Final Year Project Report

Full Unit - Final Report

Cooperative Strategies in Multi-Agent Systems

James King

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Kostas Stathis

Department of Computer Science

Royal Holloway, University of London

March 22, 2019

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 1,587 *2,000* intro + n *3,000* lit review + n *4,000* framework + n *4,000* experiment evaluation + n *1,000* conclusions + n *1,000* professional issues

Student Name: James King

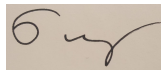Date of Submission: March 22, 2019

Signature:

Table of Contents

Abstract

Chapter 1: **Introduction**

Artificial intelligence (AI) has been an idea present in the consciousness of humanity for millennia. From Hephaestus' mighty Talos to Edgar Allan Poe's commentary on 'Maelzel's Chess-Player' the idea has inspired both awe and confusion. As we move away from the mythical and the false, AI embeds itself deeper into our lives and societies. AI techniques are being used for many novel applications in areas such as medicine [28] and game playing [23].

Many of these applications are specifically using agent techniques [52, 26, 11]. Intelligent agents (IAs) have many definitions but one popular definition is that agents are anything that perceives and acts upon its environment [48]. Agents are situated in an environment and many of these agents can be combined in one environment to form a multi-agent system (MAS).

1.1 **Motivation**

Within MASs it is generally possible for the agents to interact and/or to communicate with each other. Communication generally occurs through agent communication languages (ACLs). The interactions that occur between agents are actions by one or all of the agents in the interaction. There is a possibility that agents can act out of pure altruism and always cooperate with other agents. However, often agents will work to protect their interests i.e. they are selfish.

It is desirable for agents in a MAS to work together to complete tasks and fulfil goals. Therefore, we want to facilitate cooperation between agents which may be selfish. There are analytical tools that we can use in game theory to understand what happens when decision-making individuals interact [37]. Some of these tools are used to explain how cooperation occurs between selfish individuals.

Many of these tools rely on the idea of reciprocal altruism [60], which stipulates that cooperation can occur between selfish individuals if they expect cooperation to be reciprocated. Two such mechanisms are direct and indirect reciprocity. Indirect reciprocity refers to when an agent cooperates with another agent with the expectation that this cooperation will increase the chance of receiving cooperation from others later. Direct reciprocity on the other hand, refers to the expectation of reciprocating cooperation from the agent who initially received the cooperation.

There are many factors to consider surrounding the mechanism. One key factor is the level of visibility of the interactions within the MAS. Nowak and Sigmund [42] limit this visibility to a randomly selected group of individuals in the population (onlookers). Another factor to consider is how information about the interactions can be conveyed. Sommerfeld *et al.* suggested the use of gossip [54].

So how can we use these mechanisms to facilitate cooperation between IAs? What options can we use and how can we make use of them to encourage cooperation between these agents?

1.2   Aims and Objectives

The high level aim of this project is to study how game-theoretic techniques can be leveraged in MASs to create cooperative societies of agents. Derived from this high level aim the objectives of this work are to develop a theoretical framework/model using game-theoretic and MAS techniques inspired by past work in both fields. The next is to implement this framework in a MAS that allows transparent decision making by agents; by which I mean that agents are able to give reasons for their decisions. Another objective is to build this implementation on a distributable platform that allows users to set up games of the model, specify certain variables and view an analysis of the game in order to study how the mechanism and variables affect cooperation in the system.

A feature of the system will be that users are able to create an account and refer back to games they have previously run. I will be using the system and specifically this feature to run my own experiments in order to study the effectiveness of the techniques I have employed and to discover how the system needs to be set up in terms of the variables present. I will then present these experiments and discuss the results with a conclusion on my findings and suggestions of future work as a result of these findings.

Objectives:

1. Develop a theoretical framework

2. Implement this framework

3. Run experiments using the implementation

4. Analyse the experiments and evaluate the results

1.3   Contribution

Much of the work I have reviewed in relation to game-theoretic mechanisms has come from the field of game theory. Because the studies have come from game theorists the implementation of the models devised by the authors have not generally used MAS techniques. My implementation is not only a MAS using a game-theoretic model, but it supports transparent agent decisions due to the use of the logic programming language Prolog for the implementation of agents' decision making components. This implementation is also a web based system, allowing users across the world to experiment with the model I have devised.

Furthermore, the theoretical framework I have designed is inspired by past work on game-theoretic mechanisms, but also builds upon these approaches. One way in which I build upon these is by combining the many aspects of the past approaches such as using MAS techniques to create a game-theoretic model (combining direct and indirect reciprocity) and using gossip as an action to convey reputation information in a mixed reciprocity model.

Lastly, in my system, users associate agents with specific strategies which I have included in the system. These strategies come from past work in the topic of game theory, but are implemented using an agent architecture which I have purpose-designed for my theoretical framework. This arhcitecture included augmenting the original strategies with strategy components for whatever roles agents may have at a particular time-point in a society (such as when an agent is not in an interaction). Another development is the trust models which I have created for the original strategies for interpreting the events in the environment.

1.4   Structure

In the next chapter of this report I have explored and described the past work that has been a central inspiration to the theoretical framework I have devised. I have included other interesting information and possible mechanisms from past work that could be used to encourage cooperation in MASs in the appendix. This background reading then leads into the next chapter in which I have defined the theoretical framework for my system.

Following on from the theoretical framework in the same chapter, I have described the implementation of this theoretical framework in a web application. This application is available for users across the internet to set up games and view the events and analysis of their games. In the next chapter, I use this platform to experiment in order to review the effectiveness of the mechanism, strategies and variables in the game in encouraging cooperation between selfish agents.

The final two main chapters of this report discuss my findings from the experiments and rest of the project. I then conclude and evaluate the project. The appendix is at the end and contains further information referenced in the report but which is not necessarily central to the project.

Chapter 2: **Background**
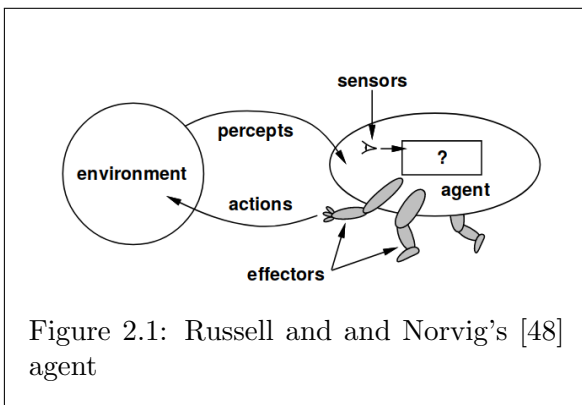
2.1 Introduction

In the following sections of this chapter I will be exploring and describing past work relevant to this project including MAS techniques and related studies in game theory. First I will describe MAS techniques and concepts that are important to this project including subsections on agents, an agents environment and agent communication. Secondly I will discuss game-theoretical approaches to modelling and studying multi-agent interactions in regards to the evolution of cooperation in MASs.

2.2 Multi-Agent Systems and Intelligent Agents

MASs are a field in computer science concerning the development of societies of agents in which agents are computer systems are computer systems that are capable of deciding on what actions to take in order to reach a delegated goal or the agent's design objectives [61]. These agents typically interact using agent communication languages (ACLs).

2.2.1 Environments

Russell and Norvig [48] state that an agent is designed to work in an environment class or a set of environments. According to Wooldridge [62] an agent is built to fulfil design objectives. These design objectives work within the restriction of the agents environment. Following from this Russell and Norvig [48] define an agent in terms of it's environment, stating that an agent is any system that uses sensors to perceive it's environment and effectors to act upon it's environment.

Figure 2.1: Russell and and Norvig's [48] agent

A generic program was designed by Russell and Norvig [48] for how an environment program should run in which there are steps that run in a cycle. In the first step, perception of the environment occurs for each agent in which the agent can use it sensors to receive percepts from the environment. In the next step agents all decide on an action. After all have decided the state of the system is then changed based on the actions that the agents have decided on. A fourth step was added to update the performance scores of each agent.

The effect of these cycle steps is to keep agents synchronised preventing agent actions from changing the state of the environment while other agents are deliberating. A static environment is one that does not change whilst an agent is deliberating. Dynamicity is one of Russell and Norvig's [48] five properties of an environment. The synchronicity of this generic environment program helps keep the environment static, unless other forces (not agent actions) have an effect on the environment's state.

The opposite to a static environment is a dynamic one, in which the environment can change during decision making. Another environment property is accessibility. An accessible environment is one that gives an agent full access to the state of that environment. In an accessible environment there is no need for an agent to keep any internal state about the environment. Unlike in inaccessible environments, in which an agent cannot simply sense all aspects of that environment that are relevant for decision making.

A third property from Russell and Norvig [48] is whether the environment is episodic or not. An episodic environment is split into episodes where the quality of actions in each episode do not depend on previous episodes.

These properties determine how complex an environment is. The final two are whether the environment is deterministic or not and whether the environment is continuous or discrete. If an environment is deterministic it means the effects of an action given some conditions is guaranteed. If an agent can access the whole state of the environment (accessibility) and the environment is deterministic the agent can guarantee the outcome of their action.

An environment is considered discrete actions and percepts are clearly defined and there are a limited number of them. The more properties of an environment that match the following properties, the more complex the environment: continuous, non-deterministic, inaccessible, dynamic, non-episodic.

2.2.2   The Agent Model

Yoav Shoham introduced the agent-oriented programming (AOP) paradigm [51]. This paradigm is similar to object-oriented programming (OOP). In Shoham's formulation of AOP [50] object units are replaced agents, the internal state of a unit is constrained to beliefs, commitments and capabilites, message passing is constrained to using speech act theory to communicate and constraints are placed on methods.

What is an agent?

Since Shoham's formulation there has been much debate as to whether systems are agent systems or simply programs [17]. The exact definition of agency varies depending upon which source is consulted and the application of the concept.

Wooldridge and Jennings [62] gave one of the most influential and powerful descriptions. They put forward a weak notion of agency by laying out four main properties: autonomy, social ability, reactivity and pro-activeness.

Autonomy refers to agents having control over their actions and internal state, and operating without outside intervention. Social ability is the use of an ACL to interact with other entities such as humans and other agents. Responding to environment changes in a timely fashion is what the property of reactivity refers to. Finally pro-activeness is the display of taking initiative to work towards a goal, not simply in reaction to environmental state changes.

Wooldridge and Jennings go on to note that stronger notions of agency endow agents with human-like mental notions. These mental notions represent information in a symbolic model, and make decisions by reasoning with this symbolic model in a symbolic way. This representation and reasoning is no easy feat - known as the representation/reasoning problem [61] - but it does allow agents to make explainable and transparent decisions.

A more up to date and stronger notion of agency may include the use of new developments such as computer vision, natural language processing and deep reinforcement learning concepts. Stronger notions of agency show a tendency towards more humanlike characteristics and AI concepts, drawing clear comparison with a long term goal of AI: to produce systems capable of beating The Imitation Game [31].

Russell and Norvig's Agent Program Components

Russell and Norvig [48] formulated a number of agent program models to describe the internal make up of agents that allow an agent to act in and understand their environment.

The first model is simple reflex agents which keep no internal state, they simply use their percepts from the environment and condition-action rules to decide on actions. These are comparable to reactive agent architectures. Model-based reflex agents use percepts from their environment to form a state which to use alongside condition-action rules and a model of how the state of their world evolved to make decisions.

Goal and utility-based focus on the drives of an agent. The goal based architecture keeps an internal state, and combines this with search and planning decision making components to reach specified goals. States in which an agent is more successful can be formulated in terms of having a higher utility. Utility based agents work to maximise that utility, even in situations when an agent has conflicting goals in which the utility is formed by creating a tradeoff between the conflicting goals.

Learning Agents

Learning agents is the final agent model described by Russell and Norvig. Programs created using this model are not formed by a programmer specifying how to act. Instead these agents use a number of components to feed into a decision making system (performance element). Feedback is provided by the critic component to the learning element which makes changes to the performance element. The problem generator is another component that suggests actions that will hopefully lead to new knowledge.

Machine learning is exploding with new concepts and developments, most notably from the agents systems persepctive deep reinforcement learning. Agents using machine learning techniques often appear to satisfy stronger notions of agency. However, these techniques do not come from a symbolic AI background, and lead to a lack of explainability.

Unifying these techniques with symbolic AI approaches to produce agents which can explain their decisions in order to produce verifiable systems is a keenly researched topic [21, 19]. However, this is an even tougher version of the representation/reasoning problem as we are not only trying to reason with symbolic data, but reason in a complex manner with deeply mathematical techniques.

Deductive Reasoning Agents

Deductive reasoning agents use a form of symbolic AI. The symbolic representation of information is in the form of logical formulae and the reasoning component is done through logical deduction or theorem proving [61].

Shoham's AOP paradigm [51] and Agent0 language [50] were developed to facilitate the creation of deductive reasoning agents. The symbolic representation was formulated in terms of beliefs, commitments and capabilities and agents were given a set of commitment rules to

reason about what commitments to make.

Concurrent MetateM is another language for the creation of deductive reasoning agents [15]. The language is known for focusing on creating agents that work by theorem proving. Programmers essentially create deductive theorem provers using temporal logic that dictate agent behaviour.

Deductive reasoning agents suffer from two issues: the transduction problem and the representation/reasoning problem. The transduction problem refers to the difficult in translating from the real world into an accurate and adequate symbolic representation. The representation/reasoning problem are the difficulties in representing symbolic information and reasoning about that information.

Practical Reasoning Agents

Symbolic AI approaches to agency aren't limited to deductive reasoning agents. Practical reasoning approaches focus reasoning towards actions, whereas theoeretical reasoning focuses on beliefs [61]. However this does not mean that practical reasoning models do not incorporate mental notions such as beliefs.

Both deductive reasoning languages such as Agent0 and practical reasoning focused languages and frameworks such as GOAL [24] and PRS (The Procedural Reasoning System) [20] have used the idea of beliefs to base actions on.

PRS uses the belief-desires-intentions (BDI) model [10]. BDI agents keep beliefs in a belief base, these are the information an agent sees as facts about the world and agents are capable of inferring beliefs based on the presence of other beliefs.

Desires are the drives behind an agent's actions. A desire is an objective or state an agent wishes to reach. When an agent is actively pursuing a desire it becomes a goal. These goals must be consistent. When an agent has committed to a desire this desire becomes an intention. Commitment to a desire is when an agent has begun executing a plan towards that desire. These plans are sequences of actions.

Rodriguez *et al.* [47] criticised agent-programming languages (APLs) for forcing the BDI agent architecture upon users of the language, where agent architecture refers to Pattie Maes' description [32]. Their claim is that the use of BDI constrains agents to be endowed with reasoning capabilities that do not always apply to some problem instances. The language SARL - specified by Rodriguez *et al.* [47] - is a more generic language which aims to be architecture independent.

Reactive and Hybrid Architectures

Applications of agent technologies in robotics and embedded systems require architecures and programs to work in resource bound environments. Brooks [12] argues for intelligence as an emergent property of complex systems.

Brooks criticised the use of symbolic AI in the creation of agent systems [12]. He claimed that the abstract reasoning and explicit representations of symbolic AI aren't required for the development of intelligent agents as intelligence is an emergent property of complex systems. In resource bound environments this abstract reasoning and explicit representation are often too resource hungry to be applicable.

In reaction to this Brooks devised the subsumption architecture. In this architecture

task-accomplishing behaviours map from percepts to actions, in order to accomplish tasks. These are also known as behviour modules. These modules are able to decide on actions at the same time as each other. Brooks employs a hierarchical structure to select which behaviours decision to use. The lower the layer a behaviour belongs to the greater it's priority.

$$K_1 \rightarrow a_1$$
$$K_2 \rightarrow a_2$$
$$\cdots$$
$$K_m \rightarrow a_m$$

Table 2.1: An example teleo-reactive program

This hierarchical selection of actions is notably similar to teleo-reactive programs [38]. These are programs which contain an ordered set of production rules. These rules map from conditions to actions, and the ordering of them defines which actions are more important.

A criticism of reactive architectures [61] is that through lack of internal state an agent using these architectures can only view the local environment, forcing a short-term view. Hybrid architectures have been developed to combat these issues. One method in which to do this is to use free variables in teleo-reactive program conditions, with which beliefs bind to at run-time.

2.2.3 Multi-Valued Fluent Cached Events Calculus

The aim of the event calculus is to reason about time periods and local events [27]. Events occur at specific timepoints, these events initiate and terminate periods of time for which a fluent holds. For example at time $t1$ event $e1$ occurs, fluent f did not hold prior to $e1$, but $e1$ initiated it and as such it holds after $t1$. A second event $e2$ then occurs at timepoint $t2$ where $t2 > t1$, this event terminates f. Following this event f no longer holds, but f still holds in between $t1$ and $t2$.

The multi-valued fluent cached events calculus (MVFCEC) is an implementation of the events calculus [26]. The implementation improves the speed of querying fluents using a caching system. In Kowalski *et al.*'s [27] original event calculus fluents were considered Boolean. Take fluent F, in the original event calculus $F = true$ if it holds or $F = false$ if not. With multi-valued fluents $F = V$ where V is a value specified by the programmer.

Agents often need to reason about time and states. Many of these agent systems are developed using a logic-based system. The events calculus is a very powerful logic-based tool for this purpose and as such variants of the event calculus have been used in the development of many MASs [6, 26].

The Agent0 belief system reasons about the initiating and terminating of beliefs with certain values through time periods. This belief system is an example of when agents require reasoning about time, and it seems natural to use MVFCEC for the representation of a beliefs system.

2.2.4 Agent Communication

To build cooperative and coordinated societies agent communication is extremely important. In fact it's so important that Wooldridge and Jennings include social ability as one of his four key properties [62]. In the AOP paradigm agents communicate socially via a message passing system.

Many of these languages use speech act theory. Speech act theory uses the idea that

communication is an action often to achieve goals and intentions [7]. Both KQML [14] and FIPA ACL use performative verbs to specify the intent of the communication. Both have different performative verbs, but in combination with the message content these languages communicative acts generally fall into one of the five categories of speech acts described by Searle [49].

In human communication we generally have a shared understanding of certain concepts. Such as the definition of a book or plant. In KQML and FIPA ACL shared ontologies are used to define the meaning of commonly used vocabulary of certain subject domains in order to facilitate shared understanding of message content.

The theory of communication is complex. According to Singh [53] a key limitation of existing ACLs (including KQML) is their focus on mental agency over social agency. This is the focus on using the mental state of an agent such as communicating about the beliefs of an agent. Singh claims that this supposes that agents can read each others' mind.

2.2.5   Multi-Agent Interactions

Strategic interaction, rationality, payoff matrices, dominant strategies, nash equilibrium, pareto optimality, social welfare maximisation

Agents are able to interact through communicative actions and by actions in the environment. In Jenning's [25] view of multi-agent interactions, interactions between agents often occur but are not limited to when the agents are linked by organisational relationships. Organisational relationships are subject to change. These agents have their own spheres of influence in an environment, which may overlap each others. An overview of this setup can be seen in figure 2.2.

Figure 2.2: Jennings' [25] MAS conceptualisation
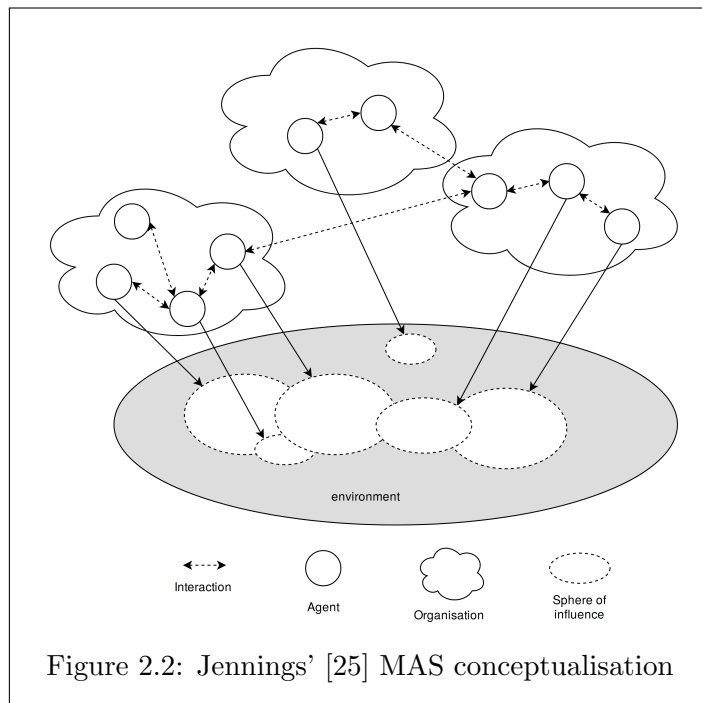
When spheres of influences overlap an agent may need to make a decisions that is dependent upon other agents decisions. These overlaps can be described as interactions. Agents in these interactions are assumed to be self-interested, and want to improve their utility. Utility is the the representation of how beneficial events and actions in a system that transform the state of that system are to agents. Game-theoretic techniques can be used to model and simulate agent interactions using these utility functions.

2.3   Game Theory

2.3.1   Introduction

2.3.2   Cooperative Phenomena

Game theory formulates mathematical models to study conflict and cooperation between intelligent rational decision-makers [37]. In these mathematical models actions are often defined in terms of cooperation and defection. These models apply to more than just multi-agent interactions, including the natural world.

Early evolutionary theory struggled to explain why cooperation is so prevalent in nature. In fact, it seemed that competition was key to evolution as individuals were competing to survive; the notion famously coined by Herbert Spencer "Survival of the fittest" [55].

Axelrod and Hamilton [8] note two key areas of study that attempt to explain cooperative phenomena in the face of competitions: Kinship Theory and Reciprocal Altruism. These two theories are useful for explaining cooperation in nature, and as such, it may be possible to apply them to MASs in order to facilitate the evolution of cooperation between agents. Due to reasons explained in my appendix in subsection 8.1.1 I will be focusing on reciprocal altruism. edit appendix entry to reflect this

2.3.3   Reciprocal Altruism

Figure 2.3: Cleaning symbioses such as that of the green sea turtle and surgeon-fish including the yellow tang show that reciprocal altruism is possible between non-humans and even interspecies [64]

Reciprocal altruism is an idea most famously put forward by Robert L. Trivers [60]. Trivers defines altruism as behaviour of one organism that benefits another to whom it is not closely related, while being apparently detrimental to the organism performing the behaviour. From this definition and from Trivers' description we can draw the meaning of reciprocal altruism to be altruism-based on the idea that the altruistic act will be returned.

This idea is a move away from limiting individuals to cooperating with only their kin - a key limitation of Kinship Theory - and towards any individual that they believe their cooperation will be reciprocated by. Axelrod and Hamilton [8] noted this concept as advantageous in explaining cooperation between unrelated individuals, such as is common between humans. I would argue that this concept is also more applicable to higher intelligence societies such as those possible from MASs.

2.3.4   Axelrod, Hamilton and The Iterated Prisoner's Dilemma

Figure 2.4: A fomulation of the Prisoner's Dilemma payoff matrix [34] similar to Rapoport *et al.* [45]
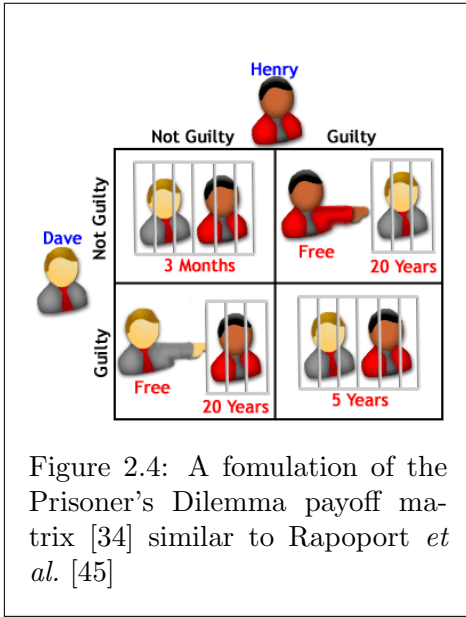
The prisoner's dilemma was described by Rapoport *et al.* [45] in terms of criminals under interrogation, but Axelrod and Hamilton formulated it in terms of cooperation and defection. This formulation allows the study of a more generic form of interaction betwenn individuals.

In a single round of the prisoner's dilemma two individuals simultaneously choose to either cooperate or defect. After each interactions each individual receives a payoff in the form of a number, the higher the payoff the better. This payoff is dependent upon the actions of both individuals and is illustrated in table 2.2.

From an agents perspective we can view the payoff in terms of utility or what is preferable to an agent [61]. Using player A and player B, with u_i being the utility function for agent i, and the payoff matrix provided by Axelrod and Hamilton [8] we have the following utility functions: $u_A(D,D) = 1$, $u_A(D,C) = 5$, $u_A(C,D) = 0$, $u_A(C,C) = 3$ for player A and $u_B(D,D) = 1$, $u_B(D,C) = 0$, $u_B(C,D) = 5$, $u_B(C,C) = 3$ for player B.

We can order the possible situations by utility for each agent to understand what situations are preferable to each agent. For player A: $(D,C) \geq (C,C) \geq (D,D) \geq (C,D)$. For player B: $(C,D) \geq (C,C) \geq (D,D) \geq (D,C)$. As you can see players are at odds as to which situations they wish to arise. If both attempt to bring about the situation that is preferable to them they will both receive only 1 as payoff each.

| | | Player B | |
		Cooperation	Defection
Player A	Cooperation	A=3 B=3	A=0 B=5
	Defection	A=5 B=0	A=1 A=1

Table 2.2: The payoff matrix in a typical iterated prisoner's dilemma game (such as Axelrod and Hamilton's [8]). A=x, B=y where x denotes the payoff for A and y denotes the payoff for B.

In the iterated prisoner's dilemma the two agents repeat these rounds and can base their decisions on previous rounds. This allows individuals to use strategies that make use of a mechanism known as direct reciprocity [41], which is a type of reciprocal altruism.

Axelrod and Hamilton [8] used games of the iterated prisoner's dilemma in their round-robin tournaments. These tournaments had a set of players that are associated with strategies. Every player entered into a game against every other player, in which rounds were repeated. For cooperation to be rational in the iterated prisoner's dilemma the 'shadow of the future' needs to present in each round [61], so games were not cut off at a set point but ended by increasing the probability the game won't continue each round.

Axelrod and Hamilton [8] combined these round robin tournaments with a genetic algorithm [35] (genetic algorithms are discussed in subsection 2.3.11). One aim of Axelrod and Hamilton's [8] paper was to review how successful the strategies submitted to them were.

The use of a genetic algorithm allowed them to ask three questions of each strategy. Is it robust? Is it stable? Is it initially viable?

Robustness refers to the ability to thrive in an environment with a variety of strategies. Stability refers to the ability to - once fully established - resist invasion by mutant strategies. Initial viability refers to whether or not a strategy can establish itself in a non-cooperative environment.

Axelrod and Hamilton [8] found two strategies with these three abilities: 'tit-for-tat' and 'all defect'. Later on, Nowak and Sigmund [39] found that 'Pavlov' ('win-stay, lose-shift') also has these abilities. The interesting part of 'Pavlov' and tit-for-tat is that they are nice strategies (they begin by cooperating) and that they also actively aid in the evolution of cooperation.

Nowak [41] found that for the evolution of cooperation to occur, the cost-to-benefit ratio of the altruistic act must be less than the probability of another encounter for cooperation to evolve: $w > c/b$. If the probability of subsequent encounters between two individuals is low, then it is very likely that cooperation will not evolve as there will be no sufficient reward. With the advent of huge networks spanning across the world and the drastic increase in devices across these networks, it is highly likely MASs will operate with IAs that are unlikely to re-meet.

Axelrod and Hamilton's program was not created as a MAS. However, the importance of remeeting is a definite limitation to the use of direct reciprocity in MASs, and as such I do not see direct reciprocity as a strong contender to encourage cooperation on its own. One required property of a mechanism to encourage cooperation is that it must work when both re-meeting is unlikely and when it is likely. As such, direct reciprocity is not completely inadequate for the problem. However, it is insufficient when not combined with some encouragement for when chances of meeting again are low.

Extensive interest in reciprocal altruism in recent years has been focused towards direct reciprocity, including a number of available libraries such as the Axelrod Python library [13]. I argue that to gain a better understanding of how we can work to facilitate cooperation between IAs, we must look at a wider berth of options.

2.3.5   Indirect Reciprocity

Figure 2.5: The main idea of indirect reciprocity from Nowak [41]
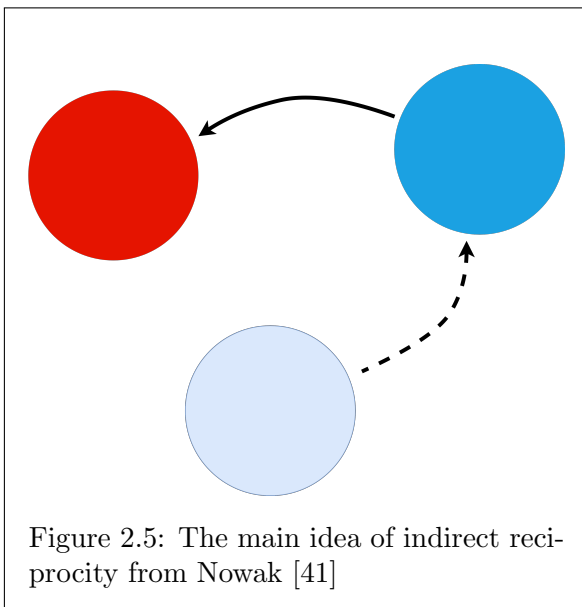
Nowak [41] presents five mechanisms that aim to facilitate the evolution of cooperation. Direct reciprocity is used in the iterated prisoner's dilemma. I discuss three others in my appendix (subsections ??) and the final one I shall discuss here: indirect reciprocity

This concept is promising for use in MASs as it solves the failure of direct reciprocity's failure with regards to when re-meeting is low. In a large society or a vast network in which agents are likely to operate in success when re-meeting is low is an important property. We will see later on that the two mechanisms can be combined.

Indirect reciprocity uses the group mechanic of reputation to encourage cooperation. Alexander [5], who was an early advocate the idea, focused on human reciprocal altruism, however, subsequent research has abstracted away from the biology [44, 33, 42, 46, 41, 29, 57, 54, 36]. The idea is that if an individual cooperates with another individual, then their reputation will be enhanced in the community. This boost of reputation makes it more likely that they will be helped by others later on. Thus this mechanism a form of reciprocal altruism.

According to Nowak and Sigmund [42] the reputation mechanic requires a higher level of intelligence than direct reciprocity, due to the complexity of group mechanics in the system. It is this kind of higher level intelligence which is required to reason about events in a group that could be a key part in the development of MASs.

Due to this reason, I feel that either indirect reciprocity or possibly a combination of both direct and indirect reciprocity is a good candidate for a mechanism to study further. For the rest of this literature review I will consider both past approaches to indirect reciprocity and other factors in the mechanism that can be used to facilitate the evolution of cooperation.

2.3.6   Nowak and Sigmund

According to Gilbert Roberts [46], Nowak and Sigmund [42] is the most influential model on indirect reciprocity. Therefore, I shall examine this model of indirect reciprocity first. Nowak and Sigmund begin by stipulating that human cooperation is due to people's 'image' of each other, which is comparable to reputation. Nowak and Sigmund converted the image to an integer score between -5 and 5 for simplicity.

The idea is simple, cooperation increases your image score by 1 and defection reduces it by 1. The higher your image score the more likely it is you will receive help. Nowak and Sigmund claim that this mechanism channels cooperation toward valuable members of the society of players.

Nowak and Sigmund note that the use of image scores and indirect reciprocity itself, leave a system open to anticipation, planning, deception and manipulation. These 4 concepts seem closely related to possible happenings in MASs. Deception and manipulation are two factors that I think are worth experimenting with. At the same time, anticipation and planning are a key part of agent design in multiple languages and frameworks. In fact the lack of planning was such an important drawback of the Agent0 language [50] Becky Thomas created a new language - PLACA [58] - which allowed agents to plan, among other improvements.

The framework created by Nowak and Sigmund is simplified from Alexander's [5] idea of human reciprocal altruism. Nowak and Sigmund describe a framework in which there is a population of individuals which act as a pool to select pairs in which one player is the donor - who can choose whether to cooperate or defect - and the other is the recipient of this action. A cooperation costs the donor c to its fitness and benefits the recipient's fitness the value of b where $b > c$. Whereas a defection costs nothing and the recipient is not benefitted. This is shown in the payoff matrix in table 2.3.

Donor Action	Payoffs	
	Donor	Recipient
Cooperation	-1	2
Defection	0	0

Table 2.3: The payoff for Nowak and Sigmund's [8] indirect reciprocity model

As noted above, these actions also affect the donor's image score, but Nowak and Sigmund add a caveat when using the idea of onlookers. The notion of onlookers limits the visibility of actions in the society by randomly selecting a group of specified size

to view each interaction. This concept is displayed graphically in figure 2.6. The concept of onlookers was added due to the realisation of Nowak and Sigmund's that in a group that is spread over a wide geographical area, not all individuals will be able to view each interaction. This sparse nature of interactions is of course especially likely in MASs. Image scores now become one player's view of another rather than a community view of the player. A matrix $ImageScore$ is used to store these scores.

The discriminator is the strategy of choice for Nowak and Sigmund. This strategy stores a number k, and when the individual u using that strategy is a donor to the individual v, u cooperates if the value $ImageScore[u, v] >= k$ otherwise it defects 2.7. This strategy is incredibly simple yet effective. The model also includes the defector and cooperator strategies. Nowak and Sigmund detailed more strategies. These strategies base their decisions not only on that of the recipient's image score but also their own. In this way, the individual will be able to decide whether it is important to boost their reputation in the system in order to receive cooperation or not.

Similarly to Axelrod and Hamilton's tournaments [8] Nowak and Sigmund [42] used a genetic algorithm to find optimal strategies under certain conditions. They varied conditions such as the length of each generation and the number of onlookers per interaction.

Nowak and Sigmund hypothesize and give evidence supporting the idea that the length of the generation is important to the evolution of cooperation. They highlight that when m donor-recipient pairs are selected in succession in a population of size n, a player is likely to be selected as part of a pair $2m/n$ times. According to their evidence, the higher this value of $2m/n$, the more likely cooperation will evolve.

Another idea put forward and evidenced by Nowak and Sigmund is that the evolution of cooperation is dependent upon the donor's knowledge of the image score of the recipient. This chance of knowing an image score is limited by the concept of onlookers. However, in Nowak's 2005 paper on the five rules of cooperation [41] he talks about using 'gossip' as an alternative to direct observation. Gossip and social ability are interesting concepts in MASs. According to Wooldridge and Jennings [62], social ability is a key property in even a weak notion agents.

The model laid out by Nowak and Sigmund is succinct and a good basis for looking at interactions in multi-agent systems. Similarly to Axelrod and Hamilton [8] Nowak and Sigmund [42] did not create a MAS to run their games.

If they had been considering MASs they may have noticed a number of limitations. The first limitation I will highlight is the way in which reputation is implemented. In the version without onlookers each player has a global image score and even in the version with onlookers there is a public matrix encoding all image scores.

Image scores could be seen as a community view of an individual. However, I argue that the idea that image scores are attempting to capture (reputation) is actually more personal. Although there might be a rough consensus as to an individual's general reputation among people in a society, this reputation is often conveyed through social means and is subject to the personal interpretation of each individual.

According to Russell and Norvig [48], IAs use an internal state. It is in this internal state, I argue, that image scores should be stored in order for an agent to have full control over them. Trust models could be devised to manage these image scores. This idea is a closer match to the MAS paradigm and it would allow for a truly distributed system.
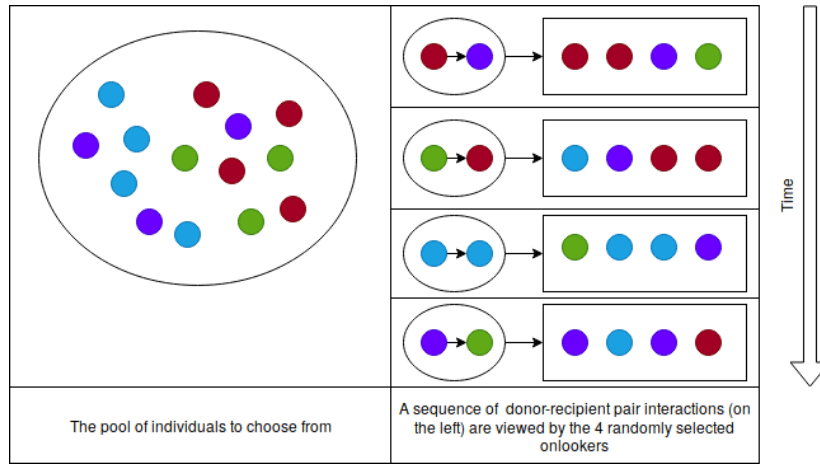
Figure 2.6: The selection of a sequence of donor-recipient pairs and onlookers in Nowak and Sigmund's [42] model of indirect reciprocity. The macro-view of indirect reciprocity. The lines and arrows show the spread of information.
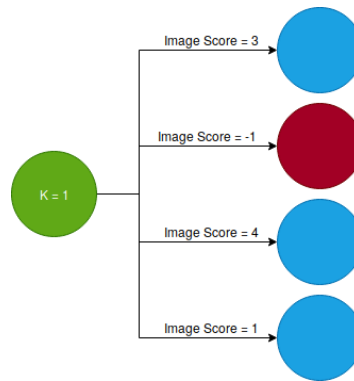
Figure 2.7: On the left an image scoring discriminator with K=1. On the right the agents the discriminator is a donor for. Blue agents represent where there would be a cooperation from the donor, red represents when the discriminator would defect against them. The rule is when $ImageScore >= K$ cooperate else defect.

Indirect reciprocity makes a good candidate for use in MASs and this model is both very influential and a good basis for those MASs. The model includes extensive interesting aspects that have been examined carefully including the onlookers and reproduction mechanisms, and also includes ideas that have not been fully researched such as a gossip systems and using deception and manipulation to affect the system. These features allow more room for me to explore if I use Nowak and Sigmund's model as a basis, with a few alterations to more closely match the MAS paradigm.

2.3.7 The Standing Strategy and Further Limitations of Nowak and Sigmund

Leimar and Hammerstein [29] criticised Nowak and Sigmund's [42] limited range of strategies. They also highlighted the abilities of another strategy, the standing strategy, over image scores. The standing strategy was first described by Sugden [57] and and uses a 'switch' [42].

Milinski *et. al* [33] described the standing strategy as the idea that an individual does not only aim for a good fitness but also a good standing. Each individual holds other individuals

in either a good or bad standing, which starts out as good until perceived otherwise. The event that causes the switch from good to bad is an individual defecting against another with a perceived good standing.

The idea is that it is morally incorrect to defect against a good individual, but defecting against a bad individual is acceptable as it punishes those members of society that are not valuable. There is conflict between the use of image scores and the standing strategy.

The argument for the standing strategy is as follows: When using image scoring, if a discriminator has $k = 0$ and is a donor to another with an image score of -1, they will defect. This action is seen as punishing a bad individual, as it channels fitness away from bad members of the society. However, the donor's image score is reduced by 1. This change in score decreases the chance that the donor will receive cooperation from others, so there is no real incentive for them to punish the bad members of that society, except for pure altruism.

Conversely, the standing strategy only reduces the reputation of those who defect against good individuals. Therefore, punishing bad members of the society has no negative effect on those members. Leimar and Hammerstein [29] argue that Sugden's version of indirect reciprocity and his use of the standing strategy are more robust than Nowak and Sigmund's image scoring.

Milinski *et. al* [33] analysed Leimar and Hammerstein's argument, suggesting that they were correct under conditions which allowed for perfect perception of events and unlimited memory capacity. However these conditions are not always true, especially in a MAS in which the environment is inaccessible [48].

Nowak and Sigmund claimed that indirect reciprocity is open to deception and manipulation, especially when we consider the use of gossip in a system. Gossip opens IAs to imperfect perception of other IAs. Therefore, it would be interesting to see which strategy is more effective in a system in which other IAs actively attempt to deceive using gossip, and the information received is not always perfect.

2.3.8   Mixed Reciprocity Models

Roberts [46] and Phelps [44] noted that indirect reciprocity is focused towards interactions with other individuals with whom the donor has no previous interactions. However, what about when re-meeting is more likely? I stipulate that an individual is affected by both being the recipient of actions and their observation of actions. Roberts engaged with the issue by introducing an experience score. This experience score works similarly to an image score but is bounded by -1 and 1. It increases when the individual is a recipient of a cooperation, and decreases when receiving a defection.

Roberts also created a version of the standing strategy which uses an image score (-1 to 1). However herein, the value changes according to the rules of the standing strategy rather than Nowak and Sigmund's rules.

Both Roberts and Phelps measured the use of indirect reciprocity against the use of direct reciprocity. Roberts concludes that indirect reciprocity is the more popular decision mode under conditions in which re-meeting was less common, and direct was more popular when re-meeting was frequent. On the other hand, Phelps' experiments garnered different results being that in small groups direct and indirect reciprocity exist in equilibrium.

Similarly to Nowak and Sigmund [42], Roberts I believe falls short in recognising how personal interpretation of events are. There are many different trust models a player could

use by mixing interpretation of events in which they are the recipient and also in which they are not the recipient. For example, it could be expected that being on the receiving end of a defection means that a player is likely to be more hardline, than when they observe a defection against another. However, Roberts limits the study to using a simple experience score.

I would suggest using a model similar to that I described in section 2.3.6, in which interpretation of events is up to an agents trust model. Some trust models could interpret events when the agent is the recipient different to those in which they are observing.

2.3.9   Gossip

Nowak and Sigmund [42] realised the issues surrounding the fact that all individuals being able to view interactions in their simulation framework is not concomitant with real life. As such, they introduced the idea of onlookers as noted in section 2.3.6. An issue with limiting observation is that knowing an accurate image score of a player is important to the evolution of cooperation in an indirect reciprocity system. Nowak [41] suggested using gossip to spread information regarding image scores in their simulation framework.

Sommerfeld *et al.* [54] conducted an empirical study on gossip between cooperative and uncooperative individuals in an indirect reciprocity setup. The experiment consisted of humans playing a number of indirect reciprocity rounds to build up a cooperation history, and then a smaller number of gossip rounds. These rounds were then repeated to see how the individuals reacted to the gossip.

The focus of the experiment was to look at gossip composition, gossip transfer and resulting behaviour. The findings included that gossip is an effective method of spreading reputation information in an indirect reciprocity system, on fulfilment of some conditions, the first of which is the truthfulness of the gossip. Gossip must accurately reflect the behaviour of the subject of the gossip.

Another condition is the comprehensibility of the gossip. The language used in the gossip must be interpretable by the recipient of the gossip. In an agent system this condition would require an agent communication language (ACL) (such as KQML [14] or FIPA-ACL [43]) and would also require the content of the messages that use this language to be interpretable by the recipient agent. The final condition is that individuals must act accordingly based on the gossip they receive.

These three conditions of a system would have to be created by building strategies for agents that attempt to uphold them, and a communication language to support accurate and interpretable information. Nowak and Sigmund [42] identified that indirect reciprocity systems can be effected by deception and manipulation from malicious players. The addition of gossip creates a meta-game on top of the indirect reciprocity system. Players must have a trust model that differentiates between dealing with gossip and with directly observed interactions, and a strategy on spreading gossip effectively.

2.3.10   Mui's Computational Models of Trust and Reputation

Mui [36] presented an indirect reciprocity simulation framework similar to Nowak and Sigmund [42] that spreads social information through 'acquaintance networks' to inform a donor's decision. An individual in this framework builds up a network of players whom they meet through interaction, their 'acquaintance network'. If this individual is then a donor in
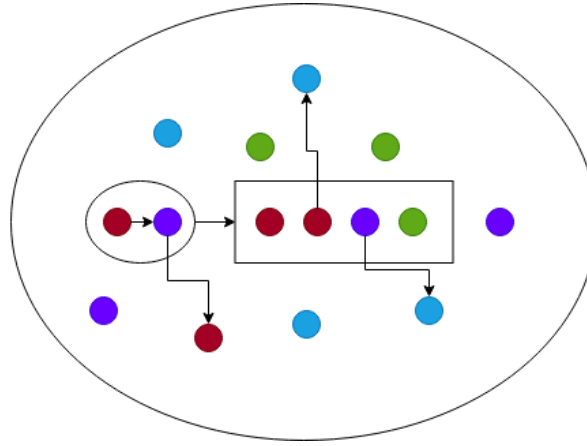
Figure 2.8: The spread of information (shown through the lines) through a population using indirect reciprocity with gossip and onlookers

a donor-recipient pairing the individual consults their acquaintance network regarding the recipient. The information gathered from this network helps a donor to decide if they can trust the recipient to reciprocate their cooperation; known as 'collective memory'.

Baumeister *et al.* [9] advocated for the idea that gossip is used for four functions, including strengthening bonds between the gossiper and recipient, enabling the spread of information about the subject (one function for each positive and negative reasons), and helping to educate individuals about the complex cultural systems they reside in.

I would argue that indirect reciprocity systems and MASs are complex cultural systems in which gossip can be applied for these four functions. Mui's acquaintance networks support this gossip functionality to a certain extent; however they are limited by the lack of proactivity of gossip. Sommerfeld *et al.* [54] highlighted the willingness of their participants to spread gossip, and the last function of gossip proposed by Baumeister *et al.* [9] (cultural education) suggests that for gossip to be most effective the gossip should be proactively spread.

As a result, I believe that a theoretical framework which uses gossip should allow a proactive social ability by using gossip as an action at times at which an agent is not acting as a donor.

2.3.11   Reproduction and Genetic Algorithms

As discussed in section 3.2.2 finding the best solution from a set of possible solutions is an optimisation problem. Genetic algorithms are an approach to solve optimisation problems, and as such my system will be acting like a genetic algorithm with simulate reproduction.

Between each generation a reproduction phase occurs, in which agents are created and associated with strategies for the next generation. Nowak and Sigmund [42] and Axelrod and Hamilton [8] (though indirectly their payoff scores do seem to equate to fitness) used a reproduction mechanism inspired by Herbert Spencer's [55] famous phrase "Survival of the fittest".

Wooldridge's [61] argument that agents attempt to fulfil a design objective using their own autonomy suggests that strategies for agents will only be used in practice if they are successful in their design objective. The main design objective of players in my game is to interact with players to increase their fitness, with some having other big objectives such as aiding in the evolution of cooperation.
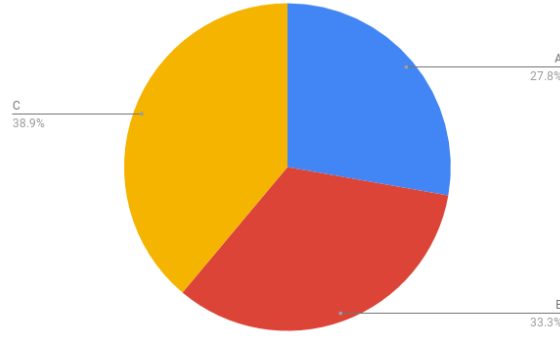
Figure 2.9: The roulette wheel from my proportional selection example

The measurement of how successful an agent is in my system is thus their fitness score. Just like in Nowak and Sigmund's, and Axelrod and Hamilton's reproduction mechanism I propose using a "Survival of the fittest approach". The real replicated information in my system is the strategy an agent uses, so the mechanism I am using makes it more likely for strategies with higher overall fitness scores across agents in a generation to be reproduced.

Francq [16] puts forward two types of reproduction: proportional selection and tournament selection.

Proportional selection translated to my system and simplified would look something like the following. Take a roulette with p (number of players) slots and divide the p slots into n (number of strategies) sections. The size of each section is directly proportionate to the average fitness of all players of a certain strategy. For example strategy A has 2 agents with fitness 4 and 6 respectively, strategy B has 4 agents with fitness 3, 8, 9 and 4 fitness respectively and strategy C has 1 agent with fitness 7. The average fitness of A is 5, B is 6 and C is 7, so A would receive 5 slots of the roulette wheel, B 6 slots and C 7 slots. The resulting roulette wheel displayed in 2.9 is spun and a ball dropped and whichever slot the ball landed on the corresponding strategy is selected for the new player.

Tournament selection could be translated in the following way. Have an empty list which will contain an ordered list of agents. In a loop (until the population is of size 1): select two agents from the population, remove the lowest fitness one and insert it into the list. After the loop insert the last agent into the top of the list. The top of the list always contains the highest fitness agent. Using the same population of agents in the proportional selection algorithm we can give an example of this in table 2.4.

Set	Test	Put in List	List
$A_1, A_2, B_1, B_2, B_3, B_4, C_1$	(A_1, B_3)	A_1	A_1
A_2, B_2, B_3, B_4, C_1	(B_1, C_1)	B_1	A_1, B_1
B_2, B_3, B_4, C_1	(A_2, C_1)	A_2	A_1, B_1, A_2
B_2, B_3, C_1	(B_4, C_1)	B_4	A_1, B_1, A_2, B_4
B_3, C_1	(B_2, B_3)	B_2	A_1, B_1, A_2, B_4, B_2
B_3	(B_3, C_1)	C_1	$A_1, B_1, A_2, B_4, B_2, C_1$
$\emptyset$		B_3	$A_1, B_1, A_2, B_4, B_2, C_1, B_3$

Table 2.4: The payoff for my indirect reciprocity model

Fitness proportionate selection seems to translate best to my system as there is an obvious and simple way to select each agents strategy for the new generation using the roulette wheel, unlike the tournament selection process which requires the crossover step. The crossover step requires a chromosome represented as a bit array to produce an offspring from two parents. This is analogous to sexual reproduction, but not the building of new agents.

Lipowski *et al.* [30] presented an efficient version of roulette wheel selection using stochastic acceptance rather than the searching method used by Francq [16]. The stochastic acceptance algorithm works like this: select randomly one of the individuals from the last generations population, with fitness of the individual as w_i and w_{max} the maximal fitness of the generation use the probability w_i/w_{max} to select whether or not to use this agents strategy for a new generation agent, if not repeat. Lipowski *et al.* proved mathematically that the probability distribution of this method and general roulette wheel selection are the same, and that the stochastic acceptance algorithm is more efficient as well.

2.3.12 Summary

Chapter 3: **Framework**

3.1   Introduction

In the following sections of this chapter I will be discussing the theoretical side of my framework and the subsequent practical implementation of this framework.

3.2   Theoretical Framework

3.2.1   Introduction

In this section on the theoretical framework I will be presenting an overall design for my mixed reciprocity model and MAS. I have taken inspiration from past work on game theory (discussed in section 2.3) and techniques for the development of MASs (from section 2.2). Franklin and Graesser [17] stipulated that to describe an agent one must describe five aspects: the environment the agent resides in, the agent's sensing capabilities, the possible actions the agent can take, the drives or primative motivators for an agent's actions and the action selection architecture for the agent. I shall describe these five aspects in the following subsections here.

3.2.2   The Game and Environment

The environment in which the agents will reside in my system will provide facilities for interaction between the agents. An instance of the environment in my system will be known as a generation. Each generation contains a set of agents. These sets of agents will act as pools to select donor-recipient pairs and onlookers for those pairs.

A generation is contained within a community. These communities contain multiple generations with a strict ordering. A game has distinct timepoints. The number of these timepoints vary according to the number and length of generations - which are variables set by the user at the start. Say these timepoints range from $1..n$ and there are generations $1..k$ where $k < n$ and $n\%k = 0$. Then the timepoints are distributed evenly across each generation like this:

$$\{\{1,..,(n/k)\},\ \{(n/k+1),..,(2n/k)\},\ ..,\ \{(n-(n/k)+1),..,n\}\}$$

The environment will work using a cycle in which each step follows the process perceive-decide-execute. Each step of the cycle is a timepoint. The perceive section of a step refers to when all agents receive perceptions generated for that timepoint - mostly from actions in the previous timepoint. All agents then decide on an action and the last part of the step is the execution of these actions in the environment.

The key to the cycle steps is that all agents perceive, after which all agents decide and then all actions are executed. The effect is to synchronise agents steps and prevent actions from a timepoint affecting other agents decisions at the same timepoint. Synchronicity keeps the environment static for the period in which an agent is deciding.

In each timepoint there is exactly one donor-recipient pair selected at random from the

generation's pool of players. For that pair there is a group of onlookers, again randomly selected from the remaining players in the generation's pool.

As discussed, each generation contains a set of players which participate in the cycle steps for each timepoint of that generation. But how are these sets of players selected? For the first generation, a number of agents and associated strategies (subsection 3.2.7) are selected by the user. For subsequent generations, the set of players is selected using fitness proportionate selection using Lipowski *et al.*'s roulette wheel selection via stochastic acceptance [30] (discussed in subsection 2.3.11).

Each player has a fitness score which is used in the reproduction mechanism. A fitness score starts at zero - it cannot drop below zero - and is affected by the actions of the agent and others as described in subsection 3.2.4. On top of this reproduction algorithm is a chance of mutation c where $0 \leq c \leq 1$ which the user is able to set at the start.

One of the aims of this project is to discover successful agent strategies for the mixed reciprocity model. This problem can be formulated as an optimisation problem, which can be solved using genetic algorithms. The effect of using fitness proportionate selection and the generations of players is to create a genetic algorithm. When experimenting with agent strategies, the strategies which were most successful will hopefully become obvious at the end of a community's generations by the concentration of agents using those strategies.

In summary, the components of this environment are the community, the generations, the sets of agents within the generations, the timepoints throughout the community's life, the onlooker mechanism, the reproduction mechanism, the cycle steps, the donor-recipient pairs, the percepts and the actions. For a discussion on the properties of my environment see appendix subsection 8.2.1.

3.2.3 Percepts

Percepts are the information received at an agent's sensors from it's environment. They are received in the first stage in each cycle step. In my system, percepts are a direct observation of an interaction, hearing gossip from another agent or sensing whether they are the donor or recipient in a donor-recipient pair.

In each timepoint, there is a donor and a recipient selected at random from that generation's pool of players. The two agents are made aware of this fact by receiving a percept of the role that they are taking and which other agent is in that pair in the cycle step's perceive stage. Agents can then act accordingly.

For each interaction, there is a set of onlookers selected at random from the generation's pool of players (not including the recipient or donor). In the cycle step after the interaction takes place the onlookers and the recipient receive percepts containing information as to who the donor and recipient were and the action the donor decided on.

In the gossip (3.2.5) and action (3.2.4) subsections, I discuss an agent's ability to act by gossiping to another agent. The action of gossip produces a percept. This percept contains the message sent using the Simple Agent Gossip Language. The percept is received by the agent given as the recipient by the gossiper.

3.2.4   Actions

My system focuses on interactions between agents and as such, agents require a number of action possibilities by which to interact with one another. To simplify the action there are 3 possible main actions: idle actions, any action when the agent is a donor and gossip actions. Idle actions are simple: an agent is idle in that timepoint, their action has no effect on the environment or other agents, except for through inaction.

Actions when an agent is a donor are more complex. As discussed in subsection 3.2.3, an agent perceives when they are a donor in a donor-recipient pair. When an agent perceives that they are a donor, they have no choice but to commit one of the following two actions: cooperate or defect. In this interaction where the actor is the donor, the recipient has no control over what happens.

The payoff matrix, taken from Nowak and Sigmund [42] in table 2.3, stipulates that when a defect action is chosen, there is no effect on either the donor's or the recipient's fitness. When a cooperate action is chosen however, there is a cost of 1 to the donor's fitness and a benefit of 2 to the recipient's fitness. As discussed in subsection 3.2.3, this action also produces percepts of the interaction event.

When an agent is not a donor, they cannot choose to cooperate or defect with anyone. However, they can choose one of the other two actions: idle or gossip. A gossip action is another type of interaction between agents. A gossiper chooses to communicate with another agent. The contents and structure of this communication are detailed in subsection 3.2.5 and the only effect is the percept discussed in subsection 3.2.3.

3.2.5   Gossip and Simple Agent Gossip Language

```
{
  "recipient": 1,
  "about": 3,
  "gossiper": 4,
  "timepoint": 7,
  "gossip": "positive"
}
```

Source Code 3.1: Example of a message in SAGL

With the design of the gossip in my system, I have focused on social agency over mental agency and keeping the layout of the gossip simple. KQML and FIPA-ACL are powerful tools, which I do not need, and as such I have decided to create my own language which I have named the Simple Agent Gossip Language (SAGL).

I have attempted to create a language which may be used by agents to facilitate three out of four of Baumeister et al.'s [9] functions - excluding cultural education, which is beyond the scope of my project. The language I propose is very simple. There are five fields. Three of the fields are identifiers (one for each of the recipient, the target and the gossiper). Another is the timepoint at which the gossip action was decided upon. The final field is the gossip.

Sommerfeld et al. [54] categorised gossip in their experiment as either positive or negative. Individuals in my system are attempting to convey similar reputation information, and as such, the gossip field of SAGL either contains the keyword positive or negative.

Agents can attempt to improve bonds between themselves and others, by spreading positive gossip about themselves. Agents can also spread negative information about others to either warn other agents of a non-trustworthy agent, or to harm the agent the gossip is about. Agents can also send positive gossip about others to spread knowledge of trustworthy agents in the system.

3.2.6   Agent Architecture

Russell and Norvig's [48] model-based reflex agent is the closest of their agent architectures to the architecture I will be using. The architecture includes using a trust model to revise beliefs from received percepts and then using these beliefs with teleo-reactive programs to make action decisions.
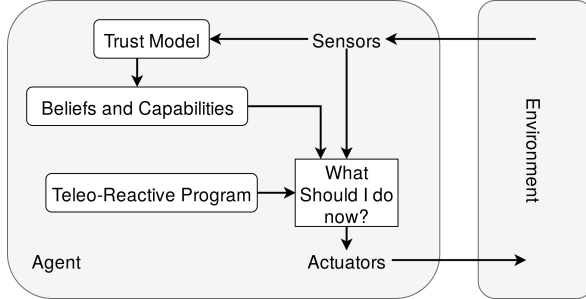

Figure 3.1: The agent architecture in my system.

The trust model and beliefs system use the multi-valued fluent cached events calculus. The beliefs are a combination of the value and the fluent of multi-valued fluents [6]. For example the fluent $standing(agent1, agent2, t) = value$ is the belief of $agent1$ on the standing of $agent2$ at timepoint t. In this fluent the value can either be 'good' or 'bad'. The trust model controls the interpretting of percepts and updating of these fluents.

Capabilities are also a part of the beliefs system that are revised during the interpretation of percepts. These capability beliefs constrain an agent as to the actions to which they can commit i.e. an agent can only commit to an action if they believe that they are capable of it. For example, at a time when an agent is a donor in a donor-recipient pair, they will only believe themself capable of a donor action, constraining their action choice.

To make a decision on an action the agent architecture combines the agent's beliefs with a teleo-reactive program that makes up an agent's strategy. A strategy is made up of strategy 'components'. Of which one I have already discussed: the trust model. The trust model is combined with a component determining how to act as a donor and another determining how to act when not a donor. The decision is a commitment at that timepoint to carry out that action.

3.2.7   Strategies and Trust Models

As I have discussed, an agent's strategy is made up of the trust model, the strategy component for when the agent is a donor and the strategy component for the agent when it is not a donor. These strategies have been inspired by past work on indirect reciprocity and expanded upon. For example, I have enhanced agent's interpretation of events in order that they respond differently when they are the recipient of an action from a donor and when they are the recipient of gossip. The action decisions made when agents are donors have generally remained the same as those in past work. However, these action decision systems have been augmented with new strategy components for when an agent is not a donor.

Defector and Cooperator

Franklin and Graesser [17] talked about an agent's primative motivator or drive. Each strategy has a drive or a combination of drives. For example, the 'Defector' strategy seeks to protect it's own interests by always defecting, thus not causing it to lose fitness points. The defector strategy has been augmented with three possible strategy components which it can use when it is not a donor: 'Lazy', 'Spread Negative' and 'Promote Self'.

The 'Lazy' strategy component always decides on being idle. The 'Spread Negative' component randomly spreads negative gossip about other agents in the system. This compo-

nent simulates what might be known as 'fake news': a form of deception and manipulation. The final component is 'Promote Self' in which the agent always spreads positive gossip about itself, aiming to increase it's reputation by spreading misinformation and encouraging others to cooperate with the agent using it.

The strategy 'Cooperator' aims to maximise social welfare by always cooperating as a donor. I have also augmented this strategy with three components when the agent agent not acting as a donor: 'Lazy', 'Promote Self' and 'Spread Positive'. 'Lazy' is the same as for the defector strategy and so to is 'Promote Self', although 'Promote Self' when used by a 'Cooperator' could be seen as coming from a drive to improve cooperation and not to deceive other agents. The third is similar to 'Spread Negative' but instead spreads randomly positive gossip in order to improve cooperation between agents in the system. This strategy component could result in accidental manipulation to cooperate with non-reciprocating agents by naive agents.

Image Scoring Discriminator

Neither 'Defector' nor 'Cooperator' have any beliefs about other agents and as such are limited to only trust models that constrain their capabilities. A more insteresting strategy is the 'Image Scoring Discriminator'. Built up around Nowak and Sigmund's [42] discriminator, strategies using this component keep beliefs about the image score of other agents in the system bounded by -5 and 5. To control these image score beliefs I have created three trust models for these discriminators: 'Naive Trusting', 'Trusting' and 'Distrusting'.

The 'Distrusting' trust model abjectly ignores any received percepts which it has not observed directly i.e. rejects any gossip. At the other end of the spectrum, 'Naive Trusting' accepts any gossip it receives, increasing the image score of the agent the gossip is about by 1 if the gossip is positive, and decreasing by 1 if the gossip is negative. The 'Trusting' model is less black and white. This trust model will only change the beliefs regarding the target of the gossip if the gossiper is trusted i.e. has an image score greater than or equal to k, where k is an integer variable set at the start.

The 'Image Scoring Discriminator' strategy component has four corresponding non-donor strategy components. These include 'Lazy' and 'Promote Self'. Again, it could be said that 'Promote Self' is used to improve cooperation towards the agent and not to deceive the recipient.

The other two components are 'Spread Accurate Positive' and 'Spread Accurate Negative'. 'Spread Accurate Positive' spreads positive gossip about trustworthy agents to other trustworthy agents. Trustworthy means that their image score is greater than or equal to k. The aim of this positive gossip is to improve cooperation between trustworthy agents. 'Spread Accurate Negative' spreads negative gossip about agents believed to be untrustorthy - agents with an image score less than k - to trustworthy agents.

It is also possible with agents using image scores to select agents with the 'Personal Grievance' option. When this option is selected, agents using the strategy increase or decrease image scores by 2 when respectively receiving a cooperation or defection. These agents use the selected trust model when they are not the recipient of these actions.

Standing Discriminator

The standing strategy is also included in my system under the name 'Standing Discriminator' with the trust models: 'Naive Trusting', 'Trusting' and 'Distrusting'. This strategy has

the following strategy components for when the agent is not a donor: 'Lazy', 'Promote Self', 'Spread Accurate Positive' and 'Spread Accurate Negative'. These are the same as for the 'Image Scoring Discriminator' strategy components except an agent is said to be trustworthy if their standing is 'good'.

The original standing strategy specification for when to change an agent's standing between 'good' and 'bad' is used for directly observed events. The 'Personal Grievance' option is not selectable for the standing strategy.

3.2.8   The Veritability Discerner

The image scoring and standing strategies were both built for systems in which gossip was not present and the reliability of the information they received did not require questioning. I have added trust models to compensate for these issues. However, these trust models are built to either accept or to reject the information from the percept and I do not believe that this procedure reflects how reputation information is interpreted in the real world.

Thus, I have developed a new strategy with corresponding trust models and strategy components. I have named this strategy the 'Veritability Discerner'. An agent using the strategy holds two beliefs about all other agents. One is the number of percepts the agent has received about the other agent n (when the other agent was a donor or the target of gossip). The second, is an integer value I have named the 'veritability rating' v. This rating is initially 0. This rating is similar to an image score but it is not bounded and does not simply increase or decrease by 1, the addition or subtraction to the veritability rating is weighted by the reliability and severity of the percepts.

Trust Models

This strategy comes with three trust models: 'Strong Reactor', 'Balanced Reactor' and 'Forgiving Reactor'. All use the same central weighting and reaction system but apply slightly different weights on top of these.

The central weighting and reaction system is as follows. Viewing of a cooperation is the most reliable indication that the donor can be trusted, and so has the highest weight (20) added to the veritability rating of the agent. Conversely viewing a defection against a recipient who is a trusted agent is the most reliable indication that the donor is untrustworthy, and so the highest weight (20) is subtracted from the veritability rating. When either of these interaction events are observed, the count of percepts received about the donor is incremented by 1. Similaryly to the standing strategy, a defection against an untrusted agent has no effect on the veritability rating of the agent but does increment the percept count.

Negative gossip about an agent from a trusted source subtracts 10 from the veritability rating of that agent, whilst positive gossip from a trusted source adds 10 to the rating. If gossip is from an untrusted source, then only 1 is respectively added or subtracted from the rating.

The 'Strong Reactor' trust model multiplies weights for negative percepts by 2. This multiplication makes the weight for defection against a trusted recipient 40. The multiplication also makes negative gossip from a trusted source subtract 20 from the veritability rating of the target of the gossip. The last effect the multiplication has is to cause a subtraction of 2 from the veritability rating of the target of gossip in response to negative gossip from an untrusted source.

The 'Balanced Reactor' does not apply a weight to either positive or negative percepts. Finally the 'Forgiving Reactor' multiplies positive weights by 2, making the weight of viewing a cooperation 40, positive gossip from a trusted source 20 and positive gossip from an untrusted source 2.

Trustworthiness

A third belief can be derived from the veritability rating and count of percepts received: trustworthiness. An agent using the strategy has a value k similar to an 'Image Scoring Discriminator' agent, though the possible values for this are -10, -5, 0, 5 and 10. The trustworthiness belief is derived as follows. Divide the veritability rating by the count of the percepts received $v/n = t$ - if n is 0 t defaults to 0. If $t \geq k$, then the agent is trustworthy, otherwise they are untrustworthy.

$holds\_at(veritability\_rating(Perceiver, Subject) = V, Timepoint) \wedge$
$holds\_at(percept\_count(Perceiver, Subject) = N, Timepoint) \wedge$
$get\_K(Perceiver, K) \wedge V/N \geq K \rightarrow$

$$trustworthy(Perceiver, Subject, Timepoint)$$

The derivation of this belief has the same effect as taking the mean of a list of weighted actions. This mean takes into account all the actions committed to by an agent but smooths out noise from possibly untrustworthy or distorted sources.

The trustworthiness belief about an agent is used when intepretting gossip and when deciding whether another agent's defection was justified or not. This belief is also used for when the agent is a donor. If the recipient is considered trustworthy then the agent will cooperate, if they are not, the agent will defect.

$get\_strategy(Actor, "VeritabilityDiscerner") \wedge$
$interaction\_pair(Actor, Recipient, Timepoint) \wedge$
$trustworthy(Actor, Recipient, Timepoint) \rightarrow$

$$action\_commitment(cooperate, Actor, Recipient, Timepoint)$$

Strategy Components

Finally we need strategy components for when an agent using the strategy is not a donor. These are similar to the discriminator strategies and include: 'Lazy', 'Promote Self', 'Spread Positive Trusted' and 'Spread Negative Untrusted'. 'Lazy' and 'Promote Self' do the same as with all other strategies. 'Spread Positive Trusted' spreads positive gossip to trusted agents about trusted agents. 'Spread Negative Untrusted' spreads negative gossip to trusted agents about untrusted agents.

'Spread Positive Trusted' formalisation:

$get\_strategy\_component(Actor, "Spread\ Positive\ Trusted") \wedge$
$not\ interaction\_pair(Actor, Timepoint) \wedge$
$findall(Trusted, trustworthy(Actor, Trusted, Timepoint), TrustedAgents) \wedge$

$$len(TrustedAgents, Len) \ \wedge Len >= 2 \ \wedge$$
$$get\_2\_random(TrustedAgents, TrustedAgent, Recipient) \rightarrow$$
$$action\_commitment(gossip(\{recipient : Recipient, about : TrustedAgent, gossiper :$$
$$Actor, timepoint : Timepoint, gossip : positive\}), Actor, Recipient, Timepoint)$$

$$get\_strategy\_component(Actor, \text{``Spread Positive Trusted''}) \ \rightarrow$$
$$action\_commitment(idle, Actor, Recipient, Timepoint)$$

'Spread Negative Untrusted' formalisation:

$$get\_strategy\_component(Actor, \text{``Spread Negative Untrusted''}) \wedge$$
$$not \ interaction\_pair(Actor, Timepoint) \wedge$$
$$findall(Trusted, trustworthy(Actor, Trusted, Timepoint), TrustedAgents) \wedge$$
$$len(TrustedAgents, Len) \ \wedge Len >= 2 \ \wedge$$
$$get\_random(TrustedAgents, Recipient) \rightarrow$$
$$action\_commitment(gossip(\{recipient : Recipient, about : TrustedAgent, gossiper :$$
$$Actor, timepoint : Timepoint, gossip : positive\}), Actor, Recipient, Timepoint)$$

$$get\_strategy\_component(Actor, \text{``Spread Negative Untrusted''}) \ \rightarrow$$
$$action\_commitment(idle, Actor, Recipient, Timepoint)$$

Summary

I have presented a new strategy for use in indirect and direct reciprocity models. The aim of this strategy is to be less susceptible to deception and manipulation by weighting the increase and decrease of the ratings by the severity and reliability of the percepts and also by using the smoothing properties of the arithmetic mean.

3.2.9   Summary

In summary, I have presented a MAS that uses a variant of a genetic algorithm to reproduce agents between instances of the environment. I have described the components of the environment and game, the percepts that can be received by agents, the actions available to agents and how these are constrained and the simple agent gossip language. I have also discussed the architecture that I am planning to use in the system and the strategies and trust models available to associate with the agents, including a new strategy that aims to be more suitable with a system in which deception and manipulation are possible.

3.3   Implementation

3.3.1   Introduction

In this section I will discuss the details of the implementation of the theoretical framework outlined above as well as the architecture of the system that is hosting it. I will also be discussing the software engineering techniques, tools and processes I have employed in the development of my system.

3.3.2   Development

Architecture
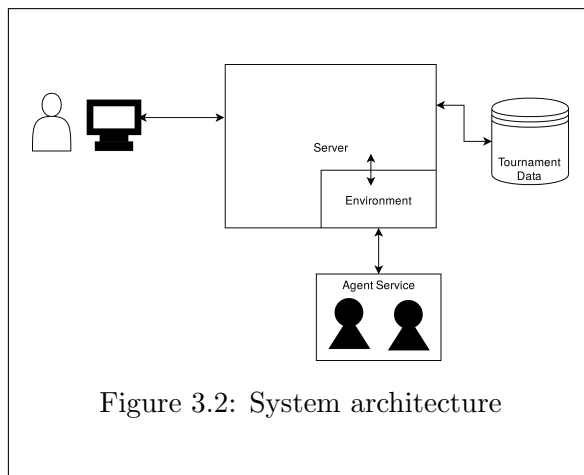
Figure 3.2: System architecture

One of the objectives of my project is to create a system that is distributable, in order for it to be an experimentation and learning tool for those interested in studying multi-agent interactions. This distributability is reflected in the architecture I have used to develop my system. A web application acts as the 'shop front' of the system and allows anyone with internet access to create, run and view analysis of the available games.

I named the web application 'The Nature Engine' and it is of typical web architecture: the client-server model. Built into this web application is an implementation of my theoretical framework which runs the environment and the agent's bodies. Notably this part of the system doesn't include agent decision making components. These components are hosted in a separate web service, that has been named the Agent Mind as a Service (AMAAS). This web service exposes an API for the web application to interact with.

The final component of the system is a database for the long term storage of data about users of the site (to enable users to record and label experiments) and game data. This is a relational database that due to the use of and ORM (SQLAlchemy) I have been able to create as both SQLite for development and MySql for deployment. An illustration of this is in figure 3.2. The system is deployed on an ubuntu server and the database is situated on another machine.

Proof of Concepts and Learning

At the very beginning of this project I had a good amount of software development experience, some experience with database development and logic programming, and less experience with web development. Before undertaking the development I decided I needed to learn more about some of these areas (especially web development).

I had decided that I wanted to extend my experience with Python due to it's nice language features, good library support and it's lack of use in the development of MASs (in comparison to Java and Prolog). As such I decided to use the Flask web development micro-framework and partake in Miguel Grinberg's course "The Flask Mega-Tutorial" [1]. Flask is
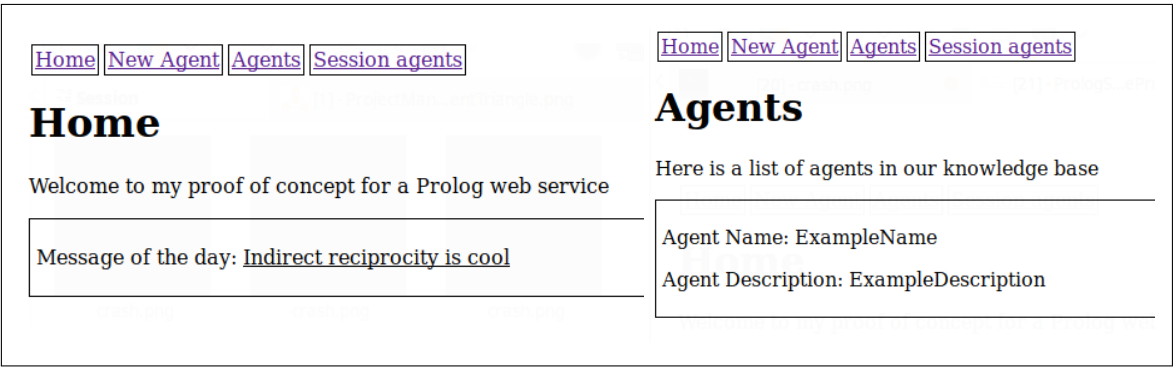
Figure 3.3: Two screenshots of two pages of the Prolog web service proof of concept

a micro-framework as it only includes a minimal amount of functionality in comparison to frameworks such as Django. Nevertheless Flask is extensible and there are a lot of libraries a programmer can use to bring in required functionality.

Using inspiration and the techniques learnt in this course I developed an early prototype of The Nature Engine. This prototype provided a number of web pages and navigation between them including a home page, pages to set up and run two player iterated prisoner's dilemma games and pages to set up and run tournaments of the iterated prisoner's dilemma and the respective analysis of both game types. Both of these games are run using the Axelrod-Python library [13]. This early prototype also included a database set up, a redis queue to run the tournaments and a front-end that used bootstrap for styling and React.js for a dynamic form to set up tournaments.

Prolog is a logic-language traditionally used for the development of symbolic AI systems. The language allows a programmer to elegantly lay out how an agent makes decisions and interprets percepts. There are limited libraries for interfacing between Prolog and Python, and creating a method for interfacing between them reliably would have taken a while. It was easier to create a web service to host agent minds and became a good opportunity to hone my skills in web development.

To develop these skills I used Anne Ogborn's 'Creating Web Applications in SWI-Prolog' tutorial [2] and the swi-prolog documentation. Using this tutorial I created a basic web application that imitates managing the creation of strategies and agents within the system. This was far simpler than The Nature Engine prototype (figure 3.3) as it did not build up to a final application (the development of AMAAS was separate to this proof of concept).

The Nature Engine

I built on top of the early prototype of The Nature Engine to develop the final web application. This included improving the front end content, adding a section to run the theoretical framework I have devised and adding a couple of extra features.

To improve the front end content I worked on improving the form that allows users to dynamically select and remove agents from a tournament. This also translated into the form that allows users to do the same for my game. Other improvements were the use of a bootstrap carousel for the homepage to prevent text overload on the screen, better text descriptions on the website and improved game and tournament analysis.
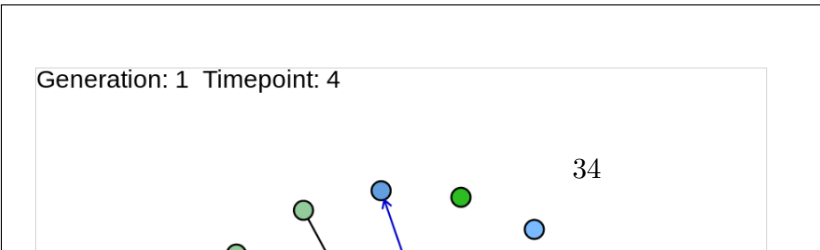
The main bulk of the development was im-

34

plementing the theoretical framework. However, to run this game in the web application required a front-end, processing of user requests and storing of data in the database. This development is a vertical slice of the web application. The process is as follows. A user sets up a game on the set up page using a React.js form (decisions are informed by strategy information in an information box), this form is submitted which sends a post request with the relevant information to the server.

The server then puts a method to create and run the game in a redis queue and then store the resulting data in the database. The user is redirected to a waiting page, which either notifies them of a timeout on their game or sends them through to the analysis page. The analysis page includes a run down of the statistics in terms of cooperation, gossip and the success of each strategy and agent. A visualisation of the game that has been run is also included in an animation on the analysis page.
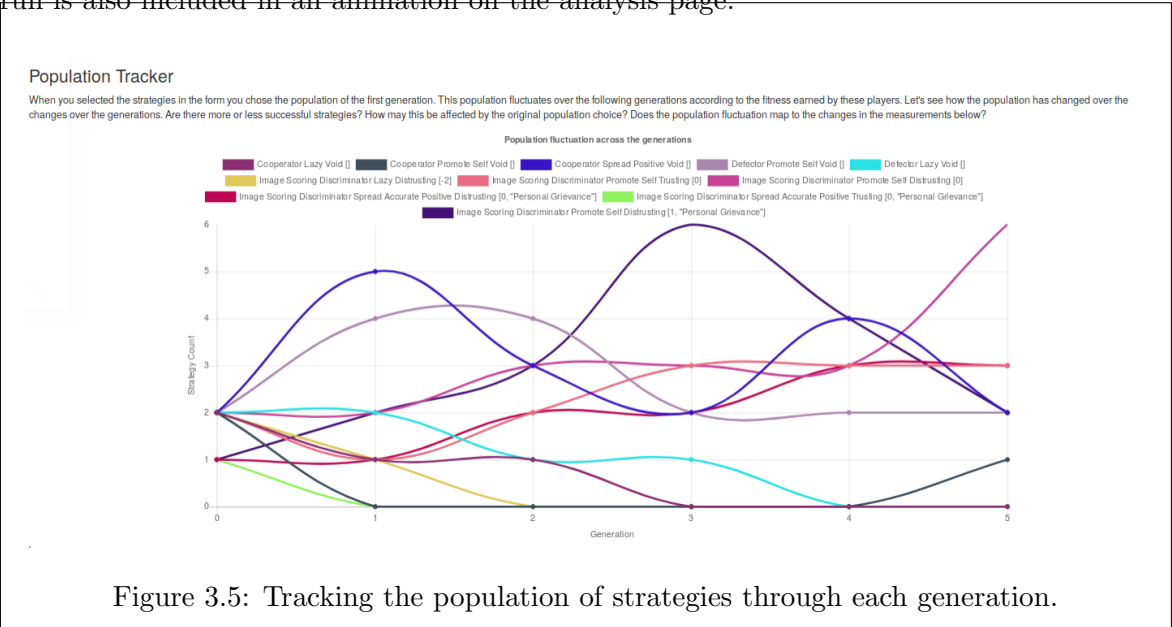
Figure 3.5: Tracking the population of strategies through each generation.

In addition to this I

have
cre-
ated
a
fea-
ture
that
al-
lows
users
to
create accounts, login and associates games of my theoretical framework to their account. These games can be given a label so they can keep track of the experiments the have run (illustrated in figure 3.6). User data is recorded in the database and passwords are hashed to keep them secret.

The web application, database and AMAAS system has been deployed to an Ubuntu Server and has a domain name (natureengine.tech) that is accessible to all across the internet.

3.3.3   Framework

Introduction

In this section I will lay out the implementation of the environment, agents and strategies I have described in the theoretical framework (section 3.2).

Environment and Agent's Bodies

The environment and agent's bodies from the theoretical framework described in section 3.2 has been implemented within the nature engine web application in it's own package. As such the environment and agent's bodies components have been written in Python.

This has taken an object oriented approach. A Community object is the overarching object that simulates the specified number of generations and handles the reproduction of strategies between generations assigning the strategies that should be associated with players in a Generation. A Generation object creates a set of players from these passed strategies.

The Generation is then simulated. Simulation of a generation involves running the synchronised cycle steps from the start timepoint of that generation to the end timepoint. In each perceive step a donor-recipient pair is selected at random by a method of the Generation object and the relevant agents are notified.

In the previous timepoint a set of percepts is generated - from the actions of that timepoint - for the current timepoint and are set into the perception bank ready to be transmitted at the current timepoint. These are sent in the perceive step. Agents are then asked to decide on an action at the current timepoint. Once these actions have been returned to the Generation they are executed, affecting players fitness and generating more percepts.

A Player is also an object in the system, and is associated with a strategy that describes how it should act at each timepoint. The Player object handles the sending of percepts and getting of actions from their decision making component in the AMAAS system. The Player object also encapsulates the player's state including the fitness of that player.

The 3 classes that implement the Action abstract class represent the three possible types of action that are available and the relevant data associated with an instance of an action.

Agent Mind as a Service (AMAAS)

The AMAAS system was developed in Prolog. This system handles the management of communities, generations and agents by assigning them with an ID generated using a similar system to the Prolog gensym library. These are assigned to the predicates $community(CommunityID)$, $generation(Community, GenerationID)$ where $Community$ unifies with a community and $agent(Strategy, Community, Generation, AgentID)$ where $Strategy$ unifies with a strategy in the system and Generation with a generation.

the community/1, generation/2 and agent/3 predicates are dynamic and can be asserted and retracted. The possible strategies that unify with $Strategy$ are not dynamic and are defined with a strategy predicate (strategy/5). The first 4 arguments are strings and are in the correct order: the strategy to use when a donor, the strategy to use when not a donor, the trust model, the description of the strategy. The final argument is a list of any type that contains options to augment the strategy that are not a necessity to define for every strategy.

Some of the available strategies (any associated with the donor strategies "Standing Discriminator", "Image Scoring Discriminator" and "Veritability Discerner") hold beliefs about other agents and the environment. These beliefs are stored with fluents in the MVFCEC. The value of these fluents are revised by the definition for initiates_at/4 and terminates_at/4 of that agents strategy and trust model and the events and conditions that cause these predicates to fire.

These predicates are caused to fire by the receiving of percepts to the agents service that then asserts a new event observation (the observed_at/2 predicate) and calling an update on the MVFCEC using the update_at/2 predicate. The process of firing these is controlled for all possible percepts by the add_new_action_gossip_percept/2, add_new_action_interaction_percept/2 and add_new_interaction_percept/2 predicates.

The beliefs formed by the firing of these predicates and the initiating and terminating of fluents is then used by the teleo-reactive program created for each strategy to decide on actions at certain points in time. An agent decides on an action by the firing of the agent_action/6 predicate. A set of agent_action/6 definitions is given for each donor and non-donor strategy. The first four arguments are input arguments being the timepoint to decide at, the id of the community and the generation the agent belongs to and finally the id of the agent.

The second to last argument stores true if a decision has been successfully made or provides an error message if it has not been. The last argument unifies with an action (the dictionary representation of that action). The program is teleo-reactive as the system is designed so only one of the bodies of the set of agent_action/6 definitions satisfies the conditions required, and this forces unification with one action.

The higher level predicate get_action/6 checks if an agent has already committed to an action at a specific timepoint. This predicate also ensures that an agent only commits to actions that it believes it is capable of using capable/5 predicate, which is only satisfied if the agent is committing to a donor action as a donor or a gossip or idle action if it is not a donor.

Simple Agent Gossip Language

The specification of SAGL (subsection 3.2.5) is remarkably similar to data structures used in Python (Dicts), the JSON structure and dictionaries in Prolog. As such, it made sense to implement SAGL by storing it as Python Dicts in the environment and agent's bodies, transferring it with JSON (as is used for the API in general) and then storing it as dictionaries in Prolog.

3.3.4   Software Engineering

Introduction

In this section I will discuss and reflect on engineering tools, techniques, methodologies and approaches that I have used, and some that I possibly should have used.

Process

The development process I decided on when starting the project was a scrum like structure, with a task backlog that was subject to change and fortnightly sprints. Although I have mostly stuck to these sprints, the backlog was in constant flux, far more than was expected. This is likely due to me still learning lots about the topic during the project. This was not due to a lack of research, but a focus on game-theory over multi-agent system design.

I feel that I could have managed this better with a more structured approach to design of the system, and by opening up my focus to all areas of the topic. However, I believe this structure allowed me to be open to new ideas and aspects to add to the project. I was then able to incorporate aspects of multi-agent systems and game-theory fields I may have not been able to with a more structured approach.

As it was, the design phase of the process was greatly interleaved with both learning the concepts for designing the system and the implementation of it. The implementation of the system then followed a Test-Driven Development (TDD) approach, developing tests and then creating code to work for those tests, and afterwards refactoring. This followed with a deeper more fully encompassing testing using a bottom-up integration testing strategy.

This highly interleaved process worked quite well for my project as it allowed me to design, develop and test features in the system separately all in the same slice of tasks. This process also allowed me to discover new features, ideas and strategies to incorporate in the system and then extend the project by adding a new slice of tasks to the task backlog which I could then choose when to implement based on urgency and importance.

I followed a bottom-up strategy when developing the components of the system. The multi-agent system is dependent on the agents service being fully functional and safe to use. As such I focused on developing and testing the AMAAS system and a base set of strategies for this first.

From this I then developed and tested a base set of components for the environment of the MAS. Once I had a safe and working base system I incorporated the MAS into the web application. From there I worked on adding new strategies and testing them in the AMAAS system, and then onto adding features to the multi-agent system such as the selection of onlookers and the reproduction mechanism.

This incremental development using slices of tasks for each feature allowed me to be

secure that the developed parts of the system were safe for use by other parts of the system.

Design

Most of the early design of my system went into learning how the operation of a MAS works, and drawing from multiple sources how to implement it myself. Further to this, I researched agent architectures and fused together multiple styles of design into my own. Both the design of the environment and agents can be viewed in my theoretical framework section (section 3.2).

I designed the API for the AMAAS system using a RAML specification and then added to this as I went along. This drove the design and implementation of the AMAAS system, by designing what I needed from the service I derived what I needed to implement for the functionality. I also created a theoretical design for the strategies in the system using logical predicates, to translate into Prolog.

The environment and agent's bodies for the multi-agent system were developed using the OOP features of Python. To approach this I used object-oriented design UML design tools (UMLet), and added to this as I enhanced the features of the system. This included a UML class diagram for designing the environment and agent body class structure (figure 3.7) and a sequence diagram to portray the message passing sequence (figure 3.8).

In the design of the environment classes I included an observer design pattern to observer player states and collect data on the player and their actions throughout the course of the game. I then created a facade for this so a simple Results class is returned after an object of the ReputationGame type has been run. The ReputationGame class is also a facade on the simulation of the communities and generations, making the system easy to set up a game with (preventing the need to know the underlying implementation).

This project is not focused on interface design and I have not researched or learnt any strategies for user experience design. However, with the knowledge I have of bootstrap components, css and some web development I developed templates for the client side as an informal method of design. I then refined these templates to improve the user experience simply by using the system and noticing any annoyances or issues I had with it.

Development

As I have mentioned the development phase has been strongly interleaved with testing and design to ensure the safety and correct running of the program. This included a TDD approach to development. I used a number of tools to assist me in this development.

The Python code that I have written follows the PEP 8 Style Guide. I used the Flask web application microframework to assist me in developing the web application and for the creation of the database I used the SQLAlchemy ORM. With SQLAlchemy way I was able to generate for both SQLite (for development) and MySQL (for deployment) databases from Python code. The use of SLQALchemy also assists in the prevention of SQL injection attacks that websites so often fall prey to.

I also used git as a version control system tool, with a master branch and various feature branches which I would develop features on and then once stable merge back into the master. The repository is hosted on GitHub. I mostly used the git command line tool for Linux, however I occasionally used the git support in the PyCharm integrated project support environment.

Results
-action_observer: ActionObserver
-player_observer: PlayerObserver
-observers: List[Observer]
-community: Community
+getGenIds(): List[int]
+getPlayerIds(): Dict[int, List[int]]
+getObservers(): List[Observer]
+getActions(): Dict[int, List[Action]]
+getActionsByGeneration():
  Dict[int, Dict[int, List[Action]]]
+getActionsByGenerationAndPlayer():
  Dict[int, Dict[int, Dict[int, List[Action]]]]
+getInteractions(): Dict[int, InteractionAction]
+getInteractionsByGeneration():
  Dict[int, Dict[int, InteractionAction]]
+getInteractionsByGenerationAndPlayer():
  Dict[int, Dict[int, Dict[int, InteractionAction]]]
+getCooperationRate(): Union[int, None]
+getCooperationRateByGeneration():
  Dict[int, Union[int, None]]
+getCooperationRateByGenerationAndPlayer():
  Dict[int, Dict[int, Union[int, None]]]
+getSocialActiveness(): Union[int, None]
+getSocialActivenessByGeneration():
  Dict[int, Union[int, None]]
+getSocialActivenessByGenerationAndPlayer():
  Dict[int, Dict[int, Union[int, None]]]
+getPositivityOfGossipPercentage(): Union[int, None]
+getPositivityOfGossipByGeneration():
  Dict[int, Union[int, None]]
+getPositivityOfGossipByGenerationAndPlayer():
  Dict[int, Dict[int, Union[int, None]]]
+getCorruptedObservations(): bool
+getFitness(): int
+getFitnessByGeneration(): Dict[int, int]
+getFitnessByGenerationAndPlayer():
  Dict[int, Dict[int, int]]
+getPopulations(): List[Dict[Strategy, int]]
+idToStrategyMap(): Dict[int, Dict[int, Strategy]]

ReputationGame
-firstGenStrategies:
  Dict[Strategy, int]
-mutation_chance: float
-num_of_onlookers: int
-num_of_generations: int
-length_of_generations: int
+getfirstGenStrategies():
  Dict[Strategy, int]
+getNumOfOnlookers(): int
+getLengthOfGenerations(): int
+getNumOfGenerations(): int
+getMutationChance(): float
+run(): Results

Community
-firstGenStrategies: Dict[Strategy, int]
-community_id: int
-mutation_chance: float
-num_of_onlookers: int
-num_of_generations: int
-length_of_generations: int
-generations: List[Generation]
-current_time: int
-generation_size: int
-strategy_count_by_gen:
  List[Dict[Strategy, int]]
-observers: List[Observer]
+getId(): int
+getNumOfOnlookers(): int
+getLengthOfGenerations(): int
+getGenerations(): List[Generation]
+getStrategyCountByGeneration():
  List[Dict[Strategy, int]]
+extendObservers(observers:
  List[Observer]): NoReturn
+simulate(): NoReturn
-buildGeneration(gen_id: int):
  Generation
-reproduce(gen_id: int):
  Generation

«Interface» Observer
+update(player_state: PlayerState):
  NoReturn
+getCommunityId(): int
+getGenerationId(): List[int]
+getPlayerIds(): Dict[int, List[int]]
+addGeneration(generation: int):
  NoReturn
+addPlayer(generation: int, player:
  int): NoReturn
+areObservationsCorrupted(): bool

ActionObserver
-community_id: int
-corrupted_observations: bool
-defection_count: int
-cooperation_count: int
-positive_social_action_count: int
-negative_social_action_count: int
-idle_action_count: int
-non_donor_action_count: int
+getActions(): Dict[int, List[Action]]
+getActionsByGeneration():
  Dict[int, Dict[int, List[Action]]]
+getActionsByGenerationAndPlayer():
  Dict[int, Dict[int, Dict[int, List[Action]]]]
+getInteractions(): Dict[int, InteractionAction]
+getInteractionsByGeneration():
  Dict[int, InteractionAction]
+getInteractionsByGenerationAndPlayer():
  Dict[int, Dict[int, Dict[int, InteractionAction]]]
+getCooperationRate(): Union[int, None]
+getCooperationRateByGeneration():
  Dict[int, Union[int, None]]
+getCooperationRateByGenerationAndPlayer():
  Dict[int, Dict[int, Union[int, None]]]
+getSocialActiveness(): Union[int, None]
+getSocialActivenessByGeneration():
  Dict[int, Union[int, None]]
+getSocialActivenessByGenerationAndPlayer():
  Dict[int, Dict[int, Union[int, None]]]
+getPositivityOfGossipPercentage(): Union[int, None]
+getPositivityOfGossipByGeneration():
  Dict[int, Union[int, None]]
+getPositivityOfGossipByGenerationAndPlayer():
  Dict[int, Dict[int, Union[int, None]]]

PlayerObserver
-community_id: int
-corrupted_observations: bool
-generation_ids: List[int]
+getFitness(): int
+getFitnessByGeneration(): Dict[int, int]
+getFitnessByGenerationAndPlayer():
  Dict[int, Dict[int, int]]

Generation
-generation_id: int
-community_id: int
-start_point: int
-end_point: int
-num_of_onlookers: int
-strategies: Dict[Strategy, int]
-players: List[Player]
-id_to_player_map: Dict[int, Player]
-observers: List[Observer]
+getId(): int
+getStartPoint(): int
+getEndPoint(): int
+getPlayers(): List[Player]
+getStategyCount(): Dict[Strategy, int]
+simulate(): NoReturn
-setAndSendDonorRecipientPair(timepoint:
  int): NoReturn
-execute(action: Action, timepoint: int):
  NoReturn
-generateOnlookers(action:
  InteractionAction): List[int]

PlayerState
-generation: int
-player: int
-observers: List[Observer]
-new_action: Action
-fitness_update: int
+attach(observer: Observer):
  NoReturn
+detach(observer: Observer):
  NoReturn
-notify(): NoReturn
+getGeneration(): int
+getPlayer(): int
+getNewAction(): Action
+setNewAction(action: Action):
  NoReturn
+getFitnessUpdate(): int
+setFitnessUpdate(fitness_update:
  int): NoReturn

Player
-community_id: int
-generation_id: int
-player_id: int
-percepts: Dict
-fitness: int
+player_state: PlayerState
-strategy: Strategy
+getId(): int
+getFitness(): int
+updateFitness(change: int):
  NoReturn
+getStrategy(): Strategy
+decide(timepoint: int): Action
+setPerception(percept: Dict):
  NoReturn
+perceive(timepoint: int):
  NoReturn

Strategy
-donor_strategy: str
-trust_model: str
-non_donor_strategy: str
-options: List[Any]
+toString(): str
+equals(): bool
+notEquals(): bool
+hash(): hashSequence

«Abstract» Action
-timepoint: int
-actor: int
-generation: int
-reason: str
+getType(): ActionType
+getGeneration(): int
+getTimepoint(): int
+getActor(): int
+getReason(): str

«Enumeration» ActionType
INTERACTION
GOSSIP
IDLE

IdleAction
+getType(): ActionType

GossipAction
-gossiper: int
-about: int
-recipient: int
-gossip: GossipContent
+getType(): ActionType
+getGossiper(): int
+getAbout(): int
+getRecipient(): int
+getGossip(): GossipContent

InteractionAction
-donor: int
-recipient: int
-action: InteractionContent
-onlookers: List[int]
+getType(): ActionType
+getDonor(): int
+getRecipient(): int
+getAction(): InteractionCon
+getOnlookers(): List[int]
+setOnlookers(onlookers: Lis

«Enumeration» GossipContent
POSITIVE
NEGATIVE

«Enumeration» InteractionContent
COOPERATE
DEFECT

uses ▶   *◀ builds*   *▼ runs*   *uses ▶*   *◀ uses*   *observes ▶*   *uses ▶*   *uses ▶*   *uses ▶*   *gen uses action*   *◀ uses*   *uses ▶*   *◀ uses*   *◀ commit to*   *uses ▶*   *◀ uses*   *◀ uses*
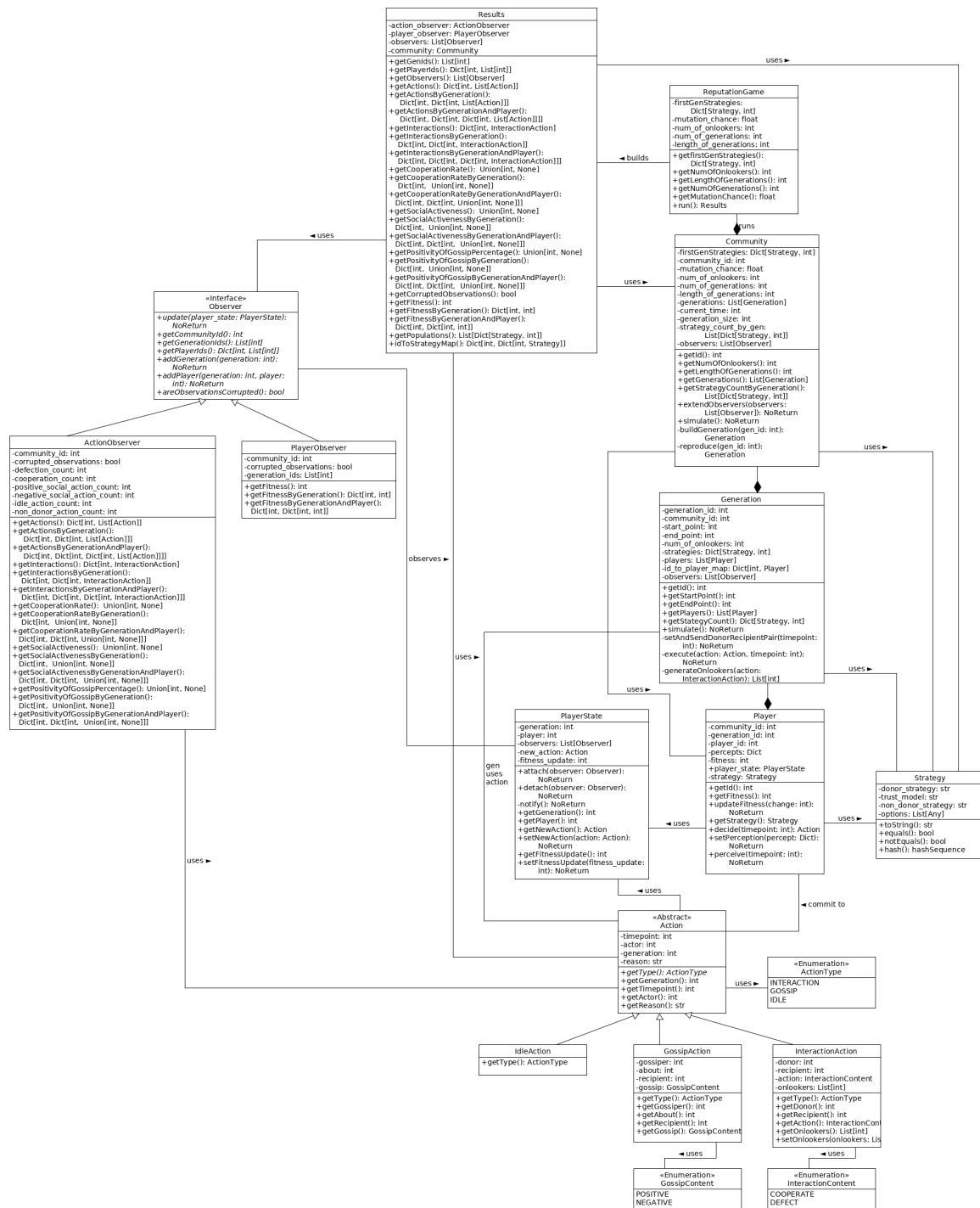
Figure 3.7: The class diagram for my environment and agent bodies Python program, also can be found in the ProjectReports/NewFinalReport directory as EnvClass.png
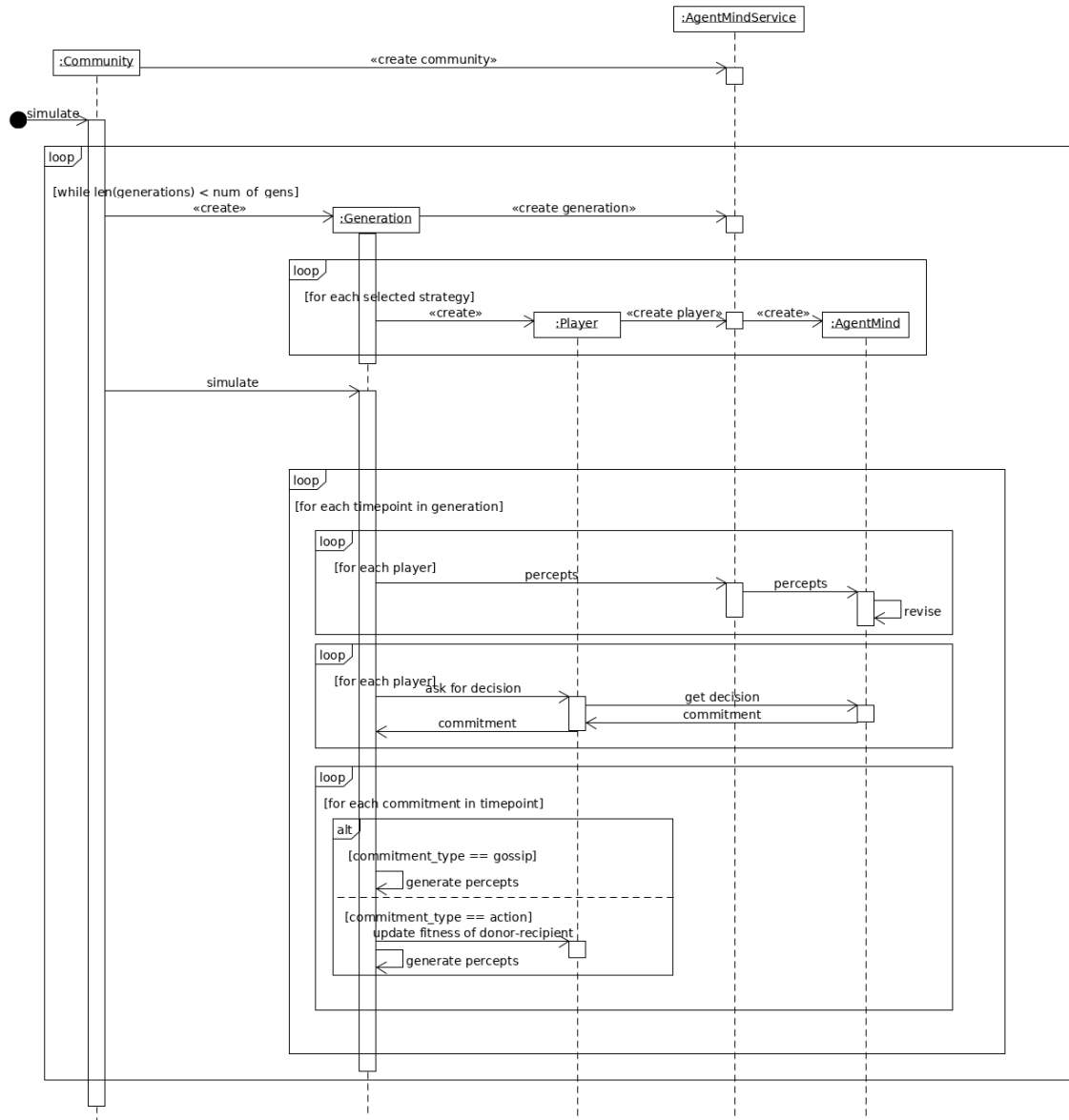
Figure 3.8: The sequence diagram for my environment and agent bodies Python program, also can be found in the ProjectReports/NewFinalReport directory as EnvSequence.png

Pycharm provides more than just version control system tools. There is also a linter and support for the Python language, as well as Jinja2, HTML, CSS and Javascript which I used for front-end development. I have often used the visual debugger, Python environment setup support, Flask support and unittest library support.

There is far less tooling available for Prolog development. I ended up using a mix of Visual Studio Code as it has a linter that helped me catch errors earlier (though I would not recommend this, as it often crashes) and Sublime Text (limited to syntax highlighting).

Again with the development of the AMAAS system I followed a TDD approach, writing unit tests using the PLUnit library. I documented this code using the PLDoc library. There is a good HTTP library for Prolog, that allowed me to easily create a web service with handlers for certain routes that easily link up to functionality in the AMAAS system.

Testing

The testing of my system can be broken down into four parts: AMAAS bottom up integration testing, environment bottom up integration testing, application programming interface (API) testing and some extra early unit testing on other parts of the web application. The bulk of this testing comes from the first two, which were interleaved with the development of the system. The early unit testing was also interleaved with development. However API testing was carried out after development.

As the AMAAS system has been developed in Prolog, the unit testing has been done with the PLUnit library. This focused on testing the functionality provided by the predicates I have created in the system. In depth testing has been undertaken for every strategy, in many different possible situations and timelines. I have also tested all predicates for managing agents, generations and strategies in the system as well as the predicates used for receiving percepts and getting actions.

Further to this the AMAAS system has been web API tested. The functionality that the API uses in the system has been thoroughly tested in the bottom up integration testing. As such, API testing focused on ensuring the API met the specification laid out in the documentation. I carried out this testing using the python unittest testing framework and the requests library. I had previously planned on using an API testing tool such as postman, but had found no free and easy way to put my tests into code to put on my github repository.

I also used this testing framework for bottom up integration testing of the environment for my game. This testing also requires the AMAAS system to be running and working correctly so acts as integration testing between the two systems. The testing was interleaved with development in a way that as I created each unit (class) or added functionality to that unit I added testing for the functionality added. I used the same framework for some earlier testing in the web app. This was mostly testing for converting produced game data for the database.

I would also like to have tested the web application including functionality, usability and compatibility testing [3, 4]. I have developed the user interface to what I believe to be a good standard, but I'm sure it can be improved and it would be good to get feedback to understand where I can improve this. I also have developed the web application so it is responsive using the Bootstrap 3 library, and have used the website on Firefox on Android and Linux, and Chrome on Windows. It would be good to find out the compatibility with Safari and have more eyes to spot bugs in the functionality.

Documentation

In the API testing phase I tested the API against the claimed functionality in the API documentation. I created the specification for this documentation in the RESTful API Modelling Language (RAML) which is based on YAML specifically for the creation of RESTful API specifications. There is a good set of tools surrounding RAML, including a linter in the Atom text editor and the raml2html tool that creates a nicely laid out web page for a user readable documentation. I used this tool to convert my .raml file into a .html for readability. This documentation can be found by opening the main.html document in the AgentsService/api_docs directory in a web browser.

I have also created documentation for the web application code. This is done in the form of writing python docstrings in the reStructuredText markup language for each package, module, class and function. These docstrings follow the "PEP 257 Docstring Conventions" proposal. I have then used the documentation generator to convert these docstrings into html format that could then be published. This documentation can be found by opening the index.html file in the NatureEngineWebApp/docs/build/html directory in a web browser.

Similarly to this I have created documentation for the AMAAS system code using PlDoc. PlDoc docstrings are similar in format to those in Javadoc. Once written I have used the utilities provided by SWI-Prolog to convert to html format. This documentation can be found by opening the index.html file in the AgentsServer/pldocs directory in a web browser.

Agent Oriented Programming

The software engineering techniques and methodologies I have used in my development have often followed a typical OOP design approach. I was not aware of the methodologies and techniques that exist for the AOP paradigm. On reflection adopting one of these methodologies to develop the environment and agents would have assisted me in the development and design process.

Many of these methodologies such as Tropos and Prometheus are extensions to the OOP methodologies [56]. Tropos is a full methodology but relies on UML extended with the Agent Unified Modelling Language (AUML). Using this would have allowed for easy integration with my traditional OOP design approach, while driving the MAS development with Tropos' four stages (early requirements, late requirements, architectural design and detailed design).

Chapter 4: **Experiment Evaluation**

Explain subset I have taken and why I have chosen it (middle, good and not so good) (Ratio for strategies and eighth) (Duplicate experiments) (R, IV, S)
Because the environment is not visible and there is so much random chance in the system unlike other game-theoretic models, wildly different results are produced each time. Affirming that past game-theoretic analysis of systems isn't adequate.

4.1   Introduction

4.2   Robustness

The experiments I have run are similar to Axelrod and Hamilton's [8] experiments on the robustness of strategies in the system. There are 199 strategies available to use in my system, if I included one of each of these in the experiments it would overload the system. As such, I selected a population using a subset of these strategies to for the experiments that remains the same throughout the robustness experiments (listed in appendix section **??**).

9 unique experiments were conducted with this population and each one was duplicated - I would have liked to triplicate the experiments, however due to time pressure this was not possible. The first 3 experiments ran for 10 generations with a generation length of 60. These 3 experiments were different due to the number of onlookers selected for each on - 5, 10 and 20 respectively.

The next 3 experiments were reduced to a generation number of 5 due to the time it took to run them with 10 generations. These experiments were conducted with a generation length of 120. Again the number of onlookers was changed between each with the values 5, 10 and 20 respectively. The final set of 3 experiments were again conducted with a generation number of 5 but this time with a generation length of 200, and the same selected number of onlookers for each as the other sets of 3 experiments.

These experiments fulfilled a number of purposes. One of these was to review my framework for how well it supports the evolution of cooperation, another was to review which strategies were most consistently successful and additionally I evaluated the effect gossip and trust models to interpret this gossip have on the games.

Remark: Consistency of Results

Originally I ran a number of experiments at generation length 20. Players in these games reached a maximum total fitness of 35. These low fitness scores did not provide the reproduction algorithm with much chance of reflecting how successful a strategy can be with the amount they were reproduced into the next generation. This led to a remarkably low level of reproducibility of results. Even duplicate experiments returned wildly dissimilar results.

I diregarded these results for the following reasons. If a particular strategy got lucky with an early round of reproduction, it is likely that they would dominate the society later on as agents using that strategy are more likely to be randomly chosen in the reproduction algorithm before being stochastically accepted or not.

Another factor in this is the way interactions are selected. The donor-recipient pairs are selected at random from the population. If an agent or multiple agents of the same strategy were lucky enough to be selected a number of times as a recipient and receive cooperation, they are more likely to be reproduced out of pure luck over a good strategy.

4.2.1   Evaluation: Generation Length

4.2.2   Evaluation: Evolution of Cooperation and The Framework

4.2.3   Evaluation: Strategies

Defection and cooperation able to exist alongside each other

4.2.4   Evaluation: Trust Models and Gossip

4.3   Discussion

Chapter 5: **Critical Analysis and Discussion**

Project achievements
Reflection on project process (difficulties, success, failure)
Successful?
Future enhancements
Focus on project
Compare to deliverables at beginning
Reflect on initial focus on game theory over MASs
Low fitness values suggest I need a better measure for how well the agents are doing.

Chapter 6: **Conclusions**

Mechanism has to work with and without re-meeting.

Mechanism has to encourage cooperation between unknown entities.

Opportunity for extra work? Further exploration of network reciprocity as outline in background and background summary. Extend indirect reciprcoty so individuals can choose who they wish to interact with. Symbolic reinforcement learning strategies [22].

Move closer to using an agent model such as BDI?

Using the GAIA methodology for development? [63].

Reaching agreements

Use an AOP language (GOAL or SARL?)

It's difficult to run a genetic algorithm with a multi-agent system in a web server, because it requires so many resources, next time I may develop a desktop application or command line application which are easily distributable to put onto powerful systems without needing to think about timeout for security (could create a mix). Make it easy to run automated experiments (Run experiments on an API????)!!!!! I have developed it in a way that the code could easily be ported for this. Also use a faster language that allows parrallelisation of perceptions, decisions, etc.

Have agents select who to interact with rather than a random choice.

A better fitness measure for the algorithm?

Lo

Bibliography

[1] The new and improved flask mega tutorial. `https://courses.miguelgrinberg.com/p/flask-mega-tutorial`, 2018. Accessed: 05/12/2018.

[2] Tutorial - creating web applications in swi-prolog. `http://www.pathwayslms.com/swipltuts/html/index.html`, 2018. Accessed: 05/12/2018.

[3] Web application testing: 8 step guide to website testing. `https://www.guru99.com/web-application-testing.html`, 2018. Accessed: 04/01/2019.

[4] Web application testing complete guide (how to test a website). `https://www.softwaretestinghelp.com/web-application-testing/`, 2018. Accessed: 04/01/2019.

[5] Richard Alexander. The biology of moral systems (foundations of human behavior). 1987.

[6] Alexander Artikis, Marek Sergot, and Jeremy Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic (TOCL)*, 10(1):1, 2009.

[7] John Langshaw Austin. *How to do things with words*. Oxford university press, 1975.

[8] Robert Axelrod and William D. Hamilton. The evolution of cooperation. *Science*, 211:1390–1396, 1981.

[9] Roy F Baumeister, Liqing Zhang, and Kathleen D Vohs. Gossip as cultural learning. *Review of general psychology*, 8(2):111, 2004.

[10] Michael Bratman. *Intention, plans, and practical reason*, volume 10. Harvard University Press Cambridge, MA, 1987.

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[12] Rodney A Brooks. Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159, 1991.

[13] Vince Knight; Owen Campbell; Marc; eric-s-s; VSN Reddy Janga; James Campbell; Karol M. Langner; T.J. Gaffney; Sourav Singh; Nikoleta; Julie Rymer; Thomas Campbell; Jason Young; MHakem; Geraint Palmer; Kristian Glass; edouardArgenson; Daniel Mancia; Martin Jones; Cameron Davidson-Pilon; alajara; Ranjini Das; Marios Zoulias; Aaron Kratz; Timothy Standen; Paul Slavin; Adam Pohl; Jochen Mller; Georgios Koutsovoulos; Areeb Ahmed. Axelrod: 4.3.0. `http://dx.doi.org/10.5281/zenodo.1405868`, September 2018.

[14] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463. ACM, 1994.

[15] Michael Fisher. Concurrent metatema language for modelling reactive systems. In *International Conference on Parallel Architectures and Languages Europe*, pages 185–196. Springer, 1993.

[16] Pascal Francq. Genetic algorithms. `http://www.otlet-institute.org/wikics/Genetic_Algorithms.html`, 2011. Accessed: 04/01/2019.

[17] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

[18] The Photographers from Mpala Live `http://mpalalive.org/sponsors`. Green wood hoopoe. `http://mpalalive.org/field_guide/green_woodhoopoe`, January 2019.

[19] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[20] Michael P Georgeff and Amy L Lansky. Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682, 1987.

[21] Mr. David Gunning. Explainable artificial intelligence. `https://www.darpa.mil/program/explainable-artificial-intelligence`, 2018. Accessed: 8/3/2019.

[22] Marc Harper, Vincent Knight, Martin Jones, Georgios Koutsovoulos, Nikoleta E Glynatsi, and Owen Campbell. Reinforcement learning produces dominant strategies for the iterated prisoners dilemma. *PloS one*, 12(12):e0188046, 2017.

[23] Demis Hassabis and Silver David. Alphago zero: Learning from scratch. *deepMind official website*, 18, 2017.

[24] Koen V Hindriks, Frank S De Boer, Wiebe Van Der Hoek, and John-Jules Ch Meyer. Agent programming with declarative goals. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 228–243. Springer, 2000.

[25] Nicholas R Jennings. On agent-based software engineering. *Artificial intelligence*, 117(2):277–296, 2000.

[26] Özgür Kafali, Alfonso E. Romero, and Kostas Stathis. Agent-oriented activity recognition in the event calculus: an application for diabetic patients. *Computational Intelligence*, 33:899–925, August 2017.

[27] Robert Kowalski and Marek Sergot. A logic-based calculus of events. In *Foundations of knowledge base management*, pages 23–55. Springer, 1989.

[28] Su-Dong Lee, Ji-Hyung Lee, Young-Geun Choi, Hee-Cheon You, Ja-Heon Kang, and Chi-Hyuck Jun. Machine learning models based on the dimensionality reduction of standard automated perimetry data for glaucoma diagnosis. *Artificial Intelligence in Medicine*, 18, 2019.

[29] Olof Leimar and Peter Hammerstein. Evolution of cooperation through indirect reciprocity. *Proceedings of The Royal Society*, 268:745–753, May 2001.

[30] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[31] Computing Machinery. Computing machinery and intelligence-am turing. *Mind*, 59(236):433, 1950.

[32] Pattie Maes. The agent network architecture (ana). *SIGART Bull.*, 2(4):115–120, July 1991.

[33] Theo C. M. Bakker Manfred Milinski, Dirk Semmann and Hans-Jürgen Krambeck. Cooperation through indirect reciprocity: image scoring or standing strategy? *Proceedings of The Royal Society*, 268:2495–2501, May 2001.

[34] Arjun Mani. What is the prisoner's dilemma in layman's terms? what are its applications? what are it's applications? `https://www.quora.com/What-is-the-prisoners-dilemma-in-laymans-terms-What-are-its-applications`, September 2013.

[35] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[36] Lik Mui. *Computational models of trust and reputation: Agents, evolutionary games, and social networks.* PhD thesis, Massachusetts Institute of Technology, 2002.

[37] Roger B Myerson. *Game theory.* Harvard university press, 2013.

[38] Nils Nilsson. Teleo-reactive programs for agent control. *Journal of artificial intelligence research*, 1:139–158, 1993.

[39] Martin Nowak and Karl Sigmund. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364:56–58, July 1993.

[40] Martin A Nowak. *Evolutionary dynamics.* Harvard University Press, 2006.

[41] Martin A. Nowak. Five rules for the evolution of cooperation. *Science*, 314, 2006.

[42] Martin A. Nowak and Karl Sigmund. Evolution of indirect reciprocity by image scoring. *Nature*, 393:573–577, 1998.

[43] Paul D O'Brien and Richard C Nicol. Fipatowards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.

[44] Steve Phelps. An empirical game-theoretic analysis of the dynamics of cooperation in small groups. *Journal of Artificial Societies and Social Simulation*, 2016.

[45] Anatol Rapoport, Albert M Chammah, and Carol J Orwant. *Prisoner's dilemma: A study in conflict and cooperation*, volume 165. University of Michigan press, 1965.

[46] Gilbert Roberts. Evolution of direct and indirect reciprocity. *Proceedings of The Royal Society*, 275:173–179, September 2008.

[47] Sebastian Rodriguez, Nicolas Gaud, and Stéphane Galland. Sarl: a general-purpose agent-oriented programming language. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 3, pages 103–110. IEEE, 2014.

[48] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[49] John R Searle and John Rogers Searle. *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press, 1969.

[50] Yoav Shoham. Agent0: A simple agent language and its interpreter. In *AAAI*, volume 91, page 704, 1991.

[51] Yoav Shoham. Agent-oriented programming. *Artificial intelligence*, 60(1):51–92, 1993.

[52] Barry G Silverman, Nancy Hanrahan, Gnana Bharathy, Kim Gordon, and Dan Johnson. A systems approach to healthcare: agent-based modeling, community mental health, and population well-being. *Artificial intelligence in medicine*, 63(2):61–71, 2015.

[53] Munindar P Singh. Agent communication languages: Rethinking the principles. *Computer*, 31(12):40–47, 1998.

[54] Ralf D. Sommerfeld, Hans-Jürgen Krambeck, Dirk Semmann, and Manfred Milinski. Gossip as an alternative for direct observation in games of indirect reciprocity. *Proceedings of the National Academy of Sciences of the United States of America*, 104:17435–17440, 2007.

[55] Herbert Spencer. The principles of biology (2 vols) london: Williams and norgate. *Google Scholar*, 1864.

[56] Professor Kostas Stathis. Lecture notes in intelligent agents and multi-agent systems, November 2018.

[57] Robert Sugden et al. *The economics of rights, co-operation and welfare.* Springer.

[58] Sarah Rebecca Thomas. Placa, an agent oriented programming language. 1993.

[59] Arne Traulsen and Martin A. Nowak. Evolution of cooperation by multilevel selection. *Proceedings of the National Academy of Sciences of the United States of America*, 103:10952–10955, 2006.

[60] Robert L Trivers. The evolution of reciprocal altruism. *The Quarterly review of biology*, 46(1):35–57, 1971.

[61] Michael Wooldridge. *An introduction to multiagent systems.* John Wiley & Sons, 2009.

[62] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[63] Michael Wooldridge, Nicholas R Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, 3(3):285–312, 2000.

[64] Sea Turtle World. Green sea turtle cleaning station. `https://www.seaturtle-world.com/green_sea_turtle_cleaning_station/green_sea_turtle_cleaning_station_600/`.

Chapter 7: **Professional Issues**

are current machine learning methods unethical? Deep symoblic reinforcement learning you can query actions, like I have done.
Replacement of people in jobs?
AI issues and risks?
Relate to project
Moral philosophy of agent societies
Refer to last chapter of russell and norvig

Chapter 8: **Appendix**

Describe the contents of my appendix.

8.1   Background

8.1.1   Kinship Theory

8.1.2   Network Reciprocity

	A	B
A	a	b
B	c	d

Table 8.1: The payoff matrix for when individuals interact. Cooperators are in blue and are called A, and defectors in red and called B. Taken from Nowak's book Evolutionary Dynamics [40].

8.1.3   Group Selection

8.1.4   Alternate Agent Architectures

8.2   Theoretical Framework

8.2.1   Environment Properties

8.2.2   Strategies

include list of strategy components and trust models, all associated together.

8.3   Robustness Experiments Population

- 1 x Cooperator, Lazy

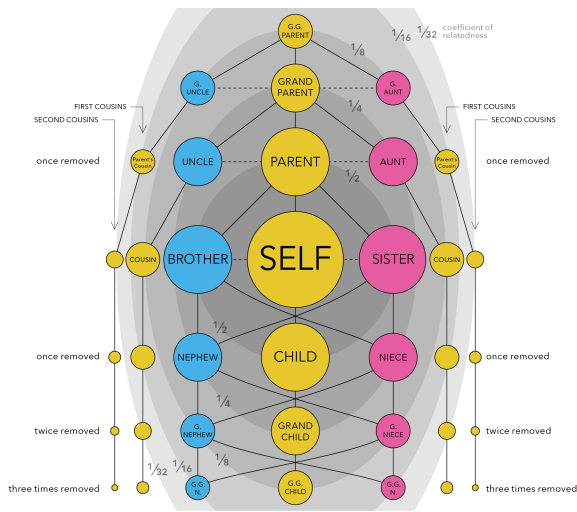- 1 x Cooperator, Promote Self

- 2 x Defector, Lazy

Figure 8.1: Wright's coeffcient of relatedness by Citynoise - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=37723128

Figure 8.2: The Green Wood-Hoopoe native to Africa participates in cooperative breeding as the bird not only looks after its own chicks, but those of other breeding pairs [18]
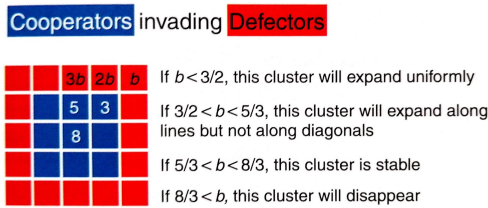

Figure 8.3: How can cooperators invade defectors? Taken from Nowak's book Evolutionary Dynamics [40]. The squares represent nodes and the players interact with the players to each side of them and diagonally. The value b is from the payoff matrix in table 8.1
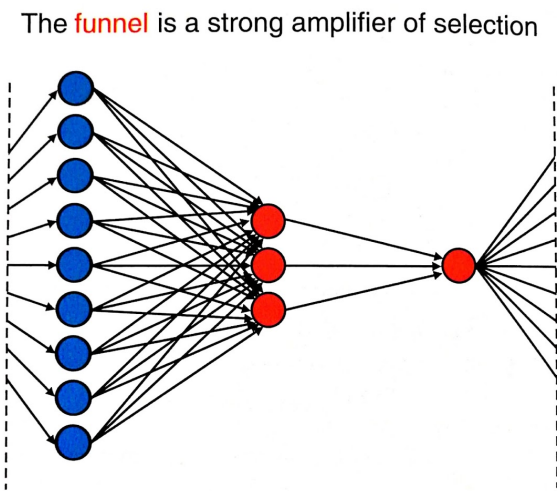
Figure 8.4: Shapes that can amplify selection include the funnel, the star and the super-star [40]
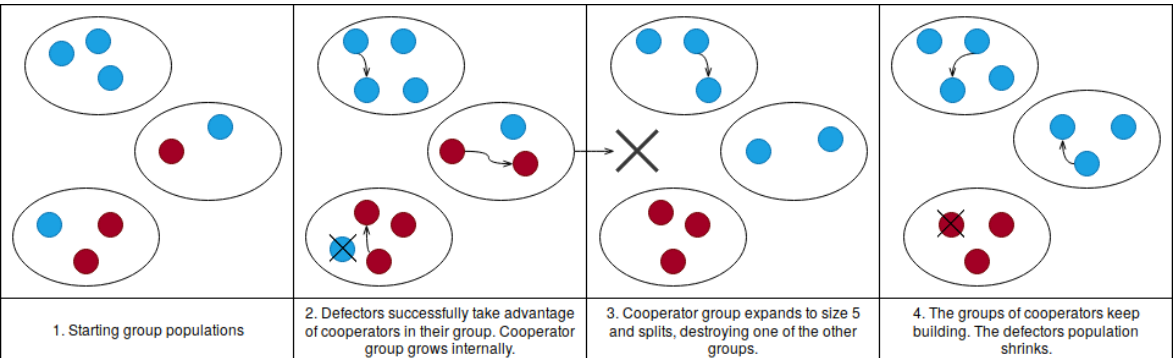

Figure 8.5: The dynamics of multi-level selection as described by Traulsen and Nowak [59].

- 2 x Defector, Promote Self

- 2 x Defector, Spread Negative

- 2 x Random

- 1 x Standing Discriminator, Promote Self, Trusting

- 1 x Standing Discriminator, Spread Accurate Negative, Trusting

- 1 x Standing Discriminator, Lazy, Distrusting

- 1 x Standing Discriminator, Lazy, Naive Trusting

- 1 x Image Scoring Discriminator, Lazy, Distrusting, K=-2

- 1 x Image Scoring Discriminator, Promote Self, Trusting, Personal Grievance, K=-2

- 1 x Image Scoring Discriminator, Spread Accurate Negative, Trusting, Personal Grievance, K=-2

- 1 x Image Scoring Discriminator, Lazy, Trusting, K=0

- 1 x Image Scoring Discriminator, Promote Self, Trusting, K=0

- 1 x Image Scoring Discriminator, Promote Self, Naive Trusting, Personal Grievance, K=0

- 1 x Image Scoring Discriminator, Lazy, Naive Trusting, K=2

- 1 x Image Scoring Discriminator, Spread Accurate Negative, Trusting, K=2, Personal Grievance

- 1 x Image Scoring Discriminator, Spread Accurate Positive, Distrusting, K=2, Personal Grievance

- 1 x Veritability Discerner, Spread Negative Untrusted, Strong Reactor, K=-5

- 1 x Veritability Discerner, Spread Positive Trusted, Balanced Reactor, K=-5

- 1 x Veritability Discerner, Lazy, Forgiving Reactor, K=-5

- 1 x Veritability Discerner, Lazy, Strong Reactor, K=0

- 1 x Veritability Discerner, Spread Negative Untrusted, Balanced Reactor, K=0

- 1 x Veritability Discerner, Spread Negative Untrusted, Forgiving Reactor, K=0

- 1 x Veritability Discerner, Lazy, Balanced Reactor, K=5

- 1 x Veritability Discerner, Promote Self, Forgiving Reactor, K=5

- 1 x Veritability Discerner, Spread Positive Trusted, Forgiving Reactor, K=5