Imperial College London

Final Report

Self-Organising Multi-Agent Systems

Department of Electrical and Electronic Engineering
Imperial College London

SOMAS Class 2021–2022

January 2022

Lecturer

Prof. Jeremy Pitt

Abstract

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Contents

Chapter 1

Introduction

Co-operative survival games refer to a subset of political choice games wherein 'players' must work together to overcome disaster, else suffer the consequences through both personal and communal damage. Instances of co-operative survival exist in varying media: computer games (*e.g.*, "Don't Starve," "Rust," "Minecraft"), board games (*e.g.*, Ravine) or sociological constructs (*e.g.*, Reducing viral transmission) all of which exhibit a necessity for collective action, with such notions of survival supported by extensive literature **?**. Survival games are often identified within the context of "Common Pool Resource Management" (CPR) **? ?** problems, where to facilitate co-operative survival, a pool of shared resources must be maintained to help mitigate disaster. In the general case of such survival games, following a disaster, if 1 player dies, all players die.

This project proposes a twofold bifurcation from conventional CPR problems: firstly, there are no provisions made to the common pool, meaning that we cannot refer to this problem as a linear public goods (LPG) game. Instead, players must maintain a collective resource without offering replenishment from their personal pool. This effectively constructs a scenario where all players make appropriation with insufficient provision, yielding a system of *N free-riders*.

Secondly, Ostrom notes that common-pool management problems are provably solvable through the introduction of self governing institutions **?**, offering a contradiction to the tragedy of the commons and zero contribution thesis **?**. This project instead offers a scenario without centralised governance and an inability to monitor other players, effectively preventing this research from complying with Ostrom's "design principles for enduring self-governing institutions."

The absence of a centralised authority, coupled with an inability to form self-governing institutions proposes a seemingly unsolvable problem: a common pool of resources must be maintained across a network of players, each acting rationally to maximise local utility **?**, without the prospect of forming enforceable rules. This necessitates the presence of a form of self-organisation sufficiently powerful as to overcome the challenges set by the nature of this problem, whilst accepting the risk that such a self-organising mechanism may act perniciously under these circumstances **?**.

Apropos of Artificial Societies, to combat the difficulties poised by such an unconventional CPR game we introduce a notion of self-organisation through governance in the form of treaties. Serving both "as a juristic act and as a rule" **?**, treaties act as a form of institutionalised power between players, with society coming to a mutual agreement upon the importance of such a legal device. Overall, it is the topic of this research to evaluate if the use of treaties is sufficient for self-organising this system, so as to achieve a level of stability when this game is played

iteratively in an *N-player* scenario.

Chapter 2

Organisation Structure

Could introduce each of the teams and a one liner about their agent strategy. This will obviously be expanded upon in the agent sections. Should contain an org chart and describe important parts of it. Justify certain organisational choices. Maybe consider what could have been done better?

Chapter 3

Simulation Structure

3.1 Simulation Environment

Goes over the backend design

3.2 Simulation Flow

The order in which events happen Need a nice diagram for this

3.3 Message Passing

Need a nice diagram for this

3.4 Health Modeling

Add diagram with a possible scenario of several days in a row?

3.4.1 Global Description

The health of the agents living in the tower is represented by Health Points (HP). Two mechanisms affect an agent's HP: how much food they eat, and their "cost of living". The cost of living represents how many calories a human needs to eat each day to stay healthy. These two mechanisms are implemented using the functions `updateHP` and `hpDecay`, respectively. These two functions are described below.

At the end of each day, agents are assigned an HP value based on how much food they have eaten and their cost of living. This HP value is an integer and has a maximum value of `MaxHP`, and a minimum value of `HPCritical`. As its name suggests, `HPCritical` is a critical HP value for the agents: they can only survive a certain number of days (`MaxDayCritical`) at this level. When in the critical state, if agents can increase their HP by `HPReqCToW` ("HP Required to move from Critical To Weak"), then they move into the "weak state" (**??**), and their HP takes the

value of `WeakLevel`. The amount that an agent's HP increases from eating is determined by the function `updateHP()`.

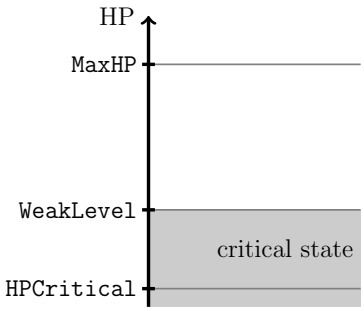


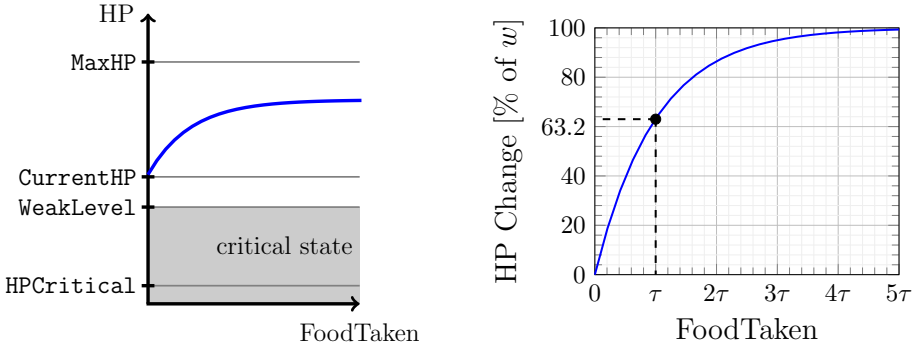
Figure 3.1: The health of the agents is represented by a HP value between `HPCritical` and `MaxHP`. All HP values which are below `WeakLevel` are classed as critical. The diagram is not drawn to scale.

3.4.2 Food and Health: `updateHP`

To increase their HP, agents need to eat. However, the amount an agent's HP improves can saturate in a single day; eating more than a certain amount will provide an agent with no extra benefit to their HP. Moreover, eating more food will lead to diminishing returns in terms of HP change. Mathematically, the ideas of diminishing returns and saturation are well captured by the step response of a 1st-order system (**??**):

$$\text{newHP} = \text{currentHP} + \underbrace{w(1 - e^{\frac{-\text{foodTaken}}{\tau}})}_{\text{HPChange}} \tag{3.1}$$

The two parameters w and τ are defined at the beginning of the simulation. The shape of this curve is given in **??** together with some important parameters.



(a) Overview (b) Detailed representation

Figure 3.2: `updateHP` as a function of the amount of food eaten ("FoodTaken").

It is not possible to gain more HP than w over the duration of one day; this is an intentional limit to prevent an agent's health from improving too quickly. As an example, we can think of an agent that starts from the weak level and wants to reach the maximum HP value. It would take several days for this agent to "recover" from this weak level and stabilise its health to a high HP value.

Note that it is possible for an agent to achieve an HP value that is larger than `MaxHP` inside `hpDecay()`. At the end of each day, the `hpDecay()` function will apply the cost of living and then bound the final HP value by `MaxHP`.

Agents in the critical state are treated differently. For these agents, HP is updated according to equation (**??**):

$$\text{newHP} = \min\left\{\text{HPCritical} + \text{HPReqCToW}, \text{currentHP} + w(1 - e^{\frac{-\text{foodTaken}}{\tau}})\right\} \tag{3.2}$$

3.4.3 Cost of Living: `hpDecay()`

At the end of each day, the HP value of the agents will be reduced by the cost of living. The cost of living is larger for an agent with larger HP value than for an agent with lower HP value. This fact is motivated by a simple observation: humans that have stronger bodies and immune systems also need more food to sustain their level of health. The exact relation between HP value, cost of living, and HP value after applying the cost of living is given by the linear relation (**??**):

$$\text{newHP} = \text{currentHP} - [b + s(\text{currentHP} - \text{WeakLevel})] \tag{3.3}$$

The parameter b is a (constant) base cost, and s is the slope of the linear function. These parameters are initialised at the beginning of the simulation. Add a diagram for this?

To ensure that the HP value at the end of the day is bounded by `MaxHP`, we slightly modify (**??**) to produce (**??**):

$$\text{newHP} = \max\{\text{MaxHP}, \text{currentHP} - [b + s(\text{currentHP} - \text{WeakLevel})]\} \tag{3.4}$$

For agents in the critical state that gain `HPReqCToW` HP in a single day, i.e. their HP after eating is

$$\text{currentHP} \geq \text{HPCritical} + \text{HPReqCToW}, \tag{3.5}$$

their HP will be set to `WeakLevel`:

$$\text{newHP} = \text{WeakLevel} \tag{3.6}$$

Agents in the critical state which do not manage to improve their HP by `HPReqCToW` will be kept in the critical state:

$$\text{newHP} = \text{HPCritical} \tag{3.7}$$

with the `daysAtCritical` counter incremented by 1. If `daysAtCritical` reaches `MaxDayCritical`, the agent dies and is replaced. This counter is reset to 0 if an agent exits the critical state.

Chapter 4

Visualisation

Important things about front end.

Chapter 5

Team 2 Agent Design

5.1 Core Idea and Overall Strategy

The objective of this agent design was to implement a form of reinforcement learning to allow for an increase in performance and thus utility over time. This must be met without compromising on the overall goal of the simulations, whereby the agent must strike a balance between prioritising individual utility or prioritising collective utility, the latter involving a level of cooperation with the other agent types. Certain parallels can be drawn to evolutionary ecology, as the Tower with the Resident agents can represent a closed ecosystem, with the Platform providing a limited food source.

5.1.1 Competitive Exclusion Principle

Competing and cooperating with the other agents may seem to contradict the Competitive Exclusion Principle. The Competitive Exclusion Principle states that two or more species competing for the same limited resource will not be able to exist at constant populations and will inevitably lead to either the weaker competitor's extinction, or a distinct change in its behaviour. The superior competitor generally will have an aggressive competitive advantage which leads to this. This principle aligns with our context, as our agents must satisfy their hunger with limited food being delivered on the Platform, while other team agents will do the same. Conversely, we must not cause the deaths of other agents by taking too much food for ourselves since all agents must also strive to achieve the highest collective utility possible. In order to balance these objectives, we have implemented functions unique to our agents to ensure the survival of our agent without leading to the demise of others.

5.1.2 Utility Considerations

It is not possible to maximise both individual and collective utility simultaneously. Maximising only individual utility would result in taking more resources than required to satisfy an agent's needs to ensure that it can survive on floors with less food. This action is contradictory to maximising collective utility as it would unnecessarily reduce the resource pool for the other agents. As a result, collective utility is prioritised to enhance the survival rates of all the agents in the Tower.

5.1.3 Dynamic Adaptation

Our goal is to create a unique advantage by allowing for the agent to experiment with different actions to different situations, until the most appropriate actions can be dynamically chosen for the future. A reward system is utilised, whereby a suitable action will be rewarded, and a detrimental action will be penalised. Given enough iterations, the agent should be sufficiently trained to consistently choose the most correct actions. This is achievable as most other agents have fixed behaviours which can be predicted by our agent; however, we also experiment with random agents to understand and quantify the change in performance and utility. Relying on a stochastic method to execute learning can result in vastly different outcomes, but it is expected that in general the agents will perform better with time.

5.1.4 Education and Reincarnation

The knowledge gained over time is stored in the agent's memory. In order to provide a more realistic simulation of the Tower scenario, each individual agent's memory will be erased upon death. This presents a limit to the effectiveness of the learning method, as the agent's learning rate cannot be so high that it only takes higher risk choices to learn faster, it must consider its survival. However, a separate setting was made where "reincarnation" was implemented. In this setting, upon the agent's death, it is respawned with all the knowledge it had gained in previous lives. We use this as a comparison metric to qualitatively assess any difference in overall utility and test the model with maximum iterations to learn as opposed to just its lifespan.

5.2 Strategies and Algorithms

5.2.1 Q-Learning

Instead of using a deterministic method, we apply Q-Learning to dynamically learn a strategy for our agent based on the environment given by the game settings. The Q-Learning method we use is an extension of the classic Temporal-Difference (TD) learning.

One-Step Temporal-Difference Learning

Temporal-Difference (TD) Learning combines the attributes of Monte Carlo methods and Dynamic Programming. Similar to Monte Carlo, TD has the ability of on-line learning from the specific feedback that the agent receives without the need to construct a model of the environment. In addition, TD is equipped with the advantage of Dynamic Programming. It is able to update the estimates of the current state based on other estimations of future states, which means that it is not necessary for TD learning to wait for the final outcome when updating the strategies **?**.

In one-step TD learning, the value of one state will be updated only in relation to the estimation of value for the next state. The update step is performed with the following function

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{5.1}$$

where V is the value of states and R is the reward for transferring to some state.

The $R_{t+1} + \gamma V(S_{t+1})$ term is the reward for transferring to the next state plus the time-decayed value of the next states. This is the approximation of the Markov Reward Process if we keep iterating this step until reaching the final state. Thus, $R_{t+1} + \gamma V(S_{t+1})$ can be considered as a better estimation of the value for the current state. In this case, it can be easily observed that $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is the error between our estimated current state value and its better approximation. Finally, the value of the current state is re-estimated with this error plus its previous estimated value.

Q-Learning Algorithm

Q-Learning is an off-policy version of the Temporal-Difference method. Instead of only considering the transition between states, policies and actions are introduced in Q-Learning. An action is a behaviour that the agent can perform under some situation. A policy is the behaviour for an agent at a given time, which can be interpreted as the mapping from a given state to a given action corresponding to that state. For an off-policy algorithm, the policy that is evaluated and improved can differ from the policy that the agent will take to introduce the action. In other words, we may update our strategy under the assumption that the agent will take one action in the next state, but the agent may eventually take another action.

In Q-Learning, the transition between state-action pairs is involved. The state-action pairs also belong to the Markov Reward Process as in TD method. Q value measures the value of a state-action pair which means that Q value tells the quality of taking one action under given state. Moreover, a Q table is a combination of all the Q values which includes all the possible states and corresponding actions along with their Q values **?**.

In terms of the actual policy of an agent in Q-Learning, we combine the exploration policy and greedy policy. In greedy methods, the agent selects an action which has the highest quality under the current state. In other words, the agent will select the action with the highest Q value in that state. The exploration policy introduces a probability δ, which is the probability that the agent will ignore past experience (i.e., the Q table) and take random actions to explore potentially better results. In summary, the agent will have a probability of δ to take a random action and a probability of $1 - \delta$ to follow the greedy policy.

The Q table is updated with the following function

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{\alpha} Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{5.2}$$

The updating function is nearly the same as that in TD learning except that we now measure the value of a state-action pair. The error between the estimated Q value and the better estimated Q value of the current state is also utilized in the updating process. Although the agent has a probability to take the random policy, we assume that the agent acts according to the greedy policy for updating which is expressed in the term $\max_{\alpha} Q(S_{t+1}, a)$. This ensures a stable but exploratory learning process and that is why Q-Learning belongs to the off-policy category.

5.2.2 Hill Climbing

To better enable the mixed strategy play for our agent we implement the Policy Hill-Climbing (PHC) algorithm, which is a simple extension of Q-learning. PHC is a simple adaptation that performs the hill-climbing algorithm in the space of mixed strategies. Here, the starting point of the policy space is initialized so that all policies have a uniform probability unlike the more common random initialization of probabilities. The PHC algorithm can be formulated as follows:

1. We let $\alpha \in (0, 1]$ and $\beta \in (0, 1]$ be two learning rates and initialise the Q and policy tables with the following values

$$Q(s, a) \leftarrow 0, \quad \pi(s, a) \leftarrow \frac{1}{|A_i|} \tag{5.3}$$

2. Repeat for each iteration

   (a) According to the mixed strategy $\pi(s)$ with suitable exploration select action a from a given state s

   (b) Observing next state s' and reward r update the Q table

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{\alpha} Q\left(s', a'\right) \right) \tag{5.4}$$

   (c) Update the policy table $\pi(s, a)$ and constrain it to a legal probability distribution

$$\pi(s, a) \leftarrow \pi(s, a) + \begin{cases} \beta & \text{if } a = \text{argmax}_{a'} Q(s, a) \\ \frac{-\beta}{|A_i|-1} & \text{otherwise} \end{cases} \tag{5.5}$$

PHC keeps the Q-values as normal Q-learning would, but additionally also retains the present mixed policy. The algorithm performs non-gradient based rational optimisation by increasing the probability of selecting the action that gave the highest Q-value according to some given learning rate α. The algorithm improves the policy by increasing the probability for selecting action with the highest Q-value according to the learning rate $\beta \in (0, 1]$. It should be noted that for a learning rate $\beta = 1$ the algorithm performs homogeneously to Q-learning as with each iteration, or step, the algorithm executes the highest Q-valued action which is precisely the greedy policy as explained earlier in Q-learning.

The PHC algorithm is only able to locate an optimal policy against stationary strategy opponents and has never been proven to converge if pitted against non-stationary strategy opponents **?**. As according to Q, that converges at some Q', the mixed strategy π is greedy and will therefore converge to the best possible policy in response to the environment the agent is placed in.

5.3 Agent Design

5.3.1 Design of states and actions

As the learning involves interaction with the environment, it is essential to define the state space and the action space. A state space is a set of states, represented by vectors of observations, that an agent could possibly experience throughout its life. An action space is a set of actions that an agent could possibly perform in the game. The design of the state space and action space is tailored to the task assigned, namely maximising both individual and collective utilities.

Design of state space

Firstly, we identified what observations are needed to determine the condition of the agent and the environment which is represented by the Tower and other agents. Considering the given task, we chose to observe four variables:

Table 5.1: State space defintion

State Number	HP	Food on platform	Days at critical	Neighbour's HP
0	0 – 9	0 – 9	0	-1
1	0 – 9	0 – 9	0	0 – 9
2	0 – 9	0 – 9	0	10 – 19
...				
11	0 – 9	0 – 9	1	-1
12	0 – 9	0 – 9	1	0 – 9
...				
44	0 – 9	10 – 19	0	-1
45	0 – 9	10 – 19	0	0 – 9
...				
440	10 – 19	0 – 9	0	-1
441	10 – 19	0 – 9	0	0 – 9
...				
4399	90 – Max HP	90 – Max Food	3	90 – Max HP

1. Our agent's HP
2. Amount of food currently on the platform
3. The number of days in critical condition
4. The HP of the neighbour that is on the floor below our agent

The first three observations served to fulfil the first aim of the task, as they could effectively reflect the health condition of our agent. The last observation was used to assess the impact of the action of our agent on the neighbouring agent, which is useful for the second aim of the task. Subsequently, we found the resolution of each observation by trial and error. Having a high resolution, the agent would be able to perceive tiny changes within the environment at the cost of increased number of states, and thus slowing the learning process. Having a low resolution, the agent would not be able to adapt to changes in the environment effectively and promptly. Finally, the state space of our agent is defined by ten state intervals for agent's HP (0 – 9, 10 – 19, . . . , 80 – 89, 90 – Max HP), ten state intervals for the food on platform (0 – 9, 10 – 19, . . . , 80 – 89, 90 – Max food), four states for Days at critical (0, 1, 2, 3), and eleven state intervals for neighbouring agent's HP (-1, 0 – 9, . . . , 80 – 89, 90 – Max HP), with an additional unknown state represented by –1, as the neighbouring agent might refuse to share such information. Table **??** illustrates the state space definition.

Design of action space

In the game, taking food is the main interaction with the Tower, which provides feedback according to the amount of food taken. Thus, we divided the action of taking food into six distinct actions with different number of intended food intake, as shown in the table **??**.

We excluded the possibility of taking more than 30 food units, as eating above this level will only bring punishment to our agent according to our reward design. Note that the intended

Table 5.2: Action space definition

Action number	Intended food intake
0	5
1	10
2	15
3	20
4	25
5	30

food intakes are multiple of 5, as we would like the agent to have less actions to learn, resulting in faster convergence.

Design of reward function

The reward function provides a way for us to set the desired behaviour of our agent. The reward function will assess the quality of a particular action under a specific state and provide feedback, either reward or punishment, to the agent. We have designed four assessment criteria, namely surviving bonus, eating bonus, wasting bonus and saving bonus. We encourage our agent to survive: if the agent survives without entering critical state, it will be rewarded, otherwise punished; we encourage our agent to eat less: if the agent eats below a certain level of HP, it will be rewarded, otherwise punished; we encourage our agent to not waste food: if the actual HP increment is lower than expected, the agent will be punished; we also encourage our agent to save neighbour: if the neighbouring agent is in critical state after our action, the agent will be punished. During the experiments, depending on what behaviour we want to observe, the reward function will be tuned accordingly.

Based on the health decay model defined in **??**, the agent reward was designed to encourage the agent to reward a health above critical and punish dropping to critical health as well as punish overeating using a reward point value for each day. The reward function with its tuned hyperparameters is as such:

- Set reward to zero for each new day
- If HP is 80-100 (Punish)
  - Reward == 0.2 × food taken that day
- If HP is 4-79 (Reward)
  - Reward = 3
- If hp is critical (Punish)
  - Reward=10 × Consecutive days agent has stayed at critical

Chapter 6

Team 3 Agent Design

Chapter 7

Team 4 Agent Design

Chapter 8

Team 5 Agent Design

The Team 5 agent operates on the basis that upon entering the tower, the need to ensure its own survival is its only aim and therefore the agent should maximise its own HP whenever given the opportunity to do so. When there is not enough food to fully satisfy all agents, the agent alters its behaviour to best increase its long-term survival.

This situation can be thought of as a 'hawk-dove' game where it is beneficial for all agents to take less food but any single agent who does so is worse off if no other agents follow suit. Communication is key to breaking out of this situation. This agent will attempt to establish relationships with surrounding agents – the agent has parameters to judge other agents and also attempts to learn about the tower – and encourage behaviours that benefit the overall tower utility.

However, the agent will also be influneced by the behaviour of agents around it: if other agents are selfish then it will also be selfish. This can be thought of as a 'tit-for-tat' strategy, but with enough HP, agents will be willing to 'make the first move', with the insurance that if others do not follow then they have some buffer time to re-establish their own selfish behaviour.

An agent will maintain a low HP only if it is confident that other agents will allow it to survive by leaving enough food, through the signing of treaties to build trust. A tower that consists solely of Team 5 agents would require those at the top to be willing to decrease their own short-term satisfaction in order to give the best chance of survival for all agents, and trust others to do the same upon reshuffling.

Section **??** provides an overview of the agent's parameters and strategy, Section **??** describes the agent's communication strategy, Section **??** describes how the agent stores its social network and how it judges other agents, and Section **??** describes how the Team 5 agent utilises treaties. Finally, Section **??** contains analysis of this agent including how it performs in a Tower consisting of itself alone as well as together with other agent types and Section **??** provides summarising remarks regarding the implementation of this agent type.

8.1 Agent Overview

Give better description of Run function

The strategy employed by this agent aims to maximise its chances of survival while also considering those who have helped the agent in the past.

Selfishness	Ranges from 0 to 10. Determines how much the agent will act for their own survival as opposed to the best interests of the surrounding agents
Last Meal	How much the agent ate in their last meal
Days Since Last Meal	How many days since the agent last ate
Aim HP	Defines the goal HP the agent would like to reach at the end of the day
Attempt Food	The amount of food the agent will attempt to take from the platform
Messaging Counter	Determines which message type to send to which floor
Leadership	A threshold value required for the agent to propose treaties
Social Memory	Stores the information the agent learns about other agents, as well as its opinion of them (see Table **??**)
Surrounding Agents	A list of the agent's neighbours

Table 8.1: The personality parameters of Team 5's agent

At the beginning of each day, the agent's 'personality' is updated using various parameters stored from previous days. These personality parameters are shown in Table **??**.

At the start of each tick, the agent checks to see if it has received any messages and if so, will respond to these[1]. The agent then sends out messages to other agents. The mechanism for messaging is described in Section **??**.

8.2 Messaging

Collaboration between different agents operating in a system requires communication. The motivation for messaging is to improve the agent's understanding of its surroundings, enable building of mutually beneficial relationships, and introduce collaboration and argeement between parties.

Strategy

The agent's approach to messaging is to adopt a "passive observer" role focussed on gathering information about other agents to organise itself with these agents in the tower. With more information at its disposal, the agent can make more informed decisions to balance its own individual utility with the collective utility of the agents in the tower.

choose whether to say 12 or 8 ticks to send all messages
choose whether to include intended food intake in messages
The agent sends three types of messages to each of the agents residing one or two floors above and below its own floor; it therefore takes 12 ticks to send all of its messages. These messages ask for the HP, intended food intake, and actual food intake of other agents. The decision of which message type to send to which floor is controlled by the Messaging Counter described in Table **??**. This counter is incremented each tick, and reset either every 25 ticks or at the end of each day, whichever comes first. This ensures that the agent is requesting up-to-date information regularly for use in its decision making.

[1]Note that an agent can receive and respond to only one message per tick.

Agent ID	An agent's unique ID
Agent HP	The agent's last known HP level
Food Taken	The amount of food the agent last took
Favour	The Team 5 agent's opinion of this agent
Days Since Last Seen	The number of days since the last message from this agent

Table 8.2: Team 5 Agent Memory

The order of sent messages is:

1. Ask for an agent's HP
2. Ask for an agent's previous food intake
3. Ask for an agent's intended food intake

The Team 5 agent first targets the agent immediately below it, then the agent immediately above, then two floors below, and finally the agent two floors above.

8.3 Social Memory

The agent stores the information gained from communication in its 'memory': the information stored by this data structure is shown in Table **??**.

This memory allows the agent to construct a social network of agents in the tower and evaluate others based on these parameters. This consequently leads to the idea of favour, a metric the Team 5 agent uses to judge other agents. If more than five days have passed since the agent's last interaction with another agent, then the memory of that agent is reset. This prevents the agent from using out-of-date information in its calculation and also allows for the removal of dead agents from its memory.

Favour

The favour metric quantifies the agent's opinion of others in the tower. It is bounded between 0 and 10, and in a single tick can either increase by 1, decrease by 3, or stay the same:

$$\texttt{hpScoreOther} = -1 \times \frac{\texttt{otherHP}^{1.7} \times \texttt{foodTaken}^{1.3}}{\texttt{maxHP}^3}$$

$$\texttt{hpScoreSelf} = \frac{\texttt{ownHP}^{1.7} \times \texttt{ownAttemptFood}^{1.3}}{\texttt{maxHP}^3}$$

$$\texttt{judgement} = 100 \times (\texttt{hpScoreOther} + \texttt{hpScoreSelf})$$

$$\texttt{unboundedFavour} = \begin{cases} \texttt{originalFavour} + 1, & \texttt{judgement} > 0.075 \\ \texttt{originalFavour} + \max\left(\frac{\texttt{judgement}}{2}, -3\right), & \texttt{judgement} < -2 \end{cases}$$

$$\texttt{favour} = \begin{cases} 0, & \texttt{unboundedFavour} < 0 \\ \texttt{unboundedFavour}, & 0 \leq \texttt{unboundedFavour} \leq 10 \\ 10, & \texttt{unboundedFavour} > 10 \end{cases}$$

Measuring favour allows the agent to act accordingly to how other agents are behaving around it, while taking on a passive conformist approach to decision making. If the agent views others around it more favourably, then it eats less food and aims for a lower HP in hopes of improving collective utility. The agent is also more likely to propose treaties to, and accept treaties from, highly favoured agents.

For agents which have low favour, the Team 5 agent maintains its passiveness and does not "punish" them by, for example, taking more food than is necessary. The effect of "punishing" an agent propagates to all floors below the agent and is therefore not deemed an appropriate course of action. Instead, the agent strives to behave positively if surrounding agents also demonstrate positive behaviour.

It should also be noted that while the Team 5 agent computes favour for agents both above and below its floor, a positive favour value for agents above is helpful to that agent only if they are moved to a floor below the Team 5 agent following a reshuffle: this is because there is no way to reward the behaviour of an agent above you in the tower.

8.4 Treaties

The Team 5 agent's treaty strategy consists of three main parts:

1. Deciding whether to accept or reject proposed treaties
2. When to propose a treaty
3. What to propose as a treaty

Responding to Proposed Treaties

A proposed treaty is always rejected if:

- It conflicts with any other active treaties. A treaty is considered to conflict with another treaty if they could ever cause the agent to take incompatible actions. For example, an agent cannot leave more than five units of food and fewer than four units of food at the same time.

- It is badly formed. For example, a treaty which requires the agent to leave more than 100% of the food on the platform is badly formed.
- Check this one It requires agents to leave less than a certain amount of food on the platform, as this is never advantageous to the tower as a whole.
- Check this one It requires the agent to leave an exact amount of food on the platform, as this would result in all but one of the agents signed up to the treaty to take no food each day.
- It makes a request based on low HP, low amounts of food on the platform, or a low floor, as in these conditions the agent will already be desperate for survival, so will not want to be further constrained by a treaty.
- Check this one It makes a request based on HP where the threshold is within the "weak" or "critical" HP bands, as this means the agent could become critical or die as a result of following the treaty.
- Check this one It would require the agent to take less food than is required to avoid entering the "critical" HP band.

If the treaty does not meet any of the above conditions, the `decision` value is then computed as:

$$\texttt{decision} = 6 - \texttt{selfishness} + \texttt{proposerFavour} - \frac{\texttt{treatyDuration}}{3}$$

This calculation is based on the following considerations:

- Selfish agents will always be less likely to sign treaties
- The proposing agent is more likely to be trusted if the receiving agent views them more favourably
- Longer treaties are viewed more negatively because factors such as the receiving agent's opinion of the proposer and the agent's selfishness can change over time, meaning the same treaty could be viewed less favourably in future. Therefore, the agent should avoid constraining its actions to satisfy an agent of which it could form a negative opinion.

If the `decision` value is positive, the treaty is accepted.

Proposing Treaties

At the start of each day, the agent checks whether it has become a 'leader': if it has, then the agent proposes a treaty. Whether an agent becomes a leader or not is based on:

- The agent's selfishness: a selfish agent is less likely to propose treaties
- The agent's floor: agents higher up in the tower are more likely to propose treaties. This is because agents higher up in the tower have more access to food and therefore have more control over the food available to those below them. This control means these agents are in a better position to effect change.

- The agent's `leadership` threshold: this is randomly set during the agent's initialisation and those with a lower leadership threshold are more likely to become leaders.

The leadership value is calculated by:

$$\texttt{leadership} = x - \texttt{selfishness} - \texttt{agentFloor}$$

The variable x is a random number between 3 and 12, inclusive. If `leadership` is greater than the agent's leadership threshold, then the agent becomes a leader and proposes a treaty.

The proposed treaty is always that agents whose HP is greater than or equal to a certain value should always leave 100% of the food on the platform, and always lasts for five days. The threshold value for HP is calculated using Equation **??**. Note that in this equation, `WeakLevel` refers to the threshold between the "critical" and "weak" HP states.

$$\texttt{hpThreshold} = \texttt{aimHP} - \frac{\texttt{aimHP} - \texttt{WeakLevel}}{10} \times (\texttt{leadership} - \texttt{leadershipThreshold}) \quad (8.1)$$

8.5 Experimentation

This section focuses on how the agent's performance varies with changes in its environment.

Homogeneous Tower Experiments

To evaluate the self-organisation abilities of the Team 5 agent, simulations were run with the agent in variations of a homogeneous tower (i.e. a tower that contains agents of only one type). The agent has been evaluated on three main metrics:

1. The number of deaths
2. The average utility
3. Whether self-organisation was achieved.

Self-organisation has been defined in this case to be when all agents take only what they need to survive and there are little to no deaths beyond a certain point in time.

Food Per Agent On Platform

The baseline experiment for testing was to vary the amount of food per agent available on the platform; it was expected that having more food available would result in fewer deaths. In this experiment, simulations were run for 100 simulation days, with a reshuffle period of 7 days and 50 agents in the tower and repeated three times, with the average of these runs taken. Figure **??** shows that the number of deaths decreases and average utility increases as we increase the amount of food available.

When the ratio of food to agent is greater than 9, the agents are able to self-organise within 100 days, and typically do so within 70 days. When food availability was less than this, the agents did not achieve self-organisation. However it was interesting to observe that more treaties were

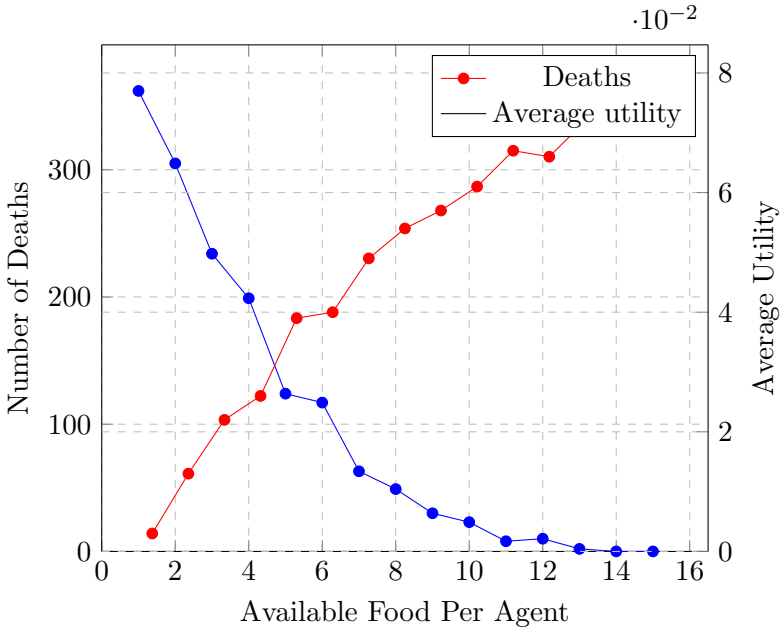


Figure 8.1: Number of Deaths and Average Utility vs Available Food Per Agent

accepted than rejected. This shows that the agents recognise the need to self-organise and are willing to make attempts to do so. It is possible that they did not achieve self-organisation because agents died too quickly to reach agreements among enough agents for this to be successful.

Number of Agents

Through experimenting with the Team 5 agent, it was found that the variable with the largest effect on the likelihood of self-organisation being achieved was the number of agents present in the tower. This is likely an effect of the agent strategy relying heavily on communicating and establishing relationships with other agents. Having more agents in the tower means that it takes longer to form opinions of every other agent, and the likelihood of being reshuffled next to other agents that you have already seen decreases as the number of agents increases.

After making this observation, it was decided that it would be interesting to investigate how long it takes for varying numbers of Team 5 agents to self-organise for a given amount of food available. Table **??** shows the results of varying the number of agents in the tower, with the food per agent ratio set to 2.

The results show that the length of time it takes for the agents to organise is very variable: randomness plays a large factor due to the many random variables present in the simulation, such as an agent's floor and its leadership threshold. For organisation to occur, agents who are more likely to become leaders need to become less selfish and be reshuffled high up in the tower. This will enable them to send treaties to surrounding agents. The surrounding agnets must view them favourably enoguh to accept the treaties they receive.

It was found that, with the food per agent ratio set to 2, there appeared to be a threshold value of approximately 18 agents were organisation was far less likely to occur within the early days of the tower. Further testing showed that this threshold increases with the amount of food available per agent. Figure **??** shows the results of a simulation where 20 agents managed to organised themselves.

Number of agents	Days to organise	Deaths before self-organisation
<10	0	0
15	80	46
15	0	0
17	0	0
18	45	22
18	95	79
20	No organisation in 500 days	
20	242	226
25	No organisation in 700 days	

Table 8.3: The number of deaths and number of days required for Team 5 agents to self-organise

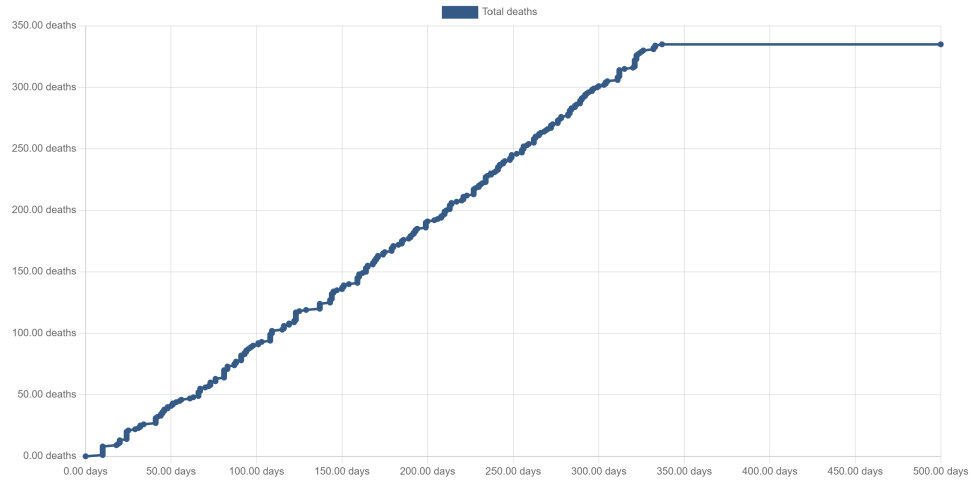


Figure 8.2: Cumulative deaths per day for a 500 day simulation of 20 Team 5 agents with 2 food per agent

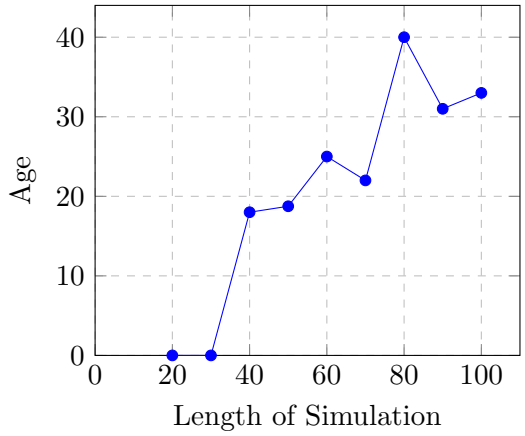


Figure 8.3: Average age upon death vs length of simulation. Points with age 0 mean no agent died during those simulations.

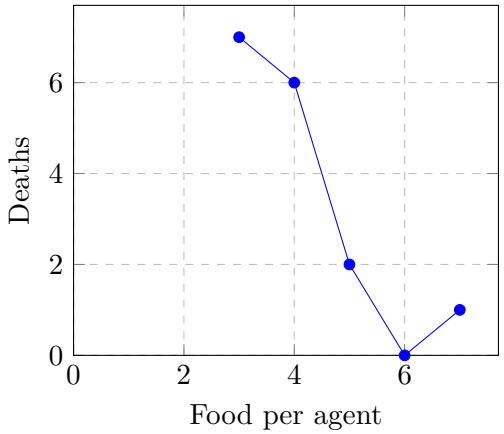


Figure 8.4: Number of deaths vs food per agent ratio

These results naturally led to the consideration of whether a homogeneous tower of Team 5 agents would achieve organisation given an infinite amount of time regardless of the number of agents provided at least two food per agent was available. We predict that this is not possible: every time an agent is replaced in the tower, its replacement is initialised with maximum selfishness and has no social network. It takes time for agents to build trust with each other but deaths occur too quickly for this happen if the settings of the tower are too strict.

Heterogeneous Tower Experiments

These experiments compare the Team 5 agent with the agents designed by the other agent teams. The base parameters for all simulations are as follows:

- 10 Team 5 agents, and two agents from each other team, excluding the random and selfish agents (i.e. half of the tower is Team 5).
- 6 food per agent
- 3 day reshuffle period
- 20 day simulation

Simulation Length

Check this result: is it actually true? Figure **??** shows the relationship between the length of the simulation and the average age of Team 5 agents when they die: the graph shows a positive trend of increasing average age, implying that Team 5 agents perform better over time.

Food Per Agent

Figure **??** shows the effect of increasing the amount of food on the platform: the graph trends downwards as expected, showing that more food reduces the number of deaths.

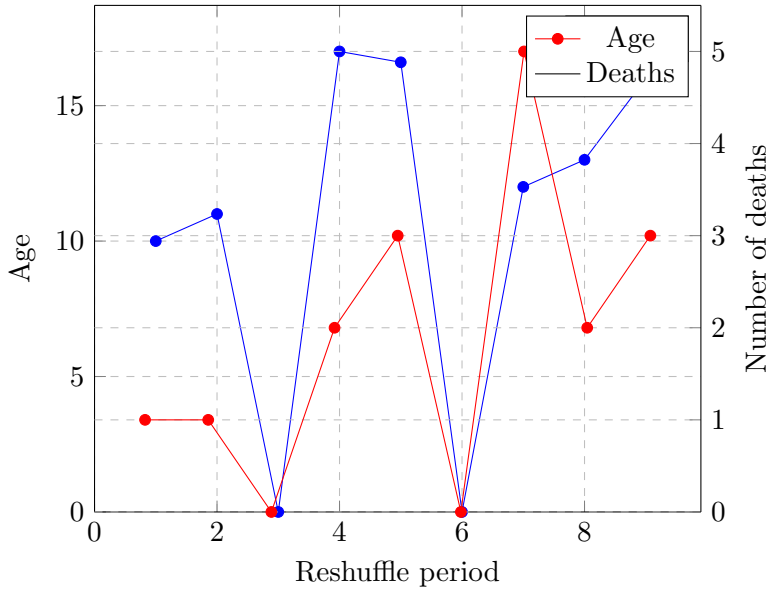


Figure 8.5: Average age upon death and number of deaths vs reshuffle period

Agent type	Team 2	Team 3	Team 4	Team 6	Team 7
Team 5 deaths	25	5	0	25	12
Other agent deaths	19	5	0	35	8
Organisation?	NO	67 days	0 days	NO	62 days

Table 8.4: Team 5 interactions with other agents

Reshuffle Period

Self-Organisation With Other Agents

The overall aim within the tower is for agents to self-organise and create a stable society. Tests were run to investigate how compatible the Team 5 agent is with each of the agents designed by each of the other teams. Simulations were run with a tower consisting of Team 5 agents and one other agent type, with seven agents each and five food available per agent, to see whether they could self-organise within 100 days. These results are shown in Table **??**

8.6 Conclusion

The Team 5 agent is capable of achieving self-organisation within the tower provided the simulation parmaeters are not too strict. A large number of agents within the tower greatly inhibits the agent's ability to self-organise, but they are capable of organising when there is the food per agent ratio is at least 2, for low agent numbers. This demonstrates the reliance on communication and sharing of knowledge in order to solve the problem of long-term survival in the tower.

During the presentation for this coursework, Professor Pitt compared the Team 5 agent's strategy to theory from Ober, who stated that "alignment is an epistemic process, predicated on the right people having and using the right information in the right context" **?**. We had not considered this theory when designing our agent, however in a sort-of convergent evolution we observed

this in the randomeness and large variation in the Team 5 agent's ability to self-organise, and the events that need to take place for this to be successful.

Chapter 9

Team 6 Agent Design

Chapter 10

Team 7 Agent Design

Chapter 11

Experiments

Experiemntal design - What are we looking to find – Conditions that lead to stability/instability - Independent and dependent variables – Experimental parameters - Measurement methods

Chapter 12

Results

Chapter 13

Discussion

Chapter 14

Conclusion

Conclusion section.

Chapter 15

Future Work

Futur Work section.

Appendix A

Appendix

Appendix Section. Org chart could go here. Potentialy could include implementation details.