHASINGGHOST(limit)
    if nearestGhost then
        return NEXTMOVEAWAYFROM(nearestGhost)
    end if
    nearestGhost ← GETNEARESTEDIBLEGHOST(limit)
    if nearestGhost then
        return NEXTMOVETOWARDS(nearestGhost)
    end if
    nearestPill ← GETNEARESTPILL()
    return NEXTMOVETOWARDS(nearestPill)
end function
```

StarterGhosts (COG)

This is the original basic controller for the previous competition to control the four ghosts. It is a "puppet-master" style algorithm, meaning it is a single block of logic that generates moves for all four of the ghosts. The controller follows some basic strategies if a ghost is allowed to make a move as shown in Algorithm 2. The ghosts will run away from Ms.

Pac–Man if she is able to eat the ghost, or near a power pill (Potential to eat ghost). If the previous rule doesn't apply then the ghost will 90% of the time chase Ms. Pac–Man and 10% of the time move randomly.

Algorithm 2 StarterGhosts basic algorithm

```
function GETMOVE()
    pacman ← GETPACMANINDEX()
    if ISEDIBLE() OR PACMANCLOSETOPPILL() then
        return NEXTMOVEAWAYFROM(pacman)
    end if
    if NEXTFLOAT < 0.9 then
        return NEXTMOVETOWARDS(pacman)
    else
        return NEXTRANDOMMOVE()
    end if
end function
```

POPacman (POP)

This is a modification of the StarterPac–Man where each strategy is followed if it is possible as shown in Algorithm 3.

Algorithm 3 POPacman basic algorithm

```
function GETMOVE()
    limit ← 20
    nearestGhost ← GETNEARESTCHASINGGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVEAWAYFROM(nearestGhost)
    end if
    nearestGhost ← GETNEARESTEDIBLEGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVETOWARDS(nearestGhost)
    end if
    nearestPill ← GETNEARESTPILL()
    if nearestPill ≠ NULL then
        return NEXTMOVETOWARDS(nearestPill)
    end if
    return NEXTRANDOMMOVE()
end function
```

Other than modifying the original strategies with guards against null, it was clear that a new default strategy was needed. This is because within the PO game, it was possible

to proceed through the previous strategies without returning a move. This new default strategy was to simply return a random move.

Starter Pacman One Junction

This is a starter agent added to provide a demonstration of determinising the game state and forwarding the resulting state to provide a basic one junction lookahead (Algorithm 4). The state is forwarded to the next junction rather than a single step of the game as this was considered to be too small a distance between decisions.

Algorithm 4 Starter Pacman One Junction algorithm

```
function GETMOVE()
    bestScore ← -1
    for move : MOVES do
        score ← EVALUATEJUNCTIONINDIRECTION(move)
        if score > bestScore then bestScore ← score bestMove ← move
        end if
    end for
    return bestMove
end function
```

POGhosts (POG)

This is a modification of the StarterGhosts where each strategy is followed if it is possible in the PO case. If there is no information available to the ghost, then the ghost will behave randomly at intersections as shown in Algorithm 5.

POCommGhosts (POGC)

This is a modification of the POGhosts that attempts to communicate each tick in order to improve its chances. If this ghost can see Ms. Pac–Man then it will send a message to everyone else. If it can't see Ms. Pac–Man then it will check if anybody else has seen it. If someone else has seen Ms. Pac–Man then it pretends it can see Ms. Pac–Man and follows the original POGhosts strategy outlined above. The pseudo code for this is shown in Algorithm 6.

Algorithm 5 POGhosts basic algorithm

```
function GETMOVE()
    pacman ← GETPACMANINDEX()
    if pacman then
        if ISEDIBLE() OR ISPACMANCLOSETOPOWERPILL() then
            return NEXTMOVEAWAYFROM(pacman)
        end if
        if NEXTFLOAT < 0.9 then
            return NEXTMOVETOWARDS(pacman)
        end if
    else
        return NEXTRANDOMMOVE()
    end if
end function
```

Algorithm 6 POCommGhosts basic algorithm

```
function GETMOVE()
    if PACMANINFONEEDSRESET( GETCURRENTTICK() ) then RESETPACMANINFO()
    end if
    pacman ← UPDATEPACMANLOCATION()
     HANDLEMESSAGES()
    pacman ← GETPACMANINDEX()
    if pacman ≠ NULL then
        if ISEDIBLE() OR PACMANCLOSETOPPILL() then
            return NEXTMOVEAWAYFROM(pacman)
        end if
        if NEXTFLOAT() < 0.9 then
            return NEXTMOVETOWARDS(pacman)
        end if
    else
        return NEXTRANDOMMOVE()
    end if
end function
```

The threshold used to determine when to forget Ms. Pac–Man's location needs tuning. Every value from 0 to 200 was put to a test on 4000 games against the COP agent and 33,300 games against the POP agents. The results are displayed in Figure 6.1 and show that the value of 50 is a good value against these two agents. Interestingly the data against the POP algorithm is significantly noisier than COP. This is presumably due to COP being deterministic and POP being non-deterministic.

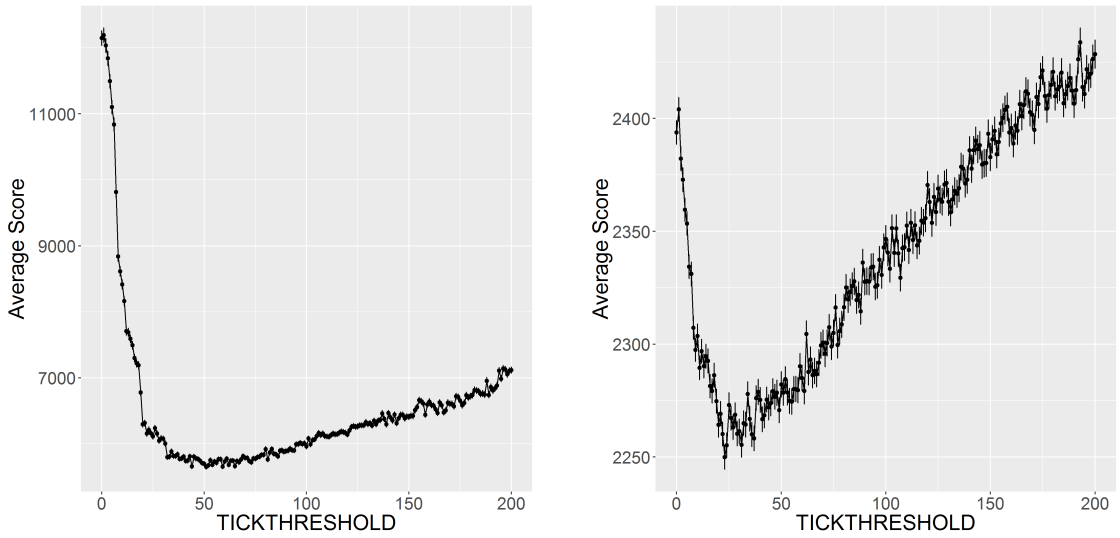


FIGURE 6.1: Tuning results of POGC against COP(Left) and POP(Right) both with error bars.

6.2.2 Sample Controller Experiment

A round-robin of games between the two Ms. Pac–Man agents (COP, POP) against the three ghost teams (COG, POG, POGC) was run with 1000 repeats for each pairing. The Complete Observability (CO) agents were given the full view of the environment, while the PO agents were given the restricted view of the environment. These basic controllers may not represent the best agents for the game, but they do provide simple comparisons between them as they rely on the same strategies. The only difference between them is the addition of PO. Where PO was enforced, it was the Radius restriction with a sight limit of 50.

6.2.3 Sample Controller Results

The results of the experiment are presented in Table 6.1. It is clear that for the same strategies, PO is a large handicap to the agent. Against COG, adding PO to Ms. Pac–Man caused the score to drop from 3,895.67 to 1,753.52. Adding communication abilities to the PO ghosts allowed CO Ms. Pac–Man to achieve only 3,895.67 points on average compared to 17,257.24 points when the ghosts couldn't communicate. This is a huge difference

between two very simple algorithms and clearly shows the benefits of communication in this scenario.

TABLE 6.1: Table of results after 1000 runs of different controllers

Agents	Mean Score	Std. Error
COP Vs COG	3895.67	48.23
COP Vs POG	17257.24	280.49
COP Vs POGC	5769.30	77.41
POP Vs COG	1753.52	26.97
POP Vs POG	2708.15	37.98
POP Vs POGC	2349.34	30.32

6.2.4 Competition Tracks

The Ms. Pac–Man Vs Ghost Team competition featured two main tracks. The first track allowed participants to submit code to control Ms. Pac–Man operating within PO constraints. The second track allowed participants to submit 4 controllers, one for each ghost, that would operate under PO constraints.

6.2.5 Entrant Ranking

Entrants were evaluated regularly and had Glicko2 ratings calculated while the competition was still open for entry. At the close of the competition, the top n agents were evaluated in a round robin tournament. The value of n was chosen based on available computational hardware. In the event that the Glicko2 ratings were used to select entry into the round robin, the ratings were given time to settle after the close of entry in case some submissions were recently updated and had changed skill levels.

6.3 Competition Results

This section will discuss the results obtained from running the Ms. Pac–Man Vs Ghost Team competition. The competition ran at CIG[1] in 2016 (Section 6.3.1), 2017 (Section 6.3.2),

[1]IEEE conference on Computational Intelligence in Games

and 2018 (Section 6.3.3).

6.3.1 2016

There were two entrants to the Ms. Pac–Man track, but unfortunately no entrants to the ghost team track in 2016. Both entrants performed significantly better than the sample agents and scored less than 500 points between them. Full results are displayed in Table 6.2.

TABLE 6.2: Results from the 2016 competition

Agent	Average	Minimum	Maximum
GiangCao	6348.85	1000	15940
dalhousie	5878.50	1140	13620
POP	2447.75	580	6730
Random	1629.85	250	5830

6.3.2 2017

In 2017 there was a large number of entrants to both the Ms. Pac–Man track and the ghost-team track. 29 Ms. Pac–Man agents[2], and 18 ghost teams successfully competed[3]. Truncated results are presented in Table 6.3. The results here are not comparable with the previous year's results due to the co-evaluation of agents but there is a healthy difference between agents in both tracks. Further analysis of the performance of these agents is performed in Chapter 8.

TABLE 6.3: Top 5 from each track in 2017. Full results available at http://www.pacmanvghosts.co.uk/results.html

Ms. Pac–Man Agent	Average Score	Ghost Agent	Average Score
SubtleBattle	10 260.38	MaFr	2223.86
giangrocker	9147.60	TiIsFePre	2853.79
thunder	8861.12	thunder	3047.42
ToSc	8388.80	POGC	3243.27
BaHe	8245.39	NiStTiTi	3276.37

[2]Only three of which were the sample agents
[3]Only two of which were the sample agents

6.3.3 2018

In 2018 there were less entrants, however a new entrant successfully took the top spot of the Pac–Man track. The ghost track was a little disappointing, with the supplied starter agents taking the top two spots. Full results are displayed in Table 6.4.

TABLE 6.4: Results from each track in 2018.

Ms. Pac–Man Agent	Average Score	Ghost Agent	Average Score
Squillyprice01	7736.63	StarterGhostComm	3859.13
GiangCao	7516.63	StarterGhost	4288.25
thunder	6733.13	thunder	4864.81
PacMaas	6275	user76	4948.88
StarterPacMan	5865.5		
StarterPacMan One-Junction	1134.25		
StarterNNPacMan	535		
user76	120		

6.4 External Research

The original paper (Williams *et al.*, 2016) and framework have been used in external research. Garduño Hernández (2017) combined a Genetic Algorithm (GA) with case based reasoning and found a high performing ghost team for the framework. Domınguez-Estévez *et al.* (2017) used Q-learning with the Q-table replaced with case based reasoning, and found that they could either survive in the game for a long time or reach high scores, but not both in the same agent. Dockhorn and Kruse (2017) used coevolution and Genetic Programming (GP) to create solutions for both the ghost team and Ms. Pac–Man with success creating a diverse set of agents. Dienstknecht (2018) used a Deep Neural Networks (DNN) merged with Monte-Carlo Tree Search (MCTS), similar to Alpha Go, and found that DNN can improve MCTS in *Ms. Pac–Man*. The agent was submitted to the 2018 competition as PacMaas. Zhang *et al.* (2018) used a new GP approach to evolve behavioural trees to play Ms. Pac–Man and found that it was superior to prior GP techniques for behavioural trees.

6.5 Conclusions

This chapter introduced the Ms. Pac–Man Vs Ghost Team competition, describing the new rules in detail as well as reviewing past research into Ms. Pac–Man AI. The addition of PO to an already co-operative environment (ghost-team) and incentivising research into this problem through a competition is the primary contribution towards this thesis. The chapter also presents and discusses an initial experiment with basic AI conducted under the new constraints. Finally the competition results so far are presented, showing the amount of interest the competition has had.

Chapter 7

Evaluating and Modelling Hanabi-Playing Agents

In this chapter, the use of agent modelling in the hidden-information, collaborative card game *Hanabi* (Section 4.3) is explored. Agent modelling involves considering how other agents will behave, in order to influence an agent's own actions. A number of rule-based agents, both from the literature and of our own devising, in addition to an Information-Set Monte-Carlo Tree Search (IS-MCTS) (Section 3.1.3) agent are implemented. Poor results are observed from IS-MCTS, so a new, predictor version that uses a model of the agents with which it is paired is constructed. A significant improvement in game-playing strength is observed from this agent in comparison to IS-MCTS, resulting from its consideration of what the other agents in a game would do. In addition, an intentionally flawed rule-based agent is created to highlight the predictor's capabilities with such an agent compared with others.

The bulk of this work is adapted from:

Joseph Walton-Rivers, Piers R Williams, Richard Bartle, Diego Perez-Liebana and Simon M Lucas (2017). 'Evaluating and Modelling Hanabi-Playing Agents'. In: *Congress on Evolutionary Computation, 2017. CEC'17. IEEE Conference On*. IEEE, pp. 1382–1389.

7.1 Introduction

The purpose of this work is to examine the effect of being able to predict teammate actions in a co-operative Partially Observable game. *Hanabi* is a good game for research into agent modelling because players are explicitly restricted in how they can communicate with their teammates, leaving modelling of what they believe that other agents will do as the only option for predicting their behaviour. *Hanabi* has attracted interesting research in recent times due to the challenges of creating Artificial Intelligence (AI) that can play effectively (Section 4.3.1).

Hanabi disfavours greedy play - choosing to play a card instead of providing a hint can be worse than saving the play action for later and using a tell action this turn. If the next player is likely to throw away a useful card unless you inform them of some of its attributes it would be a poor choice to play a card even though it would obtain a point for the team. In addition, choosing to tell could enable multiple players to play cards. For example, Table 7.1 demonstrates a situation where it is far better for P1 to inform P2 of their 2's instead of playing the red two. The + and - symbols indicate if a player knows or doesn't know the colour and value of a card. P1 knows they have a Red 2 and P3 knows they have a Blue but not the value of the Blue 1 in their hand. This allows a sequence of play actions (P2 play B2, P3 play B3, P4 play B4, P1 play R2) which is far better than the greedy action of (P1 play R2).

TABLE 7.1: Scenario demonstrating value of non greedy play.

Table	B1	R1	G0	Y0	W2
P1	?	?	?	R2 + +	
P2	W4 - -	B2 + -	W4 - -	Y2 - -	
P3	Y2 - -	B3 + +	B1 + -	R1 - -	
P4	B4 + +	B1 + -	W1 - -	G2 - -	

7.2 AI

This section will provide descriptions for how the different agents operate and are implemented in *Hanabi*, split into two main categories. Section 7.2.1 describes the use of Production Rule Agents while Section 7.2.2 describes the remaining agents including IS-MCTS based approaches.

7.2.1 Production Rule Agents

A number of the AI agents implemented in this framework operate as a Production Rule Agent (PRA). A PRA operates by:

1. Starting with an ordered list of rules
2. Consulting the next rule from the list on what to do in the current situation
3. If the rule provided an action, perform the action
4. Otherwise move to the next rule from the list and repeat from step 2.

These rules are summarised by Figure 7.1.

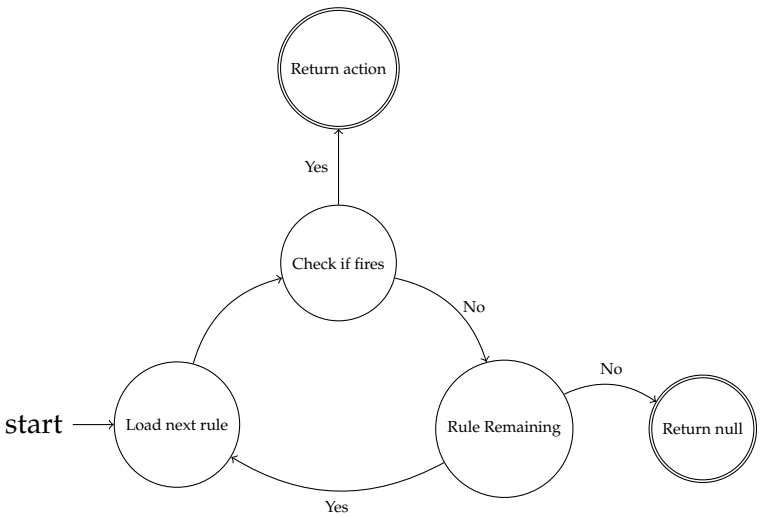


FIGURE 7.1: The operation of a Production Rule Agent

A rule in its most basic form is a mapping from a game state to either an action or null. Rules are kept simple, focusing on simpler tasks than playing the whole game, forming partial AI units. When a rule returns an action it is indicating that that is what that rule has determined should be performed at the time. When the rule returns null then that is the rule determining that it cannot make a decision. It usually requires a number of rules to create an agent, due to the specialist nature of each rule. Many agents use the same rules, so a description of each rule is presented once here before discussing individual agents:

- **PlaySafeCard:** Plays a card only if it is guaranteed that it is playable. Playable means that the agent is able to determine with certainty that the card is safe to play, whether or not that player can determine exactly what that card is.
- **OsawaDiscard:** Discards a card if it cannot be played at the end of the turn. This will discard cards that we know enough about to disqualify them from being playable. For example, a card with an unknown suit but a rank of 1 will not be playable if all the stacks have been started. This rule also considers cards that cannot be played because their pre-requisite cards have already been discarded.
- **TellPlayableCard:** Tells the next player a random fact about any playable card in their hand.
- **TellRandomly:** Tells the next player a random fact about any card in their hand.
- **DiscardRandomly:** Randomly discards a card from the hand.
- **TellPlayableCardOuter:** Tells the next player an unknown (to that player) fact about any playable card in their hand.
- **TellUnknown:** Tells the next player an unknown fact about any card in their hand.
- **PlayIfCertain:** Plays a card if we are certain about which card it is and that it is playable.
- **DiscardOldestFirst:** Discards the card that has been held in the hand the longest amount of time.

- **IfRule(λ) Then (Rule) [Else (Rule)]:** Takes a Boolean λ expression and either one or two rules. The first rule will be used if the λ evaluates to true. If it is false, and a second rule was provided, then that rule will be used instead.
- **PlayProbablySafeCard(**$Threshold \in [0, 1]$**):** Plays the card that is the most likely to be playable if its playability is at least as probable as *Threshold*.
- **DiscardProbablyUselessCard(**$Threshold \in [0, 1]$**):** Discards the card that is most likely to be useless if its playability is less probable than *Threshold*.
- **TellMostInformation(**$New? \in [True, False]$**):** Tells whatever reveals the most information, whether this is the most information in total or the most new information.
- **TellDispensable:** Tells the next player with an unknown dispensible card the information needed to correctly identify that the card is dispensible. This rule will only target cards that can be identified to the holder as dispensible with the addition of a single piece of information.
- **TellAnyoneAboutUsefulCard:** Tells the next player with a useful card either the remaining unknown suit of the card, or the remaining unknown rank of the card.
- **TellAnyoneAboutUselessCard:** Tells the next player with a useless card either the remaining unknown suit of the card, or the remaining unknown rank of the card.

The following agents will now be introduced: Internal, Outer, Cautious, IGGI, Piers, Flawed, and Bergh Rule.

Internal

This is a faithful implementation of the agent presented by Osawa that shares the same name (Osawa, 2015). It features memory of the information it has been told about its own hand but does not remember information about what other players have been told. The rules used in order are:

1. PlaySafeCard

2. OsawaDiscard
3. TellPlayableCard
4. TellRandomly
5. DiscardRandomly

Outer

This is an implementation of the agent presented by Osawa with the same name (Osawa, 2015). It features knowledge of what the other agents have been told already as well as what it has been told, to avoid repeating tell actions. The rules used in order are:

1. PlaySafeCard
2. OsawaDiscard
3. TellPlayableCardOuter
4. TellUnknown
5. DiscardRandomly

Cautious

This is an agent derived from human gameplay. The agent plays cautiously, designed to never lose a life. The rules used in order are:

1. PlayIfCertain
2. PlaySafeCard
3. TellAnyoneAboutUsefulCard
4. OsawaDiscard
5. DiscardRandomly

IGGI

This agent is a modification of Cautious. The alteration to a deterministic discard function greatly aids the predictability of this player. The rules used in order are:

1. PlayIfCertain
2. PlaySafeCard
3. TellAnyoneAboutUsefulCard
4. OsawaDiscard
5. DiscardOldestFirst

Piers

This is an agent designed to use IfRules to improve the overall score. Otherwise, it is similar to IGGI. The rules used in order are:

1. IfRule (lives $>$ 1 $\wedge$ $\neg$deck.hasCardsLeft) Then (PlayProbablySafeCard(0.0))
2. PlaySafeCard
3. IfRule (lives $>$ 1) Then (PlayProbablySafeCard(0.6))
4. TellAnyoneAboutUsefulCard
5. IfRule (information $<$ 4) Then (TellDispensable)
6. OsawaDiscard
7. DiscardOldestFirst
8. TellRandomly
9. DiscardRandomly

The first IfRule is designed as for exclusive use in the end game: if there is nothing left to lose, try to gain a point. This derives from human play, where typically during the end

game we make random plays if we know there is a playable card somewhere in our hand. This rule is more accurate, as it uses all the information it has gathered during play to calculate probabilities.

The second IfRule simply risks playing a card if there is a reasonable chance of its being safe.

The third IfRule is designed to try to provide more intelligent tell actions. If there is nothing useful to tell and we are low on information, we set another agent up to be able to discard cards that are not needed. This means that the agents can burn through cards that are not helpful so as to try to obtain useful cards from the deck.

Flawed

This is an agent designed to be intelligent but with some flaws: it does not possess intelligent tell rules, and has a risky play rule as well. Understanding this agent is the key to playing well with it, because other agents can give it the information it needs to prevent it from playing poorly. The rules used in order are:

1. PlaySafeCard
2. PlayProbablySafeCard(0.25)
3. TellRandomly
4. OsawaDiscard
5. DiscardOldestFirst
6. DiscardRandomly

Giving information is the key to getting this agent to work intelligently. Without information, the intelligent rules can't fire, thereby leaving this agent to tell randomly and discard randomly which is not a great strategy.

Bergh Rule

This is the best rule-based agent created by Bergh *et al.* It was created by observing from human play that there are four main tasks (Bergh *et al.*, 2016):

1. If I'm certain enough that a card is playable, play it.
2. If I'm certain enough that a card is useless, discard it.
3. Give a hint if possible.
4. Discard a card.

Bergh *et al.* used a Genetic Algorithm (GA) to evolve the best options for each section, resulting in the following rules as an implementation:

1. IfRule (lives $>$ 1) Then (PlayProbablySafeCard(.6)) Else (PlaySafeCard)
2. DiscardProbablyUselessCard(1.0)
3. TellAnyoneAboutUsefulCard
4. TellAnyoneAboutUselessCard
5. TellMostInformation
6. DiscardProbablyUselessCard(0.0)

7.2.2 Other Agents

These agents are implemented without the use of a PRA and include the tree search agents.

Legal Random

This agent makes a move at random from the set of legal actions available to it at any given time step.

MCS

This agent is a simple Monte Carlo Search (MCS) that uses a provided agent for the rollout phase. MCS is a technique that uses the Upper Confidence Bound (UCB) equation to select actions in a single step lookahead, with policy informed rollouts to evaluate those positions. It is essentially Monte-Carlo Tree Search (MCTS) with a tree depth limit of one turn. In this chapter, we name the agent MCS-[agent] to indicate which agent provided the rollout policy. For example, a MCS agent using IGGI as a rollout policy would be named MCS-IGGI. The agent has a one-second time limit to return a move.

IS-MCTS

This agent uses a MCTS technique for handling games with partial observability known as IS-MCTS (Cowling *et al.*, 2012).

IS-MCTS is a modification to MCTS in which, on each individual iteration of the algorithm, the partially-observable game state is determinised into one of the possible fully-observable states. This state remains consistent for the selection, expansion, rollout, and back-propagation phases before being replaced by a new determinisation in the next iteration. The implementation uses a time limit for returning moves of one second per move, and achieves between 30,000 and 60,000 iterations in that limit depending on the game state.

Predictor IS-MCTS

This agent was provided with a copy of each of the agents that it was paired with to use in its prediction. The predicted agents were initialised with random seeds: this corresponds to the predictor's having knowledge of each agent's overall strategy but having no knowledge of its internal workings.

The Predictor IS-MCTS agent modifies the selection, expansion, and rollout phases of MCTS when considering nodes for other agent turns. The modifications remove the original selection policy for other agents' turns and replaces it with a query to the agent's model to discover what that agent would do in that situation. The rollout phase is similarly modified for the agent's turns to use the model, while the predictor agent uses random for its own turns in the rollouts. When making moves for its own turn during the rollout, the predictor agent defaults to the legal random selection method used by IS-MCTS. The implementation maintains the one-second-per-move limit of IS-MCTS.

7.3 Method

This section will describe the method behind the two experiments (Sections 7.3.1 and 7.3.2).

7.3.1 Validation

In order to first validate our framework and AI implementations, we performed experiments using re-implementations of the Osawa and Van den Bergh agents. This involved recreating the experiments that they described in their papers and checking that we obtained similar results.

7.3.2 Full Test

The set of agents under test contained a mix of current research on *Hanabi* as well as some rule-based agents of our own. Not all of the possible MCS algorithms were tested due to computational budgets[1] There is also a mix of strong and poor agents for balance. We tested all of the agents from this list:

- Legal Random

[1]We also at the time didn't know Piers would outperform IGGI. The testing then revealed that MCS barely added anything to the algorithm it was based on, leaving no reason to re-run the experiment with MCS-Piers

- Outer
- IGGI
- Piers
- Flawed
- Bergh Rule
- MCS-Legal Random
- MCS-IGGI
- MCS-Flawed
- IS-MCTS
- Predictor IS-MCTS

In each experiment, one of the agent was selected from the list above and the remaining agents were selected as a group from the list below. For example, in the first experiment the Legal Random agent would be alone with four IGGI agents — a concept we call *pairing*. The agents above were all paired in turn with:

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

200 random seeds were chosen, and for each seed every agent under test played two games with every agent with which it was paired. Each pairing did this for standard *Hanabi* rules

with 2, 3, 4, and 5 players. Each agent under test played from a randomised position (first, second, third, fourth, or fifth) determined by the seed. This ensured that each agent under test was in the same position for the same seed. Every agent therefore played 200 (number of unique seeds) * 4 (2, 3, 4, or 5 player games) * 7 (Number of unique paired agents) * 2 (number of reruns) = 11200 games.

The configuration, final score, and other basic state information were logged to a file upon completion of the game. The results were collated per agent and the mean score and number of turns taken were calculated. Additional information was also stored about the final state of each game including the number of lives remaining and the information tokens remaining. When there were no lives remaining at the end of the game, this indicated that the game ended because the players ran out of life tokens.

The full (human readable) game traces for each game were also stored, for evaluating agent behaviour and the effectiveness of strategies.

7.4 Results

Results from the individual tests are given in this section. Section 7.4.1 presents the results from the Validation test, while Section 7.4.2 presents the results from the Full Test test.

7.4.1 Validation

TABLE 7.2: Results of the validation tests

Agent	Our Average	Their Average	N Games	N Players
Internal	10.12 (SD 1.98)	10.97 (SD 1.94)	10^2	2
Outer	13.83 (SD 2.23)	14.53 (SD 2.24)	10^2	2
Bergh Rule	16.95	15.4	10^4	3

The validation results are in Table 7.2. The two Osawa agents obtained similar results in our system to those reported in the original paper. The Bergh Rule agent performed

differently, appearing to be somewhat improved in our system. While there is no apparent reason for this, we have verified that the algorithm is the same as described in their paper, and also that our implementation of *Hanabi* is correct. It is possible that the values are not significantly different, but the error in their calculation is unknown.

7.4.2 Full Test

Table 7.3 shows the full results for this test. Predictor IS-MCTS outperformed IS-MCTS in this experiment, with an average score of 10.74 versus IS-MCTS's score of 5.9. MCS typically performed very similarly to the agent it was provided with for its rollouts; little benefit was apparent from using MCS with these agents over simply using their rules in the first place. Overall, Piers performed the best but not significantly better than MCS-IGGI, IGGI and Bergh Rule. The Flawed agent was only a little better than Legal Random.

TABLE 7.3: Results with Score and Standard Error of the Mean for each agent. Agents are sorted by score. N=11200

Agent	**Score** (2.d.p)	**Sem** (2.d.p)
Piers	11.18	0.06
MCS-IGGI	10.97	0.06
IGGI	10.96	0.06
Bergh Rule	10.88	0.06
Predictor IS-MCTS	10.74	0.06
Outer	10.2	0.05
IS-MCTS	5.9	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 Predictor IS-MCTS really shows its benefit with the Flawed agent as its partner. Table 7.4 shows each agent when paired with Flawed, with Predictor IS-MCTS in the clear lead ahead of other agents.

TABLE 7.4: Table of results with Score and Standard Error of the Mean for each agent paired with Flawed. Agents are sorted by score. N=1600

Agent	**Score** (2.d.p)	**Sem** (2.d.p)
Predictor IS-MCTS	4.82	0.10
IGGI	3.26	0.07
Piers	3.24	0.07
Bergh Rule	3.23	0.07
MCS-IGGI	3.21	0.07
Outer	2.96	0.06
IS-MCTS	1.80	0.05
MCS-Legal Random	1.78	0.05
MCS-Flawed	1.67	0.05
Legal Random	1.65	0.04
Flawed	1.59	0.05

Table 7.5 shows all the agents' average scores over each player count. Most agents tend to follow one of two trends: either performing better when there are more players in the game, or performing worse. Those that improve are typically poor players, with each new player added to the game on average being better than them. Those that decline are the opposite: more players added means more poorer players in the team. Predictor IS-MCTS isn't the only agent to exhibit trouble with two player games: with Outer experiencing some difficulty (despite having been designed for two-player games) and Bergh Rule displaying a more prominent drop in performance. In 3, 4, and 5 player games, the Predictor IS-MCTS is the best player from the set of agents.

TABLE 7.5: Average scores for each agent over 2, 3, 4 and 5 player games sorted alphabetically. Bold scores indicate the statistically highest scores for that column. Scores presented with standard error of mean.

Agent	2	3	4	5
Flawed	3.52±0.06	4.69±0.08	5.43±0.09	6.45±0.09
IGGI	**11.76±0.12**	11.29±0.11	10.71±0.11	10.09±0.11
IS-MCTS	4.8±0.07	5.44±0.08	6.24±0.09	7.14±0.1
Legal Random	1.68±0.03	4.3±0.06	5.83±0.08	6.53±0.08
MCS-Flawed	3.61±0.06	4.72±0.08	5.43±0.09	6.48±0.09
MCS-IGGI	**11.79±0.11**	11.34±0.11	10.68±0.11	10.09±0.11
MCS-Legal Random	3.84±0.06	5.14±0.08	5.87±0.09	6.95±0.1
Outer	10.55±0.1	10.64±0.11	9.99±0.11	9.62±0.1
Piers	**11.91±0.11**	11.67±0.12	10.89±0.12	10.26±0.11
Predictor IS-MCTS	8.36±0.1	**12.14±0.11**	**11.43±0.11**	**11.02±0.11**
Bergh Rule	10.55±0.11	11.76±0.12	10.91±0.12	10.29±0.11

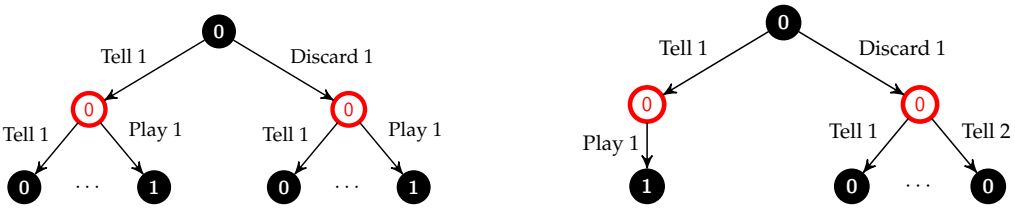
7.5 Discussion

The Predictor IS-MCTS agent outperformed the IS-MCTS agent. This is mostly due to it being better able to take advantage of the effect of communication actions. As agents cannot see their own hands, the only way they gain information about their hands is via tell actions which then inform their decision process. When IS-MCTS appraises the moves of other agents in its tree, it considers all possible outcomes from that state. Some of these states will never occur in the real game because the paired agent would never select that action. The model that is available to Predictor IS-MCTS prunes the search to branches that are likely to occur in the game, resulting in more accurate statistics for the same number of iterations (Figure 7.2). The more deterministic the model, the lower the branching factor for the tree will be. Smaller branching factors concentrate the rollouts, resulting in potentially more accurate statistics regarding those positions. More accurate statistics should result in more intelligent game play.

TABLE 7.6: Average scores for Mimicking (AgentUnderTest is Agent, AgentPaired is Agent) against Prediction (AgentUnderTest is Predictor IS-MCTS, Agent is AgentPaired). Two-player games have been excluded from these results

Agent	Mimicking	Prediction
Flawed	1.65	4.7
IGGI	15.1	15.7
Legal Random	1.25	1.9
Outer	12.18	13.89
Piers	16.17	16.39
Bergh Rule	16.26	16.51

Interestingly, Predictor IS-MCTS's poor overall score appears to come largely from two-player games, for which it scores significantly lower than usual. This can be explained by the decreased rollout length present in these games. The more players in the game, the fewer random moves[2] will be made in the rollouts (selecting random moves tends to end games very quickly with low scores, as exemplified by Legal Random).



(A) IS-MCTS (B) Predictor IS-MCTS

FIGURE 7.2: Game trees from same state for both agents paired with Cautious illustrating the difference in tree size between IS-MCTS and Predictor IS-MCTS

Prediction is shown to be an improvement over mimicking strategy in Table 7.6. This table excludes two-player games because of Predictor IS-MCTS's poor scores in 2 player games, so as to give a fairer overview. Even with intelligent agents, Predictor IS-MCTS provides a benefit compared to an agent using the same strategy as the rest of the team.

[2]Predictor IS-MCTS uses random as the rollout policy for its own moves

7.6 Conclusion

In conclusion, we found that agent modelling significantly improves playing strength for tree search algorithms such as MCTS in the game of *Hanabi*. These results are consistent with the findings from Barrett *et al.*, 2011.

This should be transferable to other co-operative games, though further testing would be required for this.

Chapter 8

Varying Partial Observability

8.1 Introduction

The primary research question this chapter investigates is how does varying the amount of Partial Observability (PO) in a game affect fun and difficulty. Secondarily, it investigates how well Artificial Intelligence (AI) agents can predict the human experience as part of an AI assisted game design experiment. To do this, we experimented using both AI agents and human players in the Ms. Pac–Man Vs Ghost Team Competition framework (Chapter 6). First the AI agents tested a range of possible PO environments, before human participants test between two PO environments. The human experiment was split into two sub-experiments, each changing only a single variable in isolation. The first sub-experiment changed the mode of PO imposed on the user and the second sub-experiment changed the presence of communication, a minor form of observation, within a PO environment.

The Ms. Pac–Man Vs Ghost Team competition was started to promote high quality research into AI in a co-operative PO environment. Competitors could submit either a Ms. Pac–Man agent or a ghost team of four agents. These agents were then played against each other in a round robin to form rankings. The legacy of the competition is that it has provided a number of different ready-made AI agents that have already been evaluated in a competitive setting. This made the work of this chapter possible.

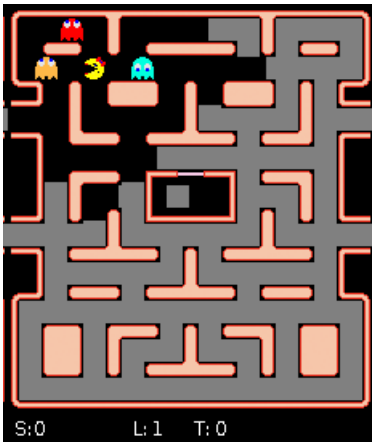
Communication can be a vital tool for agents working in a team, especially if those agents are within a PO environment. In the PO Ms. Pac–Man game, communication occurs via the message-passing Application Programming Interface (API) which, for example, allows the ghost AIs to share information about where Ms. Pac–Man was last seen. Communication is also a form of observation and varying the amount of communication available to an agent will vary the amount of the environment that they can observe.

Agents have to make decisions from a number of variables, and when working in a team some of those variables rely on predictions of what co-workers will do in the future, as shown previously in Chapter 7. In a Completely Observable environment and working with purely reactive deterministic agents, it is possible to perfectly predict all agents' actions. With this knowledge, an agent can then plan its own actions to best achieve the team goal. Non-determinism in co-workers can often be reasoned about with algorithms such as Monte-Carlo Tree Search (MCTS) Chapter 5). When an agent has capabilities beyond being purely reactive(for example, when it has hidden internal state upon which to base its decisions) it can be harder still to co-operate with, and communication can be a solution to allow co-ordination between such agents.

The rest of this chapter is structured as follows:

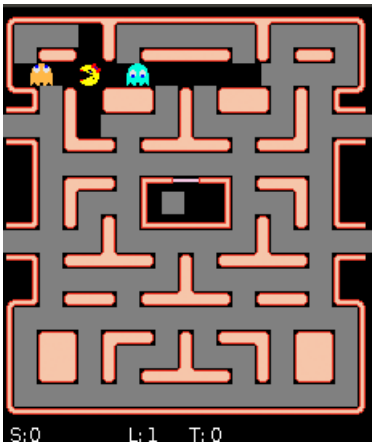
- **Section 8.2** - details the game environment that is used for the experiments as well as specific modifications made for these experiments.
- **Section 8.3** - describes an experiment performed with a series of agents from the most recent iteration of the competition to evaluate the difficulty of various PO configurations.
- **Section 8.4** - describes an experiment whereby human participants play the game in two different configurations and then answer a questionnaire to determine how difficult and enjoyable they found the different games.
- **Section 8.5** - gives a conclusion to the pair of experiments.

(a) Radius



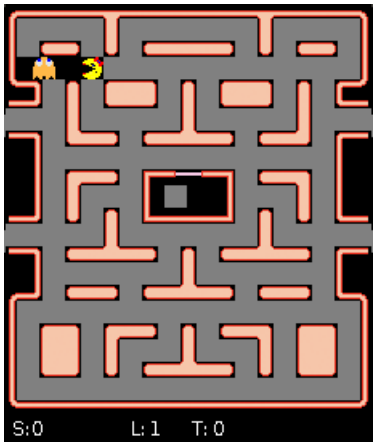
This restriction shows only what the agent can see in a circular fashion with the sight limit determining the radius of the circle of sight that can be seen.

(b) Line-of-Sight



This restriction shows only what the agent can see in the four cardinal directions up until the next maze obstacle or the sight limit; whichever is nearer

(c) Forward Facing Line-of-Sight



This restriction is the same as LOS but only works in the cardinal direction the agent is currently facing.

FIGURE 8.1: The three modes of Partial Observability visualised. Duplicate of Figure 4.4.

8.2 Game Environment

The game environment is the Ms. Pac–Man Vs Ghost Team Competition code-base as described in Section 4.2. This is a close approximation of the original arcade version of Ms. Pac–Man. It has a few minor deviations from the original game such as missing the bonus fruits and not giving Ms. Pac–Man the slight cornering advantage that resulted from the pixel-perfect collision system originally present in the arcade game[1]. There is a considerable API available in the engine to provide utility functions for path-finding and information gathering critical to the AI agents.

The Ms. Pac–Man engine supports three types of PO, each with a sight limit enforced. They are Radius, Line-of-Sight (LOS), and Forward Facing Line-of-Sight (FF-LOS), as shown in Figure 8.1.

[1]The use of pixels for collision led the rounder Ms. Pac–Man to be able to turn a corner slightly earlier than the ghosts, giving a small speed boost.

8.3 Artificial-Intelligence Experiments

The AI experiments investigate the effect of varying PO on Ms. Pac–Man's scores, as a crude measurement of difficulty for both the ghost team and Ms. Pac–Man. A series of games were run consisting of each possible permutation of the following five independent variables:

- **POType** - This is the type of PO used, and can take any of the three values in Figure 8.1.
- **Sight Limit** - This is the number of nodes in the graph that the POType uses to limit the visibility. Takes values from 5 to 50 with a step of 5.
- **Ms. Pac–Man agent** - This is the agent controlling the Ms. Pac–Man character with the possible agents shown in Table 8.1.
- **Ghost team agents** - This is the agent controlling the ghosts with the possible agents shown in Table 8.2.
- **Communication** - Communication was either on or off.

For POType and Sight Limit the constraints were applied equally to both the Ms. Pac–Man agent and the ghost team agents. They were calculated individually, so that the restrictions were from each agents different points of view, but the restriction itself was kept the same. For example, if the POType chosen was Radius with a sight limit of 50, then each of the ghosts and Ms. Pac–Man was able to see each node up to a distance of 50 from their own individual location. The constraints were applied equally to both the Ms. Pac–Man agent and the ghost team in order to reduce the search space, with 10 different values for Sight Limit and 3 different values for POType then differing constraints would have significantly increased the number of varieties tested.

The set of agents for both Ms. Pac–Man and the ghost team were decided by choosing the best five agents each from the respective tracks of the 2017 Ms. Pac–Man Vs Ghost Team Competition, as well as a middle placing agent and the bottom placing agent. These agents

TABLE 8.1: Ms. Pac–Man agents used.

Agent	Technique
SubtleBattle	One Step Lookahead
giangrocker	MCTS
thunder	Beam Search
ToSc	State Machine
BaHe	State Machine
ImHa	Multi-Objective MCTS
MaFr	MCTS

TABLE 8.2: Ghost Team agents used.

Agent	Technique
MaFr	MCTS
TiIsFePr	State Machine
thunder	Rule Based
POGC	Rule Based
NiStTiTi	State Machine
POG	Rule Based
FlBe	State Machine

represent a variety of good agents, middle agents, and poor agents, which have all been tested in an open public competition on the same code base. The original competition that these agents competed in was run with the LOS PO restriction and a sight limit of 50 with communication enabled for the ghost team. The entrants were aware of the restrictions that would be used on their AI agents. The agents chosen are listed in Tables 8.1 and 8.2 along with the basic technique they use. This was either described as part of the submission process, or manually determined by code inspection. At time of writing, the agents have not been described in published work.

8.3.1 Results

The total number of games played was 3 (types of PO) $\times$10 (number of sight limits) $\times$7 (number of Ms. Pac–Man agents) $\times$7 (number of ghost team agents) $\times$2 (Communication on or off) $\times$100 (repeats)$=$ 294,000 games. However some games were incomplete as occasionally[2] the AI agents crashed. This could happen since the AI agents were running beyond their design specification (e.g, having communication switched off, or using a different PO mode from the original competition). The result was 174,871 successfully recorded games.

Figures 8.2 to 8.4 show game scores obtained for each one of the three types of PO restriction, with a separate data series for each Ms. Pac–Man agent averaged over all of the ghost agents at each sight limit. Figures 8.5 to 8.7 show the same, but with a separate data series for each ghost agent, averaged over all of the Ms. Pac–Man agents at each sight limit. In all cases the background shading indicates the 95% confidence interval.

Figure 8.8 shows the score difference between having or not having communication, at a sight limit of 50 on the LOS PO restriction.

It is important to remember at all times that the scores correspond to Ms. Pac–Man's score - so when data on ghosts is presented a lower score indicates that the ghosts performed better.

8.3.2 Discussion

To investigate whether the difference in ability between Ms. Pac–Man and the ghost team is due to an advantage in increased sight, we compiled a graph showing on average how many nodes of the maze's graph were visible to either the ghost team (accounting for overlapping vision) or Ms. Pac–Man in Figure 8.9. It is important to remember that the ghosts cannot directly share this visibility, and there is a small delay on the information that they

[2] A single crash on any of the thousands of turns required to complete a game would invalidate the entire game's result.

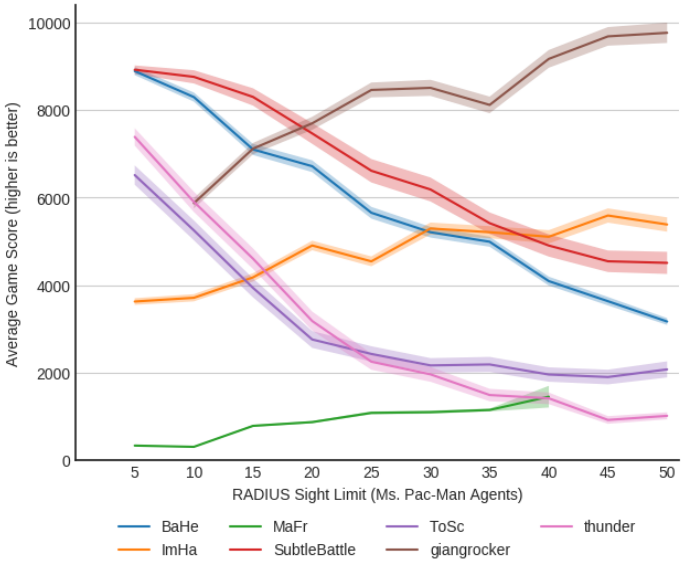


FIGURE 8.2: Effect of increasing "radius" sight limit on scores achieved by various Ms. Pac–Man agents. Background shading indicates error margins.

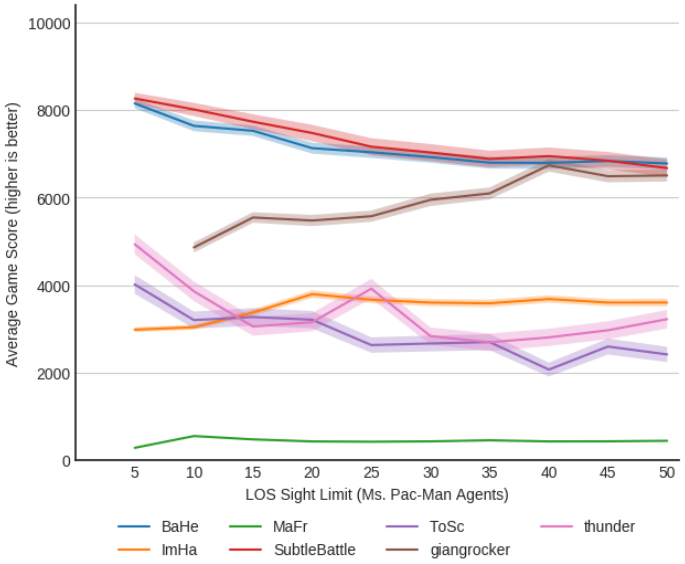


FIGURE 8.3: Effect of increasing "LOS" sight limit on scores achieved by various Ms. Pac–Man agents. Background shading indicates error margins.

can share. The two lines for FF-LOS are indistinguishable from each other while for LOS the ghosts have on average twice the number of nodes visible at one time. For the LOS and FF-LOS modes, very little is gained as the sight limit increases. This is likely due to the fact that the agents are most often in tight little corners with obstacles being the limiting factor rather than the sight limit itself. Radius gives another story, showing a reasonable increase

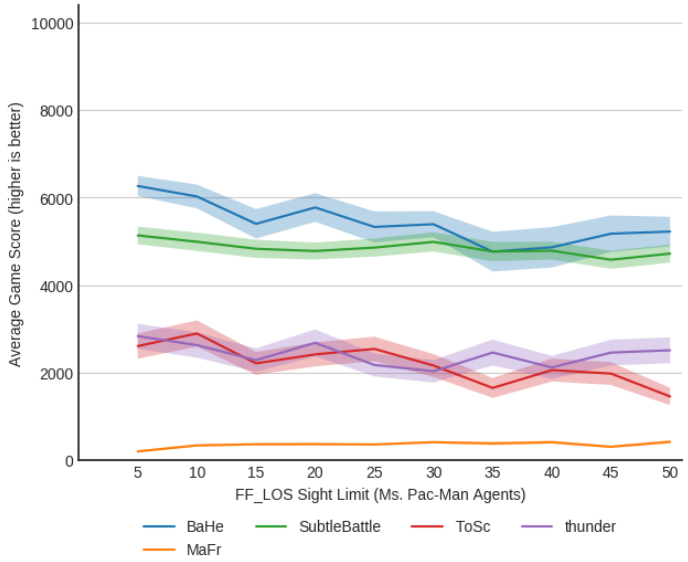


FIGURE 8.4: Effect of increasing "FF-LOS" sight limit on scores achieved by various Ms. Pac–Man agents. Background shading indicates error margins.

FIGURE 8.5: Effect of increasing "radius" sight limit on scores achieved by various ghost agents. Background shading indicates error margins.

as the sight limit increases. It is worth mentioning that the ghosts on average can see over twice as much of the map as Ms. Pac–Man for all the sight limits.

In the case of the Radius restriction most Ms. Pac–Man agents perform worse as the sight

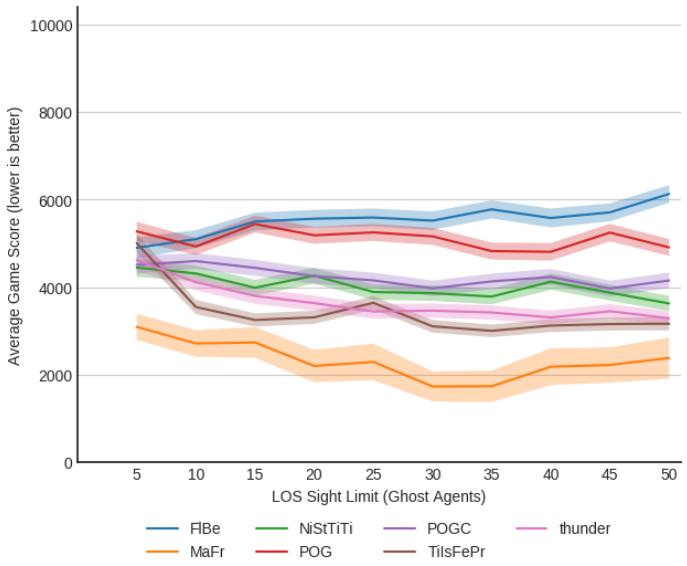


FIGURE 8.6: Effect of increasing "LOS" sight limit on scores achieved by various ghost agents. Background shading indicates error margins.

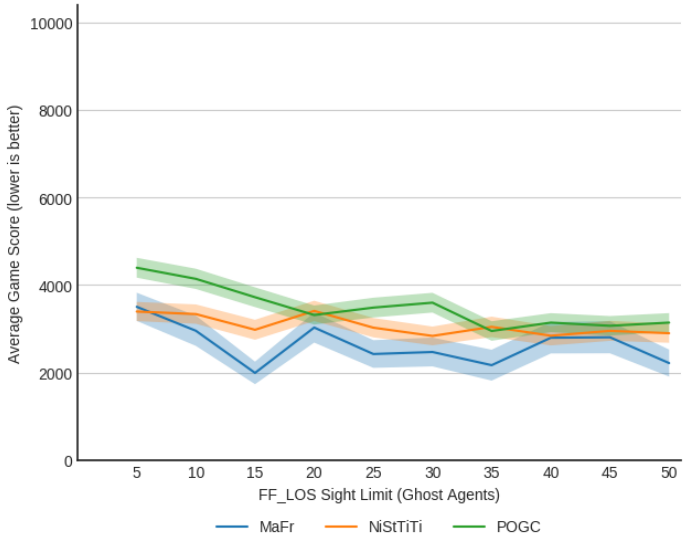


FIGURE 8.7: Effect of increasing "FF-LOS" sight limit on scores achieved by various ghost agents. Background shading indicates error margins.

limit increases with a corresponding increase in performance of the ghost agents (Figures 8.2 and 8.5). This is likely due to the ghosts enjoying over twice the benefit of extra sight than Ms. Pac–Man does (Figure 8.9). In LOS most Ms. Pac–Man agents perform roughly the same across the board with fairly flat lines as sight limit increases (Figure 8.3). The same is repeated for the five agents that completed games in FF-LOS (Figure 8.4).

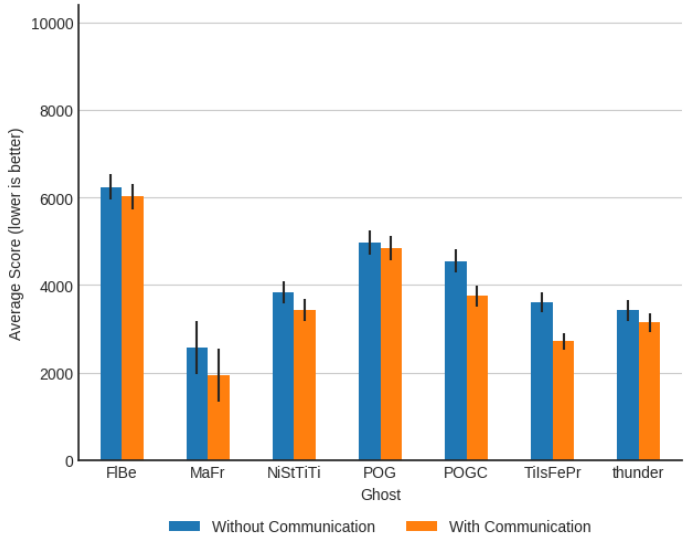


FIGURE 8.8: Effect of communication on scores by ghost agents (lower is better) at a sight limit of 50 on LOS. Error bars shown correspond to 95% confidence intervals

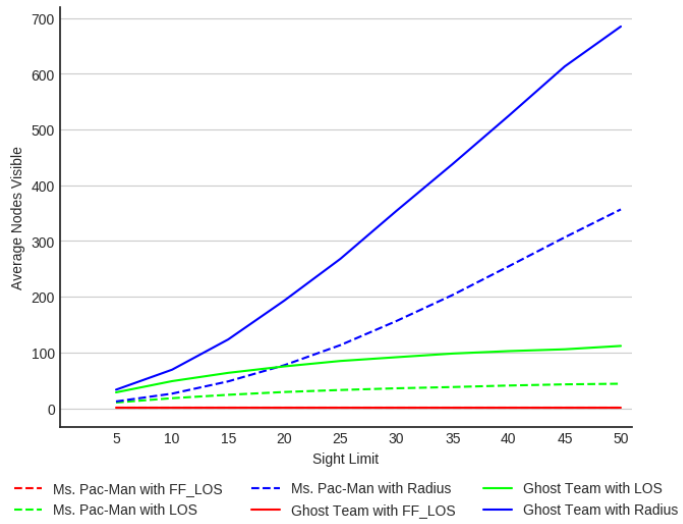


FIGURE 8.9: Average number of board nodes visible to the agents, in different PO modes, as sight limit increases.

One of the better agents from the competition, *giangrocker*, shows a significant benefit with additional sight limits (Figures 8.2 and 8.3). In the LOS mode, *giangrocker* needs the full 50-node sight limit to reach the same levels of performance as *BaHe* and *SubtleBattle* (Figure 8.3) whilst with the Radius PO restriction (Figure 8.2) *giangrocker* overtakes all other agents at the 20-node sight limit and gains an impressive lead. This improvement as sight

increases is, to a lesser degree, shown in the other two MCTS agents (*ImHa* and *MaFr*). Increasing the sight limit in the Radius restriction results in a large gain in information for the agent. MCTS in a PO environment has to make educated guesses about the environment in order to build its tree, with the accuracy of those guesses reducing the number of possible games the algorithm has to reason over. This reduction means that a higher proportion of the tree is relevant to the game leading to more accurate calculations made with that tree. This gives a likely reason for MCTS agents benefiting more from increasing the sight limits, in comparison to the other ghost agents.

Many agents struggled to complete some games, especially those under the FF-LOS restriction [3]. For example, Figures 8.4 and 8.7 show missing data series corresponding to some agents; Figures 8.2 and 8.3 as well as partially Figure 8.5 show some data series do not extend fully across the x-axis. The error logs showed exceptions in the controller code, and it is worth mentioning that the competition they were designed for did not include FF-LOS.

The effect of communication being switched on or off (Figure 8.8) shows that there is a significant difference for two agents (POGC and TiIsFePr). We can see that all ghosts show at least an insignificant positive gain (lower score) with communication on, which indicates that the ghost AIs are using the message-passing API successfully to share knowledge of the game board so as to increase their team performance.

8.4 Human-Participation Experiments

This experiment tasks participants with playing the role of a single ghost, in a small selection of games that have altered PO or communication in them, and then filling out a questionnaire to obtain results. The participants played as a ghost so that they could engage in communication with the team of ghosts, something that playing as Ms. Pac–Man would not allow.

[3] We acknowledge that the failure of some agents more often than others could have potentially skewed the data but errors are shown on the graphs to mitigate this

8.4.1 Setup

In total, 20 people[4] completed the visibility experiment and 19 people[5] completed the communication experiment.

The participants were given a short presentation containing detailed instructions on how to operate the game, as well as what the various location aids on the screen mean. Human players found it surprisingly difficult to get used to playing the role of a ghost, for example due to the inability of ghosts to reverse direction, and the difficulty for experienced Ms. Pac–Man players to focus on their ghost and pursuit, instead of on Ms. Pac–Man and fleeing. The participants were therefore allowed to play three practice games in a fully observable environment, to get accustomed to the control system and ghost behaviour, before starting the main experiment. This was to try and isolate the difficulty of the controls from the observability.

The experiment was conducted in as strict and consistent a manner as possible. All data was gathered on the same machines in the same room and configuration. The participants were tested individually, where each participant took control of the first ghost (Blinky). The remaining three ghosts were controlled by the POGC agent. The Ms. Pac–Man agent was the simple starter agent POPacman from Chapter 6.

Each participant took just one of two comparison experiments, where they compared two variants of the game, and then filled out a questionnaire about their experience. The first comparison-experiment investigated visibility and the participant played a version of the game with PO in Radius mode, and a version of the game with PO in LOS mode, for comparison. These are described as Game Radius with Communication (Radius+) and Game LOS with Communication (LOS+) respectively. The second comparison-experiment investigated communication and the participant played a version of the game in PO LOS mode with communication switched on and a game in PO LOS mode with communication

[4] M/F: 15/5, Age: [18-24: 11, 25-34: 5, 35-49: 2, 50-64: 2]
[5] M/F: 16/3, Age: [18-24: 10, 25-34: 5, 35-49: 2, 50-64: 2]

switched off. These are described as Game LOS+ and Game LOS without Communication (LOS-) respectively. The four games that participants interacted with are describe in Table 8.3. After completing the games, the participants were given the questionnaire. In total, each participant played three distinct games: the practice game and then two of Game Radius+, Game LOS+, or Game LOS-.

The game used modified visuals to give the PO view required, as well as to visualise communication from other ghosts. The modifications include covering non visible areas of the map with a dark grey colouring, as well as the display of "location aids" to represent messages passed by the ghost team. These include a yellow circle to indicate the last observed location of Ms. Pac–Man, and coloured squares to indicate the locations of each ghost. These location aids were only displayed to the user when the communication mode was switched on. A green circle was added to help the player focus on their own ghost. These modified visuals are shown in Figure 8.10. The player used standard keyboard controls to steer their ghost (i.e. arrow keys or WASD keys).

Communication is a critical part of the ghost strategy for handling PO. When communication is switched on in the game, messages are automatically sent on behalf of the player requiring no extra effort or skill from the player. These messages are shown as location aids on screen for the user and are accessible through the API to the AI agents.

Within each comparison experiment, the order that participants played the two games was randomised. To simplify things from the participant's point of view, the two game variants they played were referred to simply as "First Game" or "Second Game" during the experiment and in the subsequent questionnaire. The participants were asked to play two games of the first game followed by two games of the second game. This was then processed into either Game Radius+, Game LOS+, or Game LOS- as appropriate.

The questionnaire given to participants differed slightly between the two experiments. Both forms contained questions taking a user id, the scores obtained in the games, the age range of the participant, and the gender of the participant. Shared questions are shown in Table 8.4. The additional questions for the visibility experiment are shown in Table 8.5,

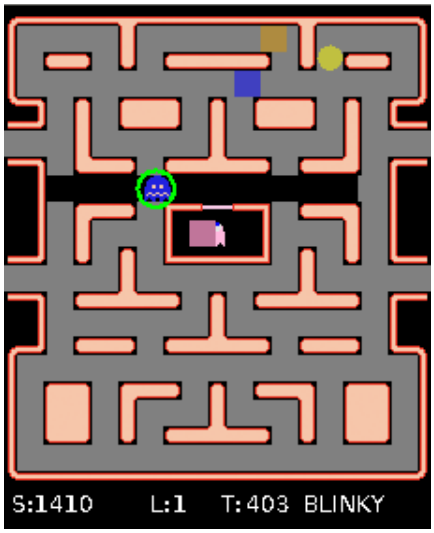


FIGURE 8.10: The graphical changes to the game environment to display required information to human participants.

TABLE 8.3: Settings for the four games in the Human Experiment.

Setting: \ Game:	Practice	Radius+	LOS+	LOS-
PO	Radius	Radius	LOS	LOS
Sight Limit	∞	50	50	50
Location Aid	No	Yes	Yes	No
Ghost AI	POG	POGC	POGC	POG

and the additional questions for the communication experiment are shown in Table 8.6. The questionnaire focuses on asking questions comparing the player experience between the two game variants under comparison.

8.4.2 Results

The results for the various questionnaires have been compiled, calculated, and displayed in Tables 8.7 and 8.8.

The results show significance on a number of the questions given to participants. Those

TABLE 8.4: Questions for both experiments. Some housekeeping questions are omitted.

#	Question	Options
5	In all three games the controls were the same. How difficult did you find the controls?	5 point scale "Easy"(1) → "Difficult"(5)
6	Which game did you enjoy the most?	First Game / Neither / Second Game
7	Which game did you find the most difficult?	First Game / Neither / Second Game
8	Which game did you find the most frustrating	First Game / Neither / Second Game
9	In which game did you feel the most claustrophobic?	First Game / Second Game / Didn't Feel / Equally Claustrophobic

TABLE 8.5: Additional questions for the visibility experiment. Questions re-ordered from questionnaire so that Question 10 matches between experiments better.

#	Question	Options
10	In which game did you find the location aid most useful?	First Game / Second Game / Didn't Find Useful / Equally Useful
11	In which game did you find the location aid more noticeable?	First Game / Second Game / Didn't Notice / Equally Noticeable

TABLE 8.6: Additional questions for the communication experiment.

#	Question	Options
10	Did you find the location aid in the first game useful?	Yes / No / Didn't Notice

completing the visibility experiment gave a small preference that they enjoyed Game Radius+ the most and significantly[6] found Game LOS+ the most difficult to play. In both experiments, participants did not feel claustrophobic.

[6] All significance tests are performed using χ^2 tests with a null hypothesis that answers are uniformly distributed. In the questions with more than two answers, significance is individually tested between all pairs of answers. P-values are not correct for multiple comparisons.

TABLE 8.7: Results of the questionnaire.

Experiment	Question	Results (%)				
Visibility	5. Game Controls	1(30.00)	2(15.00)	3(30.00)	4(20.00)	5(5.00)
Communication		1(31.58)	2(36.84)	3(21.05)	4(5.26)	5(5.26)
Visibility	6. Enjoyment	Radius+(50.00)	N(10.00)	LOS+(40.00)		
Communication		LOS+(57.89)	N(5.26)	LOS-(36.84)		
Visibility	7. Difficulty	Radius+(10.53)	LOS+(78.95)	N(10.53)		
Communication		LOS+(42.11)	LOS-(52.63)	N(5.26)		
Visibility	8. Frustration	Radius+(10.53)	LOS+(42.11)	N(42.11)		
Communication		LOS+(21.05)	LOS-(31.58)	N(47.37)		
Visibility	9. Claustrophobia	Radius+(5.26)	LOS+(21.05)	DF(68.42)	EC(5.26)	
Communication		LOS+(10.53)	LOS-(26.32)	DF(63.16)	EC(0.00)	
Visibility	10. Usefulness of Location Aid	Radius+(52.63)	LOS+(21.05)	DFU(5.26)	EU(21.05)	
Communication		Yes(84.21)	No(10.53)	DN(5.26)		
Visibility	11. Noticability of Location Aid	Radius+(36.84)	LOS+(26.32)	DN(5.26)	EN(31.58)	

TABLE 8.8: Average scores obtained in the human experiments.

Experiment	Radius+	LOS+	LOS-
Visibility	4323	5993	
Communication		5842	5421

In the visibility experiment participants found the location aid most useful in Game Radius+ rather than Game LOS+. The communication experiment concluded with significance that the location aid was useful when it was present in Game LOS+.

8.4.3 Discussion

The following subsections will individually discuss parts of the results. It is important to remember that while Game LOS+ was used in both experiments, the answers are all comparative.

Difficulty

Difficulty was measured in two ways for each experiment. The first method mirrors the AI experiments by obtaining score information for each participant. The second method is measured by question 7 and partially by question 8's focus on frustration that could be an indicator of difficulty.

Looking at the scores (Table 8.8) obtained, Game Radius+ is the easiest for humans with an average of 4323 which is much lower than the averages for Game LOS+ (5993, 5842). Game LOS- interestingly on average scores better with 5421, despite the game being theoretically harder due to not featuring communication. Game LOS- was predicted to be harder by the AI experiments (Figure 8.8). It is possible that the additional on-screen information proved to be more distracting than useful to people which is a problem AIs will not suffer from. The two cohorts found Game LOS+ similarly difficult when looking at achieved scores with close averages of 5889 and 5765.

Looking at the questionnaire results for the visibility experiment, the participants found that Game LOS+ was the most difficult with significance while the communication experiment was inconclusive between Game LOS+ and Game LOS-. The AI experiments showed a much smaller gap between Game Radius+ and Game LOS+ and a large gap in scores between Game LOS+ and Game LOS-.

Enjoyment

While there are no AI results covering enjoyment, question 6 from the questionnaire covers enjoyment of the games for humans. Enjoyment in the visibility experiment showed that Game Radius+ was slightly better than Game LOS+ but results were mixed. The communication experiment proved similarly mixed with Game LOS+ taking the lead nonetheless. The AI scores for these games showed that the more difficult games for AI tended to be those that the participants liked the least.

Communication

Communication is measured for humans through both questions as well as the scores in the communication experiment. The effect of AI communication was measured by scores alone. As mentioned previously humans performed worse with communication than without, contrasting the AI results who performed better.

In the visibility experiment, players found the location aid more noticeable and more useful in Game Radius+ than they did in Game LOS+. These results are perhaps counter-intuitive, expecting the location aids to be more useful as vision decreases. In the communication experiment the results are more one sided, with the majority saying that it was useful to have communication. Filtering the data to obtain scores for Game LOS+ and Game LOS- for those participants that stated that communication was useful yielded averages of 5407 for Game LOS+ and 5279 for Game LOS-[7]. Therefore despite believing communication was useful, on average it was not beneficial to the scores. This is an interesting result with no definitive reason why this might be the case. Comments from the participants hint at a potential information overload, perhaps making the communication hints a distraction[8] that cost them game performance.

Frustration

In both experiments[9], more participants found the most restrictive game visually (Game LOS+ for the visibility experiment and Game LOS- for the communication experiment) the most frustrating, but the majority found neither game to be frustrating.

[7] The scores are very noisy but it would take an unfeasible number of participants to bring the error to a reasonable level.

[8] Possibly divided attention

[9] One participant indicated both game Radius+ and LOS+ and so is not reflected in the table

Claustrophobia

In both experiments, the majority of participants declared that they did not feel claustrophobic in either game.

8.5 Conclusions

In conclusion we found that altering the amount of PO within the game of Ms. Pac–Man changes the balance of the game with most ghost team agents performing worst with the lower limits of visibility. AI experiments have shown that an advantage in the amount of the map that is visible tends to be an important influence on the performance of agents depending on their particular strategies.

Secondly we found that the presence of communication for the ghost team is beneficial for AIs but conversely found that it appears to hinder human performance despite the majority of participants declaring communication to be useful. This result reinforces the notion that it is challenge that players enjoy in many cases, rather than simply winning easily. It is also possible that the communication improves the feeling of control in the game, leading to less situations of random searching for the Ms. Pac–Man and more chasing which could lead to improved enjoyment.

Finally it is recommended that the use of AI proxies as indicators for human enjoyment should be done cautiously. There are aspects of human game playing that are difficult to emulate in AI agents such as the inability for humans to focus on all of even a small game state at once or the lightning fast reflexes of an AI. Making more human-like game AI in general is an important goal to enable more accurate estimation of the human player's experience, and whether this is best done by building AI agents that model specific aspects of perception and cognition, or by using general learning agents trained on human behaviour is an open question.

Part III

Conclusions

Chapter 9

Conclusions

This chapter will conclude the thesis, as well as discuss some future work.

9.1 Conclusions

This thesis has examined the use of Artificial Intelligence (AI) agents within co-operative environments with Partial Observability (PO) constraints. Early experiments showed that in the *Tiny Co-op* domain, Monte-Carlo Tree Search (MCTS) agents were superior in handling the co-operative aspect of the puzzles than the Genetic Algorithms (GAs) (Chapter 5).

The construction of the Ms. Pac–Man Vs. Ghost Team competition has raised the profile of research into PO in games as well as co-operation in games. The competition has run successfully for three years with a reasonable number of entrants submitting high quality agents (Chapter 6).

Moving into the simpler environment of *Hanabi*, where communication and PO are a part of the game mechanics, we found that MCTS benefited greatly from the addition of a model of the agents that MCTS was playing the game with (Chapter 7).

Final experiments with PO in the *Ms. Pac–Man* environment tested a barrage of AI agents[1]

[1]Drawn from the 2017 competition entrants

across a range of different PO restrictions. It was found that the more of the map is visible, the better the ghost agents tended to perform, with the exception of MCTS Ms. Pac–Man agents that gained more intelligence from the additional visibility than the ghost agents (Section 8.3). Human participant experiments were also performed, finding that communication for the ghost team proved detrimental to performance, despite questionnaire results showing that the participants found communication to be useful (Section 8.5).

9.2 Future Work

There is a lot of scope for future work to add to the research performed in this thesis. In the *Tiny Co-op* domain, the best agent presented (High MCTS) had scope for improvement, as well as the addition of more complex puzzles. Both the size of the environment and the variety of interactive elements can be increased for further research.

The Ms. Pac–Man Vs. Ghost Team competition has performed well, with a strong number of entrants each year. It does need, however, some changes to improve research interest. It would be good to see better support for deep learning techniques, possibly as part of a learning track. Another avenue of further work in the competition would be to introduce restrictions to number of messages that the ghosts can send, forcing the agents to work within a budget.

Agents in *Hanabi* using techniques such as MCTS are still performing below the maximum possible score for a game, leaving room for improvement. More advanced techniques can be researched here, as well as further work on learning the models that helped the Predictor IS-MCTS agent perform so well. This would reduce its reliance on being given a perfect model, which is an unrealistic situation.

The exploration of Ms. Pac–Man needs expanding into other games and types of games to try to correlate patterns and variations between games. If similar results are found in similar games it would be possible to derive stronger guidelines about the use of PO within games as a mechanic, opening the way to offer new and interestingly different ways to

play many classic games. Chapter 8 found that more difficult games for AI tended to be the less enjoyable games for participants, and the exact nature of this link would need a full experiment in multiple games ideally to investigate properly. Chapter 8 also explored some ways that PO could be altered in the game of Ms. Pac–Man. There are presumably a number of other interesting ways such as asymmetric PO between the ghost team and Ms. Pac–Man. This would represent a much finer grained approach to changing the game, and potentially provide better results. The use of PO as a game mechanic has been discussed at length but there are other things that could be used as well. The effect of communication within a PO environment is worth deeper analysis, with different types of communication as well as tweaking various parameters such as message delay, cost of delivery, or chance of successful delivery. Altering these parameters could provide an interesting effect on the game, potentially adding some strategy to balancing communication with the cost that is not currently present.

Finally there is an increasing amount of interest in Real-Time Strategy (RTS) games, such as Starcraft II, which feature hundreds of units that each have their own visibility restrictions and must work together as a team. A lot of the early research so far has been to control the teams with a single AI, but it could be possible to use some of the techniques from this thesis to work with a multi-agent system instead.

Bibliography

Ahlbrecht, Tobias, Jürgen Dix, Michael Köster and Federico Schlesinger (2013). 'Multi-agent programming contest 2013'. In: *International Workshop on Engineering Multi-Agent Systems*. Springer, pp. 292–318.

Ahlbrecht, Tobias, Jürgen Dix and Niklas Fiekas (2018). 'Multi-agent programming contest 2016'. In: *International Journal of Agent-Oriented Software Engineering* 6.1, pp. 58–85.

Alhejali, Atif M and Simon M Lucas (2010). 'Evolving Diverse Ms. Pac-Man Playing Agents Using Genetic Programming'. In: *Computational Intelligence (UKCI), 2010 UK Workshop on*. IEEE, pp. 1–6.

– (2013). 'Using Genetic Programming to Evolve Heuristics for a Monte Carlo Tree Search Ms Pac-Man Agent'. In: *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, pp. 1–8.

Anderson-Cook, Christine M (2005). *Practical genetic algorithms*.

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.

Bäck, T, D.B Fogel and Z Michalewicz, eds. (2000). *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol.

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.

Behrens, Tristan, Michael Köster, Federico Schlesinger, Jürgen Dix and Jomi F Hübner (2012). 'The Multi-Agent Programming Contest 2011: A Résumé'. In: *Programming Multi-Agent Systems*. Springer, pp. 155–172.

Bergh, Mark JH van den, 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.

Bertoli, Piergiorgio, Alessandro Cimatti, Marco Roveri and Paolo Traverso (2001). 'Planning in nondeterministic domains under partial observability via symbolic model checking'. In:

Beume, Nicola *et al.* (2008). 'Measuring flow as concept for detecting game fun in the Pac-Man game'. In: *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*. IEEE, pp. 3448–3455.

Bouzy, Bruno (2017). 'Playing Hanabi Near-Optimally'. In: *Advances in Computer Games*. Springer, pp. 51–62.

Browne, Cameron B *et al.* (2012). 'A Survey of Monte Carlo Tree Search Methods'. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 4.1, pp. 1–43.

Burrow, Peter and Simon M Lucas (2009). 'Evolution Versus Temporal Difference Learning For learning to Play Ms. Pac-Man'. In: *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, pp. 53–60.

Cazenave, Tristan (2005). 'A phantom-go program'. In: *Advances in Computer Games*. Springer, pp. 120–125.

– (2014). 'Sequential Halving Applied to Trees'. In:

Cazenave, Tristan and Nicolas Jouandeau (2007). 'On the Parallelization of UCT'. In: *Proceedings of CGW07*, pp. 93–101.

Chaslot, Guillaume, Sander Bakkes, Istvan Szita and Pieter Spronck (2008a). 'Monte-Carlo Tree Search: A New Framework for Game AI.' In: *AIIDE*.

Chaslot, Guillaume MJ-B, Mark HM Winands and H Jaap van Den Herik (2008b). 'Parallel Monte-Carlo Tree Search'. In: *Computers and Games*. Springer, pp. 60–71.

Clune, James (2007). 'Heuristic Evaluation Functions for General Game Playing'. In: *AAAI*. Vol. 7, pp. 1134–1139.

Coulom, Rémi (2007). 'Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search'. In: *Computers and games*. Springer, pp. 72–83.

Cowling, Peter I, Edward J Powley and Daniel Whitehouse (2012). '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.

Dienstknecht, M (2018). 'Enhancing Monte Carlo Tree Search by Using Deep Learning Techniques in Video Games'. PhD thesis. Maastricht University.

Dockhorn, Alexander and Rudolf Kruse (2017). 'Combining cooperative and adversarial coevolution in the context of pac-man'. In: *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, pp. 60–67.

Domınguez-Estévez, Fernando, Antonio A Sánchez-Ruiz and Pedro Pablo (2017). 'Training Pac-Man bots using Reinforcement Learning and Case-based Reasoning'. In:

Drake, Peter (2009). 'The last-good-reply policy for Monte-Carlo Go'. In: *Icga Journal* 32.4, pp. 221–227.

Emilio, Martin, Martinez Moises, Recio Gustavo and Saez Yago (2010). 'Pac-mAnt: Optimization based on ant colonies applied to developing an agent for Ms. Pac-Man'. In: *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, pp. 458–464.

Fang, Xiaowen, Susy Chan, Jacek Brzezinski and Chitra Nair (2008). 'Measuring enjoyment of computer game play'. In: *AMCIS 2008 Proceedings*, p. 306.

Finnsson, Hilmar and Yngvi Björnsson (2008). 'Simulation-Based Approach to General Game Playing.' In: *AAAI*. Vol. 8, pp. 259–264.

– (2010). 'Learning Simulation Control in General Game-Playing Agents.' In: *AAAI*. Vol. 10, pp. 954–959.

Garduño Hernández, Daniel *et al.* (2017). 'Study of artificial intelligence algorithms applied to the generation of non-playable characters in arcade games'. B.S. thesis.

Hanabi Agent Competition. URL: http://hanabi.aiclash.com/ (visited on 14/01/2019).

Handa, Hisashi and Maiko Isozaki (2008). 'Evolutionary fuzzy systems for generating better Ms. PacMan players'. In: *Fuzzy Systems, 2008. FUZZ-IEEE 2008.(IEEE World Congress on Computational Intelligence). IEEE International Conference on*. IEEE, pp. 2182–2185.

Hearthstone AI Competition. URL: https://dockhorn.antares.uberspace.de/wordpress/ (visited on 14/01/2019).

Hingston, Philip (2010). 'A new design for a turing test for bots'. In: *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, pp. 345–350.

Hunicke, Robin, Marc LeBlanc and Robert Zubek (2004). 'MDA: A formal approach to game design and game research'. In: *Proceedings of the AAAI Workshop on Challenges in Game AI*. Vol. 4. 1.

Isaksen, Aaron, Dan Gopstein, Julian Togelius and Andy Nealen (2015a). 'Discovering unique game variants'. In: *Computational Creativity and Games Workshop at the 2015 International Conference on Computational Creativity*.

Isaksen, Aaron, Daniel Gopstein and Andrew Nealen (2015b). 'Exploring Game Space Using Survival Analysis.' In: *FDG*.

Isaksen, Aaron, Dan Gopstein, Julian Togelius and Andy Nealen (2017). 'Exploring Game Space of Minimal Action Games via Parameter Tuning and Survival Analysis'. In: *IEEE Transactions on Computational Intelligence and AI in Games*.

Khajah, Mohammad M, Brett D Roads, Robert V Lindsey, Yun-En Liu and Michael C Mozer (2016). 'Designing engaging games using bayesian optimization'. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, pp. 5571–5582.

Kitano, Hiroaki and Satoshi Tadokoro (2001). 'Robocup rescue: A Grand Challenge for Multiagent and Intelligent Systems'. In: *AI magazine* 22.1, p. 39.

Kitano, Hiroaki, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda and Eiichi Osawa (1997). 'Robocup: The robot world cup initiative'. In: *Proceedings of the first international conference on Autonomous agents*. ACM, pp. 340–347.

Kocsis, Levente and Csaba Szepesvári (2006). 'Bandit Based Monte-Carlo Planning'. In: *Machine Learning: ECML 2006*. Springer, pp. 282–293.

Koriche, Frédéric, Sylvain Lagrue, Éric Piette and Sébastien Tabary (2017). 'WoodStock: un programme-joueur générique dirigé par les contraintes stochastiques'. In: *Revue d'intelligence artificielle–no* 307, p. 336.

Koster, Raph (2013). *Theory of fun for game design*. " O'Reilly Media, Inc."

Koza, John R (1992). *Genetic programming: on the Programming of Computers by means of Natural Selection*. Vol. 1. MIT press.

Lanctot, Marc, Kevin Waugh, Martin Zinkevich and Michael Bowling (2009). 'Monte Carlo sampling for regret minimization in extensive games'. In: *Advances in neural information processing systems*, pp. 1078–1086.

Lee, Scott and Julian Togelius (2017). 'Showdown AI competition'. In: *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, pp. 191–198.

Liberatore, Federico, Antonio M Mora, Pedro A Castillo and Juan Julián Merelo Guervós (2014). 'Evolving evil: optimizing flocking strategies through genetic algorithms for the ghost team in the game of Ms. Pac-Man'. In: *Applications of Evolutionary Computation*. Springer, pp. 313–324.

Lucas, Simon M (2005). 'Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man.' In: *CIG*. Citeseer.

Méhat, Jean and Tristan Cazenave (2010). 'Ary, a General Game Playing Program'. In: *Board Games Studies Colloquium*.

Nabi, Robin L and Marina Krcmar (2004). 'Conceptualizing media enjoyment as attitude: Implications for mass media effects research'. In: *Communication theory* 14.4, pp. 288–310.

Nelson, Mark J and Michael Mateas (2007). 'Towards automated game design'. In: *Congress of the Italian Association for Artificial Intelligence*. Springer, pp. 626–637.

Nguyen, Kien Quang and Ruck Thawonmas (2011). 'Applying Monte-Carlo Tree Search To Collaboratively Controlling of a Ghost Team in Ms Pac-Man'. In: *Games Innovation Conference (IGIC), 2011 IEEE International*. IEEE, pp. 8–11.

Ontanón, Santiago (2013). 'The combinatorial multi-armed bandit problem and its application to real-time strategy games'. In: *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

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*.

Pell, Barney (1993). 'Strategy Generation and Evaluation for Meta-Game Playing'. PhD thesis. University of Cambridge Ph. D. thesis.

Pepels, Tom, Mark HM Winands and Marc Lanctot (2014). 'Real-Time Monte Carlo Tree Search in Ms Pac-Man'. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 6.3, pp. 245–257.

Perez, 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.

Perez-Liebana, 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.

Prada, Rui, Phil Lopes, João Catarino, João Quitério and Francisco S. Melo (2015). 'The Geometry Friends Game AI Competition'. In: *CIG'2015 - IEEE Conference on Computational Intelligence and Games*. IEEE CIG. Tainan, Taiwan: IEEE Computer Society, pp. 431–438.

R0oland (2013a). *Single Point Crossover*. URL: https://en.wikipedia.org/wiki/File:OnePointCrossover.svg (visited on 16/01/2019).

– (2013b). *Two Point Crossover*. URL: https://en.wikipedia.org/wiki/File:TwoPointCrossover.svg (visited on 16/01/2019).

Rohlfshagen, Philipp and Simon M Lucas (2011). 'Ms Pac-Man versus Ghost Team CEC 2011 Competition'. In: *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, pp. 70–77.

Rose, Andrew and Steve Draper. *Sancho goes Green*. http://sanchoggp.blogspot.co.uk/2014/06/sancho-goes-green.html. Accessed: 2015-09-28.

Ryan, Richard M, C Scott Rigby and Andrew Przybylski (2006). 'The motivational pull of video games: A self-determination theory approach'. In: *Motivation and emotion* 30.4, pp. 344–360.

Samothrakis, Spyridon and Simon M Lucas (2010). 'Planning using online evolutionary overfitting'. In: *Computational Intelligence (UKCI), 2010 UK Workshop on*. IEEE, pp. 1–6.

Samothrakis, Spyridon, David Robles and Simon Lucas (2011). 'Fast Approximate Max-n Monte Carlo Tree Search for Ms Pac-Man'. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 3.2, pp. 142–154.

Schaul, Tom (2013). 'A Video Game Description Language for Model-based or Interactive Learning'. In: *Proceedings of the IEEE Conference on Computational Intelligence in Games*. Niagara Falls: IEEE Press.

Schiffel, Stephan and Michael Thielscher (2006). 'Automatic Construction of a Heuristic Search Function for General Game Playing'. In: *Department of Computer Science*, pp. 16–17.

Schrum, Jacob and Risto Miikkulainen (2014). 'Evolving multimodal behavior with modular neural networks in Ms. Pac-Man'. In: *Proceedings of the 2014 annual conference on genetic and evolutionary computation*. ACM, pp. 325–332.

Sharma, Shiven, Ziad Kobti and Scott Goodwin (2008). 'Knowledge Generation for Improving Simulations in UCT for General Game Playing'. In: *AI 2008: Advances in Artificial Intelligence*. Springer, pp. 49–55.

Silver, David and Joel Veness (2010). 'Monte-Carlo planning in large POMDPs'. In: *Advances in neural information processing systems*, pp. 2164–2172.

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 chess and shogi by self-play with a general reinforcement learning algorithm'. In: *arXiv preprint arXiv:1712.01815*.

StarCraft AI Competition. URL: https://www.cs.mun.ca/~dchurchill/starcraftaicomp/index.shtml (visited on 14/01/2019).

Tak, Mandy JW, Mark HM Winands and Yngvi Björnsson (2012). 'N-grams and the Last-Good-Reply Policy applied in General Game Playing'. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 4.2, pp. 73–83.

VIZDOOM. URL: http://vizdoom.cs.put.edu.pl/ (visited on 14/01/2019).

Walton-Rivers, Joseph, Piers R Williams, Richard Bartle, Diego Perez-Liebana and Simon M Lucas (2017). 'Evaluating and Modelling Hanabi-Playing Agents'. In: *Congress on Evolutionary Computation, 2017. CEC'17. IEEE Conference On*. IEEE, pp. 1382–1389.

Wellman, Michael P *et al.* (2001). 'Designing the Market Game for a Trading Agent Competition'. In: *Internet Computing, IEEE* 5.2, pp. 43–51.

Williams, Piers R., Joseph Walton-Rivers, Diego Perez-Liebana and Simon M. Lucas (2015). 'Monte Carlo Tree Search Applied to Co-operative Problems'. In: *CEEC'2015 - IEEE Conference on Computer Science and Electronic Engineering*. IEEE CEEC. IEEE Computer Society, pp. 219–224.

Williams, Piers R, Diego Perez-Liebana and Simon M Lucas (2016). 'Ms. Pac-Man Versus Ghost Team CIG 2016 Competition'. In: *CIG'2016 - IEEE Conference on Computational Intelligence and Games*. IEEE CIG, pp. 420–427.

Wittkamp, Markus, Luigi Barone and Philip Hingston (2008). 'Using NEAT for continuous adaptation and teamwork formation in Pacman'. In:

Yannakakis, Georgios N and John Hallam (2004). 'Evolving opponents for interesting interactive computer games'. In: *From animals to animats* 8, pp. 499–508.

Zhang, Qi, Jian Yao, Quanjun Yin and Yabing Zha (2018). 'Learning Behavior Trees for Autonomous Agents with Hybrid Constraints Evolution'. In: *Applied Sciences* 8.7, p. 1077.

Zhong, Jinghui, Xiaomin Hu, Jun Zhang and Min Gu (2005). 'Comparison of performance between different selection strategies on simple genetic algorithms'. In: *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*. Vol. 2. IEEE, pp. 1115–1121.