Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project
Social Construction in Focal Point Games

Author:
Thomas Lewis

CID:
01707143

Supervisor:
Prof. Jeremy Pitt

Second Marker:
Dr. Chen Qin

Submitted in partial fulfillment of the requirements for the MEng degree in Electronic and Information Engineering of Imperial College London

June 2023

Plagiarism Statement

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have submitted, or will submit, an identical electronic copy of my final year project to the provided Blackboard module for Plagiarism checking.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

I have not used Large Language Models as an aid in the preparation of my report.

Acknowledgements

First, my sincere thanks go to my supervisor, Prof. Jeremy Pitt, whose enthusiasm and vision has shaped this project into something for us both to be proud of.

Next, I would like to thank all the staff at Imperial who have taught or interacted with me during my four years at the college. In particular, I would like to thank my tutors, Dr. Adam Spiers and Dr. Adria Junyent-Ferre, who have helped guide me through my degree.

Finally, I would like to thank my friends and family for their encouragement and support throughout the project and my degree as a whole. Without them, I wouldn't be the person I am today.

Abstract

Schelling's concept of the focal point - a strategy or option in a coordination game that is more salient than others - is widely accepted, but there is still some uncertainty as to how they form. This project aims to further our understanding of the formation of focal points, particularly through social construction. This is achieved through the creation of a self-organising multi-agent simulator, and a range of experiments that explore different coordination mechanisms. The findings challenge the narrative that coordination has been and is achieved through centralised means. The project demonstrates how focal points can form, and therefore how coordination can be achieved, with and without agent communication, with potential applications in the field of distributed artificial intelligence.

Contents

Chapter 1

Introduction

Consider the following scenario: you are presented with five numbers, and must pick one of them. A number of other people are also presented with the same five numbers. If you all choose the same number, everyone wins and receives a large sum of money. The prize is independent of the number chosen. If more than one number is picked, nobody receives anything. However, there is a significant catch - there is no way for any two people to communicate. How would you maximise your chances of winning the money? This type of situation is commonly referred to as a tacit coordination game [1, p. 54][2].

The only difference between the presented options is the label attached to each of them. In the previous example we used numerical labels, but they can be qualitative (e.g. a colour), or a combination of the two (e.g. a place and time). If you are to stand a better than random chance of winning, you and your fellow players require a common affinity towards one of the options. As Schelling puts it, your strategy requires 'some imaginative process of introspection, of searching for shared clues' [1, p. 96]. Schelling argues, with the help of various examples and experiments, that humans do perform much better than expected in these types of games. People are able to draw some common signal from the options, and the salience of one (or more) of the options increases the likelihood with which people choose the same option. An option that is more prominent or salient than the others is a focal point [1, p. 57].

Much research has been carried out to investigate the formation of focal points. As we will see in Chapter 2, most of the focus has looked at exploiting certain properties about the options presented to players in coordination games. The objective of this project is to focus on focal point emergence through social construction, achieved through the creation of a simulator and the carrying out of various experiments.

Social constructs can be physical or beliefs; the key idea is that a social construct is shaped by culture, time, and environment [3]. They won't have existed without people, they don't need to exist, and under different circumstances, in a different society, they may exist in a completely different form [4]. One example would be the idea of unlucky numbers - in Western cultures the number 13 is unlucky, whereas in China 4 is unlucky [5].

Focal point emergence through social construction provides an alternative to many of the coordination strategies typically considered in the field of Distributed Artificial Intelligence. In situations where artificially intelligent agents need to autonomously coordinate their action, focal points offer a potential solution. If coordination is expensive or impossible, or if a decision time frame is particularly short, the ideas explored in this project form the basis of potential coordination mechanisms.

We will begin by exploring some related literature in Chapter 2, as well as introducing some basic game theory. Chapter 3 specifies the project requirements, in particular the simulator requirements, which informed the design outlined in Chapter 4. The simulator implementation is described in Chapter 5, while Chapter 6 discusses how the simulator was tested. Many different experiments were conducted using the simulator, and these are outlined in Chapter 7. Chapter 8 takes a look at the project as a whole, evaluating both the simulator and the various coordination mechanisms used in the experiments. Conclusions and potential future work are discussed in Chapter 9. Finally, Chapter 10 contains a guide for users wishing to use the simulator for their own research.

Chapter 2

Background

2.1 Introduction

A group of people are presented with the following numbers:

34 42 49 53 68

Everyone picks one number, and if they all choose the same number, everyone gets the same prize. Consider the idea that the group of people are mathematicians. Perhaps the most obvious choice would be 53, as it is the only prime number. If the group of people were instead all big fans of *The Hitchhiker's Guide to the Galaxy*, the most obvious answer might be 42. What if the group are both mathematicians and fans of the classic science-fiction book? Then the players might be drawn towards both 53 and 42. How are they to separate the two options? If the game was being played at a maths conference, then 53 might become the more obvious choice. In these examples, it is conjectured that the players' environment and surroundings are influencing their decisions. Furthermore, there are psychological factors at play - 'because we are at a maths conference, I believe the other players are more likely to choose 53, and they think that I will pick 53, which makes them more likely to pick 53 etc.'. The game is essentially centred around identifying an obvious choice, which then becomes increasingly obvious as players begin to guess which option other players will pick.

A related idea was used in 1936 by Keynes to to explain fluctuating prices in equity markets [6, p. 156]. His work introduced the concept of the Keynesian beauty contest, in which readers of a newspaper choose the six prettiest faces from hundreds, and the winner is the competitor whose choice is closest to the average preference of all competitors. In this instance, the reader needs to disregard their own opinion, and instead only consider the expected opinions of others. An example of a tacit coordination game that draws from this example would be a group of friends choosing where to meet up, without communicating. Perhaps the given options are the cinema, park, and a local bar. One cannot simply go to their favourite place, but instead must consider where their friends are most likely to go.

The fundamental idea that is consistent across all the given examples is that the players must reach consensus without communication, which is achieved by identifying focal points - courses of action, or choices, that all players converge on. The objective of this project is to explore how these focal points are formed, and how agents can identify focal points in simulation.

2.2 Game Theory

2.2.1 Strategies

In game theory, a strategy is a course of action which a player may choose to take. A pure strategy is where a player plans on playing one strategy with certainty (i.e. there is a 100% chance they will play that strategy), whereas a mixed strategy is where a plan involves playing a set of strategies with fixed probabilities (e.g. 50% chance of playing strategy 1, and 50% chance of playing strategy 2) [7].

2.2.2 Nash Equilibria

When considering pure strategies, a Nash equilibrium is a set of strategies such that no single player can change their strategy to increase their utility if all other players stick to their strategies [8]. All finite games have at least one Nash equilibrium point [9]. In addition to pure strategy Nash equilibria, there is also the concept of mixed strategy equilibria. This is where at least one player is playing a mixed strategy, and no player can improve their expected utility by solely changing their strategy [10, p. 391].

2.2.3 Pareto Efficiency

An outcome is Pareto efficient if nobody can be made better off without making somebody else worse off; all outcomes that result in somebody being better off are at the expense of somebody else [11].

2.3 Theoretical Analysis of Tacit Coordination Games

We will first consider a two-player tacit coordination game. When presented with a number of options, both players receive a reward if and only if both players choose

		Player 2	
		A	B
Player 1	A	$(1,1)$	$(0,0)$
	B	$(0,0)$	$(1,1)$

Figure 2.1: Payoff matrix for 2 player game with 2 options

the same option. The simplest case would be a game with only two options, which we will label A and B.

2.3.1 Payoff Matrices

Figure 2.1 shows the payoff matrix for this simple game. If both players choose the same option, they receive a utility of 1, else they receive a utility of 0. It is important to note that the utility received if the players choose the same option is identical regardless of the option chosen. If, for example, the utility in the top left was $(2,2)$, both players would choose option A. With the current payoff matrix, the only way a player can choose between the options is based on the labels, A and B.

The game can easily be extended to increase the number of choices, whereby the number of rows and columns in the payoff matrix are increased, while maintaining a utility of $(1,1)$ along the leading diagonal, and $(0,0)$ elsewhere. The theory can also be extended to games involving $n > 2$ players, with an n-dimensional payoff matrix.

2.3.2 Nash Equilibria

Given a payoff matrix for a two-player game, pure strategy Nash equilibria can be identified by the following rule: if the first number of the utility is the highest in the column, and the second number is the highest in the row, the point is an equilibrium point [12]. Consequently, in pure tacit coordination games, every point at which all players choose the same option is a Nash equilibrium point. Therefore, in the example in Figure 2.1, the two pure strategy Nash equilibrium points are (A,A) and (B,B).

In addition, there is one mixed strategy Nash equilibrium: (0.5,0.5), (0.5,0.5). This means both players play strategy A with probability 50%, and strategy B with probability 50%, and the two players are acting independently. Each player has an expected utility of 0.5. If just one player changes their probabilities, e.g. player 1 switches to the mixed strategy (0.6,0.4), the expected utility remains the same. Indeed, there is no way for one player to increase their expected utility if the other player sticks with the (0.5,0.5) strategy, hence (0.5,0.5), (0.5,0.5) is a mixed strategy Nash equilibrium.

2.3.3 Pareto Efficiency

Once again using the simple two-player, two-option example, there are two Pareto efficient outcomes, (A,A) and (B,B). With these strategies, there is no way to make either player better off, hence they are Pareto efficient. Conversely, (A,B) and (B,A) are not Pareto efficient, as there are outcomes where both players are better off. One may note that in this simple example, the pure Nash equilibria points coincide with the Pareto efficient points. This holds as the number of players and choices increase.

2.4 Expected Utility

We begin by developing the baseline expected utility assuming all players choose randomly using a uniform probabilistic choice function - given C choices, the probability of any specific choice being chosen by a player is $1/C$. In a game with N players, the probability of all players choosing the same option is $P = C^{1-N}$. If the payoff is U when all players choose the same option, and 0 otherwise, the expected utility for uniform random players is UP. Given 3 players playing a game with 3 options, and a payoff of 10 for total coordination, the probability of success is $1/9$ and the expected utility is $10/9$.

Through social construction, players (agents) should be able to increase the value of P from its baseline, therefore resulting in a higher expected utility for all players. Using mixed strategies to model the actions of each player. Let P_{ij} be the probability the ith player chooses the jth option. The probability that all agents choose the same option can be calculated using (2.1).

$$P = \sum_{j=1}^{C} \prod_{i=1}^{N} P_{ij} \tag{2.1}$$

If for the same game 3 players are all playing with the mixed strategy (0.8,0.1,0.1), where the first option appears to be a focal point, the value of P is 0.514, and the expected utility is 5.14, an increase of 362.6% compared to the baseline example.

2.5 Solving Coordination Games

In traditional game theory, rational players are expected to select one of the Nash equilibrium points from the set of equilibria [13]. However, in a pure coordination game where the utility is identical for all equilibria, how is a player supposed to select one? Harsanyi and Selten [14] claim that the only rational solution is to play a uniform mixed strategy, where the probability of picking any of C choices is 1/C. Whilst this may be true from a game theory perspective, humans perform much

		Player 2		
		A	B	C
Player 1	A	$(5,5)$	$(0,0)$	$(0,0)$
	B	$(0,0)$	$(4,4)$	$(0,0)$
	C	$(0,0)$	$(0,0)$	$(5,5)$

Figure 2.2: Payoff matrix for 2 player game with 3 options

better than expected at coordination games, therefore there must be something not captured by game theory [1]. The only thing humans can use to distinguish between equilibria is the labels associated with them. These labels play a significant role in the formation of focal points.

2.6 Formation of Focal Points

There are various theories as to how focal points form. At this point it is important to consider the difference between one-shot and repeated games, as the means for coordination differ between the two. In a one-shot game, the players enter the game with their knowledge about the world pre-established, but with no prior knowledge of the game in any sense. In this situation, the only means of coordination is through the salience of labels. In repeated games, players have the opportunity to learn from previous outcomes and behaviours, increasing the likelihood of coordination over time.

2.6.1 Salience / Signal

Schelling proposes that players perform better than random at tacit coordination games partly due to the 'signal' of different choices [1, p. 303]. The players need an excuse or reason to converge on one of the options. One of the equilibrium points needs to be 'better, or more distinguished, or more prominent, or more eligible' than the others. If there is no clue or rule that would suggest alternative action, all players should choose the most 'prominent' or salient option. The most salient option need not be uniquely good, indeed it could be uniquely bad [15]. Figure 2.2 shows a game where the most salient option for both players appears to be B, even though (B,B) is not the best equilibrium for either player.

Mehta et al. [16] introduce the concepts of *primary salience* and *secondary salience*. A label has primary salience if through some process it comes to a player's mind. A label has secondary salience if a player expects it to have primary salience for another player. This concept differs from Schelling's idea of salience which describes strategies that become obvious specifically when players are looking for a solution to a coordination problem.

2.6.2 Level-n Theory

There are psychological reasons why a player might choose a certain option: I'm going to choose this because I believe they will choose it because they think I will choose it... This concept was formalised by Lewis [15] in the form of *higher-order expectations*. A first order expectation is simply an ordinary expectation about something. An $(n+1)$th order expectation about something is an ordinary expectation about another person's nth-order expectation about that thing. For example, if I expect you to expect it will rain, I then have a second-order expectation that it will rain.

In the context of a coordination game, it is possible for players to replicate each other's reasoning. That is to say, a second-order expectation is the result of a player replicating another player's first-order reasoning. With the example in Figure 2.1, player 1 may try to replicate the reasoning of player 2. As a result of this replication, player 1 may expect player 2 to expect player 1 to choose A. From this, player 1 can derive a first-order expectation that player 2 will choose A. This first-order expectation is all player 1 needs to choose their course of action. To solve a coordination game, players can evaluate higher-order expectations, acquired through interactions with the world, and arrive at a first-order expectation about other players' actions, which will inform the player's choice [15].

This idea was developed further by Stahl and Wilson [17] with the concept of the *level-n theory*. The theory states that players can perform two actions: form priors about other players (predict their behaviour), and choose the best strategy in the context of those priors. The only way players can differ is in their priors or how they act on those priors. In a pure coordination game with multiple Nash equilibria, the way in which players act on priors is irrelevant if all players are capable of computing the most common choice and choosing that. Therefore, we are more interested in the players' priors.

The level-n theory is a recursive model that characterises players based on their priors. A level-0 player disregards all other players, and plays according to a uniform probability distribution. We used level-0 players to calculate baseline expected utility. Level-1 players believe all other players are level-0 players, and a level-2 player believes all other players are level-0 and level-1 players. A level-n player believes all other players are level-0 to level-(n-1).

One derivation of the level-n theory is the *cognitive hierarchy (CH) theory* [18]. The theory assumes all players believe they are the best (level-k), and all other players are levels 0 to k-1, distributed according to a normalised Poisson distribution. Experiments have shown the model to be very effective, particularly in games related to the Keynesian beauty contest.

Both higher-order expectations and level-n theory can be used to explain the formation of focal points. Whilst both definitions are recursive and can therefore continue infinitely, humans have a boundary to their level of reasoning. If players are acting based on their expectations of others, 'coordination may be rationally achieved' [15].

2.6.3 Team Reasoning

Bacharach [19] introduces the idea of *team reasoning*, which is where each member of a group determines what action an imaginary leader would prescribe for them. In a sense, each player tries to maximise the group's utility rather than their individual utility. Further work on the topic of team reasoning has been conducted by Sugden [20]. In pure coordination games, where all players must choose the same option, individual and collective utility are interchangeable. Therefore, whilst team reasoning has been used to explain the formation of focal points in some coordination games, it provides no additional insight in the types of games we are interested in.

2.6.4 Variable Frame Theory

Another concept introduced by Bacharach [21] is that of *variable universe games*, leading to the development of *variable frame theory*. It is a model that specifies how players conceive situations, and makes use of the principle of insufficient reason (IR). In an example of choosing one of many blocks, some players may be able to distinguish shape, colour, or material. Players can arrive at a focal point by evaluating their own knowledge and the probability that other players are aware of certain things. For example, given 10 cuboid blocks, 3 may be red, 7 yellow, but only one made of oak, the rest of birch. Take a two-player game where both players are aware of colour. If a player is unaware of material then the best choice is to pick one of the red blocks at random. If a player is aware of material, then the best strategy is to pick the oak block if they believe the other player might also know about material (here the probabilities and expected utilities should be calculated and a rational player will choose the strategy with the highest expected utility). Notably, Bacharach concludes that the theory may be useful when considering the view that the culture of players can influence the rational solutions of certain games [21].

2.6.5 Principle of Individual Team Member Rationality

Building on variable frame theory, Janssen [22] uses the principles of IR and individual team member rationality (TMR) to explain peoples' success at coordination or 'matching' games. Janssen extends Bacharach's theory to consider both *description symmetry*, where strategies have the same label (in some sense), and *pay-off symmetry*, where multiple strategies can't be distinguished even if the descriptions of the strategies are different. The overarching philosophy is that 'TMR acts as an optimization principle and IR acts as a constraint on the set of feasible mixed strategies' [22]. The possibility of players being aware of a concept but not able to use it themselves is also considered.

2.6.6 Convention

Convention is an idea stemming from precedent, which is an important source of salience. Precedent describes the 'uniqueness of an equilibrium because we reached it last time' [15]. Clearly precedent relies on repeated gameplay, although the games need not be unique - players can seek to find an equilibrium that uniquely corresponds to an equilibrium reached previously in a different game. Issues will arise, however, if there are multiple equilibria that correspond in unique ways to a previous equilibrium. Precedent is powerful in so far as it does not matter how previous equilibria were reached, indeed it may have been through sheer luck. Crawford and Haller have shown how two players can use precedent as focal points in repeated coordination games [23].

It is also the case that players need not share examples of precedents they have seen. Lewis uses the example of which side of the road cars drive on [15]. You may have seen almost all cars driving on the left of the road, and another person may have seen the same, yet you may have never witnessed the same car driving on the left, but still you have both become familiar with the precedent of driving on the left-hand side.

Lewis eventually reaches a final definition for convention, which states that a regularity is a convention among a population in a recurrent situation if and only if it is common knowledge that almost everyone conforms to the regularity, almost everyone expects almost everyone to conform to the regularity, and almost everyone has the same preferences regarding all possible combinations of actions [15].

Binmore and Samuelson [24] also explore the idea of convention, particularly in the context of labelling in a pure coordination game. Their idea of convention, such as choosing based on characteristics of the labels, ties in with the idea of following procedural rules, which will be explored later.

2.7 Alignment Mechanisms

In Ober's book, *Democracy and Knowledge: Innovation and Learning in Classical Athens*, he presents four mechanisms for alignment, which gave Athenian's the capacity to take joint action [25]. These mechanisms are:

- first-choice following
- leader-following

- rule-following
- commitment-following

Each of these mechanisms can be evaluated for its applicability to the types of coordination games covered by the project.

2.7.1 First-choice following

The first-choice following mechanism is where one person takes an action, and that single action coordinates the actions of everyone else. Consider diners at a round table, unsure whether to take the bread roll on the left or right. As soon as one person takes, everyone else can follow suit, taking on the same side as the first person [25]. This has achieved coordination without communication. However, such a mechanism is not strictly applicable to the games being studied, as there is no sense of turn taking, or one agent going first.

2.7.2 Leader-following

Leader-following stems from the idea of alignment cascade, most commonly observed in groups of moving animals, such as fish and birds. In a school of fish, the majority are only focused on staying close to the other fish. The movements of a minority, so called 'informed leaders', coordinate the movements of the majority. Most fish are ignorant to the identities of the leaders and the reasons behind their movements, since following is likely to be to their benefit [25]. This idea can be applied to repeated coordination games - an agent can observe and copy the actions of an agent in previous games. This clearly does not guarantee coordination, since an agent's actions will always lag behind those being copied, but if some agents do reach a stable equilibrium, this is a mechanism that would allow the equilibrium to spread through the population.

The most obvious solution to coordination games would be to appoint a leader who dictates the actions taken by all agents. This slightly differs to Ober's mechanism, since a leader in this sense would be in a position of dominance, rather than an agent holding useful information.

2.7.3 Rule-following

Rules are essential in modern society, e.g. in the UK we drive on the left, and breaking this rule can have the gravest consequences. They are effective when they are both simple (easy to learn and follow) and common knowledge - not only does each agent know the rules, each agent also knows that all other agents know (and follow, for the most part) the rules [25]. Institutions in ancient Athens operated effectively thanks in part to simple procedural rules, and such rules can also be used to guide behaviour in coordination games. Convention can be viewed as a type of rule, although differs slightly as deviation from a convention would indicate the action or choice was not a convention in the first place.

One relevant coordination game was explored by Binmore and Samuelson [24], where players must choose one of three options, possibly labelled red or with a square (each with 50% probability). In many scenarios it is easy for players to coordinate, e.g. if two options are labelled with squares, and the third option is labelled red, both players are likely to go with the third option. However, scenarios could arise where the options are labelled with, for example, a red square, an uncolored square, and red with no square - which option do the players chose? Binmore and Samuelson introduce the 'shape-then-color' convention, where players choose the odd-one-out first based on shape, and then if one doesn't exist, choosing the odd-one-out based on color (and if that doesn't exist, choosing randomly). This convention works perfectly unless all choices are indistinguishable. However, it could be argued that this type of convention is more in line with Ober's idea of procedural rules.

2.7.4 Commitment-following

The final mechanism is that of *commitment-following*, whereby players commit to a course of action before the event. These precommitments need credibility if they are to carry any weight, particularly in situations where costs of actions are high [25]. For example, if A wants to invade B, and few of B's people are up to defend, the cost to each of those people is very high (likely death in this instance). However, the majority of B's people believe that defending is the right course of action, but they will only join the defense if they believe that everyone else believes it is the right action. Even that is not enough - the fact that everyone believes it is the right action must itself be common knowledge. Ober argues that if this common knowledge is lacking, it can be fatal for a community's capacity for coordinated action. In the pure coordination games we have considered so far, if agents are aware of the options in an upcoming game, they can commit to a certain course of action, and deviation from this commitment would not be beneficial.

2.8 Monuments and Common Knowledge

Monuments play two key roles in coordination - acting as a focal point, and publicising information pertaining to or influencing the potential existence of a focal point. Firstly, monuments act as physical focal points if people need to gather in the same location. In New York City, for example, both the Grand Central Station and the Empire State Building are key monuments used to coordinate meeting locations. Secondly, as monuments are geographical focal points, they can also bear information such as announcements or news, and one can reasonably assume that most people will see this information. In ancient Athens, the Eponymous Heros monument was a site where lists of those called up for military service would be posted [25]. There was also the tyrant-killer monument, which acted as a gathering point for Athenian democrats during political crises [25].

Monuments displaying information were essential for building common knowledge in ancient Athens. In the coordination games we are studying, common knowledge provides a major solution platform. If the majority of agents are aware of certain facts, perhaps sharing the knowledge that one agent always chooses the same option, and if the agents use that knowledge in similar, productive ways, achieving some level of coordination seems plausible, if not inevitable.

2.9 Experiments

Various experiments have been conducted, to assess the performance of humans at coordination games, to evaluate the effectiveness of various strategies, and to test theoretical models. Stahl and Wilson [17] performed experiments with 3x3 coordination games, and found evidence supporting their idea of 'bounded rationality' (level-n theory). Bacharach's and Bernasconi's [26] experiments confirmed most of variable frame theory to be accurate. Meanwhile, Brandts and MacLeod [27] experimented with the idea of making recommendations to players during gameplay.

2.10 Alternative Games

Up until this point we have only looked at pure coordination games, and predominantly games with only two players. However, situations with more players open up the opportunity for other interesting games with different payoffs. A simple change that can be made is to switch to majority wins rather than consensus. If the majority of players choose the same option, they all receive some utility, and players who went for other options can either receive the same utility, or nothing at all. Conversely, the game could also be switched to minority victories - for example if you are choosing a cafe to go and work at, you might not want to be somewhere particularly busy. These games introduce the possibility for interesting behaviours and dynamics, such as committing to a certain action, then either sticking with it or deviating from it, which in the minority win case makes the outcome very unpredictable. A well designed simulator should be modular and flexible such that exploring these alternative games (and potentially others) is seamless.

2.11 Distributed Artificial Intelligence

Distributed Artificial Intelligence (DAI) is a sub-field of Artificial Intelligence, concerning distributed agents or processes solving a given problem [28]. The types of coordination mechanisms and theories explored in this chapter can be applied to real world DAI applications. Parunak describes the potential uses of multi-agent systems in the manufacturing industry, including various case studies [29]. Kraus et al. consider autonomous agents working on Mars [30]. Jennings proposes that coordination is the key problem in DAI, and argues that commitments and conventions are the ‘foundation of coordination in all DAI systems’ [31].

Chapter 3

Requirements

3.1 Deliverables

The primary deliverable of the project is a self-organising multi-agent simulator in which agents play tacit coordination (focal point) games, and learn socially constructed reasons for one option to be more salient than others. A series of experiments will be run on the simulator, and both the simulator and experiments thoroughly evaluated.

3.2 Simulator Requirements

The simulator is intended for use by researchers, and should therefore be designed with significant configurability. Additionally, the code structure should make it easy for researchers to drop in their own code as desired.

The final simulator will be evaluated against the following requirements:

1. Support for coordination games with up to 100 agents
2. Intuitive user interface (UI) to configure simulations
3. Easy to switch between different games
4. Clear visuals for clusters and game outcomes
5. Runs smoothly at $\geq$ 24fps on a mid-range laptop under ‘normal’ load
6. Logs game data
7. Clearly shows key data and statistics
8. Simple to add to the code

For some requirements, objective tests can be used for evaluation (e.g. measuring the frame rate). However, other requirements rely on subjectivity (e.g. is the UI intuitive?). For such requirements, reasoned arguments will be made during the evaluation phase.

3.3 Experiments Requirements

The objective of the experiments, which will make significant use of the simulator, will be to explore coordination through social construction. As a point of comparison, a baseline will need to be established, using agents choosing randomly. Then, various different agent implementations can be tested, each with the objective of out-performing the random agents. The implementations will be largely informed by the background material.

The experiments will clearly state the dependent and independent variables, so that it is explicitly clear what is being tested. The results can then be compared to the baseline, as well as hypotheses that will be formulated prior to running the experiments. The agent implementations will then be critically evaluated based on the agent performance.

Chapter 4

Simulator Design

There are many different approaches that could have been taken when it came to designing the simulator. This section will walk through the design choices and trade-offs that were made throughout the process.

4.1 Simulator Overview

The most important design decision to make early on in the project was to define what was going to be simulated, and specify the overall game flow. The simplest type of focal point game simulator would be to play a one-shot game, with agents asked to choose from an array of options, and if all agents choose the same option, they 're-ceive' a reward. Such a simulator would have been trivial to implement, but would massively restrict the possible agent behaviours and patterns that could emerge in the data. Indeed, coordination through social construction is much less likely to be achieved with one-shot games. Therefore, it was decided early on that the simulator should feature repeated gameplay, as it would allow for significantly more interesting experiments, while only requiring a marginal increase in the complexity of the simulator. Additionally, the number of games played would be configurable, so that the option of playing one-shot games remained.

It was also decided that there should be some variability in the agents playing the focal point games. One potential solution was for the simulator to create N agents, and in each game randomly select $< N$ agents to participate. If there were many agents left out each time, this would lead to a large variation in the agents selected to play, which again would increase the scope of possible experiments (particularly if there are agents playing different strategies). However, with the goal of witnessing coordination through social construction, randomly selecting the agents would not have well represented the behaviours we see in society (recall the example of people playing a game at a maths conference - they have not been randomly selected, but

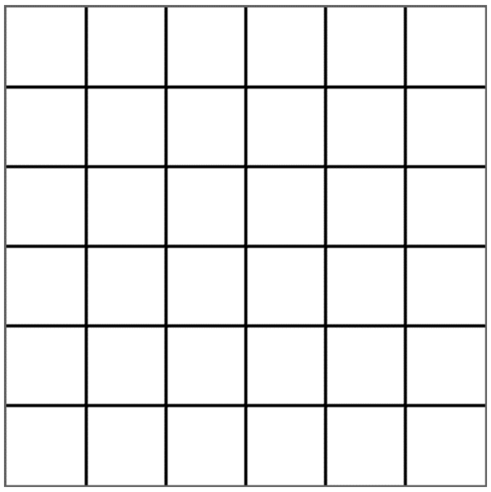


Figure 4.1: Example simulator grid

have instead come together because of a common interest). It was quickly decided that an alternative to random selection was needed, which led to the idea of the agents operating within some space.

Instead of agents existing just at a location in memory, at any given point they would also be at a location within a simulated world. To keep things simple, but without limiting the experimental possibilities, this world would be a square grid (an example is shown in Figure 4.1), with each square assigned a set of 2d coordinates. The agents would be able to move around the grid, opening up near endless possibilities for the types of interactions that could unfold. Agents would be able to meet other agents and, hopefully, socially construct ways of coordinating in the focal point game. Because of the spacial element, it was logical to select the agents that would participate in the focal point game based on their locations. One option was to define a part of the grid where the game would be played, and at regular intervals any agents in that area would be selected to participate. Alternatively, the game area could change throughout the course of a simulation, increasing the probability that different agents are included.

All options explored to this point have one thing in common - there is only ever one focal point game taking place. This was an unnecessary restriction placed on the simulator, and as with the case of repeated gameplay, running multiple games in parallel adds little code complexity, although increases the computational demands of the simulator. Therefore, it was decided that multiple games would be played in parallel (the number would be configurable), with the selection process based on the agents' locations. This begged the question: should all agents be included in a game, or should some be left out? Leaving out a number of agents would once again increase the possibilities, but at this point the total experiment space was already large, so it felt like unnecessary added complexity. In summary, agents would exist on a square grid, would be free to move around, and at regular intervals would be separated into groups based on their location, and each group would play a focal point game.

4.2 CLI vs GUI

The simplest form of a simulator, implementing the gameflow previously discussed, would be a text-based command-line interface (CLI). Almost any programming language could be used, and all effort would be focused on implementing the mechanics of the simulator, and the experimentation that would follow. However, it would be difficult for a CLI simulator to successfully convey all relevant information to the user. The grid could be drawn to show the agents' locations, but beyond that it would be hard to effectively show the groups of agents playing games, and the results of the focal-point games, without resorting to verbose text. Therefore, a graphical user interface (GUI) based simulator would be much more user-friendly, and would allow for all relevant information to be neatly displayed. The effort required to implement the simulator would vastly exceed that of a CLI simulator, however the experimentation phase of the project would be much quicker and more insightful as a result.

Choosing to implement a simulator with a GUI limited the number of technologies that could reasonably be used, but there were still ample options. The eventual choice was to use Processing to create the simulator, the reasoning for which will be outlined later.

4.3 Definition of Configuration Variables

The simulator consists of a GxG grid, with N agents moving around. Each cell of the grid has occupancy O, which is the maximum number of agents that can be in a single cell at any given time. Each simulation is split into R rounds, and in each round an agent can move M times (each time agents can move by zero or one cells). Agents can move to cells that are horizontally or vertically adjacent and are not full. After M movements, each round culminates in a subgame. The agents are divided into K groups, and each group plays a tacit coordination game with C choices. The game continues on to the next round or finishes after R rounds.

Some of the above variables are subject to certain constraints, both for graphical and logical reasons. As such, the order in which the values are verified matters. The constraints, in the order in which they are applied, are:

1. $G \in [3..20]$
2. $G^2 \geq N \geq 1$

3. $R \geq 1$
4. $4 \geq O \geq$ ceil$(2 * N/G^2)$
5. min(ceil$(N/O), 6) \geq K \geq 1$
6. $C \geq 1$
7. $M \geq 0$

Constraint 1 exists to ensure that the grid remains clearly visible - a 100 x 100 grid, for example, would be completely unrealistic to visualise on most displays, particularly as the agents need to be distinguishable within each cell. Constraint 2 ensures that the grid isn't overpopulated with agents. The maximum value of O is 4, because the computation to display 1, 2, 3, or 4 agents evenly spaced in a single cell is not too demanding, whereas for 5 or more it becomes significantly more complex. The minimum value of O ensures agents are likely to have movement options available. The maximum value of K is the lower of 6 and ceil(N/O). So that each group playing a focal point game is easily identifiable, the agents within a group are all coloured the same, and no two groups have the same colour. Beyond 6 groups it becomes harder to distinguish between the groups (as with more colours, the more alike they become), which is the intuition behind restricting K to a maximum of 6. This condition on its own is not sufficient, since there should not be more groups than agents. Additionally, it was decided that all agents in the same cell on the grid should be assigned to the same group. Therefore, the ceil(N/O) term ensures that all groups should have at least 1 agent.

4.4 Early Simulator Design

4.4.1 Simulator State Machine

As the simulation had a very structured order of events, it made sense to construct the simulator as a state machine. There were 7 states that the simulator could take, as seen in Figure 4.2 (m is the number of moves remaining, r is the number of rounds remaining).

These states were:

- **START** - The first state that the simulator would enter, where any initialisation would occur. It would check to see if any rounds are remaining, or if the simulation had concluded.
- **MOVE_DECISION** - This is where each agent chooses which location to move to from a list of permitted options.

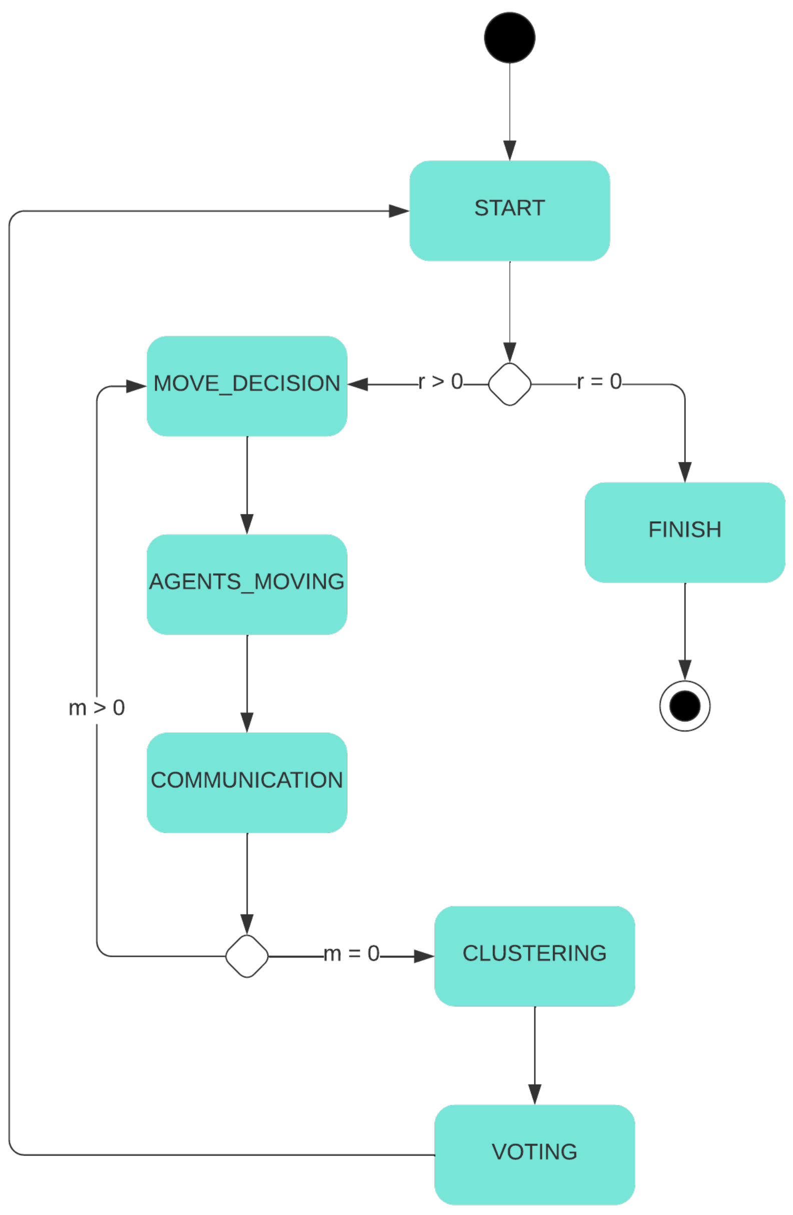


Figure 4.2: Early simulator design state diagram

- **AGENTS_MOVING** - This state would be used to keep track of the agents while they are moving.
- **COMMUNICATION** - During this state, any agents in the same cell on the grid would be able to communicate with one another. If the agents have moves remaining, the simulator would then go back to the MOVE_DECISION state, otherwise it would proceed to grouping the agents.
- **CLUSTERING** - The agents would be grouped, or clustered, into K clusters.
- **VOTING** - This is where the actual focal point games would take place. The agents would be asked to pick from a set of options, the results would be calculated, and then distributed to the agents. If there were rounds remaining, the simulator would go back to the MOVE_DECISION state, otherwise the simulation would end.
- **FINISH** - After all rounds had been completed, the simulator would end in this state.

4.4.2 Software Components

The software implementation of the simulator was very much designed with Processing in mind, making the most of its key features and libraries to simplify the design wherever possible. Processing uses Java at its core, and so the overall program structure was object-oriented. Therefore, assuming it has the necessary graphical capabilities, such a design could be ported to another language.

Figure 4.3 shows the class diagram at the highest level. A single configuration object would be created, which would store all of the variables defined in section 4.3. The server class would control the logic of the whole simulator, implemented as a state machine, with the logic for each state also implemented. A graphics handler class would be responsible for controlling all the visuals - drawing the grid, the agents, and showing the results of both clustering and the focal point games that would follow. This class would obtain all necessary information from the configuration and server objects, as well as the agents. There would be a separate agent class, instantiated N times.

4.4.3 The Focal Point Game

The focal point game played by the agents was purposefully very simple. The agents would be presented with a set of numbers, 1 to C, and if a consensus was reached (if all agents chose the same number) they would win. If just one agent chose differently, all the agents in that cluster would lose. There is a rich design space for the games that could be explored, both in terms of the win criteria, and the types

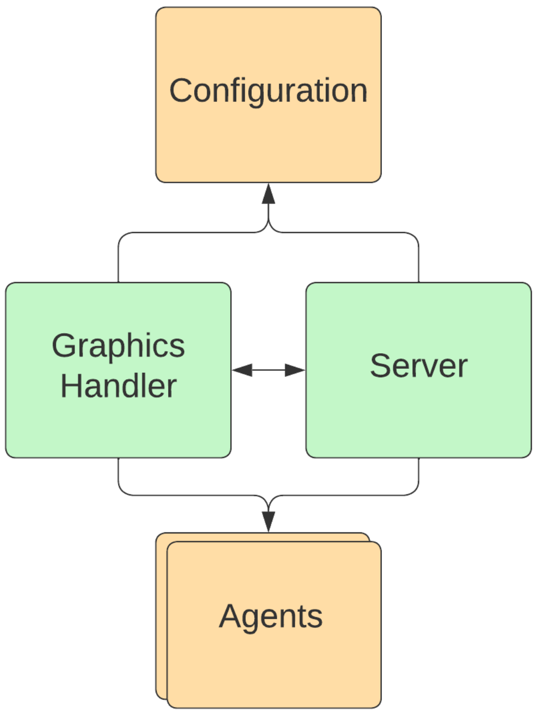


Figure 4.3: Early simulator class diagram

of options presented to the agents. Previous work has mostly centred around distinguishing between objects based on certain features or properties [21, 30]. Such choices could also have been replicated in the simulator. However, the mechanisms for focal point discovery presented in such papers rely on physics rather than social construction. Therefore, implementing a similar focal point game would have added extra complexity with little reward (the agent implementation could completely disregard all social mechanisms previously discussed).

4.4.4 Graphics

The early version of the simulator kept the graphics clean and simple, but with huge scope for improved functionality. As seen in Figure 4.4, the simulator consisted of the grid on the right, containing many agents represented by circles. The figure shows the simulator in the VOTING state, after the results have been distributed to the agents. The different colours represent the different clusters, and in this particular example the outlines of the agents are red because they were all unsuccessful in coordinating choices. Additionally, on the left there were two buttons, a 'play'/'pause' button and a 'restart' button.

It was clear that the simulator design would meet many of the goals outlined earlier,

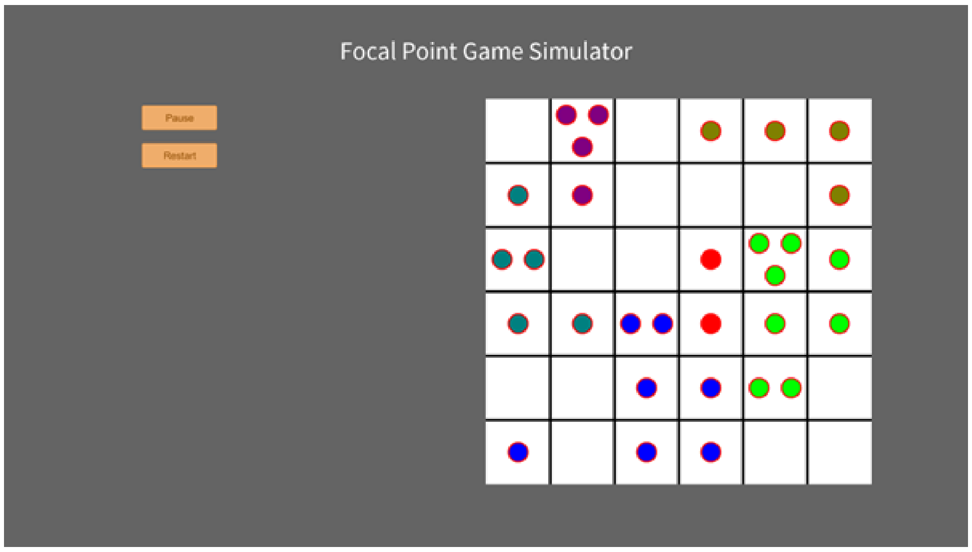


Figure 4.4: Early simulator graphics

however there were significant shortcomings, including, but not limited to:

- Experiments had to be configured in the code rather than via the GUI
- There was no way for some potential coordination mechanisms, such as monuments, to be tested
- The simulation results were only communicated via the colour of the outline of the agents - the only way of obtaining any statistics was through manual counting/observation of the results

The overall idea behind this design was that once the core infrastructure was in place, all later development would be on the agent side of things - deciding when and where to move, which agents to communicate with, what to communicate about, how to choose in the focal point games etc. However, with so many opportunities (and needs) for improvement, a more refined design was settled on.

4.5 Final Simulator Design

4.5.1 Changes to the Early Design

The final simulator design resolves the biggest issues with the earlier iteration. A big quality-of-life improvement was the introduction of text fields to the GUI, from which experiments could be configured. A subset of these fields can be seen in Figure 4.5. An additional limitation of the early design was that whenever the simulator was run, a simulation would immediately start. This is no longer the case - the user is required to manually press a start button, preventing the simulation from starting prematurely.

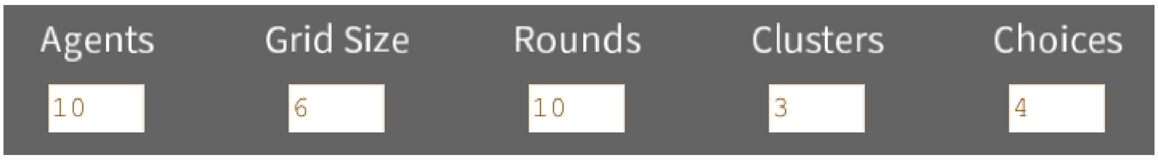

Agents 10 | Grid Size 6 | Rounds 10 | Clusters 3 | Choices 4

Figure 4.5: A subset of the text fields

The biggest visible difference was the introduction of an information and visualisations panel. This displays text while the user is configuring an experiment, error messages where applicable, and a range of graphs that update in real time as the simulation progresses. The graphs allow the user to quickly read off the headline results of an experiment, such as the overall win rate, occurrences of clusters of particular sizes etc.

One less visible (but equally useful) addition was a data logger. This creates a new log file for every experiment run, logging the configuration variables, grouping and choices of agents during each round, as well as various interesting and useful statistics. The logs created use JSON, and can therefore be easily read by other programming languages, e.g. Python for use with matplotlib to quickly create additional graphs.

Various other major changes centred around increasing the types of experiments that could be conducted, and the variety of coordination strategies that could be implemented. The first of these was the introduction of monuments. It became apparent during the background reading that monuments were central to the achievement of coordination in ancient Athens [25]. However, implementing such a mechanism could not be achieved at an agent level, due to both the visualisations required and the logic that would need to be controlled by the server (such as determining which agents can view which monuments at any given point). As the server was in control of the monuments, the ability was added for monuments to move, akin to mobile landmarks used for localisation in wireless sensor networks [32]. This addition necessitated changes to the simulator state machine. The monuments are blank canvases on which the agents can write/draw symbols. The monuments, like the agents, are always in a particular cell of the grid, and can only be written on by agents in the same cell. Additionally, the monuments have a configurable visibility variable, which defines the radius of the circle, centred at the monument, within which agents can view the monument. Figure 4.6 shows a grid containing 6 monuments, each visualised as a black monolith.

The big change that increased the types of possible experiments was the introduction of districts. A district is a defined rectangle within the larger grid. If at least one district exists, the entire grid must be covered in districts. The districts are not allowed to overlap. These design choices were made to simplify the implementation, while still affording the user lots of flexibility. Districts were purposefully designed

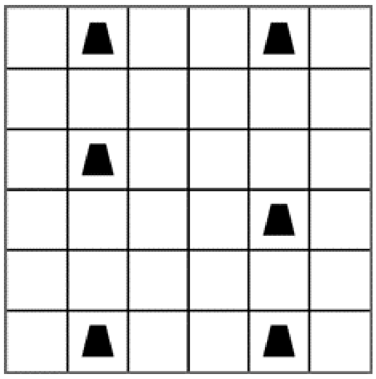


Figure 4.6: A grid with 6 monuments

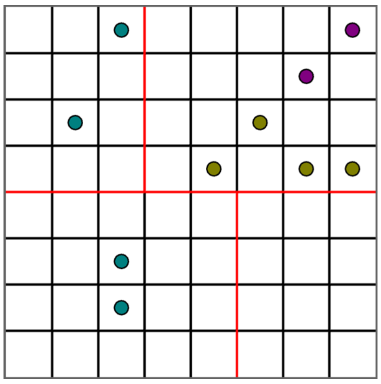


Figure 4.7: A grid with 4 districts

to add information to the grid, rather than to create hard boundaries at the server level. The idea is that agents know which district they are currently in, which district they were spawned in, as well as the districts of the squares they can move to. If the user wishes for an agent to stay within a certain district, this can be implemented at the agent level. With this complexity pushed on to the agent side, it kept the core simulator infrastructure simpler. During a simulation, the boundaries of the districts are denoted with red lines (rather than the typical black lines), as shown in Figure 4.7.

The addition of both districts and monuments allowed further possibilities, namely *localised monuments*, and *district clusters*. These work in the following ways:

- Localised monuments - When the localised monuments option is selected, monuments are only visible from the districts they are located in. For example, if an agent is in district 1, and a monument is in district 2, even if the agent is within the normal visible field of the monument, it won't be able to view it. In essence, this option represents building walls at district boundaries, but with doors or passageways so that agents remain able to travel between districts.
- District clusters - When this option is selected, the simulator no longer uses a clustering algorithm to determine the groups of agents, but instead, for each

district a focal point game is played by all agents in that district. With this setting enabled, the value of K is overridden. In addition, whilst K is ordinarily restricted to a maximum value of 6, if district clusters are used this maximum value does not apply. This is because colouring the agents by cluster is not necessary with district clusters (as the clusters are clearly denoted by the district boundaries), so all agents are coloured the same, and the issue of having many similar colours is prevented.

Enabling of these options is subject to the following prerequisites:

- Localised monuments - the grid must be divided into districts, and there must be at least one monument.
- District clusters - the grid must be divided into districts.

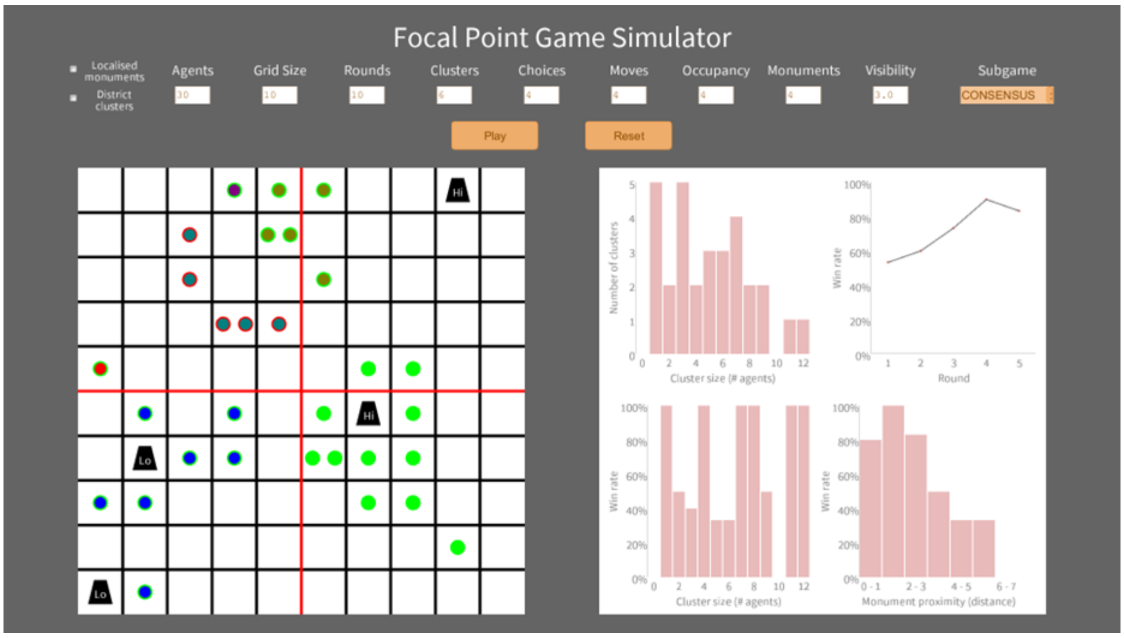
During simulation configuration these requirements are checked, and if they are not met, the issue is clearly conveyed to the user via the information and visualisations panel.

We define additional variables to track these new additions:

- L - the number of monuments. As M was already taken, L, short for 'landmarks', was the next logical choice. It is subject to the following constraint to limit the number of monuments to a reasonable level: $G^2/2 \geq L \geq 0$.
- V - monument visibility. This defines the maximum distance from which a monument can be viewed. It is subject to the following constraint: $(G-1)\sqrt{2} \geq V \geq 0$. Intuitively, the furthest away from a monument an agent can be is in the opposite corner, and the distance between the two corners is $(G-1)\sqrt{2}$.
- D - the number of districts.

A limitation of the previous design was the inflexibility of the focal point games - the results function always required consensus. The new design added the ability to experiment with different types of games, for example requiring a majority rather than consensus for success. A simple drop down menu was added to the GUI to select between different game types.

The final noticeable change from the early design was to allow the user to custom place agents (and monuments). In the previous iteration, the agents would always be randomly located around the grid. Now, the user is able to place the agents wherever they like (in any cell with $< O$ agents) through a simple click of a mouse. The same concept has been applied to allow the user to place monuments. Any unplaced agents and monuments are then randomly added by the simulator. The creation of districts is slightly more involved, although still really simple. The user can click in a cell, drag the mouse to a different cell, and release. The start and end cells denote two corners of a district, and as long as the district doesn't overlap


Focal Point Game Simulator

Localised monuments / District clusters	Agents	Grid Size	Rounds	Clusters	Choices	Moves	Occupancy	Monuments	Visibility	Subgame
	30	10	10	6	4	4	4	4	3.0	CONSENSUS

Figure 4.8: Final simulator GUI

with any existing districts, it is added to the grid. The final simulator is shown in Figure 4.8.

4.5.2 Simulator State Machine

All of the changes made to the simulator necessitated the addition of some new states to the state machine:

- CONFIGURATION - During this stage the user is able to configure an experiment by interacting with the GUI.
- PLACING - In this state the user is able to place agents and monuments, and create districts.
- MONUMENT_MOVE_DECISION - In this state the logic used is very similar to the MOVE_DECISION state, except it is the monuments deciding if and where to move, rather than the agents deciding. This state is entered after the conclusion of a focal point game, as long as some monuments exist.
- MONUMENTS_MOVING - This state controls the visual movement of monuments, and is again only entered if there are monuments in existence.

The final simulator state machine is visualised in Figure 4.9. The additional states are coloured orange, while the existing states are shown in green. All state transitions are triggered automatically with the exception of CONFIGURATION to PLACING and PLACING to START, both of which occur when the user clicks on a particular button.

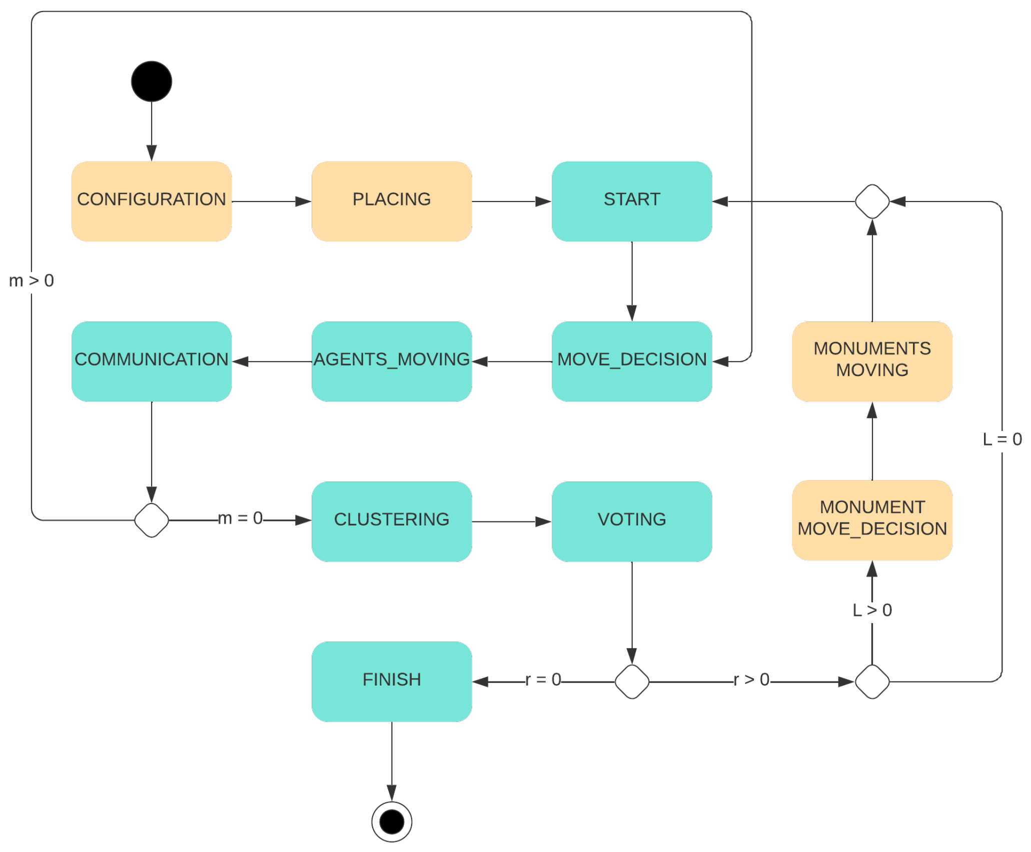


Figure 4.9: Final simulator state machine

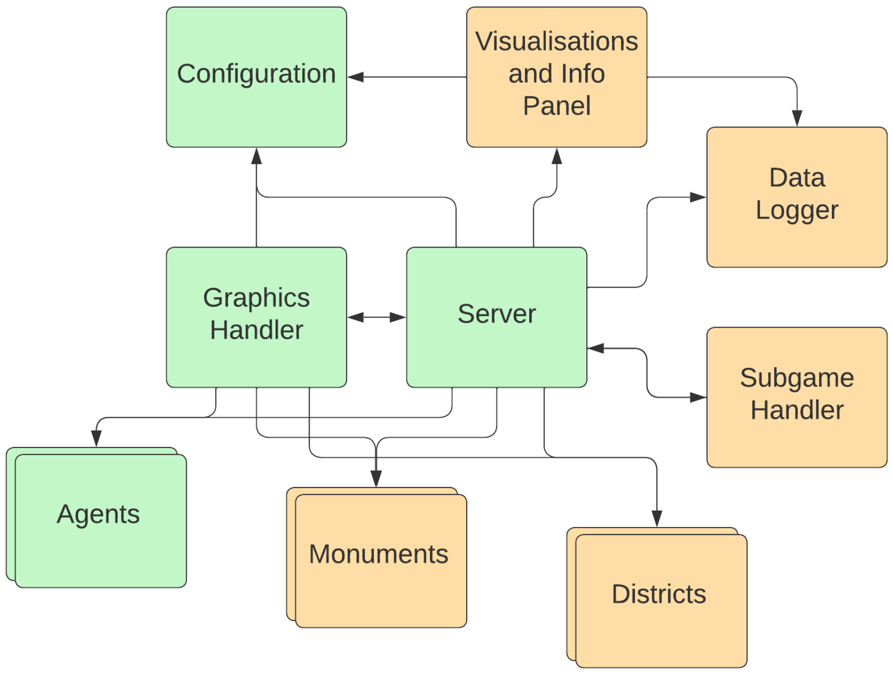


Figure 4.10: Final simulator classes

4.5.3 Software Components

The added functionality also required additional classes. The major classes are shown in Figure 4.10. The newly added classes are shown in orange. The subgame handler class was added to allow the user to easily create, and experiment with, their own reward functions. All other classes implement the elements and functionality discussed in this section.

Chapter 5

Simulator Implementation

5.1 Languages, Tools, and Practices

Various different languages and frameworks were investigated and evaluated for their suitability for the project. The first approach was to create a web-app with a framework such as React, a JavaScript library commonly used for creating user interfaces [33]. Whilst it is great for making interactive UIs, it is less easy to create a constantly updating simulator with moving agents. One major positive is that web frameworks have significant online resources and support.

Another potential option was to use Qt and C++ (or alternatively PyQt and Python). Qt is a software development framework for developing cross-platform applications [34]. Whilst it promises highly readable, maintainable code and high runtime performance, it is mainly used to create industry grade applications, which wasn't the objective of the project.

The option that was eventually chosen was to use Processing, which is a software sketchbook and language based on Java [35]. It has a low barrier to entry, and allows the user to harness the full power of Java while simultaneously providing an easy to use sketchbook API. There are also various libraries that have been developed, for GUIs, I/O etc. This all made Processing the perfect choice for creating a multi-agent simulator with a simple GUI.

With Processing the language of choice, the Processing Development Environment (PDE) was chosen for the IDE, as it consists of a text editor, compiler, and a window to display the simulation. It also allows sketches to be exported as applications. For source control, Git and GitHub have been used, along with the GitHub desktop application. All code was, where possible, written to conform with the *Google Java Style Guide* [36].

5.2 Implementation Overview

The overall implementation ideology was to utilise Processing's libraries and follow object-oriented programming principles. Additionally, as it needed to be easy for any potential users to conduct their own experiments, not only did the interface have to be intuitive and comprehensive, but the code also needed to be easily extendable.

The implementation can broadly be split into two parts:

- The game infrastructure
- Agents and mechanisms

The infrastructure consists of a number of classes that work together to implement all of the simulator logic, in addition to handling all of the graphics. The agents and mechanisms (e.g. monuments) have base classes which implement default behaviour, and these classes can be extended by the user, through the creation of child classes, to implement their own agents and mechanisms for experimentation.

Processing programs typically implement two functions: the `setup` function, and the `draw` function. The `setup` function is called once at the beginning of execution, and is used for initialisation (e.g. creating objects, setting the screen size, specifying the frame rate). The `draw` function is called every time a new frame needs to be drawn. A frame rate of 30 frames per second was used for the simulator, since this strikes a balance between smooth visuals and light computation. Therefore, the `draw` function should be called every 33.33ms.

5.3 Key Classes

5.3.1 Config Class

The Config class is used to store all of the simulation experiment variables. Setter and getter methods are used to set and get the variables. When the user has finished configuring an experiment, a Config object is created. As there are some dependencies between the variables, the constructor calls the setters in an order such that all dependencies can be checked. In some instances, it is possible that the user will try to run an experiment with some incompatible values. If this is the case, the Config class will fix any issues, and the new variable values are communicated to the user via the GUI.

5.3.2 Server Class

The Server class is the brains of the simulator. It handles the vast majority of the game logic, and instantiates other classes such as agents, monuments, and the data logger. The class has a `run` method which is called by the `draw` function. An enumeration is used to track the current state (in the state machine), and the `run` method contains a switch statement that calls a corresponding private method for each state. This design choice keeps the implementation for each state separate, and makes it easy to add in additional states, which happened on a couple of occasions during the project.

During experiment configuration, there are a few methods that can be called by other parts of the program. These are `tryAddAgent`, `tryAddMonument`, and `tryAddDistrict`. Each of these methods is called whenever the user tries to place an agent or monument, or create a new district. Various checks are performed to ensure that any attempted placements don't violate any of the simulation configuration variables. Firstly, when the user attempts to place an agent or monument, the current number of agents or monuments is compared to the number specified by the user. If the required number of agents or monuments have already been placed, a new agent or monument won't be added. Secondly, there are limits on the number of agents and monuments that can be in a cell on the grid. The agent cell occupancy is specified by the user, whilst there can only ever be at most one monument in a cell. If a cell is already full, no new agent or monument will be created in that cell.

Another important design consideration was for agents and monuments to only be created in a single place in the code. Therefore, if the user doesn't place all of the agents or monuments, when the remainder are randomly placed, the `tryAddAgent` and `tryAddMonument` methods are also used. This ensures that when a user wishes to use their own agent or monument implementation, this only has to be specified once in the code.

The checks for district creation are slightly more involved, but geometry tricks were used to keep the code simple. Districts are not allowed to overlap, and as districts are rectangular, this can simply be verified by checking that one district is above (or below) or to the left (or right) of the other. The code for this is shown in Listing 1.

The simulator mandates that if at least one district has been placed, the whole grid must be covered in districts. This simplifies internal logic, as each cell will have an associated district and district number. To verify that the whole grid is covered, another simple trick is used that makes use of the fact that each district is rectangular. Districts are represented by two coordinates, the top left and the bottom right. The product of the difference in x coordinates and difference in y coordinates gives the area of the district. As districts cannot overlap, the grid is completely covered if and only if the sum of the areas of districts equals the area of the grid. Once the user has finished the configuration stage, a HashMap that maps positions in the grid to districts is created, so that the district a cell is in can be found in $O(1)$ time.

Whenever a user places agents or monuments, or creates districts, the locations of

```
private boolean districtsOverlap(District d1, District d2) {
    if (d1.getBottomRight().getY() < d2.getTopLeft().getY() ||
     ↪  d2.getBottomRight().getY() < d1.getTopLeft().getY()) {
        return false;
    }
    if (d1.getTopLeft().getX() > d2.getBottomRight().getX() ||
     ↪  d2.getTopLeft().getX() > d1.getBottomRight().getX()) {
        return false;
    }
    return true;
}
```

Listing 1: Code to determine if districts overlap

these are remembered so that if the simulation is reset, the user's custom experiment can be restored. This is achieved by cloning each of the placed items when the simulation begins. Different data structures are used to store agents, monuments, and districts. Agents and districts are stored in ArrayLists, whilst monuments are stored in a HashMap. A separate occupancy array is used to track the number of agents in each cell. This reduces determining if a cell has free space from an O(n) operation to an O(1) operation.

When agents are asked to make a movement decision, the server computes all possible movements, and presents these to the agents along with the district corresponding to each cell. In the event that there are no districts, the entire grid is defined as district 0. A custom GridPosition class is used for the coordinate system, and utility functions allow for easy conversion between a GridPosition object and an integer index (to access the occupancy array, for example). The grid is 0-indexed, with the origin in the top left, a horizontal x-axis, and a vertical y-axis (down is positive). The agents are asked where they would like to move in the order in which they are stored in the ArrayList. The order results in no material advantage for any agent - going first means some options might not be possible as agents are yet to move out of cells, going last means some options might not be possible as agents have moved into cells. The occupancy array is constantly updated, so that if an agent vacates a cell, agents choosing afterwards can move into that cell (if adjacent to their current position).

Once agents have moved, the communication phase starts. First, if there are any monuments, these are viewed, and then edited. For monument viewing, the server computes the distance between each agent and monument, and if that distance is less than or equal to the visibility, V, the agent can view the monument. Monuments can only be edited by agents that are in the same cell as the monument. If there are multiple agents that can edit a monument, the order in which the agents edit the monument is determined by their order in the agents ArrayList. There is a benefit to editing a monument last, therefore a random order could also be used. Once monuments have been viewed and edited, inter-agent communication takes place. The server groups agents based on location, and provides each agent with a list of agents they can communicate with.

Agent clustering is handled by external functions, except if district clusters are being used, in which case the server assigns clusters based on districts. Once the agents have been clustered, the server collates the votes from each agent, and hands control over to a SubgameHandler object, which computes the results at the agent level. This allows for increased flexibility with the focal point games, as agents within a cluster can achieve different results (e.g. if you are in the minority, you win).

5.3.3 SubgameHandler Class

The SubgameHandler class computes the results of each focal point game. An enumeration is used to track the subgame implementation, with both 'consensus' and 'majority' games implemented. The consensus game requires all agents to choose the same option for them to receive the reward, whereas the majority game requires more than 50% of the agents to choose the same option. Results are calculated on a per agent basis, but are also recorded on a per cluster basis for data visualisation and logging purposes. Custom definitions can be used for any future subgames that make use of the fact that agents within a cluster can achieve differently.

5.3.4 GameGraphics Class

The GameGraphics class extends the GViewListener class from the G4P (GUI for Processing) library [37]. ViewListeners are essentially sub-sketches within the main sketch. There is an `update` method, and whenever this is called the graphics are updated. This allows the server to update the game graphics only when necessary. The GameGraphics class follows a very similar design style to the Server class. The `update` method contains a switch statement that switches over the server's state. Separate private drawing methods are implemented to draw the game at each stage.

Drawing the grid is quite straightforward - black lines are drawn at regular intervals, both horizontally and vertically, to create the cells. Red lines are drawn on top of the grid to depict districts. Agents are represented by circles. All agents are initially coloured the same, but the colours change to show the different clusters (unless district clusters are being used). Agents' outlines are black, unless the result of a focal point game is being displayed, in which case the outline turns green or red for success and failure respectively.

As multiple agents can be in the same cell at once, the coordinates of an agent

```
private float calculateMovingPosition(float position, float
 ↪  nextPosition) {
    return position + (1 + MOVE_FRAMES - server.getFramesToMove()) *
     ↪  (nextPosition - position) / MOVE_FRAMES;
}
```

Listing 2: Code to calculate position of moving objects

are not sufficient to determine exactly where to draw each agent. To complement the location, each agent within a cell is uniquely numbered (by the server). From an agent's location, number within its cell, and the total number of agents in the cell, the exact location to draw the agent can be computed. When drawing moving agents, the previous and new locations must be calculated, and then the position can be found by adding the difference between the locations, scaled by the frame number and total frames, to the previous position. It takes a total of 30 frames for agents to move, although this number can easily be changed to speed up simulations. As an example, if an agent is moving from (100, 100) to (200, 200), and we wish to draw the agent 12 frames in, the position is (140, 140), since the difference is (100, 100), and 12/30 is 0.4. The code in Listing 2 shows the exact implementation, which calls the server's `getFramesToMove` method, which returns the number of frames left in the movement sequence. MOVE_FRAMES is the constant that specifies how many frames an object should be moving for, and is set as 30.

The maximum agent occupancy of a cell is four, and this was carefully chosen as it makes the calculations simpler to determine the precise locations of agents. If there is one agent in a cell, it is drawn in the centre. When there are two agents, they are drawn equally spaced along the central horizontal line. For three agents, the first two are drawn equally spaced horizontally in the centre of the top half of the cell, while the final agent is drawn in the centre of the bottom half of the cell. With four agents, each agent is drawn in the middle of one of the quadrants. Beyond four agents, the placements become more tricky (and the agents smaller), hence the cap on occupancy of four.

Monuments are drawn as trapeziums, and are made to look like black monoliths. Since only one monument can be in a cell at a given time, a monument's location is sufficient to determine how and where to draw it. When monuments are moving, the drawing process is identical to that of drawing moving agents.

The GameGraphics class also does some event handling. This is for the placement of agents and monuments, and the creation of districts. Another enumeration is used, with a switch statement controlling the flow so that the right object is placed. The x and y coordinates of the mouse are recorded whenever the left (or right) button is clicked, dragged or released. When placing agents or monuments, only the final x and y coordinates are used, whereas for districts both the start and end coordinates are required. The raw mouse coordinates are converted into grid positions using a utility function. The grid position is then passed to the server via the appropriate method.

5.3.5 VisAndInfoPanel Class

The VisAndInfoPanel class is instantiated to create the part of the GUI which displays information during the configuration process, and graphs while the simulation is running. While the user is configuring a simulation, the panel takes the user through the process, and explains some of the requirements. If there are any errors, such as localised monuments being selected but no districts drawn, an error message is clearly displayed in red.

While a simulation is running, up to four graphs are displayed in the panel. An example is shown in Figure 5.1. The graph in the top left shows the frequency of each cluster size. The top right graph shows the win rate per round, calculated as the percentage of successful agents. The bottom left graph shows the win rate for each cluster size. The graph in the bottom right only appears if there are monuments in the simulation ($L > 0$), and shows win rate by monument proximity. This is calculated as the distance to the monument closest to the centre of a cluster. When k-means clustering is used, the centre of each cluster is just the centroid, so the centroid locations are stored to prevent unnecessary computation. However, when district clusters are being used, the centre of each cluster does need to be computed, and is found by averaging the positions of the agents in each cluster. This graph also doesn't appear if localised monuments are used, since the metric becomes a lot less meaningful. As monument proximity is continuous, the results are grouped using the floor function on the distances.

To draw the graphs, the gicentreUtils library was used [38]. The library provides easy to use BarChart and XYChart classes. The BarChart class was used for all graphs except the win rate over time chart, which used the XYChart class. Data is updated each round via methods called by the server, and internal arrays keep track of all of the data. Once a simulation has finished, the data is passed to a DataLogger object.

5.3.6 DataLogger Class

The DataLogger class is responsible for creating a JSON file that contains various information and statistics about a particular simulation. JSON was chosen for the data format since it is widely used, and JSON files are easy to read as input in other programming languages (such as Python, which was used for more data visualisation later in the project). Additionally, Processing has great built-in support for JSON, providing JSON object and array types that are not present in Java.

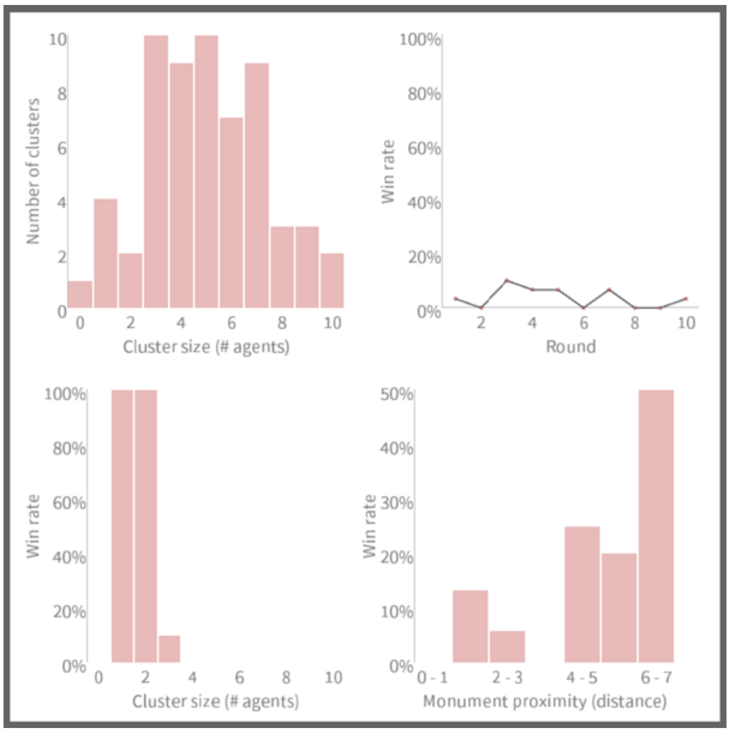


Figure 5.1: Example graphs

When a simulation begins, the server calls the DataLogger's `logConfig` method, which records all of the configuration variables. Then, at the end of each round, the server calls the `logRound` method. In this method, the results and choices of each cluster are recorded. At the end of the simulation, statistics are obtained from the VisAndInfoPanel, such as the win rate by monument proximity (if monuments are present), and this is logged too. Finally, a new log file is written in a logs directory, with the simulation start time used as a unique file name.

5.3.7 Agent Class

The Agent class provides the implementation for a random agent (i.e. one that both moves and chooses randomly), and can be extended by other classes to create more sophisticated agent strategies. The main methods to be overridden are `chooseNextGridPosition`, `voteForChoice`, `receiveVoteResult`, `communicate`, and `receiveMessage`. In addition, there are `viewMonuments` and `editMonument` methods which are only called if $L > 0$. Each agent is uniquely identifiable by an integer ID.

The `chooseNextGridPosition` method provides the agent with a list of cells it can move to, and the districts that each of those cells is in. By default, agents will choose randomly from this list. Agents choose by responding with the index of the position in the list which they would like to move to. The first position in the list is always the agent's current position, therefore if the user wishes to experiment with static agents, they can override the method to always return 0 (since $M \geq 1$).

The `communicate` method gives each agent a list of agents in its cell. It is then up to the agent to choose which agents it wishes to communicate with, and what messages it would like to send. Messages are received by the `receiveMessage` method, which provides the agent with both the sender and the message, which is simply a string. The agent designer is given free choice as to the language used by the agents.

During the focal point games, when the `voteForChoice` method is called, each agent is given the number of choices, along with a list of viewable monuments. This is because the order of events is slightly different when agents have reached their final positions before a focal point game. Ordinarily, monuments are viewed, then edited, and then communication between agents takes place. However, when the agents have no moves left, communication happens first, then monument viewing (in parallel with choosing an option), then results are distributed, and finally monuments are edited. This prevents monuments from being edited just before a game, but allows them to be updated as soon as the results of the game have been shared with the agents. Additionally, agents don't have to store the visible monuments for choosing since `voteForChoice` has the viewable monuments as a parameter. When choosing, agents respond with an integer in the range $[1, C]$.

5.3.8 Monument Class

In a lot of ways monuments are similar to agents, although with fewer and different methods. The base monument class implements a static monument, but the class can be extended to allow the monuments to move. This is achieved by overriding the `chooseNextGridPosition` method. There are two methods that agents use to interact with monuments, `getText` and `setText`, which allow agents to view and update the text displayed on the monuments.

5.4 Clustering

To cluster the agents, the k-means algorithm [39] was chosen, which is an unsupervised learning technique used to group data into k clusters. For centroid initialisation, agent locations are chosen at random, and added to a set to ensure all centroids are unique. Then, an iterative process occurs, assigning agents to clusters and calculating new centroid locations, until the agent assignment remains unchanged. This is outlined in Algorithm 1.

Algorithm 1 K-Means clustering

```
centroids ← initialiseCentroids()
assignmentUnchanged ← false
while not assignmentUnchanged do
    assignmentUnchanged ← true
    for agent in Agents do
        agent.assignNearestCentroid(centroids)
        if agent.assignmentChanged() then
            assignmentUnchanged ← false
        end if
    end for
    centroids ← calculateClusterMeans(centroids)
end while
```

Chapter 6

Testing

Processing is designed to make drawing sketches easy, and an unfortunate consequence is that it is not simple to port standard testing techniques to Processing code. Whilst Processing does have many libraries that simplify development, there are no unit testing libraries for example. Therefore, testing of the simulator required a different approach. Fortunately, Processing is inherently visual, so much of the functionality can be verified by visual inspection. Other aspects, such as testing the inner-workings of an agent implementation, used alternative testing methods.

The simulator was built in a very iterative process, whereby a small improvement or enhancement was made, and this could immediately be tested. For example, the code to draw the grid was added, and then the simulator run to check that it was correctly drawn. This only required counting the number of cells to ensure the correct grid size, and visual inspection of the sizes of the grid cells to check that they were equal. The next step was to add an agent, represented by a circle. A single agent was instantiated at a particular location, and then the simulator run, with a check to make sure the agent appeared the right size (relative to the grid size) and in the right place. This methodology was useful throughout the implementation phase of the project.

Another example was checking that the k-means algorithm had been correctly implemented. There are various visual checks that can give a high degree of confidence that this is the case. As the simulator colours each cluster differently, the colours can be counted to check that the correct number of clusters have been used. Additionally, it should be impossible for a cluster to be inside of another cluster, for example, and such cases can be visually checked for. Increasing the number of times the algorithm is run increases the degree of confidence with which we can say that the algorithm has been correctly implemented. An alternative approach would have been to use a known correct implementation, and compare the results with the simulator implementation (using the same centroid initialisations, otherwise the results would almost certainly differ).

Other parts of the simulator were verified through use as well as visual inspection, in particular during the configuration stage of a simulation. Many simulations were configured using the text fields and other GUI components, and then visual inspection of the grid would confirm that the variables were correctly set (counting the number of agents, number of rounds, number of moves etc.). Various edge cases were also tested, such as simulations with a single agent, or the largest and smallest grid sizes, as well as combinations of various extremes. Exhaustive testing of edge cases would have been infeasible due to the number of variables, but the most common edge cases were all tested. This, along with the significant number of simulations run as part of experiments, strongly indicate that the simulator correctly functions.

One very useful test was to compare theory versus experimental results when using agents choosing randomly. This is discussed in much more depth later on. The general principle is that agents choosing randomly should have the same success rate as suggested by the theory (within experimental limits). Early simulations found that the agents were actually performing worse than random, which indicated that there was an issue either with the simulator, or with the agent implementation. It was found that instead of agents choosing from C options, with an equal probability of choosing each, they were actually picking from $C + 1$ options, with 2 of the options half as likely to be chosen as any of the others. This reduced the probability of success, hence the agents' under-performance. The culprit was an erroneous random selection implementation. The final implementation generates a random number between 0 and C, and rounds up to the nearest integer, giving C possible choices, each with a $\frac{1}{C}$ chance of being selected. The previous implementation rounded to the nearest integer, which meant the agents could choose option 0, when the options were supposed to start at 1. Without the theory to compare the results to, it is much less likely that this bug would have been caught. Once it was fixed, the agents performed as expected, which suggested simulator functional correctness.

Testing other agent strategies was slightly more involved, and used a combination of visual checks in conjunction with logging information to the console. For example, to test agents viewing monuments, first the agents would log how many monuments they could see, which would be verified by visual inspection. Then, the logs could be checked to see if the choices made by the agents matched the instructions on the monuments.

When agents were communicating, logging to the console was again very useful. Implementations that involved all agents in a cell messaging each other were verified by logging the messages, and checking that the number of messages per cell corresponded to the number of agents in each cell.

Overall, the simulator was tested to a significant degree, through both directed tests and general usage during experiments. However, the testing process was almost entirely manual. For a piece of software of this size, it was just about manageable, but if the code base had been larger, an automated test suite would both save time and be more robust. Certain aspects of the simulator are much easier to test manually, particularly anything to do with the visuals, although there is the possibility of automatically testing visuals based on pixel colours. This would have required significant additional effort, and likely would have detracted from the success of the overall project.

Chapter 7

Simulations and Results

Overview

This chapter includes the results for the experiments conducted using the simulator. For each experiment, the configurations and agent implementations will be introduced, hypotheses formed, and then the results discussed. Initially, a baseline will be established using agents that choose randomly in the focal point games. Then, various other agent implementations will be used, with the hope of improving coordination relative to the baseline.

7.1 Experiment 1a: 2 Random Agents

7.1.1 Setup

Experiment 1a was conducted with two goals in mind - to establish baseline performance with random agents, and to partially verify the correctness of the simulator and background theory. The experiment was conducted using the following configuration:

- G = 3
- N = 2
- M = 4
- O = 4
- K = 1
- C = [2..20]
- L = 0
- D = 0
- R = 100

Since the configuration specified only 1 cluster, both agents were always playing a focal point game with each other. While G, N, M and O were also specified, their values were mostly irrelevant (the grid just needed to contain 2 agents).

The agents were configured to both move and play the focal point games completely randomly, as well as ignore the results of the games. Whilst the agents were moving around the grid, this had no impact on the results of the experiment - the simulation would have been equally valid and useful had the agents been stationary.

7.1.2 Hypothesis

The experiment involved controlling a single independent variable $C \in [2..20]$, and measuring a single dependent variable, the win rate. Since 100 rounds of the focal point game were played, the win rate can simply be defined as the number of rounds in which both agents chose the same option, expressed as a percentage. The theory tells us that with 2 agents and C choices, there is a $\frac{1}{C}$ probability in any given round that the agents coordinate. Therefore, the expected win rate with C choices is $\frac{100}{C}\%$. It was expected that the results of the experiment would closely match the theory, which would indicate that the simulator was working correctly (whereby 'correctly' means aligning with the theory).

7.1.3 Results

Figure 7.1 shows the win rates achieved by the agents as the number of choices was varied. The experimental results are very close to those predicted by the theory - sometimes the achieved win rate was slightly higher than expected, other times it was slightly lower. This behaviour comes as no surprise given the agents only played the focal point game 100 times during each run. We would expect the actual win rate to converge towards the expected win rate as the number of repetitions increases.

The one significant outlier was the simulation with the agents picking from 10 options. Whilst the most likely outcome is 10 successes, the agents managed to coordinate 21 times. The probability of 21 or more coordinations with 10 choices is 0.00081. In other words, such an event occurs approximately once every 1235 simulations. To check that this was simply an unlikely outcome rather than an issue with the simulator, the simulation was rerun for the 2 agents with 10 choices, but with 1000 rounds rather than 100. The outcome of this simulation was a success rate of 9.2%. The probability that the success rate is $\leq 9.2\%$ is 0.21613. Therefore, this simulation represented a much more likely outcome, and suggested no reason to believe that there was an issue with the simulator.

Overall, the experiment shows that with two agents playing focal point games randomly, their coordination success rate can be predicted using simple probability theory, and as the number of choices increases, the chance of coordination decreases.

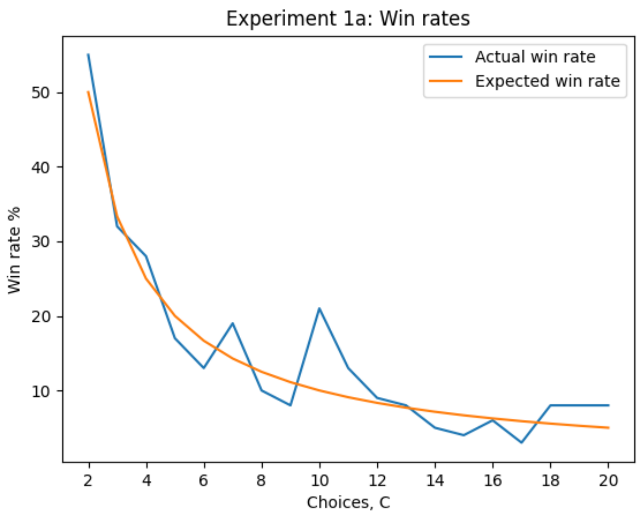


Figure 7.1: Experiment 1a results

7.2 Experiment 1b: 30 Random Agents

7.2.1 Setup

Experiment 1b built on experiment 1a in a few ways: the number of agents was increased to 30, the agents were grouped into 6 clusters, and the number of choices was restricted to a maximum of 5. This meant that there were clusters of varying sizes, and that the clustering of the agents would change each round. To accommodate the larger number of agents, a 6x6 grid was used. The lower maximum value of C was chosen because, for example, if $C = 20$ and a cluster has 5 agents, the probability of coordination is 0.000625%, therefore an unreasonable number of simulations and rounds would have been required to obtain any meaningful data. The experiment was conducted using the following configuration:

- G = 6
- N = 30
- M = 4
- O = 4
- K = 6
- C = {2, 3, 4, 5}
- L = 0
- D = 0
- R = 100

The agents were configured to both move and play the focal point games completely randomly, as well as ignore the results of the games. Whilst the agents were moving around the grid, this had little impact on the results of the experiment - the simulation would have been equally valid and useful had the agents been stationary, with the caveat that the distribution of cluster sizes would be less interesting

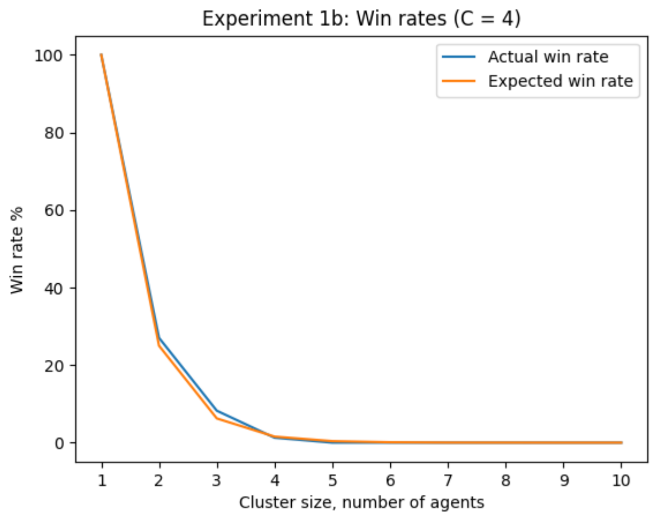


Figure 7.2: Experiment 1b results, $C = 4$

(if the agents aren't moving, even with randomly chosen centroids, there is a not insignificant chance that the clusters remain unchanged between rounds).

7.2.2 Hypothesis

As with experiment 1a, the only independent variable was C. Due to the nature of the experiment, the sizes of the clusters were varied, although this was not directly controllable due to the random movements of the agents, in addition to the random centroid initialisation used by the clustering algorithm. The dependent variable was the win rate once again. The significant difference compared to experiment 1a was that the cluster size was not fixed, therefore the win rate for each cluster size would also be measured.

The theory can be used to predict the probability of coordination for a cluster with X agents. With C choices, the probability of coordination is C^{1-X}. Clearly as the number of agents in the cluster increases, the lower the chance of success. Similarly, as the number of possible choices increases, for fixed $X > 1$, the coordination probability decreases. The results of the experiment were expected to align closely to the predicted results.

7.2.3 Results

Figure 7.2 shows the predicted and actual win rate by clusters of sizes 1 through 10, when given 4 options to choose from. The actual results are remarkably similar to the theoretical results, which provides further evidence of the correct implementation of the simulator. As the size of the cluster increased, the win rate decreased,

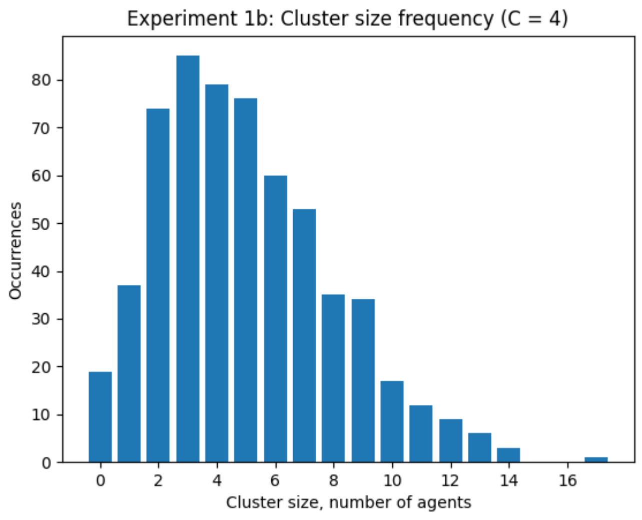


Figure 7.3: Experiment 1b cluster sizes, $C = 4$

approaching (and reaching) 0. There were a number of clusters containing just a single agent, and the win rate for these clusters was 100% since coordination was guaranteed regardless of the choice the agent made. Figure 7.2 has been truncated for increased clarity - the largest cluster size observed comprised 17 agents, but the largest winning cluster contained only 4 agents.

Figure 7.3 shows the number of occurrences for each cluster size, with $C = 4$. The cluster size is independent of the number of choices, therefore the distribution was very similar when C was varied. The largest cluster observed was a single cluster of size 20 with $C = 2$. One point of note is that clusters of size 0 were also observed (approximately 3.3% of all clusters contained no agents). This phenomenon can occur particularly when the value of K is 'wrong' - that is to say that there aren't K natural clusters in the data. As the agents are moving randomly, with enough rounds simulated it is almost certain that at some point the agents' locations, combined with the random centroid selection, lead to an empty cluster (for $K > 1$). [40] shows an example of how a dataset with 2 natural clusters, grouped using $K = 3$, can lead to an empty cluster. Modified versions of the algorithm have been proposed to prevent the issue [41].

As seen in Figure 7.4, as the number of choices increased, the win rate decreased. We can once again observe that for larger cluster sizes, the win rate is lower for a given value of C (both the predicted and achieved win rate lines for clusters of 4 agents are below the lines for 3 agents).

Figure 7.5 shows a typical win rate over time graph, with the agents picking from 4 options. There is no discernible trend, as expected, and the win rate remains low throughout, rarely exceeding 10%.

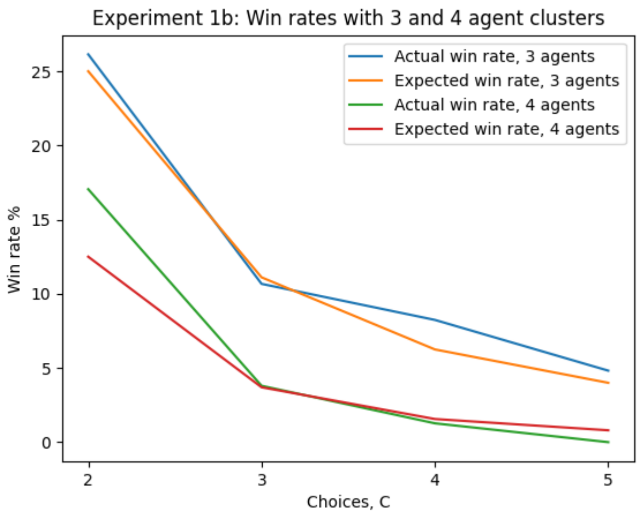


Figure 7.4: Experiment 1b win rates, clusters of 3 and 4 agents

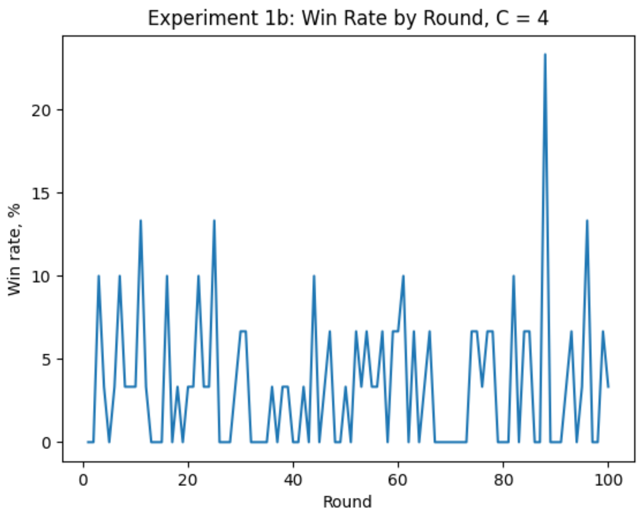


Figure 7.5: Experiment 1b typical win rate, $C = 4$

7.3 Experiment 2a: A Static Monument

7.3.1 Setup

Experiment 2a was the first to use agents with a non-random implementation, along with a monument as a means for coordination, motivated by the monuments used in ancient Athens [25]. For the monument to be useful, it was necessary that the agents understood a common language. As the agents would be presented with a list of consecutive integers, 1 to C, a possible language was to introduce the idea of 'high' and 'low' numbers, inspired by the 'extremity' focal point feature discussed by Kraus et al. [30]. The experiment was conducted using the following configuration:

- G = 6
- N = {10, 20, 30}
- M = 4
- O = 4
- K = {2, 4, 6}
- C = {2, 3, 4}
- L = 1
- V = 1.5
- D = 0
- R = 100

Both the total number of agents and number of choices were varied (for each value of N, a simulation was run with each value of C). The number of clusters, K, was set such that the average cluster size was 5 agents. The monument was placed in the same location each time (at coordinate (2,2) - the third square on the leading diagonal), and was initially blank. For each possible combination of N and C, the experiment was conducted 5 times, leading to a total of 45 simulations. The monument visibility was set constant at 1.5. This meant that the text on the monument was visible from the monument square and any adjacent square (including diagonally adjacent). As a result, the monument was visible from 25% of the squares on the grid.

7.3.2 Agent Implementation

The experiment used a new agent implementation, known as the *Monument Viewing Agent*. The agents were designed such that they knew that using the monument would be beneficial for coordination. Therefore, whenever an agent was in a position to write on the monument (i.e. at the same grid coordinates as the monument), they would write on the monument if it was blank. There was an equal chance that the agent would write either 'Hi' or 'Lo', corresponding to 'high' and 'low' respectively. All agents knew the meaning of these symbols, and would remember the most recent symbol seen (if multiple monuments were visible at the same time, the agent would remember the symbol on the closest monument, although this first experiment only used a single monument). During a focal point game, if the agent had seen a monument with a symbol on it, it would vote in accordance with the symbol - choosing the highest number, C, in the case of 'Hi', and the lowest number, 1, in the case of 'Lo'. If the agent had not observed a symbol on a monument, it would continue to choose randomly as per the previous experiments. Agent movement remained random.

7.3.3 Hypothesis

In this experiment, there were three independent variables - N, K, and C. The dependent variables were the win rate, and a new variable 'time to coordination', denoted TTC_{100}, with the subscript indicating the level of coordination. TTC_{100} is the first round where 100% coordination is achieved in that round and all subsequent rounds. If $\mathrm{TTC}_{100} = 80$, coordination was quite slow, as it took 80 rounds for all agents to learn how to choose in the focal point games. By contrast, if $\mathrm{TTC}_{100} = 10$, coordination was much faster, relatively speaking.

The agents were programmed to take whatever was written on the monument as gospel. Additionally, once the monument had been written on, the symbol would remain unchanged throughout the remainder of the simulation. Therefore, once an agent had been close enough to the monument to read what was on it, from then on they would always vote in a particular way. It was hypothesised that as more and more rounds passed, because the agents were randomly moving around the grid, eventually all agents would view the monument, and 100% coordination would be achieved. The time to coordination should have been relatively constant with increasing numbers of choices, although a slight increase would not have been unexpected, as with more choices there was a lower chance that any agents that haven't observed the monument coordinate by pure luck. Similarly, with decreasing number of agents, and therefore decreasing number of clusters (since K was scaled linearly with N), the time to coordination was likely to be slightly lower due to the increased chance of accidental coordination.

In a simulation with more agents, the hypothesis was that some level of coordination might be achieved quicker, as the monument was more likely to be encountered (and written on) earlier. Additionally, with more agents there was a higher chance that one or more agents never enter the visible region of the monument, which would either increase the TTC_{100}, or prevent 100% coordination from being achieved altogether. The random agent location initialisation meant that with fewer agents it was more likely that all agents spawned close to the monument (due to both the smaller number of agents, and also the occupancy limit of each location), therefore some lower TTC_{100} values were expected.

7.3.4 Results

The headline results of the experiment are shown in Table 7.1. In all 45 simulations, total coordination was achieved in at most 21 rounds (recorded in a run with $N = 20$ and $C = 4$). On multiple occasions (with $N = 10$ and $C = \{2, 3\}$), the TTC_{100} was as low as 3. As expected, with fewer agents in the simulations, coordination was, on average, achieved in fewer rounds, although the difference in median TTC_{100} between 20 and 30 agents was much smaller than the difference between 10 and 20 agents. The minimum TTC_{100} decreased with fewer agents, as with fewer agents the average distance from agent to monument was, on occasion, lower, hence less moves were required for all agents to observe the monument, and for coordination to be achieved.

Overall, the experiment showed that the introduction of a single monument can hugely increase the likelihood of coordination. Note that while coordination was achieved in all 45 simulations, there was no guarantee that this would be the case - even with each agent able to move up to 400 times, there was no guarantee that any single agent would enter the monument's visible region. The further an agent spawns from the monument, the less likely they are to view the monument during the simulation. As the monument was placed near the center of the grid, the average starting distance from the agents was minimised. 5 simulations were run with the monument placed in the corner (at location (1,1) so that the number of squares from which the monument was visible remained at 9), with $N = 30$, $K = 6$, and $C = 4$, and the TTC_{100} minimum, median, and maximum were 21, 26, and 50 respectively. All these numbers are significantly higher than those recorded with a more centralised monument, which shows that the location of the monument impacts the time taken for coordination to be achieved. Another factor that impacts the rate at which coordination is achieved is the size of the monument's visible field, which is investigated in experiment 2c.

Figure 7.6 shows the typical win rate over time with 30 agents picking from 4 options. As more and more agents view the monument, the win rate increases, until total coordination is reached. The increase in win rate is close to linear.

7.4 Experiment 2b: Multiple Monuments

7.4.1 Setup

As seen in experiment 2a, the introduction of a monument facilitated agent coordination. The next logical step was to increase the number of monuments. This experiment used two different grid sizes with varying numbers of monuments:

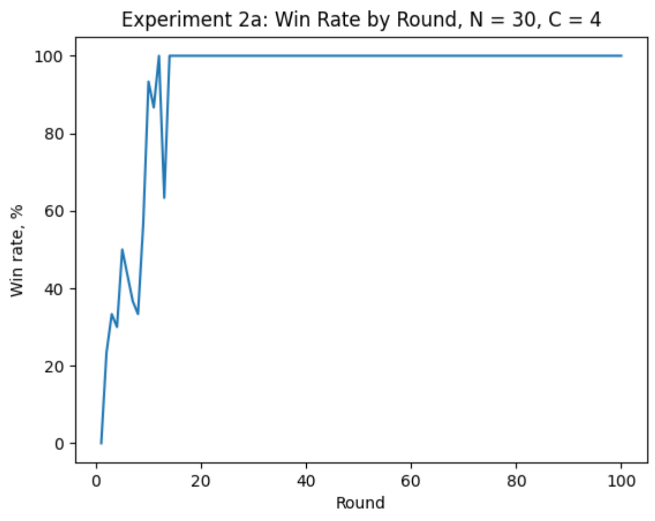


Figure 7.6: Experiment 2a typical win rate, $N = 30$, $C = 4$

Agents, N	**Choices,** C	**Min** TTC_{100}	**Median** TTC_{100}	**Max** TTC_{100}
10	2	3	9	19
	3	3	6	9
	4	7	11	15
20	2	8	13	16
	3	9	9	12
	4	7	14	21
30	2	10	13	18
	3	11	13	15
	4	10	12	19

Table 7.1: Experiment 2a results

- G = {6, 10}
- N = 30
- M = 4
- O = 4
- K = 6
- C = 4
- L = {1, 2, 3, 4, 6, 9, 15}
- V = 1.5
- D = 0
- R = 100

With $G = 6$, experiments were run with 2, 3, 4, and 6 monuments, while with $G = 10$, $L \in \{1, 2, 3, 4, 6, 9, 15\}$. C was held constant at 4, and N constant at 30, to reduce the total number of simulations. For the case where $L = 1$, the monument was placed in the top left cell of the central 4 cells of the grid. When $L \in \{2, 3, 4\}$, the monuments were placed in the centre of the quadrants of the grid (diagonally opposite in the case of 2 monuments). For $L > 4$, more complicated monument patterns were used. Diagrams of these can be found in Appendix B. As with experiment 2a, for each configuration 5 simulations were run.

7.4.2 Hypothesis

In this experiment there were two independent variables, G and L. The primary dependent variables were TTC_{100} and coordination rate (the percentage of simulations for which TTC_{100} was well-defined).

It was hypothesised that this experiment would produce varying results - sometimes coordination would be achieved quicker than with 1 monument, other times the agents would fail to reach 100% coordination. This was due to the agent implementation - they act according to the symbol on the nearest monument, and the monuments are initially written on in a near random way. The symbol written on the first monument(s) would definitely be decided randomly. Once at least one monument has been written on, any further monuments are written on randomly if the agent hasn't observed any other monuments, or the symbol is copied in the case the agent has viewed a monument. Due to the monuments being quite spaced out, and the relatively large number of agents, it was likely that most monuments would be written on randomly. It was expected that if all monuments displayed the same symbol, coordination would be achieved faster than with a single monument, everything else kept equal. If the monuments displayed different symbols, confusion was likely to occur. This is because agents in the same cluster may view different monuments saying different things, and then choose different options.

Additionally, it was expected that with all else held equal, increasing the size of the grid from 6 to 10 would increase the TTC_{100}, for a couple of reasons. Firstly, for a monument on a 6x6 grid, the visible area covers 25% of the grid, whereas for the same monument on a 10x10 grid, the visible region is only 9% of the grid. This

			TTC_{100}		
Grid Size	**Monuments**	**Coordination**	**Min**	**Median**	**Max**
6	2	40%	7	9.5	12
	3	20%	12	12	12
	4	0%	N/A	N/A	N/A
	6	0%	N/A	N/A	N/A
10	1	100%	31	45	52
	2	20%	44	44	44
	3	40%	23	26	29
	4	0%	N/A	N/A	N/A
	6	20%	4	4	4
	9	0%	N/A	N/A	N/A
	15	0%	N/A	N/A	N/A

Table 7.2: Experiment 2b results

makes it less likely that an agent will view the monument. Secondly, the increased grid size increases the average agent spawn distance from the monuments, which increases the amount of time it takes on average for an agent to reach the visible region of a monument. Another anticipated effect of a larger grid was that the coordination rate would decrease slightly. This was because the monuments would be more spread out, which increased the probability that the symbols written on the monuments were chosen at random.

7.4.3 Results

The coordination and TTC_{100} results for experiment 2b are shown in Table 7.2. As expected, on many occasions the existence of multiple monuments caused confusion amongst the agents. In all experiments with at least two monuments, the most successful situations ($G = 6$, $L = 2$, and $G = 10$, $L = 3$) only achieved 100% coordination 40% of the time. On the occasions where 100% coordination was achieved, the TTC_{100} values decreased with increasing number of monuments (this behaviour was more apparent with $G = 10$). Additionally, for the same number of monuments, the TTC_{100} values were higher for the 10x10 grid than the 6x6 grid, for the reasons outlined in the hypothesis. With large numbers of monuments such that a monument was in view of (nearly) the entire grid, coordination was never achieved - with increasing number of monuments, the likelihood that the monuments all displayed the same symbol decreased.

Figure 7.7 shows the win rate over time from one of the simulations with 2 monuments on a 6x6 grid where total coordination was not reached. There is no real trend in the data, but the win rates are significantly higher than the baseline, typically between 20% and 60%.

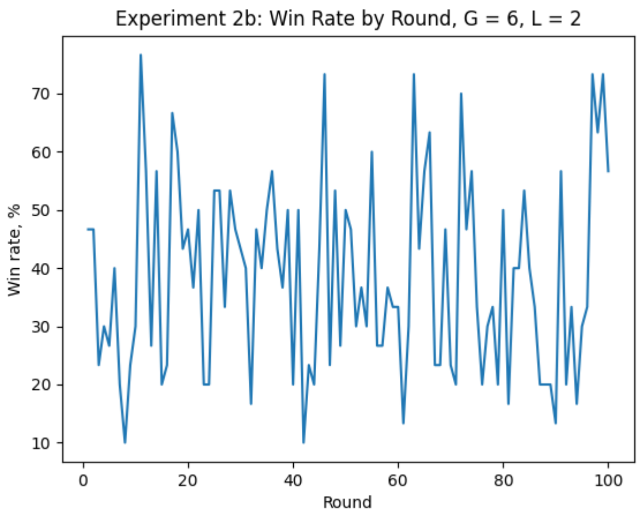


Figure 7.7: Experiment 2b typical win rate without coordination, $G = 6$, $L = 2$

7.5 Experiment 2c: Monument Visibility

7.5.1 Setup

This experiment tested the impact of monument visibility on TTC_{100}. It was configured with a single monument placed centrally on a 10x10 grid. The full experiment configuration was:

- G = 10
- N = 30
- M = 4
- O = 4
- K = 6
- C = 4
- L = 1
- V = [0..8]
- D = 0
- R = 100

With a visibility of 0, the monument was only visible from the cell it was located within. When the visibility was 8, the monument was visible from the whole grid. There was only one cell from which the monument was not visible with $V = 7$.

7.5.2 Hypothesis

This experiment had one independent variable, V, and two dependent variables, the coordination rate and TTC_{100}. It was expected that as V increased, the coordination rate would increase and TTC_{100} would decrease. This is because there was a higher likelihood that the agents would see the monument, and therefore follow a rule in the focal point games, rather than choosing at random.

		TTC_{100}		
Visibility	**Coordination**	**Min**	**Median**	**Max**
0	20%	78	78	78
1	60%	69	75	84
2	100%	29	38	77
3	100%	21	26	29
4	100%	7	11	16
5	100%	2	4	7
6	100%	1	1	4
7	100%	1	1	5
8	100%	1	1	2

Table 7.3: Experiment 2c results

7.5.3 Results

The results, as seen in Table 7.3, show two clear trends: as monument visibility increases, the coordination rate increases, and the time it takes for the agents to coordinate decreases. On multiple occasions the agents were able to coordinate from round one. Even with the monument visible from the whole grid ($V = 8$), there was one simulation where the agents were not able to coordinate fully until round 2. This was because the monument had not been written on before the first focal point game took place. This shows that even if all agents can view a monument, this in itself is not sufficient to guarantee coordination. There is a requirement for a symbol to be on the monument for coordination to be guaranteed (with this agent implementation).

At lower monument visibilities, the coordination rate was not 100%. For example, with $V = 0$, on only one occasion were the agents able to reach full coordination. This was because the monument was only visible from a single cell, so it was unlikely that all 30 agents would visit (and therefore view) the monument.

7.6 Experiment 2d: Monument Editing Agents

7.6.1 Setup

Building on experiments 2a and 2b, this experiment aimed to solve the issue of multiple monuments instructing the agents to behave differently. The objective was for the agents to realise this, and update the monuments so that the same symbol would be found on every monument. The configuration for this experiment was:

- G = {6, 10}
- N = 30
- M = 4
- O = 4

- K = 6
- C = 4
- L = {2, 3, 4, 6, 9, 15}
- V = 1.5
- D = 0
- R = 100

With $G = 6$, experiments were run with 2, 3, 4, and 6 monuments, while with $G = 10$, $L \in \{2, 3, 4, 6, 9, 15\}$. This meant that the settings matched those used in experiment 2b (with the exception of skipping the case where $L = 1$), therefore the results would be directly comparable. To ensure a fair experiment, the same monument placements were used as in experiment 2b, and each configuration was simulated 5 times.

7.6.2 Agent Implementation

This experiment used the *Monument Editing Agent*, which incorporates previous results into its decision making, in addition to the symbols visible on monuments. Algorithm 2 shows the logic used by the agents when updating monuments. The logic used during the focal point games was almost identical to that of Algorithm 2, except for the fact that the agents were choosing in accordance with symbol, rather than writing a symbol. Additionally, the random element was a choice across all C options, rather than just the extremes. Whenever an agent viewed a monument, made a choice in a focal point game, or received the result of a game, the relevant variables would be updated.

Algorithm 2 Monument Editing Agent - monument update logic

```
if mostRecentResult $=$ Win and mostRecentChoice $\in$ {“Hi”, “Lo”} then
    monument.setText(mostRecentChoice)
else if mostRecentSeenText $\in$ {“Hi”, “Lo”} then
    monument.setText(mostRecentSeenText)
else
    if ceil(random(2)) $= 2$ then                ▷ 50% chance condition is met
        monument.setText(‘Hi”)
    else
        monument.setText(‘Lo”)
    end if
end if
```

7.6.3 Hypothesis

This experiment had two independent variables, G and L, and two primary dependent variables, TTC_{100} and coordination rate. The expectation was that the agents would be able to update the monuments such that the symbol on each monument

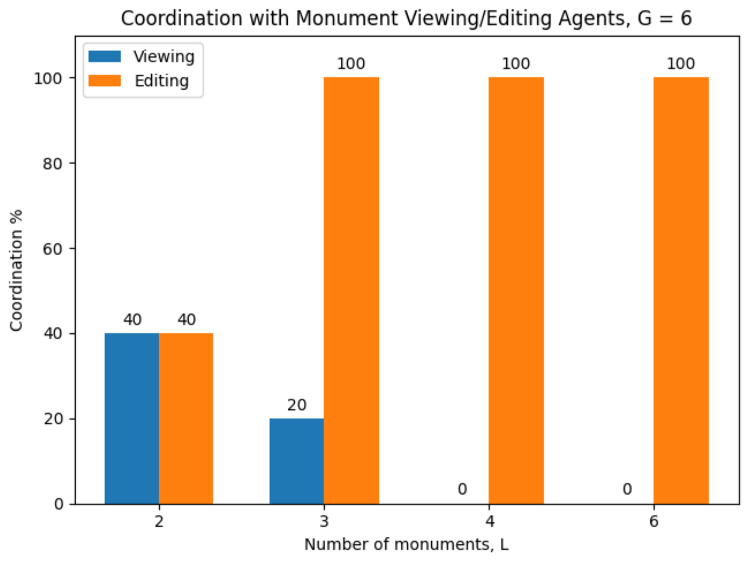


Figure 7.8: Experiments 2b and 2d coordination rates, $G = 6$

would be the same. This would ensure coordination, as long as all agents had most recently viewed a monument with the right symbol. Therefore, it was anticipated that the coordination rate would be higher compared to experiment 2b. However, it was hypothesised that the average TTC_{100} would be higher due to the time taken for the agents to edit the monuments.

There was less certainty as to the impact L would have on the results. With more monuments, the distance between them is decreased, which makes it more likely that agents will be able to update the monuments. However, with fewer monuments there are less monuments for the agents to update. Therefore, it was expected that changing L wouldn't have much of an impact (unless one of the two factors previously discussed heavily outweighed the other).

7.6.4 Results

Figure 7.8 and Figure 7.9 show the coordination rates achieved by both the Monument Viewing and Monument Editing agents, for the cases where $G = 6$ and $G = 10$.

For a 10x10 grid, it is clear that more monuments led to a higher rate of coordination, except for the case when $L = 2$, as there is approximately a 50% chance that the first symbols written on the monuments match (thus the coordination rate is higher). This shows that the presence of more monuments improves coordination because of the reduced distance between the monuments, and the higher probability that agents will encounter and update monuments. This factor outweighed the fact that there were more monuments that all needed to display the same symbol to guarantee universal coordination. For 3, 4, and 6 monuments on the 6x6 grid, the agents were able to communicate 100% of the time. This was due to a combination

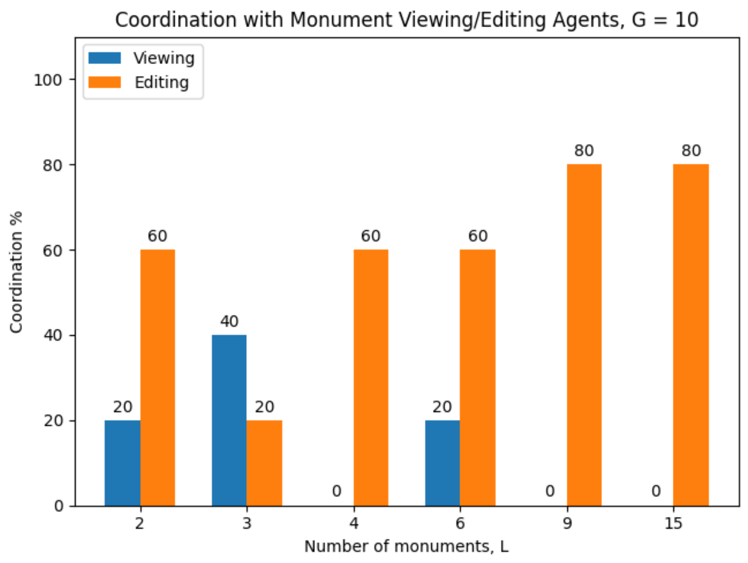


Figure 7.9: Experiments 2b and 2d coordination rates, $G = 10$

of the low number of monuments to write on, and the fact that the monuments were close together compared to those on the 10x10 grid.

Figure 7.10 shows the median TTC_{100} with $G = 10$. The red dots indicate the median values from experiment 2b (only if coordination was achieved). All median values recorded in this experiment were higher than those in experiment 2b, as expected. As L increased, there was no discernible pattern or trend - further simulations would be required to investigate this further.

The win rates over time for a typical simulation where coordination was reached, with 6 monuments on a 6x6 grid, are shown in Figure 7.11. Until 100% coordination is reached, the win rates are very inconsistent, fluctuating significantly, but with an overall increasing trend. Eventually when all monuments have been updated with the same symbol, and all agents have viewed that symbol, total coordination is reached.

7.7 Experiment 2e: Monument Separation and Agent Movement

7.7.1 Setup

This experiment aimed to improve understanding around the importance of agent movement. When random agents are used, movement has no impact on the outcome of the focal point games. However, with the introduction of monuments, movement is very important, as it allows agents to view and edit different monuments. To test its importance, the experiment used two monuments on a 10x10 grid, with the

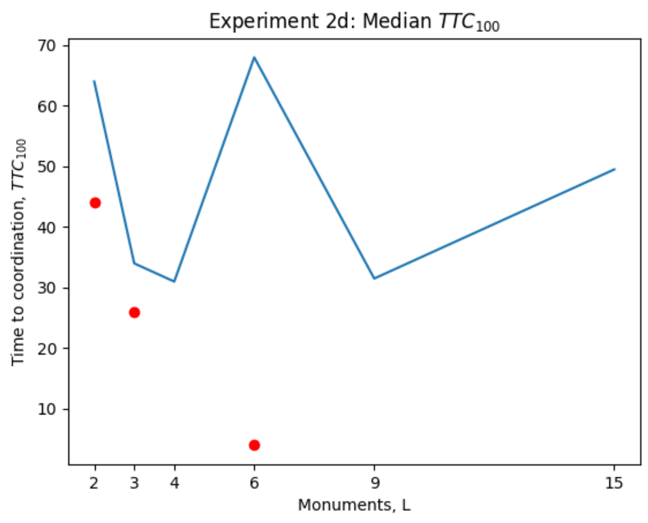


Figure 7.10: Experiment 2d: TTC_{100}, $G = 10$

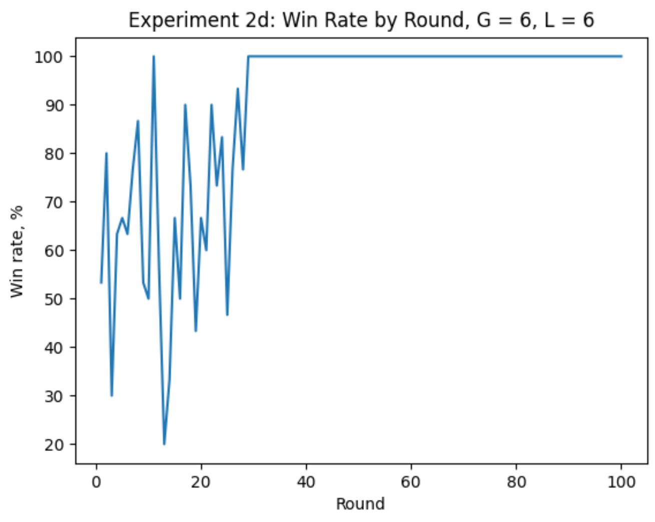


Figure 7.11: Experiment 2d typical win rates, $G = 6$, $L = 6$

distance between the monuments gradually increased. At each separation, a number of simulations were run with varying levels of agent movement. The full experiment configuration was:

- G = 10
- N = 30
- M = {2, 4, 6, 8}
- O = 4
- K = 6
- C = 4
- L = 2
- S = {1, 3, 5, 7}
- V = 1.5
- D = 0
- R = 100

For the purpose of this experiment, we defined a new variable, S, which represents the monument separation. More specifically, it is defined as the difference in the x coordinates of the two monuments, as the monuments had the same y coordinate.

This experiment used the same agent implementation as experiment 2d, i.e. using Monument Editing Agents.

7.7.2 Hypothesis

This experiment focused on two independent variables (S and M), and two dependent variables, TTC_{100} and coordination rate. It was hypothesised that as S decreased, the coordination rate would increase , due to the higher probability that agents would update the monuments so that they would display the same symbol. Additionally, it was expected that as M increased, the coordination rate would increase and the TTC_{100} would decrease. The coordination rate would increase due to the higher probability of a monument being reached by an agent that had previously been near the other monument. The TTC_{100} would decrease as the agents were moving more, increasing the chance of viewing a monument.

7.7.3 Results

The main results of this experiment are shown in Figure 7.12 and Figure 7.13. As seen in Figure 7.12, as M increased the time it took for the agents to coordinate decreased. Somewhat surprisingly, the figure seems to show a downwards trend - as S increased, TTC_{100} decreased. This can mostly be explained by the reduced total visible field of the monuments with a separation of 1. As the monuments were so close together, their visible fields overlapped. Consequently, a monument was only visible from 15 cells, rather than 18 in the other cases. Ignoring the results for $S = 1$, the downwards trend is no longer apparent. Therefore, for a given amount

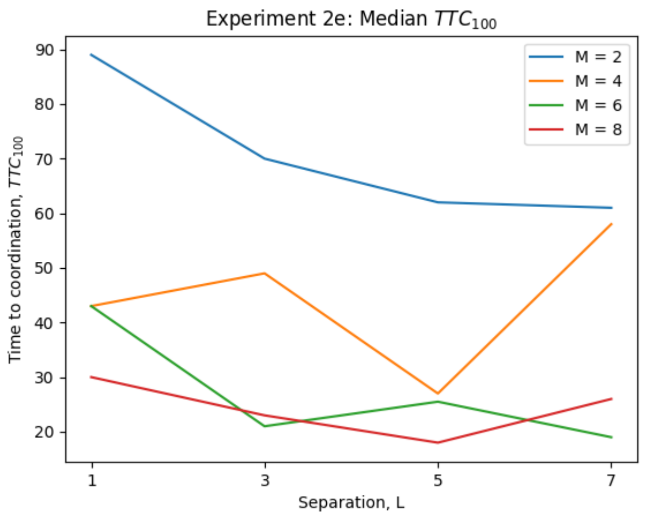


Figure 7.12: Experiment 2e TTC_{100}

of movement, on the 10x10 grid monument separation did not affect the time taken for coordination (M was the limiting factor).

Whilst S didn't affect TTC_{100}, it did affect the coordination rate, which can be seen in Figure 7.13. There is a striking difference when the monuments were furthest apart ($S = 7$). When the agents were able to move 6 or 8 times between each focal point game, they were always able to coordinate. However, when the amount of movement was less, the coordination rate was only 20%. This was for two reasons. Firstly, if both monuments displayed the same symbol, the agents were able to coordinate if an agent viewed either monument, which was less likely if the agents couldn't move as much. Secondly, if the monuments were initially written on differently, if the agents couldn't move much they were unable to make their way over to the other monument and update it before another focal point game took place. The second of these reasons is due to the agent implementation - the agents remember the outcome of the most recent game, and not the most recent win. An alternative agent implementation where the agents remember both of these things could be experimented with.

The importance of movement was particularly highlighted by the results with $S = 2$, as across all simulations the agents were only able to coordinate 50% of the time, which corresponds to the approximate probability of the monuments being initialised with the same symbol, therefore there was no noticeable benefit of using the Monument Editing Agents, and similar results would be expected with the Monument Viewing Agents. The same was not true for $M > 2$.

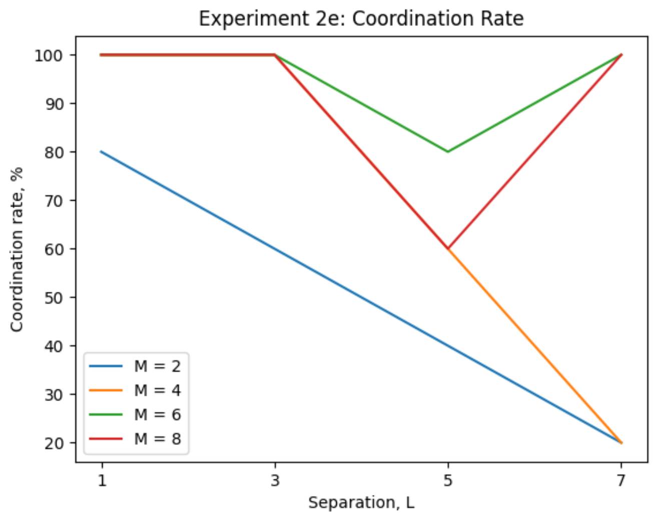


Figure 7.13: Experiment 2e coordination rates

7.8 Experiment 2f: Districts and Mobile Monuments

7.8.1 Setup

This was the first experiment to use districts. There were a few objectives:

- Test introduction of districts
- Measure effect of district clusters
- Compare static and mobile monuments

The experiments used a new agent implementation, known as *Homely Agents*. These agents behave identically to Monument Viewing Agents, except in their movement. Each agent is aware of the district it was spawned in, and it prefers to stay in that district rather than visiting others. During each move decision round, if there are both ‘home’ and ‘other’ districts that the agent can move to, there is a 90% chance it will stay in its ‘home’ district. All other choices are random (i.e. if it can only stay in a given district, it will choose a cell at random).

The full experiment configuration was:

- G = 6
- N = 30
- M = 4
- O = 4
- K = {4, 6}
- C = 4
- L = {0, 1} (static / mobile)
- V = 1.5
- D = {4, 6}
- R = 100

- District clusters = {Yes, No}

K was varied to match the number of districts, such that the number of clusters was constant regardless of the clustering mechanism used (k-means or district clusters). Each configuration was simulated 5 times. The districts used were either 3x3 squares or 3x2 rectangles, depending on the number of districts. When the static monument was used, this was placed in the top-left of the 4 central cells. The mobile monument started at location (1,1), and circled clockwise, maintaining a gap of 1 cell from the edge at all times, such that the boundary of the visible region corresponded with the edge of the grid.

7.8.2 Hypothesis

In this experiment, there were 4 independent variables - D, L, the clustering mechanism, and monument movement. The dependent variables were win rate by cluster size, TTC_{100}, and coordination rate. In the absence of a monument, the agents were voting randomly, therefore the win rate by cluster size was expected to be in line with the random agents. When a static monument was introduced, it was expected that the agents would perform slightly worse with the districts than they would if there weren't any districts (as was the case in experiment 2a), due to the more restricted agent movement. However, it was hypothesised that coordination would still be achieved some of the time. It was expected that with more districts the coordination rate would be lower (and TTC_{100} higher), because of the further restriction on movement.

For all configurations it was anticipated that the choice of clustering mechanism wouldn't affect the coordination rate or TTC_{100}. The main impact of using district clusters was expected to be less variety in the sizes of the clusters.

7.8.3 Results

Figure 7.14 shows the win rate by cluster size for the simulations with 4 districts, using k-means clustering. Both the actual win rate and the expected win rate (calculated from the theory) are shown. Clearly the introduction of districts hasn't changed the success rate at which the agents coordinate. The same was true for $D = 6$, and using district clusters instead of k-means clustering.

Table 7.4 shows the coordination rates and TTC_{100} values for the simulations with both a static and mobile monument. With 4 districts, coordination was achieved 100% of the time, even with a static monument. This was because the monument was visible from at least one cell in each district. However, with 6 districts this was not the case, therefore with the static monument the coordination rates were below 100%. For the same configuration without districts (found in experiment

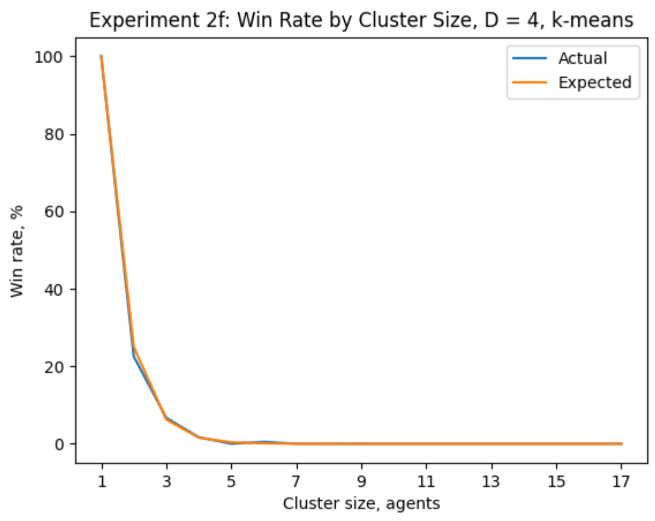


Figure 7.14: Experiment 2f win rate by cluster size, $K = 4$

				TTC_{100}		
Monuments	**D**	**District Clusters**	**Coordination**	**Min**	**Median**	**Max**
Static	4	No	100%	9	14	17
		Yes	100%	8	12	18
	6	No	40%	24	51	78
		Yes	80%	9	55.5	70
Moving	4	No	100%	8	9	10
		Yes	100%	8	10	10
	6	No	100%	7	9	10
		Yes	100%	9	9	14

Table 7.4: Experiment 2f: Results with monuments

2a), coordination was achieved 100% of the time. With $D = 6$, the increase in coordination when switching from k-means clustering to district clusters is most likely a result of the small sample size.

The mobile monuments increased the coordination rate to 100% in all cases. Through one complete circle of the grid, the monument's visible region covered the entire grid. This, coupled with the fact that the agents weren't travelling far, meant the times taken for coordination to be reached were very low. Changing the number of districts made no significant difference, and the same was true for the choice of clustering algorithm.

There was significant variation in the time required for the agents to coordinate when $D = 6$ and the monument was static. This was primarily due to the random agent spawn locations. If all the agents spawned in districts that could view the monument (or very close to these districts), coordination was achieved quickly (in as little as 9 rounds). However, sometimes it took up to 78 rounds, and on other occasions coordination was not achieved at all. This occurred when many agents spawned in districts that couldn't view the monument. An extension experiment would be to adjust the agent implementation, varying the likelihood with which agents travel to districts other than their home district.

7.9 Experiment 3: Remembering Agents

7.9.1 Setup

This experiment aimed to improve agent coordination through use of previous results rather than a monument. It shares similarities with the work of Crawford and Haller [23]. The experiment configuration was:

- G = {6, 9}
- N = {30, 60}
- M = 4
- O = 4
- K = 6
- C = {2, 3, 4, 5}
- L = 0
- D = 0
- R = 100

A 6x6 grid was used when N was 30, and a larger 9x9 grid was used in the cases where $N = 60$. Each configuration was simulated 5 times. The number of clusters remained the same throughout all simulations, therefore with 60 agents the average cluster size was double the size compared to when there were 30 agents.

7.9.2 Agent Implementation

This experiment used a new agent implementation, known as the *Remembering Agent*. The idea behind this agent is that it would use the outcome of each focal point game to influence its future choices. This was achieved through the use of a sentiment system - each agent would maintain a score for each of the options, representing its opinion about that option. The score for each option was initialised to 0, and for every successful focal point game, the option that the agent chose would have its score increased by 5. After a loss, the score for the chosen option would be decreased by 1. When playing focal point games, the agents would choose the option they had the highest opinion of (the option with the highest sentiment score). In the event of multiple options being tied for the highest score, the agent would choose randomly from these options. The agents' movement was random.

The agent implementation centres around the idea of contributive justice - the agents are 'rewarded' for positive contributions to the collective. The score increment in the case of a win was purposefully higher than the decrement in the case of a loss, which made positive outcomes have a longer lasting effect. An alternative would have been to treat success and failure equally, however the impact of a win would immediately be wiped out by a single loss. Such a mechanism would arguably be less conducive to coordination.

7.9.3 Hypothesis

The hypothesis for this experiment was that coordination would be improved relative to the random agent baseline. There were three independent variables, N, G, and C, and three dependent variables, win rate by cluster size, TTC_{100}, and coordination rate. It was expected that with fewer agents per cluster, the difference compared to the random baseline would be bigger, because the agent implementation relies initially on wins by chance (to significantly distinguish one of the options), and this was more likely for smaller cluster sizes. Similarly, it was anticipated that the difference compared to the baseline would be bigger for fewer choices, for the same reason that 'lucky' wins are more likely.

7.9.4 Results

The results of the simulations showed varying degrees of success. With 30 agents and 2 choices, coordination was achieved 100% of the time, with a median TTC_{100} of 30 rounds. As the number of choices increased, the agents were less successful. With 3 choices, total coordination was achieved on a single occasion, but with 4 and 5 choices, the agents were unable to fully coordinate. This shows that wins by chance increased the likelihood of coordination (note that for a cluster of 5 agents,

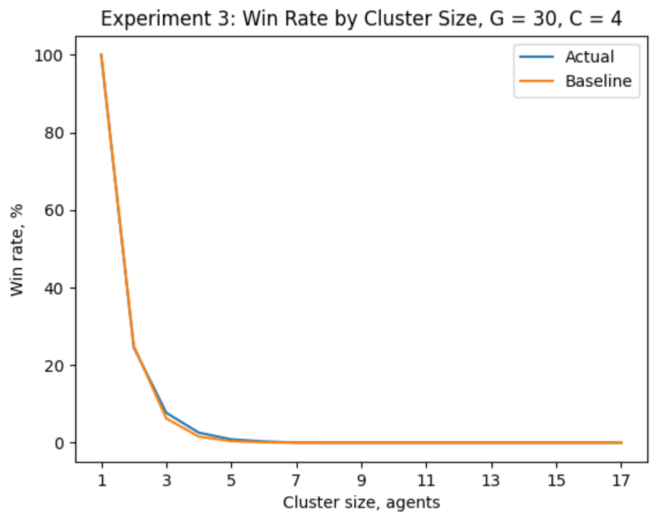


Figure 7.15: Experiment 3 win rate by cluster size, $G = 30$, $C = 4$

random coordination with 5 choices is 97.44% less likely than with 2 choices). With 60 agents, total coordination was never achieved.

Figure 7.15 shows that with 4 choices the 30 agents performed about as well as the random agents. A small out-performance was seen in clusters of 3 and 4 agents, but for all other cluster sizes the difference was negligible. Increasing the number of agents to 60 saw a slight decrease in coordination, as seen in Figure 7.16. In this case, for clusters of 3 or 4 agents the Remembering Agents performed slightly worse than random. These results can be explained by the increased cluster stability observed with fewer agents; that is, with fewer agents it is more likely that the same agents will be clustered together in consecutive rounds. This, coupled with the fact that the agent implementation ensures that once a cluster has coordinated, it will continue to coordinate if the agents in the cluster remain the same, explains why performance was better with 30 agents rather than 60.

Figure 7.17 shows a typical win rate graph with 30 agents and 2 choices. The graph is quite different to those seen in previous experiments. The win rates begin low, approximately the same as the baseline, and then slowly increase. The gradient gradually becomes steeper, until coordination finally reaches 100%. Overall, we see an exponential increase in the win rates, capped at 100%. Intuitively this makes sense, as it takes some time for the agents to build knowledge of the game, and form opinions of the options.

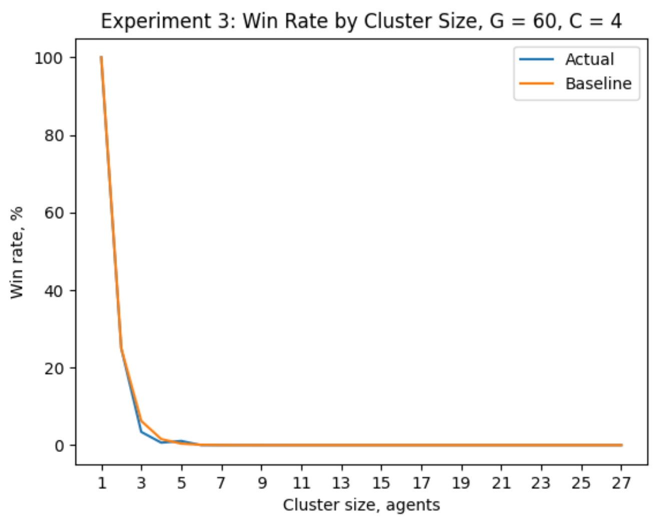


Figure 7.16: Experiment 3 win rate by cluster size, $G = 60$, $C = 4$

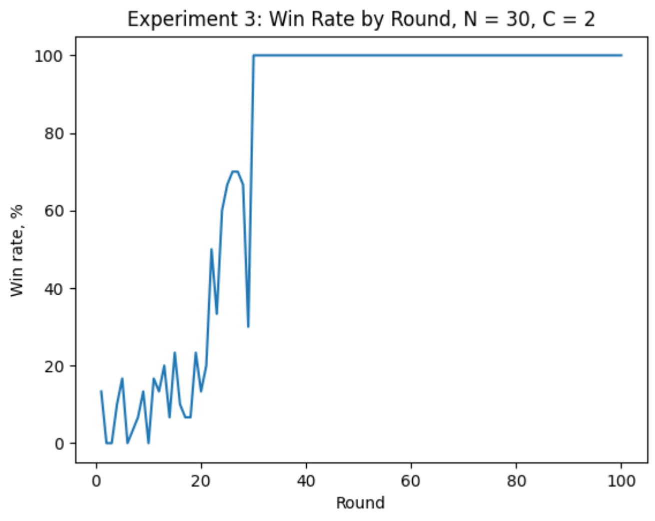


Figure 7.17: Experiment 3 typical win rate, $C = 2$, $N = 30$

7.10 Experiment 4a: Talkative Agents

7.10.1 Setup

This experiment utilised agent communication for the first time. The idea was to build on the Remembering Agent, with agents sharing their experiences with each other. The experiment configuration was:

- G = {6, 9}
- N = {30, 60}
- M = 4
- O = 4
- K = 6
- C = {2, 3, 4, 5}
- L = 0
- D = 0
- R = 100

The configurations matched those used in experiment 3, so a 6x6 grid was used for 30 agents, and a 9x9 grid for 60 agents. The only change was that each configuration was simulated 10 times rather than 5.

7.10.2 Agent Implementation

A new agent implementation, the *Talkative Agent*, was used for this experiment. The same sentiment system was used, but with additional adjustments to the scores. During each communication round, each agent sent exactly one message to each other agent in the same cell. The message sent by each agent contained their most recent choice and most recent result. The receiving agent would then update their sentiment scores based on the message, increasing the score for a choice by 5 if coordination was achieved, and decreasing the score by 1 otherwise. The benefit to this strategy is that agents can learn about successes without being in the clusters. Additionally, agents have to be in the same cell to communicate, and as they are clustered by location, there is a higher chance that agents that have communicated will be clustered together (compared to randomly grouping the agents).

7.10.3 Hypothesis

It was hoped that the new agent implementation would lead to better coordination. This would be measured through three dependent variables - win rate by cluster size, coordination rate, and TTC_{100}. As with experiment 3, the independent variables were G, N, and C. It was expected that with fewer choices, higher rates of coordination would be achieved. As we saw with experiment 3, wins by chance are important for the sentiment scores to be useful, and these are more likely with

			TTC_{100}		
Agents	**Choices**	**Coordination**	**Min**	**Median**	**Max**
30	2	100%	5	18	94
	3	30%	16	31	79
	4	40%	23	45.5	68
	5	20%	13	21.5	30

Table 7.5: Experiment 4a results with 30 agents

fewer choices. Additionally, smaller clusters are more likely with 30 agents rather than 60, so better coordination was expected with 30 agents. Even if full coordination wasn't achieved, it was hypothesised that the win rate by cluster size would be higher relative to the random agent baseline.

7.10.4 Results

The results with $N = 30$ are displayed in Table 7.5. As with the Remembering Agents, coordination was achieved 100% of the time when the agents only had two options to choose from. However, the median time for coordination to be achieved was reduced by 40%, down to 18 rounds from the 30 in experiment 3. This shows that agent communication can further improve coordination. Additionally, in some simulations total coordination was achieved with $C > 2$. This result was previously only observed for $C = 2$. On average, coordination was achieved 30% of the time when the agents had more than 2 options to choose from. This shows that coordination is not hardwired into the system; some confluence of events is required for coordination to emerge. In the successful coordination instances, the series of events resulted in norm emergence - all the agents knew which option to choose, and there was no external influence that affected the choice.

Figure 7.18 shows the win rate by cluster size for the simulations with $G = 60$ and $C = 4$. There was no noticeable difference between the performance of the Talkative Agents and the random agent baseline. The same was true for $C = 2$. With $C = 3$ the agents coordinated slightly less than the random agents, but with $C = 5$ the agents performed slightly better than the baseline.

As the agent implementation doesn't guarantee coordination, and instead a series of events are required for its emergence, the number of rounds for which the simulation takes place has an impact on the coordination rate. If a higher number of rounds had been simulated, it is likely that the coordination rates would have been higher (and maybe some coordination would have been achieved even with 60 agents). Figure 7.19 shows the win rate by round for one of the simulations with $G = 30$ and $C = 4$. Here, win rate is defined as the percentage of agents that successfully coordinate in a round. For much of the simulation, the win rate was quite low

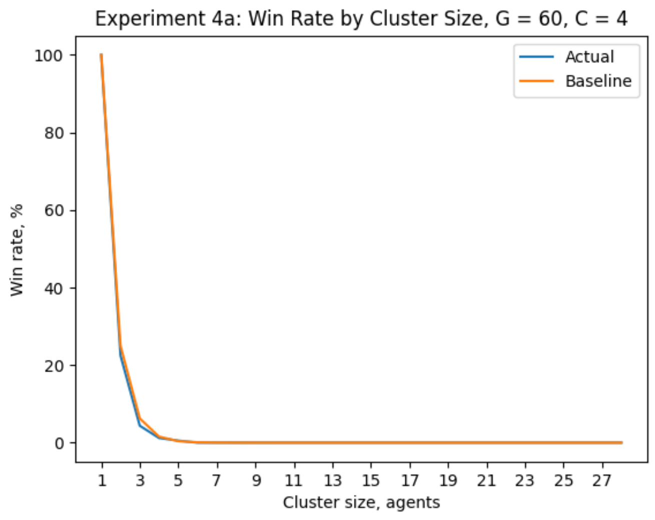


Figure 7.18: Experiment 4a win rate by cluster size, $G = 60$, $C = 4$

and consistent. However, significant improvements were seen in the last 20 rounds. Whilst full coordination wasn't achieved, it appears as though the agents were close to establishing a norm, and if the simulation were to have continued, it is likely that a norm would be agreed upon quickly.

Figure 7.20 shows the win rates for a simulation with the same configuration where 100% coordination was reached. Coordination initially begins very low, as the agents don't have any preference and just choose randomly. Then, after about 30 rounds, the coordination begins to increase, as the news of a win by chance spreads among the agents. Eventually, all agents become aware of the norm that has emerged, and all are able to coordinate.

7.11 Experiment 4b: Talkative Agent Density

7.11.1 Setup

The objective of this experiment was to explore the effect of agent density on coordination, using the Talkative Agent implementation. With the agents communicating, agent density is particularly important as it determines how frequently agents will meet and interact. The simulation configurations were:

- G = {4, 5, 6}
- N = {10, 15, 20, 22, 25, 30}
- M = 4
- O = 4
- K = {2, 3, 4, 5, 6}
- C = 4
- L = 0
- D = 0

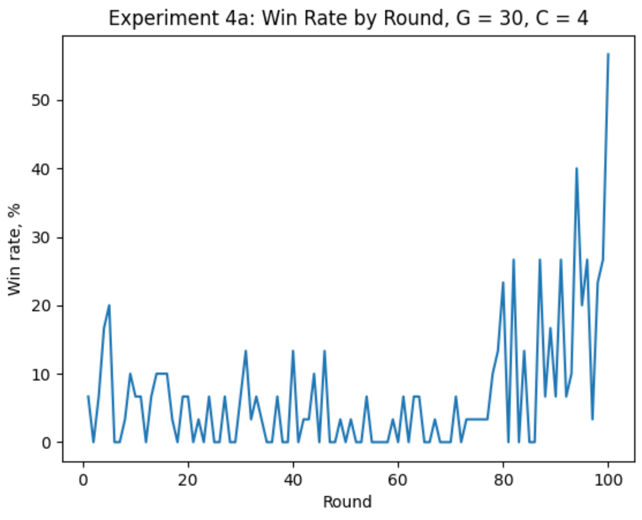


Figure 7.19: Experiment 4a win rate by round, $N = 30$, $C = 4$

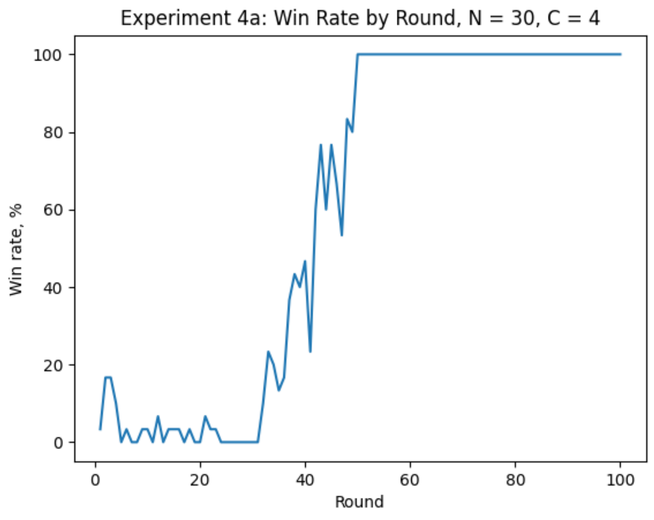


Figure 7.20: Experiment 4a typical win rate, $N = 30$, $C = 4$

- R = 100

The experiment was split into two sub-experiments. For each configuration in each sub-experiment, the simulation was run 10 times.

The first sub-experiment varied the agent density while keeping the grid size constant (at $G = 6$). The number of agents was increased from 10 to 30 in steps of 5, while K was set such that the average cluster size was 5 agents.

The second sub-experiment kept the agent density approximately constant, while changing the grid size. For these simulations, the grid size was $G \in \{4, 5, 6\}$, the number of clusters was $K \in \{2, 3, 4\}$, and the number of agents was $N \in \{10, 15, 22\}$. The aim was for the density to be kept as close as possible to 0.6 agents per cell, and the average cluster size kept close to 5. With 22 agents there were 4 clusters, which meant that the average cluster size was slightly larger than it was for the other configurations. Theoretically this made coordination more difficult, and will be taken into consideration during analysis of the results. The simulations with $N = 10$ and $G = 4$ had the highest agent density (0.625), while the simulations with $N = 15$ and $G = 5$ had the lowest density (0.6).

7.11.2 Hypothesis

This experiment varied three independent variables: G, N, and K. The dependent variables were win rate by cluster size, coordination rate, and TTC_{100}. The main hypothesis was that as density increased, the coordination rate would increase, and the time taken for coordination to be achieved would decrease. This was because with a higher agent density, each agent is more likely to meet and coordinate with other agents. As information reaches more agents, coordination should improve as the agents were able to make more informed choices. Additionally, it was expected that for the same agent density, a larger grid with more agents would be better for coordination. This was because with more agents there would be more clusters, which increases the likelihood of a win by chance, and it had previously been found that wins by chance were necessary for achieving total coordination with this agent implementation.

7.11.3 Results

By increasing the agent density for a given grid size, we observed increasing coordination. This can be seen in Table 7.6, with the coordination rate increasing from 0% with a density of 0.2778 agents per cell to 40% with a density of 0.8333. Due to the low coordination rates, the sample size of times taken for the agents to coordinate was too small for analysis, however the hypothesis that coordination rate would increase proved true. This was for two reasons - information was able to spread more

Agents	Density	Coordination Rate
10	0.2778	0%
15	0.4167	0%
20	0.5556	10%
25	0.6944	10%
30	0.8333	40%

Table 7.6: Experiment 4b results, varying density

				TTC_{100}		
Grid	**Agents**	**Density**	**Coordination**	**Min**	**Median**	**Max**
4	10	0.625	50%	3	29	99
5	15	0.6	20%	43	69.5	96
6	22	0.61111	0%	N/A	N/A	N/A

Table 7.7: Experiment 4b results, varying grid size

easily due to the increased number of agent interactions, and with a higher density there were more clusters, so even though the average cluster size was the same, the likelihood of coordination by chance in any given round was higher. In the simulations where coordination was not achieved, the success rates were either in line with or marginally above the baseline.

The second part of the experiment kept the agent density approximately constant, while varying G. The results from these simulations are shown in Table 7.7. Contrary to the hypothesis that a larger grid would be better for coordination, it is clear from the results that for a given agent density, a smaller grid is more conducive to coordination. One important point is that the average cluster size with $G = 6$ was 5.2 agents, whereas for the other simulations the average cluster size was 5. This made coordination slightly less likely with $G = 6$. Looking at the frequency of cluster sizes for each configuration, and assuming baseline performance, with $G = 4$, 7.64% of clusters would be expected to coordinate, compared to 7.25% of clusters with $G = 6$. Therefore, it was approximately 5% harder for coordination to be achieved with the 6x6 grid and 22 agents, so whilst coordination was never achieved, the reduced coordination rate can only be partially explained by the increased average cluster size. The rest of the reduction must therefore be due to the larger grid size.

These results came as a surprise, but there are a few possible reasons why better coordination was observed with smaller grids and fewer agents. With fewer agents and clusters, when a cluster wins this represents a larger proportion of the agents. Additionally, with fewer agents and a smaller grid, information doesn't need to travel as far. Also, with fewer agents the likelihood that the same group of agents get clustered again is higher, so after a cluster coordinates, there is a higher probability that cluster will persist and continue to coordinate.

7.12 Experiment 5a: Social Agents

7.12.1 Setup

This experiment introduced another agent implementation, the *Social Agent*. This agent shares a lot of similarities with the Talkative Agent, except for the fact that it shares and receives information over a social network rather than via communication with agents based on location. The configuration for this experiment was:

- G = {6, 9}
- N = {30, 60}
- M = 4
- O = 4
- K = 6
- C = {2, 3, 4, 5}
- L = 0
- D = 0
- R = 100

The configuration was the same as it was for experiments 3 and 4a, making the results directly comparable. Each configuration was simulated 10 times.

7.12.2 Agent Implementation

As previously mentioned, this agent used a social network for communication with other agents. Whenever two agents met, if they hadn't previously met they would add each other to their social networks. This was implemented as a HashSet to store agent IDs, and an ArrayList to store the agents. This allows agents to send messages to each other whenever they like (as long as execution is with the agent at that point), and not just when the agents' `communicate` method is called. Upon receiving a result, the Social Agent broadcasts it to all the agents in its social network. The same sentiment system is used as with the two previous agent implementations.

7.12.3 Hypothesis

This experiment varied three independent variables, G, N, and K. The main dependent variables were win rate by cluster size, coordination rate, and TTC_{100}. It was hypothesised that coordination would be improved with the introduction of the social network. This was because information is able to spread quicker and further. As a result, it was expected that coordination rates would increase, and the time taken for coordination to be achieved would decrease. Additionally, it was once again expected that the agents would be better at coordinating with fewer choices, and when the average cluster size was lower (i.e. with $G = 30$ rather than $G = 60$).

		TTC_{100}		
Choices	**Coordination**	**Min**	**Median**	**Max**
2	100%	5	8.5	18
3	100%	7	14.5	27
4	90%	9	15	62
5	100%	8	14	19

Table 7.8: Experiment 5a results, $G = 30$

		TTC_{100}		
Choices	**Coordination**	**Min**	**Median**	**Max**
2	60%	11	12.5	28
3	10%	16	16	16
4	0%	N/A	N/A	N/A
5	0%	N/A	N/A	N/A

Table 7.9: Experiment 5a results, $G = 60$

7.12.4 Results

The results for the simulations with 30 agents are shown in Table 7.8. The first point to note is that the coordination rates are significantly higher than they were for the same simulations with the Talkative Agents (100% coordination with 5 choices, up from 40%, for example). Coordination was achieved in all simulations except from once with $C = 4$. It was also observed that it took less time for the agents to coordinate - the median TTC_{100} with $C = 2$ was 8.5 rounds compared to 18 rounds with the Talkative Agents. Similar results were seen for all values of C tested. The time taken to coordinate was relatively stable as the number of choices increased from 3. The only exception was with 4 choices, when on one occasion it took 62 rounds for coordination to be achieved, and in another simulation the agents failed to reach total coordination at all. The results would suggest that the difficulty of coordination is not monotonically increasing as the number of choices increases. Further experiments would be required to explore this in more detail, with larger sample sizes and higher values of C tested.

With 60 agents, coordination was achieved less often, as can be seen in Table 7.9. For $C = 4$ and $C = 5$, total coordination was never achieved. When coordination was achieved, it took longer than it did for the same number of choices with 30 agents. Comparing performance to that observed of the Talkative Agents in experiment 4a, coordination was improved. Previously, the agents were unable to totally coordinate, even with 2 choices, but the Social Agents were able to coordinate 60% of the time. When coordination wasn't achieved, the success rates were in line with the baseline.

Figure 7.21 shows a typical win rate graph for a simulation with 30 agents picking

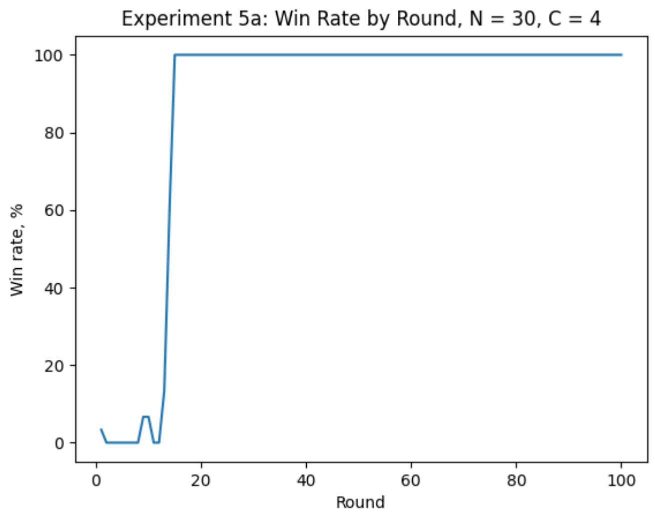


Figure 7.21: Experiment 5a typical win rate, $N = 30$, $C = 4$

from 4 choices. Another new pattern is observed, whereby a step change occurs, with coordination increasing from 0% to 100% in just 2 rounds. Thanks to the social networks, news of a win by chance can spread much more effectively than with the Talkative Agent. Hence, it was often the case that if a win by chance for a large enough cluster occurred, total coordination would be achieved very quickly afterwards. Wins by chance earlier in the simulation are less likely to lead to coordination, as the social networks haven't had enough time to establish. Additionally, if the social networks are of a reasonable size, but the size of the winning cluster is insignificant (1 or 2 agents, for example), then less of an emphasis is placed on that success, and coordination again may not be achieved.

Overall, we have seen that the Social Agent is more effective at coordinating, particularly with smaller cluster sizes. However, as both average cluster size and number of choices increased, the Social Agent was unable to perform better than the random agent baseline ($G = 60$, $C = \{4, 5\}$). The increased performance does come at a cost - there was much more communication between agents, and if communication is expensive this may not always be feasible. Another observation was that coordination tended to be achieved early on or not at all - this was particular pronounced in the simulations with $C = 2$ and $N = 30$, where the maximum recorded TTC_{100} was 28 rounds, yet coordination wasn't achieved in 4 of the 10 simulations. The reason for this was looked at in detail in a later experiment.

7.13 Experiment 5b: Social Agent Density

7.13.1 Setup

This experiment was very similar to experiment 4b in that it explored the effect of agent density on coordination, although this time using the Social Agent implementation rather than the Talkative Agent implementation. The configuration for this experiment was:

- G = {4, 5, 6}
- N = {10, 15, 20, 22, 25, 30}
- M = 4
- O = 4
- K = {2, 3, 4, 5, 6}
- C = 4
- L = 0
- D = 0
- R = 100

Once again, the experiment was split into two sub-experiments, first varying density for a fixed grid size, and then varying grid size with a fixed density.

7.13.2 Hypothesis

This experiment varied three independent variables: G, N, and K (with N and G combined to vary density). The dependent variables were win rate by cluster size, coordination rate, and TTC_{100}. Based on the results of experiment 4b, it was hypothesised that increasing density would improve coordination for a fixed grid size, and decreasing grid size would improve coordination for a fixed agent density. It was also expected that the coordination rates seen would be higher than they were in experiment 4b, as the agent implementation had already proved effective in experiment 5a.

7.13.3 Results

The results for the first sub-experiment, where grid size was kept constant, are shown in Table 7.10. In experiment 4b we observed that coordination was achieved more often as the agent density increased, when using the Talkative Agent implementation. However, the same was not true for the Social Agent - there was no overall trend observed in the data. This is most likely because the coordination values recorded are much higher than they were in experiment 4b, so the range of values is much lower. Therefore, the lack of a trend may be because of the sample size; if more simulations had been run, a trend may have emerged. Additionally, as the agents are much better at coordinating, the coordination rate even at a density of

			TTC_{100}		
Agents	**Density**	**Coordination**	**Min**	**Median**	**Max**
10	0.2778	70%	8	17	22
15	0.4167	80%	11	14.5	51
20	0.5556	100%	8	11.5	71
25	0.6944	80%	6	9.5	25
30	0.8333	90%	9	15	62

Table 7.10: Experiment 5b results, varying grid size

				TTC_{100}		
Grid	**Agents**	**Density**	**Coordination**	**Min**	**Median**	**Max**
4	10	0.6250	100%	4	9.5	69
5	15	0.6000	90%	6	8	63
6	22	0.6111	50%	12	18	60

Table 7.11: Experiment 5b results, varying grid size

0.2778 agents/cell is quite high, so lower densities could also be experimented with (this would require a change in the average cluster size to avoid a single cluster).

The small range in coordination rates tells us that agent density has little impact on coordination. When compared to the Talkative Agent, this can be explained by the fact that agents don't need to meet in order to communicate (as long as they have met at some point in the past), so agent density has a lesser effect on communication rate.

In the second sub-experiment, varying the grid size with an approximately constant density, we observed the same pattern as in experiment 4b - coordination increased as the grid size decreased. This can be seen in Table 7.11. The time taken for coordination to be achieved was similar for the 4x4 and 5x5 grids, but increased significantly with the 6x6 grid (the TTC_{100} increased to 18 rounds, up from 9.5 and 8 for $G = 4$ and $G = 5$ respectively). Compared to the Talkative Agent, coordination rates were significantly higher, and when the agents were able to fully coordinate, they achieved this quicker.

With a smaller grid size, information did not need to travel as far, and with fewer agents, there was an increased probability that an agent would meet all other agents, and add them to their social network. Having a larger social network allows an agent to send and receive more information - this can help the agent to make a more informed choice, but there is also the possibility that the agents could receive too much information, such that the important points become lost amongst mostly noise. The better performance with fewer agents could also be partially explained by this. The idea is explored in detail in the final experiment.

7.14 Experiment 5c: Social Network Size

7.14.1 Setup

As we have observed in the previous two experiments, it was often the case that with the Social Agent implementation, coordination would either be achieved early on in the simulation, or not at all. One potential theory was that the agents would only coordinate if their social network was below a certain size (with the size varying based on the simulation configuration). To test this theory, this experiment used two techniques to limit the size of an agent's social network. If the theory was correct, the introduction of this limit would increase coordination rate.

For this experiment, two configurations were chosen from previous experiments where the observed pattern, quick coordination or no coordination, was most noticeable. These were:

- G = {6, 9}
- N = {10, 60}
- M = 4
- O = 4
- K = {2, 6}
- C = {2, 4}
- L = 0
- D = 0
- R = 100

The first configuration featured a 6x6 grid, with 10 agents picking from 4 choices, with 2 clusters. The other configuration used a 9x9 grid, with 60 agents picking from 2 choices, with 6 clusters. The previously observed coordination rates were 70% and 60%, with maximum TTC_{100} values of 22 rounds and 28 rounds respectively.

7.14.2 Agent Implementation

Two adjustments were made to the Social Agent implementation to limit the size of the social network. First, a check was applied so that if the social network was full, no further agents would be added to it. Second, the limit was applied on a rolling basis. This required a change to one of the data structures used in the implementation, as the agent would keep only the most recent connections in its social network. To achieve this, the ArrayList of agents was replaced with a List of agents, which allows O(1) removal of the front (head) of the list. Whenever an agent met another agent not already in its social network, the agent would be added to the back of the list. If this caused the social network to exceed the size limit, the agent at the front of the list would be removed. This implements a least-recently-added (LRA) eviction policy.

Grid	Agents	Choices	SN Limit	Coordination
6	10	4	None	55%
			7 (static)	50%
			7 (LRA)	55%
9	60	2	None	55%
			18 (static)	50%
			18 (LRA)	40%

Table 7.12: Experiment 5c results

To find the limits to apply to the social network sizes, experiments were conducted using the previous Social Agent implementation without a limit. Each of the configurations previously outlined was simulated 20 times. Code was added to the simulator to log the average social network size in each round. From this, the average network size when coordination was achieved could be found, and this would be used as the limit.

7.14.3 Hypothesis

This experiment varied the social network size and limitation technique, and measured the effect on, in particular, coordination rate. It was expected that by limiting the size of an agent's social network, they wouldn't be overwhelmed with information, and coordination rates would increase. Additionally, it was hypothesised that coordination rates would be higher with the LRA eviction policy, rather than the unchanged social network when the limit was reached. This was because the LRA policy ensures nearby agents are added to the social network, so the agents are more likely to receive information from and send information to agents in the same or nearby clusters.

7.14.4 Results

The median social network sizes for the two configurations were found from a series of 40 (20 for each) simulations. These were 7.6 agents (with $N = 10$) and 18.7 agents (with $N = 60$). The floor of these values (7 and 18) were then used as the limits in the subsequent experiments. The full results are shown in Table 7.12.

Contrary to the hypothesis, limiting the size of a social network did not improve coordination rates. All were unchanged or marginally lower, with the exception of the simulations with 60 agents and the LRA policy, where coordination dropped to 40%. Therefore, the hypothesis did not prove true, and there must be some other reason why agents seemed to coordinate early on or not at all.

To investigate further, a final agent implementation was used, which built on the

Grid	Agents	Choices	Coordination
6	10	4	100%
9	60	2	95%

Table 7.13: Experiment 5c results - part 2

original unrestricted social network, but with an added mechanism which would reset an agent's sentiment scores after an unsuccessful period. This was achieved by setting each sentiment score back to zero after 20 rounds of not being in a coordinating cluster. The idea behind this mechanism is that it appeared as though the agents were 'stuck in a rut', and by wiping the scores clean, it was possible that the agents would be able to escape the cycle of incoordination. A 'refresh rate' of 20 rounds was chosen as in the vast majority of simulations with the Social Agent, coordination was achieved within this time-frame.

The results of this new agent implementation are shown in Table 7.13. It is clear from the increased coordination rates that this simple change was massively beneficial, and goes a long way to explaining the issues of previous implementations. Looking at logs from previous experiments, it became clear that after a while each agent would consistently make the same choice, even if it was unsuccessful. This pattern is likely due to the fact that after a while each agent has met every other agent, and therefore the social network graph is an all-to-all graph, and each agent makes the same updates to their sentiment scores each round. By resetting the sentiment scores, we re-introduce randomness to the system, increasing the likelihood of wins by chance, and therefore overall coordination.

Chapter 8

Evaluation

8.1 Simulator Evaluation

The requirements for the simulator were outlined in Chapter 3. Each of these requirements has a corresponding check to see if the requirement was met, with the numbers of the checks matching the numbers of the requirements:

1. Simulation run with 100 agents
2. Visual inspection and usage
3. Code evaluated to assess easiness of adding new games
4. Visual inspection of clustering and results
5. Measure frame rate under ‘normal’ load - same configuration as for baseline experiment with 30 agents
6. Data exported from simulation and imported successfully into Python
7. Visual inspection of graphs
8. Inspection of the code to understand the steps necessary to add code

The results of each of these checks were:

1. Simulations were successfully run with 100 agents, with various configurations of the other variables.
2. Most of the simulation configuration is completed by typing in the text fields. These text fields have built-in checks to ensure values are within the allowed ranges. It is clearly visible when these checks are violated. The placement of agents, monuments, and districts is quite intuitive, although multiple buttons could have been added so that it takes less clicks to choose what to place, as currently the user must cycle through all options via a single button.

3. There are four steps that the user has to take to add their own focal point game to the simulator. Three of the steps are very quick and simple, whilst the fourth step involves creating the win function itself. The instructions to add a game are made very clear through the use of comments in the code.

4. Agent clustering is very clear, as each cluster has a distinct colour. There may be issues for colourblind users, but the code makes it easy to change the cluster colours if the user desires. The results of each focal point game are visualised by changing the colour of the agents' outlines from black to green or red. For the most part this is clear, although sometimes there are agents with the same outline colour and fill colour (e.g. red cluster that isn't successful), which makes the result less clear. The cluster colours could possibly be changed so that this doesn't happen. Furthermore, there is no way of seeing what option each of the agents has chosen without looking at the logs after the simulation. One option would be to write the option inside of each agent, but with larger grids the agents are quite small, so the text may not be legible. An alternative method of showing choices would be preferable. Finally, the colours chosen for the clusters are very distinct, but this came at the cost of limiting the number of clusters to 6 (unless district clusters are used). An improvement would be to allow more than 6 clusters if the user is okay with not seeing the clusters visualised using different colours.

5. A simulation was run using a baseline configuration, with 30 agents and a 6x6 grid. Processing provides access to a `frameRate` variable, which was accessed to establish the frame rate. The simulator is configured to run at 30 frames per second, and during the simulation the frame rate was 29 or above more than 95% of the time, and 24 or above more than 99% of the time, hence we can conclude that the simulator did run smoothly at at least 24 frames per second.

6. The simulator logs data using JSON, and during the experiments these logs were used in Python to extract the data and produce various graphs.

7. Up to four graphs are displayed that show statistics in real time as the simulation happens. These graphs are easy to read, particularly as exact numbers are less important than overall trends and patterns. One small issue is that the graph displaying the win rate over time does not have an x-axis until 5 rounds have passed. This is because before 5 rounds are reached, fractional round labels would be displayed, which is undesirable. Delaying showing the x-axis until 5 rounds in prevents this issue, and before it appears so few rounds have been completed, so the issue is very minor. There is significant scope to improve the selection of graphs and statistics, and to allow the user to configure what they would like to see. For example, a leaderboard showing the most and least successful agents could be added, which would be particularly useful if multiple agent implementations are used simultaneously.

8. The three main ways the user will most likely add to the code are custom focal point games, which have already been discussed, new agent implementations, and new monument implementations. The code has been designed so that adding agent and monument implementations is as simple as possible. Each has a base class that can be extended with the custom implementation. There are not too many methods to override, and it is clear what each method does. To instantiate any custom implementations, there is only one place in the code that needs changing. This makes it simple for the user, and significantly reduces the chance that the user misses a step. Additionally, the steps are outlined in the user guide.

Overall, the simulator has met (to some degree) all of the specified requirements. However, it is by no means perfect, and there are many possible improvements and enhancements that could be made. This will be discussed in Chapter 9.

8.2 Strategy and Solution Evaluation

Each agent implementation was separately evaluated during the discussion of the results of the experiments. The strategies had differing degrees of success, which is to be expected given the different types of implementations tested. The strategies employed used two main methods to encourage coordination - monuments, and a sentiment system. The methods using monuments were the most successful under the right conditions, and were less affected by the number of choices presented to the agents. The sentiment system implementations increased in performance as the amount of communication between agents increased. In applications where communication is difficult or expensive, balancing this trade off will be key.

Previous work on focal points has tended to focus on focal point games where the options have distinguishable differences, be it physical [21, 30] or non-physical [16], and experiment with strategies that lever these differences. This project took an alternative approach, particularly with the sentiment system based strategies, as the options are much more arbitrary. There need not be a single red block amongst a group of blue blocks. Instead, the strategy can succeed if the blocks are all different colours, and with no other differences. Numbers were used to represent the different choices, but the agents didn't need any understanding of numbers for the sentiment strategies to work. While the monument-based strategies did use the concept of high or low numbers, the required understanding of numbers could easily be removed by changing the symbols, for example writing numbers on the monuments (then the agent would just choose the option written on the monument).

8.3 Overall Evaluation

This project has resulted in the creation of a novel focal point game simulator. The concept of agents moving in space, playing repeated focal point games, has not knowingly been explored to this degree before. Experiments carried out in related works appear to use command-line based simulators, and there is no other known simulator that visualises the playing of focal point games in the way this project does.

The breadth of potential studies in this field is significant, hence the solutions and strategies explored in this work differ massively from related works. In particular, each study tends to investigate coordination methods in different scenarios, therefore it is not straightforward to compare the effectiveness of strategies without testing them under the same conditions. Clearly different coordination mechanisms will have merits in different situations - if coordination is impossible, the social-network based strategies won't be viable, but the monument based strategies offer a promising alternative.

One of the challenges of the project was deciding which solution mechanisms to simulate, given the almost limitless number of possibilities. For that reason, the mechanisms were deliberately chosen to be different to those explored before, and to align as closely as possible to the idea of coordination through social construction. The mechanisms centred around decentralisation, one of the key facets of social construction. Whilst the variety of experiments was good, a slightly smaller number of experiments, but with more repeats per experiment, may have been better to obtain more meaningful results. Later experiments featured 20 repetitions per configuration, which was a good improvement on the 5 repetitions of the earlier experiments, but even this could be viewed as insufficient.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

9.1.1 Coordination Mechanism Effectiveness

Overall, we have seen how agents can learn socially constructed reasons why one choice is better than another in focal point games. Different mechanisms were used to achieve this, with differing degrees of success, and different potential applications.

We began by experimenting with random agents - agents that would both move and choose randomly. The results obtained supported the underlying theory, and showed that coordination by chance, particularly with many options and many agents, is very unlikely. Humans are able to perform much better than random [1], and we saw this in the later experiments.

Communication increased the levels of coordination observed, in the first instance with the Talkative Agent implementation. We observed how norms could emerge, with exponential increases in coordination (up to 100%) as agents shared results with one another. Additionally, we saw how increasing agent density, or decreasing the grid size, increased the coordination rate.

Agents were often very successful when using social networks to communicate and coordinate. However, there were times when social networks would saturate, meaning each agent would receive exactly the same information, which wasn't conducive to coordination, and the agents would end up stuck in a losing cycle. Some difference in information received, resulting in coordination by chance, was better. By resetting the sentiment values after a long period of unsuccessful coordination attempts, we saw how agents were able to start from a clean slate, significantly increasing the likelihood of coordination.

One of the downsides of using a social network is that in practice communication may be expensive, or even impossible. The strategies centred around monuments overcame this issue, and allowed agents to coordinate even with many choices to choose from. We saw how multiple monuments could increase the speed of coordination, but this required the agents to be proactive at updating the symbols on the monuments, in the hope that eventually all monuments would be consistent. These experiments also showed the importance of movement - if two monuments were far apart and displaying different symbols, and the agents weren't moving much, coordination was much less likely than it was if the agents were moving more freely.

We also observed that if the agents aren't willing to move very far, as was the case with the Homely Agents, moving the monument offers another route to quick coordination.

9.1.2 Challenges and Achievements

One of the early challenges was choosing the technologies to use to build the simulator. Thorough research and advice informed the decision to use Processing, even though it was an unfamiliar language. This was arguably one of the best decisions of the project, as Processing proved lightweight enough that many aspects of the simulator were simpler to implement than anticipated, while also providing access to the best parts of Java, such as the collections.

The design and implementation of the simulator was very methodical, with the complexity increased in small increments to allow for continuous testing. The resulting simulator met the requirements specified at the beginning of the project, and is unlike anything found in related work. The simulator embodies the success of the project - it is comprehensive and easy to use, although with scope for enhancements in the future.

The hardest part of the project was deciding what experiments to run, and how to conduct them. There are a significant number of variables and parameters, and many of these can take hundreds of different values. This meant that in each experiment the independent variables, and the values they took, needed to be carefully chosen to keep the total number of simulations down to a manageable level. Additionally, the design space for agent implementations is vast, hence the project focused on some of the more novel or less researched coordination mechanisms. If time had allowed, more complicated experiments would have been conducted, and more elaborate agent strategies implemented. Many of the possibilities are outlined in section 9.2.

9.2 Future Work

The scope for extensions to this project is limited only by one's imagination. There are improvements that can be made to the simulator, and there is a significant experiment space that is still to be explored. This section will outline both types of extensions in detail.

9.2.1 Simulator Improvements

The simulator was designed to be as easy and intuitive to use as possible, while providing an expansive experiment space. There are some quality of life improvements, of differing levels of complexity, that would improve the user experience:

- Simulation speed slider - Currently there are constants in the code that specify how many frames it should take for agents to move, how long clusters should be displayed for, and how long results are displayed for. If the user wishes to speed up a simulation, they must change these constants in the code. A better solution would be to add a slider which the user can move to change the simulation speed. This would allow, for example, the user to watch the start of a simulation at a slower speed to understand agent behaviour, then speed the simulation up so that it concludes quicker. The GUI 4 Processing library provides a slider implementation that could easily be integrated into the simulator.
- Control agent implementation in GUI - Currently the user can change the agent implementation by editing the class used in the code. A drop-down menu, similar to that used to choose the focal point game, could be added to the simulator, allowing the user to switch between agent implementations without the need to change any code.
- Simultaneous different agent implementations - Currently the simulator only supports using one agent implementation at the time. The code could be changed to allow multiple implementations, although this would require some thought as to the best way to update the GUI, since the user would need to be able to specify how many of each type of agent they would like. Additionally, it would need to be possible to place specific agent types on the grid during the configuration phase.
- Visualise agent choices - Currently the user cannot tell what choice an agent has made without looking into the logs after the simulation. Writing the number an agent has chosen inside of the agent's circle is a potential solution, but with larger grid sizes the agents are quite small, so an alternative may be better.
- Increased number of graphs and statistics - The four graphs the simulator cur-

rently produces convey the most important information, but there are certainly more graphs that could be added. For example, a graph that shows the highest percentage of agents that chose an option without fully coordinating would indicate whether or not the agents are starting to reach some level of coordination, and this wouldn't be apparent from the win rate graph.

- Add or remove agents mid simulation - Once a simulation has begun, the agents remain unchanged. The ability to add or remove agents while a simulation is running would open up the possibility of many more experiments. One could investigate the effect of swapping out half of the agents once coordination has been established, for example.

9.2.2 Future Experiments

Whilst there are various improvements that could be made to the simulator, most future work would focus on running different experiments on the simulator, testing various theories and strategies. The following are just a few examples of the sorts of experiments that would be interesting to try out.

- Some of the theories mentioned in the background research, such as the level-n theory and cognitive hierarchy theory could be implemented in some way to evaluate their correctness.
- The experiments conducted with social networks could be furthered in various ways, for example using initialised social networks rather than ones that are dynamically built. Social networks could also be leveraged in different ways - for example, agents could have opinions of each other (based on their success, perhaps) and use this opinion to weight any information received from that agent. Agents could also choose which agents to include in and exclude from their social network - do cliques form, and are the agents in cliques better at coordinating?
- Monuments were used in various experiments, but only with one agent implementation at a time. It would be interesting to expand these experiments to multiple agent implementations. For example, it was shown that movement was important for communication, so 'explorer' agents could travel around the grid, telling agents about a monument. 'Monks' could stay near to monuments, potentially helping other agents to understand what the symbols in the monument mean. This builds on the idea that for coordination common knowledge is not sufficient - all agents need the same information and need to interpret that information (i.e. act) in the same way. If one agent sees 'P' and thinks it refers to prime numbers, while another agent believes it means powers of two, the agents are unlikely to coordinate.

- Various different subgames could be tested, not just the consensus game. The majority game is already built into the simulator. Another interesting possibility is the minority game, whereby agents are successful if they are among the minority when choosing. For example, a person may be deciding when to visit a theme park - they will want to be in the minority so that the queues for the rides are short.
- The idea of districts could be explored much further, and the presence of districts also opens up the idea of other types of games. For example, the simulator could be changed to require districts to choose differently, with a monument in each district signalling intentions, similar to the water temples in Bali [42].
- Seasonal variation could be introduced to experiments, whereby the conditions change periodically. This may encourage nomadic migration among agents, similar to pastoralists such as the Turks and Mongols [43, 44].
- It is also possible to delegate the thought process of agents to Prolog using Projog, a Prolog interpreter for Java [45]. This would allow for more complicated agent reasoning, and would make the simulator more accessible for users proficient in Prolog, and less well versed in Java/Processing.
- The mechanisms explored, as well as some of the ideas mentioned here, could be combined to see which are constructive and deconstructive. For example, what would happen if the Social Agents were adapted to view monuments too - would the abundance of information be a help or a hindrance? This also opens up the possibility of some form of basic reinforcement learning - do the agents learn to solely use the monuments or their social networks?

Chapter 10

User Guide

10.1 Installing Processing

Processing can be installed on Windows, macOS, and Linux. The download includes the Processing Development Environment (PDE), a simple text editor that allows you to run Processing ‘sketches’. The download can be found here, and additional installation instructions are located here.

10.2 Cloning the Repository

The code for the simulator is located here. The repository can either be forked or cloned.

10.3 Running the Simulator

Locate the repository from within the PDE, or open the file fyp.pde with the PDE.

The fyp.pde file contains various setup information. If you are using multiple monitors, the monitor which you wish to see the simulation on can be specified by changing the second argument in the call to the `fullScreen` function, near the top of the `setup` function. Please note that the simulator is designed to run at 1080p resolution.

Press the play button in the top left to start the simulator. The square button can be used to stop the simulation, or alternatively you can use the escape key.

10.4 Configuring an Experiment

Various variables need to be set before a simulation runs:

- Agents - this is the number of agents that will appear on the grid. Min value: 1. Max value: 100.
- Grid Size - this specifies the number of rows and columns. Min value: 3. Max value: 20.
- Rounds - this specifies the number of times the agents will play the focal point games. Min value: 1. Max value: 1000.
- Clusters - this is the number of groups of agents that will play the focal point games. Min value: 1. Max value: 6.
- Choices - in each focal point game, this is the number of options presented to each agent. Min value: 1. Max value: 1000.
- Moves - this is the number of times agents can move between each focal point game. Min value: 1. Max value: 20.
- Occupancy - the maximum number of agents that can be in a grid cell at any given time. Min value: 1. Max value: 4.
- Monuments - the number of monuments that will appear on the grid. Min value: 0. Max value: 50.
- Visibility - the maximum distance from which a monument can be seen by an agent. Min value: 0. Max value: 20.
- Subgame - specifies which win condition to use for the focal point games. 'CONSENSUS' requires all agents in a cluster to choose the same option. 'MAJORITY' requires more than half the agents to choose the same option.
- Localised monuments - if there are both districts and clusters, when this option is selected monuments are only visible from the districts they are located in.
- District clusters - if there are districts, when this is selected agents are clustered by district rather than using the k-means clustering algorithm.

Any violations of the maximum and minimum values will be visualised by turning the text box orange.

In addition to the minimum and maximum values specified above, there are some additional restrictions on some of the variables:

- The maximum number of agents is bounded by the number of cells. E.g. if using an 8x8 grid, the maximum number of agents is 64.

- The maximum number of monuments is bounded by half the number of cells. E.g. if using an 8x8 grid, the maximum number of monuments is 32.
- The minimum agent occupancy is bounded by: ceil(2 x agents / cells). E.g. if using 100 agents on a 10x10 grid, the minimum occupancy is 2.
- The number of clusters is bounded by: ceil(agents / occupancy). E.g. if using 10 agents with an occupancy of 4, the maximum number of clusters is 3.
- The maximum monument visibility is bounded by: (grid size - 1) x sqrt(2). E.g. if using an 8x8 grid, the maximum visibility is 9.90 (to 2 d.p.).

These requirements are automatically rectified if there are any issues (when the configure button is pressed), with the values updated in the text boxes. You can always click the 'Reset' button to go back and change any of the values.

The next stage of the configuration process is the placing stage. This is where agents and monuments can be placed, and districts created. By default, this stage begins with creating districts. Districts are rectangular areas of the grid that agents can use to inform their thinking. To create a district, click on the grid in a square that will be one of the corners of the district. Drag the mouse and release in the diagonally opposite corner of the district. Districts cannot overlap, and the whole grid must be covered if at least one district exists.

To change what is being placed, use the button just above the top left of the grid. Agents and monuments are placed by clicking in squares. Occupancy limits apply - there can only be one monument per cell, while the agent occupancy is whatever was set in the previous step. Any unplaced monuments or agents will be randomly created by the simulator.

When you are happy with the placements, click the 'Start' button to begin the simulation. If there are any errors, such as district clusters being selected but no districts created, these will be communicated to you via the panel on the right. All error messages will appear in red.

10.5 During / After the Simulation

At any point during a simulation, execution can be started or stopped using the 'Play'/'Pause' button. As the simulation progresses, the graphs on the right-hand panel will update with statistics in real time. Simultaneously, the server will log information about the simulation to a JSON file. Logs can be found in a 'logs' directory, which should be located in the same place as the code. If the simulator cannot find or create this directory, you may need to create it yourself. Each log file is named with the date and time at which the simulation began.

10.6 Custom Code

Various parts of the simulator have been designed to make it as easy as possible for you to add your own code. There are three ways this is envisaged:

- Custom agent implementations - To create your own implementation, create a new class which extends the Agent class. The only method that must be overridden is the clone method. This method must return an agent whose type matches your new class. For example, if you create the SneakyAgent class, this method must return a SneakyAgent. All other non-final methods can optionally be overridden. Once the class has been created, the agent can be instantiated in the `tryAddAgent` method in server.pde. For examples of other agent implementations, see all files with 'Agent' in the name.
- Custom monument implementations - The process is almost identical to the process for creating agents, except that the new class must extend the Monument class. The `clone` method must be overridden in the same way, and the `chooseNextGridPosition` method can optionally be overridden. The new monument should be instantiated in the `tryAddMonument` method in server.pde. For an example implementation, see circlingMonument.pde.
- Custom focal point games - Please follow the instructions in subgameHandler.pde to add a new subgame.

Bibliography

[1] Thomas Crombie Schelling. *The Strategy of Conflict*. Cambridge, Massachusetts: Harvard University Press, 1980.

[2] John B. Van Huyck, Raymond C. Battalio, and Richard O. Beil. "Tacit Coordination Games, Strategic Uncertainty, and Coordination Failure". In: *The American Economic Review* 80.1 (1990), pp. 234–248.

[3] Viv Burr and Penny Dick. *Social constructionism*. Springer, 2017.

[4] Paul A Boghossian. "What is social construction?" In: (2001).

[5] Zili Yang. ""Lucky" numbers, unlucky consumers". In: *The Journal of Socio-Economics* 40.5 (2011), pp. 692–699.

[6] John Maynard Keynes. *The General Theory of Employment, Interest, and Money*. London: Macmillan and Co, 1936.

[7] Philip D Straffin. *Game Theory and Strategy*. Washington, DC: The Mathematical Association of America, 1993.

[8] David M. Kreps. "Nash Equilibrium". In: *Game Theory*. London: Palgrave Macmillan UK, 1989, pp. 167–177.

[9] John Nash. "Non-Cooperative Games". In: *The Annals of Mathematics* 54.2 (1951), pp. 286–295.

[10] R. Preston McAfee. *Introduction to Economic Analysis*. Washington, DC: Saylor Foundation, 2009.

[11] OECD. *Glossary of Industrial Organisation Economics and Competition Law*. `https://www.oecd.org/regreform/sectors/2376087.pdf`. Accessed: 2023-01-18. 1993.

[12] David P Roberts. "Pure Nash equilibria of coordination matrix games". In: *Economics Letters* 89.1 (2005), pp. 7–11.

[13] Nicholas Bardsley, Judith Mehta, and Robert Sugden. "Explaining Focal Points: Cognitive Hierarchy Theory vs Team Reasoning". In: *The Quarterly Journal of Economics* 120.543 (Mar. 2010), pp. 40–79.

[14] John C. Harsanyi and Reinhard Selten. *A general theory of equilibrium selection in games*. Cambridge, Massachusetts: MIT Press, 1992.

[15] David Lewis. *Convention: A philosophical study*. New York, New York: John Wiley & Sons, 1969.

[16] Judith Mehta, Chris Starmer, and Robert Sugden. “The Nature of Salience: An Experimental Investigation of Pure Coordination Games”. In: *The American Economic Review* 84.3 (1994), pp. 658–673.

[17] Dale O. Stahl and Paul W. Wilson. “On Players Models of Other Players: Theory and Experimental Evidence”. In: *Games and Economic Behavior* 10.1 (1995), pp. 218–254.

[18] Colin F. Camerer, Teck-Hua Ho, and Juin-Kuan Chong. “A Cognitive Hierarchy Model of Games”. In: *The Quarterly Journal of Economics* 119.3 (Aug. 2004), pp. 861–898.

[19] Michael Bacharach. “Interactive team reasoning: A contribution to the theory of co-operation”. In: *Research in Economics* 53.2 (1999), pp. 117–147.

[20] Robert Sugden. “The Logic of Team Reasoning”. In: *Philosophical Explorations* 6.3 (2003), pp. 165–181.

[21] Michael Bacharach. “Variable Universe Games”. In: *Frontiers of Game Theory*. Ed. by Ken Binmore, Alan Kirman, and Piero Tani. Cambridge, Massachusetts: MIT Press, 1993.

[22] Maarten C.W. Janssen. “Rationalizing Focal Points”. In: *Theory and Decision* 50.2 (2001), pp. 119–148.

[23] Vincent P Crawford and Hans Haller. “Learning how to cooperate: Optimal play in repeated coordination games”. In: *Econometrica: Journal of the Econometric Society* (1990), pp. 571–595.

[24] “The evolution of focal points”. In: *Games and Economic Behavior* 55.1 (2006), pp. 21–42.

[25] Josiah Ober. *Democracy and Knowledge: Innovation and Learning in Classical Athens*. Princeton, New Jersey: Princeton University Press, 2008.

[26] Michael Bacharach and Michele Bernasconi. “The Variable Frame Theory of Focal Points: An Experimental Study”. In: *Games and Economic Behavior* 19.1 (1997), pp. 1–45.

[27] “Equilibrium Selection in Experimental Games with Recommended Play”. In: *Games and Economic Behavior* 11.1 (1995), pp. 36–63.

[28] Greg MP O’Hare and Nicholas R Jennings. *Foundations of distributed artificial intelligence*. Vol. 9. John Wiley & Sons, 1996.

[29] H Van Dyke Parunak. “Applications of distributed artificial intelligence in industry”. In: *Foundations of distributed artificial intelligence* 2.1 (1996), p. 18.

[30] Sarit Kraus, Jeffrey S Rosenschein, and Maier Fenster. “Exploiting focal points among alternative solutions: Two approaches”. In: *Annals of Mathematics and Artificial Intelligence* 28 (2000), pp. 187–258.

[31] Nick R Jennings. “Coordination techniques for distributed artificial intelligence”. In: (1996).

[32] Dimitrios Koutsonikolas, Saumitra M Das, and Y Charlie Hu. “Path planning of mobile landmarks for localization in wireless sensor networks”. In: *Computer Communications* 30.13 (2007), pp. 2577–2592.

[33] React.js. *React*. `https://reactjs.org`. Accessed: 2023-01-31.

[34] Qt Group. *The Framework*. `https://www.qt.io/product/framework`. Accessed: 2023-01-31.

[35] Processing Foundation. *Welcome to Processing!* `https://processing.org/`. Accessed: 2023-01-31.

[36] Google. *Google Java Style Guide*. `https://google.github.io/styleguide/javaguide.html`. Accessed: 2023-01-31.

[37] Peter Laga. *G4P (GUI for processing)*. `http://www.lagers.org.uk/g4p/`. Accessed: 2023-02-01.

[38] giCentre. *giCentre Utilities*. `https://www.gicentre.net/utils`. Accessed: 2023-06-09.

[39] John A Hartigan and Manchek A Wong. “Algorithm AS 136: A k-means clustering algorithm”. In: *Journal of the royal statistical society. series c (applied statistics)* 28.1 (1979), pp. 100–108.

[40] Middle East Technical University. *K-Means Empty Cluster Example*. `https://user.ceng.metu.edu.tr/~tcan/ceng465_f1314/Schedule/KMeansEmpty.html`. Accessed: 2023-05-23.

[41] Malay Pakhira. “A Modified k-means Algorithm to Avoid Empty Clusters”. In: *International Journal of Recent Trends in Engineering* 1 (Jan. 2009).

[42] J Stephen Lansing and James N Kremer. “Emergent properties of Balinese water temple networks: Coadaptation on a rugged fitness landscape”. In: *American Anthropologist* 95.1 (1993), pp. 97–114.

[43] Rudi Paul Lindner. “What was a nomadic tribe?” In: *Comparative studies in Society and History* 24.4 (1982), pp. 689–711.

[44] Rada Dyson-Hudson and Neville Dyson-Hudson. “Nomadic pastoralism”. In: *Annual review of anthropology* 9.1 (1980), pp. 15–61.

[45] Webber, S. *Projog*. `http://projog.org/`. Accessed: 2023-06-16.

Appendix A

Code

The source code for the simulator can be found at:

https://github.com/thl19git/FocalPointGameSimulator

Appendix B

Monument Locations

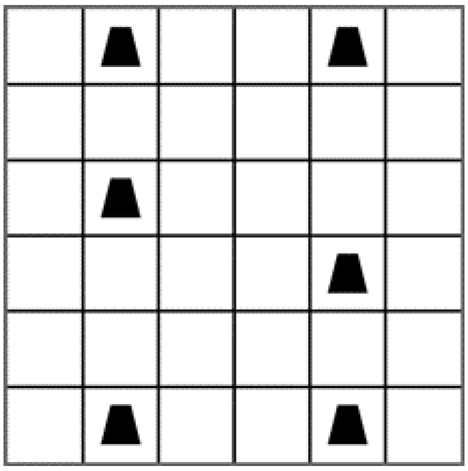


Figure B.1: Monument locations, $G = 6$, $L = 6$

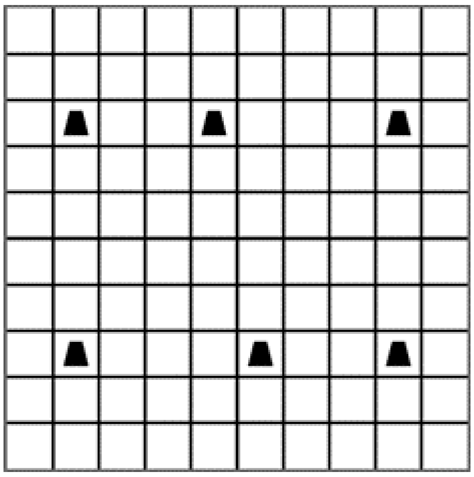


Figure B.2: Monument locations, $G = 10$, $L = 6$

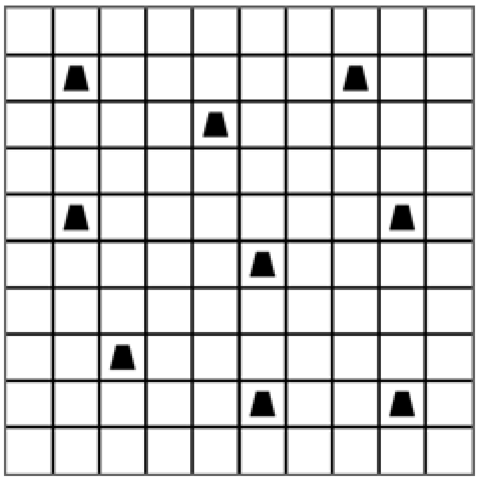


Figure B.3: Monument locations, $G = 10$, $L = 9$

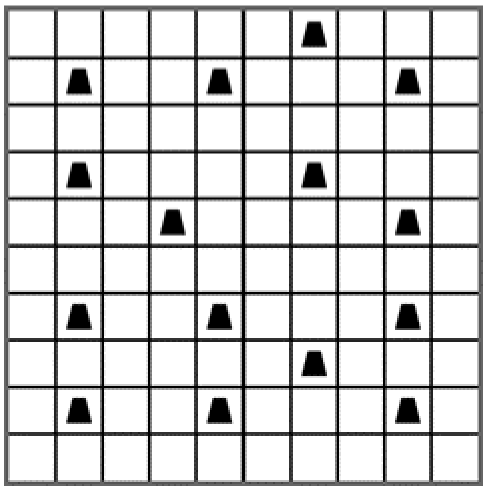


Figure B.4: Monument locations, $G = 10$, $L = 15$