Final Year Project Report

Full Unit - Interim 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

December 6, 2018

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 Where other published and unpublished source materials have been used, these have been acknowledged.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:

Student Name: James King

Date of Submission:

Signature:

Table of Contents

Abstract

Chapter 1: **Aims, Objectives and Literary Survey**

Note: Documentation of the structure of my submission directory is in the appendix in chapter 5, and description of where to find each report and program can be found in the summary of completed work in chapter 3.

1.1 The Problem

To understand the aims and objectives of this project we must first grasp the history of the study of interaction between biological and computational agents (often known as intelligent agents in the field of computer science). It is important to distinguish between biological and computational agents as the study of natural selection and the evolution of cooperation in biological agents has a long and separate history from the study of intelligent agents in computer science.
The background to my project is covered in my first two reports (included in the appendix in chapter 5). I will be applying this knowledge in a more convenient format for the interim report here, my two reports provide a more in depth study on the topic of the evolution of cooperation and indirect reciprocity.
As explained in my first report (on the evolution of cooperation in relation to game-theory and different game-theoretic mechanisms to aid it), early Darwinian evolutionary theory was groundbreaking, however both opponents and even proponents of evolution such as Peter Kropotkin [8] (a Russian evolutionary scientist and political activist) found it fell short in explaining cooperative phenomena.
Cooperation is the idea of helping others at a detriment or cost to yourself. Of course, this concept is a key part of the natural world. For example, the act of parents supporting their children and vice-versa in human family groups. There are even interspecies examples such as that of the Meerkat and Drongo Bird [1].
The problematic question posed by the phenomena of cooperation is the idea from the theory evolution: the survival of the fittest [15]. This idea pushes individuals to compete for resources, so why does cooperation exist when there is a process actively pushing for competition? How did cooperation evolve in a world focused on competition?

1.2 Cooperation Aids

There have been a number of attempts to explain the phenomena of cooperation [8, 6, 5, 11]. In his book 'The Selfish Gene' [6] Richard Dawkins attempted to explain the idea that the real replicator in natural selection is the gene itself, which is self-interested in replication. As such, individuals are driven to working towards replication of the gene, not to improving our own personal fitness.
The idea of being driven towards the replication of our genes comes under the banner of Kinship Theory, highlighted by Axelrod and Hamilton in their paper [5] as one of the theories seeking to explain cooperation. Hamilton wrote a paper on Kinship Theory earlier in his academic career [7], explaining his version in which agents are encouraged to cooperate as they are interested in improving 'inclusive fitness' over their own individual fitness. Inclusive fitness refers to the fitness of both themselves and their relatives; the closer the relative the

more they have an effect on an agent's inclusive fitness.
The main mechanism highlighted by Axelrod and Hamilton [5] is, in popular culture, known as the game 'the iterated prisoner's dilemma'. The iterated prisoner's dilemma uses the idea of direct reciprocity to aid in the evolution of cooperation. This idea being: if I scratch your back, you'll scratch mine. In a round of the prisoner's dilemma, both agents simultaneously choose whether to cooperate or defect. A single prisoner's dilemma round provides no aid, but if the rounds are repeated, agents are encouraged to cooperate if historically there has been cooperation from the other agent with whom they're interacting. Another reason to cooperate is the payoff matrix given in table 1.1. Multiple rounds of both agents cooperating gives both higher social welfare (overall points accrued [16]) and greater individual payoff than multiple rounds of mutual defection.

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

Table 1.1: The payoff matrix in a typical iterated prisoner's dilemma game (such as Axelrod and Hamilton's). A=x, B=y where x denotes the payoff for A and y the payoff for B.

Direct reciprocity and Kinship Theory (or kin selection) are two of the five rules presented in Martin A. Nowak's paper 'Five Rules for the Evolution of Cooperation' [11]. The other three rules use different versions of reciprocity mechanisms. Network reciprocity is similar to the iterated prisoner's dilemma. Axelrod and Hamilton's approach [5] uses a round-robin tournament where all players interact with every other player. Network reciprocity is set aside from this style of direct reciprocity as it is played on a graph with players as the nodes, and the edges denoting who interacts with whom.
Another of the five rules, group selection, builds upon direct reciprocity. In group selection a population is separated out into smaller subpopulations. The players in these subpopulations only interact and reproduce within their subpopulation group. The group sizes fluctuate based on how fast agents reproduce inside them (dependent on the agents' fitness scores). If a group gets to a certain size, it has a chance of splitting, and when one group splits another group is removed. The effect of this is selection on two levels: individually and between groups, aiding the evolution of cooperation.
The last mechanism presented by Nowak and the one I am most interested in, is indirect reciprocity. The idea behind this mechanism is slightly less intuitive than direct reciprocity: if I scratch your back, hopefully later on someone else will remember and scratch my back. Actions in indirect reciprocity are different to direct reciprocity as they do not involve both agents acting. Only the donor of the donor-recipient interaction pair can choose to cooperate or defect, the payoffs of which are presented in table 1.2. The indirect reciprocity mechanism utilizes the idea of reputation. Cooperative acts may result in bettering the donor's reputation whereas defecting may result in a loss of reputation. How agents view the reputation of another, depends upon their implementation. Two important implementations described in my second report are the standing strategy [9] and using image scores [12]. Agents have the opportunity to base their decisions on these reputations. Sommerfeld *et al.* [14] put forward that the social action of gossip can help agents spread information in regards to agents reputations.

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

Table 1.2: The common payoff for most indirect reciprocity models

1.3 Theory Surrounding Intelligent Agents and Multi-Agent Systems

In my third report in the appendix chapter 5 I have covered the theory surrounding intelligent agents and multi-agent systems. Agents, as described by Russell and Norvig [13], are anything that perceives and acts upon its environment. Agents can be built using a number of architectures, some of which are seemingly more intelligent than others. Russell and Norvig highlight 5 such architectures ranging from agents which simply act in reflex to their environment to agents that actively learn from their percepts.
The idea that an agent is anything the perceives and acts upon it's environment could be considered a 'weak' notion of agency, even in comparison to Wooldridge and Jennings' 'weak' notion of agency [17]. Their paper stipulates that agents must be able to exhibit these 4 properties: autonomy, social ability, reactivity and proactivity.
Autonomy is considered to be operating without direct human or other agent intervention, with the agent having control over their internal state and actions. Social ability is defined as the ability of agents to interact with other agents (possibly biological) using an agent communication language. Reactivity is the ability to perceive their environment and act upon the changes in it. Proactivity makes Wooldridge and Jennings' [17] notion stronger than Russell and Norvig's [13] as it stipulates agents must not simply be reactive to their environment but are also able to exhibit goal-directed behaviour.
To be able to satisfy the properties laid out by Wooldridge and Jennings [17] and the definition of an agent from Russell and Norvig [13] agents need an environment to reside in. The possible environments are endless. An agent is built to work in a specific environment, the reason for an agents existence varies but often agents will attempt to complete specific tasks delegated to them.
Multi-agent systems are systems consisting of a combination of an environment and a number of agents perceiving and acting within it. Interactions can generally occur between both the environment and other agents in the system.

1.4 Relevance of The Problem and Indirect Reciprocity to Multi-Agent Systems

The societies which are simulated in the theories and experiments presented by Axelrod, Hamilton and Nowak [5, 11] are strikingly similar to multi-agent systems. The models include groups of players residing in an environment interacting with each other, leading naturally

to using these game-theoretic approaches to modelling interactions in multi-agent systems.
As agents act autonomously in a multi-agent system, why should one agent cooperate with another at a cost to themselves? From a developer's viewpoint, they can see the possibilities that if agents work together agents can accomplish so many brilliant things. From the agent's viewpoint though, why should that agent trust that if they cooperate, their good actions will not be taken advantage of?
This is the problem in section 1.1 reformulated for multi-agent systems. From a developer's perspective working on these sorts of systems, their goal is to encourage cooperation between agents but also not leave their agents or system of agents open to abuse. How can developers achieve this goal?
One way to approach this problem is to study how other environments have achieved this. One such environment that has achieved the evolution of cooperation, that we have identified earlier in this report, is the environment we live in, nature itself. Though nature encourages competition it has also led to cooperation between biological agents. So how can we learn from the processes nature employs?
We can apply the processes identified by game-theory and Kinship Theory to model multi-agent systems by implementing multi-agent systems that run these processes. Using a similar mechanism to indirect reciprocity Mui *et al.* [10] explored the role of reputation in the evolution of cooperation in multi-agent systems. The paper uses a number of different notions for reputation, one similar to Nowak's observer-based reputation, where an agent changes their view of another agent depending on the actions it views of the other agent. Their experiments found a large number of interactions were needed per generation for cooperative agents to dominate.
The mechanism/process I have chosen and outlined in my report on indirect reciprocity was chosen as I believe it applies well to multi-agent systems for a number of reasons. Firstly agents who interact across large networks such as the internet may never interact again, so it is imperative to have a strategy for deciding on how to act in these singular interactions. My models use of indirect reciprocity and reputation will allow us to study specifically these sort of 'sparse' interactions and the effectiveness of strategies in these interactions.
Another reason I believe my model matches multi-agent systems well is the social ability that it comes with. Nowak and Sigmund highlighted the importance of knowing another agents reputation to the success of the evolution of cooperation in an indirect reciprocity system [12]. They found that the probability of knowing another agents reputation had to be greater than the cost to benefit ratio of the action for cooperation to evolve.
In my system, agents who know each other will be able share information via gossip on other agents. The shared information increases the chances of knowing a certain agents reputation and in turn, helps defend against generally cooperative or 'nice' agents from being exploited. However, it also leads to another avenue of attack through spreading misinformation. This mechanism of information sharing could be used as a defence against exploitation, but it is important to understand how best to implement it and how effective it is before such an option is chosen.
One of the four properties laid out by Wooldridge and Jennings [17] is autonomy, which includes control over the agent's own internal state. The fact that in my model reputation is not a centrally managed concept but is part of the internal state, brings the system closer to a truly decentralised multi-agent system architecture.
Though reproduction itself doesn't, in fact, occur in multi-agent systems, it is likely that agent systems will be replaced and upgraded. The replacing systems will probably follow strategies that are known as effective by the developers. My reproduction mechanism aims to show how this replacement will take place, by assigning more likelihood for reproduction of an agent to occur if the strategy is successful. The reproduction may tend towards a defecting agent if they are more successful and vice-versa.
For further information on my model please refer to my second report on indirect reciprocity, strategies for agents and the development of a concrete model to implement in the appendix in chapter 5.

1.5 Project Specification, Aims and Objectives

From the sections above we can recognise that the study of indirect reciprocity and other mechanisms is important for the development of multi-agent systems and the prevention of exploitation by malicious agents. My project aim is to work on studying the model of indirect reciprocity I have defined and it's effectiveness in aiding the evolution of cooperation.
I will be doing this by implementing the model as a multi-agent system with a variety of strategies including the image scoring discriminator and standing strategy described in my second report. The specification for my system design has been described in my third report.
Further to the multi-agent system, I will be implementing a web application to run the environment. The web application will allow a user to select agents for their simulation and then run the simulation. The web application aims to act as an educational system on direct and indirect reciprocity (direct reciprocity has already been implemented using a preexisting library as a proof of concept for the web application).
After the simulation has ended a user will be redirected to an analysis of the simulation, where they will be able to examine the success of the model and strategies in regards to the evolution of cooperation. Analysis data will be held in a database for reuse later on. I will be using this system myself to examine the model and strategies effectiveness, alongside how the setting of parameters such as the number of interactions per generation are required for cooperation to evolve - as was examined by Mui *et al.* [10].

To sum up my aims:

- Implement the mechanism I have outlined in a multi-agent system
- Implement a number of strategies for use in the model
- Examine the relevance and success of the mechanism I have outlined in regards to the evolution of cooperation in multi-agent systems
- Examine the success of different strategies and trust models for agents in the system
- Explore how social ability/gossip can affect the evolution of cooperation in a multi-agent system
- Explore the various parameters that are important in the system in regards to what setting is required for cooperation to evolve

Chapter 2: **Planning and Timescale**

In my project plan (which can be found in my appendix in chapter 5) I decided on using 2-week sprints to manage the time at which milestones should be met. In sprint 1, I was to reach 3 milestones: 2 proof of concept applications and a report on the evolution of cooperation. Having done a lot of research over the summer on the evolution of cooperation and a tutorial for Flask I believed these reachable targets. Unfortunately, I overestimated how much I could do. As it can be seen in my diary in chapter 4 I did not complete this work in the first two weeks.

This was, unfortunately, a recurring theme; milestones were often not met. I feel that I underestimated the duration needed to reach milestones for 4 reasons. The first is that I work hard to make what I do the best standard I can, meaning I will often add time to a task to perfect it, which sets me back. Another is that I did not leave enough time for prototyping and designing my programs. The third is that I did not plan my tasks in enough detail, I simply attempted to hit a deadline each time. Lastly, I underestimated the amount of work other modules would require of me and the coursework I would need to complete for these.

This is not to say that I am completely off balance in the project. I have the milestones for term 1 completed, even if they were late. I have even implemented the gossip and onlooker features from the first two milestones of the second term. The issue with the implementation that I have completed so far is that it was often not completed in time for the milestone and as such, I ended up rushing some of the work. This led to a lack of extensive documentation and testing, and also a lack of detailed design work.

I am going to do two things to mitigate the issues I have had in this term in second term. One of which is to use the Christmas period to ensure documentation and testing is up to scratch before I begin any second term work.

Moving into the second term, I will also address these issues by creating a stricter plan. This includes more carefully planning design and prototyping time, and the processes required around implementation such as documentation and testing. Now I am versed more thoroughly in the project, I am in a better position to split milestones into far more fine-grained tasks. I shall again be organising my work in sprints but will be more strict on when these fine-grained tasks should be completed rather than having high-level milestones to work towards. Fine-grained tasks suit the scrum methodology more than high-level milestones, so I hope to be more successful using this.

Chapter 3: **Summary of Completed Work**

3.1 Proof of Concept

3.1.1 Proof Of Concepts

After finishing my project plan I embarked on the implementation of two proof of concept programs. Both involved learning technologies that are new to me. The first proof of concept I had already begun work on in the summer was implementing a web application to select agents and parameters to run matches and tournaments of direct reciprocity using the Axelrod-Python library to run the matches and tournaments in the back-end.
For this web app, I used Python and learnt the Flask microframework to develop the server-side alongside the SQLAlchemy ORM and SQLite for development. To learn the Flask microframework and its ecosystem I had taken the course 'The New and Improved Flask Mega Tutorial' [3] in preparation over the summer. This greatly improved my web development knowledge and skills, but I found I needed more knowledge for developing the front-end to select a dynamic number of agents. As such I took a crash course in React.js [2]. React has a number of features that Flask's template engine Jinja2 just can't accomplish. In my work I used the PyCharm IPSE and its various tools, such as the Flask framework support tools. Using the skills I accrued from these two courses and my knowledge of direct reciprocity, the Axelrod-Python library and Python itself, I successfully developed the web application. This application uses a wide array of technologies such as Redis, Bootstrap and others previously mentioned. I feel that it really helped develop my skills in web development, which will be beneficial to my development of a nice graphical user interface for running and analysis of tournaments and also the educational aspects of my final product. To run the web application see my notes in the README.md file in the top-level of my submitted directories.
My second proof of concept application was a small web service written in Prolog. Again the technologies used for this were new to me so I followed a tutorial 'Creating Web Applications in SWI-Prolog' [4] and continuously consulted the SWI-Prolog documentation. The purpose of this proof of concept application was to practice and demonstrate that I can create web services using Prolog before I went ahead and developed the agents service.
This second proof of concept was successful and I proved to myself that I could create an adequate web service. The outcomes of this were a knowledge that I immediately put to use in creating the agents service and also a guide on how I can do this using the JSON format and certain methods of error handling among other things.

3.2 Practical

Moving on from the proof of concept programs I started on design and implementation of the multi-agent system including the agents service and environment.

3.2.1 Agents Service and API

For SWI-Prolog there is a lack of an integrated project support environment or even an integrated development environment (except for in GNU EMACS which I do not have experience with). The tooling I have been using for the development of the agents service is the syntax

highlighter in sublime text, and for testing the experiment bed (provided as part of the multi-valued fluent cached event calculus) and the Prolog unit-test framework. I have used this to test belief revision in the file tests.pl under the directory AgentsService/src/mvfcec/src. To run this compile the tests.pl in SWI-Prolog (tested on v7.6.4) and run the predicate run_tests(percepts).
Further unit testing and documentation of the system is required to improve coverage and understandability of the code. However, testing of the functionality of the system using API testing has been extensive, the API testing code can be found in the directory NatureEngineWebApp/tests/agents_service_testing. I decided to write my own testing using the python requests library and unittest framework instead of a tool such as Postman as I can easily commit the tests using git without needing to pay and the python requests library and unittest framework are both extremely easy and nice to use. I have also used these previously so they required no extra learning.
The API has been documented using the RESTful API Markup Language RAML (based on YAML). RAML has tooling surrounding it to easily convert a .raml file into a user understandable format, such as the raml2html tool that I have used to generate the file main.html under the directory AgentsService/api_docs.
The API is designed to use the correct HTTP methods, uniform resource indicators to access and manipulate resources and to generally conform to RESTful API principles outlined at `https://restfulapi.net/`.
The agents service code for the web service side of the program has been based on my learning from the proof of concept Prolog service. The code that runs the multi-valued fluent cached event calculus (mvfcec) has come from my supervisor Kostas Stathis. I have built on top of mvfcec with initiates_at/3 predicates to define how an agent revises its beliefs using mvfcec (this is in the AgentsService/src/revise.pl file). Revision of beliefs has been implemented for when an agent is in an interaction, and when a standing strategy agent (both trusting and distrusting variants) receives gossip and interaction observation percepts.
A whole vertical slice has been implemented for defector, cooperator and standing strategy strategies. This means that they can receive percepts (refer to AgentsService/src/percepts.pl file), revise their beliefs and decide on actions at specific timepoints based on their knowledge at that point (refer to AgentsService/src/actions.pl).
The agents are built in the deductive reasoning style and using the concepts of beliefs, commitments, capabilities and logical theories as described in my system design report.
Further to this vertical slice it is also possible to query agents beliefs on when they were a donor, recipient, when two believe they were both in an interaction together and, if the agent is using the standing strategy, what they believe about the standing of another agent.
All this slice would be useless if users of the service were unable to find what strategies are implemented in the system. Users are able to retrieve the strategies from the /strategies URI as described in the main.html file. The implementation can be found in AgentsService/src/strategies.pl. The slice would also be useless if the service were unable to manage agent creation and assigning of strategies to them, this has been implemented in the agents.pl and communities.pl file. An agent is assigned to a community and generation, and also associated with a player id and strategy when created.

3.2.2 The Environment

The environment has been slightly tricky to implement because of the nature of the cycle step: send percepts to agents, get decisions from the agents and then execute these decisions (producing percepts). The cycle step easily introduces cyclic dependencies, high coupling and low cohesion in object oriented design.
I have been using umlet to design a class structure and execution sequence for the environment. Other tools in the development I have been using include the tools available in the Pycharm IPSE (linting, syntax highlighting, refactoring, built-in terminal, git support and

unittest framework support), reStructuredText docstrings and sphinx for documentation, and the python venv which I had already set up for the web application.
For implementation I have the logic to select a number of strategies and simulate a community. The community simluation creates a first generation and after a number of cycles steps reproduces into the next generation, repeating the reproduction into a new generation a specified number of times. Each generation is simulated by running, for a set number of cycles, the steps: perceive, decide and execute. The perceive step handles sending percepts to the agents (including the percepts generated from the last cycle step's actions and the donor-recipient pair for this step). The decide step then asks the agents for their commitment to an action at the given timepoint. After decision making, the execute step, if necessary, alters player fitness and generates percepts for the next cycle. The environment also creates communities, generations and players correctly in the agents service.
Though the environment was written using a TDD approach, I recognise that the test coverage is not enough and feel that there are likely to be a few latent bugs. I will be writing more testing for this soon, in order to reveal these bugs using a smoke testing strategy.
Furthermore I will be working on increasing cohesion in the class structure as currently the generation class is large and responsibilities could be separated out. I will also be creating a class above the community class in the hierarchy, that calls the simulate method on the community object and then uses visitors or other analysis classes to produce results and analysis on the simulation.

3.2.3 Tools, Techniques and Processes

All my work is in a git repository hosted on GitHub. I have been using git by creating branches for specific tasks, such as API testing, API documentation etc. and then merging back into the master when complete.

3.3 Theory

I have completed 3 reports with the first focusing on the problem at hand as described in section 1.1, the second focusing on a specific solution to this problem, and the third on designing a system to simulate the model outlined in the second.

3.3.1 Evolution of Cooperation Report

This report focuses on the problem at hand and the various theories and approaches taken to explain the evolution of cooperation. For the report, I reviewed past work surrounding the evolution of cooperation and theories to explain it. This review guided my development of a concrete model as delineated in my second report and served as a basis to extend my knowledge of the problem.
The report can be found in the directory ProjectReports/EvolCoop under the name EvolCoopReport.pdf.

3.3.2 Indirect Reciprocity Report

This report focused on examining the mechanism of indirect reciprocity in regards to being a mechanism aiding in the evolution of cooperation, its application to multi-agent systems and setting out a concrete model to implement.
I reviewed past work on indirect reciprocity and compared the various work. From this knowledge, I then specified the model I wished to implement and a few strategies I should implement and how should they be implemented.
The report can be found in the directory ProjectReports/IndirRec under the name IndirRec.pdf.

3.3.3 System Design Report

So far in my reports and background work, I had mostly focused on the problem and game-theoretic aspects of the project. This report was different and focused on the design of my multi-agent system using the model I had set out in the second report.
I worked towards matching an architecture and design of the system and agents to the model and strategies I had set out previously. This report was influenced by my background reading in the domain of intelligent agents and multi-agent systems, but also the background work on indirect reciprocity I have completed and the implementation I already had.
Due to my findings in the report, I made some revisions to my design decisions and implementation, causing me (alongside other reasons) to redesign the environment.
The report can be found in the directory ProjectReports/SysDesign under the name SysDesign.pdf.

Bibliography

[1] Africa, 2013. BBC television series created by the BBC Natural History Unit.

[2] Learn react - react crash course 2018 - react tutorial with examples — mosh. `https://www.youtube.com/watch?v=Ke90Tje7VS0`, 2018. Accessed: 05/12/2018.

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

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

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

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

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

[8] Peter Kropotkin. *Mutual aid: A Factor of Evolution*. 1902.

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

[10] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. Notions of reputation in multi-agents systems: A review. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, pages 280–287, New York, NY, USA, 2002. ACM.

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

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

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

[14] 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.

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

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

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

1

[1] A lot of my background theory work has been completed in my earlier reports so many of the references appear in those reports (see appendix)

Chapter 4: **Diary**

3:30 pm on September 28, 2018

A catchup of the happenings so far.
Early steps and planning:
Towards the end of term project details were released. I began my research into the topic by watching youtube videos (https://www.youtube.com/watch?v=BOvAbjfJ0x0) on the iterated prisoners dilemma, reading the evolution of cooperation by Robert Axelrod and William D. Hamilton, and playing the evolution of trust game online (https://ncase.me/trust/).
The discussions with Kostas were helpful in augmenting this research. These meetings helped me decide that Python was the best option to implement the project in and introduced me to the concepts of gossip and communities which I would not pin down until much later in the summer.
Research over the summer:
In the summer I researched into the topic of the iterated prisoners dilemma more and also how I should be using Python to implement this. I settled on using the Flask framework to create a web app to run these games, taking the Miguel Grinberg course Flask Mega-Tutorial to learn this. It was not until later that I would realise that using Flask would hamper my ability to interface between Python and Prolog to run the event (though I may still be able to use Prolog to implement strategies).
I continued with the idea that I would implement games, tournaments and a what I had believed to be a new idea called communities for this web application. For this I began development of the front-end and database of my web application that I dubbed The Nature Engine.
PyCon UK 2018 and a change of tack:
As a result of my research into python I decided to attend PyCon UK 2018 to expand my skills in it. At this conference, I joined in many workshops and listened to many talks on various parts of the python ecosystem. However the keynote speech on Monday was extremely interesting, the speaker is studying a PhD game theory at Cardiff University and spoke about the iterated prisoners dilemma and it impacts. It turns out that two other experts in the field were in attendance as well, all of which were core contributors to a library named Axelrod-Python. This library runs games, tournaments and much more.
With the realisation that most of my plan has already been implemented in a well tested mature library, I decided a change of tack was needed. One idea that the speaker had given me in answer to a question about whether there were models for community-level interaction was spatial tournaments. She had studied them for her MSc and wanted to point me in the right direction. This got me thinking that there are surely other models in game theory very closely related to this.
Post-PyCon research and the reformulation of my project:
It turns out these models do exist which I discovered in the paper Five rules for the evolution of cooperation by Martin A. Nowak and do not seem to be implemented in code at this current moment. There is still quite a decent amount of research about them. One that took my eye, which seemed to be very similar to what I had thought to be a new development of communities, known as group selection. This led to down to a reformulation of my ideas for The Nature Engine. To explore these alternative mechanisms noted in the five rules for the evolution of cooperation.
Return to university and meeting with Kostas:
With the past happenings in mind, I had a rough plan. I then had a meeting with Kostas to discuss my ideas with him. We talked at length about the different mechanisms and it soon became clear that I focused my project heavily on the game theory aspect of it and had moved away from the intelligent agents aspect. With this in mind, we bounced ideas around

about how to integrate my ideas within an intelligent agents project. We decided to explore a very interesting idea to both me and Kostas that involved creating a Prolog service to run the decision-making component of the agents. This would involve slimming down parts of the project to keep the scope small enough for the time I had. My interest in the onlookers and gossip aspect of indirect reciprocity and completely separate strategies to direct reciprocity lead me to focus on indirect reciprocity.

10:57 am on October 4, 2018

For this week of work, I have been focusing on finalising a solid project plan. Fleshing out the details of milestones, communicating my motivation for the project properly and exploring various mitigations I can take for foreseen risks. I feel as if I have got a bit bogged down with perfecting this (though I have begun a couple of pieces of proof of concept work) and I am now happy to have submitted.
This will now allow me to push on with the proof of concept work for the Prolog service and Flask web application, and also allow me to move onto my first report. I have a lot prepared for all 3 of these tasks and am ready to really get stuck into them, if I manage to get these finished early I may even be able to move onto the 2 more difficult reports.

9:43 am on October 12, 2018

My two proof of concept apps are coming along nicely. For the Flask web app proof of concept, I now have a front-end working with its Axelrod-Python match library counterpart in the server, and I have designed and implemented a database structure to record the outcome of games. I have also created unit tests for the functions that convert from the representation in the Axelrod-Python library for game results and stats to the database model I have created and then to a representation suitable for displaying on my front end.
For the Prolog web service proof of concept, I have worked out the handling of URI endpoints, HTTP errors, HTML in Prolog (which is super weird and I wont use it), sessions, running the server, building responses to requests, POST and GET methods, links and a few other bits. I am currently building them up using HTML to get data and display the results before I can use XML or JSON as these come later in the tutorial that I am using.
I am optimistic that working with a Prolog web service and Flask web app will work for my project quite nicely, but am concerned that I may be already getting behind as I said there is a report that I would hope to finish this week but simply havent had time. Luckily this report is linked to the next one I will be doing so I will do the reading for both at the same time and write at the same time, as I still need to finish the proof of concept web app.

5:16 pm on October 19, 2018

I have been adding to my Flask web application proof of concept application, I have taken tutorials on using React for front-end development and now have a web page where you can select any number of players and then send them to the server. The server runs these tournaments and all is working on that front. My next steps in this are to run this tournament in a Redis queue, design and implement a database to store the data gathered and create a front-end to display the analysis of the tournament.
I have been slowly adding to the Prolog web service as well as reading papers for my reports, though this progress has been slow due to my need to work on two assignments and apply to jobs. Soon I will begin on writing the reports and will hopefully have my proof of concept Prolog service finished up.

9:51 am on October 22, 2018

Proof of concept Flask application is finished! Id like to do some more testing on it there are some bugs Ive probably missed Id guess. But now a user can select a number between 3 and 50 players strategies and send them to the server, this then redirects the user to a waiting screen while the server runs the tournament in a Redis queue (so as not to block the server). In the waiting screen, the client pings a server route to see if the tournament has finished yet if it has it redirects to the URL sent by the server.
Now I need to reboot my work on the Prolog service, which I hope to have finished by the end of the week and continue my reading/start writing properly for my reports. This will be a real boost I was starting to wonder if I really had thrown myself in too deep so early, but I am far more optimistic now.

10:25 am on October 25, 2018

Having finished my Flask application I have moved onto writing the Prolog service proof of concept. I think that this is nearing completion, and will soon be moving onto writing the first two reports. This will allow me to get a better grasp on the model I wish to implement for indirect reciprocity. With a concrete model to implement, I will then be able to go about designing the architecture for the actual Prolog service and the environment in the Flask application.
While I have been writing this Prolog service I have also begun an outline for my final project report, so as to have sections to make notes in the whole way through the project. With these notes made throughout the project, I will have an easier time writing the final report as a whole. Next steps are to finish the proof of concept for the Prolog service and do more reading for the first two reports and begin their planning and writing.

1:40 pm on November 1, 2018

I have read up further on indirect reciprocity, the papers have been very interesting often arguing clearly against each other giving me ample information to define my own model. I have defined the model now, but it may need refining. I have gone over it at a very high level with Kostas and will refine it soon.
With this information, I will begin to design my environment and agent mind system which I already have a distinct plan for in my head. I have also begun writing another one of my reports to be ready soon. Hopefully, soon I will be able to get a good solid system architecture to implement and an idea of how to implement it. On this architecture and implementation, I will produce a report.

12:41 am on November 8, 2018

With my model defined I decided to start building an algorithm to run the environment and to gather an understanding of exactly what I needed from the agents service. This turned out to be a mistake as my design decisions led me away from the multi-agent system paradigm. This was talked about between me and Kostas, and I have decided that I should redesign to more closely match a multi-agent system as I would really like to be able to say that is what I have made at the end of this project. I have now got a rough redesign of my environment, which is also now partially implemented in order to gain a greater understanding of my design decisions.
I shall soon be finishing my design fully be developing on this code to add more functionality,

writing more in-depth testing for it and further documenting it. I also had a series of interesting lectures in my agents course over the last couple of weeks relating to agent design, which has given me much food for thought on how I should be implementing the agents service. Design decisions could move towards a reactive based agent model, deliberative based agent model or even a hybrid between the two. I am currently swaying more towards a deliberative model as this may suit Prolog and allow me to query and agents decision.

9:23 pm on November 15, 2018

With a lot of the design for the environment done and some implementation, I am currently working on testing the environment and the designing the agents service. The agents service is going to take a bit of time with deciding on a way to implement agents, I will most likely go with a deliberative over reactive approach due to Prologs brilliance in symbolic reasoning but how to implement this is a different matter. I have a number of alternatives to consider such as the BDI model, I will need to further look into these and decide for myself.

2:06 pm on November 22, 2018

This week I have been working on writing my evolution of cooperation report, covering a basic introduction to the theoretic side of my project (evolutionary dynamics, genetic algorithms, the iterated prisoners dilemma, game theory, reciprocation theory and kinship theory). I have finished all but the discussion and conclusion and then I need to go over it and fine comb the language and writing. I have also been working on understanding and using the mvfcec, it turns out this is quite a complex subject. However, Kostas has a nice experiment bed to run and test the changing of fluents and causing of other events.
Using this I have been writing Prolog predicates to control the changing of an agents beliefs depending upon their strategy. I have tested these for the standing strategy and they are working nicely, I have hit hitch with connecting this to the API which for some reason either does not correctly assert an event has occurred or the interpretation of the event doesnt correctly change the agents beliefs (though I have tested this in the test bed, there is a possibility that it doesnt recognise my other predicates correctly).

12:39 pm on November 27, 2018

With the functionality for my agents service approaching a usable point I have decided to change parts of the API to conform to a Rest API design, I have been documenting this using RAML and converting it to HTML to display nicely using raml2html. The testing for the API is now 75% complete and will hopefully soon be completed. Both my theory reports have been completed and checked by my supervisor, I will now be going over them to correct any mistakes and improve on them. At the same time, I am working on my viva presentation and interim review report.

4:32 pm on December 4, 2018

I have redesigned and reimplemented the environment using a TDD approach, which now works though further testing is required. I am planning to smoke test this to reveal issues. The Agent service now has 4 working strategies for this environment to use and the API is fully tested and documented. I have also completed the system design report and am continuing to work on the viva presentation and interim report. Once I have wrapped these

up I will begin work on improving the environment and analysis of a tournament.

Chapter 5: **Appendix**

My appendix contains 4 documents and a directory structure definition. The first document is my report on the evolution of cooperation, talked about in subsection 3.3.1. The second is my report on indirect reciprocity and my model to implement, talked about in subsection 3.3.2. The third is my report on system design, talked about in subsection 3.3.3. The last is my original project plan, referenced in 'Planning and Timescale' chapter 2. These all come after the directory structure definition.

5.0.1 Directory Structure Definition

```
FullUnit_1819_JamesKing
├── AgentsService
│   ├── api_docs
│   ├── src
│   │   ├── mvfcec
│   │   │   ├── domains
│   │   │   │   ├── percepts
│   │   │   │   │   └── exp0
│   │   │   ├── src
│   │   │   │   ├── compiler
│   │   │   │   └── lib
├── Design
├── NatureEngineWebApp
│   ├── app
│   │   ├── errors
│   │   ├── indir_rec
│   │   ├── main
│   │   ├── static/images
│   │   ├── templates
│   │   │   └── errors
│   ├── design
│   ├── migrations
│   ├── tests
│   │   └── agents_service_testing
│   ├── venv
├── ProjectReports
│   ├── 2ProjectPlan
│   ├── EvolCoop
│   ├── Final Report/latex
│   ├── IndirRec
│   ├── Interim Report/interim
│   ├── SysDesign
│   ├── Thesis
│   ├── VivaPres
├── PrologServiceProof
│   ├── src
│   │   └── assets
```

The top-level of my directory structure contains the interim report and a README.md file

describing my programs and how to run them. The top-level is then split up into 5 parts. Each part contains a different section of my project. The AgentsService directory contains all the programs and documentation relating to the agents service. API documentation is under api_docs and is viewable by putting the main.html file into any browser with javascript enabled. The src folder contains my work on the service and under the next directory mvfcec is my supervisor's work on mvfcec, the only part I have programmed in the mvfcec is testing for percepts (under mvfcec/src/tests.pl).
The second section is Design, this contains a couple of uml diagrams, viewable under umlet. These are not the only designs I have for the project as some are part of the reports and are not in this section.
The third section (NatureEngineWebApp) contains my work for the proof of concept web application, this is mostly under the app directory. which contains code for the database in models.py and the initialization for the flask app in __init__.py The errors directory inside the app directory contains the handlers for when 404 and 500 errors are thrown, to deliver a nice error page.
The indir_rec directory contains my code that runs the environment and the handlers for the web app related to indirect reciprocity, including both the business logic and testing. The main directory contains all the handlers and business logic for the rest of the web app. The static/images directory contains images for display in the web pages. The templates directory contains all the code involved in being sent to the client side using html, jinja2, css, react etc. In the templates directory the errors directory is specifically for the 404 and 500 error pages. Back in the NatureEngineWebApp directory we also have the tests directory which contains testing for the business logic surrounding direct reciprocity and the API testing for the agents service under the directory agents_service_testing.
The next section is ProjectReports. This contains all my reports and theory work for the project. EvolCoop contains my evolution of cooperation report, IndirRec contains my indirect reciprocity report, Interim Report/interim contains my interim report, SysDesign contains my system design report and VivaPres contains the presentation for my Viva.
The last section is PrologServiceProof, this contains my proof of concept application for a prolog service.

Report on the evolution of cooperation in relation to game-theory and different game-theoretic mechanisms to aid it

James King
Supervisor: Kostas Stathis

October 2018

Abstract

Since the early days of Darwinian evolutionary theory the phrase "Survival of the fittest" has become synonymous with much thinking on evolutionary dynamics. This idea has come along way since then but is still seen by many to promote selfish attitudes. However, we can see throughout nature that cooperation between biological agents is prevalent. So how did cooperation evolve and why has it flourished? This question fundamentally challenges what seemed concrete ideas about evolutionary dynamics, and has been approached from many angles. Game-theory is a branch of mathematics that has spawned many mathematical models of evolutionary dynamics in an attempt to solve the problem, some have been formulated programmatically to analyse the results. Study of the evolution of cooperation also has wider impacts in the world of computer science - namely on agent-based systems. A large component of agent-based system design is how interactions between agents works and how cooperation can be garnered within societies of agents. In this report, I shall explore past game-theoretic approaches to this problem. This exploration has helped me gather a deeper understanding of evolutionary dynamics, the reasoning behind these mathematical approaches to the problem of the evolution of cooperation and their application to the area of intelligent agents and multi-agent systems.

I. Introduction

Early Darwinians often focused on the struggle for survival as a means to explain the phenomena of evolution. This focus falls short on explaining the phenomena of altruistic behaviour, highlighted early on by Kropotkin [10]. The phenomena presented a problem for evolutionary biologists, how could it have evolved from an inherently selfish world?
Altruistic acts are actions where an individual puts others above themselves, these acts benefits recipients and are at a cost in some way to the actor. Examples of these acts and behaviours have been well documented and pervade both the natural world and human society.

In the seminal paper on the evolution of cooperation [1] Axelrod and Hamilton identify two theories proposed to solve the problem: kinship theory and reciprocation theory, focusing on the latter - particularly the Iterated Prisoner's Dilemma.
Here I will explore these mechanisms that attempt to explain altruistic behaviour and I will endeavour to relate them to multi-agent systems or even point out how they are inapplicable to this domain if appropriate.

II. Content and Knowledge

i. Kinship Theory

Kinship theory is an umbrella term for a number of models that attempt to solve the problem of the evolution of cooperation. The tie between these theories is that the individuals who choose to cooperate with each other are in some way 'kin'. The definition of what 'kin' is varies depending on the theory and the purpose for that theory.

Richard Dawkins popularized the idea of the 'selfish gene' [2]. This idea argues that as genes are the actual replicators, actors are hardwired to propagate the gene. This propagation does not only involve reproduction but also acts of cooperation to support and maintain others who share the gene. This is evident in many areas of nature especially in family groups, shown even recently in the BBC series Dynasties.

W.D. Hamilton supports a similar view [4]: that individuals with shared genes will work together to preserve those genes. He argues that to do this individuals don't work to add to their own fitness, but work to improve their 'inclusive fitness', which includes the fitness of other related individuals. His model converts the extent of the relatedness of an individual to a quantitative value using Wright's Coefficient of Relationship, which is shown in figure 1 on page 2. The model then uses this coefficient of relatedness to calculate inclusive fitness.

Hamilton finds that for that the value for the coefficient of relatedness must be greater than the cost-to-benefit ratio of the altruistic act for cooperation to evolve: $r > c/b$.

Axelrod and Hamilton [1] highlight that although the theory works well to explain cooperation between related individual it falls short in explaining the evolution of cooperation between unrelated individuals. This issue also makes it difficult to apply kinship theory as an aid to the evolution of cooperation in multi-agent systems. Kinship theory requires a reproductive drive and metrics for relatedness in order to motivate altruistic acts, both of which don't relate closely to multi-agent systems though in some applications could be present.

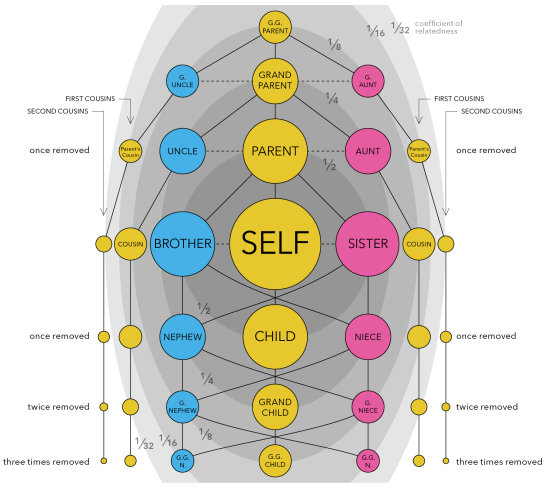

Figure 1: *Wright's coeffcient of relatedness by Citynoise - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=37723128*

ii. The Iterated Prisoner's Dilemma

The Iterated Prisoner's Dilemma is one such example of a game-theoretic model that uses direct reciprocity to attempt to solve the problem that the evolution of cooperation puts forward. Direct reciprocity is the idea: "If I cooperate now, you may cooperate later" [7]. The Dilemma is, in fact, a game played between two individuals. The game is made up of a number of repeated rounds, in each round the two players can choose between cooperating with or defecting against each other.

A payoff matrix is provided such as the one in figure 1 on page 3. These matrices provide a temptation to defect, but a reward if both players cooperate over multiple rounds. In a single round game it is mathematically best to defect, but in repeated rounds agents are encouraged to cooperate by a mutual gain in score. Nowak [7] reports that the cost-to-benefit ratio ratio of the altruistic act must be less than the probability of another encounter for cooperation to evolve: $w > c/b$.

The dilemma has been extended into tourna-

ments such as the one in Axelrod and Hamilton [1]. You can even run tournaments directly in Python using the Axelrod-Python library [3]. Tournaments can come in a variety of styles but the most popular is the round-robin tournament where every player plays a match of the iterated prisoner's dilemma against every other player. Players accumulate points throughout these games, based on the payoff matrix.
This has been even further extended to include a genetic algorithm to simulate evolution by reproducing players into the next generation. Multiple different algorithms have been used, one common one is the Moran Process (A stochastic process used to model evolution in a finite, unstructured population) available in the Axelrod-Python library.

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

Table 1: *The payoff matrix in a typical iterated prisoner's dilemma game (such as Axelrod and Hamilton's). A=x, B=y where x denotes the payoff for A and y the payoff for B.*

iii. Strategies

Players in the iterated prisoner's dilemma have to be able to decide whether to defect or cooperate in any given round against any given opponent. The player has access to the history of interaction between the two individuals to base their decision on - the simplest of strategies, however, don't use the history, one example being a pure defector.
Some more interesting strategies include the world famous tit-for-tat, grudger and Pavlov (win-stay, lose-shift [6]). The classical form of tit-for-tat begins by cooperating in the first round and then copying the other player's last move, this strategy is so effective because it gives the other player a chance to cooperate (so working well with cooperators) but still punishes a defecting player preventing them from taking advantage. There are many other versions of tit-for-tat, a good example being forgiving tit-for-tat which only defects if the other player defects for two turns in a row.
Grudger works by beginning to cooperate but if the other player defects the individual switches and defects for the rest of the time, this is not so effective as tit-for-tat because it never forgives the other player, preventing further cooperation.
Pavlov works by continuing to act the way it has in the previous interaction if the action was successful if it is not successful the strategy switches to the other action. Nowak and Sigmund [6] claim that Pavlov outperforms tit-for-tat as tit-for-tat is unsuccessful in non-deterministic environments (such as one that has a random chance that an action will not be what a player chooses). The real world is non-deterministic and so tit-for-tat doesn't generalise into it so well. Pavlov considers a success in a round to be: you both cooperate or you defect and the other player cooperated, and considers a failure to be: you both defect or you cooperate and the other player defects.

iv. Other Reciprocation Theories

Using reciprocity to aid in the evolution of cooperation has not been limited to the iterated prisoner's dilemma. Martin A. Nowak presents 3 other mechanisms that use reciprocation [7]: indirect reciprocity, network reciprocity (also known as spatial tournaments) and group selection. A graphical description of the 5 rules for the evolution of cooperation is provided in figure 2.
Indirect Reciprocity is a set of mechanisms that leverage the group mechanic of reputation. Each member of the set of mechanisms uses different metrics for reputation and the spread of information that influences reputation. Nowak and Sigmund's version uses the idea of image scoring [9]. An image score represents how well an agent is thought of, the

higher the better.
Which mechanism is more effective is the subject of extensive debate, in two separate papers both Leimar and Hammerstein [5], and Roberts [11] concluded that the standing strategy is more effective. In the standing strategy agents do not have an image score but have a standing which can be good or bad, everyone starts off with a good standing but if they defect against a player with good standing they are considered to have a bad standing.
Nowak [7] reports that the probability of knowing someone's reputation must be greater than the cost-to-benefit ratio of the altruistic act for cooperation to evolve: $q > c/b$. How information is spread throughout a population playing a game of indirect reciprocity thus becomes important.
Though not all agents can realistically be onlookers, another mechanism for the spread of information is through gossip [12]. Gossip can act as a mechanism for knowing the reputation of others, but can also muddy the water as to whether agents know the "correct" reputation of the recipient, or have they received "incorrect" information.
Network reciprocity (also known as spatial tournaments) uses direct reciprocity but not in a round-robin tournament. Spatial tournaments use a graph where nodes are agents and the edges represent channels of connection between agents [13], direct reciprocity games are then played between agents with connections. Nowak and May [8] found that groups of cooperators can work together to support each other and fend off invasion by defectors - depending on the structure of the graph.
Direct Reciprocity is also used in group selection. In this model the population is subdivided into smaller groups, in these smaller groups individuals interact with each other and reproduction occurs from the fitness they gain in these interactions.
Though offspring are reproduced into the same group, reproduction occurs on two levels: group sizes fluctuate due to reproduction, if a group gets to a certain size it may split in two and when a group divides another is eliminated to keep the number of groups fixed. Groups with faster reproduction do better than those with slower reproduction. Traulsen and Nowak found this favours the evolution of cooperation under certain conditions [14].

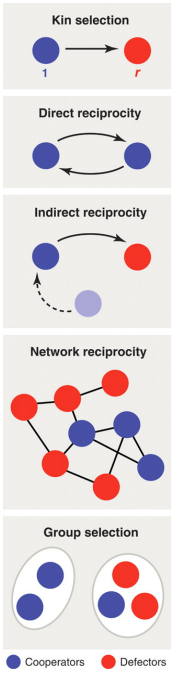

Figure 2: *A graphical representation of the five rules of cooperation from Nowak [7]*

III. Discussion and Conclusion

All these mechanisms to aid in the evolution of cooperation have extremely interesting properties and consequences in both biological and intelligent agents. Network Reciprocity can be used to represent a physical telecommunications network across which agents communicate, group selection is useful to model the spread of successful agent populations displacing others and kinship theory is useful to model interactions between different family groups in nature just to name a few applications.
With the internet being such a vast medium across which agents may interact it is very likely that agents may not have many repeated

meetings, so direct reciprocity is not always applicable. However, agents may often repeat interactions, so indirect reciprocity is not always applicable.
Due to the incredible interconnectedness of the internet, the graphs used by network reciprocity would have to be highly connected and this makes it less useful for modelling agent systems using the internet. Due to the lack of general reproductive drive and inherent relatedness between agents across the internet Kinship theory is unlikely to apply well to all but exceptional agent systems.
Thus as my project will be aiming to simulate a multi-agent system with agents decentralised across the internet a combination of indirect and direct reciprocity will likely be the most germane aid to the evolution of cooperation for my project.

References

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

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

[3] 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 MÃijller; Georgios Koutsovoulos; Areeb Ahmed. Axelrod: 4.3.0. `http://dx.doi.org/10.5281/zenodo.1405868`, September 2018.

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

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

[6] 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.

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

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

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

[10] Martin A. Nowak, Karl Sigmund, and Robert M. May. The arithmetics of mutual help. *Scientific American*, 272:76–86, 1995.

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

[12] 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.

[13] György Szabó and Gábor Fáth. Evolutionary games on graphs. *Physics Reports*, 446:96–216, 2007. Section: The structure of social graphs.

[14] 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.

Report on indirect reciprocity, strategies for agents and the development of a concrete model to implement

James King
Supervisor: Kostas Stathis

October 2018

Abstract

Indirect reciprocity is a mechanism that uses reciprocation theory to aid in the evolution of cooperation. Cooperation being an action taken by an individual that benefits another individual but at a cost to itself. Indirect reciprocity is a promising motivator for cooperation in societies with agents of higher intelligence levels and of greater sizes, such as human societies or even multi-agent systems. I plan to implement the mechanism programmatically, but there are many models and many possible additions to these models. In this report I will explore past approaches to indirect reciprocity, comparing and contrasting the variations proposed. The outcome of this exploration of approaches will be a concrete model to implement in this project.

I. Introduction

The evolution and preservation of cooperation has been a puzzle for evolutionary theorists for a long time. Many different approaches have been taken to give an explanation to cooperative phenomena, especially from the field of game theory. Often these approaches have come from the idea of reciprocity, where agents can grow mutually beneficial relationships through repeated interactions. The most popular mechanism of this being direct reciprocity where interactions are repeated between the same two individuals, and thus they may reciprocate directly with each other.
There is another game-theoretic mechanism known as indirect reciprocity, which works on the idea that nice agents will help those who help each other - "if I scratch your back, others will hopefully scratch mine" displayed graphically in figure 1. This means learning information about another player does not require direct interaction. A number of models have been proposed to run indirect reciprocity. It is these models I shall be describing and reviewing, before using them to formulate my own to implement in my project.

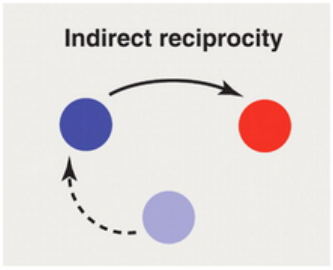

Figure 1: *The main idea of indirect reciprocity from Nowak [4]*

II. Review of Past Work

i. Nowak, Sigmund and Image Scoring

Nowak and Sigmund's model has a population structure of just one population of players per generation, the first generation of players are chosen at the start of an indirect reciprocity tournament [5]. A population then reproduces into the next generation of players (more on reproduction further down). From the population of a generation a set of pairs of players are chosen at random to interact. A pair is made up of a donor and recipient. The donor can choose to cooperate at a cost c, from this the recipient receives b points (b>c), or defect at no cost to the donor's fitness but the recipient also receives no points.
The model utilizes the idea of image scoring. An image score is comparable to the idea of a player's reputation. A player's image score increases to those who see them cooperate and decreases to those who see them defect. The actual image score is an integer, bounded between -5 and 5. One key part of the model is where these image scores are held.
In the first formulation of this all image scores are public, but later on, it is suggested that this model is unlikely to hold true in real life as individual's views on other individuals' reputations tend to differ. As an alternative, the idea of onlookers is suggested, where a set of onlookers are chosen from the generation for each interaction. In both models, image scores are increased by one for a cooperation as a donor or decreased by one for a defection as a donor, though in the second model only the onlookers, recipient and donor can add this to their image score for the donor. The use of onlookers does make cooperation harder to establish, as it is dependent on whether a player knows an accurate image score of the recipient.
The points accrued through interactions gives the fitness score of each player. The amount a player reproduces into the next generation is dependent on the fitness score. Mutation is also an option in this model, this phenomenon is when a player reproduces at random their offspring can be a different strategy. Mutation can lead to loops where defectors are invaded by discriminators and then discriminators are undermined by cooperators - allowing defectors to take hold again.
The two simplest strategies are pure defection and pure cooperation. A key strategy for the evolution of cooperation is the discriminator strategy as illustrated in figure 2. In fact, there is a baseline number of discriminators required for cooperation to evolve. This strategy is given a number k, if the image score of a player is greater than or equal to k a discriminator will cooperate with them. More interesting strategies take into account their own image score or theirs and the recipient's image score, these can aid in the evolution of cooperation too. In the absence of information on other players, these strategies can believe other players to have an image score of k with a certain probability.
The model is known to be dependent on the probability of knowing the image of another player. Cooperation can only be stable if $q > c/b$, q being the probability of knowing the image, c being the cost and b being the benefit.
Leimar and Hammerstein recognised the problems of genetic drift in this model [2]. Genetic drift being "a situation in which the frequency of a particular gene in a small population of living things changes without a known cause" [1]. Adopting an island population model to restrict genetic drift. Rather than one single population group, in the island population model, the group is divided up into g amount of groups each with population n. This model also uses a different reproductive strategy where relative reproductive success locally is calculated as a normalisation of the sum of the fitness of all strategies within the same group. For global expected reproductive success sum each strategy and normalize over the whole population. A new individual is then locally derived with probability p or globally derived with probability 1-p. The strategy for the individual produced is randomly generated with a distribution corresponding to the

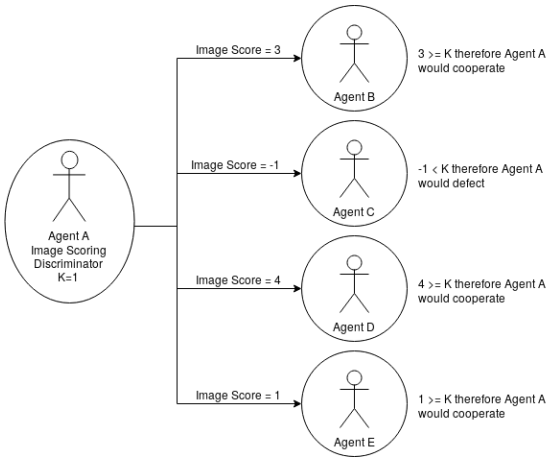

Figure 2: *How an image scoring discriminator strategy acts*

local or global reproductive success.

ii. Standing Strategy

The standing strategy was suggested by both Leimar and Hammerstein [2] and Roberts [7] to be superior to image scoring. Their models used the island population structure given above. Leimar and Hammerstein replaced image scoring with the standing strategy whereas Roberts mixed direct and indirect reciprocity (with both image scoring and the standing strategy).

The standing strategy as described by Milinski *et al.* [3] is that players should not just aim for a good fitness, but also good standing. Everyone starts on a good standing, but if a player is a donor to a good standing recipient and does not cooperate with them the donor loses their good standing.

The idea of this is that committing to bad actions against good individual is immoral, but it is not necessarily true that committing to good actions towards bad individual is moral and according to most game-theoretic models it does not produce stable cooperative society. This is seen as a benefit over image scoring as with image scoring it is not always in a players interest to punish those with a low image score, as they lose reputation themselves and in turn reduces the likelihood of them being the recipient of a cooperation. Whereas reputation is only lost in the standing strategy if the recipient is not of a good standing.

iii. Mixed Reciprocity Models

Gilbert Roberts puts forward mixed direct and indirect reciprocity models [7] using image scoring in one model similar to Nowak and Sigmund [5] and standing strategy. Both models use the island and reproductive systems Leimar and Hammerstein [2] utilized. Roberts put forward this notion due to his perception that indirect reciprocity alone is not a generalisable concept due to the close-knit nature of many societies.

Decisions can be made by agents in Roberts' model on the basis of either a reputation or an experience score. Reputation scores use image scoring in one model, differing from Nowak and Sigmund's as the score is within the range -1 to 1, initially at 0, decreasing by one for a defection and increasing by 1 for a cooperation. The standing strategy version assigns a -1 to a player when they defect against an individual or 1 to a player for anything else.

Experience scores are analogous to direct reciprocity for Roberts. When using image scoring, the experience scores between 2 players are initially 0 and go to -1 if they have defected or 1 if they have cooperated in the previous interaction. For standing strategy -1 is assigned only if the players has defected with a partner that has an experience score of 0 or 1 with them.

As you can see experience scores only reflect the previous interaction between the two individuals. This is a clear limitation as the model is seeking to describe a society made up of agents that have a higher intelligence.

Interestingly Roberts also considers the case where an agent has no resources to cooperate and thus must defect.

Steve Phelps [6] also recognises the same issue with indirect reciprocity as Gilbert Roberts - namely that remeeting in groups is likely, especially in smaller groups according to Phelps. Due to this Phelps suggests a mixed frame-

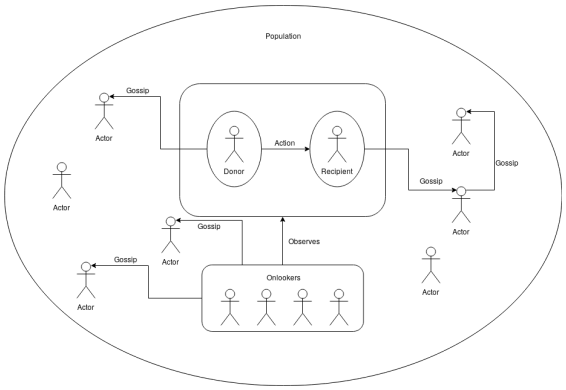

Figure 3: *The spread of information through a population using indirect reciprocity with gossip and onlookers*

work, to test how group sizes affect the use of direct or indirect reciprocity by agents.

iv. Gossip and Onlookers

It is recognised by Nowak and Sigmund [5] that it is unrealistic to expect all players to have directly observed a given interaction. Yet their answer - onlookers - hampers the evolution of cooperation, which relies on the probability that a player knows the image score of another player.
Sommerfeld *et al.* highlighted the importance of communication to the management of reputations. Gossip is a type of communication that is key part of societal mechanics, the structure of which is highlighted in figure 3. The findings of their experiment were that gossip needs to accurately reflect the interaction and the recipient of the gossip needs to correctly acted upon for gossip to be an adequate replacement for direct observation.

v. Comparison of Models

The aim of Roberts [7] is to compare the effectiveness of indirect reciprocity models and direct reciprocity. Roberts' results support Nowak and Sigmund's 1998 result that image scoring can support cooperation is robust even when criticisms relating to genetic drift and errors are taken into consideration.
However, it is highlighted that image scoring does not take into account who the donor is defecting against, whereas standing strategy does. Due to this image scoring suffers from an inability to distinguish between pure defectors and those who would cooperate with a cooperator. Standing strategy gives information both about how agents have interacted and the context of this action, an advantage over image scoring.
Counter to standing strategies perceived benefits it is seen that players are more susceptible to errors in perception, as standing falls faster than image score. Roberts also criticizes the standing strategy as not being a representation of the evolution of cooperation, as good standing can't have been established before the game had begun.
Roberts also points out that cooperation is only worth it when that cooperation is improving your chance of being cooperated with. This could be a counterpoint to the effectiveness of gossip as an alternative to direct observation. In gossip, there is a possibility of interactions being distorted by incorrect gossip. It could be interesting to see how the spread of misinformation could permeate society.

III. A Concrete Model

i. Specification of the Model

Both image scoring and standing strategy are good models for indirect reciprocity. They allow an agent to interpret a reputation and act as they see fit. However, the reputation of each player being universally viewable is unrealistic. The idea of onlookers seems closer to real life but is restrictive to the evolution of cooperation. The addition of gossip from these onlookers could be a sufficient alternative to direct observation, but gossip can be distorted by players wishing to influence proceedings against cooperation.
Gossip complicates a model because there is a need to interpret the information gossiped by another player. This also highlights an issue with the models that use purely standing

strategy or image scoring: reputation is not a centralised system with everyone interpreting events the same way. Reputation has a very subjective nature, especially when gossip is introduced. Due to the subjectivity of perception about an agents reputation, I do not believe that one specific model for all agents can be considered appropriate. How a player views another players reputation must be up to their inside workings.
This requires an agent to have a view of past events and also a way to interpret these events in order to make decisions about another player. A model that uses this decentralised judgement on past events easily becomes a hybrid indirect and direct reciprocity game, as agents can have their own way to interpret interactions.
For an agent to interpret gossip, the gossip must be comprehensible by them and thus should have a given structure. In our model, gossip shall be split into two categories: positive and negative. Gossip will also involve 3 agents, the agent gossiping, the recipient of the gossip, and the gossip that the agent is about (who shall not be able to view that the gossip has taken place). Gossip is intended to change how the recipient of the gossip views the agent the gossip is about - for example, if agent 2 trusts gossip from agent 3 that is negative about agent 1 and agent 2 is using the standing strategy, then agent 2 will revise their beliefs about agent 1 to have a bad standing.
The structure of the game will be as follows. There is one singular population (this is to not overly complicate the model, though changing this to an island population model such as in Leimar and Hammerstein [2]), a population is replaced by their offspring after the conclusion of a generation. A generation is made up of a number of interactions, all at a distinct time point (the number of interactions per generation is a variable).
In each time point, each agent will perceive their environment and then decide what action to take, this action may be as a donor in an interaction or maybe a piece of gossip. Each interaction shall be made up of a donor-recipient pair, chosen at random from the population. For each interaction, a subset of the population should be chosen as direct observers (being onlookers and the donor-recipient pair).
Finally, reproduction occurs at the end of every generation and will be relative to each players fitness. The fitness of a player represents how successful they are, fitness is altered in each interaction if the donor cooperates they lose a fitness point but the recipient gains two, if the donor defects they both gain nothing and lose nothing see table 1. Reproduction represents the replacement of agents with newer agents in a network. The new agents developed for a system would likely be influenced by successful strategies in past generations thus the reproduction in our system represents the trend towards certain successful strategies in a multi-agent system.

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

Table 1: *The payoff for my indirect reciprocity model*

The group size remains the same over generations in order to measure the success of different strategies in certain group sizes, so in a new generation individuals are generated and assigned strategies. The chance an individual will be a certain strategy is worked out by summing the fitness of all individuals of that strategy and dividing it by the fitness of the overall group. Reproduction in this way has been chosen to represent the way that new agents will tend to be developed based on past successful strategies. The user will be able to select if mutation is active or not, if it is there will be a 1% chance that a new individuals strategy will be chosen completely at random.

ii. Specification of Agent Strategies

At each time point in the generation that an agent is a part of, the agent may perceive new information about interactions and gossip that has occurred in the game by way of an environment sending new percepts to agents. The agent will then be asked to decide on what they want to do at that same time point.
The percepts that the agent can receive is that they are a donor at this time point, that they are a recipient at this time point, gossip from another agent and observation of an interaction at the last time point.
Based on the percepts an agent will be able to revise their beliefs about other agents and the environment. For all agents, if they perceive that they are a donor in a donor-recipient pair they revise their beliefs to know that they are a donor at this time point. However, the revision of beliefs is generally strategy dependent, for example, in an agent using the standing strategy if they observe an interaction where a donor defects against an agent that you consider to have good standing the observing agent will revise their beliefs to give a bad standing to the donor.
Agents will have a theory on the best action to take based on their beliefs, if they have not yet received percepts they may not have any beliefs yet, so will have to have a default action. Based on the agents' beliefs and strategy they will have a theory encoded on how is best to act at each time point. The theory encoded is dependent on the strategy they are using, for example, if a discriminator (with k=2) has perceived they are a donor and that the recipient of their action has an image score of 1 the theory will say it is best to defect, as the image score is lower than their value of k. It will also be important to encode a theory on how to act if an agent is not a donor (how do they spread gossip? Do they remain idle?).
Of course, some agents will not care about percepts on other agents and will purely decide according to a plan such as always defect, or always cooperate. However, there needs to be a distinction on how an agent reacts to gossip and how they react to an observation. I propose two classes of interpretation: trusting and distrusting. Distrusting agents will never take into account gossip, however, trusting agents will have a trust model, for example trusting agents using the standing strategy will only believe gossip and allow it to change their view of another agent's standing if the agent gossiping to them has a good standing from their point of view. Another example of this is if an agent is using a distrusting standing strategy, negative gossip will not change their beliefs on the standing of a certain agent.
Using this design, strategies that will be implemented include: trusting and distrusting 'neutral' standing discriminator (similar to Roberts' version), trusting and distrusting image scoring discriminator (with varying values of k), cooperator, defector. This is not an exhaustive list and I will be including more strategies based on 'neutral' standing and image scoring. These strategies only describe how to act as a donor based on beliefs on another agent. An agent will also need a strategy of how to act when it is not a donor. Some strategies may include: stay idle, always gossip accurate information to random players (if none then stay idle), always gossip inaccurate information to random people, always gossip negative or positive information to random players, randomly gossip positively or negatively. More advanced agents will have a better-developed strategy for when they are not a donor.
In conclusion, basic agents will need a way to take in percepts, they will then need a strategy to act if they are a donor and if they are not a donor. More advanced agents will use percepts to revise their beliefs and then use their beliefs to decide on how to act at any given time point. Even more advanced agents than these will develop beliefs or act on beliefs in more interesting ways than discriminating based on their view of whether an agent has a good standing or whether their view of an agents image score is above a certain value. Examples of these more advanced strategies are: basing their action on both their own and the recipients standing or image score or using a

reinforcement algorithm to decide an action based on a belief.

IV. Discussion and Conclusion

Multi-agent systems require agents to be autonomous, they should not be working on centralised beliefs such as is used by Nowak and Sigmund [5] at first. My model not only removes those centralised beliefs and makes them part of the agent's inner workings but also makes the whole strategy they use part of the agent's inner workings.
The lack of individuality is where the work on indirect reciprocity I have seen falls short, because of the nature of indirect reciprocity being a community-driven mechanism the implementors see beliefs as community held and fail to realise that the beliefs may be community driven, but are actually individually held.
This model is also very open to extension. Extensions on the population model (maybe using the island idea), the strategies (different interpretations of gossip and observations, strategies for the spread of gossip and strategies for actions as a donor) and mixing in indirect reciprocity by viewing observations where the agent is a recipient differently.

References

[1] Definition of "genetic drift" - english dictionary. https://dictionary.cambridge.org/us/dictionary/english/genetic-drift, 2018. Accessed: 27/11/2018.

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

[3] 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.

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

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

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

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

System design report: Multi-Agent System

James King
Supervisor: Kostas Stathis

October 2018

Abstract

Architectures and designs for intelligent agents and the environment they reside come in many different variations. For my multi-agent system I need to make some decisions to match a design for agents and their environment to the model I laid out in my report on indirect reciprocity. To do this I will consider different agent architectures, features and designs, considering how closely they match the model before laying out an architecture of my own. I shall also consider the properties of the environment the agents reside in before laying out a concrete execution for a game of indirect reciprocity. From these two component definitions I will formulate a method of communication between the environment and the agents service.

I. Introduction

One of the aims of my project is to produce a multi-agent system to play games of indirect reciprocity, the model for which I have defined in my second report: "Report on indirect reciprocity, strategies for agents and the development of a concrete model to implement". For this I have designed a system that includes two main components: The environment and a web service to host agent's decision making components (see figure 1).
I have already decided on a web service to host agent's minds for a number of reasons. One of the reasons being that in interactions across a network agents provide an API to work with (much like the API I will be providing). Meaning that it is good practice for me to consider how an API for an agent can be designed and deployed. Another reason is the difficulty interfacing between prolog and python and finally that a key use case for agents is as part of distributed systems, so using a web service to deploy them brings me closer to a real use case.
In my project, I will be hosting the environment in my Flask web application, but, the aim is for the environment to be used as if it is a library, as long as it is connected to an application/service that runs agent's decision making component. As such I will not be addressing how the web application hosting the environment, but the environment itself and the agents service.

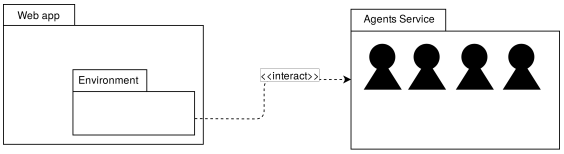

Figure 1: *UML Package diagram to display the system components*

II. Content and Knowledge

This section I will split up into 3 parts: one for each of the two main components - the environment and agents minds service and another for the communication between these two components.

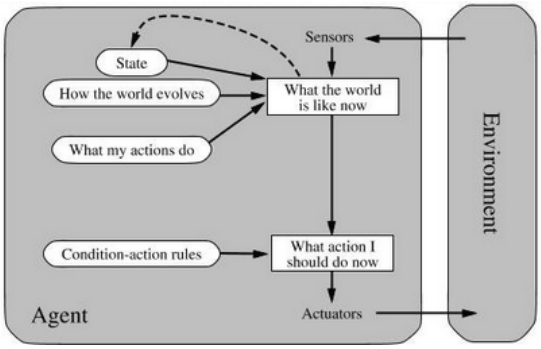

Figure 2: *A model-based reflex agent from Russell and Norvig [3].*

i. Agents

There are many different definitions of what properties a system needs in order to be considered an agent. One definition containing 4 properties was proposed by Wooldridge and Jennings [6]: autonomy, reactivity, proactivity and social ability.
Autonomy being considered the ability to operate without direct human intervention and having control over the agent's own actions and internal state. Reactivity is defined as agents perceiving their environment and responding to changes in it in a timely fashion. Proactivity is given as the ability to exhibit goal-driven behaviour, rather than just reacting to its environment. Lastly, social ability is interaction between agents and/or humans via an ACL (agent communication language).
This notion of using the 4 properties to define an agent is considered a 'weak' notion. Wooldridge and Jennings suggest that stronger notions of agency include more human-like features. One such human-like feature that exists in game-theoretic models is the idea of memory or an internal state that the agent can act on. An internal state is used in Russell and Norvig's model-based reflex agent which takes percepts, modifies it's internal state and uses the internal state and a model of the environment as input to condition-action rules as seen in figure 2. The model-based reflex agent architecture can be applied as an example to a game-theoretic agent strategy known as an image scoring discriminator, as laid out by Nowak and Sigmund [2]. An image scoring discriminator's condition-action rules state: if the discriminator's internal state is greater than or equal to k (a variable set at the initialization of the agent) it is best to cooperate, else the agent will defect. The image scores are held in an agent's internal state and may change depending on the percepts the image scoring discriminator agent receives.
Some game-theoretic strategies don't use memory or internal state and just seek to provide a theory to act on. Though this does not qualify them as an agent according to Wooldridge and Jennings [6] they are still relevant strategies for game-theory.
These indirect reciprocity strategies bear a remarkable resemblance to both model-based reflex agents and how theories are encoded in deductive reasoning agents. Deductive reasoning agents use theories to encode how it is best to act under any given situation [5]. Agents who follow the image score discriminator strategy could have a theory encoded in them such as:

$$
\begin{aligned}
&interaction(me, Recipient, time) \quad \wedge \\
&image\_score(me, Recipient, Score, time) \quad \wedge \\
&Score \geq k \rightarrow do(cooperate(Recipient)) \\
\\
&interaction(me, Recipient, time) \quad \wedge \\
&image\_score(me, Recipient, Score, time) \quad \wedge \\
&Score < k \rightarrow do(defect(Recipient)) \\
\\
&\neg interaction(me, \_, time) \rightarrow do(idle)
\end{aligned}
$$

Part of the deductive reasoning agents method of implementing agents is the logical database that includes information on the current state of the world. In the example above this would possibly include logical data such as:

$$
\begin{aligned}
&image\_score(agent1, agent2, 2, 9). \\
&interaction(agent1, agent2, 6).
\end{aligned}
$$

This logical database is similar to the model-based reflex agent architecture's internal state and the human-like concept of belief. Beliefs as a mental category is included in the language Agent0 presented by Yoav Shoham [4] as one of his two mental categories: belief and commitment.
He defines beliefs as a fact that is thought to be true by an agent at a specific time about a specific time (an agent is constrained to not believe contradictory facts). Commitments are given as a commitment to act (restricted by the agent's capabilities) not a commitment to pursue a goal.
Agent capabilities are another concept Shoham uses and are defined as relations between the agent's mental state and their environment. An agent is only capable of committing to an action iff they believe themselves to be capable. In my example above this is shown by the agent only being capable of defecting or cooperating if they are a donor in an interaction.
In my system, I wish to incorporate similar ideas to those I have mentioned: beliefs (internal state), commitments, capabilities and strategies (deductive reasoning theories). Russell and Norvig specify that an agent can be thought of as a system that perceives its environment and acts within it [3]. They then go on to explain that an agent maps percept sequences to actions. In the system I am producing, this mapping will be similar to model-based reflex agents. The mapping will be provided by changing the agent's internal state based on the percepts an agent receives and then deciding on an action to take, based on this internal state/beliefs, by way of a deductive reasoning theory.
Percepts in my system will include an observation of an action (either cooperating or defecting), an observation of a gossip action (either positive or negative) and perceiving that they are a member of a donor-recipient pair at a given timepoint (and which role they hold). This raises the question: how do agents change their internal state based on the percepts they receive? The changing of the internal state is strategy dependent, but the overriding concept

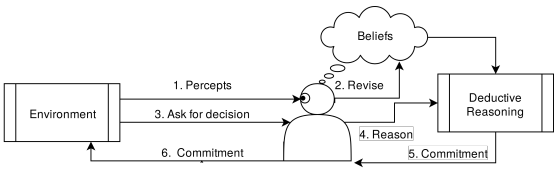

Figure 3: *Visual description of the process my agents' will follow*

is for the agents to follow a game-theoretic approach.
To extend the example of the image scoring discriminator, let us say that agent1 perceives agent2 defecting against agent3. The image scoring strategy laid out by Nowak and Sigmund [2] states that when a defection is perceived, the perceiving agent will lower the donor's image score in their beliefs.
An observation on this method: percepts cause an agent's beliefs about its environment and other agents to change at specific timepoints. This method of changing the internal state is remarkably similar to an approach to reasoning about events (similar to percepts) and time and how events change 'fluents' (similar to beliefs) known as the events calculus [1].
Due to these remarkable similarities, the use of the event calculus seems an intuitive way to program an agent to react to percepts and query beliefs. There is a version of the event calculus known as the multi-valued fluent cached event calculus (kindly provided by my supervisor Professor Kostas Stathis) which will make querying beliefs efficient.
I have covered how agents will map from percepts to beliefs and then beliefs to actions. In each timepoint of an indirect reciprocity game, this mapping will be executed as part of a cycle step. The timepoints are executed in sequence, each cycle step includes in this order the processes perceive, revise (called as a consequence of perceiving) and decide. In the perceive step agents will receive percepts from the environment, passed through an API, from these percepts they will revise their beliefs. An agent will then decide on an action to take, the agent will commit to an action at the cycle step time-

point based on their beliefs.
The capability of an agent will be constrained at a given timepoint by way of an agent perceiving that they are the donor of a donor-recipient pair at that timepoint. The decision on which agents are donors and recipients in pairs falls to the environment. If an agent perceives they are a donor they can only cooperate or defect, but when they are not they cannot cooperate or defect. At any other point when they are not a donor they will be able to gossip or be idle.
In conclusion, I believe that the agent structure I have delineated matches the 4 properties Wooldridge and Jennings described [6]. Autonomy has been satisfied by the way that agents have control over their beliefs, control over the actions they commit to based on these beliefs and merely choose to act based on their environment, not human action. Reactivity is satisfied by the process of receiving percepts, formulating beliefs from them, and then being able to respond to the changes in the environment by acting based on these beliefs.
In my system agents have 5 options as to the actions they will be able to take: idle, positive gossip, negative gossip, cooperate as a donor and defect as a donor. Both the last two properties are met by the possibility of gossip actions, and for proactivity the ability to choose between cooperating and defecting. Gossip actions are communicative actions where a gossiper communicates to another agent either positive or negative ideas about a third agent, displaying social ability. Gossip actions also show proactivity as agents can actively seek to inform other agents in a way that will affect the overall game to bring them closer to their goal.
An example of this is an agent whose goal it is to induce the evolution of cooperation gossiping to another agent a negative idea about an agent that it has viewed defecting, in the hope that the recipient of the gossip will then not help the defector by cooperating with them if they're a donor.

ii. Environment

So far I have discussed agents in my system, but agents are useless without an environment to observe and act in. My environment will run the indirect reciprocity model I outlined in my report on indirect reciprocity. The sequence of how the environment runs is given in figure 4 on page 8.
The environment consists of a community that contains 'generations' of players. The community simulates the indirect reciprocity model by creating and simulating a series of generations. Simulating a generation consists of running for each timepoint from start to end a cycle step. Each cycle step consists of 3 parts: perceive, decide and execute. Revision of beliefs is done as a consequence in an agent's mind of the perceive part, so is excluded from the environment's cycle step.
Each part of the cycle step is done for all agents at once, that is perceive sends all the percepts for a cycle step in one go, decide asks all agents to commit to a decision and then execute executes the action in the environment. Execution depends on the action if it is a gossip action or cooperate/defect action percepts are generated from this (including percepts for the selected onlookers for a cooperate/defect action) and for a cooperate action fitness is updated as per the payoff matrix in table 1.

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

Table 1: *The payoff used in my indirect reciprocity model*

The execution of an action raises the question of whether my environment is or is not deterministic. Russell and Norvig specify that when deciding on this we should ignore the actions of other agents when considering the certainty of a specific actions outcome. So when we

consider that a gossip action may or may not change the reputation of the agent it is about in the recipient's internal state this does not prevent the environment from being deterministic, as the internal state of an agent is controlled by that agent. Thus I adjudge that my environment is deterministic.
This is one of Russell and Norvig's 7 properties of a task environment, here I shall discuss further how the rest of their properties match my environment.
Due to the features of the indirect reciprocity model the environment is only partially observable. When I say the features of the indirect reciprocity model I am referring to actions which are only observable by a subset of the agents in a generation. An example of this is cooperate/defect actions which are only observable by the chosen onlookers and the donor-recipient pair.
Of course, this also highlights the environments property of being a multi-agent environment rather than a single agent environment. A more interesting question to answer on this is is the environment cooperative or competitive. This, of course, depends on whether cooperation has managed to evolve and if it is stable. But from a game-theoretic argument, even cooperative agents are competing to increase their fitness the most, the model simply encourages competition in a cooperative way.
The key idea behind indirect reciprocity is: if I as a donor help this recipient, agents who have seen this will, later on, cooperate with me. Due to this key idea, my environment must be sequential, that is a decision at one timepoint may have an effect on any other decision at a later timepoint.
Russell and Norvig state that for an environment to be discrete it must handle time and actions in a discrete way. Time in the environment I am implementing will be discrete timepoints at which each cycle step executes. Actions and percepts are also discrete as they are limited to the percepts and actions mentioned in earlier sections.
I have noted previously that agents all perceive, revise and decide at the same time. The consequence of this is that the environment does not change whilst the agents are deliberating over their action, thus making the environment static.
Finally, the rules of the environment are known to the agent. These rules include the payoff matrix for cooperate/defect actions, gossiping may or may not affect an agent depending on the recipient agents belief revision implementation and the idle action has no effect on the environment.
To sum up the properties of the environment I propose it is deterministic, partially observable, multi-agent, sequential, discrete, static and known.

iii. Communication and API Design

As I have mentioned previously the agents' minds will be hosted in a web service for the environment to interact with. The web service will be hosting agents' mind's and as such will need to have a creation process, this includes assigning a new mind with no previous internal state (except for any initial state which holds) and associating it with a strategy.
To follow the design principles of a RESTful system (REpresentational State Transfer) an individual agent is considered a resource, and multiple agents a collection. There is a specific path associated with the agent resource which includes a POST request associated with it to create a new agent. A POST request is also used to comply with RESTful design as it asks the server to create a resource. This is not a PUT request as the resource is not updated if the same request is attempted again, once an agent is created it cannot be changed, and thus the result is different (fails) if you repeat it.
In my system, an agent is assigned a player id, but this is not a universal identifier that distinguished it from all other agents. A player can be thought of similarly to a weak entity in a database, it is reliant on being part of a generation, which in turn is reliant on being part of a community. As such the 'primary key' of a player would be the id of the community,

generation and the player itself. This makes communities and generations resources as well which require universal resource indicators (endpoints). These endpoints both have POST requests which like the agents endpoint you can request the creation of a new resource too. When selecting agents to start a game with a user needs to know what strategies there are to select. As such, there is an endpoint associated with the strategy resource which a GET request can be sent to in order to get a list of strategies.
A cycle step combines the sequence of 4 possible steps: perceive, revise, decide and execute. Perceive and decide are mandatory steps which need to be initiated by the environment. A percept is the resource linked to the perceive step, the endpoint for this needs to be able to take new percept information and create the percept resource. Creation of the percept resource involves creating an event in the event calculus, this then may or may not change values of fluents (depending on the initiates_at and causes_at predicates associated with the event).
In the decide step an agent commits to an action, the resource that is created is an action. Action has a resource endpoint which you can send a GET request to (with parameters defining which player you are asking to decide on an action and the timepoint at which they are deciding) that returns the action that the player has decided on.
The final resource that I have so far decided that a user of the API should is the belief resource. Agents base their decisions on beliefs about other agents and their environment, and it would be great to be able to understand why an agent has made the decision that it has. A user can make a GET request to a specific type of belief (donor, recipient, interaction and standing) with parameters depending on which type of belief the user is asking for. The GET request made will return the belief of the agent at that time, which will hopefully give a clue as to why an agent acted as they did at a specific timepoint.
The API is designed to follow the 6 constraints of a RESTful API: uniform interface (use of JSON throughout, resources with one endpoint, correct use of HTTP methods etc.), independent client and server, stateless communication, cacheable resource declaration and a layered system architecture (though for me the system will be one entity). Documentation can be found for the API in the repository of this project under the folder "FullUnit_1819_JamesKing/AgentsService/api_docs".

III. Discussion and Conclusion

In summation of my report, my agents will follow a similar architecture to model-based reflex agents. These agents will be able to make use of the event calculus to reason about how percepts change their beliefs. From this, they will be able to deductively reason as to what actions to commit to. These actions will be constrained by the idea of agent capabilities inspired by the Agent0 language.
The environment comprises of a number of generations, which for a certain number of cycles executes the steps perceive, decide and execute. Actions of cooperation come at a cost of 1 to the donor but a benefit of 2 to the recipient, while defections cost1 none but gives none back. Cooperate/defect actions also generate percepts for onlookers and gossip actions for recipients.

References

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

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

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

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

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

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

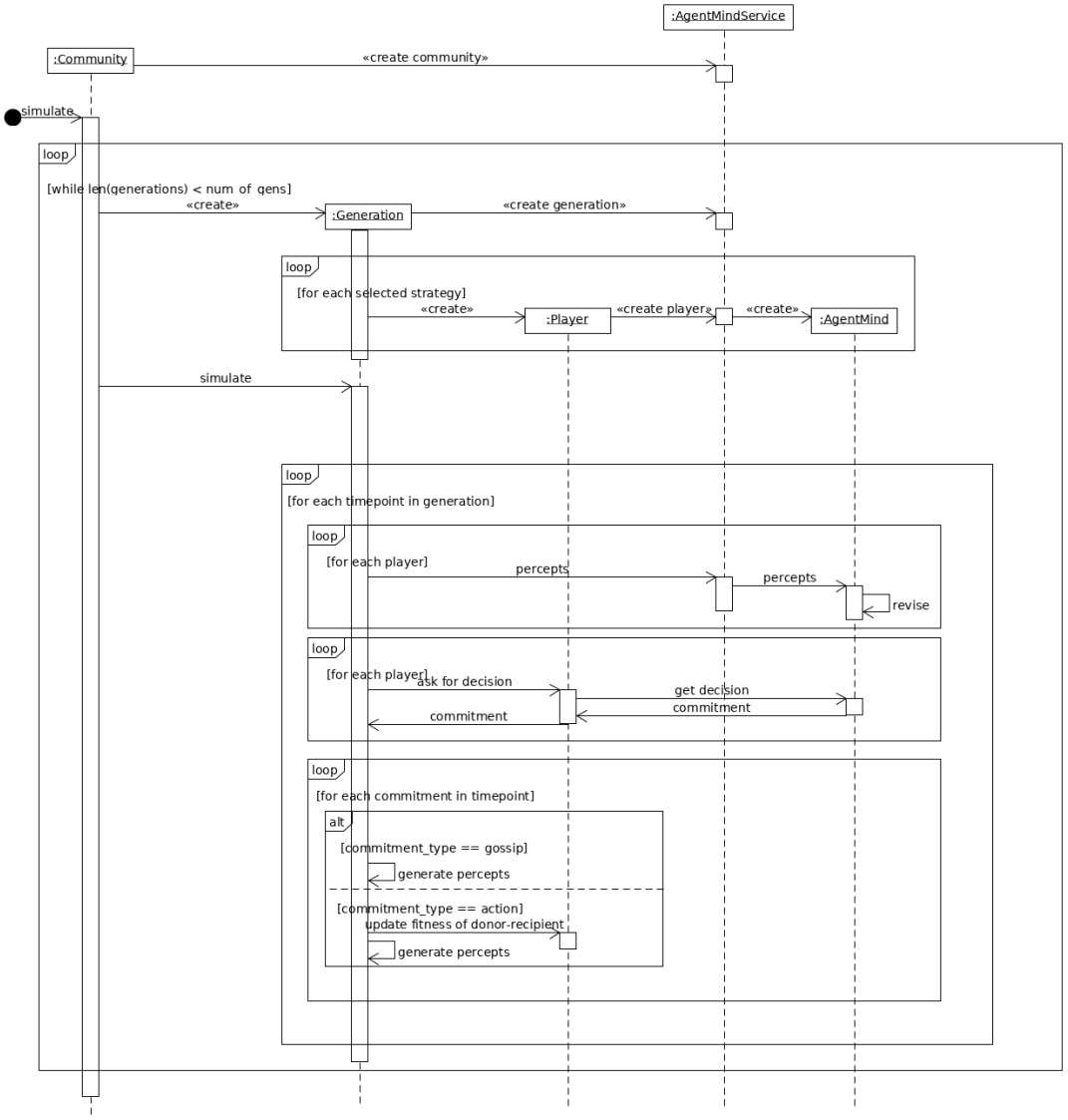

Figure 4: *Sequence diagram for the running of the environment*

Cooperative Strategies in Multi-Agent Systems

Full Unit Project: Plan

James King
Supervisor: Kostas Stathis

September 2018

Abstract

Background

The Iterated Prisoner's Dilemma uses a mechanism known as direct reciprocity to aid in the evolution of cooperation, which was popularized in Axelrod's tournaments and research on it [1]. There is a well-tested library that is able to run matches and tournaments for the Iterated Prisoner's Dilemma [2]. This is not the only game-theory mechanism used to model the evolution of cooperation. Nowak presents 4 alternative methods [4] including kin selection, group selection, network reciprocity and indirect reciprocity.

Of these indirect reciprocity stands out as an interesting model for interaction between humans or intelligent agents. Pairs of agents are chosen at random to interact (they may not necessarily interact again), in each pair one agent is the donor and the other is the recipient. A donor chooses whether to defect or cooperate, defecting gives them a higher payoff so is always tempting. However, the model utilizes the idea of reputation (also known as image score [6]) to create a condition where an agent may want to cooperate. If the donor cooperates their reputation will be enhanced, if they defect it will fall. Some strategies are known as discriminators, these will choose to defect if the recipient has a low reputation or cooperate if they have a good reputation.

Nowak and Sigmund present the idea of onlookers [6] a notion limits the number of direct observers of interactions. As is noticeable in real life, direct observation of other's interactions is not always possible, so this seems to more closely match real life. In societies with advanced communication however direct observation is not the only method of finding out about interactions, gossip has been put forward as an alternative to direct observation [7]. Previously the outcome depended upon the probability of witnessing another's decision, now with gossip, it depends upon how the information has been spread and how agents base their behaviour on gossip. Ralf D. Sommerfeld *et al* state that gossip needs to truthfully describe other's behaviour, and when hearing correct gossip an agent should act accordingly or gossip will fail.

We will develop this model as follows. To implement the agent's decision-making component we will consider using the logic programming language Prolog because it can support the implementation of transparent decisions. To implement the environment and how a user will control its evolution we will consider the procedural language Python because it has the potential to deal with the user interface and other useful control features. The issue with this is interfacing between the two languages, there are libraries such as PySwip [9] that are able to do this but are limited. For instance, PySwip struggles to work in multi-threaded environments (such as a web application).

Project Description

I will explore indirect reciprocity, by both researching and reporting on Nowak and other authors research on the mechanism, drawing on their research and possibly adding on to the ideas of onlookers and gossip to create a model for implementation. I shall be using the Axelrod-Python library to create a proof of concept

that I am able to create a web application that allows users to set up and run games and tournaments. This web application will be hosted on Heroku and created using Flask and Python.

The solution that we are exploring for the interfacing issues between Python and Prolog is to create a web service for the agent's decision-making component. This will have a Prolog server containing what [8] refers to as "agent's minds". A mind will represent information on the state of the environment using an efficient version of the Event Calculus reported in [3]. Percepts can be added to the agent's mind and based on these percepts an agent will be able to decide on an action and act on it by returning an action to the environment. Separating out the head and body as in Stathis *et al.* 2002 [8], though this solution would take this separation further, having the head based in the Prolog service and the body in the environment (Python web application).

The capabilities of the Prolog service are to be demonstrated on the web application, which will run indirect reciprocity tournaments creating the environment that agents reside in and containing the body of the agent. The web application will use the service to input percepts and get actions.

These have been the core parts of the project however, I could extend this to the other mechanisms presented by Nowak [4] if I have time. Axelrod-Python has an implementation for network reciprocity - which would be interesting to use so I could create an especially nice frontend for the patterns created in the population [5]. Due to the nature of group selection working on top of direct reciprocity [10] it would be possible to implement it on top of the Axelrod-Python library. Leaving kin selection to implement using the Prolog service.

Goals

- To create a Prolog service that is able to support the decisions of a game-theoretic agent which interacts with other systems that wish to choose an agent, provide percepts from an environment and get actions from the agent.
- To explore in a game-theoretic context a mechanism to aid in the evolution of cooperation known as indirect reciprocity.
- To create a web application that allows users to set up and run game-theory tournaments of both direct and indirect reciprocity.
- To use the web application to demonstrate an environment that combines the Prolog service and a set of game-theory strategies and evaluate the results.

Motivation

For many years researchers from many fields have speculated and attempted to explain as to why cooperation is so widespread in our world. Cooperation is extremely varied, examples include interactions between humans, between meerkats and even interspecies such as between the drongo bird and meerkat. Cooperation has been a major factor in our development as a species, allowing us to create fantastic civilisations and develop massive trade networks. It is also becoming important in technology, cooperation is key in the development and spread of multi-agent systems. Agents working together can help us solve more difficult problems.

The mechanisms that aid in the evolution of cooperation are important to understand as a matter of understanding the development of life itself and guide us in future actions. These future actions may include the development of multi-agent systems or interactions with other biological/intelligent agents. Implementing game-theoretic tournaments and agents will help further my understanding of cooperation and game-theory, allowing me to correctly aid in situations where cooperation is key. Due to the nature of the product I will be implementing, I will be furthering my programming skills especially in the areas of web development and AI. Two areas in which I have a keen interest, the development of these skills will hopefully aid me in securing a job in one of the two sectors.

Timeline

I'm approaching this project in an agile way, using 2-week sprints to complete tasks, as such the milestones here shall be assigned to a sprint week or multiple sprint weeks if they are more likely to take longer. This does not mean that I will not be working on a task beforehand if possible, but that I plan to finish that task by the end of the later sprint. In the winter period, I will take my time to consolidate and finish current work, whilst planning the next terms work. Here are the timings for each sprint:

Sprint 1: 1-12 October	Sprint 7: 14-15 January
Sprint 2: 15-26 October	Sprint 8: 28 January - 8 February
Sprint 3: 29 October - 9 November	Sprint 9: 11-22 February
Sprint 4: 12-23 November	Sprint 10: 25 February - 8 March
Sprint 5: 26 November - 7 December	Sprint 11: 11-22 March

First Term

Milestone: A proof of concept program for running a Prolog web service
I will be using tutorials online to learn how to create a Prolog web service, and from this learning, I will attempt to create my own service. This service should resemble a very early version of the Prolog service I wish to provide. This will prove that I can create a Prolog web service and give me confidence, but also reduce the risk of the project failing if I cannot create one.
Link to goal: This will act as an early version of the Prolog web service.
Date: Sprint 1

Milestone: A proof of concept web application to run direct reciprocity games and tournaments and provide statistics on them
The environment for the agents will be on a web application, this proof of concept will show that I am capable of creating a web application that can run similar games without the need to program the environment inside it yet. I will be using the Axelrod-Python library to run the games for a convincing mock-up.
Link to goal: This will aid in the creation of my web application.
Date: Sprint 1

Milestone: Report on the evolution of cooperation in relation to game-theory and different game-theoretic mechanisms to aid it
Report on past work in this area such as Axelrod's tournaments [1] and Nowak's five rules for the evolution of cooperation [4].
Link to goal: This will act as a starting point for my exploration of the mechanisms for the evolution of cooperation and help motivate the implementation of the web application.
Date: Sprint 1

Milestone: Report on indirect reciprocity, strategies for agents and the development of a concrete model to implement
Report on past work on indirect reciprocity including the strategies agents have employed playing this game. From this collated work I shall define a model which I am to implement and a number of key strategies to implement as well.
Link to goal: This is a part of my exploration of the mechanisms that aid in the evolution of cooperation and will help provide a model to implement in system for running indirect reciprocity tournaments across the Prolog service and environment.
Date: Sprint 2

Milestone: Report on the design of the Prolog service and Agents, and the web application and agent's environment
Will include the following: an agent definition and the inner workings of an agent's decision-making component, a definition of the structure of the Prolog service and management of agents (as well as the design of the API), a definition of the agent's environment and how this will work with the Prolog service, and a

design of the web application. This is key to a well-planned architecture for the Prolog service and agent environment. It will also help me research better methods of implementation for the Prolog service.
Link to goal: This will lay out a concrete way of implementing the Prolog service, web application and link between them.
Date: Sprint 2

Milestone: Design and implementation of a GUI for indirect reciprocity tournaments
This will follow on from my proof of concept web application, to which I will add a GUI to set up, run and provide analysis on indirect reciprocity tournaments.
Link to goal: This is a part of the creation of the web application.
Date: Sprints 2 and 3

Milestone: Design and implementation of the structure of the Prolog service
Taking inspiration from my proof on concept Prolog service and previous architectures for intelligent agents and multi-agent systems [8] I will define the structure of the Prolog service programmatically.
Link to goal: This is a part of the creation of the Prolog service.
Date: Sprints 3 and 4

Milestone: Design, implementation and testing of the management of agents and agent templates in the Prolog service
Once the structure of the Prolog service has been implemented I can then develop the components that will manage agents in the system. The system will have to deal with the management of multiple agents at one time, so this must be carefully implemented and thoroughly tested (I will need to learn how to test a Prolog system for this). For agents to work correctly in the system they will have to follow a regular pattern (naming and functionality). This pattern I will define in a template in the Prolog service.
Link to goal: This is a part of the creation of the Prolog service.
Date: Sprints 3 and 4

Milestone: Design, implementation and testing of the environment for a simple indirect reciprocity tournament
The environment is a key aspect of running indirect reciprocity tournaments. It will run the cycles of the game and record information about the game. This will be written in the web application and needs to be thoroughly tested.
Link to goal: This is a part of the creation of the web application, and will help me explore my model for indirect reciprocity.
Date: Sprints 3 and 4

Milestone: Design, implementation and testing of defector, cooperator and discriminator strategies for a simple indirect reciprocity tournament
We need strategies in the Prolog service to play indirect reciprocity tournaments. The simplest of which are defector and cooperator strategies, whilst discriminator strategies make indirect reciprocity tournaments varied. For these reasons I will be implementing them first, using the agent template in the Prolog service.
Link to goal: This is a part of the creation of the Prolog service, and will help explore how agents work in the indirect reciprocity mechanism.
Date: Sprint 4

Milestone: Linking together the Prolog service and web application environment
After the implementation of a subset of indirect reciprocity functionality - without onlookers and gossip - I should now be ready to link together the tested web application environment and Prolog service.
Link to goal: This will be the starting point in which the web application becomes a demonstrator of the Prolog service.
Date: Sprints 4 and 5

Milestone: Preparing the interim report and presentation

With the programs working together and finished first term reports I shall prepare the reports and presentation for the interim review.
Date: Sprint 5

Second Term

Milestone: Design, implementation and testing of the onlooker and gossip features in the web application environment
The model that I'm exploring becomes more interesting with onlookers and gossip features, with the simple version of the web application and Prolog service linked together I can now move onto developing the onlooker and gossip features. These features need support from the web application environment.
Link to goal: This is a part of the implementation of the web application and will help me explore my model for indirect reciprocity.
Date: Sprints 7 and 8

Milestone: Design, implementation and testing of the onlooker and gossip features in the Prolog service
While developing the onlooker and gossip features in the web application environment, in parallel I can develop the Prolog service support for these features. All the time ensuring that both the environment and the Prolog service can work together.
Link to goal: This is a part of the implementation of the Prolog service and will help me explore my model for indirect reciprocity.
Date: Sprints 7 and 8

Milestone: Design, implementation and testing of a development upon current strategies to play the gossip and onlookers meta-game
The onlookers and gossip features need support from agents, so they can spread gossip, listen to gossip and have appropriate reactions for this gossip.
Link to goal: This is a part of the implementation of the Prolog service and will help me explore agents in indirect reciprocity.
Date: Sprints 7 and 8

Milestone: Relinking the Prolog service and web application environment with the onlooker and gossip meta-game implemented
With the onlookers and gossip features fully support by agents, the Prolog service and the environment, I can move onto linking them together. Ensuring there aren't issues communicating and that the applications work together correctly.
Link to goal: This is a part of using the web application as a demonstrator for the Prolog service.
Date: Sprint 8

Milestone: Implementation and testing of further strategies for an indirect reciprocity tournament
So far we have 3 strategies, but there are other strategies available for these tournaments that don't just base their actions on the recipient's reputation but also their own reputation or even both. These will again use the agent template and be implemented in the Prolog service.
Link to goal: This is a part of the implementation of the Prolog service and will help me explore more advanced strategies used for indirect reciprocity.
Date: Sprints 8 and 9

Milestone: Enhanced analysis post-tournament for indirect reciprocity
Part of the reason for using Prolog for the agent's decision-making component is that we are hopefully able to query why an agent has made the decision that it has. In the analysis post-tournament using an Ajax request, users should be able to query the decisions. Furthermore, analysis of games should be enhanced

using graphs and possibly animation to display a timeline of actions. This will enhance the capabilities of the system in explaining what interactions occurred and why they occurred as they did.
Link to goal: This is part of the development of the web application and will help me understand how this mechanism for the evolution of cooperation acts.
Date: Sprints 8 and 9

Milestone: Enhanced GUI features
I will have focused a lot on developing the features of the application. With these features now implemented I can work on giving a facelift to the web application using bootstrap, React.js and CSS. This will enhance the web applications abilities as an educational system and also be a good preparation for the demonstration.
Link to goal: This is a part of the development of the web application.
Date: Sprints 8 and 9

Milestone: The report should describe the theory behind indirect reciprocity and it's strategies in relation to game-theory
Indirect reciprocity is a game-theoretic modelling tool for interactions, it is key to understanding the project to understand this model and the strategies used in it.
Link to goal: This will help me explore mechanisms for the evolution of cooperation.
Date: Sprints 9 and 10

Milestone: The report should provide an analysis and evaluation of strategies in indirect reciprocity tournaments
While implementing the project I will have gained in-depth knowledge of the strategies in indirect reciprocity and should be able to analyse them and evaluate their effectiveness in indirect reciprocity tournaments. This should use examples of tournaments run by my system.
Link to goal: This will help me explore strategies used in the indirect reciprocity mechanism for the evolution of cooperation.
Date: Sprints 9 and 10

Milestone: The report should describe the onlookers and gossip aspects of indirect reciprocity
These extensions on indirect reciprocity are an area of interest for understanding how the spread of information affects decisions made by agents. Understanding these extensions is key to understanding a lot of the motivation behind my project.
Link to goal: This will help me explore aspects of the indirect reciprocity mechanism for the evolution of cooperation.
Date: Sprints 9 and 10

Milestone: The report should describe the link between indirect reciprocity theory and real life biological and intelligent agent interactions
Indirect reciprocity is a modelling tool, but how is this modelling tool useful to understand real interactions between agents? In this section, I wish to describe the link between the theory and the real world, its uses and also it's limitations.
Link to goal: This will help me explore mechanisms for the evolution of cooperation and their application.
Date: Sprints 9 and 10

Milestone: The report should contain a design of the web application and environment, Prolog service and agents, and the connection between them
Program design is key to developing understandable, working applications. The design of the system as a whole and also individual programs have been motivated by both past research on multi-agent systems and my experiences when creating them.
Link to goal: This will describe the final product including the web application, Prolog service and their link.
Date: Sprints 9 and 10

Milestone: The report should contain a discussion of the software engineering techniques, tools and processes used and issues encountered
The use of software engineering techniques, tools and processes is key to writing good code that works. In my project, I will be using a variety of them in the hope of achieving this. Here I will review this use, both successes and failures.
Link to goal: This will allow me to review what I have learnt from the creation of the web application and Prolog service.
Date: Sprints 9 and 10

Milestone: The report should contain any interesting programming techniques employed to develop the final prototype
To create the system I will need to employ techniques used in web development and logic programming among other fields. With such a wide variety of techniques available this will be interesting to review and report on.
Link to goal: This will allow me to review some of the successes of the web application and Prolog service.
Date: Sprints 9 and 10

Milestone: Draft report
A draft report will be a useful point to review what I have written so far. Allowing me to check my progress and develop on what I have written.
Date: Sprint 9

Milestone: Development of content for the web application to make it into an educational system for the evolution of cooperation
The website should have content that enhances the user's experience, explaining the motivation behind concepts and features of the application.
Link to goal: This is a part of the development of the web application.
Date: Sprint 10

Sprints 10 and 11
These two sprints will be dedicated to finishing unfinished parts of the end products, the report as well as preparing for the demonstration. This period should act as a buffer if I fall behind, or a period to perfect or extend what I have produced if not.

Risk Assessment

Risk: Using SWI-Prolog to run a web service

This is new to both me and my supervisor. It has been done before and there are resources online, but it is still new territory to me, as is writing web services in general. This could delay the delivery of the Prolog service.
Mitigation: If I fail to be able to deliver this on time I shall switch my plan to one of the following two options: write a service in Python to replace the Prolog service or forget the service and code the program to run agents decision making directly in the web application.
Likelihood: Possible
Importance: Moderate

Risk: Failure to understand and use mvfcec

Mvfcec or the multi-valued fluent cached event calculus is a version of the event calculus where querying is faster. I have knowledge of the event calculus itself, but this is a more complicated form of it. If I fail to understand and use it, I cannot write a report on it and either the Prolog service will be far slower or I will fail to deliver it
Mitigation: I shall attempt to avoid this by working with Kostas to understand it. However, if I still fail

to grasp mvfcec, I shall either switch to using the simplified events calculus, write a service in Python to replace the Prolog service or forget the service and code the program to run agents decision making directly in the web application.
Likelihood: Possible
Importance: Moderate

Risk: My inexperience of writing large Prolog base applications

I have used Prolog in the AI course in my second year, however, these applications were nowhere near as large or complex as the Prolog service will have to be. I am doing it this way to push myself, but recognise this could cause a failure to deliver the service.
Mitigation: switch from Prolog to Python for writing the service or simply include the agent decision-making component in the web application.
Likelihood: Possible
Importance: Moderate

Risk: My inexperience in writing large web applications

In my first year, we used HTML, CSS, Javascript etc. to learn about writing web applications. However, writing a large back end and front end of a web application is new to me. Meaning there will be obstacles that I have not yet anticipated.
Mitigation: I have taken the Miguel Grinberg Flask Mega-Tutorial, to further practice my skills in writing web applications. I feel that my practice in this has allowed me to give a low likelihood that this will affect my project, and due to the resources available for Flask the importance of any obstacles to my project will be minor.
Likelihood: Unlikely
Importance: Minor

Risk: The scale of the project

With so many new technologies and techniques to learn, programs to implement and so much theory to understand, I believe this project to be quite large. If the project is too large it could delay the delivery of multiple areas of the project.
Mitigation: I have already started on much of the learning for the project, which should help reduce the time for learning. If there is too much to implement I will have to carefully scale down the project parts at a time, this may include: reducing the features of the indirect reciprocity tournaments, removing the Prolog service and replacing it with functionality inside the web application or a reduction to the 3 key strategies (defector, cooperator and discriminator). To manage this possible scaling down or up of the program I have gone for an agile approach using scrum techniques.
Likelihood: Likely
Importance: Minor (due to the many mitigations I can employ, the impact should be minimal)

References

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

[2] 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 Müller; Georgios Koutsovoulos; Areeb Ahmed. Axelrod: 4.3.0. `http://dx.doi.org/10.5281/zenodo.1405868`, September 2018.

[3] 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.

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

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

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

[7] 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.

[8] Kostas Stathis, Antonis C. Kakas, Wenjin Lu, Neophytos Demetriou, Ulle Endriss, and Andrea Bracciali. *PROSOCS: a platform for programming software agents in computational logic*, pages 523–528. 4 2004.

[9] Yuce Tekol. Pyswip. `https://pypi.org/project/pyswip/`, 2018.

[10] 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.

Of what value are these references to the project?

[1] - This paper and Axelrod's tournaments helped popularize the iterated prisoner's dilemma, being a good introduction to game theory that has a programmatic approach for computer science.
[2] - This project is a relatively new implementation of the iterated prisoner's dilemma, pushing me towards focusing on other mechanisms, whilst helping me write a proof of concept web application.
[3] - The efficient version of the Event Calculus as described in this paper is what I plan to understand and use for the agent's memory and decision making component.
[4] - This paper introduced me to the other mechanisms that aid in modelling the evolution of cooperation.
[5] - This paper extended my research on network reciprocity and pushed me towards an interest in indirect reciprocity.
[6] - This paper is key to understanding and reasoning about indirect reciprocity models and is central to past research on this mechanism.
[7] - This paper gives a good explanation of the concept of gossip in the spread of information as well as how gossip should be used.
[8] - This paper is on an architecture for the design of a multi-agent system and has helped motivate my ideas on how to design and implement the Prolog service.
[9] - This library attempts to implement an interface between Python and Prolog. It has issues and this has motivated me to find a solution using a Prolog web service.
[10] - This paper is a good description of a model for group selection and has given me an idea for the group level selection extension to my project.