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

April 2, 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 Where other published and unpublished source materials have been used, these have been acknowledged.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: 1,152 intro + 1,837 agents background + 3,333 game theory background + 270 intro and summary background + 3,509 theoretical framework + 2,124 implementation framework and development + 1,877 software eng + 1,634 experiment evaluation + 2,381 critical analysis, discussion and conclusion + 1,238 professional issues

Student Name: James King

Date of Submission: April 2, 2019

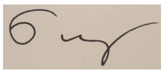
Signature: 

Table of Contents

Abstract

Multi-agent systems is a field of computer science that concerns itself with the development of societies of autonomous agents. Agents within these societies often find themselves in interactions with other agents, in which both agents often act strategically to increase their utility or payoff. We can model these strategic interactions using game-theoretic models, which enables us to research the techniques we can use to ensure societies of agents are co-operative.

In this report we delve into the background of multi-agent system development and game theory. We then use this background to formulate and present a theoretical framework for a multi-agent system. This theoretical framework includes a number of agent strategies interacting in a game like environment that incorporates a mixture of game-theoretic mechanisms. We then delve into the implementation that has taken place to build this framework into a group of interacting software components.

After this, we evaluate and discuss the results of a number of experiments. These experiments focus on testing the abilities of the mechanisms built into the environment and the agent strategies in facilitating the evolution of cooperation within a society of agents. Finally we critically analyse and discuss the projects achievements and qualities, and give a conclusion on the overall success of the project.

We found that the implementation of the model presented in the theoretical framework did not facilitate stable cooperation between either initially varied or predominantly cooperative societies. This may be due to a number of reasons, and we believe the lack of stable co-operation highlights the difficulty of creating agent strategies and methods of interpretting perceptions in complex environments.

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 [36] and game playing [30].

Many of these applications are specifically using agent techniques [65, 33, 15]. Intelligent agents (IAs) have many definitions but one popular definition is that agents are anything that perceives and acts upon its environment [61]. 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 [47]. 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 [73], 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 [53] 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 [68].

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 architecture 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. How to run the application and a user manual is included in the appendix in section 7.4. In the experimental evaluation 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. I then critically discuss and evaluate the project and provide conclusions on it. 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, agent environments and agent communication. Secondly I will discuss game-theoretical approaches to modelling and studying multi-agent interactions in regards to the evolution of cooperation between agents.

2.2 Multi-Agent Systems and Intelligent Agents

MASs is a field in computer science concerning the development of societies of agents. Traditional AI techniques often ignored the social aspects that often arise from intelligent systems. The field of MASs focuses on both the development of agent systems and the interactions and communication between the agents. Here I will explore the agent development and societal interactions background relevant to this project.

2.2.1 Intelligent Agents and Their Environment

What is an Agent?

One of Russell and Norvig's properties [61] of an environment is whether the environment is multi-agent or single-agent. How we classify an environment as to this property is very much determined by what we class as an agent, which is not a trivial question. How do we define agency?

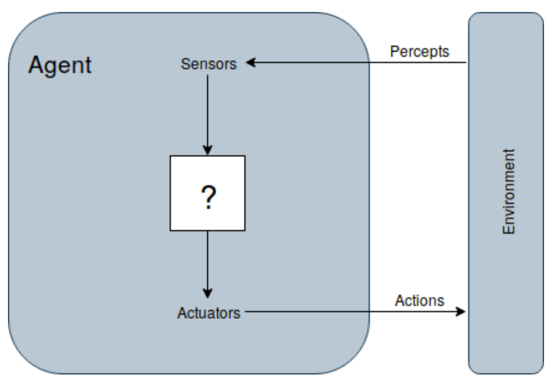


Figure 2.1: Russell and Norvig's [61] agent

The definition of agency greatly differs depending upon which source is consulted and the application of the concept. Shoham [64] supported the notion of agents being ascribed mental characteristics such as beliefs, free will, intentions etc. There is a disparity between Shoham's mental characteristics and Brooks' argument [16] against the use of abstractions in the creation of AI systems. Brooks' argument mostly focused on the symbolic AI representation of real world data, but it extends to Shoham's mental characteristics as these are arguably symbolic AI abstractions for agents.

Brooks believes that intelligence is an emergent property of complex systems and that the development of AI systems should focus on the more difficult side - perception and motor-skills. Intelligence is a term that is difficult to define, and a lot of later MAS research has not defined agents in terms of their intelligence.

Wooldridge and Jennings [76] saw the definition of agents on a sliding scale. Their weak notions of agency provides the base properties by which an agent can be defined: 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 or other interaction medium to communicate 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.

Stronger notions of agency may include higher levels of 'intelligence' including that of human-like mental notions [76] similar to Shoham's mental characteristics or even a complex system with intelligence as an emergent property like Brooks. Advancements in techniques and technology over recent years are pushing the boundaries of these strong notions with developments such as in computer vision, natural language processing and deep reinforcement learning.

Some notions of agency define an agent in terms of it's environment. These definitions include Russell and Norvig's [61] in which agents are a system that uses sensors to perceive it's environment and actuators to act upon it (figure 2.1).

Agent Architectures

An agent architecture is defined by Pattie Maes [39] as a methodology for developing agents and a set of modules used to build an agent and a specification for how these interact. Shoham formulated his Agent-Oriented Programming paradigm [64] surrounding his ideas of mental characteristics and for this paradigm he designed a language to guide agent development know as Agent 0 [63].

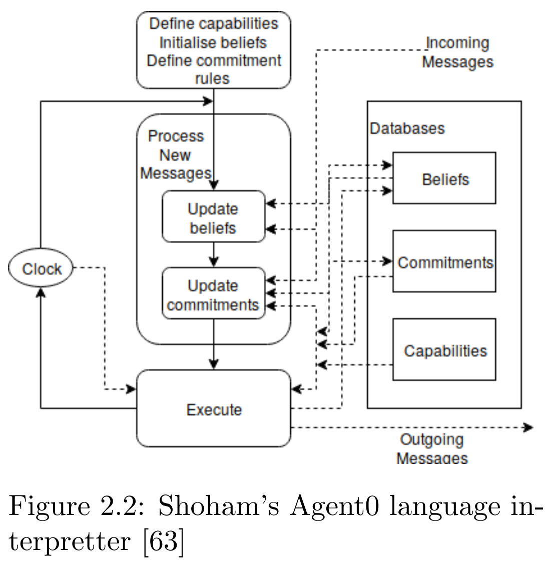


Figure 2.2: Shoham's Agent0 language interpretter [63]

The Agent0 language used an interpretter that defined the architecture of an agent created with the Agent0 language, as illustrated in figure 2.2. Agent0 also defines the communication between agents. The clock in the interpretter is used to synchronise the perceive and act cycles of agents using a cycle number.

Agent0 is very limited in terms of endowing agents with the ability to plan and as such a successor language PLACA (Planning Communicating Agents) was developed [71]. Both these languages utilise a symbolic AI approach known as deductive reasoning in which the symbolic representation of information is in the form of logical formulae and the reasoning component is done through logical deduction or theorem proving [75].

Rodriguez *et al.* [58] criticised agent-programming languages for forcing specific agent architectures on users of the language. Their belief is that programmers should have a language which allows them to choose the architecture the wish and created a generic agent language (SARL) which aims to be architecture independent.

Rodriguez *et al.*'s argument is aimed at, but not limited to, languages that use the beliefs-desires-intentions (BDI) model [14]. The BDI model can be used by the developers of agent systems to develop practical reasoning agents. Similarly to Agent0 the BDI model ascribes mental characteristics to agents - beliefs, desire and intentions.

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

Table 2.1: An example teleo-reactive program

Beliefs are the information an agent sees as fact about the world, these can change and can be inferred from the presence of other beliefs. Desires drive actions and are an objective state an agent wishes to reach. Actively pursued desires are goals, these goals cannot contradict each other. Agents commit to desires i.e. are executing a plan to achieve them. When the commitment has been made the desires become intentions. This model has inspired a number of languages and frameworks including GOAL [31] and PRS (The Procedural Reasoning System) [25].

The subsumption architecture was developed by Brooks [16] as a reaction to architectures that use symbolic AI approaches such as that of BDI and Agent0. This architecture was built in a hierarchical structure of behaviours which can fire at the same time. Say three behaviours fire at one given time, an agent selects the behaviour with the lowest level as it has the greatest priority.

Teleo-reactive programs [48] follow the same attitude as the subsumption architecture but replace the layers with an ordered set of production rules which map from conditions to actions. The first condition to be satisfied in the defined order maps to the action that the agent will take.

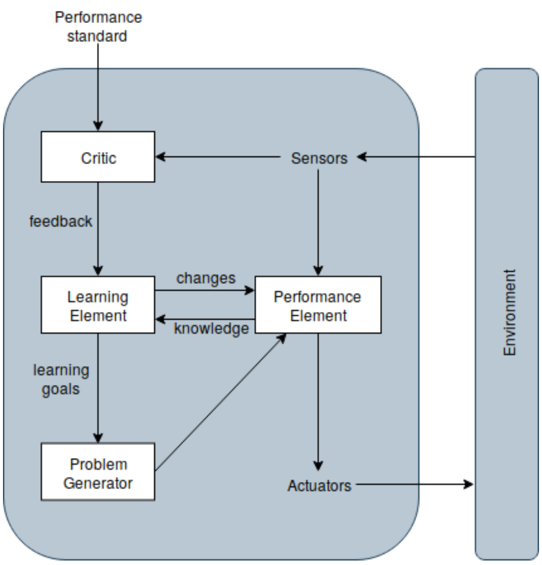


Figure 2.3: Russell and and Norvig's [61] learning agent

A criticism of reactive architectures [75] 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 implement hybrid architectures is to use free variables in teleo-reactive program conditions, with which beliefs bind to at runtime.

Russell and Norvig [61] presented a number of architectures. Their simple reflex agents keep no internal state and simply act based on their received percepts and condition-action rules. Model-based reflex agents build upon this with an internal state and a model of how the state of their environment evolves to add information on how to act.

Goal and utility-based agent models 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.

The last agent program model Russell and Norvig present is the learning agent. This combines a critic component, learning element, performance element and problem generator to learn from their actions in order to decide on future actions.

2.2.2 Multi-Valued Fluent Cached Events Calculus

The aim of the event calculus is to create symbolic representations to reason about time periods and local events [35]. 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 [33]. The implementation improves the speed of querying fluents using a caching system.

In Kowalski *et al.*'s [35] 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. Agent systems can be developed using a symbolic approaches and their systems are often logic-based. 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 [8, 33].

The Agent0 belief system reasons about the initiating and terminating of beliefs with certain values through time periods. This belief system accentuates the prevalence of agents requiring methods to reason about time, and it seems natural to use MVFCEC for the representation of a beliefs like system.

2.2.3 Agent Environments, Communication and Interaction

Jennings presented a view of MASs - as illustrated in figure 2.4 - in which agents interact with their environment within their sphere of influence and interact directly with each others. These spheres of influence and interaction are dynamic and often evolve between, but are not limited to, agents in the same organsational relationship.

According to Wooldridge agents often work towards fulfilling design objectives and delegated goals. These objectives and goals are both designed around and constrained by the environment the agent is situated in. Agents may commit to actions within the environment and interact with other agents to achieve these goals. Classical examples of these environments include the blocks world and game-theoretic techniques such as the iterated prisoner's dilemma in which agents interact with each other to boost their payoff. You can find a detailed discussion of environment properties in my appendix in section 7.1.1.

This project is focused upon studying interactions and communication between agents, where actions becomes strategic interactions between agents. Interactions can come in many forms but these are often simplified into communication and the two actions defect and cooperate - in which a donor commits a cooperate or defect action and the recipient and posisbly the donor gets a payoff or a cost from this action.

Cooperate represents an agent committing to an action which is beneficial to the recipient, whereas a defect action represents either a withholding of beneficial resources or an action that directly negatively effects the recipient.

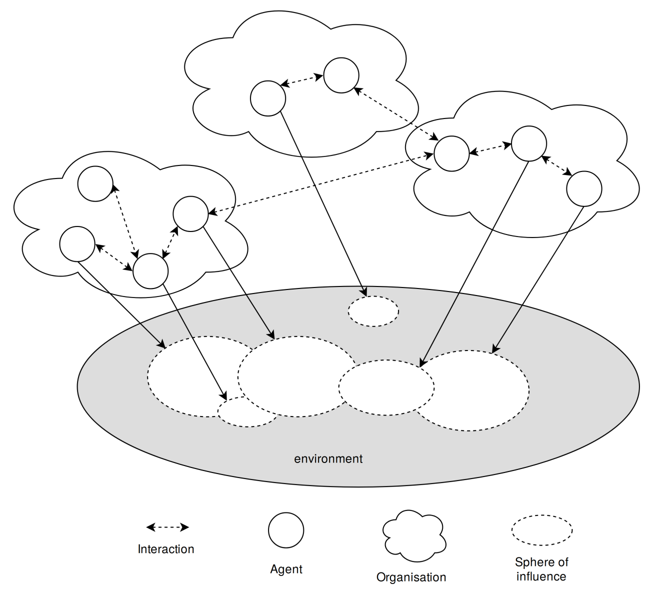


Figure 2.4: Jennings' [32] MAS conceptualisation

Speech act theory defines communication as communicative acts that work towards achieving goals and intentions [9]. This theory is used by ACLs such as KQML [19] and FIPA-ACL. These languages use performative verbs in combination with message content to standardise the communication. The verbs differ between languages, but generally fall into one of the five categories of speech described by Searle [62]. Shared ontologies are used by KQML and FIPA-ACL to define the meaning of commonly used vocabulary for given subject domains to facilitate shared understanding of message content.

Singh [66] criticised ACLs including KQML on their focus on mental agency over social agency. To simplify his argument Singh claims that to focus on communicating about beliefs of an agent is to suppose that agents can read each others' minds.

2.3 Game Theory

2.3.1 Introduction

As discussed we can use game-theoretic techniques to model and study multi-agent interactions. In the following subsections I will explore past studies that model societies of individuals interacting with certain games, and the surrounding concepts that are relevant to this project.

2.3.2 Cooperative Phenomena

Game theory formulates mathematical models to study conflict and cooperation between intelligent rational decision-makers [47]. 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" [69].

Figure 2.5: Cleaning symbioses such as that of the green sea turtle and surgeonfish including the yellow tang show that reciprocal altruism is possible between non-humans and even interspecies [77]

Axelrod and Hamilton [10] 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 7.2.2 I will be focusing on reciprocal altruism.

2.3.3 Reciprocal Altruism

Reciprocal altruism is an idea most famously put forward by Robert L. Trivers [73]. Trivers defines altruism as behaviour of one organism that benefits another to whom it is not closely related, while being 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.

Axelrod and Hamilton [10] 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

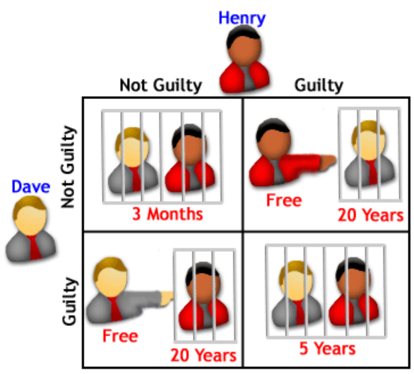


Figure 2.6: A fomulation of the Prisoner's Dilemma payoff matrix [41] similar to Rapoport *et al.* [56]

The prisoner's dilemma was described by Rapoport *et al.* [56] in terms of criminals under interrogation, but Axelrod and Hamilton [10] 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. We can think of these payoffs from an agents perspective in terms of utility [75] - which define what is preferable to an agent

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 [51], which is a type of reciprocal altruism.

		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 [10]). A=x, B=y where x denotes the payoff for A and y denotes the payoff for B.

Axelrod and Hamilton [10] 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 enters into a game against every other player, in which rounds were repeated.

Axelrod and Hamilton [10] combined these round robin tournaments with a genetic algorithm [43] (genetic algorithms are discussed in subsection 2.3.11). One aim of Axelrod and Hamilton's [10] 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 [10] found two strategies with these three abilities: 'tit-for-tat' and 'all defect'. Later on, Nowak and Sigmund [49] 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.

For cooperation to be rational in the iterated prisoner's dilemma the 'shadow of the future' needs to present in each round [75], so games were not cut off at a set point but ended by increasing the probability the game won't continue each round. In other words for the evolution of cooperation to occur in a direct reciprocity system, the cost-to-benefit ratio of the altruistic act must be less than the probability of another encounter for cooperation to evolve: $w > c/b$ [51].

It is possible in MASs that the chance of two agents meeting again is very low, especially across the large networks that agents work across today. This is a key limitation to the application of direct reciprocity to MASs. We require a mechanism to encourage cooperation both when agents are likely and unlikely to meet again.

2.3.5 Indirect Reciprocity

Nowak [51] 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 [7.2.2, 7.2.4, 7.2.3]) and the final one I shall discuss here: indirect reciprocity. This concept is promising for use in MASs as it solves the direct reciprocity's failure with regards to when re-meeting is low.

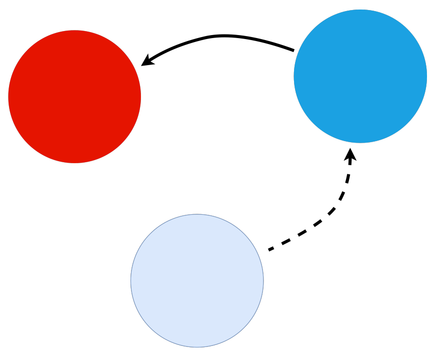


Figure 2.7: The main idea of indirect reciprocity from Nowak [51]

Indirect reciprocity uses the group mechanic of reputation to encourage cooperation. Alexander [6], who was an early advocate the idea, focused on human reciprocal altruism, however, subsequent research has abstracted away from the biology [54, 40, 53, 57, 51, 37, 70, 68, 46]. 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 receive cooperation from others later on. Thus this mechanism a form of reciprocal altruism.

According to Nowak and Sigmund [53] 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.

2.3.6 Nowak and Sigmund

According to Gilbert Roberts [57], Nowak and Sigmund [53] 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.

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

Table 2.3: The payoff for Nowak and Sigmund's [10] indirect reciprocity model

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.

The framework created by Nowak and Sigmund is simplified from Alexander's [6] 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.

As noted above, these actions may 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 2.8.

The concept of onlookers was added due to the realisation of Nowak and Sigmund's that in a group that is spread over a network or geographicaly area in which not all individuals will be able to view each interaction. This sparse nature of interactions is of course very possible 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.

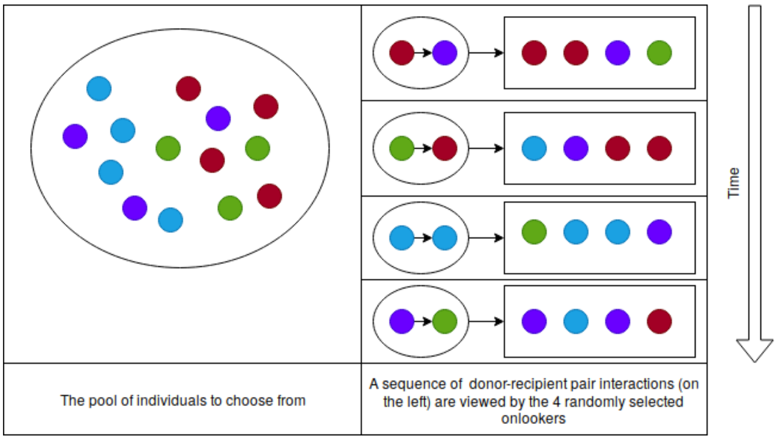


Figure 2.8: The selection of a sequence of donor-recipient pairs and onlookers in Nowak and Sigmund's [53] model of indirect reciprocity. The macro-view of indirect reciprocity. The lines and arrows show the spread of information.

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 2.9. 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 [10] Nowak and Sigmund [53] 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.

Through their experiments Nowak and Sigmund came to two main conclusions. The first is that the higher the value of $2m/n$ - where m is the number of dono-recipient pairs in a game and n is the population size - the more likely it is cooperation will evolve. The second 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 [51] he talks about using 'gossip' as an alternative to direct observation. Social ability is a key property of agents in MASs [76].

The model laid out by Nowak and Sigmund is succinct and a good basis for looking at interactions in MASs. Similarly to Axelrod and Hamilton [10] Nowak and Sigmund [53] 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. I argue that in a MAS agents should own their own understanding of other agents' reputations in their internal state.

2.3.7 The Standing Strategy and Further Limitations of Nowak and Sigmund

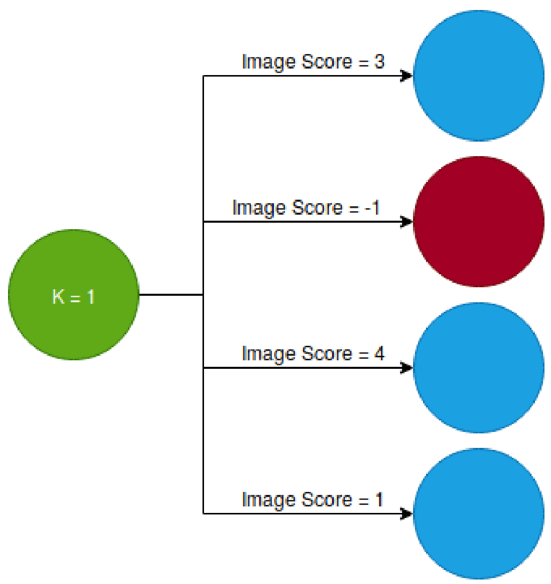


Figure 2.9: On the left an image scoring discriminator with K=1. On the right the agents the discriminator is a donor for. Blue agents represent where there would be a cooperation from the donor, red represents when the discriminator would defect against them. The rule is when $ImageScore >= K$ cooperate else defect.

Sugden [70] developed another game that uses indirect reciprocity and another strategy known as the standing strategy. This strategy aims for both good payoff from it's interactions and also good standing [40]. Each agent has a view of another agents standing which is either bad or good.

From an observers perspective there are 4 possible types of interactions: donor cooperates with good, donor cooperates with bad, donor defects against good and donor defects against bad. The first two both end in the observer giving the donor a good standing, the third ends with the observer giving the donor a bad standing and the fourth doesn't have any effect on the donor's standing.

Leimar and Hammerstein [37] criticised Nowak and Sigmund's [53] limited range of strategies and suggest that their discriminator strategy is far less effective than the standing strategy. The standing strategy does not punish donor's who punish agents with a bad standing.

Conversely, 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 can be 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.

Milinski *et. al* [40] 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 partially observable [61].

Nowak and Sigmund [53] claimed that indirect reciprocity is open to deception and manipulation, especially when we consider the use of gossip in a system. Incorrect 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 [57] and Phelps [54] 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 [53], 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 that for a mechanism to be well developed a number of 'trust models' must be developed to interpret the events an agent sees. These trust models can combine direct and indirect reciprocity to solve each mechanisms limitations.

2.3.9 Gossip

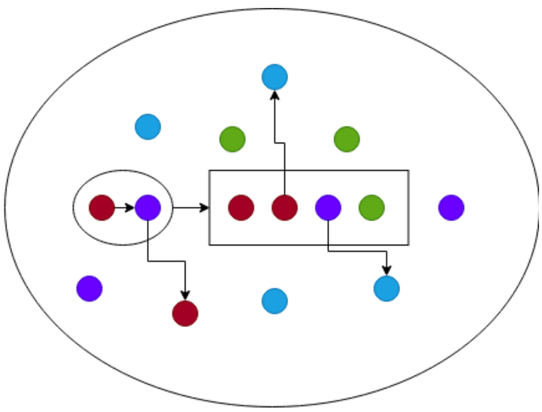


Figure 2.10: The spread of information (shown through the lines) through a population using indirect reciprocity with gossip and onlookers

Nowak [51] suggested the use of gossip to spread reputation information in place of direct observation. Sommerfeld *et al.* [68] 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 concluded that gossip is an effective method of spreading reputation information in an indirect reciprocity system, on fulfilment of some conditions.

The first condition is that gossip must accurately reflect the behaviour of the subject of the gossip. Another condition is that the gossip must be comprehensible by the recipient. The final condition is that individuals must act accordingly based on the gossip they receive.

To use gossip effectively in an agent system; strategies for spreading gossip correctly, trust models for interpretting gossip, strategies for acting upon the gossip must be developed as well as the correct use of an ACL and shared understanding of that data - maybe using an ontology.

2.3.10 Mui's Computational Models of Trust and Reputation

Mui [46] presented an indirect reciprocity simulation framework that used social ability to spread reputation information through 'acquaintance networks' to inform a donor's decision. Acquaintance networks are built up throughout the game by an individual and then consulted when the individual is a donor to inform their decision. The information gathered from the network is known as 'collective memory'.

Mui's acquaintance network supports the spread of positive and negative information about the subject of the gossip. This support fulfils two of Baumeister *et al.*'s [12] four functions of gossip. The other two of which are strengthening bonds between the gossiper and recipient and helping to educate individuals about the complex cultural systems they reside in.

Much like in Nowak and Sigmund's [53] framework donor-recipient pairs are selected at random and individuals in Mui's games use communication reactively not proactively. Wooldridge and Jennings' agent properties [76] and Baumeister's final two properties [12] suggest the need for proactive gossip, possibly using speech act theory.

2.3.11 Reproduction and Genetic Algorithms

Axelrod and Hamilton [10] wanted to find the best strategy out of the set of strategies they had under certain conditions. This is an instance of an optimisation problem - we want to find the best solution out of a set of possible solutions [21].

Axelrod and Hamilton and Nowak and Sigmund [53] both used similar genetic algorithms [43] known as fitness proportionate selection algorithms where the payoffs generated by the interactions are the fitness of the individual. Fitness becomes a performance measure for the agents. These algorithms are inspired by Herbert Spencer's [69] famous phrase "Survival of the fittest", the likelihood of an agent being reproduced into the next game is proportionate to the agents fitness.

I considered two options for reproduction: proportional selection and tournment selection which you can find examples for in my appendix in section 7.2.5. One algorithm for fitness proportionate selection is roulette wheel selection in which agents get a size of a rouletter wheel which is proportionate to their fitness - the higher the fitness the larger the chance that the ball will land in one of that strategy's slots. Lipowski *et al.* [38] presented an efficient version of roulette wheel selection using stochastic acceptance rather than the searching method used by Francq [20].

The stochastic acceptance algorithm works like this: randomly select one of the individuals from the last generations population. w_i is the fitness of that individual and w_{max} the maximal fitness of the generation. With the probability w_i/w_{max} add one of the strategy of that individual to the next generation. Repeat this until the next generation is full. 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 also more efficient.

2.4 Summary

I have discussed and described a number of concepts in the field of multi-agent systems, including multi-agent interactions which are the focus of this project. I have also explored game-theoretic techniques that can be used for modelling and simulating agent interactions. This includes the mechanisms of direct and indirect reciprocity as well as a review of a number of past experiments with these mechanism.

In my review I came to the decision that direct reciprocity is not likely to be adequate for facilitating cooperation in multi-agent systems due to the sparse number of interactions between these agents. I have also remarked as to how many past studies have used programs that do not satisfy the multi-agent system paradigm.

I seek to create a system that does match this paradigm, and uses a mixed reciprocity model in which agents can utilise communicative acts to inform others about agents that they have had no past view one. Further to this, I aim to build agents with a variety of possible strategies that are capable of explaining their actions, using a symbolic AI approach. The multi-agent system will be run within a genetic algorithm to select the optimal agents.

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 [22] 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 [38] (discussed in subsection 2.3.11).

Each player has a fitness score (the system's performance measure) 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 7.3.1.

3.2.3 Sensors

Percepts are the information received at an agent's sensors from it's environment. In my system they are transmitted to each agent in the first stage of 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 Actuators

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 [53] 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 [12] 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.* [68] 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 [61] 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.

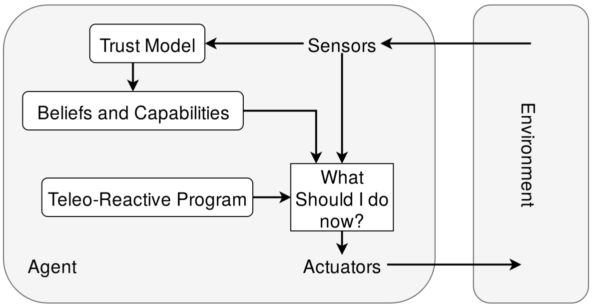


Figure 3.1: The agent architecture in my system.

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 [8]. 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 [22] 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 component 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 [53] 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.

$$
\begin{array}{l}
holds\_at(veritability\_rating(Perceiver, Subject) = V, Timepoint) \wedge \\
holds\_at(percept\_count(Perceiver, Subject) = N, Timepoint) \wedge \\
get\_K(Perceiver, K) \ \wedge V/N \geq K \rightarrow \\
\hfill trustworthy(Perceiver, Subject, Timepoint)
\end{array}
$$

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.

$$
\begin{array}{l}
get\_strategy(Actor, \text{``}VeritabilityDiscerner\text{''}) \wedge \\
interaction\_pair(Actor, Recipient, Timepoint) \wedge \\
trustworthy(Actor, Recipient, Timepoint) \ \rightarrow \\
\hfill action\_commitment(cooperate, Actor, Recipient, Timepoint)
\end{array}
$$

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:

$$
\begin{array}{l}
get\_strategy\_component(Actor, \text{``}Spread\ Positive\ Trusted\text{''}) \wedge \\
not\ interaction\_pair(Actor, \_, Timepoint) \wedge \\
findall(Trusted, trustworthy(Actor, Trusted, Timepoint), TrustedAgents) \wedge
\end{array}
$$

$len(TrustedAgents, Len) \ \wedge Len >= 2 \ \wedge$
$get\_2\_random(TrustedAgents, TrustedAgent, Recipient) \rightarrow$

$$action\_commitment(gossip(\{recipient : Recipient, about : TrustedAgent, gossiper : Actor, timepoint : Timepoint, gossip : positive\}), Actor, Recipient, Timepoint)$$

$get\_strategy\_component(Actor, \text{``}Spread\ Positive\ Trusted\text{''}) \ \rightarrow$

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

'Spread Negative Untrusted' formalisation:

$get\_strategy\_component(Actor, \text{``}Spread\ Negative\ Untrusted\text{''}) \ \wedge$
$not\ interaction\_pair(Actor, , Timepoint) \ \wedge$
$findall(Trusted, trustworthy(Actor, Trusted, Timepoint), TrustedAgents) \ \wedge$
$len(TrustedAgents, Len) \ \wedge Len >= 2 \ \wedge$
$get\_random(TrustedAgents, Recipient) \rightarrow$

$$action\_commitment(gossip(\{recipient : Recipient, about : TrustedAgent, gossiper : Actor, timepoint : Timepoint, gossip : positive\}), Actor, Recipient, Timepoint)$$

$get\_strategy\_component(Actor, \text{``}Spread\ Negative\ Untrusted\text{''}) \ \rightarrow$

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

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 development of that system and the software engineering techniques, tools and processes I have employed for this development. Descriptions for running the software are given in my appendix in section 7.4.

3.3.2 Development

Architecture

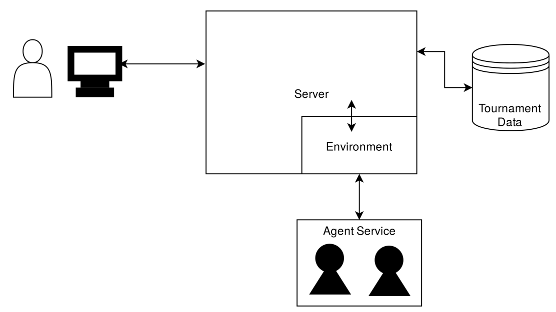


Figure 3.2: System architecture

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 provides the client side interface for users across the internet. This interface allows users 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 to facilitate sending percepts and getting decisions.

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 an ORM (SQLAlchemy) I have been able to create as both a SQLite database for development and a MySql database for deployment. An illustration of the overall architecture 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 being often overlooked for the development

Figure 3.3: Two screenshots of two pages of the Prolog web service proof of concept

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 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 in the backend using the Axelrod-Python library [18]. This early prototype also included a database set up, a redis queue to run the tournaments, 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 been a project within itself. However, Prolog has a good HTTP support and as such I saw an opportunity to further hone my web skills by developing a web service to host agent minds.

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 used skills learnt from this proof of concept, but was developed separately).

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.

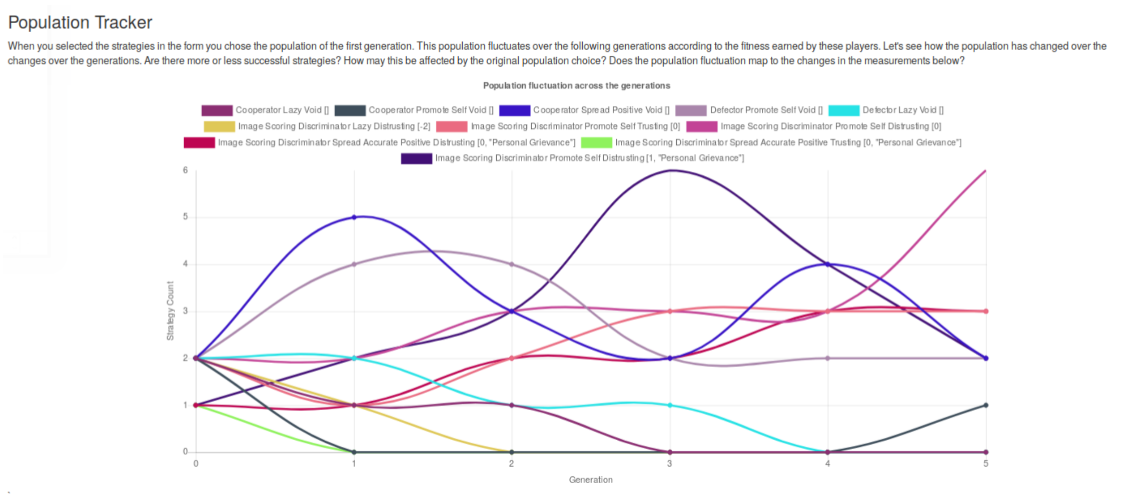


Figure 3.5: Tracking the population of strategies through each generation.

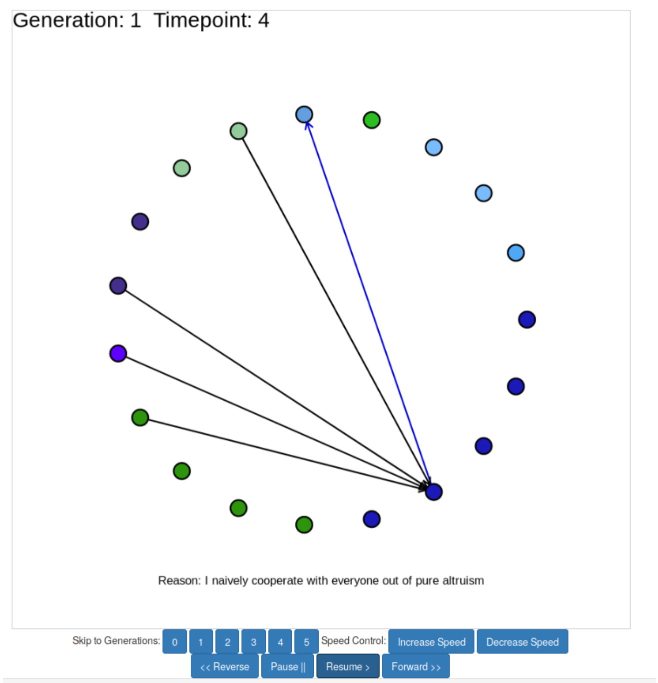


Figure 3.4: The animation of a game.

The main bulk of the development was implementing 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 (part of this is illustrated in figure 3.5). A visualisation of the game that has been run is also included in an animation on the analysis page (figure 3.4).

In addition to this I have created a feature that allows users to create accounts, login and associate 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.

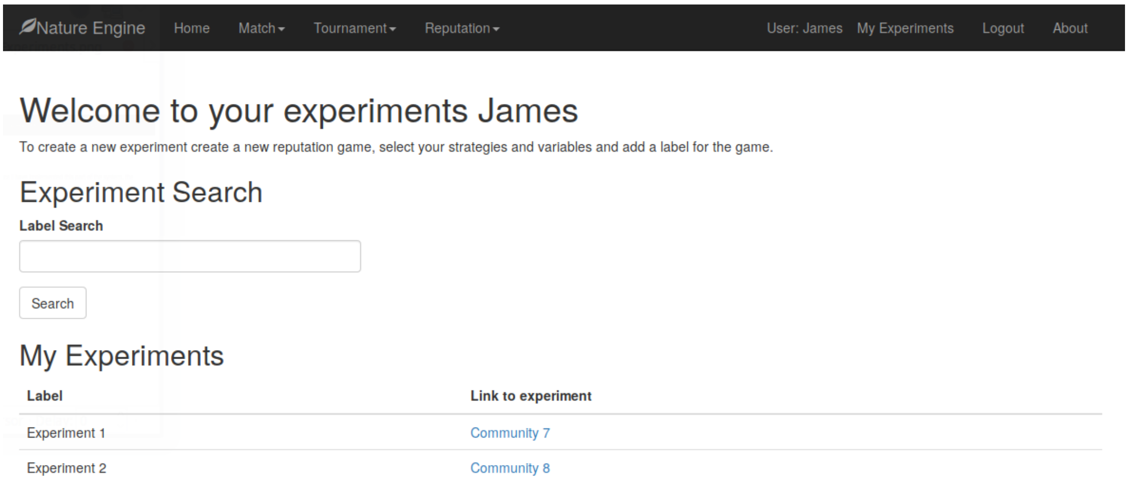


Figure 3.6: The user experiment tracker

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 system provided by my supervisor Kostas Stathis and originally developed for [33]s. 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

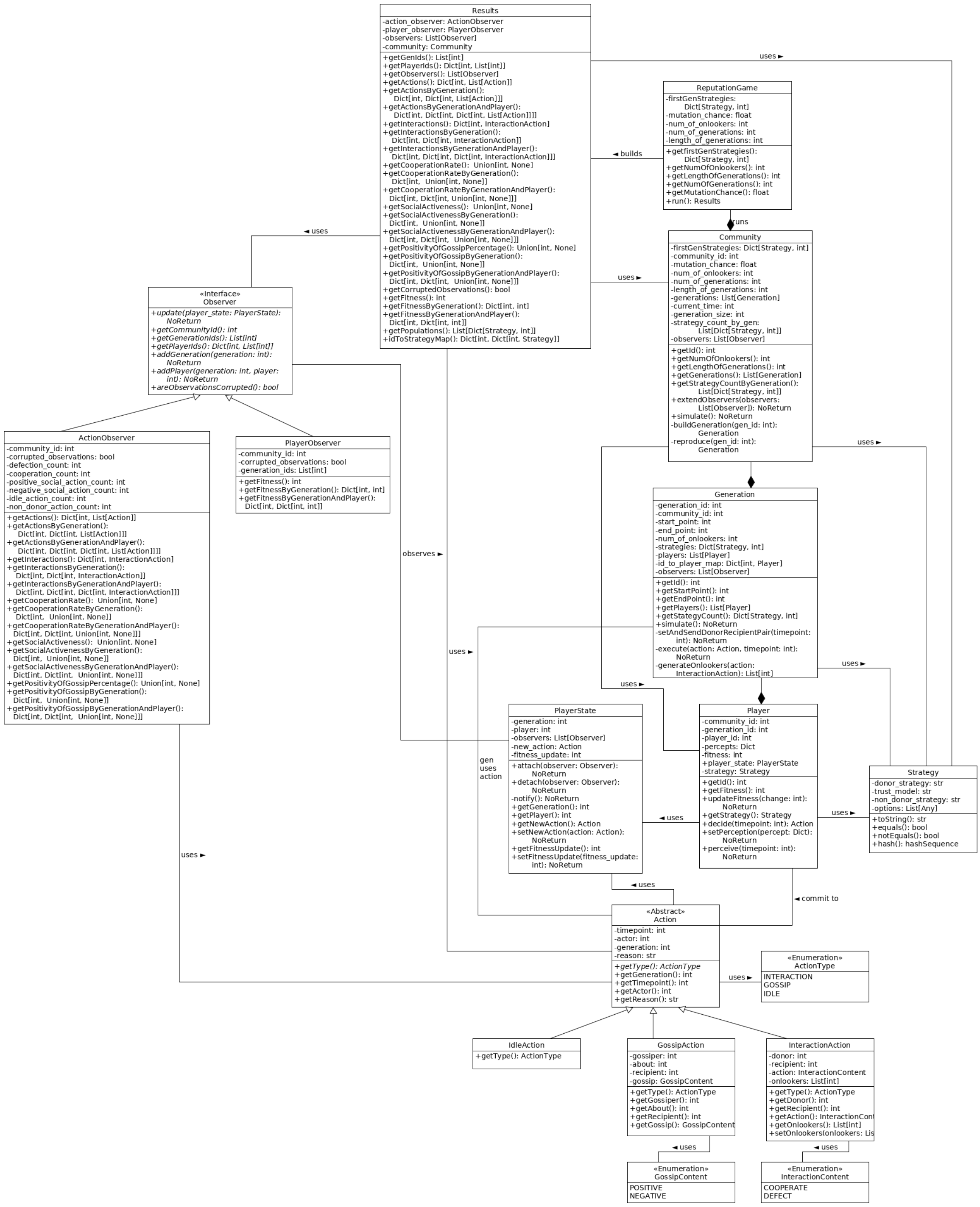


Figure 3.7: The class diagram for my environment and agent bodies Python program, also can be found in the ProjectReports/NewFinalReport directory as EnvClass.png

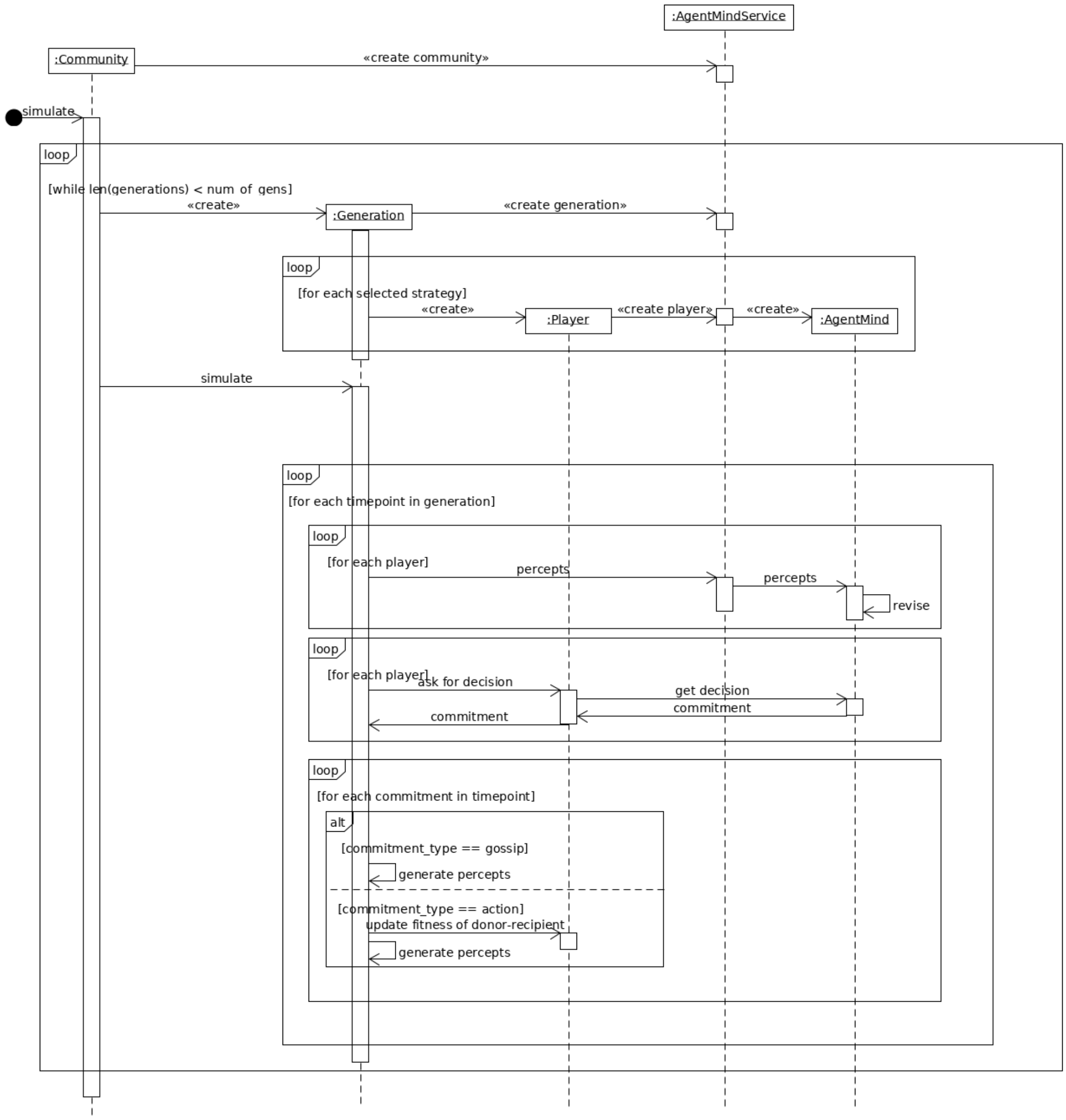


Figure 3.8: The sequence diagram for my environment and agent bodies Python program, also can be found in the ProjectReports/NewFinalReport directory as EnvSequence.png

for Linux, however I occasionally used the git support in the PyCharm integrated project support environment.

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.

Chapter 4: **Experiment Evaluation**

4.1 Introduction

In the following chapter I will be running experiments with the implementation of my theoretical framework to review the effectiveness of the mechanisms and concepts I have used as to facilitating the evolution of cooperation. I will also be discussing and evaluating the effectiveness of the strategies available to associate agents with. These experiments will use the three experiment ideas put forward by Axelrod and Hamilton: robustness, initial viability and stability.

4.2 Robustness

The experiments I have run are similar to Axelrod and Hamilton's [10] experiments on the robustness of strategies in the system. However, there are 199 strategies available to use in my system, if I included one of each of these in the experiments it would take far too long to execute them.

As such, I selected a population using a subset of these strategies for the experiments that remains the same throughout the robustness experiments (listed in appendix section 7.6.1). This population was extremely varied in terms of how cooperative each agent is and the strategies and trust models used by the agents. I would have liked to experiment with a number of different varied populations, however due to time constratints this was not possible.

9 unique experiments were conducted with this population. I would have liked to duplicated all the experiments (only those of generation length 60 and 200 were duplicated). However, again due to time constraints (the experiments take hours to run with long generation lengths) and the overwhelmingly similar results between all experiments I did not deem this necessary nor beneficial. These experiments can be divided into sets of 3.

All experiments were run for 10 generations, and in the sets of 3 one experiment ran with 5 onlookers per interaction, the next with 10 and the final with 20. The first set of 3 experiments was run with a generation length of 60, the next set 200 and the final set 400.

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

4.2.1 Evaluation: Evolution of Cooperation

The overwhelming outcome of the experiments is that cooperation does not evolve under the conditions of the experiments. By roughly halfway through the generations for all but 2 of the experiments cooperating agents were completely wiped out and the cooperation rate had fallen to zero. All the strategies that were left were either Defectors or Veritability Discerners and Image Scoring Discriminators with high K values that consistently defected.

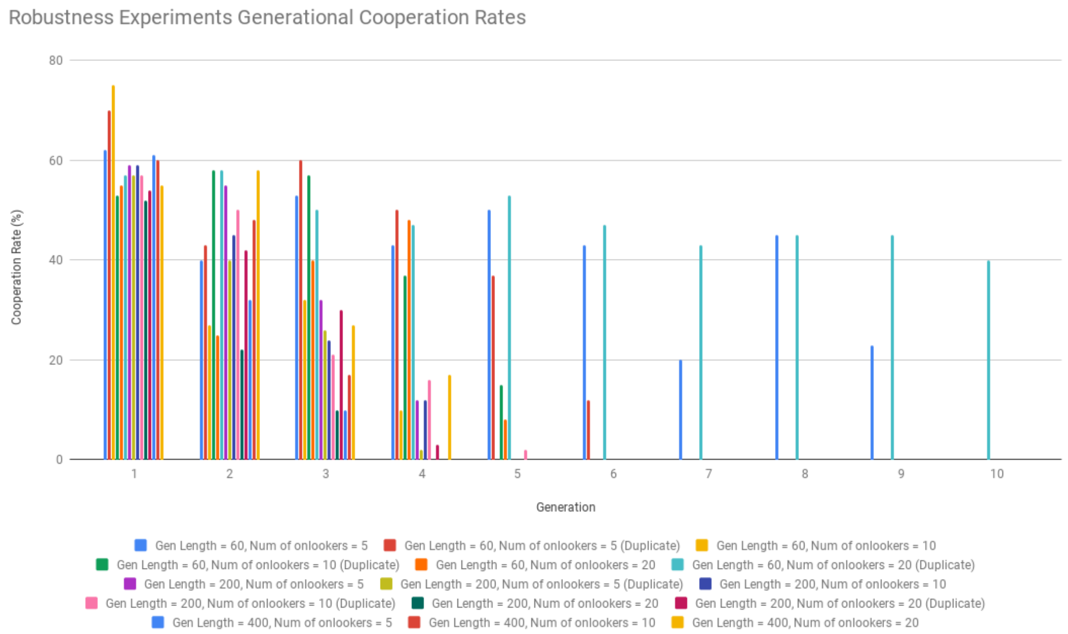


Figure 4.1: A chart mapping the cooperation rate of the ten generations of each experiment, look to the legend for details on what the variables were set to for each experiment.

Nowak and Sigmund [53] discovered that for their game cooperation could occur under certain conditions. These conditions included an increased generation length. This variable seemed to have little if no effect on the evolution of cooperation in my system. With the increase in length of generations, cooperation did not survive any longer and did not increase in earlier generations. Though if I had time I would have liked to try for again longer generations to see if I simply had not found the shortest length of generation at which cooperation evolves.

For their model Nowak and Sigmund [53] also found that for cooperation to evolve it is extremely important for agents to know the image score of other agents. I would surmise that my agents also need to be well informed as to the reputation of others. Following this idea, increasing the number of onlookers per interaction, would logically increase cooperation. However this is not backed up by the data I have received from my experiments. In fact at some points the data seems contradictory, with the number of onlookers increasing and the cooperation rate decreasing.

The only 2 exceptional experiments in which cooperation managed to survive until the later generations - one of which cooperation remained present until the end - were of generation length 60. The more cooperative of these experiments had 20 onlookers per interactions, and the less cooperative had five.

It is very possible these two experiments were simply lucky chance, especially as both had duplicates that were notably unsuccessful. Maybe cooperative agents were consistently lucky with reproduction, or it just happened to be that donors and recipients were chosen far more often as cooperative agents. Having said these I would have liked to have time to repeat the experiments many more times to either confirm or deny this speculation.

4.2.2 Evaluation: Strategies

Due to the lack of stable cooperation in the experiments overall there were not many stand out strategies. From the most cooperative experiment one cooperative strategy remained to the end in large numbers. This was a Standing Strategy with a non-donor strategy of Spread Accurate Negative and a trust model of Trusting. This strategy may have been able to survive so long as the agents used it warn each other of the defectors in the population.

In the second most cooperative experiment the second to last generation still remained partially cooperative. In this generation one Image Scoring Discriminator using the Lazy non-donor strategy, Trusting trust model and with K=0 survived and four Veritability Discerner's using the Spread Positive Trusted non-donor strategy, Balanced Reactor trust model and with K=-5 survived.

4.2.3 Evaluation: Trust Models and Gossip

As I have noted above increasing the number of onlookers per interaction in a game does not seem to increase the cooperation rate of the agents in that game. This could be due to a number of factors, but the data points to a possibly overall negative effect of gossip on the evolution of cooperation. Sommerfeld *et al.* reported that gossip had to both accurately describe the behaviour of the subject and be acted upon appropriately to help facilitate cooperation.

I believe there are three possible explanations (that may overlap) as to how gossip has not helped facilitate cooperation in these experiments. The first is the use of gossip to deceive and manipulate other agents. This feature was actively designed into some of the agents in the system to study how misinformation may affect games. There is a distinct possibility that this has had the effect of reinfocing defection in these experiments.

Secondly there is a possibility that gossip that did not intend to deceive others did not accurately represent the actions of the subject of the gossip. For example, Discriminators and Discerners often base their communicative actions on reputation information that can be formed by gossip from others. If the original gossip is innaccurate, the effect may be to perpetuate the innaccurate information.

Lastly gossip may have not been acted on correctly. The trust models that I have developed may not adequately interpret the received gossip to revise beliefs about other agents. As such, the agents may not be able to use these formed beliefs to correctly act towards the subject of the gossip.

4.3 Stability

Axelrod and Hamilton [10] also ran another set of experiments to test what strategies could prevent defection from taking hold in societies that are predominantly cooperative. Again I had planned to take successful strategies from my robustness experiments and use them in this. However, of course I did not find many successful strategies.

As such I designed and ran experiments similar to Axelrod and Hamilton's stability experiments with 9 selected strategies. I selected 9 strategies (including the three most successful cooperative strategies from my robustness experiments) with a variety of strategies and trust models (listed below) that I believe could be successful in upholding cooperation against a group of 6 defectors.

In this group of defectors are 2 for each of the non-donor strategies: Lazy, Promote Self and Spread Negative. For each of the 10 selected strategies I ran 6 experiments. Where I matched up 25 of the selected strategy against the group of defectors. These experiments follow the same layout as the initial viability experiments except with the populations I have described here.

To see an in depth analysis into how well each strategy did in the experiments go to the appendix in section 7.6.2.

4.3.1 Strategies

1. Standing Discriminator, Spread Accurate Negative, Trusting
2. Standing Discriminator, Spread Accurate Positive, Distrusting
3. Standing Discriminator, Promote Self, Naive Trusting
4. Image Scoring Discriminator, Spread Accurate Negative, Trusting, K=-2, Personal Grievance
5. Image Scoring Discriminator, Lazy, Trusting, K=0
6. Image Scoring Discriminator, Spread Accurate Positive, Distrusting, K=0, Personal Grievance
7. Veritability Discerner, Spread Positive Trusted, Balanced Reactor, K=-5
8. Veritability Discerner, Spread Negative Untrusted, Strong Reactor, K=0
9. Veritability Discerner, Promote Self, Forgiving Reactor, K=0

4.3.2 Evaluation: The Stability of Cooperation

The first 3 strategies selected for this experiment failed to uphold cooperation in the societies they were apart of. The first generation of each experiment started with a cooperation of around 80% but this usually dropped of generation by generation until roughly the 6th or 7th generation in which it fell to 0%. This pattern held except for one experiment which appears to be an outlier.

4.3.3 Evaluation: Trust Models and Gossip

4.4 Initial Viability

My plan had originally been to review the outcomes of my robustness experiments and pick ten strategies which had been successful for both the stability and initial viability experiments. However, my robustness experiments revealed a very limited amount of cooperative strategies capable in surviving in a varied environment.

Furthermore, my stability experiments proved that even the successful strategies could not protect from invasion by a small amount of non-cooperative agents. Due to this, I have decided that there is no need to run initial viability experiments, as these experiments would befar more difficult for the strategies than the stability experiments.

4.5 Discussion

Chapter 5: **Critical Analysis, Discussion and Conclusion**

5.1 Introduction

On reflection I can divide this project into 4 phases. The first of which was discovery, in which I researched and developed ideas around game theory, multi-agent systems and web development. The next phase was theoretical foundation in which I focused on the specific theoretical model, architecture and design for my software.

The third phase was implementation. This phase focused on creating the application and testing. The final phase is consolidation. Consolidation focused on polishing a lot of the application, adding features such as the experiment tracker and the “Veritability Discerner” strategy. These phases by no means had solid boundaries between each other, but represent my view on the high level project process.

In the following sections I will go through these phases giving critical analysis and discussion on them. I will then discuss my final achievements, project process and future enhancements I wish to make.

5.2 Discovery

At the beginning of this phase of the project, I had little experience of web development and next to no knowledge of either game theory or multi-agent systems. I tackled these issues by partaking in web development courses and tutorials [1, 2], researching using academic papers on game theoretic mechanisms and learning around multi-agent systems.

I was successful in my learning of web development. I believe this was due in part to the way I tackled the problem; by deciding early on the technology I wished to use based on the recommendations of experts in the Python community and my own knowledge on security and web development.

Early on I also developed an in-depth knowledge of game theoretical concepts including reciprocal altruism and it’s variants (indirect, direct, network, etc.) and kinship theory. I also believe this was in part to the approach I took to learning it. By following the specification in terms of papers to read and reaching further into the literature I uncovered a number of ideas and mechanisms put forward in the area to compare.

On reflection, I focused on learning web development and game theory. This unfortunately meant I did not give enough time to learning about multi-agent systems. I believe this was simply due to me not recognising it’s importance in the project originally as I had a lack of understanding of the topic of multi-agent systems. This oversight is, however, a lesson to learn for any projects I undertake in future.

If I were able to guide myself with what I know now, I would tell myself to look at a wide overview of all the topics and concepts the project encompasses. Multi-agent systems is the basis of the project which only became clear to me whilst I was deigning the theoretical framework. Though I think this attitude - to look at a wide overview of the project’s topics -

is important not only to this project to any project, and will be an attitude I take to projects in the future.

In fact, web development isn't even necessary for the project in it's purest form. However, I am pleased I undertook the web development side as I have learnt a lot and believe this level of web development is important for someone who has studied computer science to know and will hopefully benefit my career.

5.3 Theoretical Foundation

It is with this new found knowledge on game theory and not enough multi-agent system knowledge that I went on to develop the theoretical framework that my implementation uses. I combined aspects of both fields for this phase and am very pleased with the mixed reciprocity model I have designed. This model draws from many past studies on reciprocal altruism mechanisms and attempts to fuse the ideas with multi-agent systems.

Though so far the system has failed to support cooperation between agents, I believe this highlights the difficulty of designing strategies in partially observable and multi-agent environments, rather than being an issue with the framework. This does not mean to say that the strategies are useless.

I am happy with my design for these strategies as they build upon past studies and convert them into the AOP paradigm. I am also pleased with my work on strategies as I have designed and implemented a new strategy to add to these.

Using the powers of hindsight, I would have liked to have learnt more about MASs and especially agent architecture design before creating the theoretical foundation. I believe that my agent architecture is rather loose and could have been far more refined. If I had the knowledge that I do now, I could have defined a more succinct and informed architecture.

5.4 Implementation

I was very successful in my learning of web development, and have implemented a web application and a web service with a number of technologies and techniques that were new to me. The web application does have it's shortcomings in terms of performance. It turns out that simulating multi-agent systems with genetic algorithms in the backend is particularly CPU and RAM intensive.

This certainly doesn't scale well for long games or many users and would be very open to denial of service attacks in it's current form. However, the application is not part of a critical system. If it were important to scale up for performance, it would be quite easy to do so due to my use of a redis server. I could have another set of servers as a private network running redis workers and send the data to run games to those. These could be easily scaled up to meet demand. However, this is expensive and the cost to benefit ratio is unlikely to be in favour of doing it.

In terms of the genetic algorithm and MAS I have implemented, I am happy with it's ability to run without fault so far - I have not experienced a single fault in my experiments that I have detected. I am also happy with the way in which I have written the environment and agent code. I believe the class structure and architecture to be logical and the agent program to be as well.

However, in retrospect if I had been starting this project over again I would have made some improvements. Firstly I would not have written the genetic algorithm, environment and agent's bodies in Python. The game is extremely CPU intensive and would be in any language. However, this could be improved by using a language capable of concurrency and one that is generally known to be faster on certain benchmarks.

I would suggest either the use of Rust or Java for this. Rust is a language comparable in speed to C and would have allowed me to create a program that run in far less time with far less resource usage than Python. Rust would also allow me to parrallelise sending percepts to agents and getting decisions from them, dramatically reducing the time it takes to get these. However, Rust is not commonly used in MAS development and is not an OOP lanaguage. Thus Rust does not match the paradigm of AOP as well as Java and does not have the tools for agent development (such as Jade) that Java does.

I am pleased with the achievement I have made in creating a usable front end for users of my web application. I find the interface relatively intuitive and I am able to create and run games without issue and view a well presented analysis and simulation of the game. There are definitely a few areas that could be improved, and the application should ideally be usability tested to uncover these.

One issue I have with the interface I have created is that if you are wanting to run many experiments and are time constrained, it is a cumbersome having to select a poulation of agents and set the parameters, then wait for the experiments to run and repeat. To solve this problem I have simply written scripts that run games on a server that is not the web application's server.

The creation of scripts is acceptable for me, but I believe I could have split the code base up better by creating a library to run the games separately to the web application. This library could then be imported for use in the web application and other applications. I could also have provided functionality in this library by automating the running of a set of experiments using a markup language or json file to specify the population and experiment parameters.

After running the match the results could be written to a file and graphs produced using a library like matplotlib. Further to this, I could have implemented functionality in the web application for automating a set of experiments for a registered user and associating this set with their account to access them later.

Lastly, I am pleased with my development of a simple web service in Prolog. This service is powerful in terms of hosting agents minds, allowing for logical decision making and belief formulation, and managing those agent minds in terms of communities and societies.

I believe using logic predicates to define agent decision making components was the correct choice as it more accurately reflects a primarily logical process taking place. In comparison procedural programming does not as naturally represent and reason with data in a symbolic manner that is appropriate for this kind of decision making process.

I also believe that creating a web service in Prolog was the right choice as I have demonstrated the ability to develop autonomous agents in a distributed setting. A setting which I believe to be key in the development of MASs.

5.5 Consolidation

In this phase I polished the application and added features that I believe have enhanced the application beyond my original specification. My original project plan did not include web application facilities for tracking experiments and I did not plan to invent any new strategies.

I also did not specify that I would deploy the web application. Deploying has taught me about the practicalities of running web applications - with which I have learnt that my early learning and use of redis queues and the SQLAlchemy ORM was good planning, as these sorts of technologies would allow me to easily scale up my application.

5.6 Project Process

I believe that the 4 phases of my project match the timeline I created for my project proposal. I think this high level process has been effective in solving the problem of how to run a successful project and followed a natural progression.

Further to this I believe my scrum like process of using a task backlog and selecting a group of sprint tasks in two week sprints has allowed me to be responsive to change within the project, while also prioritising aspects of the project that are key to it's success. Though at times I could have been more effective in managing the task backlog, I believe the end product and project itself both reflect the success of this process.

I am likely to employ similar processes again for projects, whether it be in a team or as an individual I believe this has been good practice for the management of a project. Further to this I have affirmed my belief that using the right tools for managing a project is key to the process of that project.

Trello was the tool I originally used, however Trello is such a malleable tool that it is not focused on the workflow that I was using and was not integrated into my work environment. When thinking about my workflow and process I want a tool that fits to both and integrates easily into my work environment. When starting a new project I will be more careful in considering which tools I use based on this criteria to improve my process management.

Finally a lot of my process has come from my learning of software engineering in OOP. Although this is good, there are multiple methodologies available for the development of a multi-agent system. Including methodologies such as Tropos which extend OOP with methodologies. I believe looking back on the process that is could have been beneficial to use Tropos and the Agent Unified Modelling Language (AUML) alongside OOP techniques.

5.7 Future Enhancements

I have already talked about a number of enhancements that can be applied to this project. These enhancements include; adding the ability to easily scale up the application to use servers to run the redis queues, improving the speed of running the game, splitting the game code into a separate library and the automation of multiple experiments.

There are also improvements that can be made in terms of the strategies in the system. I have not found any successful strategies that support cooperation, but maybe I have simply not found this model's tit-for-tat. Developing new trust models and strategies for when the agent is not a donor for existing strategies could improve the chances of finding a successful strategy.

It would also be interesting to see the effectiveness of reinforcement learning - or deep symbolic reinforcement learning when that becomes practical - for use as agent decision making components in relation to their ability to survive against non-cooperative agents and support cooperation within the community. It has already been proved that reinforcement learning techniques lead to dominant strategies in other models [29]. A future enhancement could be to add a strategy which is able to use a technique such as reinforcement leaning.

Another enhancement I would like to add to the project is a whole new game for the agents to participate in. This new game could be inspired by another game-theroretic mechanism such as network reciprocity and the ideas I have laid out in my appendix about network reciprocity (section 7.2.3).

5.8 Conclusion

In conclusion I believe I have delivered an end product comprised of a number of technically complex but stable and functionally developed components. I believe these components were well researched with academic and software engineering considerations.

I believe this project has facilitated me to develop skills in managing and running a sizeable project. Further to this the project has helped me practice my software engineering skills and learn skills in multi-agent systems development. Although there are a number of aspects that could have been improved I believe I am aware of most of these, and have developed my skills in critical self-analysis.

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] David Abel, James MacGlashan, and Michael L Littman. Reinforcement learning as a framework for ethical decision making. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

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

[7] Stuart Armstrong. Motivated value selection for artificial agents. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

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

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

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

[11] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

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

[13] Nick Bostrom. Ethical issues in advanced artificial intelligence. *Science Fiction and Philosophy: From Time Travel to Superintelligence*, pages 277–284, 2003.

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

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

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

[17] Richard Dawkins. *The Selfish Gene*. Oxford University Press, 2016.

[18] 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.

[19] 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.

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

[21] Pascal Francq. Optimisation problems. `http://www.otlet-institute.org/wikics/Optimization_Problems.html`, 2011. Accessed: 04/01/2019.

[22] 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.

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

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

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

[26] Irving John Good. Speculations concerning the first ultraintelligent machine. In *Advances in computers*, volume 6, pages 31–88. Elsevier, 1966.

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

[28] W. D. Hamilton. The genetical evolution of social behaviour. *Journal of Theoretical Biology*, 7:1–16, July 1964.

[29] 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.

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

[31] 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.

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

[33] 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.

[34] Atef Khedher, Kamel Benothman, Didier Maquin, and Mohamed Benrejeb. State and sensor faults estimation via a proportional integral observer. In *2009 6th International Multi-Conference on Systems, Signals and Devices*, pages 1–6. IEEE, 2009.

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

[36] 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.

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

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

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

[40] 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.

[41] 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.

[42] John Stuart Mill and Jeremy Bentham. *Utilitarianism and other essays.* Penguin UK, 1987.

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

[44] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[45] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

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

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

[49] 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.

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

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

[52] Martin A. Nowak and Robert M. May. Evolutionary games and spatial chaos. *Nature*, 359:826–829, 1992.

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

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

[55] Karl Raimund Popper. *The open society and its enemies: The spell of Plato*, volume 1. Princeton University Press, 1971.

[56] 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.

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

[58] 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.

[59] Casey Ross. Ibm pitched its watson supercomputer as a revolution in cancer care. its nowhere close. `https://www.statnews.com/2017/09/05/watson-ibm-cancer/`, 2017. Accessed: 25/03/2019.

[60] Stuart Russell and Peter Norvig. Artificial intelligence a modern approach 3rd edition pdf. *Hong Kong: Pearson Education Asia*, 2011.

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

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

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

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

[65] 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.

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

[67] Aaron Smith and Janna Anderson. Ai, robotics, and the future of jobs. *Pew Research Center*, 6, 2014.

[68] 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.

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

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

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

[72] 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.

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

[74] Vernor Vinge. The coming technological singularity: How to survive in the post-human era. *Science Fiction Criticism: An Anthology of Essential Writings*, pages 352–363, 1993.

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

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

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

[78] Eliezer Yudkowsky. Creating friendly ai 1.0: The analysis and design of benevolent goal architectures. *The Singularity Institute, San Francisco, USA*, 2001.

Chapter 6: **Professional Issues**

Autonomous Agents and AI

The field of multi-agent systems incorporates many aspects and concepts from AI. AI is a keenly debated ethical topic. From discussions on the singularity in which humans are overtaken in intelligence [13] to a deep seated concern of job loss [67], those working on AI developments must be acutely aware of the risks and issues surrounding the field.

The field of AI is, at the moment moving, in an incremental fashion, developing specialised technologies which are capable of doing certain tasks better than humans and then combining these into then more intelligent applications. Nick Bostrom [13] believes that there is the potential for these technologies to eventually cohere into the creation of a superintelligence. In fact, he identifies that superintelligent autonomous agents may be the eventual end product of these advances.

One key concern is; as we move closer to these higher level artificially intelligent agents, how will we enable these agents to act ethically? There is widespread concern that the development of higher level AI will lead to damaging consequences for lives across the world. Some have even suggested that if we cannot either avoid the singularity, develop upon the human form or find a way to teach AI an ethical approach then AI may lead to the downfall of humanity [74].

On a very small scale, a part of my project is attempting to develop agent strategies that can discern between those who deserve cooperation and those that do not. If there were stakes to this game, this would be a distinctly ethical dilemma, with agents attempting to judge others based on the morality of their actions.

AIs are making ethical decisions already [59] and it is likely that AIs in multi-agent systems will have to make ethical decisions in regards to each other and biological lifeforms in the future as well. It is the technologies and methods that AI developers and researchers create today that will directly influence the quality and safety of the AI of tomorrow.

Moral philosophy and ethics have been studied since early civilisation, and ethical decisions have been made and questioned even before then. So what are the current approaches on how to ensure AI makes ethical decisions? What do people believe are the best approaches?

Seed AI

Many believe that as superintelligence approaches, the most important way to teach AI ethics is to ensure that the 'Seed AI' is benevolent [78]. The idea is as follows; Once superintelligence has been created, this superintelligence will recursively spawn further intelligence and an explosion of intelligent systems will happen very quickly [26]. The original superintelligence that begins this process is known as the Seed AI, which is capable of self-understanding, self-modification and recursive self-improvement. If this belief is correct the question now becomes how do we avoid creating malevolent Seed AI?

Explainable AI

One of the steps we can take towards ensuring the benevolence of AI is the development of techniques for explainable AI [27]. Explainability is the notion of a human being able to understand the reasoning behind a decision or recommendation made by an AI.

If humans are able to understand why an AI is making the decisions it is making we will be able to detect early if an AI has become malevolent, and perhaps even be able to glean information as to why it has. Within the field of autonomous agent development, deep reinforcement learning has been identified as a very promising piece of technology [45, 44].

However, deep reinforcement lacks explainability, so it is not useful in domains that require verifiability [24]. As such, it is my view that researchers and developers in the AI space should work towards unifying these techniques as soon as possible with explainable models - possibly through symbolic AI techniques.

This is one of the key reasons for using Prolog and a symbolic AI approach in developing my agent strategies. As the agents are capable of explaining their actions and the causes behind them, we are able to understand more as to why a certain chain of events played out as it did.

State Estimation

Russell and Norvig [61] described 3 sources of risk which may cause AI to either malfunction and cause damage, or to malevolently cause damage. The first of which is incorrectly estimating the state of the world. State estimation malfunctions could cause actions to be based on faulty data.

With AI making it's way into more and more safety critical systems, this could have disasterous effects. Khedher *et al.* [34] studied a method for estimating sensor faults using a mathematical transformation. Steps like Khedher *et al.*'s are integral to the development of safe AI systems, and developers of AI systems should take a keen interest in techniques such as these.

Utility Functions

The second source of risk presented by Russell and Norvig [61] is the design of AI utility functions. Many AI decision making systems are based on trying to maximise a specific utility. This utility is often designed by the creator of that AI system to described a system's preferred states.

But when it comes to agents that are situated in complex environments, especially one such as our world, a number of seemingly simple but in reality tough questions arise in utility function design. Firstly, what are the preferred states? The preferred state of many games is simple, but when making ethical decisions this is hotly debated.

Proponents of 'Positive Utilitarianism' [42] may argue that the correct action is the one that maximises the well-being or happiness. Whilst 'Negative Utilitarianism' [55] principles would put forward that we should seek to minimise suffering or unhappiness.

Both seem like reasonable proposals. However, even if we were to settle on one of the many thousands of acceptable ethical models, the terminology often used in them is ambiguous and loosely defined. For example, those using Negative Utilitarianism wish to minimise suffering. How can we define suffering to an AI when it is such a subjective concept? Beyond that, how can we specify degrees of suffering?

Russell and Norvig [61] point out that if an AI misunderstands the concept, it could come to the conclusion that all humans are suffering. This could lead to a decision that mass murder will maximise it's utility by minimising suffering.

Learning

Some have suggested that, as it is difficult to define and hardcode specific preferences for a utility function, agents should learn through observation [7, 5]. Learning from observation leads to it's own dilemmas, and is in fact a part of Russell and Norvig's third source of risk: learning functions causing AI systems to evolve unintended behaviour.

If AI systems do use learning functions to learn utility functions on what actions are considered 'ethical' there is a risk that they will develop behaviours considered 'unethical' to us. Further to this, if they act on these in safety critical systems they could cause great damage.

Conclusion

AI systems are incrementally developing greater and greater capability. It is very likely in the future we will develop systems that achieve high levels of intelligence. We need to consider our design decisions to ensure that we develop a platform from which to develop AI systems capable of ethical action.

A few of the steps we can take to work towards this are the development of explainable AI systems (possibly by fusing current learning techniques with symbolic AI approaches), the careful consideration of how to develop ethical utility functions and imposing checks that prevent AI systems from using learned behaviours that are unintended or unethical.

Chapter 7: **Appendix**

Describe the contents of my appendix.

7.1 Multi-Agent System Background

7.1.1 Agent Environment Properties

According to Russell and Norvig [61] to specify a task environment one needs to specify four aspects of the system: performance measure, environment, actuators and sensors. In the 2013 version of the textbook 'Artificial Intelligence: A Modern Approach' [60] Russell and Norvig described a generic program for running an environment.

In this program a set of steps are 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 [61] seven 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 observability. A fully observable/accessible environment is one that gives an agent full access to the state of that environment. In a fully observable environment there is no need for an agent to keep any internal state about the environment. Unlike in a partially observable environment, 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 [61] is whether the environment is episodic or sequential. An episodic environment is split into episodes where the quality of actions in each episode do not depend on previous episodes. Whereas, in a sequential environment an action taken at timepoint t may be affected by actions from previous episodes and the action at timepoint t may affect all future actions.

Another property is whether the system is multi-agent or single-agent. This seems simple, however, it massively depends on what we see as an agent. For example take a surveillance system, with a central agent collating the data and a number of connected pieces of surveillance systems (for example CCTV cameras). There are blurred lines as to whether these connected components are agents or are they simply governed in how to act by regular systems level programs?

The environment has a set of laws that govern how the state changes based on events and actions. Agents may or may not know these laws. It is said that if an agent knows the outcome or probability of the outcome of it's actions (for stochastic environments) then the environment is known by the agent. If not, the environment is unknownen by the agent.

These properties determine how complex an environment is. The final two are whether the environment is deterministic or stochastic 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 (fully observable) and the environment is deterministic the agent can guarantee the outcome of their action.

An environment is considered discrete if 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, stochastic, partially observable, dynamic, sequential, multi-agent and unknown.

7.2 Game Theory Background

7.2.1 Introduction

The theories that attempt to explain cooperative phenomena are not simply limited to direct and indirect reciprocity. Those two are simply the ones I chose, as I believe they could be used best as a mechanism to help cooperation evolve in multi-agent interactions. There are a multitude of mechanisms proposed including the three I will described here: Kinship Theory, Network Reciprocity and Group Selection.

7.2.2 Kinship Theory

Axelrod and Hamilton [10] described the way in which cooperation in nature (with the exception of homo-sapiens) is almost always between related individuals. An earlier paper by Hamilton [28] argues that individuals don't only work toward improving their own fitness, but towards what Hamilton defines as 'inclusive fitness'. Inclusive fitness is the sum of a player's fitness and the fitness of each of their relations multiplied by a coefficient. The coefficient used by Hamilton is Wright's coefficient of relatedness, as illustrated in figure 7.1. It could be possible to create a similar coefficient of relatedness for use in a MAS.

Richard Dawkins [17] advocated for the idea of the selfish gene. From a biological perspective, this idea postulates that actors are hardwired to propagate their genes. Dawkins asserts that this drive is due to the fact that genes are the true replicators evolutionarily rather than the actors themselves. Those with a high coefficient of relatedness to an individual are far more likely to carry their genes and to help them proliferate. This mechanism is similar to that presented by Hamilton [28], but with a closer biological backing. From a biological perspective this may make sense. However, it does not seem natural to translate an agent's strategy to the idea of genes.

Further, although it is possible to create a coefficient and an idea of relatedness similar to that of Hamilton's model [28] for a MAS, it does not seem a natural translation. Another limitation to the use of Kinship Theory for MASs is that systems are ideally inclusive of individuals that can contribute to the society. For example, if an agent is looking to actively contribute to a society, but is not kin with the members, a MAS using kinship theory would exclude them and thus limit the abilities of that society.

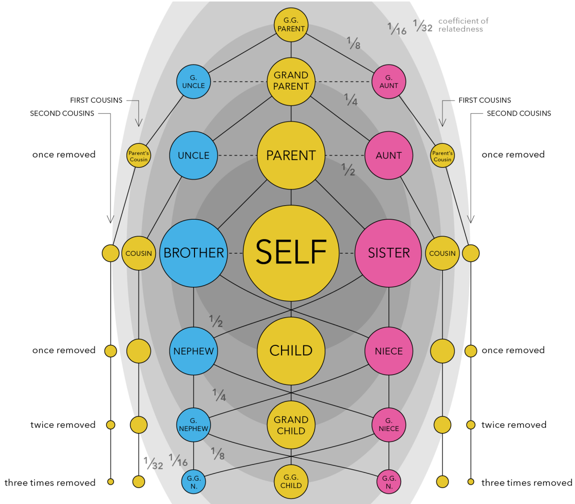


Figure 7.1: Wright's coeffcient of relatedness by Citynoise - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=37723128`

Figure 7.2: 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 [23]

Furthermore, Axelrod and Hamilton [10] highlight that humanity is the exceptional society which does not limit itself to cooperating only with kin. I would surmise that this exception is due to the higher level of intellect of homo-sapiens in comparison to other species. Many have suggested that the capabilities of AI could match or even surpass the intelligence of humans. Therefore, I would suggest that societies of IAs should also not be limited to the use of kinship theory to facilitate cooperation.

7.2.3 Network Reciprocity

Nowak - in his paper 'The Five Rules of Cooperation' [51] - identified and compared five key mechanisms that can aid in the evolution of cooperation, three of which I have already discussed (direct reciprocity in section 2.3.4, indirect reciprocity in section 2.3.5 and kin selection in 7.2.2). The other two are network reciprocity and group selection.

Network reciprocity uses a graph of players and their connections. The players are represented by the nodes in the graph, with arcs representing connections between players. This idea ties closely to the networks that IAs may work across. Players with arcs between them interact with each other in rounds of The Prisoner's Dilemma. Nowak and May's [52] ealier work - which inspired Nowak's later paper [51] - did not give individual's any memory of past interaction.

This lack of memory limited Nowak and May to pure cooperators and pure defectors. In Nowak's book 'Evolutionary Dynamics' [50], his exploration of evolutionary graph theory and spatial games (chapters 8 and 9) showed that the shapes of the lattice linking the players and different concentrations of cooperators and defectors on those shapes has a great effect on the evolution of cooperation. Visualised in figures 7.3 and 7.4.

Nowak's [51, 50] and Nowak and May's [52] work on these games on graphs is limited in terms of strategies and also in terms of the fixed shape of its graphs. However, the work proves a key point: the structure of who interacts with whom can play a key role in supporting cooperation in large populations.

In real life, individuals will often mostly interact in their close social circles. For example, a Meerkat may interact with others of their family group, a drongo bird which calls to warn of predators, the predators and others who are geographically close to them. The graph in this case represents the close geographic ties.

I imagine the use of Nowak [51, 50] and Nowak and May's [52] work to employ a network not as a representation of a physical network structure or geography, but as a representation of the choices made by IAs with whom they wish to interact with.

This network would be a constantly changing and adapting network of IAs. The IAs would not concern themselves with the strategy they employ towards whom they are forced to interact with. Instead, their strategy is to select those whom they wish to interact with, thus, effectively constructing a graph of network reciprocation. How these changing graph connections would affect cooperation is unbeknownst to me. Indeed, whether Nowak's rules would still apply would be interesting to find out.

Nowak [50] found that some shapes supported cooperators in groups. Cooperators could make use of these shapes by deliberately forming them to protect one another. While another set of shapes were found as 'amplifiers' for evolution, maybe defectors could make use of these sorts of shapes to invade groups of cooperators.

I can see IAs having strategies as to how to build these shapes. However, an issue may arise in which cooperative agents find it hard to reach out to other cooperators which are not part of their current shape. As such, there may be possible prevention of the spread of cooperation, thus limiting these groups. However, this concept is worth investigating, and the problem could possibly be overcome using some kind of bridging mechanism.

This is comparable to Jennings' [32] organisational relationships between agents, and could described strategies for agents to form and manage these relationships. Furthermore it would be interesting to see how this work could compare to Barabàsi's breakthroughs in network science [11] and the impact it could have on technologies such as the semantic web.

	A	B
A	a	b
B	c	d

Table 7.1: 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 [50].

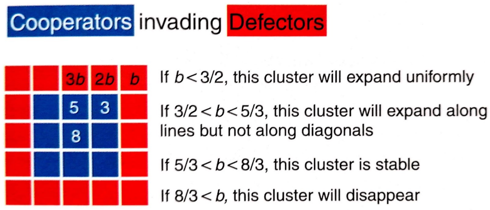


Figure 7.3: How can cooperators invade defectors? Taken from Nowak's book Evolutionary Dynamics [50]. The squares represent nodes and the players interact with the players to each side of them and diagonally. The value b is from the payoff matrix in table 7.1

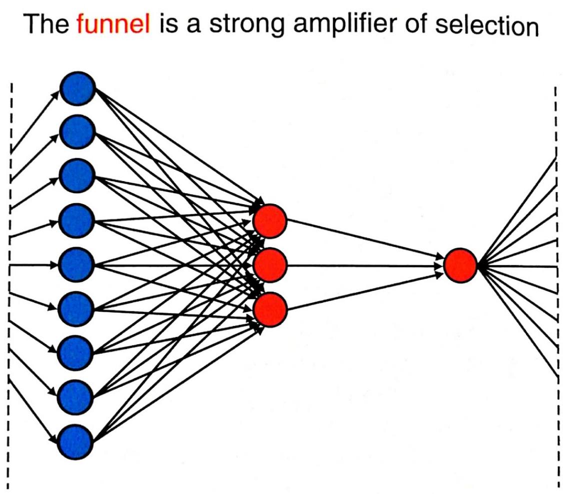


Figure 7.4: Shapes that can amplify selection include the funnel, the star and the superstar [50]

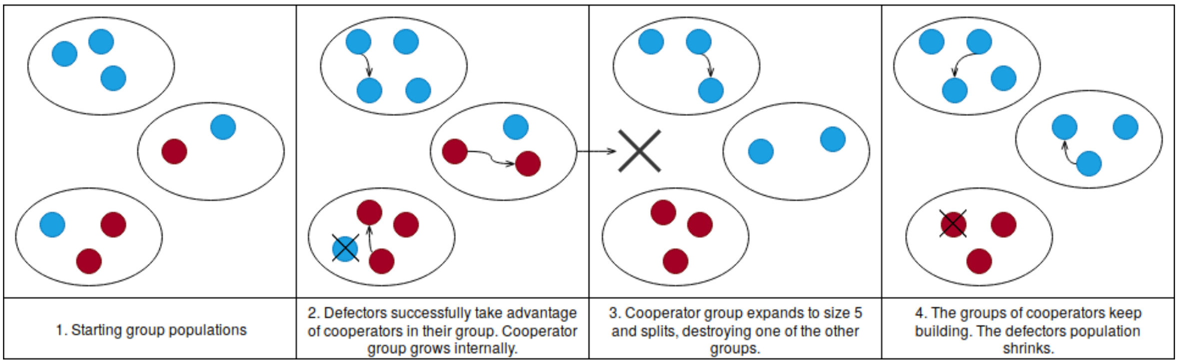


Figure 7.5: The dynamics of multi-level selection as described by Traulsen and Nowak [72].

7.2.4 Group Selection

Group selection is another mechanism described by Nowak [51]. This mechanism splits one population into multiple groups. Within these groups, The Prisoner's Dilemma is played and reproduction occurs which is proportional to each players payoff. If using the payoff matrix in table 2.2, then cooperators can work together to produce a payoff of three, while defectors can only produce five or two for both players in the interaction.

The group size increases until a certain point, at which the group may split. If the group does split (which is stochastically chosen) then another group is destroyed. The effect is multi-level selection.

Nowak found that due to the higher payoff between cooperators, reproduction will occur more quickly within the groups they dominate, than the groups filled with defectors. The faster the reproduction, the quicker the group size grows, making it more likely for groups of cooperators to survive while groups of defectors will shrink and be destroyed. These dynamics are displayed in figure 7.5.

Traulsen and Nowak [72] limited themselves to cooperators and defectors, but noted that other strategies could be built into their model. A limitation to applying group selection to MASs is the groups themselves. In this model, individuals do not interact with individuals in other groups, creating a barrier between them.

Even our current networks span the globe and don't always have harsh barriers between them - even when security is high, these barriers can often be broken. Furthermore, IAs are often built to be of service to others and limiting them to being of service only to one group greatly reduces the service an agent can supply and limits the society as a whole. Finally, the group mechanics of splitting and destroying another group of individuals does not naturally match the paradigm of MASs.

I would suggest that this is not a mechanism which would be useful to apply to MASs unless you are modelling clusters of agents split into local area network like structures, with each cluster in competition. This idea is not the aim of my project; however it could be another interesting project to take up.

7.2.5 Reproduction and Genetic Algorithms

Francq [20] puts forward two types of reproduction: proportional selection and tournament selection.

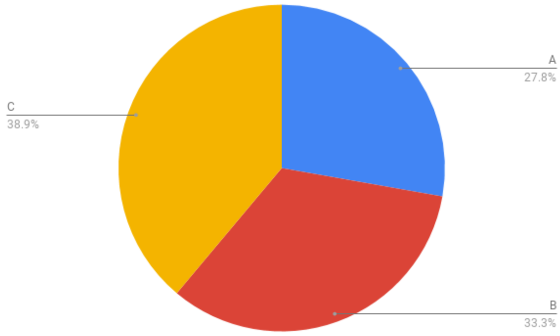


Figure 7.6: The roulette wheel from my proportional selection example

Here is an example of how proportional selection could be used to find optimal strategies in a MAS. 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 7.6 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 used to find optimal strategies in a MAS 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 as we did in the proportional selection algorithm we give an example of this in table 7.2.

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 7.2: Tournament selection

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.

7.3 Theoretical Framework

7.3.1 Environment Properties

I have already discussed how the synchronised cycle steps makes my environment static, but what about Russell and Norvig's [61] 6 other properties. Knowing the intention of other agents is key to deciding on whether other agents should cooperate or not. However, agents cannot view all interactions, and gossip may be distorted so an agent may not know all details it needs to know for action decisions. As such, the environment I have delineated is only partially observable.

I also argue that the environment is deterministic. Milinski *et al.* [40] claimed that individuals using the standing strategy aim for a good standing, and Leimar and Hammerstein [37] argue that the standing strategy is good as it allows individuals to punish bad individuals. However take 3 agents a, b, c and d. Agent b has defected previously against d who has a good standing according to a. Agent a then chooses to punish b but is still believing this won't reduce their standing. However, agent c did not observe b's defection against d but is an onlooker for c's defection against b.

From the proceedings the environment appears to be non-deterministic, as there is more than one outcome to an action. However, this is due to the partial observability of the environment not the determinism, which is highlighted as possible by Russell and Norvig [61]. The current state (c's lack of knowledge on b) and the actions selected by a and b completely determine the next state of the system.

The environment is also sequential. To take a proof by contradiction approach suppose the environment is episodic. According to Russell and Norvig [61] an environment is episodic if subsequent episodes do not depend on what action occurs in previous episodes. Episodes consist of perceiving and then deciding. But from the percepts and actions subsections (3.2.3 and 3.2.4) we know that actions from one timepoint can generate percepts in the next. This is a contradiction and as such the environment is nonepisodic.

My environment is used to study interactions between multiple agents in an environment and as such has the property multi-agent. My environment is also known unto the agents, there is nothing stopping an agent being aware of the utility produced by their actions and the generated percepts of their actions. Though these may not have the effect an agent wishes for in terms of increasing their image score etc. this is down to the mental states of other agents not the environment itself.

In summary the environment I have outlined in this section is partially observale, deterministic, sequential, static, discrete, multi-agent and known.

7.4 Running the Software

It is possible to view a version of the website that is deployed as the natureengine.tech website. When attempting to access the website there will be a security warning flash up, this is simply due to the use of self signed certificates which will soon be rectified by getting some Let's Encrypt certificates.

I would actually suggest running the software locally as the server can be temperamental with timeouts due to the lack of processing power which I can afford for the server. The steps to set up and run this software on your local computer are described below. Please note I have tested and run this setup procedure on Kubuntu 18.04, and as such this set up is likely to work for Debian distributions, but some steps may vary on other Linux distros, Mac and Windows. My belief is that most of the commands stipulated are generic to at least Linux and Mac as far as I know.

7.4.1 AMAAS

Running the web application requires AMAAS to be running on port 8080. To do this ensure you have Prolog installed (this has been thoroughly tested on version 8.0.2, but has run on version 7.6.4 as well) and navigate to the directory:
FullUnit_1819_JamesKing/AgentsService/src. Run the command: 'prolog run.pl', which starts the web service on port 8080.

Alternatively to specify a port for the service to run on. Navigate to the same directory run the command: 'prolog main.pl'. Then run the predicate: 'server(Port)' where Port is the number of the port you wish to run the service on. To integrate this with the web application you will have to either set the environment variable AGENTS_URL to 'http://127.0.0.1:Port/' or edit the config.py file in the FullUnit_1819_JamesKing/NatureEngineWebApp directory to set the AGENTS_URL variable to that url.

This web service can also be tested using an application like Postman or Insomnia REST clients or simply by sending requests using a web browser.

7.4.2 The Nature Engine

To run the web application locally ensure that you have Python 3.6.7 or later, pip3, virtualenv, redis, sqlite installed and a redis-server running as a service on the url:
'http://127.0.0.1:6379/'. Here I will detail how to set up and run the application. There are many different ways to set up and run the web application, including using an IDE, however, this is my tried and tested method.

To create the virtual environment for the application navigate to the
FullUnit_1819_JamesKing/NatureEngineWebApp directory and run the command: 'python3 -m virtualenv venv'. Then run the command: 'source venv/bin/activate'. The next step is to run the command: 'pip install -r requirements.txt'. This will install all the required libraries for the project.

From here we need to set up the database - with the virtual environment still active - we can run the commands: 'flask db migrate' and then 'flask db upgrade' which runs the whole set up process. In another command line window we then navigate to the FullUnit_1819_JamesKing/NatureEngineWebApp directory, run 'source venv/bin/activate' and then 'rq worker nature_engine_tasks'. This spins up a redis queue which is used to run tournaments and reputation games in the background.

In the original command line window with the virtual environment still active we can run the command 'flask run' and the server will start. To use the website open up a browser and type in the url 'http://127.0.0.1:5000/'.

7.4.3 The Multi-Agent System

The MAS I have created can also be run in a standalone manner using the Python interpreter. This requires AMAAS to be running and the first two paragraphs of The Nature Engine setup subsection to be complete. When these have been completed navigate to FullUnit_1819_JamesKing/NatureEngineWebApp directory and run 'source venv/bin/activate'.

Then run the python interpreter with 'python'. From here you may need to extend you Python's system path with 'import sys' and then 'sys.path.extend(PATH)' where PATH is the full path to the FullUnit_1819_JamesKing/NatureEngineWebApp directory. You can then import the relevant classes with 'from app.indir_rec.facade_logic import ReputationGame, Results'.

To run a game create a population of players as a python list. The entries to this list should be dictionaries, for example:

```
{
  "donor_strategy": "Veritability Discerner",
  "non_donor_strategy": "Lazy",
  "trust_model": "Balanced Reactor",
  "options": [5],
  "count": 6
}
```

The fields should be filled with strategies in the system - viewable in the appendix section 7.7 - and the count entry is the number of these strategies to include in the population. Using this population create a ReputationGame object 'game = ReputationGame(population, num_of_onlookers, num_of_generations, length_of_generations, mutation_chance)'. Where the last parameter is a float between 0 and 1, the num_of_onlookers is an integer greater than 0 and so are the num_of_generations (must be greater than 2) and length_of_generations (must be greater than 5).

A note before running the game: running this takes time and a lot of computational resources. The more you increase the length of generations and number of generations the more time it will take. I would suggest using a generation length of around 30-40 to start with and a number of generations around 5 to begin with, and then increasing if you see fit.

Running the game returns a Results object which you can use to view certain statistics about the game, to run the game: 'results = game.run()'. There are a number of statistics available (documented in the nature engine documents) including the cooperation rate of the whole game ('results.cooperation_rate') and the social activeness of each generation ('results.social_activeness_by_generation').

7.5 User Manual

Following is a user manual for The Nature Engine web application. This manual presumes that you either have an instance of the server running locally (http:/127.0.0.1:5000/) or are on the website (natureengine.tech).

Figure 7.7: The navigation bar

Figure 7.8: The navigation bar when logged in

7.5.1 Navigation

To begin with you will find yourself at the home page. At the top of the page is a black navigation bar with a number of options to click on to navigate through the website. At the furthest left of the nav bar is a glyphicon and the name of the website, click on this or the 'Home' button next to it to return home at any time.

To the right of the 'Home' button is the 'Match' button, which takes you to a page where you can set up a match of the iterated prisoner's dilemma. If you press it you will be presented with the options advanced and basic, when pressing basic you will only be presented with the most popular of strategies that you can use, if you press advanced a far longer list of strategies are available.

The advanced and basic options are available when clicking on the 'Tournament' button as well. The page that the 'Tournament' button takes you to allows you to set up and run a round-robin tournament similar to that of Axelrod and Hamilton's [10].

Both the tournament and match pages use the Axelrod-Python library in the back-end. However the button to the right of the 'Tournament' button (the 'Reputation' button) takes you to a page where you can set up a custom 'reputation game' using the theoretical framework I have devised.

You can log into the system using the login page reachable from the 'Log In' button. The final button is 'About' which takes you to a page that gives some limited deatils on the website and project. The nav bar is displayed in figure 7.7.

7.5.2 Log In

For the reputation game a registered user can attach a label to revisit the results of the experiment later on. To do this you must have an account, create one by using the navbar 'Login' button - the second from the right. Click the link with the text 'Click to Register' and fill in the details to register.

If successful this will return you to the login page in which you can log in. Input your username and password, and click 'Sign In'. When setting up a reputation game you may now attach a label in the form, which you can the search in the my experiments page. To navigate to this page make sure you are logged in and click the 'My Experiments' button - third from the right. The button to the right of this allows you to log out.

A screenshot of the nav bar when logged in is displayed in figure 7.8 and a screenshot of the my experiments page is displayed in figure.

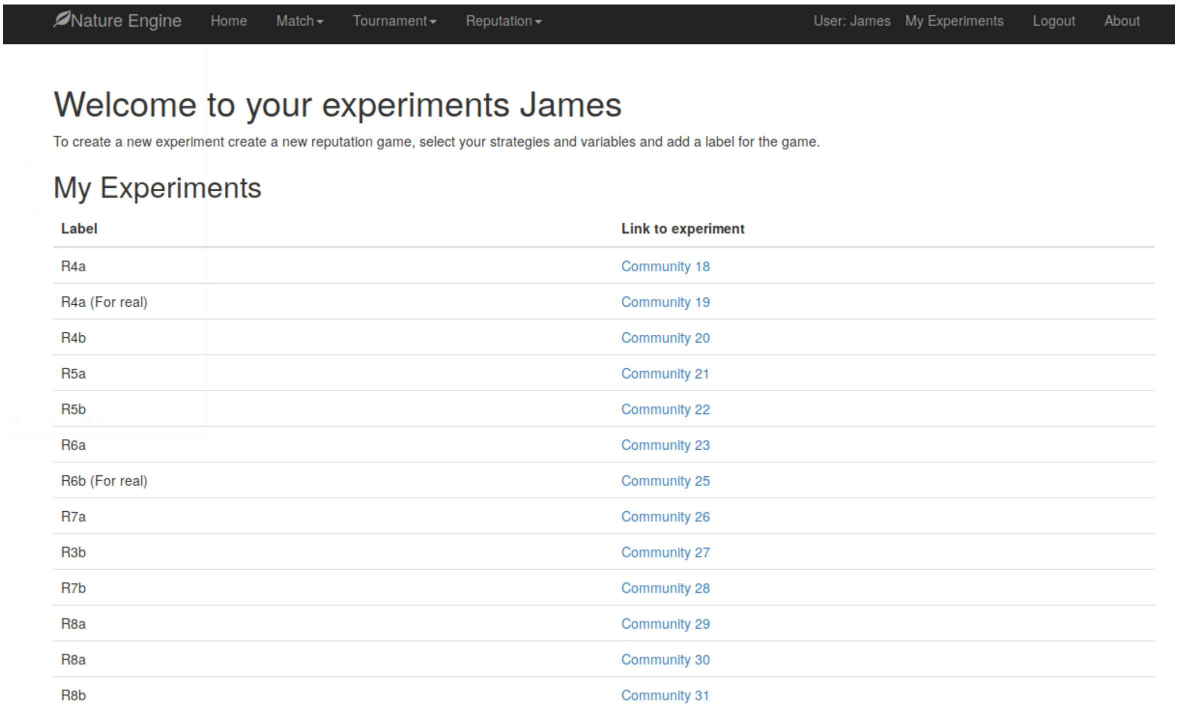


Figure 7.9: The my experiments page

7.5.3 Home Page

The home page displays information in relation to the project's game theoretic grounding and some multi-agent system information. This information is displayed in a carousel with images displaying themes of cooperation, teamwork and technology. A screenshot of this is displayed in figure 7.10.

7.5.4 Match Setup and Analysis

The match page allows you to set up a match of the iterated prisoner's dilemma. Select a strategy for each player from the two drop down menus and the amount of rounds to play (must be between 2 and 50). A description of the available strategies is available in the information box at the bottom.

The game is then run and you will be presented with an analysis screen, at the top of which is an animation of what has happened. Below this the player's points are displayed below this animation and a table of what interactions occurred. A reminder of how strategies are assigned to players is available in the information box on the left and the information on the player strategies in the information box at the bottom.

Screenshots can be seen in figures 7.11, 7.12 and 7.13.

7.5.5 Tournament Setup and Analysis

The tournament page allows a user to set up a round-robin tournament of the iterated prisoner's dilemma. Select a number of strategies between 3 and 50 by clicking the add buttons on the relevant strategies. You can add multiple of the same strategies, the strategies and their count will appear on the right hand side where you can remove an individual from

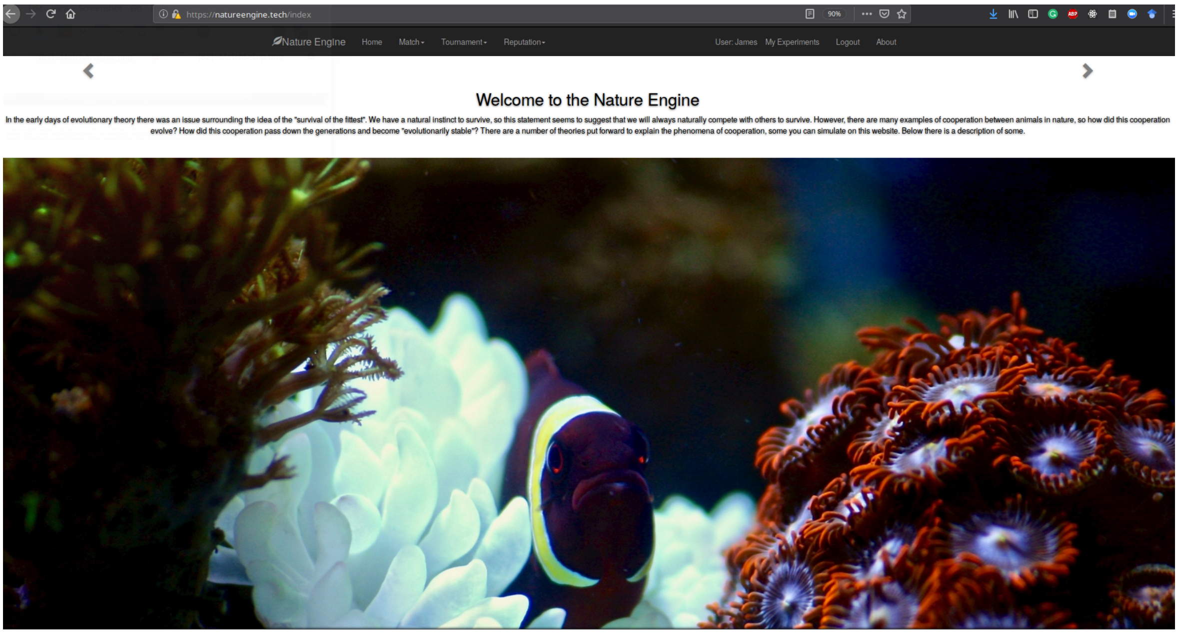


Figure 7.10: The home page

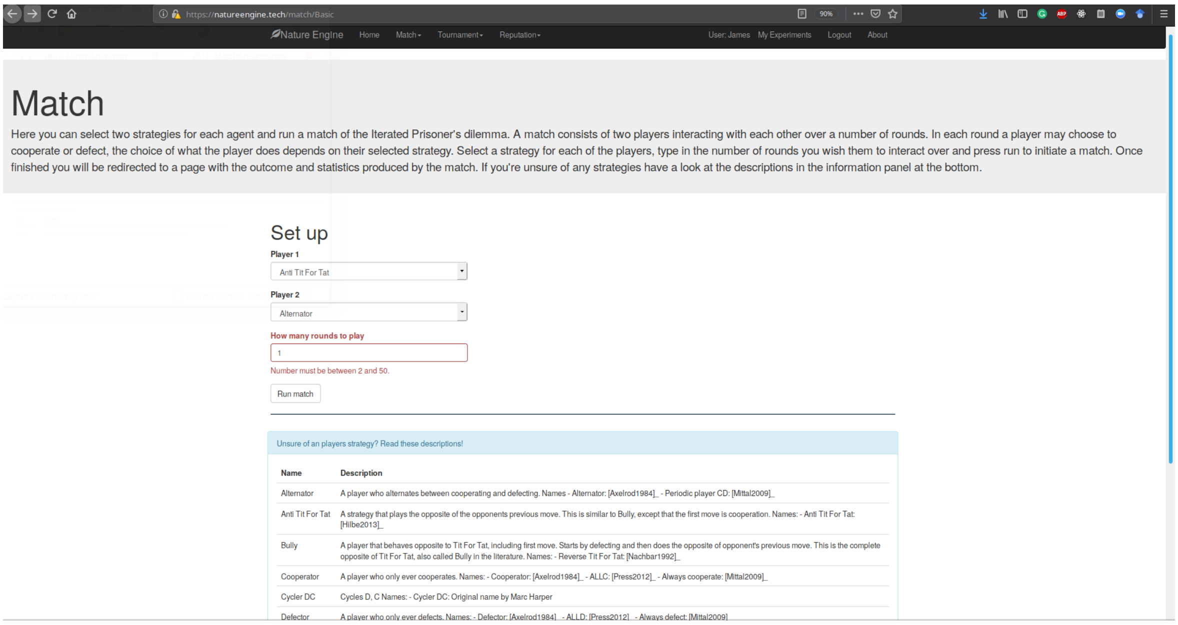


Figure 7.11: Setting up a match

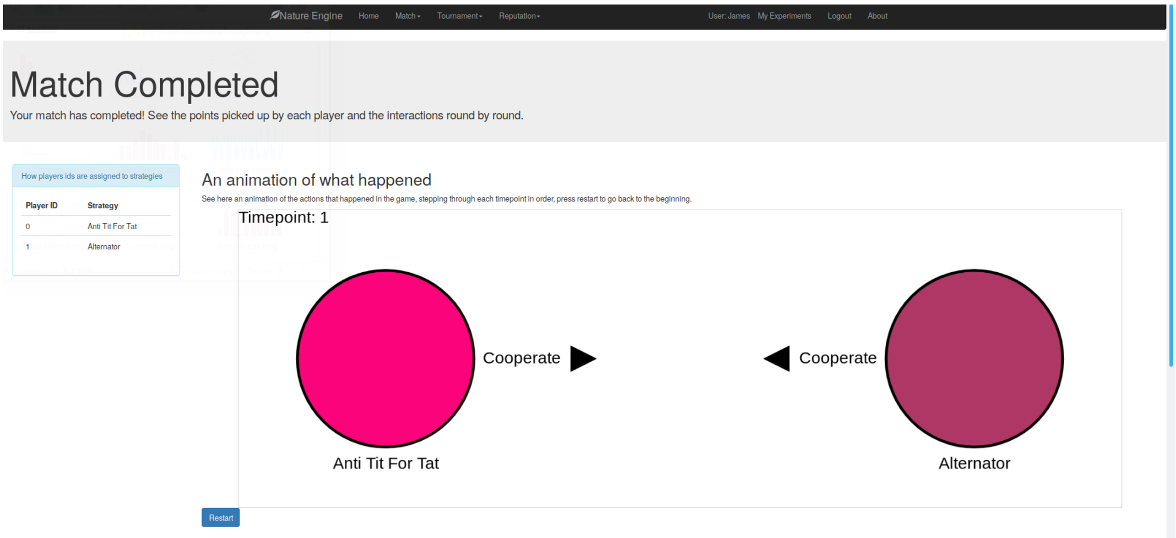


Figure 7.12: The animation part of the match analysis

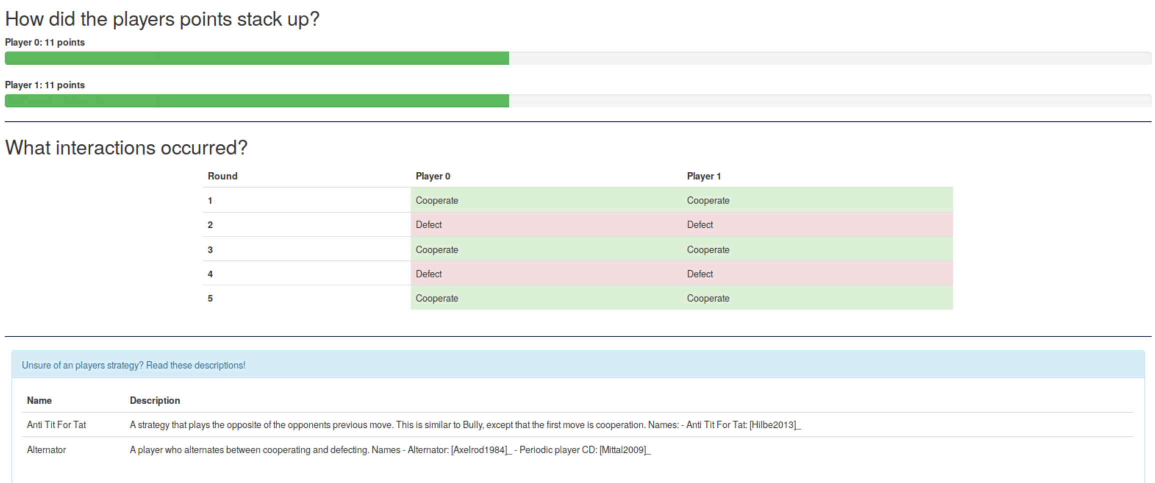


Figure 7.13: Further down on the match analysis page

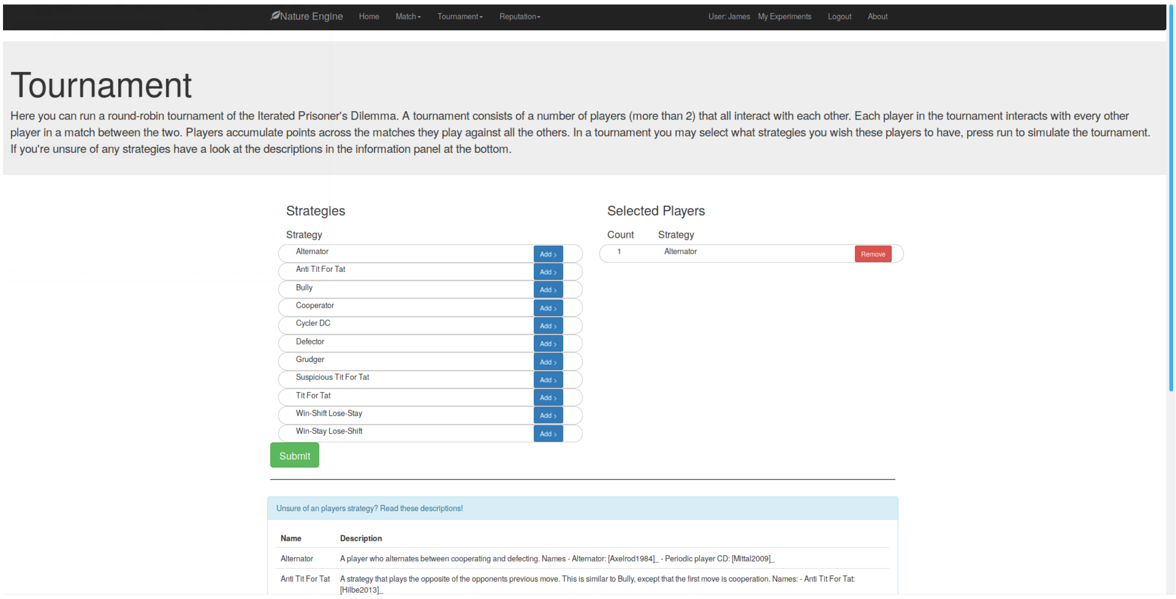


Figure 7.14: Setting up a tournament

that list using the red 'Remove' button. A description of the strategies can be seen further down the page in the information box. Run the game by clicking the green 'Submit' button. A screenshot of this page can be seen in figure 7.14.

If this tournament times out you will see a timeout page and will not be able to view any statistics, to prevent this either reduce the amount of players you select or run the server locally. However, if the tournament simulation completes, you will be taken to an analysis page where you can view the top 3 players in terms of points, view the points each agent has accrued, and see which agents have been cooperative and which have been non-cooperative. Similarly to the match analysis you can see on the left hand side in an information box which agents are associated with which strategy and what the strategies do at the bottom in another information box. Screenshots of the page can be seen in figures 7.15 and 7.16.

7.5.6 Reputation Game Setup and Analysis

Setting up a reputation game is similar to a tournament. Select a number of strategies using the relevant 'Add' buttons and set the parameters in the form above it. In this form you can select the number of onlookers that are chosen to select for each interaction. You can also select the number of generations to use for the genetic algorithm.

In the third field select the length of each generation, this is the number of timepoints that the perceive-decide-execute cycle is run for and in each timepoint an interaction occurs. For every agent created their strategy is selected in a reproduction step, the fourth field defines the likelihood (between 0 and 1) that each agents strategy is mutated from the one originally selected. The final field (only available to registered users) allows a user to attach a label to the experiment for easy reference later.

When all the fields have been filled and a number of strategies selected click the 'Submit' button at the bottom of the form to run the reputation game. This page is displayed in the screenshot in figures 7.17 and 7.18. You will then be redirected to a waiting page.

The running of the game may timeout, if it does not you will be redirected to an analysis where you can view statistics on the overall community, graphs on the statistics of

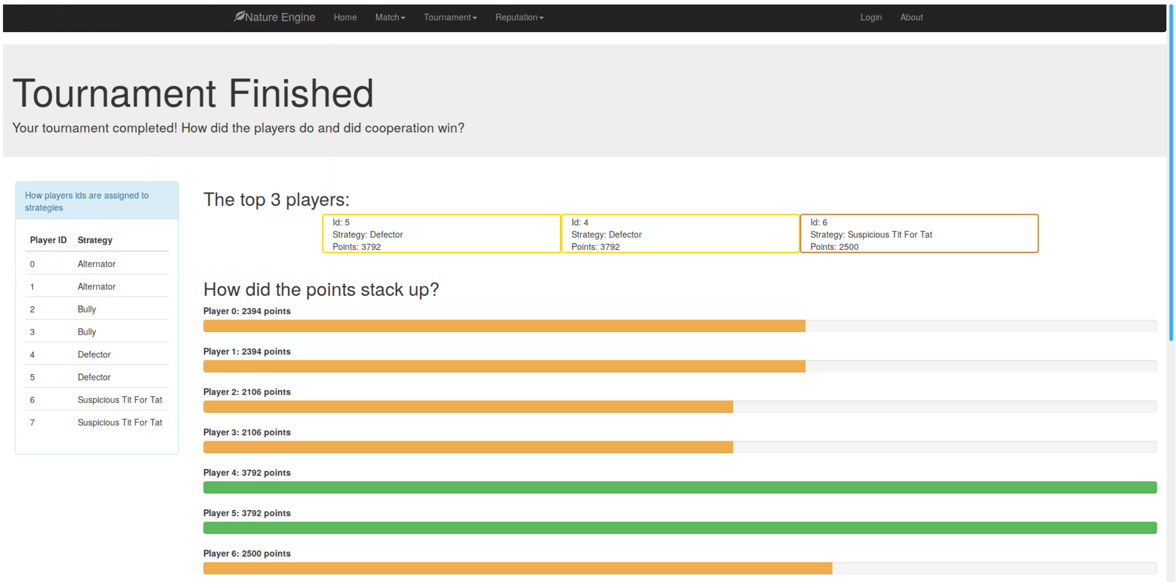


Figure 7.15: Analysis of a tournament

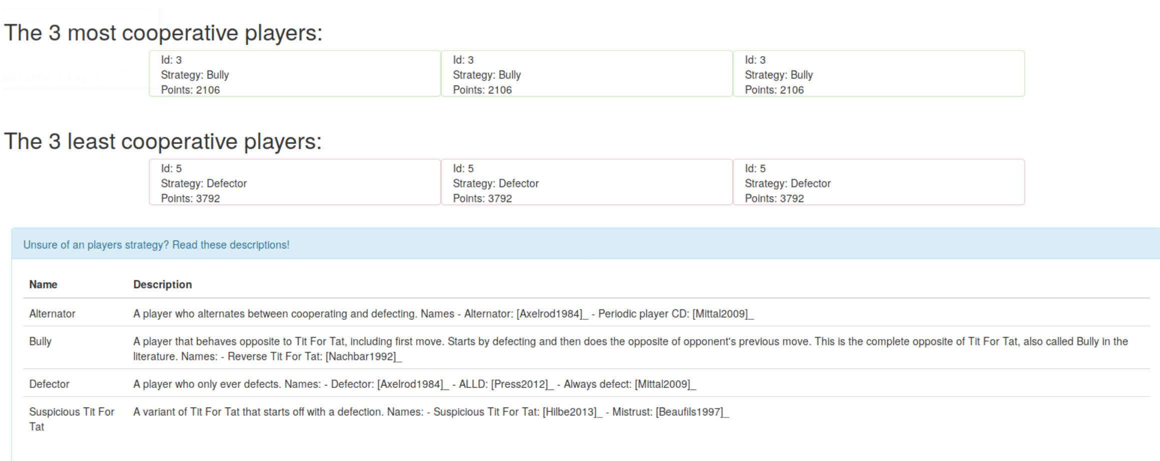


Figure 7.16: Analysis of a tournament

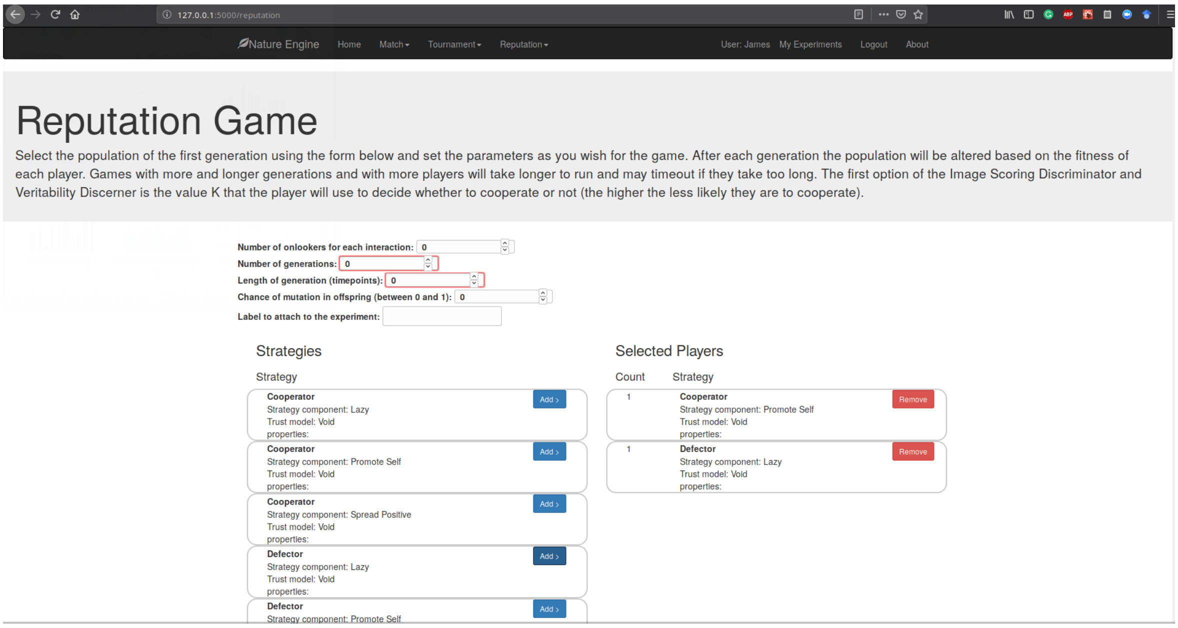


Figure 7.17: Set up of a reputation game

each generation including a population tracker, an animation of the interactions in the system and a table documenting all the interactions. A description of the statistics can be viewed in the information box on the left and strategy descriptions at the bottom of the page.

There is a screenshot of the analysis page in figures 7.19, 7.21, 7.20, 7.22 and 7.23.

7.6 Experiments

7.6.1 Robustness Experiments Population

- 1 x Cooperator, Lazy
- 1 x Cooperator, Promote Self
- 2 x Defector, Lazy
- 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

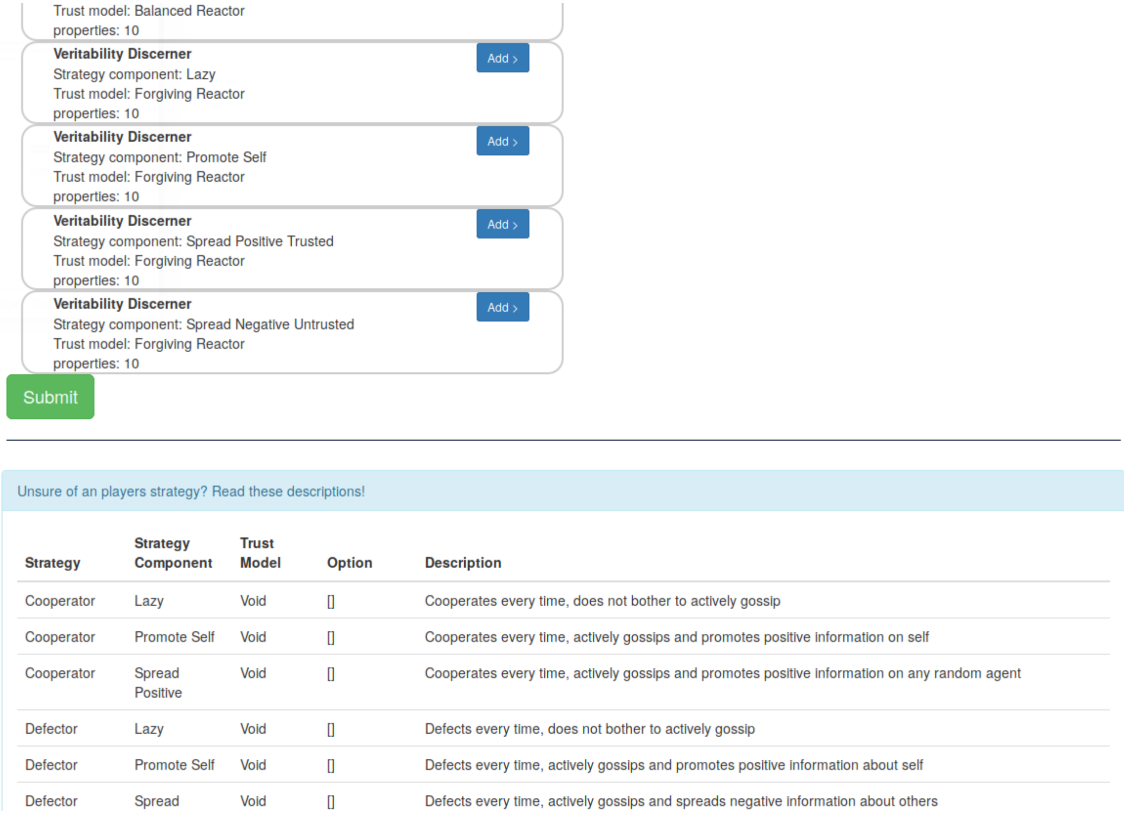


Figure 7.18: Set up of a reputation game

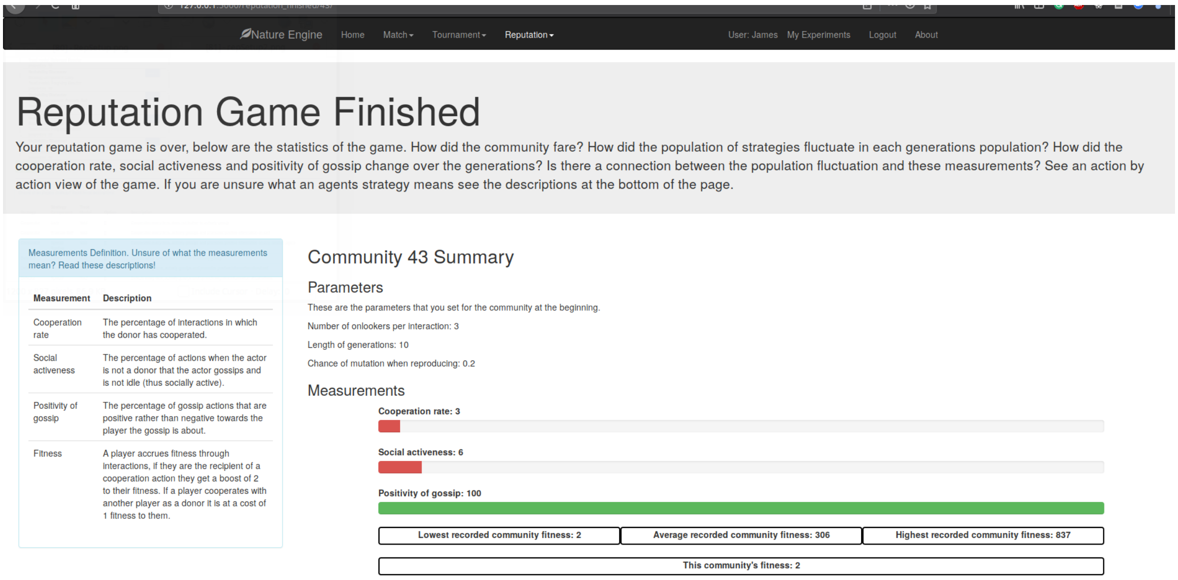


Figure 7.19: An analysis of the overall community playing the reputation game

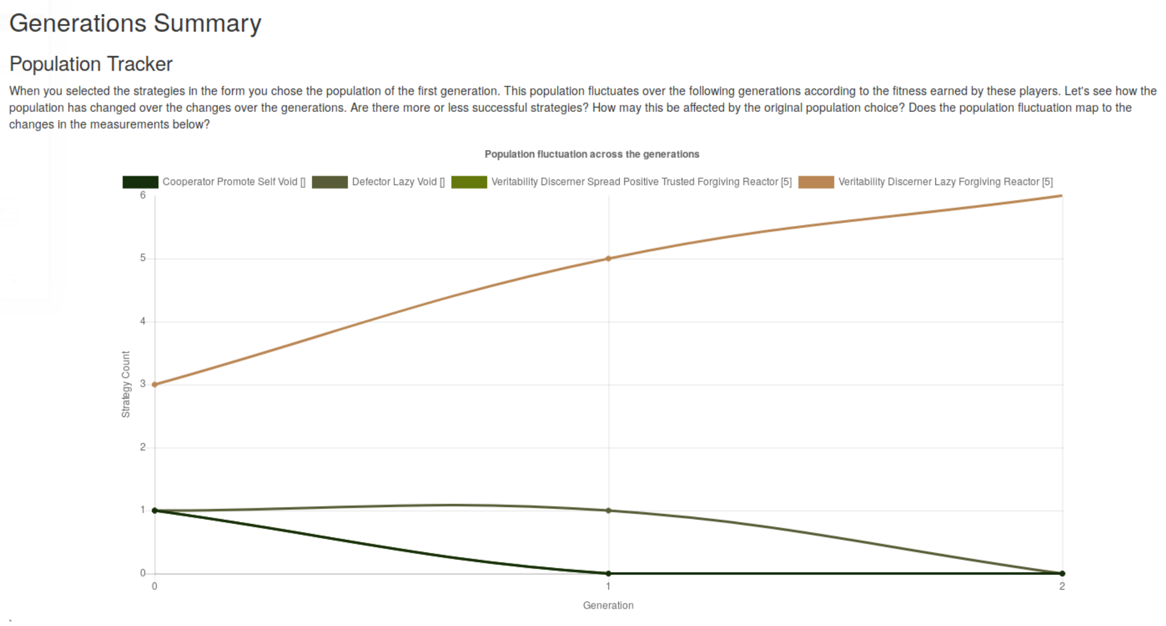


Figure 7.20: The population tracker for the reputation game

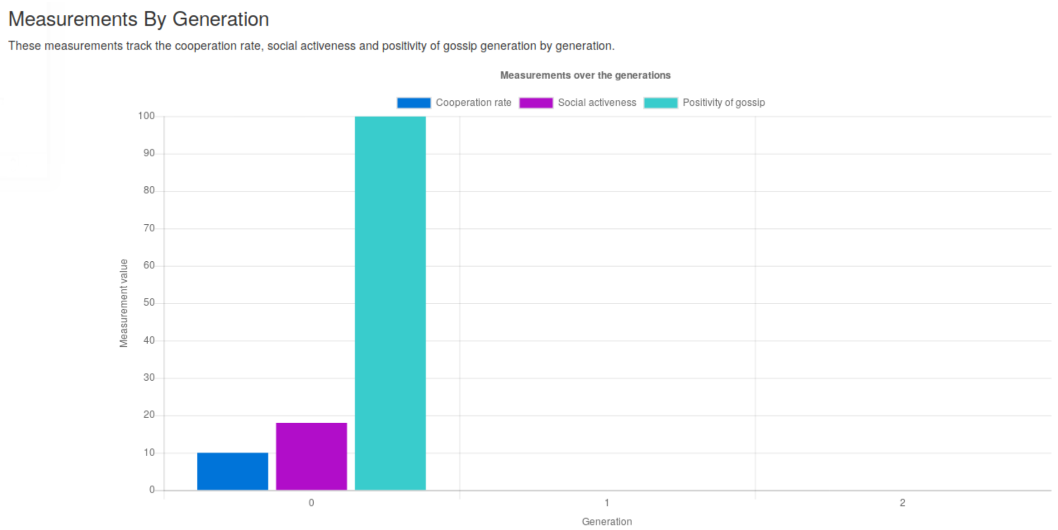


Figure 7.21: Measurements from each generation

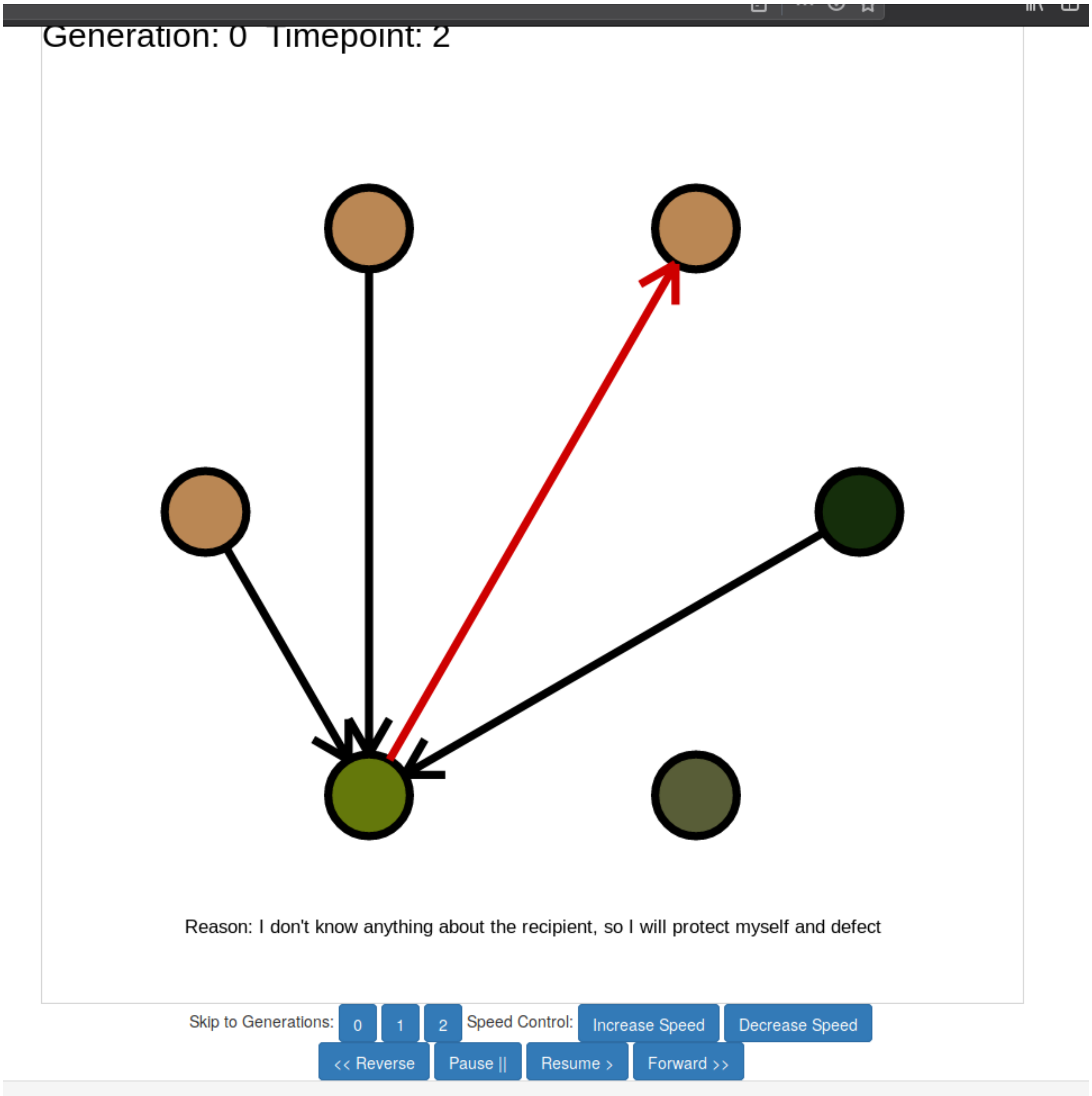
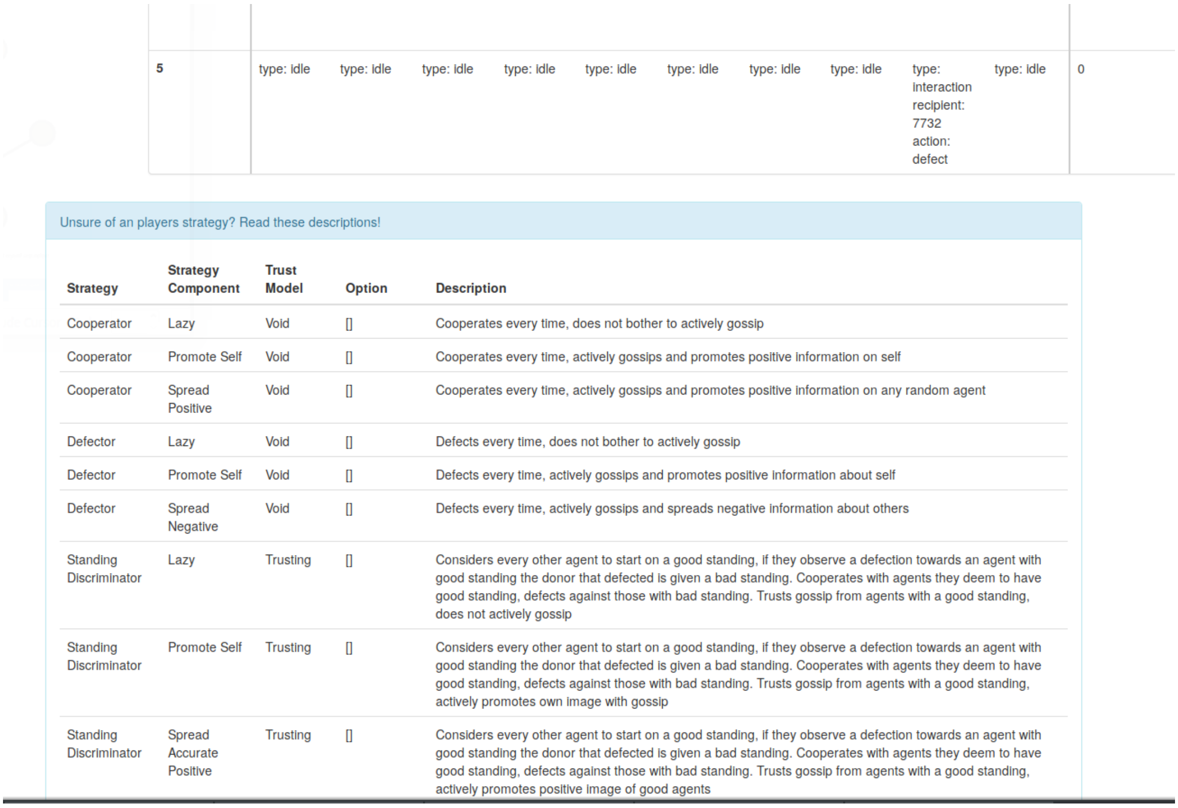


Figure 7.22: An animation of the simulated reputation game



5	type: idle	type: idle	type: idle	type: idle	type: idle	type: idle	type: idle	type: idle	type: interaction recipient: 7732 action: defect	type: idle	0

Unsure of an players strategy? Read these descriptions!

Strategy	Strategy Component	Trust Model	Option	Description
Cooperator	Lazy	Void	[]	Cooperates every time, does not bother to actively gossip
Cooperator	Promote Self	Void	[]	Cooperates every time, actively gossips and promotes positive information on self
Cooperator	Spread Positive	Void	[]	Cooperates every time, actively gossips and promotes positive information on any random agent
Defector	Lazy	Void	[]	Defects every time, does not bother to actively gossip
Defector	Promote Self	Void	[]	Defects every time, actively gossips and promotes positive information about self
Defector	Spread Negative	Void	[]	Defects every time, actively gossips and spreads negative information about others
Standing Discriminator	Lazy	Trusting	[]	Considers every other agent to start on a good standing, if they observe a defection towards an agent with good standing the donor that defected is given a bad standing. Cooperates with agents they deem to have good standing, defects against those with bad standing. Trusts gossip from agents with a good standing, does not actively gossip
Standing Discriminator	Promote Self	Trusting	[]	Considers every other agent to start on a good standing, if they observe a defection towards an agent with good standing the donor that defected is given a bad standing. Cooperates with agents they deem to have good standing, defects against those with bad standing. Trusts gossip from agents with a good standing, actively promotes own image with gossip
Standing Discriminator	Spread Accurate Positive	Trusting	[]	Considers every other agent to start on a good standing, if they observe a defection towards an agent with good standing the donor that defected is given a bad standing. Cooperates with agents they deem to have good standing, defects against those with bad standing. Trusts gossip from agents with a good standing, actively promotes positive image of good agents

Figure 7.23: The description of each strategy in the game

- 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

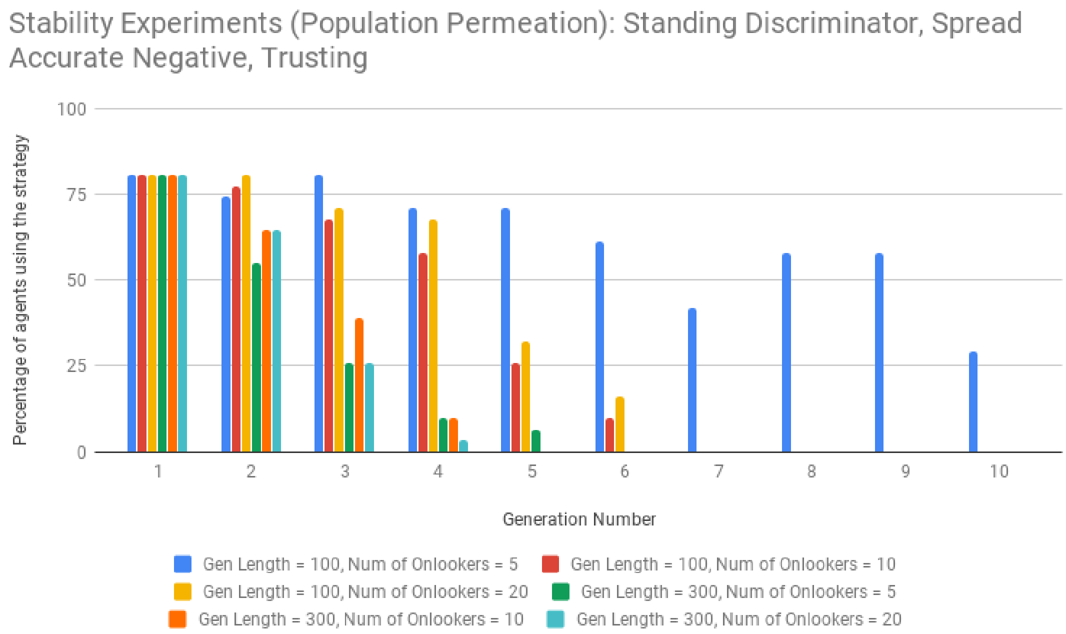


Figure 7.24: Did this strategy (Standing Discriminator, Spread Accurate Negative, Trusting) survive in the population

7.6.2 Stability Experiments Evaluation: Strategies

Strategy 1: Standing Discriminator, Spread Accurate Negative, Trusting

Figures 7.24 and 7.25 illustrate the effectiveness of strategy 1 as to how well it survived within the population (in comparison to the group of defectors) and how well the strategy maintained cooperation in the society. As we can see, the answer is not well. Cooperation died off in all but one experiment, and even in that experiment there is a trend towards cooperation falling.

This strategy was effective in one of the robustness experiments, but these experiments have not backed up that effectiveness.

Strategy 2: Standing Discriminator, Spread Accurate Positive, Distrusting

Figures 7.26 and 7.27 illustrate the effectiveness of strategy 2 as to how well it survived within the population (in comparison to the group of defectors) and how well the strategy maintained cooperation in the society. This strategy maintained cooperation for longer than strategy 1 in more experiments, but did not even once maintain cooperation or last in the population until the last generation. This strategy did not pass the test in terms of maintaining stable cooperation within the society.

Strategy 3: Standing Discriminator, Promote Self, Naive Trusting

This strategy failed the test as to whether it can uphold cooperation and survive an invasion by a small group of non-cooperative agents. Similarly to strategies 1 and 2 this strategy began each experiment with a cooperation rate around the 80% mark but this

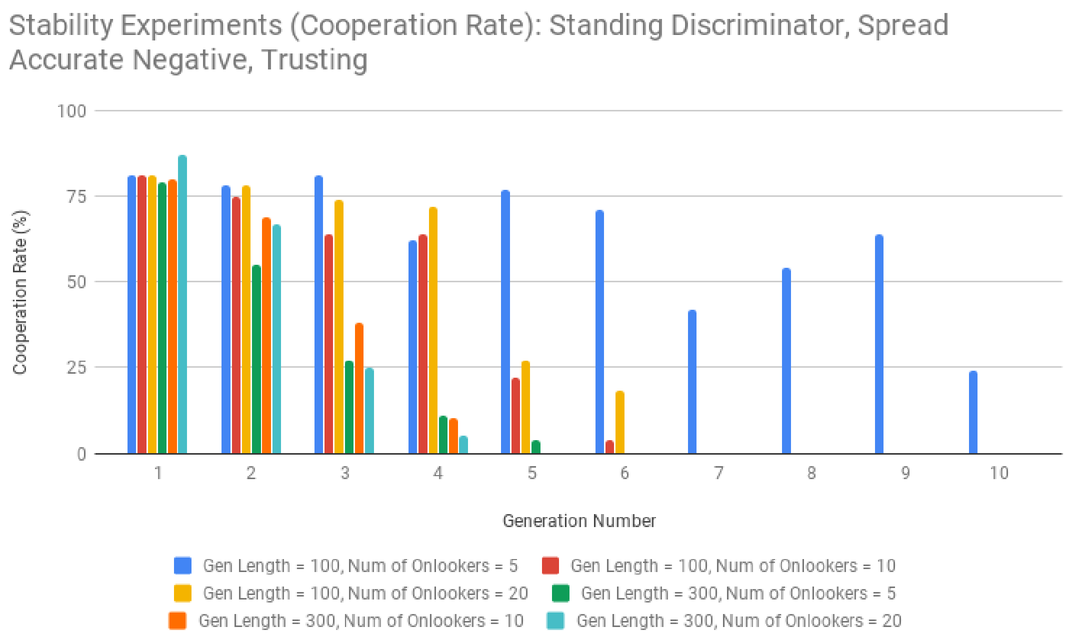


Figure 7.25: Did this strategy (Standing Discriminator, Spread Accurate Negative, Trusting) maintain cooperation in the society

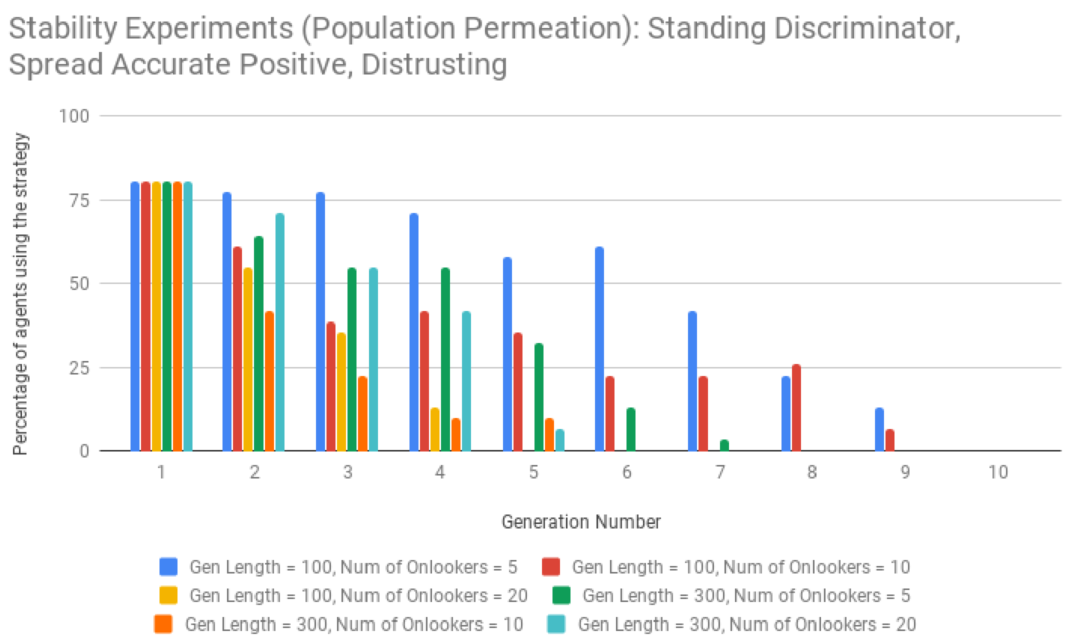


Figure 7.26: Did this strategy (Standing Discriminator, Spread Accurate Positive, Distrusting) survive in the population

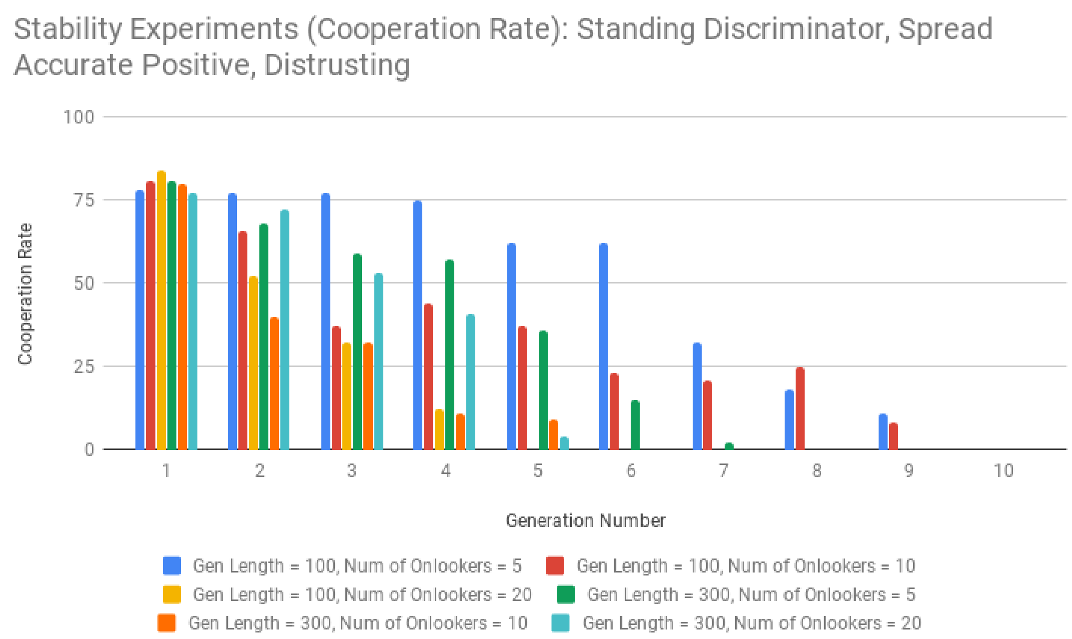


Figure 7.27: Did this strategy (Standing Discriminator, Spread Accurate Positive, Distrusting) maintain cooperation in the society

subsided fast as the strategy failed to maintain a foothold in the population - as illustrated in figures 7.28 and 7.29.

Strategy 3: Image Scoring Discriminator, Spread Accurate Negative, Trusting, [-2, Personal Grievance]

The overall trend of the experiments with this strategy is that stable cooperation has not been established within the community. Once non-cooperative agents have a large enough foothold within the society, the strategy being experimented with is inevitably wiped out. One experiment demonstrates that it is possible over a number of generations to support a high level of cooperation. This is illustrated in figures 7.31 and 7.30.

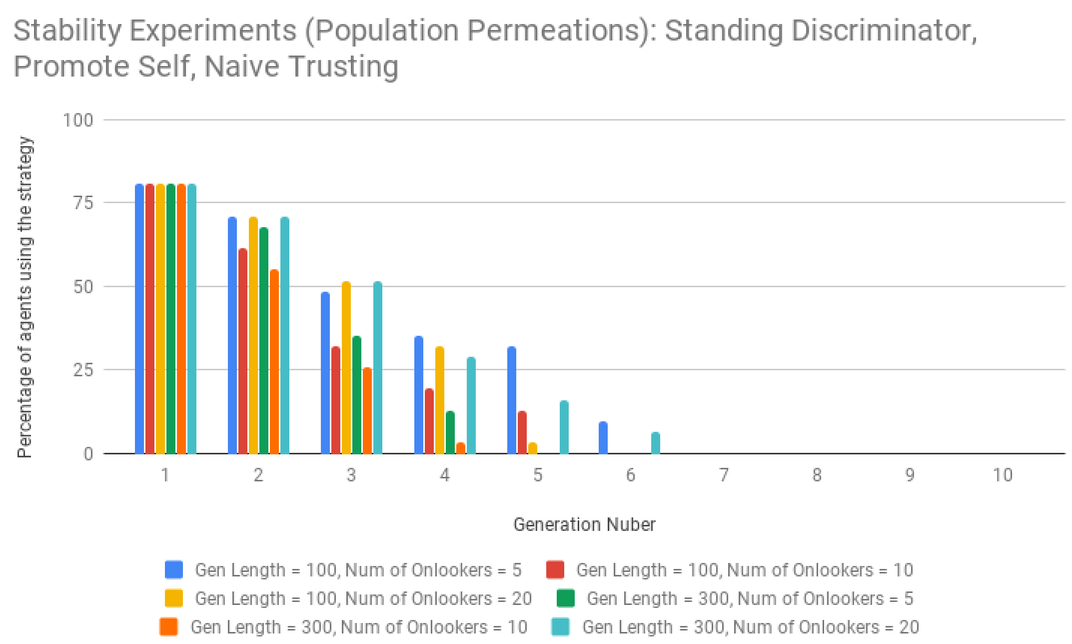


Figure 7.28: Did this strategy (Standing Discriminator, Promote Self, Naive Trusting) survive in the population

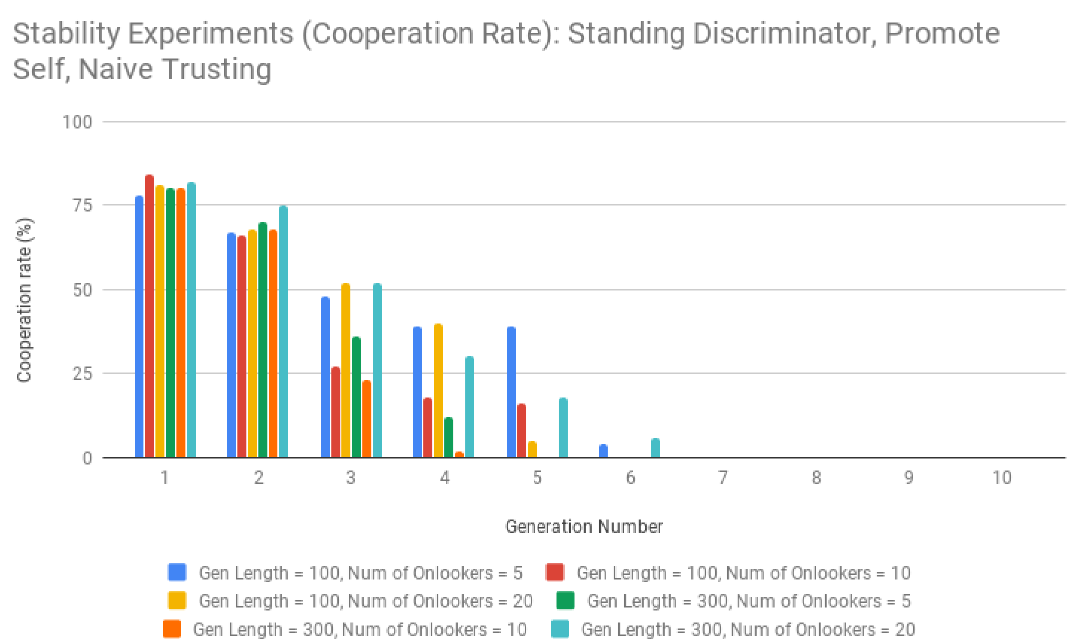


Figure 7.29: Did this strategy (Standing Discriminator, Promote Self, Naive Trusting) maintain cooperation in the society

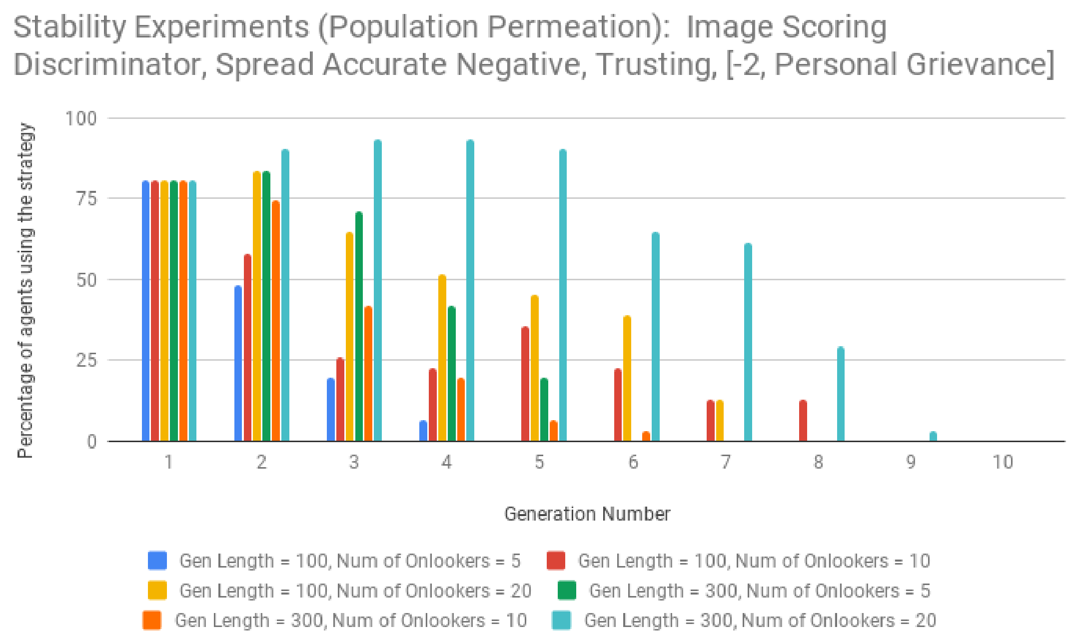


Figure 7.30: Did this strategy (Image Scoring Discriminator, Spread Accurate Negative, Trusting, [-2, Personal Grievance]) survive in the population

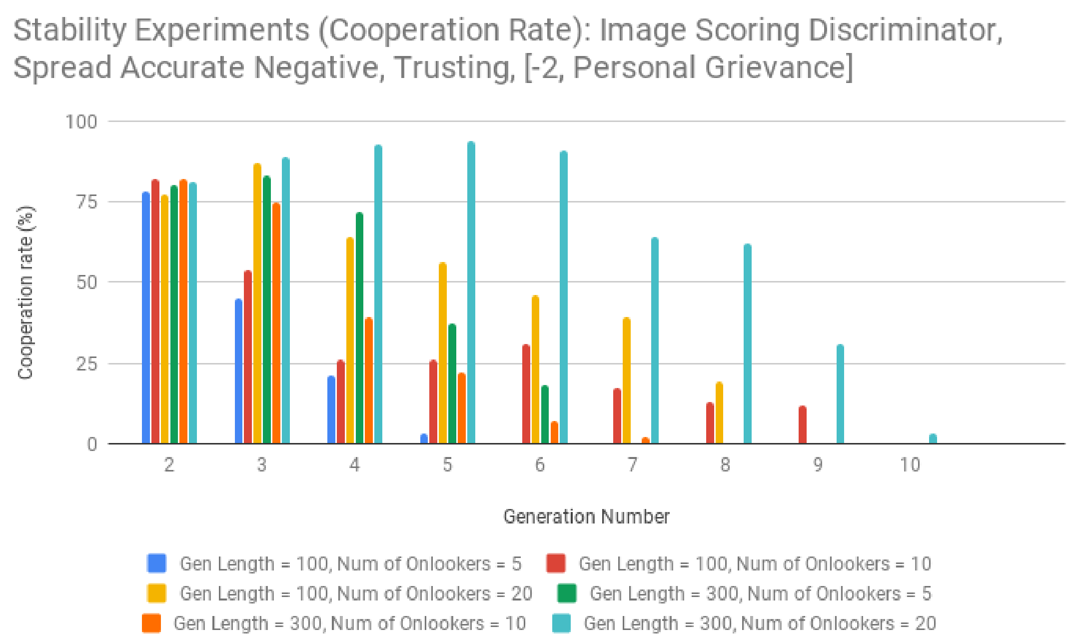


Figure 7.31: Did this strategy (Image Scoring Discriminator, Spread Accurate Negative, Trusting, [-2, Personal Grievance]) maintain cooperation in the society

7.7 Strategies

Below is a list of all the strategies available and a description for each strategy which can be associated with an agent in a game in my MAS.

```
[
  {
    "description":"Cooperates every time, does not bother to actively
    ↪  gossip",
    "donor_strategy":"Cooperator",
    "non_donor_strategy":"Lazy",
    "options": [],
    "trust_model":"Void"
  },
  {
    "description":"Cooperates every time, actively gossips and promotes
    ↪  positive information on self",
    "donor_strategy":"Cooperator",
    "non_donor_strategy":"Promote Self",
    "options": [],
    "trust_model":"Void"
  },
  {
    "description":"Cooperates every time, actively gossips and promotes
    ↪  positive information on any random agent",
    "donor_strategy":"Cooperator",
    "non_donor_strategy":"Spread Positive",
    "options": [],
    "trust_model":"Void"
  },
  {
    "description":"Defects every time, does not bother to actively gossip",
    "donor_strategy":"Defector",
    "non_donor_strategy":"Lazy",
    "options": [],
    "trust_model":"Void"
  },
  {
    "description":"Defects every time, actively gossips and promotes
    ↪  positive information about self",
    "donor_strategy":"Defector",
    "non_donor_strategy":"Promote Self",
    "options": [],
    "trust_model":"Void"
  },
  {
    "description":"Defects every time, actively gossips and spreads negative
    ↪  information about others",
    "donor_strategy":"Defector",
    "non_donor_strategy":"Spread Negative",
    "options": [],
    "trust_model":"Void"
  },
```

```
{
  "description":"Considers every other agent to start on a good standing,
   ↪ if they observe a defection towards an agent with good standing the
   ↪ donor that defected is given a bad standing. Cooperates with agents
   ↪ they deem to have good standing, defects against those with bad
   ↪ standing. Trusts gossip from agents with a good standing, does not
   ↪ actively gossip",
  "donor_strategy":"Standing Discriminator",
  "non_donor_strategy":"Lazy",
  "options": [],
  "trust_model":"Trusting"
},
{
  "description":"Considers every other agent to start on a good standing,
   ↪ if they observe a defection towards an agent with good standing the
   ↪ donor that defected is given a bad standing. Cooperates with agents
   ↪ they deem to have good standing, defects against those with bad
   ↪ standing. Trusts gossip from agents with a good standing, actively
   ↪ promotes own image with gossip",
  "donor_strategy":"Standing Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [],
  "trust_model":"Trusting"
},
{
  "description":"Considers every other agent to start on a good standing,
   ↪ if they observe a defection towards an agent with good standing the
   ↪ donor that defected is given a bad standing. Cooperates with agents
   ↪ they deem to have good standing, defects against those with bad
   ↪ standing. Trusts gossip from agents with a good standing, actively
   ↪ promotes positive image of good agents",
  "donor_strategy":"Standing Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [],
  "trust_model":"Trusting"
},
{
  "description":"Considers every other agent to start on a good standing,
   ↪ if they observe a defection towards an agent with good standing the
   ↪ donor that defected is given a bad standing. Cooperates with agents
   ↪ they deem to have good standing, defects against those with bad
   ↪ standing. Trusts gossip from agents with a good standing, actively
   ↪ promotes negative image of bad agents",
  "donor_strategy":"Standing Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [],
  "trust_model":"Trusting"
},
{
```

```
    "description":"Considers every other agent to start on a good standing,
    ↪  if they observe a defection towards an agent with good standing the
    ↪  donor that defected is given a bad standing. Cooperates with agents
    ↪  they deem to have good standing, defects against those with bad
    ↪  standing. Doesn't trust other agents gossip, does not actively
    ↪  gossip",
    "donor_strategy":"Standing Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [],
    "trust_model":"Distrusting"
  },
  {
    "description":"Considers every other agent to start on a good standing,
    ↪  if they observe a defection towards an agent with good standing the
    ↪  donor that defected is given a bad standing. Cooperates with agents
    ↪  they deem to have good standing, defects against those with bad
    ↪  standing. Doesn't trust other agents gossip, actively promotes own
    ↪  image with gossip",
    "donor_strategy":"Standing Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [],
    "trust_model":"Distrusting"
  },
  {
    "description":"Considers every other agent to start on a good standing,
    ↪  if they observe a defection towards an agent with good standing the
    ↪  donor that defected is given a bad standing. Cooperates with agents
    ↪  they deem to have good standing, defects against those with bad
    ↪  standing. Doesn't trust other agents gossip, actively promotes
    ↪  positive image of good agents",
    "donor_strategy":"Standing Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [],
    "trust_model":"Distrusting"
  },
  {
    "description":"Considers every other agent to start on a good standing,
    ↪  if they observe a defection towards an agent with good standing the
    ↪  donor that defected is given a bad standing. Cooperates with agents
    ↪  they deem to have good standing, defects against those with bad
    ↪  standing. Doesn't trust other agents gossip, actively promotes
    ↪  negative image of bad agents",
    "donor_strategy":"Standing Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [],
    "trust_model":"Distrusting"
  },
  {
    "description":"Randomly selects from actions it is capable of at each
    ↪  timepoint",
    "donor_strategy":"Random",
    "non_donor_strategy":"Random",
    "options": [],
    "trust_model":"Void"
```

```
},
{
  "description":"Considers every other agent to start on a good standing,
  ↪  if they observe a defection towards an agent with good standing the
  ↪  donor that defected is given a bad standing. Cooperates with agents
  ↪  they deem to have good standing, defects against those with bad
  ↪  standing. Trusts other agents gossip, does not actively gossip",
  "donor_strategy":"Standing Discriminator",
  "non_donor_strategy":"Lazy",
  "options": [],
  "trust_model":"Naive Trusting"
},
{
  "description":"Considers every other agent to start on a good standing,
  ↪  if they observe a defection towards an agent with good standing the
  ↪  donor that defected is given a bad standing. Cooperates with agents
  ↪  they deem to have good standing, defects against those with bad
  ↪  standing. Trusts other agents gossip, actively promotes own image
  ↪  with gossip",
  "donor_strategy":"Standing Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [],
  "trust_model":"Naive Trusting"
},
{
  "description":"Considers every other agent to start on a good standing,
  ↪  if they observe a defection towards an agent with good standing the
  ↪  donor that defected is given a bad standing. Cooperates with agents
  ↪  they deem to have good standing, defects against those with bad
  ↪  standing. Trusts other agents gossip, actively promotes positive
  ↪  image of good agents",
  "donor_strategy":"Standing Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [],
  "trust_model":"Naive Trusting"
},
{
  "description":"Considers every other agent to start on a good standing,
  ↪  if they observe a defection towards an agent with good standing the
  ↪  donor that defected is given a bad standing. Cooperates with agents
  ↪  they deem to have good standing, defects against those with bad
  ↪  standing. Trusts other agents gossip, actively promotes negative
  ↪  image of bad agents",
  "donor_strategy":"Standing Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-2 for the recipient they cooperate, else they defect. Agent never
  ↪  spreads gossip, but trusts others gossip (if the gossiper is of a
  ↪  value greater than K) ",
```

```
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [-2 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip and never trusts gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [-2 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip, but always trusts gossip no matter who is gossiping.
    ↪  ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [-2 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K) and
    ↪  trusts other trusted agents gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [-2 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K) but
    ↪  never trusts others gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [-2 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K). Always
    ↪  trusts gossip no matter who is gossiping. ",
```

```
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [-2 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=-2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip about those it trusts (value >= K) and
    ↪ trusts others trusted agents gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [-2 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=-2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip about those it trusts (value >= K), but
    ↪ never trusts others gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [-2 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=-2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip about those it trusts (value >= K). Always
    ↪ trusts gossip no matter who is gossiping. ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [-2 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=-2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip to promote themself, trusts gossip from
    ↪ agents it trusts (value >= K) ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [-2 ],
    "trust_model":"Trusting"
  },
  {
```

```
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=-2 for the recipient they cooperate, else they defect. Agent
   ↪  spreads positive gossip to promote themself, doesn't trust other
   ↪  agents gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [-2 ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=-2 for the recipient they cooperate, else they defect. Agent
   ↪  spreads positive gossip to promote themself. Always trusts gossip no
   ↪  matter who is gossiping. ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [-2 ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=-2 for the recipient they cooperate, else they defect. Agent never
   ↪  spreads gossip, but trusts others gossip (if the gossiper is of a
   ↪  value greater than K) Cooperation and defection against an agent
   ↪  using this strategy have a doubly large effect on the donors image
   ↪  score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Lazy",
  "options": [-2, "Personal Grievance" ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=-2 for the recipient they cooperate, else they defect. Agent never
   ↪  spreads gossip and never trusts gossip Cooperation and defection
   ↪  against an agent using this strategy have a doubly large effect on
   ↪  the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Lazy",
  "options": [-2, "Personal Grievance" ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=-2 for the recipient they cooperate, else they defect. Agent never
   ↪  spreads gossip, but always trusts gossip no matter who is gossiping.
   ↪  Cooperation and defection against an agent using this strategy have
   ↪  a doubly large effect on the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
```

```
  "non_donor_strategy":"Lazy",
  "options": [-2, "Personal Grievance" ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-2 for the recipient they cooperate, else they defect. Agent
  ↪  spreads negative gossip about those it distrusts (value < K) and
  ↪  trusts other trusted agents gossip Cooperation and defection against
  ↪  an agent using this strategy have a doubly large effect on the
  ↪  donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [-2, "Personal Grievance" ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-2 for the recipient they cooperate, else they defect. Agent
  ↪  spreads negative gossip about those it distrusts (value < K) but
  ↪  never trusts others gossip Cooperation and defection against an
  ↪  agent using this strategy have a doubly large effect on the donors
  ↪  image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [-2, "Personal Grievance" ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-2 for the recipient they cooperate, else they defect. Agent
  ↪  spreads negative gossip about those it distrusts (value < K). Always
  ↪  trusts gossip no matter who is gossiping. Cooperation and defection
  ↪  against an agent using this strategy have a doubly large effect on
  ↪  the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [-2, "Personal Grievance" ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-2 for the recipient they cooperate, else they defect. Agent
  ↪  spreads positive gossip about those it trusts (value >= K) and
  ↪  trusts others trusted agents gossip Cooperation and defection
  ↪  against an agent using this strategy have a doubly large effect on
  ↪  the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [-2, "Personal Grievance" ],
```

```
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip about those it trusts (value >= K), but
    ↪  never trusts others gossip Cooperation and defection against an
    ↪  agent using this strategy have a doubly large effect on the donors
    ↪  image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [-2, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip about those it trusts (value >= K). Always
    ↪  trusts gossip no matter who is gossiping. Cooperation and defection
    ↪  against an agent using this strategy have a doubly large effect on
    ↪  the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [-2, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself, trusts gossip from
    ↪  agents it trusts (value >= K) Cooperation and defection against an
    ↪  agent using this strategy have a doubly large effect on the donors
    ↪  image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [-2, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself, doesn't trust other
    ↪  agents gossip Cooperation and defection against an agent using this
    ↪  strategy have a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [-2, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself. Always trusts gossip no
    ↪  matter who is gossiping. Cooperation and defection against an agent
    ↪  using this strategy have a doubly large effect on the donors image
    ↪  score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [-2, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-1 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip, but trusts others gossip (if the gossiper is of a
    ↪  value greater than K) ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [-1 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-1 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip and never trusts gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [-1 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-1 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip, but always trusts gossip no matter who is gossiping.
    ↪  ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [-1 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-1 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K) and
    ↪  trusts other trusted agents gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [-1 ],
    "trust_model":"Trusting"
```

```
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent
  ↪  spreads negative gossip about those it distrusts (value < K) but
  ↪  never trusts others gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [-1 ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent
  ↪  spreads negative gossip about those it distrusts (value < K). Always
  ↪  trusts gossip no matter who is gossiping. ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [-1 ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent
  ↪  spreads positive gossip about those it trusts (value >= K) and
  ↪  trusts others trusted agents gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [-1 ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent
  ↪  spreads positive gossip about those it trusts (value >= K), but
  ↪  never trusts others gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [-1 ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent
  ↪  spreads positive gossip about those it trusts (value >= K). Always
  ↪  trusts gossip no matter who is gossiping. ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [-1 ],
```

```
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent
  ↪  spreads positive gossip to promote themself, trusts gossip from
  ↪  agents it trusts (value >= K) ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [-1 ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent
  ↪  spreads positive gossip to promote themself, doesn't trust other
  ↪  agents gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [-1 ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent
  ↪  spreads positive gossip to promote themself. Always trusts gossip no
  ↪  matter who is gossiping. ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [-1 ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪  interacting if the agent holds a value of greater than or equal to
  ↪  K=-1 for the recipient they cooperate, else they defect. Agent never
  ↪  spreads gossip, but trusts others gossip (if the gossiper is of a
  ↪  value greater than K) Cooperation and defection against an agent
  ↪  using this strategy have a doubly large effect on the donors image
  ↪  score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Lazy",
  "options": [-1, "Personal Grievance" ],
  "trust_model":"Trusting"
},
{
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-1 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip and never trusts gossip Cooperation and defection
    ↪  against an agent using this strategy have a doubly large effect on
    ↪  the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [-1, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-1 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip, but always trusts gossip no matter who is gossiping.
    ↪  Cooperation and defection against an agent using this strategy have
    ↪  a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [-1, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-1 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K) and
    ↪  trusts other trusted agents gossip Cooperation and defection against
    ↪  an agent using this strategy have a doubly large effect on the
    ↪  donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [-1, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=-1 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K) but
    ↪  never trusts others gossip Cooperation and defection against an
    ↪  agent using this strategy have a doubly large effect on the donors
    ↪  image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [-1, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
```

```
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=-1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads negative gossip about those it distrusts (value < K). Always
  ↪ trusts gossip no matter who is gossiping. Cooperation and defection
  ↪ against an agent using this strategy have a doubly large effect on
  ↪ the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [-1, "Personal Grievance" ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=-1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip about those it trusts (value >= K) and
  ↪ trusts others trusted agents gossip Cooperation and defection
  ↪ against an agent using this strategy have a doubly large effect on
  ↪ the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [-1, "Personal Grievance" ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=-1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip about those it trusts (value >= K), but
  ↪ never trusts others gossip Cooperation and defection against an
  ↪ agent using this strategy have a doubly large effect on the donors
  ↪ image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [-1, "Personal Grievance" ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=-1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip about those it trusts (value >= K). Always
  ↪ trusts gossip no matter who is gossiping. Cooperation and defection
  ↪ against an agent using this strategy have a doubly large effect on
  ↪ the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [-1, "Personal Grievance" ],
  "trust_model":"Naive Trusting"
},
{
```

```
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=-1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip to promote themself, trusts gossip from
  ↪ agents it trusts (value >= K) Cooperation and defection against an
  ↪ agent using this strategy have a doubly large effect on the donors
  ↪ image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [-1, "Personal Grievance" ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=-1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip to promote themself, doesn't trust other
  ↪ agents gossip Cooperation and defection against an agent using this
  ↪ strategy have a doubly large effect on the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [-1, "Personal Grievance" ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=-1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip to promote themself. Always trusts gossip no
  ↪ matter who is gossiping. Cooperation and defection against an agent
  ↪ using this strategy have a doubly large effect on the donors image
  ↪ score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [-1, "Personal Grievance" ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=0 for the recipient they cooperate, else they defect. Agent never
  ↪ spreads gossip, but trusts others gossip (if the gossiper is of a
  ↪ value greater than K) ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Lazy",
  "options": [0 ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=0 for the recipient they cooperate, else they defect. Agent never
  ↪ spreads gossip and never trusts gossip ",
  "donor_strategy":"Image Scoring Discriminator",
```

```
    "non_donor_strategy":"Lazy",
    "options": [0 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=0 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip, but always trusts gossip no matter who is gossiping.
    ↪  ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [0 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=0 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K) and
    ↪  trusts other trusted agents gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [0 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=0 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K) but
    ↪  never trusts others gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [0 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=0 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K). Always
    ↪  trusts gossip no matter who is gossiping. ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [0 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=0 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip about those it trusts (value >= K) and
    ↪  trusts others trusted agents gossip ",
```

```
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [0 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
     interacting if the agent holds a value of greater than or equal to
     K=0 for the recipient they cooperate, else they defect. Agent
     spreads positive gossip about those it trusts (value >= K), but
     never trusts others gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [0 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
     interacting if the agent holds a value of greater than or equal to
     K=0 for the recipient they cooperate, else they defect. Agent
     spreads positive gossip about those it trusts (value >= K). Always
     trusts gossip no matter who is gossiping. ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [0 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
     interacting if the agent holds a value of greater than or equal to
     K=0 for the recipient they cooperate, else they defect. Agent
     spreads positive gossip to promote themself, trusts gossip from
     agents it trusts (value >= K) ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [0 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
     interacting if the agent holds a value of greater than or equal to
     K=0 for the recipient they cooperate, else they defect. Agent
     spreads positive gossip to promote themself, doesn't trust other
     agents gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [0 ],
    "trust_model":"Distrusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=0 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip to promote themself. Always trusts gossip no
    ↪ matter who is gossiping. ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [0 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=0 for the recipient they cooperate, else they defect. Agent never
    ↪ spreads gossip, but trusts others gossip (if the gossiper is of a
    ↪ value greater than K) Cooperation and defection against an agent
    ↪ using this strategy have a doubly large effect on the donors image
    ↪ score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [0, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=0 for the recipient they cooperate, else they defect. Agent never
    ↪ spreads gossip and never trusts gossip Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [0, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=0 for the recipient they cooperate, else they defect. Agent never
    ↪ spreads gossip, but always trusts gossip no matter who is gossiping.
    ↪ Cooperation and defection against an agent using this strategy have
    ↪ a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [0, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=0 for the recipient they cooperate, else they defect. Agent
    ↪ spreads negative gossip about those it distrusts (value < K) and
    ↪ trusts other trusted agents gossip Cooperation and defection against
    ↪ an agent using this strategy have a doubly large effect on the
    ↪ donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [0, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=0 for the recipient they cooperate, else they defect. Agent
    ↪ spreads negative gossip about those it distrusts (value < K) but
    ↪ never trusts others gossip Cooperation and defection against an
    ↪ agent using this strategy have a doubly large effect on the donors
    ↪ image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [0, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=0 for the recipient they cooperate, else they defect. Agent
    ↪ spreads negative gossip about those it distrusts (value < K). Always
    ↪ trusts gossip no matter who is gossiping. Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [0, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=0 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip about those it trusts (value >= K) and
    ↪ trusts others trusted agents gossip Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [0, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
```

```
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=0 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip about those it trusts (value >= K), but
  ↪ never trusts others gossip Cooperation and defection against an
  ↪ agent using this strategy have a doubly large effect on the donors
  ↪ image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [0, "Personal Grievance" ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=0 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip about those it trusts (value >= K). Always
  ↪ trusts gossip no matter who is gossiping. Cooperation and defection
  ↪ against an agent using this strategy have a doubly large effect on
  ↪ the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [0, "Personal Grievance" ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=0 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip to promote themself, trusts gossip from
  ↪ agents it trusts (value >= K) Cooperation and defection against an
  ↪ agent using this strategy have a doubly large effect on the donors
  ↪ image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [0, "Personal Grievance" ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=0 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip to promote themself, doesn't trust other
  ↪ agents gossip Cooperation and defection against an agent using this
  ↪ strategy have a doubly large effect on the donors image score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [0, "Personal Grievance" ],
  "trust_model":"Distrusting"
},
{
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=0 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself. Always trusts gossip no
    ↪  matter who is gossiping. Cooperation and defection against an agent
    ↪  using this strategy have a doubly large effect on the donors image
    ↪  score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [0, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=1 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip, but trusts others gossip (if the gossiper is of a
    ↪  value greater than K) ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [1 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=1 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip and never trusts gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [1 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=1 for the recipient they cooperate, else they defect. Agent never
    ↪  spreads gossip, but always trusts gossip no matter who is gossiping.
    ↪  ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [1 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=1 for the recipient they cooperate, else they defect. Agent
    ↪  spreads negative gossip about those it distrusts (value < K) and
    ↪  trusts other trusted agents gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [1 ],
    "trust_model":"Trusting"
```

```
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=1 for the recipient they cooperate, else they defect. Agent
   ↪  spreads negative gossip about those it distrusts (value < K) but
   ↪  never trusts others gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [1 ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=1 for the recipient they cooperate, else they defect. Agent
   ↪  spreads negative gossip about those it distrusts (value < K). Always
   ↪  trusts gossip no matter who is gossiping. ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Negative",
  "options": [1 ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=1 for the recipient they cooperate, else they defect. Agent
   ↪  spreads positive gossip about those it trusts (value >= K) and
   ↪  trusts others trusted agents gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [1 ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=1 for the recipient they cooperate, else they defect. Agent
   ↪  spreads positive gossip about those it trusts (value >= K), but
   ↪  never trusts others gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [1 ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=1 for the recipient they cooperate, else they defect. Agent
   ↪  spreads positive gossip about those it trusts (value >= K). Always
   ↪  trusts gossip no matter who is gossiping. ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Spread Accurate Positive",
  "options": [1 ],
```

```
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip to promote themself, trusts gossip from
  ↪ agents it trusts (value >= K) ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [1 ],
  "trust_model":"Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip to promote themself, doesn't trust other
  ↪ agents gossip ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [1 ],
  "trust_model":"Distrusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=1 for the recipient they cooperate, else they defect. Agent
  ↪ spreads positive gossip to promote themself. Always trusts gossip no
  ↪ matter who is gossiping. ",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Promote Self",
  "options": [1 ],
  "trust_model":"Naive Trusting"
},
{
  "description":"Holds a value for each player starting on 0, when
  ↪ interacting if the agent holds a value of greater than or equal to
  ↪ K=1 for the recipient they cooperate, else they defect. Agent never
  ↪ spreads gossip, but trusts others gossip (if the gossiper is of a
  ↪ value greater than K) Cooperation and defection against an agent
  ↪ using this strategy have a doubly large effect on the donors image
  ↪ score.",
  "donor_strategy":"Image Scoring Discriminator",
  "non_donor_strategy":"Lazy",
  "options": [1, "Personal Grievance" ],
  "trust_model":"Trusting"
},
{
```

```
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=1 for the recipient they cooperate, else they defect. Agent never spreads gossip and never trusts gossip Cooperation and defection against an agent using this strategy have a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=1 for the recipient they cooperate, else they defect. Agent never spreads gossip, but always trusts gossip no matter who is gossiping. Cooperation and defection against an agent using this strategy have a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=1 for the recipient they cooperate, else they defect. Agent spreads negative gossip about those it distrusts (value < K) and trusts other trusted agents gossip Cooperation and defection against an agent using this strategy have a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=1 for the recipient they cooperate, else they defect. Agent spreads negative gossip about those it distrusts (value < K) but never trusts others gossip Cooperation and defection against an agent using this strategy have a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=1 for the recipient they cooperate, else they defect. Agent
    ↪ spreads negative gossip about those it distrusts (value < K). Always
    ↪ trusts gossip no matter who is gossiping. Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=1 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip about those it trusts (value >= K) and
    ↪ trusts others trusted agents gossip Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=1 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip about those it trusts (value >= K), but
    ↪ never trusts others gossip Cooperation and defection against an
    ↪ agent using this strategy have a doubly large effect on the donors
    ↪ image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=1 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip about those it trusts (value >= K). Always
    ↪ trusts gossip no matter who is gossiping. Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=1 for the recipient they cooperate, else they defect. Agent
   ↪  spreads positive gossip to promote themself, trusts gossip from
   ↪  agents it trusts (value >= K) Cooperation and defection against an
   ↪  agent using this strategy have a doubly large effect on the donors
   ↪  image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=1 for the recipient they cooperate, else they defect. Agent
   ↪  spreads positive gossip to promote themself, doesn't trust other
   ↪  agents gossip Cooperation and defection against an agent using this
   ↪  strategy have a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=1 for the recipient they cooperate, else they defect. Agent
   ↪  spreads positive gossip to promote themself. Always trusts gossip no
   ↪  matter who is gossiping. Cooperation and defection against an agent
   ↪  using this strategy have a doubly large effect on the donors image
   ↪  score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [1, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=2 for the recipient they cooperate, else they defect. Agent never
   ↪  spreads gossip, but trusts others gossip (if the gossiper is of a
   ↪  value greater than K) ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [2 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
   ↪  interacting if the agent holds a value of greater than or equal to
   ↪  K=2 for the recipient they cooperate, else they defect. Agent never
   ↪  spreads gossip and never trusts gossip ",
    "donor_strategy":"Image Scoring Discriminator",
```

```
    "non_donor_strategy":"Lazy",
    "options": [2 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=2 for the recipient they cooperate, else they defect. Agent never spreads gossip, but always trusts gossip no matter who is gossiping. ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [2 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=2 for the recipient they cooperate, else they defect. Agent spreads negative gossip about those it distrusts (value < K) and trusts other trusted agents gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [2 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=2 for the recipient they cooperate, else they defect. Agent spreads negative gossip about those it distrusts (value < K) but never trusts others gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [2 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=2 for the recipient they cooperate, else they defect. Agent spreads negative gossip about those it distrusts (value < K). Always trusts gossip no matter who is gossiping. ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [2 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when interacting if the agent holds a value of greater than or equal to K=2 for the recipient they cooperate, else they defect. Agent spreads positive gossip about those it trusts (value >= K) and trusts others trusted agents gossip ",
```

```
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [2 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip about those it trusts (value >= K), but
    ↪  never trusts others gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [2 ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip about those it trusts (value >= K). Always
    ↪  trusts gossip no matter who is gossiping. ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [2 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself, trusts gossip from
    ↪  agents it trusts (value >= K) ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [2 ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself, doesn't trust other
    ↪  agents gossip ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [2 ],
    "trust_model":"Distrusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip to promote themself. Always trusts gossip no
    ↪ matter who is gossiping. ",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [2 ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=2 for the recipient they cooperate, else they defect. Agent never
    ↪ spreads gossip, but trusts others gossip (if the gossiper is of a
    ↪ value greater than K) Cooperation and defection against an agent
    ↪ using this strategy have a doubly large effect on the donors image
    ↪ score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=2 for the recipient they cooperate, else they defect. Agent never
    ↪ spreads gossip and never trusts gossip Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=2 for the recipient they cooperate, else they defect. Agent never
    ↪ spreads gossip, but always trusts gossip no matter who is gossiping.
    ↪ Cooperation and defection against an agent using this strategy have
    ↪ a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Lazy",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads negative gossip about those it distrusts (value < K) and
    ↪ trusts other trusted agents gossip Cooperation and defection against
    ↪ an agent using this strategy have a doubly large effect on the
    ↪ donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads negative gossip about those it distrusts (value < K) but
    ↪ never trusts others gossip Cooperation and defection against an
    ↪ agent using this strategy have a doubly large effect on the donors
    ↪ image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads negative gossip about those it distrusts (value < K). Always
    ↪ trusts gossip no matter who is gossiping. Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Negative",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪ interacting if the agent holds a value of greater than or equal to
    ↪ K=2 for the recipient they cooperate, else they defect. Agent
    ↪ spreads positive gossip about those it trusts (value >= K) and
    ↪ trusts others trusted agents gossip Cooperation and defection
    ↪ against an agent using this strategy have a doubly large effect on
    ↪ the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip about those it trusts (value >= K), but
    ↪  never trusts others gossip Cooperation and defection against an
    ↪  agent using this strategy have a doubly large effect on the donors
    ↪  image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip about those it trusts (value >= K). Always
    ↪  trusts gossip no matter who is gossiping. Cooperation and defection
    ↪  against an agent using this strategy have a doubly large effect on
    ↪  the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Spread Accurate Positive",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself, trusts gossip from
    ↪  agents it trusts (value >= K) Cooperation and defection against an
    ↪  agent using this strategy have a doubly large effect on the donors
    ↪  image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Trusting"
  },
  {
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself, doesn't trust other
    ↪  agents gossip Cooperation and defection against an agent using this
    ↪  strategy have a doubly large effect on the donors image score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Distrusting"
  },
  {
```

```
    "description":"Holds a value for each player starting on 0, when
    ↪  interacting if the agent holds a value of greater than or equal to
    ↪  K=2 for the recipient they cooperate, else they defect. Agent
    ↪  spreads positive gossip to promote themself. Always trusts gossip no
    ↪  matter who is gossiping. Cooperation and defection against an agent
    ↪  using this strategy have a doubly large effect on the donors image
    ↪  score.",
    "donor_strategy":"Image Scoring Discriminator",
    "non_donor_strategy":"Promote Self",
    "options": [2, "Personal Grievance" ],
    "trust_model":"Naive Trusting"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-10 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Always commits to idle actions when not a
    ↪  donor. Weights for negative percepts are 2, no weight for positive
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [-10 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-10 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip about itself when not a
    ↪  donor. Weights for negative percepts are 2, no weight for positive
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [-10 ],
    "trust_model":"Strong Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-10 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip to trusted agents about
    ↪  trusted agents. Weights for negative percepts are 2, no weight for
    ↪  positive percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [-10 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-10 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads negative gossip to trusted agents about
    ↪  untrusted agents. Weights for negative percepts are 2, no weight for
    ↪  positive percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [-10 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-10 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Always commits to idle actions when not a
    ↪  donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [-10 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
     ↪ is affected by percepts when that agent is a donor or gossip about
     ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪ viewing a defecting against a trusted agent, +10*weight for positive
     ↪ gossip from a trusted source, -10*weight for negative gossip from a
     ↪ trusted source, +1*weight for positive gossip from an untrusted
     ↪ source and -1* weight for negative gossip from an untrusted source).
     ↪ Keeps a count of the percepts received about that agent n. If the
     ↪ v/n>-10 the agent the beliefs are about is trusted. Will cooperate
     ↪ with trusted agents. Spreads positive gossip about itself when not a
     ↪ donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [-10 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
     ↪ is affected by percepts when that agent is a donor or gossip about
     ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪ viewing a defecting against a trusted agent, +10*weight for positive
     ↪ gossip from a trusted source, -10*weight for negative gossip from a
     ↪ trusted source, +1*weight for positive gossip from an untrusted
     ↪ source and -1* weight for negative gossip from an untrusted source).
     ↪ Keeps a count of the percepts received about that agent n. If the
     ↪ v/n>-10 the agent the beliefs are about is trusted. Will cooperate
     ↪ with trusted agents. Spreads positive gossip to trusted agents about
     ↪ trusted agents. No weights for either positive or negative
     ↪ percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [-10 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
     ↪ is affected by percepts when that agent is a donor or gossip about
     ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪ viewing a defecting against a trusted agent, +10*weight for positive
     ↪ gossip from a trusted source, -10*weight for negative gossip from a
     ↪ trusted source, +1*weight for positive gossip from an untrusted
     ↪ source and -1* weight for negative gossip from an untrusted source).
     ↪ Keeps a count of the percepts received about that agent n. If the
     ↪ v/n>-10 the agent the beliefs are about is trusted. Will cooperate
     ↪ with trusted agents. Spreads negative gossip to trusted agents about
     ↪ untrusted agents. No weights for either positive or negative
     ↪ percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [-10 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>-10 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Always commits to idle actions when not a
    ↪ donor. Weights for positive percepts are 2, no weight for negative
    ↪ percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [-10 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>-10 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Spreads positive gossip about itself when not a
    ↪ donor. Weights for positive percepts are 2, no weight for negative
    ↪ percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [-10 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>-10 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Spreads positive gossip to trusted agents about
    ↪ trusted agents. Weights for positive percepts are 2, no weight for
    ↪ negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [-10 ],
    "trust_model":"Forgiving Reactor"
  },
  {
```

```
  "description":"Holds a veritability rating v for each other agent which
   ↪  is affected by percepts when that agent is a donor or gossip about
   ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪  viewing a defecting against a trusted agent, +10*weight for positive
   ↪  gossip from a trusted source, -10*weight for negative gossip from a
   ↪  trusted source, +1*weight for positive gossip from an untrusted
   ↪  source and -1* weight for negative gossip from an untrusted source).
   ↪  Keeps a count of the percepts received about that agent n. If the
   ↪  v/n>-10 the agent the beliefs are about is trusted. Will cooperate
   ↪  with trusted agents. Spreads negative gossip to trusted agents about
   ↪  untrusted agents. Weights for positive percepts are 2, no weight for
   ↪  negative percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Spread Negative Untrusted",
  "options": [-10 ],
  "trust_model":"Forgiving Reactor"
},
{
  "description":"Holds a veritability rating v for each other agent which
   ↪  is affected by percepts when that agent is a donor or gossip about
   ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪  viewing a defecting against a trusted agent, +10*weight for positive
   ↪  gossip from a trusted source, -10*weight for negative gossip from a
   ↪  trusted source, +1*weight for positive gossip from an untrusted
   ↪  source and -1* weight for negative gossip from an untrusted source).
   ↪  Keeps a count of the percepts received about that agent n. If the
   ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
   ↪  with trusted agents. Always commits to idle actions when not a
   ↪  donor. Weights for negative percepts are 2, no weight for positive
   ↪  percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Lazy",
  "options": [-5 ],
  "trust_model":"Strong Reactor"
},
{
  "description":"Holds a veritability rating v for each other agent which
   ↪  is affected by percepts when that agent is a donor or gossip about
   ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪  viewing a defecting against a trusted agent, +10*weight for positive
   ↪  gossip from a trusted source, -10*weight for negative gossip from a
   ↪  trusted source, +1*weight for positive gossip from an untrusted
   ↪  source and -1* weight for negative gossip from an untrusted source).
   ↪  Keeps a count of the percepts received about that agent n. If the
   ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
   ↪  with trusted agents. Spreads positive gossip about itself when not a
   ↪  donor. Weights for negative percepts are 2, no weight for positive
   ↪  percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Promote Self",
  "options": [-5 ],
  "trust_model":"Strong Reactor"
},
{
```

```
    "description":"Holds a veritability rating v for each other agent which is affected by percepts when that agent is a donor or gossip about that agent (+20*weight when viewing a cooperation, -20*weight when viewing a defecting against a trusted agent, +10*weight for positive gossip from a trusted source, -10*weight for negative gossip from a trusted source, +1*weight for positive gossip from an untrusted source and -1* weight for negative gossip from an untrusted source). Keeps a count of the percepts received about that agent n. If the v/n>-5 the agent the beliefs are about is trusted. Will cooperate with trusted agents. Spreads positive gossip to trusted agents about trusted agents. Weights for negative percepts are 2, no weight for positive percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [-5 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which is affected by percepts when that agent is a donor or gossip about that agent (+20*weight when viewing a cooperation, -20*weight when viewing a defecting against a trusted agent, +10*weight for positive gossip from a trusted source, -10*weight for negative gossip from a trusted source, +1*weight for positive gossip from an untrusted source and -1* weight for negative gossip from an untrusted source). Keeps a count of the percepts received about that agent n. If the v/n>-5 the agent the beliefs are about is trusted. Will cooperate with trusted agents. Spreads negative gossip to trusted agents about untrusted agents. Weights for negative percepts are 2, no weight for positive percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [-5 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which is affected by percepts when that agent is a donor or gossip about that agent (+20*weight when viewing a cooperation, -20*weight when viewing a defecting against a trusted agent, +10*weight for positive gossip from a trusted source, -10*weight for negative gossip from a trusted source, +1*weight for positive gossip from an untrusted source and -1* weight for negative gossip from an untrusted source). Keeps a count of the percepts received about that agent n. If the v/n>-5 the agent the beliefs are about is trusted. Will cooperate with trusted agents. Always commits to idle actions when not a donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [-5 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip about itself when not a
    ↪  donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [-5 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip to trusted agents about
    ↪  trusted agents. No weights for either positive or negative
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [-5 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads negative gossip to trusted agents about
    ↪  untrusted agents. No weights for either positive or negative
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [-5 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Always commits to idle actions when not a
    ↪  donor. Weights for positive percepts are 2, no weight for negative
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [-5 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip about itself when not a
    ↪  donor. Weights for positive percepts are 2, no weight for negative
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [-5 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip to trusted agents about
    ↪  trusted agents. Weights for positive percepts are 2, no weight for
    ↪  negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [-5 ],
    "trust_model":"Forgiving Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>-5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads negative gossip to trusted agents about
    ↪  untrusted agents. Weights for positive percepts are 2, no weight for
    ↪  negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [-5 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Always commits to idle actions when not a
    ↪  donor. Weights for negative percepts are 2, no weight for positive
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [0 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip about itself when not a
    ↪  donor. Weights for negative percepts are 2, no weight for positive
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [0 ],
    "trust_model":"Strong Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
     ↪ is affected by percepts when that agent is a donor or gossip about
     ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪ viewing a defecting against a trusted agent, +10*weight for positive
     ↪ gossip from a trusted source, -10*weight for negative gossip from a
     ↪ trusted source, +1*weight for positive gossip from an untrusted
     ↪ source and -1* weight for negative gossip from an untrusted source).
     ↪ Keeps a count of the percepts received about that agent n. If the
     ↪ v/n>0 the agent the beliefs are about is trusted. Will cooperate
     ↪ with trusted agents. Spreads positive gossip to trusted agents about
     ↪ trusted agents. Weights for negative percepts are 2, no weight for
     ↪ positive percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [0 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
     ↪ is affected by percepts when that agent is a donor or gossip about
     ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪ viewing a defecting against a trusted agent, +10*weight for positive
     ↪ gossip from a trusted source, -10*weight for negative gossip from a
     ↪ trusted source, +1*weight for positive gossip from an untrusted
     ↪ source and -1* weight for negative gossip from an untrusted source).
     ↪ Keeps a count of the percepts received about that agent n. If the
     ↪ v/n>0 the agent the beliefs are about is trusted. Will cooperate
     ↪ with trusted agents. Spreads negative gossip to trusted agents about
     ↪ untrusted agents. Weights for negative percepts are 2, no weight for
     ↪ positive percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [0 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
     ↪ is affected by percepts when that agent is a donor or gossip about
     ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪ viewing a defecting against a trusted agent, +10*weight for positive
     ↪ gossip from a trusted source, -10*weight for negative gossip from a
     ↪ trusted source, +1*weight for positive gossip from an untrusted
     ↪ source and -1* weight for negative gossip from an untrusted source).
     ↪ Keeps a count of the percepts received about that agent n. If the
     ↪ v/n>0 the agent the beliefs are about is trusted. Will cooperate
     ↪ with trusted agents. Always commits to idle actions when not a
     ↪ donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [0 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
     ↪  is affected by percepts when that agent is a donor or gossip about
     ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪  viewing a defecting against a trusted agent, +10*weight for positive
     ↪  gossip from a trusted source, -10*weight for negative gossip from a
     ↪  trusted source, +1*weight for positive gossip from an untrusted
     ↪  source and -1* weight for negative gossip from an untrusted source).
     ↪  Keeps a count of the percepts received about that agent n. If the
     ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
     ↪  with trusted agents. Spreads positive gossip about itself when not a
     ↪  donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [0 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
     ↪  is affected by percepts when that agent is a donor or gossip about
     ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪  viewing a defecting against a trusted agent, +10*weight for positive
     ↪  gossip from a trusted source, -10*weight for negative gossip from a
     ↪  trusted source, +1*weight for positive gossip from an untrusted
     ↪  source and -1* weight for negative gossip from an untrusted source).
     ↪  Keeps a count of the percepts received about that agent n. If the
     ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
     ↪  with trusted agents. Spreads positive gossip to trusted agents about
     ↪  trusted agents. No weights for either positive or negative
     ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [0 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
     ↪  is affected by percepts when that agent is a donor or gossip about
     ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪  viewing a defecting against a trusted agent, +10*weight for positive
     ↪  gossip from a trusted source, -10*weight for negative gossip from a
     ↪  trusted source, +1*weight for positive gossip from an untrusted
     ↪  source and -1* weight for negative gossip from an untrusted source).
     ↪  Keeps a count of the percepts received about that agent n. If the
     ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
     ↪  with trusted agents. Spreads negative gossip to trusted agents about
     ↪  untrusted agents. No weights for either positive or negative
     ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [0 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Always commits to idle actions when not a
    ↪  donor. Weights for positive percepts are 2, no weight for negative
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [0 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip about itself when not a
    ↪  donor. Weights for positive percepts are 2, no weight for negative
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [0 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip to trusted agents about
    ↪  trusted agents. Weights for positive percepts are 2, no weight for
    ↪  negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [0 ],
    "trust_model":"Forgiving Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>0 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads negative gossip to trusted agents about
    ↪  untrusted agents. Weights for positive percepts are 2, no weight for
    ↪  negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [0 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Always commits to idle actions when not a
    ↪  donor. Weights for negative percepts are 2, no weight for positive
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [5 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip about itself when not a
    ↪  donor. Weights for negative percepts are 2, no weight for positive
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [5 ],
    "trust_model":"Strong Reactor"
  },
  {
```

```
  "description":"Holds a veritability rating v for each other agent which
   ↪ is affected by percepts when that agent is a donor or gossip about
   ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪ viewing a defecting against a trusted agent, +10*weight for positive
   ↪ gossip from a trusted source, -10*weight for negative gossip from a
   ↪ trusted source, +1*weight for positive gossip from an untrusted
   ↪ source and -1* weight for negative gossip from an untrusted source).
   ↪ Keeps a count of the percepts received about that agent n. If the
   ↪ v/n>5 the agent the beliefs are about is trusted. Will cooperate
   ↪ with trusted agents. Spreads positive gossip to trusted agents about
   ↪ trusted agents. Weights for negative percepts are 2, no weight for
   ↪ positive percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Spread Positive Trusted",
  "options": [5 ],
  "trust_model":"Strong Reactor"
},
{
  "description":"Holds a veritability rating v for each other agent which
   ↪ is affected by percepts when that agent is a donor or gossip about
   ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪ viewing a defecting against a trusted agent, +10*weight for positive
   ↪ gossip from a trusted source, -10*weight for negative gossip from a
   ↪ trusted source, +1*weight for positive gossip from an untrusted
   ↪ source and -1* weight for negative gossip from an untrusted source).
   ↪ Keeps a count of the percepts received about that agent n. If the
   ↪ v/n>5 the agent the beliefs are about is trusted. Will cooperate
   ↪ with trusted agents. Spreads negative gossip to trusted agents about
   ↪ untrusted agents. Weights for negative percepts are 2, no weight for
   ↪ positive percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Spread Negative Untrusted",
  "options": [5 ],
  "trust_model":"Strong Reactor"
},
{
  "description":"Holds a veritability rating v for each other agent which
   ↪ is affected by percepts when that agent is a donor or gossip about
   ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪ viewing a defecting against a trusted agent, +10*weight for positive
   ↪ gossip from a trusted source, -10*weight for negative gossip from a
   ↪ trusted source, +1*weight for positive gossip from an untrusted
   ↪ source and -1* weight for negative gossip from an untrusted source).
   ↪ Keeps a count of the percepts received about that agent n. If the
   ↪ v/n>5 the agent the beliefs are about is trusted. Will cooperate
   ↪ with trusted agents. Always commits to idle actions when not a
   ↪ donor. No weights for either positive or negative percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Lazy",
  "options": [5 ],
  "trust_model":"Balanced Reactor"
},
{
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>5 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Spreads positive gossip about itself when not a
    ↪ donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [5 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>5 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Spreads positive gossip to trusted agents about
    ↪ trusted agents. No weights for either positive or negative
    ↪ percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [5 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>5 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Spreads negative gossip to trusted agents about
    ↪ untrusted agents. No weights for either positive or negative
    ↪ percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [5 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
     ↪  is affected by percepts when that agent is a donor or gossip about
     ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪  viewing a defecting against a trusted agent, +10*weight for positive
     ↪  gossip from a trusted source, -10*weight for negative gossip from a
     ↪  trusted source, +1*weight for positive gossip from an untrusted
     ↪  source and -1* weight for negative gossip from an untrusted source).
     ↪  Keeps a count of the percepts received about that agent n. If the
     ↪  v/n>5 the agent the beliefs are about is trusted. Will cooperate
     ↪  with trusted agents. Always commits to idle actions when not a
     ↪  donor. Weights for positive percepts are 2, no weight for negative
     ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [5 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
     ↪  is affected by percepts when that agent is a donor or gossip about
     ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪  viewing a defecting against a trusted agent, +10*weight for positive
     ↪  gossip from a trusted source, -10*weight for negative gossip from a
     ↪  trusted source, +1*weight for positive gossip from an untrusted
     ↪  source and -1* weight for negative gossip from an untrusted source).
     ↪  Keeps a count of the percepts received about that agent n. If the
     ↪  v/n>5 the agent the beliefs are about is trusted. Will cooperate
     ↪  with trusted agents. Spreads positive gossip about itself when not a
     ↪  donor. Weights for positive percepts are 2, no weight for negative
     ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [5 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
     ↪  is affected by percepts when that agent is a donor or gossip about
     ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
     ↪  viewing a defecting against a trusted agent, +10*weight for positive
     ↪  gossip from a trusted source, -10*weight for negative gossip from a
     ↪  trusted source, +1*weight for positive gossip from an untrusted
     ↪  source and -1* weight for negative gossip from an untrusted source).
     ↪  Keeps a count of the percepts received about that agent n. If the
     ↪  v/n>5 the agent the beliefs are about is trusted. Will cooperate
     ↪  with trusted agents. Spreads positive gossip to trusted agents about
     ↪  trusted agents. Weights for positive percepts are 2, no weight for
     ↪  negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [5 ],
    "trust_model":"Forgiving Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>5 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads negative gossip to trusted agents about
    ↪  untrusted agents. Weights for positive percepts are 2, no weight for
    ↪  negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [5 ],
    "trust_model":"Forgiving Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>10 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Always commits to idle actions when not a
    ↪  donor. Weights for negative percepts are 2, no weight for positive
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [10 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>10 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads positive gossip about itself when not a
    ↪  donor. Weights for negative percepts are 2, no weight for positive
    ↪  percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [10 ],
    "trust_model":"Strong Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which is affected by percepts when that agent is a donor or gossip about that agent (+20*weight when viewing a cooperation, -20*weight when viewing a defecting against a trusted agent, +10*weight for positive gossip from a trusted source, -10*weight for negative gossip from a trusted source, +1*weight for positive gossip from an untrusted source and -1* weight for negative gossip from an untrusted source). Keeps a count of the percepts received about that agent n. If the v/n>10 the agent the beliefs are about is trusted. Will cooperate with trusted agents. Spreads positive gossip to trusted agents about trusted agents. Weights for negative percepts are 2, no weight for positive percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [10 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which is affected by percepts when that agent is a donor or gossip about that agent (+20*weight when viewing a cooperation, -20*weight when viewing a defecting against a trusted agent, +10*weight for positive gossip from a trusted source, -10*weight for negative gossip from a trusted source, +1*weight for positive gossip from an untrusted source and -1* weight for negative gossip from an untrusted source). Keeps a count of the percepts received about that agent n. If the v/n>10 the agent the beliefs are about is trusted. Will cooperate with trusted agents. Spreads negative gossip to trusted agents about untrusted agents. Weights for negative percepts are 2, no weight for positive percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [10 ],
    "trust_model":"Strong Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which is affected by percepts when that agent is a donor or gossip about that agent (+20*weight when viewing a cooperation, -20*weight when viewing a defecting against a trusted agent, +10*weight for positive gossip from a trusted source, -10*weight for negative gossip from a trusted source, +1*weight for positive gossip from an untrusted source and -1* weight for negative gossip from an untrusted source). Keeps a count of the percepts received about that agent n. If the v/n>10 the agent the beliefs are about is trusted. Will cooperate with trusted agents. Always commits to idle actions when not a donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Lazy",
    "options": [10 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>10 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Spreads positive gossip about itself when not a
    ↪ donor. No weights for either positive or negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Promote Self",
    "options": [10 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>10 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Spreads positive gossip to trusted agents about
    ↪ trusted agents. No weights for either positive or negative
    ↪ percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Positive Trusted",
    "options": [10 ],
    "trust_model":"Balanced Reactor"
  },
  {
    "description":"Holds a veritability rating v for each other agent which
    ↪ is affected by percepts when that agent is a donor or gossip about
    ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪ viewing a defecting against a trusted agent, +10*weight for positive
    ↪ gossip from a trusted source, -10*weight for negative gossip from a
    ↪ trusted source, +1*weight for positive gossip from an untrusted
    ↪ source and -1* weight for negative gossip from an untrusted source).
    ↪ Keeps a count of the percepts received about that agent n. If the
    ↪ v/n>10 the agent the beliefs are about is trusted. Will cooperate
    ↪ with trusted agents. Spreads negative gossip to trusted agents about
    ↪ untrusted agents. No weights for either positive or negative
    ↪ percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [10 ],
    "trust_model":"Balanced Reactor"
  },
  {
```

```
  "description":"Holds a veritability rating v for each other agent which
   ↪ is affected by percepts when that agent is a donor or gossip about
   ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪ viewing a defecting against a trusted agent, +10*weight for positive
   ↪ gossip from a trusted source, -10*weight for negative gossip from a
   ↪ trusted source, +1*weight for positive gossip from an untrusted
   ↪ source and -1* weight for negative gossip from an untrusted source).
   ↪ Keeps a count of the percepts received about that agent n. If the
   ↪ v/n>10 the agent the beliefs are about is trusted. Will cooperate
   ↪ with trusted agents. Always commits to idle actions when not a
   ↪ donor. Weights for positive percepts are 2, no weight for negative
   ↪ percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Lazy",
  "options": [10 ],
  "trust_model":"Forgiving Reactor"
},
{
  "description":"Holds a veritability rating v for each other agent which
   ↪ is affected by percepts when that agent is a donor or gossip about
   ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪ viewing a defecting against a trusted agent, +10*weight for positive
   ↪ gossip from a trusted source, -10*weight for negative gossip from a
   ↪ trusted source, +1*weight for positive gossip from an untrusted
   ↪ source and -1* weight for negative gossip from an untrusted source).
   ↪ Keeps a count of the percepts received about that agent n. If the
   ↪ v/n>10 the agent the beliefs are about is trusted. Will cooperate
   ↪ with trusted agents. Spreads positive gossip about itself when not a
   ↪ donor. Weights for positive percepts are 2, no weight for negative
   ↪ percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Promote Self",
  "options": [10 ],
  "trust_model":"Forgiving Reactor"
},
{
  "description":"Holds a veritability rating v for each other agent which
   ↪ is affected by percepts when that agent is a donor or gossip about
   ↪ that agent (+20*weight when viewing a cooperation, -20*weight when
   ↪ viewing a defecting against a trusted agent, +10*weight for positive
   ↪ gossip from a trusted source, -10*weight for negative gossip from a
   ↪ trusted source, +1*weight for positive gossip from an untrusted
   ↪ source and -1* weight for negative gossip from an untrusted source).
   ↪ Keeps a count of the percepts received about that agent n. If the
   ↪ v/n>10 the agent the beliefs are about is trusted. Will cooperate
   ↪ with trusted agents. Spreads positive gossip to trusted agents about
   ↪ trusted agents. Weights for positive percepts are 2, no weight for
   ↪ negative percepts.",
  "donor_strategy":"Veritability Discerner",
  "non_donor_strategy":"Spread Positive Trusted",
  "options": [10 ],
  "trust_model":"Forgiving Reactor"
},
{
```

```
    "description":"Holds a veritability rating v for each other agent which
    ↪  is affected by percepts when that agent is a donor or gossip about
    ↪  that agent (+20*weight when viewing a cooperation, -20*weight when
    ↪  viewing a defecting against a trusted agent, +10*weight for positive
    ↪  gossip from a trusted source, -10*weight for negative gossip from a
    ↪  trusted source, +1*weight for positive gossip from an untrusted
    ↪  source and -1* weight for negative gossip from an untrusted source).
    ↪  Keeps a count of the percepts received about that agent n. If the
    ↪  v/n>10 the agent the beliefs are about is trusted. Will cooperate
    ↪  with trusted agents. Spreads negative gossip to trusted agents about
    ↪  untrusted agents. Weights for positive percepts are 2, no weight for
    ↪  negative percepts.",
    "donor_strategy":"Veritability Discerner",
    "non_donor_strategy":"Spread Negative Untrusted",
    "options": [10 ],
    "trust_model":"Forgiving Reactor"
  }
]
```