Universidad Autónoma de Madrid
Escuela Politécnica Superior

Trabajo fin de grado

STUDY OF ARTIFICIAL INTELLIGENCE ALGORITHMS APPLIED TO THE GENERATION OF NON-PLAYABLE CHARACTERS IN ARCADE GAMES

Grado en Ingeniería Informática

Daniel Garduño Hernández
July 2017

STUDY OF ARTIFICIAL INTELLIGENCE ALGORITHMS APPLIED TO THE GENERATION OF NON-PLAYABLE CHARACTERS IN ARCADE GAMES

STUDENT: Daniel Garduño Hernández
ADVISOR: Antonio González Pardo
CO-ADVISOR: David Camacho Fernández

Grupo de la EPS: Applied Intelligence & Data Analysis (AIDA)
Dpto. de Ingeniera Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
July 2017

Resumen

Resumen

En la actualidad, el auge de la Inteligencia Artificial en diversos campos está llevando a un aumento en la investigación que se lleva a cabo en ella. Uno de estos campos es el de los videojuegos.

Desde el inicio de los videojuegos, ha primado la experiencia del usuario en términos de jugabilidad y gráficos, sobre todo, prestando menor atención a la Inteligencia Artificial. Ahora, debido a que cada vez se dispone de mejores máquinas que pueden realizar acciones computacionalmente más caras con menor dificultad, se están pudiendo aplicar técnicas de Inteligencia Artificial más complejas y que aportan mejor funcionamiento y dotan a los juegos de mayor realismo. Este es el caso, por ejemplo, de la creación de agentes inteligentes que imitan el comportamiento humano de una manera más realista.

En los úlimos años, se han creado diversas competiciones para desarrollar y analizar técnicas de Inteligencia Artificial aplicadas a los videojuegos. Algunas de las técnicas que son objeto de estudio son la generación de niveles, como en la competición de Angry Birds; la minería de datos sacados de registros de juegos MMORPG (videojuego de rol multijugador masivo en línea) para predecir el compromiso económico de los jugadores, en la competición de minería de datos; el desarrollo de IA para desafíos de los juegos RTS (estrategia en tiempo real) tales como la incertidumbre, el procesado en tiempo real o el manejo de unidades, en la competición de StarCraft; o la investigación en PO (observabilidad parcial) en la competición de Ms. Pac-Man mediante el diseño de controladores para Pac-Man y el Equipo de fantasmas.

Este trabajo se centra en esta última competición, y tiene como objetivo el desarrollo de una técnica híbrida consistente en un algoritmo genético y razonamiento basado en casos. El algoritmo genético se usa para generar y optimizar un conjunto de reglas que los fantasmas utilizan para jugar contra Ms. Pac-Man.

Posteriormente, se realiza un estudio de los parámetros que intervienen en la ejecución del algoritmo genético, para ver como éstos afectan a los valores de fitness obtenidos por los agentes generados.

Palabras Clave

Inteligencia artificial, inteligencia computacional, algoritmo genético, computación evolutiva, evolución, crossover, mutación, razonamiento basado en casos, videojuego pac-man, videojuego, obervabilidad parcial

Summary

Abstract

Recently, the increase in the use of Artificial Intelligence in different fields is leading to an increase in the research being carried out. One of these fields is videogames.

Since the beginning of videogames, the user experience in terms of gameplay and graphics has prevailed, paying less attention to Artificial Intelligence for creating more realistic agents and behaviours. Nowadays, due to the availability of better machines that can perform computationally expensive actions with less difficulty, more complex Artificial Intelligence techniques that provide games with better performance and more realism can be implemented. This is the case, for example, of creating intelligent agents that mimic human behaviour in a more realistic way.

Different competitions are held every year for research into AI techniques through videogames. Some of the techniques that are object for study are level generation, such as in the Angry Birds AI Competition, data mining from MMORPG (massively multiplayer online role-playing game) game logs to predict game players' economic engagement, in the Game Data Mining Competition; the development of RTS (Real-Time Strategy) game AI for solving challenging issues such as uncertainty, real-time process and unit management, in the StarCraft AI Competition; or the research into PO (Partial Observability) in the Ms. Pac-Man Vs Ghost Team Competition by designing agents for Ms. Pac-Man and the Ghost Team.

This work is focused on this last competition, and has the objective of designing a hybrid technique consisting of a genetic algorithm and case-based reasoning. The genetic algorithm is used to generate and optimize set of rules that the Ghosts use to play against Ms. Pac-Man.

Later, we perform an analysis of the parameters that intervene in the execution of the genetic algorithm to see how they affect the fitness values that the generated agents obtain by playing the game.

Key words

Artificial intelligence, computational intelligence, genetic algorithm, evolutionary computation, evolution, crossover, mutation, case-based reasoning, pac-man game, games, partial observability

Agradecimientos

A mis padres, por haber hecho esto posible y por toda la paciencia y confianza depositadas en mí durante estos años de carrera. Estoy muy agradecido.

A Cristina, por todo su apoyo, por sus ánimos, por su paciencia, y por la tranquilidad que me ha transmitido todo este tiempo.

A mis compañeros de piso, José Luis y Héctor, y especialmente a este último por todo el conocimiento y la ayuda prestados.

A Antonio, mi tutor, por la ayuda y la guía prestadas en la elaboración de este trabajo. Al resto de profesores que he tenido a lo largo de la carrera por su dedicación y por todo lo que han ayudado.

A todos mis amigos, compañeros de clase y de prácticas, por haber hecho del paso por la universidad una experiencia muy agradable.

Contents

List of Figures

List of Tables

Glossary

- **RTS**: Real-Time Strategy, a game genre based on the tactical decisions of the player, who controls a team.
- **FPS**: First Person Shooter, a game genre where the player controls a character in first person point of view.
- **MMORPG**: Massively Multiplayer Online Role-Playing Game, a game genre combination of roke-playinf and multiplayer online where a large number of people interact with each other in a virtual world.
- **ANN**: Artificial Neural Network, is a set of artificial neurons that model biological neurons.
- **MCTS**: Monte-Carlo Tree Search, a heuristic search algorithm for decision processes.
- **CBR**: Case-Based Reasoning, a paradigm in automated reasoning and machine learning that aims to solve new problems based on previously experienced solutions.
- **ACO**: Ant Colony Optimization, a probabilistic technique for shortest path optimization based on ants behaviours.
- **PSO**: Particle Swarm Optimization, a stochastic optimization approach to Swarm Intelligence, modelled on the social behaviour of bird flocks.
- **MLP**: Multi-Layer Perceptron,a feedforward artificial neural network model that maps sets of inputs onto sets of outputs.
- **AI**: Artificial Intelligence
- **GALR**: Genetic Algorithm with Lexicographic Ranking
- **EC**: Evolutionary Computation, a subfield of AI that studies the algorithms for global optimization inspired by biological evolution.
- **CI**: Computational Intelligence, a sub-branch of AI that studies adaptive mechanisms to facilitate intelligent behaviour in changing environments
- **GA**: Genetic Algorithm, family of evolutionary computation which models genetic evolution based on a linear vector representing the genotype.
- **EA**: Evolutionary Algorithm, a stochastic search for an optimal solution to a given problem.
- **SI**: Swarm Intelligence, a paradigm of Computational Intelligence that studies the resulting social problem-solving behaviour of organisms in swarms that emerges from the interactions of such agents.
- **FS**: Fuzzy Systems, a paradigm in Computational Intelligence based on fuzzy logic.

1 Introduction

1.1 Motivation

Recently, a lot of research in the field of Computational Intelligence (CI) has been done. In the past years, the IEEE Conference on Computational Intelligence and Games (CIG)[1] has held several international challenges and competitions for research of game agent development. Since games provide dynamic and competitive elements that are germane to real-world problems, they can be used as a challenging scenery for benchmarking methods from CI.

In this project we are going to focus on the Ms. Pac-Man Vs Ghost Team Competition, described in [Williams et al., 2016]. This competition promotes the research into Partial Observability as the participants design controllers for the Ghost Team or Ms. Pac-Man.

We have done some research in the competition held in the past years, both looking at the different agents presented for the Ghost Team and Ms. Pac-Man, and we have analysed the different AI techniques that have been presented.

Some of the techniques that have been used in Pac-Man include the use of different Evolutionary Algorithms to train weights for a neural network to evaluate moves [Lucas, 2005]; temporal difference learning to train a Multi-Layer Perceptron [Burrow and Lucas, 2009]; evolutionary fuzzy systems [Handa and Isozaki, 2008]; Ant Colony Optimization algorithms to find optimal paths for safe routes [Emilio et al., 2010].

For the Ghost Team agents using techniques such as a Monte-Carlo Tree Search to control the ghosts [Nguyen and Thawonmas, 2011] and Swarm Intelligence algorithms together with a GALR [Liberatore et al., 2016] have been developed. Liberatore et al. also compared homogeneous and heterogeneous flocking strategies in their work.

We have developed a new approach based on a Genetic Algorithm to optimize a Case-Based Reasoning system. An overview of the proposed approach to the method is described in the following section.

[1] `http://www.ieee-cig.org` - Last accessed 27 June 2017

1.2 Objectives and approach

This project has two main objectives:

1. To create an algorithm that generates agents for the Ghost Team of the game Ms. Pac-Man. This algorithm will be based on a Genetic Algorithm to optimize a Case-Based Reasoning system of rules.
2. To study how the different parameters of the Genetic Algorithm affect the performance of the algorithm and how they affect the fitness achieved by the generated Ghost Team agents.

1.3 Structure of the project

This project has been divided into 6 chapters, listed below;

- Chapter 1 - Introduction: This first chapter is intended for introducing the project and its main objective, as well as providing its structure. The motivation and the approach taken for this project is presented.
- Chapter 2 - State of the art: This chapter provides a definition for what is AI, presents the different paradigms of Computational Intelligence and Case-Based Reasoning, and gives an overview on Artificial Intelligence applied to games. It also explains in detail the Ms. Pac-Man Vs Ghost Competition for which we are designing our technique.
- Chapter 3 - Genetic Algorithm: In this chapter a detailed description about evolutionary computation and Genetic Algorithms are explained. The goal of this chapter is to analyse and describe, the key concepts that are used in the next chapter.
- Chapter 4 - Description of the designed system: In this chapter the implementation of a Genetic Algorithm and the Case-Based Reasoning system for the Ghost Team of the game Ms. Pac-Man is proposed.
- Chapter 5 - Experimental phase and results: This chapter shows how the Ms. Pac-Man agent that our agents compete against is chosen. After this, we explain how we have selected our individuals and how the tests have been carried out.
- Chapter 6 - Conclusions and future work: The last chapter presents the conclusions that have been drawn out from the project and some lines of future work that we think might be interesting for further development of the project.

2

State of the art

2.1 What is AI?

Giving a definition for Artificial Intelligence has proven to be difficult, and many have been proposed. According to Russel [Russell and Norvig, 1995], four different definitions are given, concerned either about thought processes and reasoning or behaviour; and success measurement in terms of human performance or against an ideal concept of intelligence called rationality.

- Systems that think like humans: "The exciting new effort to make computers think... machines with minds, in the full and literal sense" [Haugeland, 1985]. "The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." [Bellman, 1978].

- Systems that think rationally: "The study of mental faculties through the use of computational models" [Charniak and McDermott, 1985]. "The study of the computations that make it possible to perceive, reason, and act" [Winston, 1992].

- Systems that act like humans: "The art of creating machines that perform functions that require intelligence when performed by people" [Kurzweil, 1990]. "The study of how to make computers do things at which, at the moment, people are better" [Rich et al., 1991].

- Systems that act rationally: "A field of study that seeks to explain and emulate intelligent behaviours in terms of computational processes" [Schalkoff, 1990]. "The branch of computer science that is concerned with the automation of intelligent behaviour" [Luger and Strubblefield, 1993].

Modern definitions differ a bit and still an exact answer to "What is Artificial Intelligence?" is not conceived. The Association for the Advancement of Artificial Intelligence (an international, non-profit, scientific society which is devoted to promote research and responsible use of artificial intelligence as well as increase its public understanding) attempts to define artificial intelligence as: "the scientific understanding of the mechanisms underlying thought and intelligent behaviour and their embodiment in machines."

The National Academy of Science [National Research Council, 2004] proposes the following short summary of artificial intelligence: "One of the great aspirations of computer science has been to understand and emulate capabilities that we recognize as expressive of intelligence in humans. Research has addressed tasks ranging from our sensory interactions with the world (vision, speech, locomotion) to the cognitive (analysis, game playing, problem solving). This quest to understand human intelligence in all its forms also stimulates research whose results propagate back into the rest of computer science, for example, lists, search, and machine learning."

The Turing's Test was developed in 1950 by Alan Turing [Turing, 1950] in an attempt to provide an operational definition of intelligence. The test consists in a computer and a human being interrogated by another human (the interrogator) via a teletype, considering the former to pass the test if the interrogator cannot tell if there is a computer or a human at the other end, that is, if the computer is able to exhibit intelligent behaviour. Turing defined intelligent behaviours as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator.

The computer would need to have the following: Natural language processing for it to communicate successfully, knowledge representation to store information provided before or during the interrogation, automated reasoning to use that information to answer questions and draw conclusions, and machine learning to adapt to new circumstances and detect and extrapolate patterns.

There is also a "total Turing Test", in which a video signal is included so that the interrogator can test the subject's perceptual abilities and for the interrogator to pass physical objects. To pass this test, the computer will also need computer vision to perceive objects and robotics to move them.

2.2 Computational Intelligence

Computational Intelligence is a sub-branch of artificial intelligence, and it can be defined as the study of adaptive mechanisms to enable or facilitate intelligent behaviour in complex and changing environments. These mechanisms include AI paradigms that present an ability to learn or adapt to new situations, to generalize, abstract, discover and associate. Since no one paradigm is superior to the others in all situations, the current trend is to develop hybrid techniques.

Intelligent algorithms in CI include artificial neural networks, evolutionary computation, swarm intelligence, artificial immune systems, and fuzzy systems. Together with logic, deductive reasoning, expert systems, case-based reasoning and symbolic machine learning systems, these intelligent algorithms form part of the field of AI.

2.2.1 Artificial neural networks (ANN)

Brain abilities, such as learning and memorizing have prompted research in algorithmic modelling of biological neural networks. An artificial neuron is a model of a biological neuron. Each of them receives signals from the environment or other neurons, gathers the signals and when fired transmits a signal to all its connected neurons. Input signals are inhibited or excited through numerical weights associated with each connection to the AN.

An activation function controls the firing of ANs and the strength of the exiting signals. AN collect all incoming signals and compute a net input signal as a function of the respective

weights. This net input signal serves as input to the activation function which calculates the output signal of the AN.

An artificial neural network is a layered network of ANs. It may consist of an input layer, an output layer and hidden layers. ANs in one layer are connected fully or partially to the ANs in the next layer, and it is also possible to have feedback connections to previous layers.

Several types have been developed, such as: single-layer NNs, multilayer feedforwards NNs, temporal NNs, self-organizing NNs, combined supervised NNs, convolutional NNs and unsupervised NNs.

Applications of these NNs include disease diagnosis, speech recognition, data mining, image processing, robot control and planning game strategies.

2.2.2 Swarm Intelligence (SI)

A swarm can be defined as a group of agents that communicate with each other by acting on their local environment. The interactions among them result in distributive collective problem-solving strategies.

Swarm intelligence studies the resulting social problem-solving behaviour of organisms (individuals) in swarms that emerges from the interaction of such agents. A more formal definition refers to swarm intelligence as the property of a system whereby the collective behaviours of unsophisticated agents interacting locally with their environment cause coherent functional global patterns to emerge. The interaction among these individuals can be direct or indirect.

The study of computational swarm intelligence has prompted the design of very efficient optimization and clustering algorithms, such as Particle Swarm Optimization and Ant Colony Optimization.

Particle Swarm Optimization (PSO)

PSO is a stochastic optimization approach, modelled on the social behaviour of bird flocks. It is a population-based search procedure where the individuals, referred to as particles, are grouped into a swarm. Each particle in the swarm represents a candidate solution to the optimization problem.

Each particle is flown through the multidimensional search space, adjusting its position according to its own experience and that of the neighbouring particles. A particle, therefore, makes use of the best position encountered by itself and the best position of its neighbours to position itself towards an optimum solution. The performance of each particle is measured according to a predefined fitness function related to the problem being solved.

Applications of PSO include function approximation, clustering, optimization of mechanical structures and solving systems of equations.

Ant Colony Optimization (ACO)

Shortest path optimization algorithms have been developed by modelling the pheromone depositing of ants in their search for the shortest path to food sources. Also, clustering and structural optimization algorithms have been the result of studies of the nest building of ants and bees.

Other applications of ant colony optimization include routing optimization in telecommunication networks, graph colouring and scheduling among others.

Flocking Strategies

Proposed by Reynolds in [Reynolds, 1987], it is a SI technique proposed for the coordinated movement of multiple AI agents. Originally, flocking algorithms were developed to mimic behaviours of groups of beings such as herds of animals or schools of fishes.

A flocking system typically consists of a population of simple agents (boids) interacting locally with one another depending on the distance between them. Three simple steering behaviours are followed: separation, alignment and cohesion. The interactions between the agents lead to the emergence of intelligent global behaviour, unknown to the individual agents [Spector et al., 2003]. Flocking algorithms applications in games can be found in [Scutt, 2002, Rabin, 2010]. Flocking strategies are designed with Genetic Algorithms [Goldberg et al., 1989].

2.2.3 Artificial Immune Systems (AIS)

AIS model some of the aspects of a Natural Immune System (NIS), and they are mainly applied to solve pattern recognition problems, to perform classification tasks, and to cluster data. One of the main application areas of AISs is in anomaly detection, such as fraud detection, and computer virus detection.

2.2.4 Fuzzy systems (FS)

Fuzzy sets and fuzzy logic allow what is referred to as approximate reasoning. With fuzzy sets, an element belongs to a set to a certain degree of certainty. Fuzzy logic allows reasoning with these uncertain facts to infer new facts with a degree of certainty associated with each fact. In a sense, fuzzy sets and logic allow the modelling of common sense.

The uncertainty in fuzzy systems is referred to as nonstatistical uncertainty, and should not be confused with statistical uncertainty. Statistical uncertainty is based on the laws of probability, whereas nonstatistical uncertainty is based on vagueness, imprecision and/or ambiguity. Statistical uncertainty is resolved through observations. Nonstatistical uncertainty, or fuzziness, is an inherent property of a system and cannot be altered or resolved by observations

Applications of fuzzy systems include gear transmission, braking system in vehicles, and traffic signals control.

2.2.5 Evolutionary computation (EC)

It is the subfield of artificial intelligence that studies the algorithms for global optimization inspired by biological evolution. Evolutionary computation refers to computer-based solving systems that use computational models of evolutionary processes such as natural selection, survival of the fittest and reproduction.

An evolutionary algorithm is a stochastic search for an optimal solution to a given problem. The steps of an EA are applied in an iterative way until some condition is satisfied in which the system stops.

Evolutionary computation is explained in depth in Chapter 3, as it is one of the cores of this project.

2.3 Case-Based Reasoning

Case-Based Reasoning (CBR) is a paradigm in automated reasoning and machine learning. The aim of these kind of systems is to solve new problems by comparing them with one or several previously solved problems and by adapting the known solutions instead of coming up with a new one from scratch.

This method is different from other AI approaches in may aspects. Instead of using domain dependent heuristic knowledge, it uses the specific knowledge of previous experiences with problems. An advantage of Case-Based Reasoning is that it implies incremental learning, as new experiences with problems are memorized and available for future situations.

There are two main kinds of CBR: interpretive or classification CBR and problem solving CBR. In interpretative CBR the objective is to decide whether or not a new situation can be treated like previous ones, based on similarities and differences among them. In problem solving CBR, the goal is to create a solution to a new problem based on the adaptation of previous solutions to past cases.

Given a case to solve, the steps to follow describe a case-based reasoner as a cyclic process comprising:

- RETRIEVE relevant previously experienced cases, indexing the cases by appropriate features.
- REUSE the information and knowledge from the retrieved cases to solve the new problem. This may involve adapting the retrieved solutions.
- REVISE the solution after it has been tested.
- RETAIN the parts of this experience likely to be useful in the future, storing them in the case memory.

2.4 AI in games

Pac-Man is considered to be one of the first games including AI in the behaviour of the set of four ghosts that chase you. It had a very simple AI technique: a state machine. Each of the four ghosts can be chasing the player or running away. For each state they take a semi-random route at each junction. In chase mode, each ghost has a different chance of chasing the player or choosing a random direction. In run-away mode, they either run away or choose a random direction. This was in 1979, and from that point until the mid-90s game AI did not change much.

In 1997 Goldeneye 007 was released. Still relying on characters with a small number of well-defined states, it added a sense simulation system: characters could see their colleagues and would notice if they were killed. At that time, sense simulation systems were a trend, with other games such as Metal Gear Solid basing their game design on the technique.

At the same time, RTS games were beginning to take off. Warcraft was one of the first games in which pathfinding was noticed in action (though it had already been used before). Warhammer: Dark Omen (1998) used emotional models of soldiers and robust formation motion.

Creatures (1997) still has one of the most complex AI systems, with a neural network-based system for each creature.

Bots in FPS have seen more interest from academic AI than any other genre. RTS games have co-opted much of the AI used to build training simulators for the military. Sports and driving games have their own AI challenges, such as dynamically calculating the fastest way around a race track. RPGs with complex character interactions still implemented as conversation trees feel overdue for some better AI.

The AI in most modern games addresses three basic needs: the ability to move characters, the ability to make decisions about where to move, and the ability to think tactically or strategically.

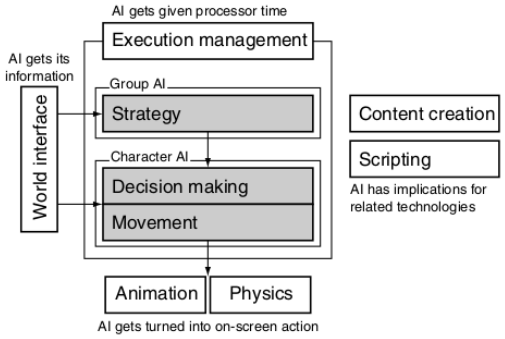


Figure 2.1: AI model. Figure retrieved from [Millington and Funge, 2009]

Figure 2.1 illustrates a model that splits AI in games into three sections: movement, decision-making and strategy. Movement and decision-making contain algorithms for a character-by-character basis, and strategy operates on a team or side. Not all games require these three levels, as, for example, some board games such as chess, require only the strategy level. On the other hand, there are games, such as Jack and Daxter [Naughty Dog, Inc., 2001] where the characters are purely reactive and make their own decisions and act on them, not needing the strategy level.

2.4.1 Movement

Movement refers to algorithms that turn decisions into motion. For example, if an enemy character without a gun needs to attack the player, it then heads directly for the player and when it is close it can attack. The decision to attack is carried out by a set of movement algorithms that home in on the player's location and then the attack animation can be played and the player's health is decreased.

Movement algorithms can be more complex, as, for example, a character may need to avoid obstacles on the way, or work their way through some rooms. An example of this can be seen in Splinter Cell (2002), where an enemy responds to seeing the player by raising an alarm. Walking towards that alarm may involve complex navigation around obstacles or through corridors.

2.4.2 Decision making

It is the process by which a character works out what to do next. Typically, each character has a range of behaviours from where they can choose to perform. The decision making system has to work out which one is the most appropriate at each moment. The chosen behaviour can then be executed using movement AI and animation technology.

At its simplest, characters might have very simple rules for selecting a behaviour, such as moving away or towards from the player when a minimum distance is reached. For example,

at the other extreme, enemies in Half Life 2 (2004) display complex decision making, where they will try a number of different strategies to reach the player, chaining together intermediate actions such as throwing grenades at the player and laying down suppression fire to achieve their goals.

Some decisions need movement AI to carry them out. A melee attack will need the character to get close to its enemy. Other decisions are just handled by animation or handled without any kind of visuals, for example, when an enemy researches new technology in a RTS game.

2.4.3 Strategy

Strategic AI is needed to coordinate whole teams of characters. It can be defined as an overall approach used by a group of characters. Algorithms belonging to this category influence the behaviour of a whole set of characters. Each character in a group has its own decision making and movement algorithms, though, but overall their decision-making is influenced by the group strategy.

In the first Half-Life (1998), enemies worked as a team to surround and eliminate the player, often rushing past the player to take up a flanking position.

2.4.4 Infrastructure

A whole set of additional infrastructure is needed to build AI for a game. Animation and physics simulation are needed to turn into action the movement requests.

The AI needs information from the game to make sensible decisions. This is called perception: working out what each character knows, what it can see, hear and all interfaces between the world and the AI. It is normally the largest proportion of the AI debugging effort.

The whole AI system needs to be managed so that it uses the right amount of processor time and memory. Managing the AI raises a whole set of techniques and algorithms.

2.4.5 Agent-Based AI

It is about producing autonomous characters that take information from the game data, determine which actions to take based on the information, and carry out those actions. It can be seen as a bottom-up design, starting by working out how each character will behave and implementing the AI needed to support that. The overall behaviour of the whole game is a function of how the individual character behaviours work together. Movement and decision making elements make up the AI for an agent.

In contrast, a non-agent-based AI builds a single system to simulate everything, as happens in the traffic and pedestrian simulation in the cities of Grand Theft Auto 3 (2001). The overall traffic and pedestrian flows are calculate based on the time of day and city region and are only turned into individual cars and people when the player can see them.

2.4.6 The Kind of AI in Games

One of the biggest barriers between game AI people and AI academics is what qualifies as AI. AI is equal parts hacking, heuristics and algorithms.

Hacks

For in-game AI, behaviourism is often the way to go. The interest resides in right-looking characters, not in the nature of reality or mind. This means starting from human behaviours and trying to work out the easiest way to implement them in software. Developers rarely build a great new algorithm and then ask themselves, "*So what can I do with this?*" Instead, you start with a design for a character and apply the most relevant tool to get the result.

This means that what qualifies as game AI may be unrecognisable as an AI technique. For some games, complicated cognitive models, learning or genetic algorithms are just not needed, but they just need a simple bit of code that performs an animation at the right point.

Heuristics

A heuristic is a rule of thumb, an approximate solution that might work in many situations but is unlikely to work in all. There is a trade-off between speed and accuracy in areas such as decision making, movement and tactical thinking. When accuracy is sacrificed, it is usually by replacing the search for a correct answer with a heuristic.

A wide range of heuristics can be applied to general AI problem that do not require a particular algorithm. In Pac-Man, the rule of thumb (move in the current direction of the player) works and provides sufficient competence for the player to understand that the ghosts are not purely random in their motion. Another example is found in Warcraft (and other RTS games), where there is a heuristic that moves a character forward slightly into ranged-weapon range if an enemy is a fraction beyond the character's reach. Later, RTS games allowed the player to switch the behaviour on and off.

In many strategic games, both digital or board ones, units are given a single numeric value to represent how good they are. That value is a heuristic that replaces complex calculations about the capabilities of a unit with a single number that can be predefined by programming.

Some common heuristics that appear over and over in AI are:

- Most constrained: Given the current state of the world, one item in a set needs to be chosen. The chosen item should be the one that would be an option for the fewest number of states.
- Do the most difficult thing first: A case of the heuristic above. It is better to do the most difficult thing because it might affect other actions rather than finding that an easy strategy is wasted.
- Try the most promising thing first: If there is a number of options open to the AI, it is often possible to give each one a really rough-and-ready score. Even if the score is dramatically inaccurate, trying the options in decreasing score order will provide better performance than trying things purely at random.

Algorithms

Movement, decision-making and tactical thinking benefit from tried and tested methods that can be reused, but it is important to remember that for every situation where a complex algorithm is the best way to go, there are likely to be at least five where a simpler hack or heuristic will get the job done.

2.5 AI Algorithms in Games

2.5.1 Movement

All movement algorithms have the same basic form: they take geometric data about their state and the world as an input and come up with a geometric output that represents the movement to make.

Often, in older games, characters only had stationary and running speeds, so the output was just a direction to move in. This is kinematic movement; it does not account for acceleration and slowing down.

Recently, there has been greater interest in steering behaviours, that are not kinematic but dynamic movement algorithms. They take into account the current motion of the character, and need to know its velocities and its position. The output forces or accelerations have the aim of changing the velocity of the character.

A kinematic algorithm simply gives the direction to the target, whereas a dynamic one needs to accelerate, and accelerate in the opposite direction when approaching the target.

Kinematic Movement Algorithms

These algorithms use static data, position and orientation, and output a desired velocity. Many games simplify things and force the orientation of a character to be in the direction it is travelling. If the character is stationary, it faces a pre-set direction or the one it was moving in before.

There are two main types of algorithms:

- Seek: It takes as input the character's and its target's static data, calculates the direction from the character to the target and requests a velocity along this line.
- Wandering: It moves in the direction of the character's current orientation with maximum speed. Rapid changes of direction are less likely, but still possible.

Steering Behaviours

They add velocity and rotation to the previous ones. They are separated into fundamental behaviours and behaviours that can be built up from combinations of these. These behaviours are gaining larger acceptance in PC and console game development.

Most steering behaviours have a similar structure. They take as input the kinematic of the moving character and a limited amount of target information, that depends on the application. Many steering behaviours operate on a group of targets, as in the flocking behaviour, that relies on being able to move towards the average position of the flock.

Some basic algorithms include:

- Seek and Flee: Seek tries to match the positions of the character and the target and finds the direction as in the kinematic seek algorithm. Because the steering output is now an acceleration, it will accelerate as much as possible until a maximum that is checked regularly, and it is trimmed back if it exceeds the maximum speed. Flee is the exact opposite of seek.

- Arrive: It slows down the character so that it arrives at the right location when in seek behaviour.

- Align: It tries to match the orientation of the character with that of the character.

Some more complex algorithms are:

- Pursue and Evade: When moving towards a moving target, we need to predict where it will be at some time in the future and aim at that position. The algorithm works out the distance between the character and target and how long it would take to get there at maximum speed. It uses this interval as its prediction lookahead and calculates the position of the target if it continues to move at its current velocity. The calculated position is used as the target position for a seek behaviour. Evade works just as the opposite of Pursue.

- Face: It delegates to the align behaviour for performing the rotation but calculates the target orientation first.

2.5.2 Pathfinding

For moving characters in games that do not know where they need to move until they are ordered to do so, the AI must be able to calculate a route to get from their actual position to their goal. The vast majority of games use path planning solutions based on the so called A* algorithm. Even though it is efficient and easy to implement, it cannot work directly with game level data. It requires that the game level is represented in a particular data structure: a directed non-negative weighted graph.

Another used algorithm for path planning is the Dijkstra algorithm, a simpler version of A*, more often used in tactical decision making than in pathfinding.

Dijkstra

Given a directed non-negative weighted graph and two nodes in that graph, this algorithm generates a path such that the total path cost is minimal among the possible paths from the start node to the goal one. It works by spreading from the start node along its connections, keeping a record of the direction. Once the goal is reached it can follow back the route.

A*

Unlike the Dijkstra algorithm, A* is designed for point-to-point pathfinding and it is not used to solve the shortest path problem in graph theory. The problem to solve is identical to that solved by the Dijkstra algorithm. The algorithm works similar to the Dijkstra, but the nodes choice is done by choosing the node that is most likely to lead to the shortest overall path. "Most likely" notion is controlled by a heuristic.

2.5.3 Decision Making

Most games use very simple decision making systems: state machines and decision trees. Rule-based systems are rarer, but also important. More sophisticated tools, such as fuzzy logic and neural networks have gained more attention. All the discussed techniques are applicable to both intra-character and inter-character decision making.

The process of decision making can be depicted into: the character processes a set of information that it uses to generate an action that it wants to carry out. The input to the decision making system is the knowledge that a character possesses, and the output is an action request. Knowledge can be internal or external. External knowledge refers to the information that a character knows about the environment around it, and internal knowledge is the information about the character's internal state or thought processes.

Typically, the same external knowledge can drive any of the algorithms, whereas the algorithms themselves control what kinds of internal knowledge can be used.

Actions can have two components: they can request an action that will change the external state of the character or only affect the internal state. Changes to the internal state are not so noticeable in games as the external ones, but are significant in some decision making algorithms, as they might correspond, for example, to decisions made based on the emotional state or opinion on the player.

Decision trees

They are used extensively to control characters and for in-game decision making, such as animation control. They have the advantage of being very modular and easy to create, used for everything from animation to complex strategic and tactical AI. Decision trees can also be learned. Random decision trees also exist to add some element of random behaviour to add unpredictability, interest and variation.

State machines

Often, characters in games will carry on doing an action until some event or influence makes them change. The characters will have a set of states that are affected by events that make them move from one state to another.

A game finite state machine can be of many types: hard-coded, hierarchical... Also, they can be combined with decision trees. As decision trees are an efficient way of matching a series of conditions, this has application in state machines for matching transitions. Both techniques can be combined by replacing transitions from a state with a decision tree so that leaves of the tree are transitions to new states.

Behaviour trees

They have become a popular tool for creating AI characters, for example, in Halo 2 (2004). They are a synthesis of other techniques: Hierarchical State Machines, Scheduling, Planning, and Action Execution. Their strength comes from their ability to interleave these concerns in a way that is easy to understand and easy for non-programmers to create.

They are similar to Hierarchical State Machines but, instead of a state, the main building block is a task, that can be something as simple as looking up the value of a variable in the game state. Tasks are composed into sub-trees to represent more complex actions, composed again into higher level behaviours.

2.5.4 Tactical and Strategic AI

Decision making techniques have two limitations: they are aimed for use by a single agent, and they do not infer from the knowledge they have a prediction of the whole situation. Each of these limitations is broadly in this category.

Coordinated Action

Sometimes in games various characters need to cooperate together to get their job done. This can be seen in games from RTS squads or teams of individuals in FPS. Another change in games is the ability of AI to cooperate with the player. This has been normally done by the player giving orders to a bunch of characters that depend on them. Increasingly, we are seeing games in which this cooperation occurs without orders being given, as characters detect the player's intent and act to help them.

This problems are more complex than simple cooperation. AI characters can communicate with each other to tell what they are planning, for example using a messaging system. A player can only indicate their intentions through actions that need to be understood by the AI.

This change in gameplay has brought many different approaches to achieve these goals, such as a Multi-tier AI, that has behaviours at multiple levels. Each character has its own AI and squads have their own set of AI algorithms as a whole. They take group decisions, follow a group movement and have their own group pathfinding algorithms.

2.5.5 Learning

Learning AI has the potential and objective of adapting to the players, learning their techniques and providing a challenge. Through this, more believable and realistic characters can be produced that can learn from the environment and use it. Creating these characters would also reduce the effort of creating game-specific AI, as the characters would be able to learn from the world they reside in.

A first classification divides these techniques into online or offline learning, depending on whether the learning process is carried out during the game, while the player is playing, or if it is carried out "outside" of the game, when it is in development.

Parameter Modification

The simplest learning algorithms are those that aim to calculate numerical values of parameters. These parameters are used throughout AI development in steering calculations, cost functions for pathplanning, weights for blending tactical concerns, probabilities in decision making, and other fields.

Action Prediction

It is usually useful to be able to guess what players will do next. Guesses can be made on which paths are going to be taken, weapons that will be selected, attack planning... A game that can predict the player's actions will offer a more challenging opposition. These guesses can be predicted with probabilities based on previous decisions or choices made by the player.

Decision Learning

Learning is probabilistic, and normally you will have some probability of carrying out an action. Learning these constraints is difficult to combine with learning general patterns of behaviour suitable for outperforming human opponents.

The decision learning process can be simplified into an easy model. The learning character has some set of behaviour options. It also has some set of observable values that can be obtained from the game level. Therefore, we need to learn to associate decisions with observations, and this way, over time, the AI can learn which decisions fit with which observations and improve its performance.

We need to provide the learning algorithm with some feedback to improve performance, and we call this feedback supervision. There two kinds:

- Strong supervision: It takes the form of a set of correct answers. Observations are each associated with the behaviour to choose. The algorithm learns to choose the correct behaviour given the observation inputs.
- Weak supervision: In this case, feedback is given as to how good its action choices are. Most commonly, the feedback is provided by an algorithm that monitors the AI's performance in-game.

Four of the most important decision learning techniques, that have been used in games are: Naive Bayes classification, Decision Tree learning, Reinforcement learning and Neural Networks.

2.6 CIG Ms. Pac-Man VS. Ghosts competition

Several international challenges and competitions have been held throughout the past years in the field of Computational Intelligence for research in game agent development. There are many competitions currently active in the area of games. Some examples are: The Starcraft AI Competition, the Fighting Game AI Competition, the Geometry Friends Game AI Competition, the General Video Game AI Competition, the Angry Birds AI Competition, the Visual Doom AI Competition, the Artificial Text Adventurer and the Ms. Pacman Vs. Ghosts competition. We are focusing in the latest for the purpose of this paper.

Ms. Pac-Man is an arcade game that was released in 1982. Ms.Pac-Man is a revision of the original game, Mr. Pac-Man,that added better graphics, additional mazes, and the most important: new Artificial Intelligence behaviour for the ghosts. This difference has drawn the attention of academics and researchers. The difference between the ghost team in Ms. Pac-Man and Mr. Pac-Man resides in the addition of a semi-random element to the ghost behaviours, making them non-deterministic, unlike in Mr. Pac-Man in which they behave in a deterministic manner. [Williams et al., 2016]

There were two competitions previous to the current one. The Ms. Pac-Man a screen-capture competition which only allowed entries for the Ms. Pac-Man character and provided the agents with a pixel map of the game and requested the direction of travel [Rohlfshagen and Lucas, 2011]; The second one already allowed agents for both the ghost team and the Ms. Pac-Man character, and was based on a simulator that mimicked the original Mr. Pac-Man game. [Lucas, 2007]

The new competition adds a change in the way the agents work: Partial Observability, that increases the challenge. The limited available information about the ghosts makes it difficult for Ms. Pac-Man to plan effectively against the ghost team. The limited information about Ms.

Pac-Man makes it difficult as well for the ghost team, forcing them to behave collectively and communicate in an efficient way to win the game.

2.6.1 Recent research

Pac-Man

Lucas [Lucas, 2005] explored using a simple Evolutionary Algorithm to train the weights for a Neural Network used by the control algorithm to evaluate the possible next moves.

Robles and Lucas [Robles and Lucas, 2009] took an approach to expand a route-tree based on possible moves that the agent can take to depth 40 and evaluate the best path using hand-coded heuristics. A simulator of the game was used to evaluate the game in future ticks.

Burrow and Lucas [Burrow and Lucas, 2009] compared two approaches: Temporal Difference Learning and Evolutionary Algorithms. With these techniques they trained a Multi-Layer Perceptron evaluated within the game. The results showed that the EA performed better than the TDL.

Handa and Isozaki [Handa and Isozaki, 2008] proposed evolutionary fuzzy systems for playing Ms. Pac-Man. The method employs fuzzy logic and the evolution of fuzzy rule parameters. The basic structure of the fuzzy rule is given in advance, and it is later evolved using a (1+1) Evolutionary Algorithm.

Wirth and Gallagher [Wirth and Gallagher, 2008] created an agent based on an influence map model that drives the agent by assigning positive influence to pills and edible ghosts and negative influence to the ghost team. The maximum influence is chosen after checking the map in the four cardinal directions.

Samothrakis et al [Samothrakis et al., 2011] experimented with a Monte-Carlo Tree Search on a 5 player max^n tree representation of the game with limited depth both or Ms. Pac-Man and for the Ghost team. It outperformed previous non-MCTS opponent approaches to the game.

Emilio et al [Emilio et al., 2010] presented an Ant Colony Optimization Algorithm together with a Genetic Algorithm to optimize the parameters of the artificial ants. The ACO had two main objectives with a different ant type used for each of them: one to find paths with points where pills or ghosts can be captured, and another to find safe routes. The first ones are called collector ants and the second ones explorer ants.

An example of a hand-crafted algorithm is the one developed by Flensbak and Yannanakis [Flensbak and Yannakakis, 2008], a rule-based heuristic-search that entered the competition in 2008. The controller was implemented in a screen-capture framework that managed a lot of the functionality, and its main tactics were pill hunting and ghost avoidance.

Foderaro et al. [Foderaro et al., 2012] presented an on-line approach for optimizing paths for a pursuit-evasion problem, where an agent (Ms. Pac-Man) must visit several target positions within an environment while simultaneously avoiding one or more actively-pursuing adversaries (the Ghosts). The maze is abstracted into a connected graph of cells. The methodology utilizes cell decomposition to construct a modifies decision tree, which balances the reward associated with visiting target locations and the risk of capture by the adversaries. As it computes paths on-line, it can quickly adapt to unexpected changes of adversary behaviours and dynamic environments.

Ghost team

Nguyen and Thawonmas [Nguyen and Thawonmas, 2011] presented an application of Monte-Carlo Tree Search to control the ghosts. They perform MCTS on three ghosts on each ghost's tree that represents the game state from the ghost's perspective, whilst using a completely rule-based approach on the other ghost. They won the competition in 2011.

Wittkamp et al [Wittkamp et al., 2008] created a learning system designed for team strategy development in a real time multi-agent domain. They evolve adaptive strategies for the ghosts in simulated real time against a well-performing Pac-Man player. The ghosts are controlled by neural networks, whose weights and structure are incrementally evolved via an implementation of the Neuro-Evolution of Augmenting Topologies algorithm.

Liberatore et al [Liberatore et al., 2016] made use of swarm intelligence to present an application of flocking strategies [Reynolds, 1987]them together with genetic algorithms to control the Ghost Team.

2.6.2 The game

The Ms. Pac-Man game is composed of five agents, Ms. Pac-Man and four Ghost agents: Pinky, Inky, Blinky and Sue. They are shown in figure 2.2 The world consists of a maze environment with non-traversable walls. It has a ghost lair in the centre, where the ghosts start and respawn. The world is filled with a collection of pills placed in corridors for Ms. Pac-Man to collect and four large pills called Power Pills that give Ms. Pac-Man the power to eat the ghosts for additional points.

Figure 2.2: The different characters in the game. Image retrieved from [Williams et al., 2016]

Points in the game are given to Ms. Pac-Man by: eating a pill earns 10 points, eating a ghost earns 200 points for the first one and doubling each time up to 1600 points for the fourth one. The maximum score for a maze with 'n' pills is, therefore: 10n + 4 x (200 + 400 + 800 + 1600).

Restrictions

This new competition adds a restriction to the game: Partial Observability (PO), defined as the impairment of the ability of an agent to completely observe the environment that it is situated within. The method of partial observability implemented in the game is based on LOS (Line-of-Sight), where the agents can see in straight lines up to a specified limit unless there is an obstacle in the way (walls in the maze). Agents can see forwards, backwards and sideways, but not around corners. The game supports a range limit to the sight larger than the longest corridor.

Ghost controllers are given a 40ms shared time budget, equal to the original competition. The ghosts are called sequentially in order (Blinky, Pinky, Inky and Sue), which allows the flexibility to adjust how much time is spent on each ghost in each trick. This flexibility is useful due to the game rules that force the hosts to have no decision ability when not at a junction in the maze.

Messaging

Communication in the competition is heavily controlled by the game to force agents to share information rather than attempt to control the actions of each other. The communication system is composed of two elements, the messenger and the message. Messages asking for location were removed and controllers are allowed to pass on that information spontaneously. Locations are represented as indices of the node graph, with only an integer required.

Messages can be either sent to a single recipient or broadcast to all ghosts on the map. The types of messages that can be communicated can be found in Table 2.1. Possible extension for this system includes more complex messages for communication among ghosts. The messenger system delivers messages at the time specified by a formula. Each message type has its own cost. At present it makes no charge for delivering its messages.

Message Type	**Description**
Pacman Seen	A message informing others that PacMan has been seen.
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.

Table 2.1: Table of messages allowed in Ms. Pac-Man. Retrieved from [Williams et al., 2016]

3

Genetic Algorithms

3.1 Evolutionary computation (EC)

It is the subfield of artificial intelligence that studies algorithms for global optimization inspired by biological evolution. An evolutionary algorithm is a stochastic search for an optimal solution to a given problem.

The main components of an Evolutionary Algorithm are: an encoding of solutions to the problem known as chromosomes; a function to evaluate the fitness of individuals; the initialization of the initial population; selection operators; and reproduction operators. The steps of an EA are applied in an iterative way until some condition is satisfied in which the system stops. The steps are presented in Algorithm 1.

Algorithm 1: Generic Evolutionary Algorithm

```
1  t ← 0
2  P_t ← n initialized individuals
3  while termination criteria is not satisfied do
4  |   EvaluateFitness(I_i) ∀I_i ∈ P_t
5  |   P* ← new empty population
6  |   while P* is not full do
7  |   |   parents ← selectParent(P_t)
8  |   |   offspring ← reproductionProcess(parents)
9  |   |   includeOffspring(P*,offspring)
10 |   end
11 |   t ← t + 1
12 |   P_t ← P*
13 end
```

Different evolutionary computation paradigms derive from the ways in which the components of evolutionary algorithms are implemented. The four fundamental families of algorithms are listed below:

- Genetic algorithms, which model genetic evolution based on a linear vector representing

the genotype [Holland, 1975, Goldberg and Holland, 1988].

- Genetic Programming, based on genetic algorithms, but the population is composed of programs represented as trees [Koza, 1992].
- Evolutionary programming, derived from the simulation of adaptive behaviour in evolution, where the structure that is evolved is based on a finite automata [Fogel, 1962, Fogel and Fogel, 1995].
- Evolutionary Strategies, focused on the numeral optimization in the space of real numbers, geared towards modelling the strategic parameters that control variation in evolution [Rechenberg, 1965, Beyer and Schwefel, 2002].

3.1.1 Chromosome

In an evolutionary algorithm each individual is a candidate solution to an optimization problem. The characteristics of an individual are represented by a chromosome, or genome, and they are the variables of the optimization problem. Each variable to be optimized is referred as a gene, and the assignments of values to these variables are known as alleles.

There are two classes of characteristics: genotypes, which describe the genetic composition of the individuals as inherited from their parents (the possessed alleles); and phenotypes, which are the expressed behavioural traits of individuals in a specific environment.

The efficiency of the search algorithm depends on the representation scheme of candidate solutions. Most Evolutionary Algorithms represent solutions as vectors of a specific data type. Some representation schemes that have been used include integer representations, finite-state representations or tree representations.

3.1.2 Initial Population

The first step of an Evolutionary Algorithm is to create an initial population of individuals. A random selection of values from the allowed domain to each of the genes of each chromosome ensures an uniform representation of the search space, so that no uncovered regions are neglected by the search process.

The size of the initial population is critical. A large number of individuals increases diversity improving the exploration abilities of the population, but increasing the computational complexity per generation, although fewer generations might be needed to locate an acceptable solution. On the other hand, a small population represent just a small area of the search space, but needing more generations to converge.

3.1.3 Fitness Function

Following the Darwinian model of evolution [Darwin, 1859], in which individuals with the best characteristics have a higher chance of survival and reproduction, a mathematical function is used to determine the ability of an individual in an Evolutionary Algorithm to survive and to quantify how good the solution is. The fitness function represents the objective function, which describes the optimization problem. This function maps a scalar value to a chromosome:

$$f : \Gamma^{n_x} \rightarrow \mathbb{R} \tag{3.1}$$

Usually, the fitness function provides an absolute measure of fitness, but, for some applications, such as game learning, it is not possible to find an absolute fitness function. A relative fitness measure is used to evaluate the performance of individuals in a competing population.

There are four types of optimization problems: Constrained, unconstrained, multi-objective and dynamic.

3.1.4 Selection operators

The main objective of the selection operators is to focus on better solutions. A new population is created after the algorithm finishes with a generation for the next one. This new population is selected both from the offspring or the parents and the offspring. The purpose of selection is to ensure that well-performing individuals survive to next generations.

Selection operators are characterized by the selective pressure, the time it requires to produce an uniform population. It is the speed at which the best solution occupies the entire population by the repeated application of the selection operator alone. High selective pressure decreases diversity in the population, which may lead to premature convergence to suboptimal solutions [Goldberg and Deb, 1991, Back, 1994].

One of the most used selection operators is the proportional selection. It biases selection towards the most-fit individuals. A probability distribution proportional to the fitness is created, and the distribution is sampled to select the individuals. As his operator is directly proportional to the fitness value, it has a high selection pressure. In case the goal is to maximize the fitness, the equation that defines the probability of an individual i (I_i) to be selected is Eq. 3.2,

$$p(I_i) = \frac{f(I_i)}{\sum_{l=1}^{n} f(I_l)} \tag{3.2}$$

where $f(I)$ is the fitness function for individual I.

On the other hand, if the goal is to minimize the fitness value, Eq. 3.2 cannot be used, as worst individuals would have a higher probability of being selected. If this is the case, an inverse proportional selection (defined in Eq. 3.3) is needed.

$$p(I_i) = \frac{f^*(I_i)}{\sum_{l=1}^{n} f*(I_l)} = \frac{f_{max} - f(I_i)}{\sum_{l=1}^{n}(f_{max} - f(I_l)} \tag{3.3}$$

A popular method in proportional selection is roulette wheel selection. Fitness values are normalized, and the probability distribution can be seen as a roulette wheel where each slice is proportional to the normalized selection of an individual. Selection can be regarded as the spinning of the roulette wheel, and the slice that ends up at the top is the selected individual. Algorithm 2 describes the process.

Elitism

It is the process of ensuring that the best individuals of the current population survive to the next generation. This means that those individuals are just copied to the new population without being mutated. The result is that the more individuals that survive, the less the diversity.

Algorithm 2: Roulette Wheel Selection

$i \leftarrow 1$
$sum \leftarrow p(I_i)$ (using Eq. 3.2)
$r \sim U(0, 1)$
while $sum < r$ **do**
    $i \leftarrow i + 1$
    $sum \leftarrow sum + p(I_i)$
end
Return individual I_i as the selected individual

3.1.5 Reproduction operators

Reproduction is the process of creating offspring from the selected parent individuals by applying crossover and mutation.

Crossover: It is the process of creating new individuals by combining genetic material randomly selected from parents.

Mutation: It is the process of randomly changing alleles in a chromosome. The main objective is to add new genetic material to the population, increasing genetic diversity. Mutation probability rate is normally set to a low value not to disturb good genetic material in well-performing individuals.

3.1.6 Stopping conditions

The evolutionary algorithm runs iteratively until a stopping condition is met. One of the stopping conditions is to limit the number of generations or the number of fitness function evaluations. A convergence criterion is also usually used together with a limit on execution time. Convergence is defined as the moment when there is no genotypic or phenotypic change in the population.

Some convergence criteria can be: terminate when no improvement is observed over a number of consecutive generations; terminate when there is no change in the population; terminate when an acceptable solution is found; terminate when the objective function slope is approximately zero.

3.2 Genetic Algorithms

In this section we focus on Genetic Algorithm, one of the main kind of Evolutionary Algorithm, as it is the one that will be developed for the purpose of optimizing intelligent non-playable agents.

The canonical Genetic Algorithm, proposed by Holland [Holland, 1975] follows Algorithm 1 in the previous section with the specifics: a bitstring representation, proportional selection as the selection operator for recombination, one-point crossover and uniform mutation. In this original implementation, mutation was considered to have little importance.

3.2.1 Crossover

Three main classes of crossover operators exist:

- Asexual, where one parent produces an offspring.
- Sexual, where two parents produce one or two offspring.
- Multi-recombination, where more than two parents produce one or more offspring.

Although parents are selected, it is possible that they do not reproduce, as recombination is applied probabilistically. Parents have a probability of producing offspring, and it is normally set to a high value.

Two issues need to be attended in selection of parents:

- It might happen that the same individual is chosen as both parents, so this need to be tested for prevention.
- It can also happen that the same individual takes part in more than one application of the crossover operator, so it can become problem in fitness-proportional selection schemes.

The crossover also features a replacement policy, apart from the parent selection and recombination process. Offspring may replace the worst parent. Such replacement can be based on the restriction that the offspring must be more fit than the worst parent, or it may be forced.

Binary Representation

Most of the crossover operators for this kind of representation are sexual. If X1(t) and X2(t) denote the selected parents, the process is shown in Algorithm 3. m(t) is a mask that specifies the bits to be swapped to generate the offspring.

Algorithm 3: Generic Algorithm for Bitstring Crossover

```
1  x̃1(t) ← x1(t) and x̃2(t) ← x2(t)
2  if U(0,1) ⩽ pc then
3  |  Compute the binary mask, m(t)
4  |  for j = 1,...,nx do
5  |  |  if mj = 1 then
6  |  |  |  //swap the bits
7  |  |  |  x̃1j(t) ← x2j(t)
8  |  |  |  x̃2j(t) ← x1j(t)
9  |  |  end
10 |  end
11 end
```

Some operators developed to compute the mask are:

- **One-point crossover:** Holland suggested swapping segments of genes to produce offspring [Holland, 1975]. This kind of operator selects one random point and the bitstrings after that point are swapped between both parents.
- **Two-point crossover:** This operator is similar to the One-point one, but here two positions are selected, and the bitstrings between both points are swapped. The mask is calculated using Algorithm 4.
- **Uniform crossover:** The mask is created randomly, as shown in Algorithm 5. p_x denotes the bit-swapping probability.

Algorithm 4: Two-point crossover mask calculation

```
1 Select the crossover points, ξ1, ξ2 ~ U(1, n_x)
2 Initialize the mask: m_j(t) = 0, for all j = 1, ..., n_x
3 for j = ξ1 + 1 to ξ2 do
4 |   m_j(t) = 1
5 end
```

Algorithm 5: Uniform crossover mask calculation

```
1 Initialize the mask: m_j(t) = 0, for all j = 1, ..., n_x
2 for j = 1 to ξ2 do
3 |   if U(0, 1) ⩽ p_x then
4 |   |   m_j(t) = 1
5 |   end
6 end
```

3.2.2 Mutation

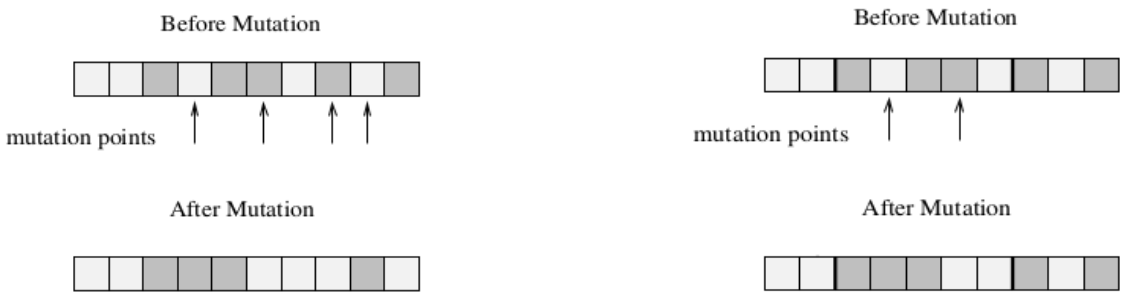
The aim of mutation is to add diversity to the genetic traits of the population, and it is used in support of crossover to assure that the whole range of alleles are accessible to the genes.

Mutation is also applied at a certain rate, usually set to a small value between 0 and 1, for good solutions not to be disturbed.

Binary Representation

Some mutation operators for binary representations have been developed:

- **Uniform mutation:** Bit positions are chosen randomly, and the bit values of those positions are negated. The uniform mutation works as in Algorithm 6 and is illustrated in Figure 3.1a.
- **Inorder mutation:** Two mutation points are chosen and the bits between these points undergo random mutation. The method is depicted in Algorithm 7 and illustrated in Figure 3.1b.



(a) Random Mutation.

(b) Inorder Mutation.

Figure 3.1: Mutation operators for binary representations.

- **Gaussian mutation:** Hinterding in [Hinterding, 1995] proposed, for binary representations of floating-points decision variables, that the bitstring that represents a decision variable could be converted back to a floating-point value and mutated with Gaussian noise. For this process, for each chromosome, random numbers are drawn from a Poisson distribution to determine which genes will be mutated. The bitstrings that represent the genes are then converted. The stepsize $N(0, \sigma_j)$, where σ_j is 0.1 of the range of that decision variable. The mutated floating-point value is then converted back to a bitstring. Hinterding's work shows that this mutation operator provided better results than bit flipping for this representation.

Algorithm 6: Uniform/Random Mutation

for $j = 1, \ldots, n_x$ **do**
  if $U(0,1) \leqslant p_m$ **then**
    $x'_{ij} = \neg \tilde{x}_{ij}(t)$
  end
  $m_j(t) = 1$
end

Algorithm 7: Inorder Mutation

Select the mutation points, $\xi_1, \xi_2 \sim U(1, n_x)$
for $j = \xi_1, ..., \xi_2$ **do**
  if $U(0,1) \leqslant p_m$ **then**
    $x'_{ij} = \neg \tilde{x}_{ij}(t)$
  end
end

4

Description of the designed system

4.1 Case-based Reasoning

The way of applying case-based reasoning to our agents consists of creating a set of 225 rules that the agents will follow depending on the current situation of the game and the states of the agents. We have defined four variables to represent the current state of a ghost in the game:

- Direction: Current direction in which the ghost is moving, obtained by checking the previous move made by the agent.

$$D = \{UP, DOWN, LEFT, RIGHT, NEUTRAL\}$$

- Environment: Number of adjacent nodes. We only consider junctions of more than one node, as one node means it is the only path that can be taken.

$$E = \{2, 3, 4\}$$

- Actor: Different elements that can be in the line of sight of the agent. To get them, we check whether they are visible from the node in which the agent is.

$$A = \{PacMan, PowerPill, Hunter, Hunted, Flash\}$$

- State: Current state in which the ghost is. The ghost can be in hunter mode (default), in hunted mode if it is edible and in flash (or blinking) mode if its edible time is finishing.

$$S = \{Hunted, Hunter, Flash\}$$

We create a file with all the different combinations of these values that is read in the genetic algorithm and that serves as the base of our rules. Each individual in each population will have a copy of this part of the rules that will be completed with the movement to be made during the execution of the genetic algorithm and tested in the game.

$$M = \{UP, DOWN, LEFT, RIGHT, NEUTRAL\}$$

Therefore, a rule will look this when it is completed:

$$Rule = \{Direction, Environment, Actor, State; Move\}$$

4.2 Genetic Algorithm

In this section the different aspects of the implemented Genetic Algorithm will be explained and specified, as well as the connection with the Ms. Pac-Man game.

4.2.1 Individual

Each individual in the population has a 225-long array representing its genotype. Each gen in the chromosome represents one rule, that is, a set of variables that define the state of the Ghost in a specific situation and a movement to be made when the game requires it.

Rules are represented with a left part, including the state of the game depicted in the four variables explained and a right part, which is the move for the agent to execute according to the state of the game. This right part of the rules is initialized with random values of moves, and a HashMap structure is used to store both left and right parts of the rules so they can be easily indexed and accessed.

We can see the representation of rules in Figure 4.1. The first four values, separated by commas, represent the Direction, Environment, Actor and State. The last value, separated by a semicolon, is the Move. The four variables of the left part of the rule, stored in arrays, are common to all individuals. They are the key of the HashMap. The variable in the right part of the rule, the Move, is created in each individual and it is stored in an array. It is the value of the HashMap.

```
{UP,2,Hunted,Hunter;DOWN}
{RIGHT,4,Hunted,Hunter;LEFT}
{NEUTRAL,3,PowerPill,Hunted;NEUTRAL}
```

Figure 4.1: Rules' composition in the individuals.

4.2.2 Population

First of all, our population creates the files corresponding with each of its individuals. For this, it takes the left and right parts of all the rules and prints them in different files, so that we have access to each of the individuals and can later load them for further testing.

The main function of our population is to contain one of the core methods of the algorithm: the evolution. The evolution is in charge of the creation of a new generation, and it controls the selection of parents from the previous generation, the crossover of parents to create children and the mutation of children.

Once a generation has finished, the population stores all the fitness values of each of its individuals, the average fitness of the generation and the best fitness in a file from which we get the statistics for the tests' analysis in the next chapter.

4.2.3 Fitness Function and Game Running

The fitness function can be regarded as the link with the game of Ms. Pac-Man. To evaluate the fitness of an individual we run the Ms. Pac-Man game once per individual per population. The fitness value of each individual is obtained by launching the game and testing the set of rules against the selected Ms. Pac-Man agent.

From within the game, while it is running, we get the values every time that the game requires an action from each of our four ghosts. The first step is to get the values for the four variables that compose the left part of the rules. This is done by using the functions available from the game framework:

- Direction: For getting the direction in which the Ghost is currently moving, we get the last move made by it.
- Environment: For obtaining the environment, that is, the number of accessible nodes, we get the number of neighbouring nodes from the current position of the Ghost.
- Actor: First of all, we check whether Ms. Pac-Man is observable, and if this is the case, the variable actor is set to "PacMan" and its position is stored for a special case not contemplated by the rules. If Ms. Pac-Man is not visible, we check whether any of the PowerPills is. And, if not, we check ghost by ghost if it position are observable and if it is, the state in which it is, and it is the value stored in the variable.
- State: The state of the Ghost is obtained by checking the edible time. If this time has a value higher than zero, it means that either the ghost is being "hunted" or is blinking in "Flash" state, if the time is less than a constant value. If the edible time is zero or less it means that our Ghost is in a "Hunter" state.

Before building the rule, the special case is controlled. We call this special case when either there is no visible actor or when the Ghosts are in a corridor and have no choice for moving but following that path. In this case, if the Ghost at hand is in "Hunted" state it will get a random movement, and if it is in "Hunter" state, it will try to get to Ms. Pac-Man's last seen position.

If we have at hand a normal case, the game is constantly checking the HashMap structure by creating the set of variables that compose the left part of a rule and accessing the movement associated to it in the structure.

Therefore, the game is constantly playing with this rules and adopts an aggressive seeking strategy in case there is no actor visible in the game.

Once the game finishes it returns the score that the Pac-Man agent has obtained after playing against our ghosts. All these fitness values of each population are stored and used for the selection of individuals for the next generation. The objective of our algorithm is to minimize Ms. Pac-Man's score, as it means the less score the better our agent.

4.2.4 Genetic Algorithm parameters

Different parameters control different aspects of the algorithm's performance. Given that we defined the genotype length as the number of rules available, the case the genotype length is fixed to 225.

From the set of parameters used in any GA algorithm, in the experimental phase we are going to establish the values for the population size and the number of generations. For the number of children we have selected one, so that two parents generate only one offspring.

After that we have to choose the selection strategy that the algorithm will use and the type of crossover to perform, as well as its probabilities. We have selected a proportional selection strategy, as it biases selection towards the most-fit individuals. For the crossover we have chosen the two-point crossover.

As for the crossover and mutation probabilities we have performed different tests on their possible values to evaluate the ability of the algorithm to find better individuals, and they are depicted in Chapter 5.

Crossover

We have chosen and implemented the two-point crossover operator for our experiments. The two-point crossover takes both parent individuals and divides them in three sections. The sections of the parents are combined to create a new child individual.

This operator, given that it selects two points instead of one, in the case of the one-point operator, generates a higher solution diversity, so that is why we have chosen it.

Mutation

Our mutation operator selects randomly one gene of an individual and changes its right part of the rule, the movement to make in the next step, to one of the possible moves that can be made.

First of all we make the random selection from the set of possible movements, $M = \{UP, DOWN, LEFT, RIGHT, NEUTRAL\}$, excluding the opposite movement to the one that is currently being followed, as it is not permitted to reverse unless a probability managed by the game does so and reverses all the characters. After this, we filter the available movements by checking that the selected movement is possible to execute according to the available paths in the maze of the level.

Selection

As we want to minimize the fitness of the agents that we are creating, we have chosen and implemented Inverse Proportional Selection for performing our experiments. Inverse proportional selection works as proportional selection, but it biases the selection towards the least fit individuals, which in our case, are the best found solutions. A probability distribution inversely proportional to the fitness is created, and individuals are selected by sampling the distribution.

5

Experimental phase and results

5.1 Ms. Pac-Man agent Selection

First of all, we have to select a Ms. Pac-Man agent in order to test the performance of the genetic algorithms. In order to do that, different Ms. Pac-Man agents included in the Ms. Pac-Man game framework have been taken into account and tested:

- NearestPill: Basic controller included in the competition framework that moves the Ms. Pac-Man agent to the closest pill available not considering the state and position of the ghosts.
- NearestPillVS: More aggressive improvement of the NearestPill Ms. Pac-man agent.
- RandomPacMan: Basic controller that moves the Ms. PacMan agent by getting a random move every time the game requires an action.
- RandomNonRev: Implementation similar to the RandomPacMan but not allowing to move in the opposite direction to the one executed in the previous action.
- StarterPacMan: Original basic controller in the previous competition. Simple state machine based controller. This controller follows a very basic algorithm that makes it avoid close ghosts, chase edible ones and travel to the nearest pill.
- StarterPacManOneJunction: Improvement of the StarterPacman that creates its own ghosts to get lookahead moves.
- POPacman: New implementation that follows a different strategy depending on the sight of the ghosts. If Ms. Pac-Man can see any of the ghosts it will move away from it, if the ghosts are edible it will try to go after them and, otherwise, it will go after the pills and power pills that it can see.

It has not been possible to test our implementation of the Ghost Team against previously developed Ms. Pac-Man agents that participated in the competition in previous years due to the changes that have been applied in the competition in the present year. Trying to test the

game against these agents results in code failures. The change in the game is the application of Partial Observability (PO) based on LOS (Line Of Sight). As these agents are not programmed to support this feature, they fail to execute.

We have performed the different Pac-Man agents tests in the game with the number of generations of the genetic algorithm set to 10 and the number of individuals set to 8. The results of these tests have been used for the selection of the Ms. Pac-Man agent to be used for the final experiments.

After performing the tests, some of the Ms. Pac-Man agents have been discarded due to the low score obtained in all rounds, this meaning that all our solutions outperform these agents, normally because they get stuck and do not behave in a correct way, being not worthy for our experiments. Examples of this errors can be seen in Figures 5.1a, 5.1b and 5.1c.

(a) NearestPacMan. (b) StarterPacMan. (c) StarterOneJunction.

Figure 5.1: Character stuck errors found during testing of the different PacMan agents.

Figure 5.1a shows that the NearestPillPacMan sometimes gets stuck after eating a PowerPill, Figure 5.1b shows the agent StarterPacMan getting stuck in a corner, and Figure5.1c shows the StarterPacManOneJunction getting stuck at the beginning, after the Ghost Team has defeated it the first time. In this last case, the agent stays stuck there until it is eaten or the level is changed after the level time is finished.

Due to the changes in the competition previously mentioned and the fact the the Ms. Pac-Man agent that is used in the competition to test the Ghosts entrants is unknown, we have selected the POPacMan agent for the tests in the next section, as it provides a test for the PO constraint, which is the subject of study of the competition, and makes it more difficult for our agents to minimize its score.

5.2 Individual Selection

5.2.1 Number of generations

The next step is to tune the parameters of the Genetic Algorithm and test them in order to configure the GA in the best way.

First of all we have to define the number of generations of the algorithm. Different tests are launched with the number of individuals set to 8 and the number of generations set to 10, 20 and 30.

As we can see in Table 5.1, the best results in terms of the best found agent (the one that minimizes the score of Ms. Pac-Man) are obtained with 10 generations and 30 generations. The

Analysing the best found agent				
Individuals	Generations	Best	Average	Std
8	10	900	2140.45	469.01
	20	1080	2152.38	544.91
	30	**900**	**2126.67**	**558.66**

Table 5.1: Data of best found agents' fitness in Generations tests

Analysing the averages				
Individuals	Generations	Best	Average	Std
8	**10**	**2716.25**	**3764.49**	**618.97**
	20	2773.75	3790.95	689.49
	30	2785	3802.45	562.49

Table 5.2: Data of average fitness in Generations tests

best individual obtained in all the runs performed has a fitness of 900 in both cases. Looking at the averages of the best individuals across all generations, we see that the best is the 30 generation one.

Analysing the results in terms of the averages across generations of all the obtained fitnesses, in table 5.2 we see that the algorithm gets a better average with 10 generations than with 30, although they are similar. The same happens with the best average of all executions, 10 generations perform better than 30.

Both in the analysis of the best individuals and the averages, the standard deviations are similar, and this indicates that there are no significant differences between the results.

As in Table 5.1 the best individual found in 10 generations has the same fitness value as the one found in 30 generations. For this reason, we conclude that the best number of generations is 10 based on the time needed to complete the execution. A single execution of the algorithm with 10 generations takes two hours and fifteen minutes to complete, while a single execution with 30 generations takes approximately three times more time, around seven hours, so executing it several times for the tests that follow is not feasible.

5.2.2 Number of individuals

Once the number of generations is fixed, the next step is to fix the number of individuals. To do so, we now launch different tests with the number of individuals set to 8, 15 and 30.

As seen in Table 5.3, in terms of the best found agent, the best result is obtained with 30 individuals, but we discard this because the execution takes more than eight hours, so for the purpose of the tests in the next section it is not feasible. The next best found individual is with 8 individuals, as it was in the previous analysis.

In terms of the averages across generations, in Table 5.4 we see that with 8 individuals the algorithm performs better than with 15 or 30 individuals and it is also the fastest in terms of execution time. It has the best average of all executions and the best average of the averages of populations.

Therefore, we conclude that the population size must be set to 8 because this configuration gets the best results in a reasonable execution time.

Analysing the best found agent				
Generations	Individuals	Best	Average	Std
10	8	900	2140.45	469.01
	15	1130	1690.91	401.93
	30	**830**	**1353.64**	**397.35**

Table 5.3: Data of best found agents' fitness in Individuals tests

Analysing the averages				
Generations	Individuals	Best	Average	Std
10	**8**	**2716.25**	**3764.49**	618.97
	15	2933.33	3784.45	401.52
	30	3213.67	3768.45	305.82

Table 5.4: Data of average fitness in Individuals tests

5.2.3 Mutation and Crossover probabilities

The last set of experiments that have carried out study the effects of changing the probabilities of mutation and crossover on the fitness values of the individuals across the generations of the Genetic Algorithm. The results are shown in Table 5.5 and Table 5.6

Analysing the averages				
Crossover	Mutation	Best	Average	Std
0.6	0.1	2752.5	3787.35	619.37
	0.05	**2716.25**	3764.49	618.97
	0.01	2748.75	**3755.83**	**603.99**
0.75	0.1	2902.5	3754.47	509.95
	0.05	**2418.75**	3750.72	607.8
	0.01	2693.75	**3650.91**	**573.15**
0.9	0.1	2611.25	3857.84	781.1
	0.05	**2493.75**	3749.24	699.39
	0.01	2668.75	**3585.91**	**415.89**

Table 5.5: Data of averages with different Crossover probabilities

Overall, the best mutation probability in terms of getting the best agent is 0.05 with the three possible values of the crossover probability. However, the best average of the population is obtained with a probability of mutation of 0,05 if the crossover probability is 0.75; a mutation probability of 0.1 in case the crossover probability is 0.9; and a mutation probability of 0.01 if the crossover probability is 0.6.

When it comes to analysing the mutation probability in terms of averages, the best average of all is always obtained with a mutation probability of 0.05 despite the crossover probability. The best average of the averages of the populations, nonetheless, is always obtained with a mutation probability of 0.01.

Given that the objective of this project is to obtain the best agent, the one who has the least fitness value, to be presented to the CIG Ms. Pac-Man Vs. Ghosts Competition, the selected mutation probability is 0.05 and the selected crossover probability is 0.9.

Analysing the best found agent				
Crossover	Mutation	Best	Average	Std
0.6	0.1	1290	2158.79	432.25
	0.05	**900**	2140.45	469.01
	0.01	1130	**2066.36**	**454.26**
0.75	0.1	1130	2068.79	497.79
	0.05	**1080**	**1982.42**	**448.58**
	0.01	2260	2420	144.22
0.9	0.1	1080	**2033.94**	**596.8**
	0.05	**810**	2082.12	466.18
	0.01	1110	2129.1	448.76

Table 5.6: Data of best found agents' fitness with different Crossover probabilities

6

Conclusions & future work

6.1 Conclusions

After performing all the experiments carried out in Chapter 5, we have come up with a good solution that beats Ms. PacMan fast and gets a small score from it. This solution is obtained with the parameters showed in Table 6.1, and it gets a fitness of 810.

Generations	Individuals	Crossover	Mutation
10	8	0.9	0.05

Table 6.1: Best configuration of the Genetic Algorithm.

Nevertheless we have also realised that a main part of the performance of our agents depend on the stochastic creation of the rules. We can also observe, though, that in general all these values are low, meaning that the objective of minimizing Ms. Pac-Man's score has been achieved.

The second aim of this work was to analyse the effect of the crossover and mutation parameters by performing tests with different probabilities values, has also provided us with some valuable information.

As it has been explained in the previous chapter, the best agent has been obtained with a Crossover probability of 0.9 and a Mutation probability of 0.05. The fact that the best player is found with a high Crossover probability might have to do with the stochastic nature of the agents. If an agent has a high or low fitness value, recombination of the parents might result into a big change in its performance. The worst best players have been found with a crossover probability of 0.75.

We can conclude that the presented proposal, consisting in a hybrid of a Genetic Algorithm and Case-Based Reasoning is a valid algorithm for designing intelligent Ghost Team agents for the Ms. Pac-Man. We have also experimentally proved that the Ghost Team's behaviour generated by this system presents a challenge for the players, making the game more attractive.

6.2 Future Work

Finally, this section provides some future work that must be taken into account to extend, or continue, this work. These future works are focused on the communication amongst the ghost and the exploration of different algorithms that define the ghosts behaviour.

6.2.1 Communication

A line of future work would be to make use of the communication system implemented in the competition this year. Through the use of this system, we may also add to the rules the messages that can be send and received by the Ghost Team, that can be found in Table 2.1.

As stated in the competition presentation [Williams et al., 2016], the communication system needs further improvement, to be done for future editions of the competition. The use of more complex messages will also be of use in our work.

A simple first way of adding this communication system can be by adding two more variables in our actual rules, a message received and the message to be sent. The rule after this modification would look like this:

$$Rule = \{Direction, Environment, Actor, State, ReceivedMessage; Move, MessageSend\}$$

where ReceivedMessage and MessageSend would be:

$$ReceivedMessage = \{PacmanSeen, IAm, IAmHeading\}$$

$$MessageSend = \{PacmanSeen, IAm, IAmHeading\}$$

These messages would imply having the positions of the Ghosts and Ms. Pac-Man in the rules, and, therefore, acting according to them. Knowing that Ms. Pac-Man has been seen from a Ghost's position and knowing its position in the rule would be of use for the other Ghosts to create more complex strategies, and contribute to generate a better case-based reasoning system.

This cannot be done in our implementation now, as moving towards the position of Ms. Pac-Man is only done when no rule is applicable, and its position is not known in the rules. Our algorithm would optimize this new rules so that the Ghosts move towards or away from Ms. Pac-Man when possible without information external to the rules.

6.2.2 Different Algorithms

Another line of future work would be to try another kind of Computational Intelligence algorithm, such as a Particle Swarm Optimization. An example of use of PSO with flocking strategies can be found in the work of Liberatore et al. [Liberatore et al., 2016]. Defining the individuals in such an algorithm as we have done in ours, with a set of rules, is something that we believe is worth trying.

One other option that we have thought about is to create a hybrid between a Neural Network and a Genetic Algorithm. Neural Networks and Genetic Algorithms have been already developed, but in no case it has been implemented with a set of rules of a Case-Based Reasoning system, so it is also definitely something that can be done to look for further improvement of our technique.

Bibliography

[Back, 1994] Back, T. (1994). Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, number 1, pages 57–62.

[Bellman, 1978] Bellman, R. (1978). *An introduction to artificial intelligence: Can computers think?* Thomson Course Technology.

[Beyer and Schwefel, 2002] Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies – A comprehensive introduction. *Natural Computing*, 1(1):3 – 52.

[Burrow and Lucas, 2009] Burrow, P. and Lucas, S. M. (2009). Evolution versus temporal difference learning for learning to play Ms. Pac-Man. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pages 53–60.

[Charniak and McDermott, 1985] Charniak, E. and McDermott, D. (1985). *Introduction to artificial intelligence*. Pearson Education India.

[Darwin, 1859] Darwin, C. (1859). *On the origin of species by means of natural selection*. Murray, London.

[Emilio et al., 2010] Emilio, M., Moises, M., Gustavo, R., and Yago, S. (2010). Pac-mAnt: Optimization based on ant colonies applied to developing an agent for Ms. Pac-Man. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG2010*, pages 458–464.

[Flensbak and Yannakakis, 2008] Flensbak, J. and Yannakakis, G. N. (2008). Ms. Pacman AI controller.

[Foderaro et al., 2012] Foderaro, G., Swingler, A., and Ferrari, S. (2012). A model-based cell decomposition approach to on-line pursuit-evasion path planning and the video game Ms. Pac-Man. In *2012 IEEE Conference on Computational Intelligence and Games, CIG 2012*, pages 281–287.

[Fogel and Fogel, 1995] Fogel, D. B. and Fogel, L. J. (1995). An Introduction to Evolutionary Programming. In *European Conference on Artificial Evolution (AE 95)*, volume 1063, pages 21–33. Springer Berlin Heidelberg.

[Fogel, 1962] Fogel, L. (1962). Autonomous Automata. *Industrial Research*, 4(2):14–19.

[Goldberg and Deb, 1991] Goldberg, D. E. and Deb, K. (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. *Foundations of Genetic Algorithms*, 1:69–93.

[Goldberg and Holland, 1988] Goldberg, D. E. and Holland, J. H. (1988). Genetic Algorithms and Machine Learning. *Machine Learning*, 3(2):95–99.

[Goldberg et al., 1989] Goldberg, D. E., Korb, B., and Deb, K. (1989). Messy Genetic Algorithms : Motivation , Analysis , and First Results. *Engineering*, 3(5):493–530.

[Handa and Isozaki, 2008] Handa, H. and Isozaki, M. (2008). Evolutionary fuzzy systems for generating better Ms.PacMan players. In *IEEE International Conference on Fuzzy Systems*, pages 2182–2185.

[Haugeland, 1985] Haugeland, J. (1985). *Mind Design II : Philosophy, Psychology, Artificial Intelligence*.

[Hinterding, 1995] Hinterding, R. (1995). Gaussian mutation and self-adaption for numeric genetic algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, page 384.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

[Kurzweil, 1990] Kurzweil, R. (1990). *The Age of Intelligent Machines*. MIT Press, Cambridge.

[Liberatore et al., 2016] Liberatore, F., Mora, A. M., Castillo, P. A., and Merelo, J. J. (2016). Comparing Heterogeneous and Homogeneous Flocking Strategies for the Ghost Team in the Game of Ms. Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):278–287.

[Lucas, 2005] Lucas, S. (2005). Evolving a neural network location evaluator to play ms. pac-man. *IEEE Symposium on Computational Intelligence and {...}*, pages 203–210.

[Lucas, 2007] Lucas, S. M. (2007). Ms Pac-Man competition. *ACM SIGEVOlution*, 2(4):37–38.

[Luger and Strubblefield, 1993] Luger, G. F. and Strubblefield, W. A. (1993). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, volume 5th.

[Millington and Funge, 2009] Millington, I. and Funge, J. (2009). *Artificial Intelligence for Games, Second Edition*.

[National Research Council, 2004] National Research Council (2004). *Computer Science: Reflections on the Field, Reflections from the Field*. National Academies Press.

[Nguyen and Thawonmas, 2011] Nguyen, K. Q. and Thawonmas, R. (2011). Applying Monte-Carlo Tree Search to collaboratively controlling of a Ghost Team in Ms Pac-Man. In *2011 IEEE International Games Innovation Conference, IGIC 2011*, pages 8–11.

[Rabin, 2010] Rabin, S. (2010). Artificial Intelligence: Agents, Architecture, and Techniques. In *Introduction to Game Development*, pages 521–557. Charles River Media.

[Rechenberg, 1965] Rechenberg, I. (1965). *Cybernetic solution path of an experimental problem*. Royal Aircraft Establishment, Farnborough.

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34.

[Rich et al., 1991] Rich, E., Knight, K., and Nair, S. (1991). *Artificial Intelligence*. McGraw-Hill Series in Artificial Intelligence.

[Robles and Lucas, 2009] Robles, D. and Lucas, S. M. (2009). A simple tree search method for playing Ms. Pac-Man. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pages 249–255.

[Rohlfshagen and Lucas, 2011] Rohlfshagen, P. and Lucas, S. M. (2011). Ms Pac-Man versus Ghost Team CEC 2011 competition. *2011 IEEE Congress of Evolutionary Computation, CEC 2011*, pages 70–77.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*.

[Samothrakis et al., 2011] Samothrakis, S., Robles, D., and Lucas, S. M. (2011). Fast Approximate Max-n Monte-Carlo Tree Search for Ms Pac-Man. *IEEE Trans. Comp. Intell. AI Games*, 3(2):142–154.

[Schalkoff, 1990] Schalkoff, R. J. (1990). *Artificial Intelligene: An Engineering Approach*. McGraw-Hill, New York.

[Scutt, 2002] Scutt, T. (2002). Simple swarms as an alternative to flocking. In *AI Game Programming Wisdom*, pages 202–208. Charles River Media.

[Spector et al., 2003] Spector, L., Klein, J., Perry, C., and Feinstein, M. (2003). Emergence of Collective Behaviour in Evolving Population of Flying Agents. In *Genetic Programming and Evolvable Machines*, volume vol. 6, pages pag. 61–73.

[Turing, 1950] Turing, A. (1950). Computing Machinery and Intelligence. *Mind*, 59(236):433–460.

[Williams et al., 2016] Williams, P. R., Perez-Liebana, D., and Lucas, S. M. (2016). Ms. Pac-Man Versus Ghost Team CIG 2016 Competition. In *2016 IEEE Conference on Computational Intelligence and Games, CIG 2016*, pages 1–8.

[Winston, 1992] Winston, P. H. (1992). *Artificial Intelligence*, volume 110.

[Wirth and Gallagher, 2008] Wirth, N. and Gallagher, M. (2008). An influence map model for playing Ms. Pac-Man. In *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, pages 228–233.

[Wittkamp et al., 2008] Wittkamp, M., Barone, L., and Hingston, P. (2008). Using NEAT for continuous adaptation and teamwork formation in pacman. In *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, pages 234–242.