KING'S College LONDON

University of London

Modelling Civil Unrest: Competition Between Cooperative Agent Groups

Final Project Report

Author: Jake Butter

Supervisor: Dr Sanjay Modgil

Student ID: 1223862

April 27, 2015

Abstract

This project aims to create an agent-based model of two opposing groups of people during a hypothetical civil unrest scenario. More specifically, the simulation will model utility-motivated spatial movement of two opposing agent types, where one group (representing the party in a state of unrest) attempts to move towards one or more points and the other group (representing, for example, law enforcement) attempts to impede this movement. The model aims to simulate human conflict in a compelling way using agents that possess the ability to both learn and share knowledge with each other.

The goal of this project is to create a functional piece of software that could be used by organisations interested in studying crowd-based conflicts such as sociology researchers, law enforcement agencies or architects for defensive structures. The created software will also be used to carry out some sample experiments, exploring the effects of various agent modelling techniques and demonstrating how the model could be used in practice.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Jake Butter

April 27, 2015

Acknowledgements

I would like to thank my advisor, Dr. Sanjay Modgil, for offering his support, guidance and expertise throughout the creation of this project, and for always responding to my ideas with enthusiasm.

I would also like to thank Dr Samhar Mahmoud who, although not officially my project advisor, took a significant amount of time out of his schedule to provide advice and specialist knowledge regarding agent-based systems.

Contents

Chapter 1

Introduction

An **agent-based model** is a computational model that simulates the behaviour of autonomous agents. They are employed to simulate a wide variety of different systems, with the primary aim being to draw conclusions about the effects of individual behaviours on the system as a whole.

A **multi-agent system** is a computational system comprised of multiple intelligent agents which interact with each other in a virtual environment. Such systems are often used to solve specific real-world problems that would be unfeasible using a single agent.

ABMs and MASs[1] are closely related but are not synonymous. ABMs typically model simpler, not necessarily intelligent, agents with the focus being primarily on the macro-scale conclusions that can be drawn from their behaviour. MASs are most often tasked with finding a *solution* to a specific *problem* using agents which may have the ability to make more intelligent decisions or learn from their experiences. A piece of software simulating agents can be an ABM, an MAS or both.

1.1 Project aims

The primary objective of this project is to create a compelling, customisable simulation of human behaviour in a civil unrest scenario. To do this, two distinct agent types need to be modelled that possess the ability to interact with other agents of both the same and opposing type.

This project aims to combine the macro-scale experimentation applications of a typical agent-based model with the greater agent complexity and intelligence of a multi-agent system.

[1] Used hereafter to refer to 'agent-based models' and 'multi-agent systems' respectively

As an MAS, the simulation will utilise agents that have the ability to learn from their experiences and communicate this learned knowledge to other agents. As an ABM, the simulation will use this agent behaviour to create a model reminiscent of real-world scenarios, from which useful experimental data can be extracted.

The secondary aim of the project is to use the final software product to perform some experiments, demonstrating the functionality of the model. These experiments can be split into two categories. The first consists of those which deal with model-specific variables, such as those used by learning algorithms, which can be used to draw conclusions about the model itself and the various techniques it uses for simulating agent behaviour. The second category consists of those experiments that are based on real-world variables, such as the numbers of different agent types or the shape of their environment, which can be used to draw conclusions about real-world scenarios.

1.2 Report Structure

This report will begin with a background overview of the various topics relevant to the project, as well as a review of existing works that have similar technical functionality or that simulate similar real world scenarios. Following this are the requirements, specification and design of the project software.

Next, the report will follow the implementation of the software in a chronological fashion, using the distinct software versions as milestones. This chapter will address any problems faced during development and the solutions employed in response.This chapter will also describe the ways in which the fully featured software was optimised to improve performance when using large numbers of agents.

The next major chapter will cover the experimentation phase of the project, presenting data collected using the software and making some conclusions about any trends that arise. This chapter will be split into two sections: model-specific experiments and real-world experiments. Following this will be a brief chapter addressing the professional and ethical considerations that were considered in the development of this project.

Subsequently, the outcomes of software development and experimentation will be evaluated and the strengths and weaknesses of the project as a whole will be ascertained. Finally, a conclusion chapter will summarise what has been learnt throughout the project and outline possible future continuations of the work conducted so far.

Chapter 2

Background

2.1 Civil unrest

Throughout history, civil unrest has played a key role in the development of a wide variety of different societies and cultures. The fact that such unrest continues to occur in the present day, all over the world, is testament to its importance [14]. Much study has been done on the subject, aiming to further our understanding of how and why civil unrest occurs and why participants act in the way they do. The practical application of such research is obvious: the established order, the 'target' of unrest, suffers for as long as rioting or demonstration continue and those in power must stop it as quickly as possible if they wish to maintain this order [8]. The more knowledge that those in power possess about the psychology behind unrest, the better they should be able to contain it.

The act of participation in certain forms of unrest can be seen as a departure from 'civilised' existence and an opportunity for people to act without an explicit structure or system of control [5]. It is this fact that makes unrest especially interesting when studying the psychology of people and how they decide to act and behave in various situations. In abstraction, we can consider civil unrest to be a state of 'raw' human psychology, wherein people act in consideration of simple goals without being mindful of more complex concerns (e.g., the distant future and past, self-image, intellectualism). If we view unrest in such a simple way, it becomes clear that it is particularly well suited for representation by a computational model. The additional consideration of how humans act within a crowd also translates well to realisation in a multi-agent system. This is what this project aims to implement: a multi-agent simulation of crowds in a state of unrest and efforts to control such crowds. This project does not address the

issues of justification, morality or politics relating to either the act of participating in unrest or attempting to control unrest.

2.2 Agent-based modelling

Using agent-based modelling (ABM) to simulate crowd behaviour is not a new idea. Perhaps the earliest example is that of Boids, a model used to simulate the movement of flocking birds [16]. Even earlier examples such as Conway's Game of Life and other cellular automata, while not agent-based models themselves, can be seen as simulations of crowd behaviour in that they model the actions of pseudo-agents (the cells) that perceive others in their environment (immediately surrounding cells) and use this perception to affect their behaviour [10].

Since its introduction, agent-based social simulation has flourished and there now exists a wealth of different implementations of crowd simulation in a wide variety of different scenarios and with different practical aims [9]. Part of the reason for this, as well as a result of this, is that this sort of modelling is increasingly being seen as having potential as a tool for studying and simulating real world phenomena. The existence of crowds is, by nature, resource intensive. Deliberately establishing a crowd is expensive, time-consuming and may alter the particular behaviour that is intended for study and observing 'natural' crowds in research-relevant contexts is often dangerous (e.g., during natural disasters, combat or unrest). Conversely, computational models can be used (after an initial cost of design and implementation) relatively cheaply and in a much more flexible way. Simulations can be run at any time, with highly variable parameters and enormous numbers of agents, and can also be repeated incredibly quickly, a large number of times. One particularly interesting example of crowd simulation being used to mitigate cost is the employment of agent-based models in filmmaking. Rather than hire vast quantities of background actors, some film studios have opted to use crowd simulation technologies to create CG crowds that behave in a realistic manner [17].

However, agent-based modelling of crowds is still a relatively young field. Examples of the practical usage of ABM in real world development and decision-making are rare and many criticise ABM as being not yet developed enough to simulate complex phenomena, particularly human behaviour [15]. It can be argued that computational modelling is intrinsically limited by being artificial. A model cannot output or demonstrate anything that is not a logical consequence of the information provided to it by a human designer and any such output would be subject to the rules and biases imposed on it by said designer. It is in this light that we must carefully consider the way in which ABMs are useful to us in practical applications.

In creating a model of a specific scenario (civil unrest, in this case) we should not intend to create a perfect replica of a real world phenomenon, as this is obviously impossible. Instead, the model should act as a tool with which we can manipulate, evolve and contextualise knowledge that we already possess, enabling us to find behaviours and trends that we otherwise would be unaware of. The utility of a multi-agent system comes not from the complexity of individual agents, but from the complexity that develops from the actions and interactions of a large number of such agents. One key way to encourage the development of new data and behaviours from an MAS is to enable the agents within it to learn.

2.2.1 Existing computational models of civil unrest and crowds

Modeling civil violence: An agent-based computational approach

[6] This relatively simple ABM aims to model the dynamics of revolution within human populations in two distinct scenarios. It deals with high level concepts, using simple variables to represent broad, abstract concepts such as "grievance", "hardship" and "legitimacy". While lacking the detailed, spatial aspect of unrest that this project aims to represent, it does quite elegantly model the interaction between two opposing 'teams', the "agents" and the "cops" in the first model and two distinct ethnic groups in the second. The simplicity of agent behaviour and the structure in which they are organised (closely resembling a cellular automaton) mean that this model is well suited to making macro-scale conclusions about population groups without being particularly concerned with the experiences of individual agents, as is typical of ABMs used in social science applications.

Modeling Protest in Lafayette Square: An agent-based modeling approach

[2] This model uses a similar level of abstraction to the previous example, again employing two teams of agents and a "grievance" variable. However, this model has a much greater focus on the spatial aspect of unrest, focusing on agent movement within an urban environment designed to represent a real location. The end result of this allows the analysis of where protesting agents are likely to concentrate or become most 'aggrieved'. While the use of geographical data certainly lends this model a greater degree of 'realism', the agents within the environment still lack any great depth of behaviour, particularly the ability to learn.

Agent-based Simulation of the 2011 Great East Japan Earthquake/Tsunami Evacuation: An Integrated Model of Tsunami Inundation and Evacuation

[12] This model aims to simulate crowds in emergency scenarios, modelling a large scale evacuation from a coastal urban area during a tsunami. This paper presents a particularly interesting usage of agent-based crowd modelling by simulating micro scale agent movement through physical space and then using data collected from the simulation to model the movement of large numbers of agents in macro scale. This demonstrates the kind of application where this sort of model has most value, revealing trends and general behaviours which can be used in other contexts without being overly focused on minutiae (which are unlikely to be accurate or valuable when modelling complex scenarios).

2.3 Machine learning

The field of machine learning is an enormous, multi-disciplinary area of research that is far too broad to summarise briefly. Instead this section will focus exclusively on a particularly relevant form of machine learning, namely reinforcement learning.

Reinforcement learning relates specifically to the decision-making process of agents that exist in an environment wherein their actions have certain rewards associated with them (a Markov decision process) [7]. These rewards enable agents to make value judgements about their actions by providing a framework for ascertaining utility. In a general sense, agents aim to take actions which offer the greatest reward. The matter of learning arises when agents start forming their own expectations of value, based on previous experiences, e.g., an agent traversing a maze took a left turn at a certain point and found that it led to an exit (the action of turning left had a high reward), upon reaching this point again the agent expected turning left to be useful, based on its previous experience.

There are three variable factors to consider when utilising reinforcement learning [18]. The first is the issue of long-term reward versus short-term reward (known as a discount factor). In some reinforcement learning implementations, agents can judge actions not just on their expected value, but on the expected value of later actions following them. Altering how an agent uses this information can have significant effects on the overall simulation.

The second factor is the balance of exploration versus exploitation. Exploration describes agents taking actions that do not necessarily have the highest expected reward of their available actions. Exploitation describes the inverse, agents taking the 'safe bet'. Exploring offers agents

the chance to discover higher value actions than the ones they already have knowledge of by 'covering new ground'. In the maze example, an agent that does not explore would, once it had found a successful path to the exit, follow the same path over and over again forever. An exploring agent on the other hand would occasionally take what it initially perceived to be 'wrong turns' (low value actions) but may also stumble upon a shorter path to the exit. In implementation, exploration versus exploitation is based on how an agent decides which action to take. An agent that only ever chooses the action with the highest expected utility would never explore and an agent that chooses actions completely at random would be maximally explorative (but largely useless). A middle ground solution is for agents to use the expected utility of actions to generate a probability distribution, meaning that they still act 'logically', choosing high value actions most of the time, but sometimes explore lower value possibilities. In some cases, the preference of an agent to explore or exploit can be altered dynamically, mid-simulation, to better approximate reasonable behaviour, i.e., an agent that is 'doing well' in its environment will be less likely to explore than one that is 'doing poorly' as it has less reason to want to explore alternative actions.

Finally, implementations of reinforcement learning can employ a 'learning rate'. This is a factor that is used to determine how greatly new information affects the knowledge held by an agent. A minimal learning rate (a value of 0) would result in new information having no effect on an agent's knowledge and as such, the agent wouldn't learn anything. A maximal value (1) would mean that new information would entirely overwrite old knowledge, resulting in an agent that only ever considers the most recent information regarding a certain action. This is an optimal learning rate for deterministic environments as an agent has no reason to consider past occurrences or trends when the result of taking an action never changes. In other environments a learning rate between the two extremes is typically used, which means agents learn in a more gradual way. As with exploration, a variable learning rate can also be used. The WoLF (Win or Learn Fast) algorithm proposed by Bowling and Velosos [3] has an agent adopt a low learning rate when it is succeeding and a higher rate when it is failing. This was shown to result in significantly better performance from agents in various scenarios.

The use of reinforcement learning by agents navigating a spatial environment represents the core of the functionality that this project aims to implement and build upon. This kind of basic functionality can be seen in existing implementations, such as Chen's *Grid World*[4].

2.4 Agent cooperation

In simpler agent-based models, agents act in an entirely independent fashion. There may exist many identical agents within an environment, but most of the time these agents will simply perceive each other as being part of this environment, rather than acknowledging their shared goals or behaviours. By introducing an element of cooperation, a system wherein agents aim to solve a problem may benefit from faster convergence on a solution by allowing agents to take in more information than if they acted independently[11]. In models designed to mimic real-world phenomena, cooperation may help to make the model much more realistic, especially in examples where agents represent people or groups of people.

Implementation of cooperation in multi-agent systems is typically closely tied to the way in which agents learn, most commonly using reinforcement learning algorithms. The core concept is that of *sharing*, however there is some flexibility with regards to what exactly is shared between agents. An agent could communicate simple information about the environment to others, leaving the responsibility of making any sort of judgement using that information to those receiving it. Alternatively, an agent could make a utility judgement about a certain action and communicate the result of that judgement to others, ignoring the possibility that other agents may have judged the information differently. In a more extreme example of cooperation, agents could store their knowledge in a completely shared structure, meaning that all agents would use the exact same knowledge in their decision making, forming what might be referred to as a 'hivemind'[19].

2.4.1 Existing implementations of reinforcement learning with cooperation

Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents

[19] The focus of this paper is to study the effects of various different types of inter-agent cooperation on the effectiveness of agents in a given task (in this case, hunting prey). It firstly grants agents the ability to share "sensation", simple information about their environment, and demonstrates significant improvements in agent effectiveness with both dedicated "scout" agents that do not hunt, and hunters who have the secondary ability to 'scout' themselves.

The second sharing technique allows agents to share their *policies* (how they decide upon their actions), either by having all agents follow a mutually updated central policy (as in the 'hivemind' example given above), or by exchanging and averaging policies at certain intervals.

Agents were even given the ability to share the details of entire episodes (an episode represents a full 'hunt' in this case). Again, all of these options were shown to offer an improvement over completely independent agents.

A study of reinforcement learning with knowledge sharing

[11] This paper is particularly interesting in that it demonstrates the practical value of cooperative reinforcement learning by applying it to a real world scenario: that of moving robots. The key problem addressed is the very slow rate at which a real robot learns due to being limited by its movement speed. A solution is proposed that allows robots to learn from the real world, as well as constructing an "imaginary" world internally wherein they can play out different scenarios. The key to building a useful internal representation of the world is the sharing of experiences between agents, which is shown to improve robot performance even when the robots have distinct tasks.

Chapter 3

Requirements and Specification

3.1 Brief

The primary goal of this project is to create a piece of software that will enable an end-user without significant IT knowledge to run agent simulations mimicking real-world civil unrest scenarios. The software must allow for a high degree of customisability and must be able to output data in a way that is useful for research applications.

3.2 Requirements

The software can be split into four parts: The front end that the user will see, the back-end model environment and the two back-end agent types. These parts each have their own unique requirements.

3.2.1 Front End Requirements

- The user must be able to initialise, run and stop the model.
- The user must be able to configure a suitable number of model variables to customise the model as required.
- The front end must be able to display a clear graphical representation of the model.
- The front end must be able to output data in a suitable format for research use, including the ability to perform batch runs.

3.2.2 Environment Requirements

- The environment must represent a real-world physical space with a suitable degree of accuracy and precision.
- The environment must support the definition of different areas with different properties: Open space, blocked areas, dissenter spawn points, enforcer patrol areas and dissenter targets.
- The user should be able to customise and create environments with relative ease.
- The environment should be structured in such a way that agents are able to move realistically but also able to make space-dependent calculations efficiently.

3.2.3 Agent Requirements

The model must include two distinct agent types representing the two opposing groups involved in a civil unrest scenario. These two groups will hereafter be referred to as the *dissenters* (those in a state of unrest) and the *enforcers* (those attempting to control the dissenters).

Mutual Requirements

- Agents must move around in the environment in a suitably realistic way.
- Agents must observe their environment and react to changes in it.
- Agents must choose their actions based on individually held knowledge.
- Agents must learn from the outcomes of their actions and alter their knowledge.
- Agents must share their knowledge with others of the same type.
- Agents must be able to operate under a variety of different configurations to allow for varied experiments to be performed.

Dissenters

- Dissenters must be able to move towards a predefined point, or points.
- Dissenters must be able to attempt to subvert efforts made by the enforcers to stop them.
- Dissenters must assess their performance to decide whether they are performing 'successfully' or 'unsuccessfully'.

- Dissenters must alter their behaviour depending on how successful they consider themselves to be.

Enforcers

- Enforcers must move within a predefined 'patrol area'.
- Enforcers must attempt to stop dissenters from reaching their target and repel them if they reach it.
- Enforcers must ascertain the success of the dissenters and alter their behaviour accordingly.
- Enforcers must possess the ability to behave in a more organised way than the dissenters, mimicking rudimentary 'tactics'.

3.3 Specification

3.3.1 Development Environments

There exist many development environments specifically designed for creating agent-based models. These packages typically streamline the process of implementing common model functionality and often come with their own front end solutions. Because this project focuses primarily on the design of the agents and the results the model produces, rather than the details of exactly *how* the basic functionality is implemented, the project software will be created using one of these packages.

After consideration of the project's requirements the package *Repast Simphony* was chosen. This Java-based solution features a useful front end framework, comprehensive documentation and support for all necessary functionality. While this package takes care of much of the work involved in establishing the model's overall structure, it still allows for the creation of entirely bespoke agent behaviour.

3.3.2 Front End Specification

The program front end will use Repast Simphony's included, customisable solution. This application features basic simulation controls and allows for the creation of user-editable parameters to alter model variables. It also allows for the creation of graphical model representations,

graphs and output data streams using wizards, meaning that users with more particular requirements can extend the functionality of the front-end themselves. Finally, the Repast front-end features native support for batch runs on single or multiple machines.

Although the front end will allow for user customisation, a default setup will be created with support for common functionality containing:

- A graphical representation, showing both agent types and the model environment
- A file output data stream, recording pertinent information to a file on the local machine
- A collection of predefined user-editable parameters

3.3.3 Environment Specification

When modelling the movement of humans in crowds, the concept of 'height' in physical space is not particularly important, as human beings are not typically able to independently travel upwards towards the sky in any meaningful way. Therefore, the model environment will be configured to represent a 2-dimensional birds eye view of physical space.

The most important factor to consider when designing the environment for an agent-based model is the balance between accuracy and efficiency. The environment must provide enough detail to create a suitable virtual representation of the intended real-world space, while also allowing the model to run at a reasonable speed when hundreds of agents are interacting with it simultaneously.

To achieve this, the model environment will consist of *two* structures, both representing the same physical space. The first will be the structure that the end-user will see, a continuous space in which agents can exist at precise positions and move through realistically. The second will be a grid that is not represented visually. This grid will be the structure used by agents to perform all calculations, allowing them to efficiently approximate the state of the environment around them. Because the agents will not perceive the environment in any greater detail than it provides, the grid represents the maximal precision with which a physical space can be recreated. Each grid space will have one or more of the required environment properties (open space, blocked areas, etc.).

Finally, to allow the simple creation of new environments, the model will read the properties of its environment from a 'map file', a simple PNG image. Each pixel of this image will represent one grid space and its colour will determine that grid space's properties. By using this approach, end users will be able to define new environments using a vast array of applications. Almost

all deployment platforms will have access to a piece of software that can edit PNG files.

3.3.4 Agent Specification

Actions and Utility

To model agent movement in a relatively realistic, efficient manner, agents will be able to move in eight directions (the *cardinal* and *ordinal* directions on a compass) between grid cells. While the continuous environment structure will show agents moving gradually between points, the agent's actions will be based purely on the cell they currently occupy and the cells in the immediate neighbourhood that they can move to. To improve the realism of the model, agents will move to random fractional points within grid cells, as opposed to simply moving to their centre or origin.

In order for agents to learn, they must be able to ascertain the *utility* of an action that they have taken. To do this, there must be a *reward* associated with performing an action. This reward might be static and predetermined, like a dissenter agent moving into a goal cell and receiving a large reward, or dynamic, like an enforcer moving into a cell containing a number of dissenters and receiving a reward proportional to this number.

Learning

The basis of agent learning in the model will be the Q-learning reinforcement learning algorithm. The two variables it depends on, learning rate and discount factor, offer a great degree of customisability when performing experiments. When using Q-learning, an agent's *expected utility* for an action is stored as a 'Q-value'.

To supplement Q-learning, agents will also employ a variable learning rate mechanism, inspired by the WoLF (Win or Learn Fast) algorithm. When agents consider themselves to be performing successfully they will lower their learning rate and when performing unsuccessfully will raise it.

Finally, an agent's Q-values will decay over time, approaching zero. This decay means that 'old knowledge' that is less likely to be relevant to the current situation has less impact on an agent's decision making.

Sharing

As discussed earlier, there are different types of information that an agent can share to transfer information. In this model agents will transfer *Q-values*, as opposed to simple sensation. That

is to say that judgement of utility is carried out prior to information transfer. Although this sharing strategy could result in strange behaviour when used in a system where agents have diverse goals, agents in this model only share knowledge with others possessing the exact same goal meaning that behaviour should remain largely consistent. The key benefit of this strategy is efficiency, as a judgement of utility need only be carried out once when knowledge is transferred to many other agents.

As with other calculations, agents will make use of the grid structure to share knowledge only with those within a certain radius.

Choosing

Agents will use their held knowledge (in the form of Q-values) to assign probabilities to their choices of actions. Having agents choose actions based on probabilities means that they will not be fully exploitative and will sometimes choose actions that are not optimal (at least from the agent's perspective). This exploration is an important factor in modelling dynamic, unpredictable systems like crowds of people.

Other factors will also influence agent choice. For example, an enforcer agent will be more likely to move into a neighbouring cell containing a large number of dissenters than one that doesn't contain any. This is not necessarily a choice based on the expected utility of making such a move, but rather on the simple assumption that an enforcer agent is able to see dissenter agents that are directly in front of it.

Success

The goal of the dissenters is to minimise their displacement from a goal point. We can say that for a dissenter to consider themselves to be working successfully towards this goal they should either be moving towards a goal point, or have already reached one.

The goal of the enforcers is to maximise the dissenters' displacement from the goal point. As this is the opposite of the dissenters' goal, we can say that if the dissenters are successful, the enforcers are unsuccessful and vice versa.

Additional Enforcer Requirements

In addition to the basic agent behaviour, enforcers must also be able to repel dissenter agents away from the goal point. For the sake of simplicity, this will not be implemented as a specific action. It can be assumed that enforcers are actively 'repelling' at all times, affecting the

movement of dissenter agents in the same grid cell.

Enforcers are also expected to behave in a more organised way than the dissenters, in order to better mirror real-world scenarios involving organised groups such as law enforcement agencies. To achieve this, enforcer agents will be split into 'squads'. Enforcers will only share knowledge with members of their squad, enabling them to behave less as a single crowd. The enforcer team will also consist partially of 'scout' agents, a special subtype of enforcer. These scouts will not work towards the goal of the enforcers, instead traversing the environment randomly in order to collection information and share it with others. Finally, the enforcers will also have a single leader agent. This leader will collect information about the location of dissenter agents and will give the normal enforcer agents 'orders'. These orders will affect the way in which enforcer decide which action to take, making them prefer to move towards areas with a known high population of dissenters.

Chapter 4

Design

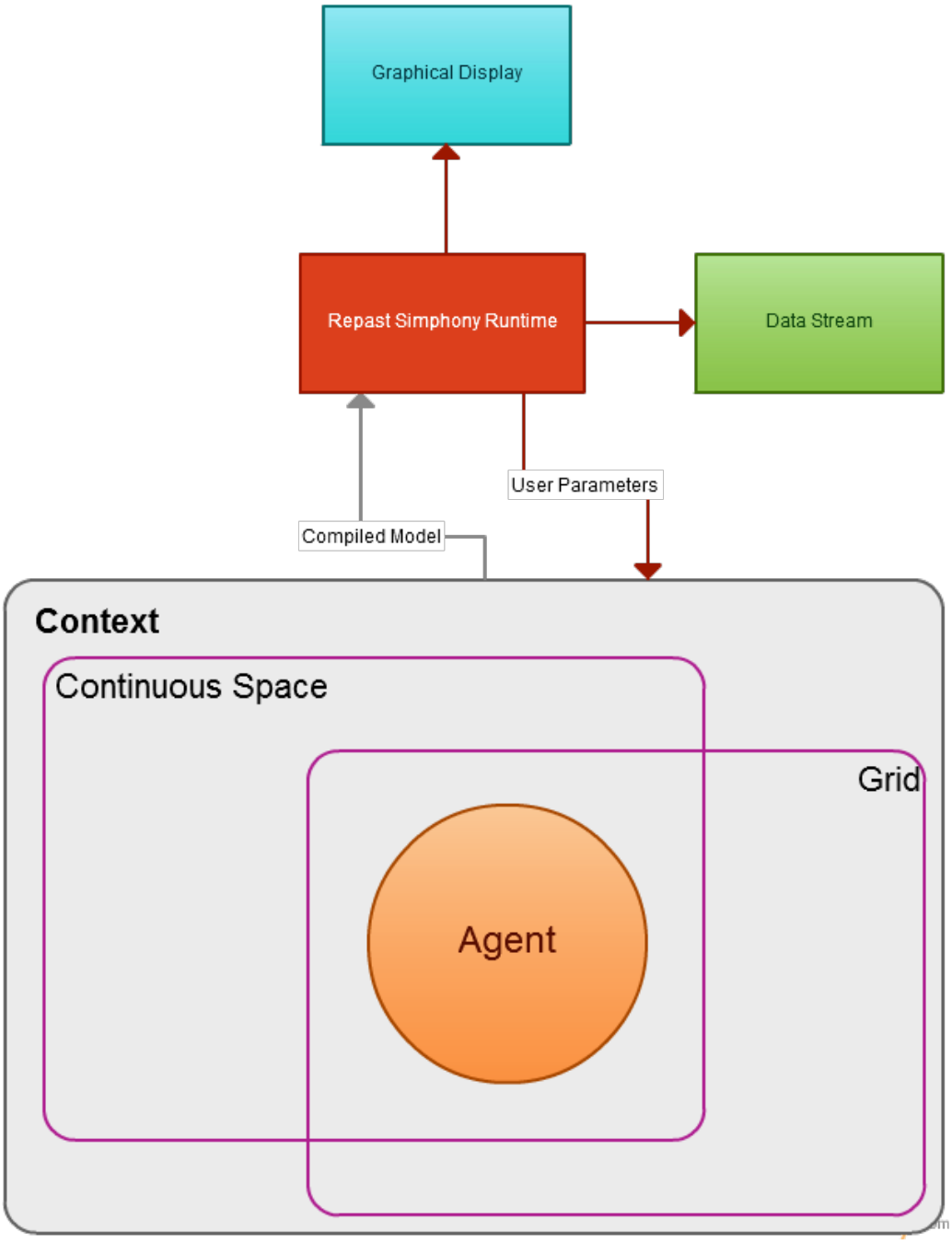
4.1 Model

Although the model has been described multiple times already in this report, it is important to have a clear understanding of exactly what it is trying to simulate when considering the model's design. To reiterate, the model simulates two agent types. The first, the dissenters, attempt to move towards designated 'goal points' and remain there if they reach them. The second, the enforcers, attempt to prevent the dissenters from reaching the goal points and move them away again if they do.

Dissenters act relatively independently. They do share knowledge about the utility of the actions they take, but they only act according to their own knowledge and in pursuit of their own goals, judging their success as an individual.

Enforcers act as a team. They can be split into squads and share their knowledge only with other members of their squad. Their decision making can be affected by the orders of a 'commander' agent and some enforcers can act without consideration of their goal to aid others. They judge their success as a whole.

4.1.1 Structure



4.1.2 Execution

The model executes in 'steps'. Each step can be thought of as one execution of a program loop.

Each step, scheduled methods are carried out. Repast Simphony allows for methods to be scheduled to start on specific steps and repeat with specified intervals. Every scheduled method in a step must finish executing before the next step will begin.

Before steps begin execution the model is initialised, setting the initial state of the environment and the agents.

4.2 Agents

Most behaviour is shared between both agent types and can be described together. Divergent behaviour will be addressed in relevant subsections.

4.2.1 Overview

In the most basic sense, agents in this model are defined by three factors: states, actions and goals. At each step of the simulation, all agents have a state, a set of available actions and a goal. Actions are chosen by agents in an attempt to achieve their goals.

The project model focuses entirely on agent movement in physical space. All agent actions are movements, and when an agent takes an action they move from one position to another. We can say that the initial state of the agent is its initial position and the resultant state of taking a movement action is the resultant position. From this we can ascertain that the state space[1] for this model is the set of all positions in the environment that an agent can occupy. The set of available actions for an agent in a certain state is the set of directions that an agent can move in from that position.

The dissenter agents' goal is to minimise the distance between themselves and the target area. As this model supports multiple target areas this goal can be further specified to say that dissenters aim to minimise the distance between themselves and the *closest* target. All goal points have equal value and a dissenter standing in a goal cell is performing optimally even if there are other goal cells a great distance away from it. This goal can be easily related to the states and actions of the dissenters. If a dissenter moves towards its target it is closer to achieving its goal.

The goal of the enforcers is less easily defined. As they aim to prevent the dissenters from achieving their goal, the enforcers' goal could be defined as the exact opposite of that of the dissenters: To *maximise* the distance between a dissenter and its closest target. However, this goal cannot be easily related to the states and actions of the enforcers. There is no intuitive way to connect the simple action of moving to the more abstract idea of keeping an entire crowd of opposing agents away from a collection of points. In light of this, the enforcers' goal must be simplified in such a way that their behaviour can be designed relatively easily while

[1] The set of all states an agent can occupy

still *resembling* more complex real-world behaviour. To this end we can assume that the key element involved in stopping the dissenters is the simple act of being close to them, meaning that the enforcers' goal can be drastically simplified to say that they must aim to occupy a space with as many dissenters in it as possible.

The enforcers are again simplified when considering the requirement that they be able to move the dissenters and push them away from the goal cells. To retain the simplicity of the model, the act of 'pushing' is not explicitly defined as an action that the enforcers can take. Instead, it is assumed that the enforcers are constantly exerting force upon any dissenters that are in close proximity to them. In this way, the previously defined enforcer goal of simply moving towards dissenters continues to offer a satisfactory compromise between simplicity and realism.

Decisions and Learning

In order for agents to behave in a logical way, they must be able to relate their three key factors. That is, they must be able to choose an action that they believe will result in them occupying a state which is more in line with their goal than the state they currently occupy. In order to *choose* this action, agents must have some way of judging the effectiveness of their available actions. This effectiveness is called *expected utility*, how useful an agent *thinks* an action will be.

The expected utility of an action is based on prior experience. Agents have no way of knowing how effective an action will be until they have performed that action at least once and found out its *actual* utility. This generation of expected utility from actual utility is the basis of *learning* in the model. Exactly how this expected utility is generated is based on the agent's *reinforcement learning algorithm*: Q-learning.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t)(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))) \tag{4.1}$$

$Q_t(s_t, a_t)$ represents the expected utility, called the *Q-value*, of action a in state s at step t. $Q_{t+1}(s_t, a_t)$, the new Q-value, is calculated as the sum of the old value and a newly learned value, based on the agent's learning rate, $\alpha_t(s_t, a_t)$, the actual reward received, R_{t+1}, and an estimate of the future value of the action, proportional to the optimal future value $\max_a Q_t(s_{t+1}, a)$ and the agent's discount factor, γ. In simple terms, the speed at which an agent learns new information is based on the learning factor, and the degree to which the agent considers future possible rewards is determined by the discount factor.

All of the agent's Q-values are stored in a *Q-table* which represents the entirety of an agent's learned knowledge.

4.2.2 Step

The execution of scheduled methods common to both agent types is as follows:

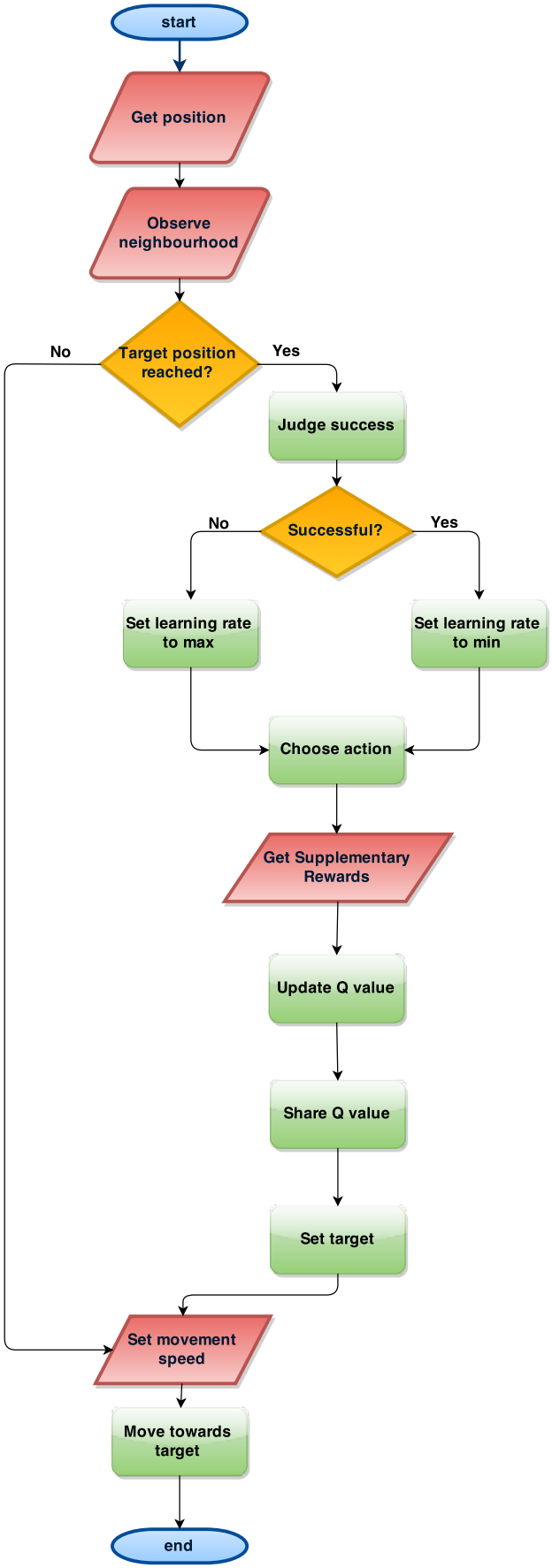


Figure 4.1: Agent behaviour flowchart

4.2.3 Positioning

Agents have a position in both the grid structure and the continuous space of the environment. While calculations are done using the grid position, the agents' behaviour also depends somewhat on their exact location within a grid cell, meaning that both must be updated each step.

4.2.4 Observe Neighbourhood

Some agent behaviour depends on the population of an agent's current cell as well as others nearby. A collection of cells that need to be considered are called a 'neighbourhood', generated using Repast's *nghCreator* class, the size of which depends on an agent's 'range' attribute. These neighbourhoods are generated for the purpose of storing references to objects contained within them. When declaring a neighbourhood, a single class must be explicitly declared, meaning that a neighbourhood can only store references to objects of one class. Agents have behaviours that depend on the number of both agent types in their area, meaning that two neighbourhoods must be generated by each agent.

Agents only require the population of each agent type in each neighbourhood cell, rather than references to the individual agent instances. This means that storage of this information can be simplified. Agents access this data from a three-dimensional array, where the first two indices specify the x and y coordinates of the cells and the third specifies whether the value is the dissenter population (index 0) or the enforcer population (index 1).

4.2.5 Target Position

Although agents decide where to move instantaneously, the actual process of moving to a target location takes a variable amount of time, depending on the local environment. In this model, agents only decide on and take a new action once they have completed their previous action. An agent target position is both an environment grid reference and a specific point in the continuous space.

4.2.6 Success

Agents can be configured to alter their learning rate based on whether they consider themselves to currently be performing 'successfully'. Obviously a key part in this is deciding whether they are successful or not. Each agent type does this differently.

Dissenters

The task of the dissenters is to minimise their displacement from their closest goal point. To determine whether a dissenter is successfully carrying out this task the model can use a relatively simple assumption: "A dissenter is successful if it is moving towards, or has reached, a goal cell."

Determining whether an agent has reached a goal cell is trivial, but the requirement that a dissenter be moving towards a goal cell is slightly more complex. It would be simplest to say that if an agent's current location is closer to the nearest goal point that its previous location it can be said to be moving towards that goal. This is technically true but only accounts for the most simplistic of possible scenarios. There may arise a situation wherein a dissenter must move away from a goal point in the short term to avoid an obstacle and minimise its displacement in the long term. To account for this sort of situation, the agent instead judges its current displacement against its *average* displacement over the course of the simulation. This prevents agents from rapidly changing their judgement of their success before any real results of their actions have emerged.

Enforcers

The idea of an overall 'goal' for the enforcers is slightly different. Rather than attempting to specifically achieve something themselves, they are instead attempting to *prevent* the dissenters from achieving their goals. Although this goal had to be simplified when designing the basic behaviour of the enforcers, the assessment of success can be designed in a more abstract fashion. It can simply be assumed that the success of the dissenters is directly at odds with the success of the enforcers. If most of the dissenters are performing unsuccessfully, then the strategy of the enforcers can be said to be a success.

Behaving more as a single unit, the enforcers judge their success uniformly against the dissenters, meaning that all enforcers have the same 'success' value at all times. This makes sense when considering a real-world scenario: An enforcer holding off one or two dissenters at one entrance to a building would not be a 'success' if there were hundreds of others pouring in through another. The aim of the enforcers is to prevent the dissenters *as a whole* from reaching their goal.

Application

Using a Boolean 'success' attribute, agents choose between two user-defined learning rates: a minimum when successful and a maximum when unsuccessful. This decision is based on a simple logical assumption: If an agent is successful, it must be doing something 'right'. It is assumed that if an agent's current knowledge is leading it to perform well, then it should not endeavour to 'overwrite' this knowledge too quickly. Conversely, if an agent is doing poorly then it is assumed that the agent should be looking to change its behaviour as quickly as possible by taking in a maximum amount of new knowledge about the environment.

4.2.7 Choosing Actions

Both agents have the same set of actions that they can perform. Agents can move to any of the neighbouring cells, including diagonals, or not move at all.

Not all movement actions can be taken in every state. The most obvious example of this is that no agent may move into a blocked cell. Impossible actions, those that have a negative reward in the environment reward table described in section 4.3.3, are not considered as options.

'Weighting' is the process of assigning a value to each possible action which determines how likely it is that the agent will choose it. The higher the weight of an action, the more likely it is to be chosen. The two agent types assign weights to their actions differently.

Dissenter Weighting

Dissenters begin by calculating an initial weight using the following formula.

$$w = \max(Q(s,a) \times |Q(s,a)| + \delta m \times h, 0.1) \tag{4.2}$$

δm represents the difference in the agent's displacement from the closest goal cell between the current state and the resultant state of the action, approximated using a simple Manhattan heuristic. h is the *heuristic factor*, determining how much of an effect this heuristic has on the weighting of the action. The purpose of this heuristic is to simulate basic spatial awareness that would be possessed by people in a real-world scenario. If a dissenter has no expected utility associated with any movement actions it makes sense that they would still move in the general direction of their goal.

The formula ensures that a possible action can never be given a weight of zero or less. There is always some chance, however small, that an action will be chosen regardless of how low its

expected utility is, as long as that action is not absolutely impossible.

The value returned by this formula is used by dissenters in typical situations, but is further modified if the agent is being pushed, as described in section 4.2.1. 'Pushing' is not actually modelled as part of enforcer behaviour, as its simplicity means that only the *effect* of pushing has any bearing on the model. For this reason, the pushing effect is calculated on the side of the dissenter being pushed, as follows.

if $e > d \times p$ **then**
  if $\delta m > 0$ **then**
    $w = 0$;
  else if $\delta m = 0$ **then**
    $w = 0.01$;
  else
    $w = w \times (e^2 + 1)$;
  end
end

Algorithm 1: Push weighting

e and d represent the currently occupied cell's population of enforcers and dissenters respectively. p represents the enforcer 'push factor', denoting how many dissenter agents a single enforcer is able to push. If the population of enforcers is sufficient to push the population of dissenters then all dissenters in the cell are pushed. In order to decide the direction in which dissenters are pushed, the possible movement actions are weighted differently. Those actions that would bring the dissenters closer to their goal are given a weight of zero, as the enforcers would never intentionally aid the dissenters in reaching a goal cell. Actions that do not change the dissenter displacement are given a very small weight, meaning that they can still be chosen if no better options exist. Actions that move the dissenters away from their goals are given much greater weight, which is also proportional to the number of enforcers occupying the position that would result from that action. This helps to establish a simplified 'cooperation' where enforcers are more likely to 'pass' dissenters to other enforcers, who will continue to try and move them away from the goal cells. This is an important behaviour to establish, as if the dissenters are pushed into a cell that is free of enforcers they can immediately move back into the cell they were pushed from.

Enforcer Weighting

The weighting process is where the enforcer agents enact several of the 'team' features that distinguish them from the dissenters. The first of which is the effect of the commander agent

described in section 4.2.13. Where the dissenters use a Manhattan heuristic to approximate their displacement from goal cells, enforcers use this heuristic to approximate their own displacement from the commander's directive, represented by δm in the following formula.

$$w = \max(Q(s,a) \times |Q(s,a)| + \delta m \times h + d \times p, 0.1) \tag{4.3}$$

The initial weighting formula is similar to that of the dissenters but adds an additional factor, d, the population of dissenters in the resultant position of the action. p, the *population factor* affects how much impact this has. This factor encourages enforcers to move towards local groups of dissenters, even if the expected utility of this movement doesn't indicate that many dissenters were encountered there previously. It makes sense to assume that the enforcers would be able to see groups of dissenters in front of them and approach them, instead of relying entirely on their knowledge of where dissenters had gathered previously.

Like the dissenters, enforcers further modify weights after initial calculation. This introduces the second team mechanic, scouts.

if *scout* **then**
  if *action = stay still* **then**
    $w = 0$;
  else
    $w = 1$;
  end
else if $e > c$ **then**
  $w = 0.01$;
end

Algorithm 2: Scout and crowding weighting

Scouts do not act with consideration of the utility of their actions, acting solely in pursuit of knowledge. To achieve this, all of their actions are weighted equally except standing still, which they never have a reason to do. See section 4.2.13 for further explanation of special enforcer types.

e represents the population of enforcers in the resultant position of the action and c represents the 'crowding value'. If e exceeds c then the cell is deemed to be 'crowded' and enforcers will prefer moving into other cells. As long as the crowding value is set to an appropriate number this behaviour makes sense, as having a few enforcers standing in one cell is generally just as useful as having a much larger number.

4.2.8 Supplementary Rewards

The only rewards in the model that are static are those granted when a dissenter agent moves into a goal cell, as the goal cells never change location. All other rewards are dynamic and based on variable factors. These *supplementary rewards* are calculated as required.

Dissenters

The dissenters should try to avoid enforcers and find alternative routes if a path is blocked by them. To encourage this, dissenters are given a negative supplementary reward each time they move to a cell, proportional to the population of enforcers in that cell. The magnitude of this negative reward is determined by an *avoidance factor*.

Enforcers

The enforcers' supplementary reward is directly related to their goal. They receive a positive reward proportional to the population of dissenters in the target cell. As this is the only reward that the enforcers receive it does not need a variable factor to weight it against other reward sources.

4.2.9 Updating Knowledge

Agents update their knowledge using the Q-learning algorithm described in section 4.2.1. The reward value used in the equation is the sum of the value in the agent's reward table and the supplementary reward.

4.2.10 Sharing Knowledge

After an agent updates its knowledge it shares the updated Q-value with other agents within range. Agents treat the shared knowledge they receive similarly to the knowledge they learn themselves, using the following formula.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + b\alpha_t(s_t, a_t)(Q_s(s_t, a_t) - Q_t(s_t, a_t))) \tag{4.4}$$

$Q_s(s_t, a_t)$ represents the shared knowledge that has been received and b is the *belief factor*, which determines to what degree an agent trusts the information that it receives. A belief factor of 1 will cause an agent to treat shared knowledge as if it were its own. A value of 0 means an agent will not consider shared knowledge at all.

4.2.11 Movement

The final action taken by agents during a step is to move towards their current target location. Agents set their movement speed every step according to the number of other agents occupying the same cell, with agents of opposing type having a greater effect than those of the same type.

4.2.12 Knowledge Decay

The knowledge decay method serves to make agents 'forget' the knowledge they have learnt. At each execution, every Q-value held by an agent is multiplied by a *decay factor* between 0 and 1, lessening the influence it has when weighting actions. The result of this is that newer knowledge that has not decayed as much is treated with greater importance when making decisions and information that is no longer relevant gradually fades away.

4.2.13 Special Enforcer Types

As described in section 4.1, there are two additional agent sub-types which can be used as part of the enforcer team.

Scouts

Scouts are enforcer agents that are designed to focus on collecting information, rather than taking part in the opposition of the dissenter agents. They move around the patrol area (see section 4.3.1) randomly, without considering any sort of goal. However, they do still judge the utility of their actions with regards to the enforcers' goal and share this knowledge with others.

Scouts can help the enforcers react more quickly to changing situations by remaining spread throughout the patrol area, rather than staying in areas with high expected utility. They also have a wider range when sharing information and do not restrict their information to members of a single squad.

Commander

The commander is an agent that aims to coordinate the enforcer team somewhat. The commander polls the rest of the team at scheduled intervals to ascertain which enforcer is currently surrounded by the greatest number of dissenters. The location of this enforcer becomes the *directive*, which is used in weighting calculations as described in section 4.2.7. To facilitate this behaviour, normal enforcer agents also have a scheduled method where they record the number of dissenter agents in their immediate vicinity.

4.3 Environment

As specified previously, the model environment consists of two distinct structures: A continuous space and a grid. Both of these structures are generated at runtime from a map file, saved in PNG format.

4.3.1 Environment Properties

Each cell in the environment grid has a 'type', determining the effect that it has on the agents in the model. There are five possible cell types:

- Blank - Allow passage for dissenter agents but not enforcers
- Dissenter Spawn - Dissenters begin the simulation within these cells and can move through them.
- Patrol Area - Enforcers begin the simulation within these cells. Both agent types can move through them.
- Goal - Dissenters attempt to reach these cells. They allow passage for both agent types.
- Blocked - No agents can pass through

Because they cannot move through blank spaces, enforcers are restricted to moving within the patrol area and the goal area. Dissenters can move through all cells except those which are blocked.

In order for the model to function as intended, an environment must contain at least one dissenter spawn cell, patrol area cell and goal cell. There is no upper limit on the number of any cell type that can appear in an environment.

4.3.2 Mapping

The size and shape of an environment is defined simply by the dimensions of the input map. The type of each cell is defined by the colour of each pixel in the map.

Within the PNG file format there is some variation in the way that the colour of each pixel is saved. In order to ensure a simple, easy process for creating map files the software will only make use of three colour channels: Red, Green and Blue. The use of an alpha channel could increase the flexibility of the mapping system, especially if more environmental features are

implemented in the future. However, alpha channel editing is not supported by all common image editing software and requires a certain degree of prior experience to perform confidently.

Generally speaking, PNG files record pixel colour values with a depth of 8 bits per channel, meaning that each grid cell in a map can contain three integer values in the range 0-255. This offers a vast amount of different combinations[2]. Because the model environment only requires the distinction of five different cell types, the way in which colour channels are read can be simplified considerably. Each channel need only be read as 'on' or 'off', effectively treating each byte in the map as a single bit. This system provides eight distinct integer values per pixel, enough to cover the required five types and several more that may need to be added in the future.

To ascertain if a channel is 'off' or 'on', the software reads each channel as an integer value. If this value is greater than 127 the channel is 'on', otherwise it is 'off'. Reading colour values this way offers considerable error-tolerance with user-created maps. A person with experience using image editors may know to ensure that all pixels intended to be 'red' have an RGB colour value of exactly (255, 0, 0). However inexperienced users may simply pick a red colour from a swatch that may also contain some amount of blue or green. This error tolerance also extends to map files that may have been mistakenly saved in lossy formats and converted to 24-bit PNG images later. Although the exact nature of lossy compression is highly variable, this system should lessen its impact considerably.

Type	Red	Green	Blue
Blank	1	1	1
Dissenter Spawn	1	0	0
Patrol Area	0	0	1
Goal	0	1	0
Blocked	0	0	0

Table 4.1: Cell type colour values

It may seem at first to be slightly unintuitive to associate a 'blank' cell with completely 'filled' colour values. However, the majority of image editors will fill a new image with pure white by default, making this a good choice for what can be seen as the most basic or 'default' state of a cell.

[2] 16,777,216 to be exact

4.3.3 Environment Rewards

The goal points of the environment determine the rewards received by dissenter agents, whose aim it is to reach these goal points. In terms of our model we can say that, for dissenter agents, if taking a certain action in a certain state results in a state where the agent is positioned at a goal point, that state-action pair should have a reward associated with it. These goal point rewards are the only ones that need to be considered when generating the model environment. No other movements have an intrinsic reward associated with them and any other rewards that either agent type could receive are calculated dynamically as and when they are required. The goal point rewards are the only *static* rewards.

These static rewards are stored in a *reward table* represented by a three-dimensional array, with the first and second indices representing grid locations on the x and y axes and the third determining which movement action (i.e., which direction) is associated with the reward value contained inside. This reward table is also useful for defining which actions agents cannot take. For example, if a state-action pair has a resultant state where an agent would be positioned inside a blocked cell that state-action pair should never be performed by an agent. To represent this, a negative reward value is used.

As each agent type has different restrictions on movement, two reward tables are generated at runtime. Each agent is passed the appropriate table.

4.4 Front End

4.4.1 Parameters

The model contains a large number of variable factors that can affect the behaviour of the agents. To ensure a strong potential for experimentation, a wide variety of these variables should be user-editable from within the front end. Within Repast, user-editable variables are called *parameters*. The following parameters are implemented (items marked with * have separate parameters for each agent type):

- Discount factor*
- Communication range*
- Knowledge decay factor*
- Heuristic factor*

- Maximum and minimum learning rate*
- Belief factor*
- Number of agents*
- Number of enforcer squads
- Number of enforcer scouts
- Enforcer leader (Boolean)
- Enforcer push factor
- Enforcer population factor
- Dissenter avoidance factor

Chapter 5

Implementation

For the development of the project software, an iterative, agile development cycle was chosen. Planning out distinct versions of the software provided useful milestones and helped development to maintain a consistent schedule.

One of the primary considerations when developing the model was the idea of modularity. Ensuring that elements of the model could be altered independently of one another was key to fostering an evolutionary workflow and helped to facilitate the rapid addition of extra features.

5.1 Version 1 - Q-learning

The proper implementation of agent reinforcement learning is the central focus of the project, therefore it was the first consideration when development began. Ensuring that the Q-learning algorithm was suitable for the kind of agent behaviour the model aimed to simulate was important to avoid wasting time later in development if this turned out not to be the case.

The model environment at this stage consisted of only a grid structure, as opposed to the later two structure solution, and featured just a single agent. This agent replicated the intended dissenter behaviour by learning the shortest route from its starting location to a goal cell. This agent did not make use of a heuristic like the dissenters in the final version and when it reached the goal cell it was moved back to its start location to try again. The agent would begin the simulation with no knowledge about the environment and would explore at random until happening upon the goal. Gradually, the agent would build up its knowledge from reaching the goal cell over and over again and would complete the course more quickly, until it eventually found an optimal solution. This agent behaviour was tested in a variety of environments ranging

from completely open spaces to winding mazes and was able to find an optimal solution in all cases. At this stage the environment grid had four cell types: blank, blocked, spawn and goal. Unlike later iterations, this version could only support a single spawn and a single goal.

These early tests indicated that the desired dissenter behaviour could be implemented and that Q-learning was a suitable reinforcement learning algorithm to use for the task. This version also cemented the use of an environment grid structure with a generated reward table.

5.2 Version 2 - Movement and Enforcers

While the behaviour showcased in the first version of the model was encouraging, the grid-based movement quickly became dissatisfactory, especially when considering the intention to use the model to mimic real world scenarios. Expanding the environment to facilitate more realistic agent movement was the first aim of the second version of the model.

The decision to add a continuous space structure, wherein agents could exist at non-integer positions, was obvious. However, the existing agent behaviour functionality was dependent on having the environment represented by a grid. It was for this reason that the two structure environment solution was implemented. Agents were able to continue using the grid to perform calculations and assess the utility of their actions, but the user could view them moving in a smooth, realistic way. It was at this stage that multiple dissenter agents were used simultaneously for the first time. Because these agents were designed to act completely autonomously, no changes to their design were required to achieve this.

Continuous agent movement allowed for the introduction of a new factor affecting agent behaviour: movement speed. Agents could now check how many others they were sharing their currently occupied grid cell with and adjust this variable accordingly. Having agents move more slowly when crowded together was another major step in improving the realism of the model.

This model version also introduced the enforcers, the second agent type. At this stage, the enforcers' behaviour was almost identical to that of the dissenters, except that they received rewards for encountering dissenters rather than reaching the goal. At this stage the enforcers were not at all effective at repelling the dissenters. While they did cause the dissenters to move more slowly when occupying the same grid cell, they had no means of pushing the dissenters back and dissenters did not receive any negative rewards for encountering them. At this stage, because the two agent types behaved so similarly, most agent behaviour was moved into a general 'agent' superclass, with unique behaviour being overridden in individual subclasses.

To facilitate the addition of the enforcers and the introduction of simulating multiple agents

simultaneously, the environment grid was updated with some additional functionality. The 'patrol area' type was introduced to spawn and contain the enforcer agents. The environment was also updated to support multiple dissenter spawn points.

5.3 Version 3 - Opposition and Cooperation

The intention with version 3 was to balance the model by introducing behaviours that would allow the enforcers to effectively oppose the dissenters. The implementation of the 'pushing' functionality described in section 4.2.7 was crucial to achieving this. Before this addition the dissenters could only be slowed, they would never lose the progress they had made. Now that the enforcers were able to push back it was possible for the model to reach a relatively stable state where the dissenters were unable to reach the goal at all.

Pushing added a new way for dissenters to be removed from the goal cell, as opposed to the simple 'respawning' that had been used since version 1. The respawn functionality was removed in this version, leaving the responsibility of clearing the dissenters from the goal to the enforcer agents. This was obviously a big step up in terms of realism and also improved the way the model 'progressed' over time. Rather than having the dissenters reach the goal cell and simply stay there, having 'won', the model now possessed an 'ebb and flow' where one side would seem to have won for a while but would then be overcome by the other.

To improve the behaviour of the dissenters, now that they could be effectively opposed, a negative supplementary reward was added to dissenter movements which led to encounters with enforcers. Making these encounters undesirable encouraged the dissenters to seek out alternative pathways to the goal in the case that one was being effectively defended.

Finally, knowledge sharing was introduced. This change drastically affected the way in which agents behaved, bringing the model much closer to its goal of simulating two opposing *groups*, rather than two collections of simply like-minded agents. Introducing cooperation meant that the dissenters would much more quickly find new routes to the goal, but would be met by a much quicker response from the enforcers as well. It was at this point that the behaviour exhibited in the model really started to resemble that seen in the real world.

5.4 Version 4 - Additional Features

The completion of version 3 marked the completion of what could be described as the *core* model functionality. At this stage the model was able to satisfactorily replicate the basic behaviours

required of the agents and the resultant simulation offered quite a compelling recreation of a real world phenomenon. The aim of version 4 was to introduce some additional, more detailed agent behaviours and user functionality, as well as to continue tweaking the existing functionality.

5.4.1 Multiple Goals

In version 3, environments still only supported one goal cell. Version 4 added support for any number of goal cells, either adjacent to one another, forming a 'goal area', or in entirely separate locations. This addition was relatively simple, but did require some adjustment of agent behaviour, particularly the heuristic calculations used by dissenters when weighting choices. It was important to ensure that dissenter agents valued goal cells equally regardless of the proximity of other goal cells (e.g., a dissenter will not favour a group of several adjacent goal cells over a single goal cell).

5.4.2 Map Reading

In earlier versions, creating and editing environments for use with the model was incredibly tedious, relying on a manually filled two-dimensional array to set the type of each grid cell. This was obviously not an appropriate method for an end user. To streamline the process, the map reading functionality described in section 4.3.2 was implemented. Using a visual approach to mapping made the process of environment design significantly faster and more enjoyable.

5.4.3 Knowledge Decay

A problem identified with the version 3 model was that agents reacted to new information quite slowly, continuing to act on old knowledge long after it had ceased to be relevant. For example, following the successful blocking of a group of dissenters in one doorway of a structure, enforcer agents would continue to gather around that doorway long after the dissenter agents had moved away. To remedy this sort of behaviour, knowledge decay, as described in section 4.2.12, was introduced to speed up agent reactions times. By making old knowledge less important than new knowledge, agents were observed to adjust their behaviour in a much more convincing way.

5.4.4 Special Enforcer Types

The final requirement that had yet be fulfilled in previous versions was that enforcers should behave in a more organised way than the dissenters. To achieve this, scouts, squads and commanders were implemented, as described in section 4.2.13. These changes altered enforcer

behaviour in a more subtle way than other additions but added a considerable amount of depth for experimentation purposes, giving the end user three additional parameters to change.

5.5 Testing

In using an agile development cycle, testing of the functionality of the model was done iteratively with each new software version. This meant that by the time version 4 was completed there was little in the way of testing required insofar as ensuring the software performed correctly. However, significant time needed to be invested in testing the many different variables used by the model itself. Small adjustments in the numerical values used by the model could result in significant changes in agent behaviour. This, combined with the sheer number of such numerical values that the model relies on meant that a great deal of time was spent using trial and error testing to decide on a 'default' setting for the model, with the aim being to create a reasonably balanced, realistic simulation. Initial testing of this nature was done in a relatively laissez-faire manner with more detailed, methodical experimentation performed later, described in chapter 7.

5.6 Optimisation

Optimisation is a particularly important factor when implementing multi-agent systems, because they rely on simultaneously simulating large numbers of agents, each perform their own calculations every step. To ensure that end users have the freedom to run the model using large numbers of agents, it is important to make individual agent behaviour as streamlined as possible. Another consideration is that the project software is intended for use in experimental applications where models are required to run many times. An end user should be able to run the model several hundred times in a reasonable amount of time.

5.6.1 Scheduled methods

The most significant optimisation came from moving some behaviour from methods that executed every step to those scheduled to run at certain intervals.

The higher the interval between the executions of a scheduled method is, the less performance improvement can be achieved by raising it. Raising the interval of a method from one step to two steps has a drastically greater effect than raising the interval of an equivalent method from 100 steps to 110 steps. It was this realisation that led to a focus on high frequency methods

when attempting to optimise the model. The first methods to be optimised in this way were those that agents used to gather information about their surroundings. Both agent types would execute methods every step to generate multiple 'neighbourhood' data structures and record the population of other agents in the surrounding cells. While this was an optimum solution as far as model accuracy is concerned, it brought with it significant computational expense. An agent having a slightly incorrect view of their immediate surroundings could result in some marginally different behaviour, but would never result in the model failing to execute or agents behaving in a noticeably 'incorrect' way. It was for this reason that a compromise was made and the frequency with which agents updated their neighbourhood structures was reduced to once every three steps. While extensive, methodical testing might reveal a slight change in the behaviour of agents as a result of this alteration, any such change is not noticeable by a human observer and brings with it the benefit of a significantly optimised agent execution loop.

Knowledge decay would originally occur every step, affecting agents' Q-values by a tiny amount. The decay function works by cycling through every value in an agent's Q-table, which is represented by a relatively large three-dimensional array, making the operation quite computationally expensive. The function was originally scheduled to run every step in order to ensure that it wouldn't cause any jarring changes in agent behaviour. It was later realised that an agent's Q-values are only used when an agent makes a decision, which does not happen anywhere near as frequently as once per step. The model was tested using different intervals for decay and it was found that there was no noticeable difference in agent behaviour if the decay function was scheduled to run only once every 50 steps, a drastically more streamlined approach. Other scheduled methods, like the enforcer commander's 'direct' function, wherein all enforcers in a large range are polled to ascertain the location of the highest concentration of dissenters, were tested similarly. Although this function was already scheduled to be performed less frequently than every step, it was still possible to increase the execution interval even further without affecting the model in a noticeable way.

Chapter 6

Professional and Ethical Issues

6.1 Ethical Concerns

The phenomenon that this project is based around, that of civil unrest, has a long and contentious history. Civil unrest is inescapably linked to matters of politics, economics and culture. It is for this reason that particular care was taken throughout the creation of this project and the associated software to always remain impartial. While the project should strive to accurately recreate the dynamics of unrest, it should never make any statements or conclusions about the value or legitimacy of such unrest.

It is also especially important not to 'take sides' when creating a model that simulates two opposing teams. The names 'dissenter' and 'enforcer' were chosen to clearly communicate the roles played by the agents without relating them too closely with any specific real world scenarios or designating either as the 'good guys'. The project does not aim to create a model which 'prefers' either side, it focuses entirely on realism and accuracy.

6.2 British Computing Society Code of Conduct & Code of Good Practice

The Code of Conduct and the Code of Good Practice laid out by the British Computing Society offer sets of guidelines to consider when working in computing. All parties involved in the creation of this project were aware of the BCS codes and strived to follow them at all times. Very few of the guidelines laid out in the codes were actually relevant to the development of this project, considering its small scale.

Chapter 7

Experimentation

Creating a model that is viable for experimental applications was a primary consideration of this project. This chapter will summarise some experiments performed using the model which demonstrate its utility as a research tool.

Experiments using this model can be split into two types. The first type, a *model experiment*, is used to explore the impact of changing the way in which agents work. These experiments focus on the behaviour of the model itself, rather than the scenarios that it tries to recreate, and how different agent programming techniques (e.g., communication, learning/discount rates, etc.) affect the performance of agents. The conclusions of this type of experiment could be useful in designing agent-based models that replicate other real world phenomena or multi-agent systems designed to efficiently solve specific problems.

The second type of experiment, a *scenario experiment*, treats the model as a tool for replicating real world conditions. This is the sort of experiment that would be performed by sociological researchers or law enforcement agencies. They are not concerned with the specifics of exactly how the agents work, instead focusing on testing variables that can be applied to the real world, like the number of participating agents, or by using specific environment layouts.

7.1 Testing Standards

In order to ensure consistency of results, elements of the model that are used in multiple tests must be explicitly defined, including both agent behaviour and the model environment.

7.1.1 Defaults

To maintain consistency across multiple experiments, a set of default values for the model parameters need to be specified. The default values chosen are as follows:

Parameter	Value
Dissenter Avoidance Factor	80
Dissenter Belief Factor	0.5
Dissenter Communication Range	3
Dissenter Discount Factor	0.9
Dissenter Heuristic Factor	0.2
Dissenter Knowledge Decay	0.95
Maximum Dissenter Learning Rate	0.9
Minimum Dissenter Learning Rate	0.1
Enforcer Belief Factor	0.5
Enforcer Communication Range	4
Enforcer Population Factor	10
Enforcer Discount Factor	0.9
Enforcer Heuristic Factor	10
Enforcer Knowledge Decay	0.9
Enforcer Push Factor	2
Enforcer Leader	True
Enforcer Maximum Learning Rate	0.9
Enforcer Minimum Learning Rate	0.1
Number of Enforcer Squads	4
Percentage of Enforcers as Scouts	10
Number of Dissenters	200
Number of Enforcers	100

These values were decided gradually throughout development as new functionality was added. They result in generally balanced behaviour using a variety of different environment maps. The exact reasoning behind choosing these values as default is not hugely important as they serve mainly to facilitate comparison across different experiments.

7.1.2 Map Formalisation

A typically sized map for use with this model is usually at least ten tiles square. This results in an environment with at least 100 individual tiles, each with five possible states. Obviously the total number of distinct maps that can possibly be created at this size is enormous. In order to methodically test the impact of environmental properties on the behaviour of the model, these properties must be formalised.

While the model is designed to be able to use environments that represent a wide array of different real-world locations, there is one 'type' of environment in particular that has provided particularly interesting results throughout development. It attempts to model a situation where enforcer agents are charged with defending a structure from 'invading' dissenters. Enforcers patrol the interior of the structure and the dissenters spawn on the perimeter of the map, their goal being positioned at the structure's centre. In order to ensure that this environment type is easy to use, the structure itself is a simple square with a 'doorway' at the centre of each 'wall'. The size of the doorways is adjusted to alter how 'tight' or 'open' the environment is. The corners of the structure are filled in as early observations showed that agents would sometimes get stuck in these areas. The perimeter of the map is a fixed distance from the walls of the structure.

7.1.3 Testing Procedure

There is no 'win condition' for either agent type within the model and no end state for any of the environment types. Because of this, model runs are simply concluded after a fixed number of steps have executed. In order to strike a balance between experiment quality and efficiency, each run of the model is conducted for 50,000 steps.

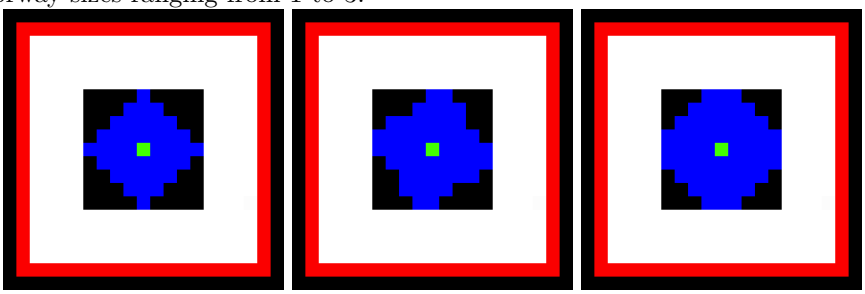
Despite the large number of variables that can be changed in experimentation with this model, there is only one value that needs to be considered in the results of the vast majority of possible tests: how 'well' the agents do. The model value used to determine this is the average of all dissenter agents' displacement from their nearest goal point. The lower this value, the better the dissenters are doing. This value indicates the effectiveness of both the dissenters and the enforcers, as they are in direct competition.

7.2 Model Experiments

7.2.1 Considering Learning Rates and Communication in Different Structures

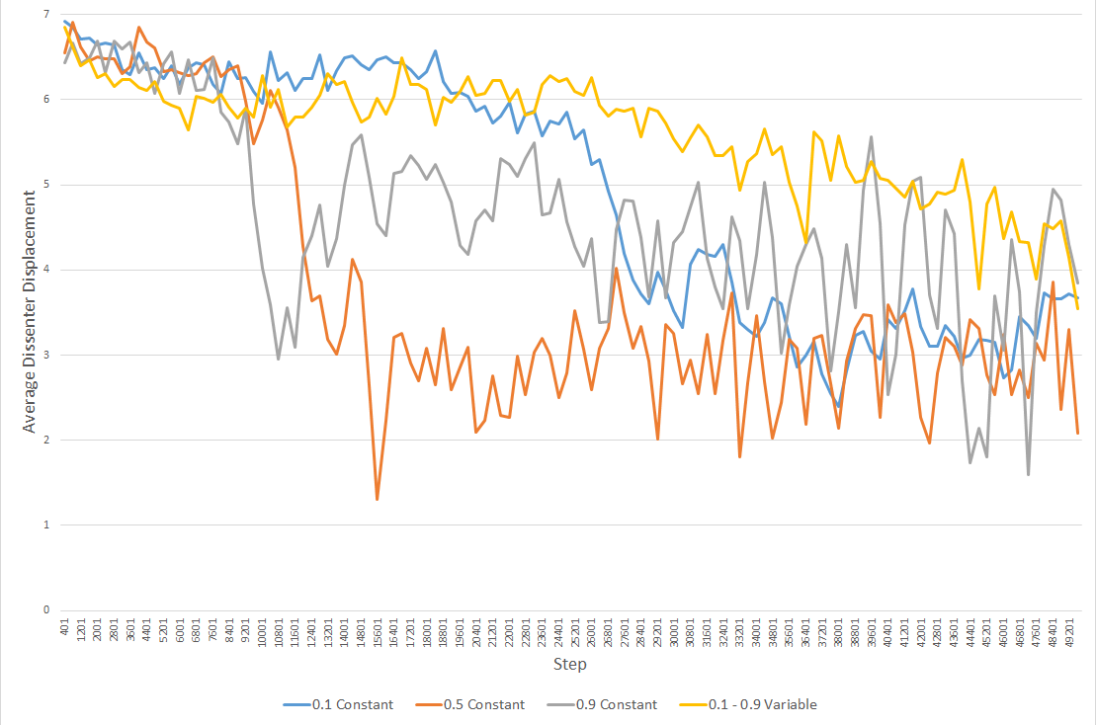
Learning rate, as explained in previous chapters, basically determines how quickly agents take on new information. This factor can either be constant or variable depending on agent success. This experiment aims to investigate how altering the 'openness' of an environment affects the role played by learning rate and other factors in agent performance.

The environment used for these experiments is a structure with a size of 9 cells across and doorway sizes ranging from 1 to 3.



Dissenter Learning Rate in 'Tight' Environments

This experiment demonstrates the impact of altering dissenter learning rate in a structure with a doorway size of 1. Four rates were tested: constant 0.1, constant 0.5, constant 0.9 and variable 0.1-0.9.



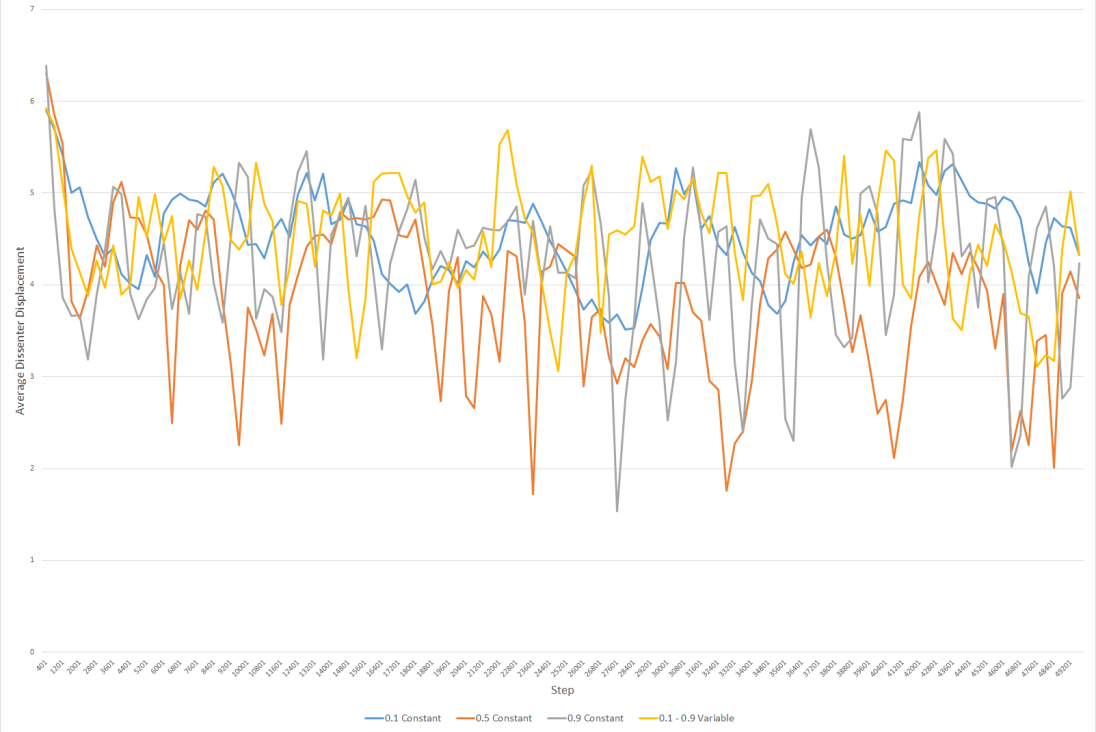
There is a clear difference in agent effectiveness when using different learning rates. The graph shows a constant rate of 0.5 to perform significantly better than the other options, achieving a low displacement early on in the run. The higher rate of 0.9 also reaches a low point quite quickly but fluctuates greatly throughout the model, while the 'middle ground' option performs more consistently. When thinking intuitively about the model this makes sense: A high learning rate allows dissenters to take on new information about possible routes to the goal more quickly, but also means that when encountering resistance dissenters are very quick to 'give up' on a route, learning that it is not viable and returning to exploration.

A constant low learning rate performs as expected: Performance is more consistent but worse overall. Agents utilising the low learning rate do reach a favourable position later on in the run but this takes significantly longer than those using higher rates. This indicates that a low learning rate might be preferable in scenarios where speed is not an important factor and long-term consistency is valued.

The variable learning rate performs particularly poorly in this experiment. The sound reasoning behind this variability has not translated into practical performance. Determining whether this result is unique to this scenario would require substantial additional experimentation.

Dissenter Learning Rate in 'Open' Environments

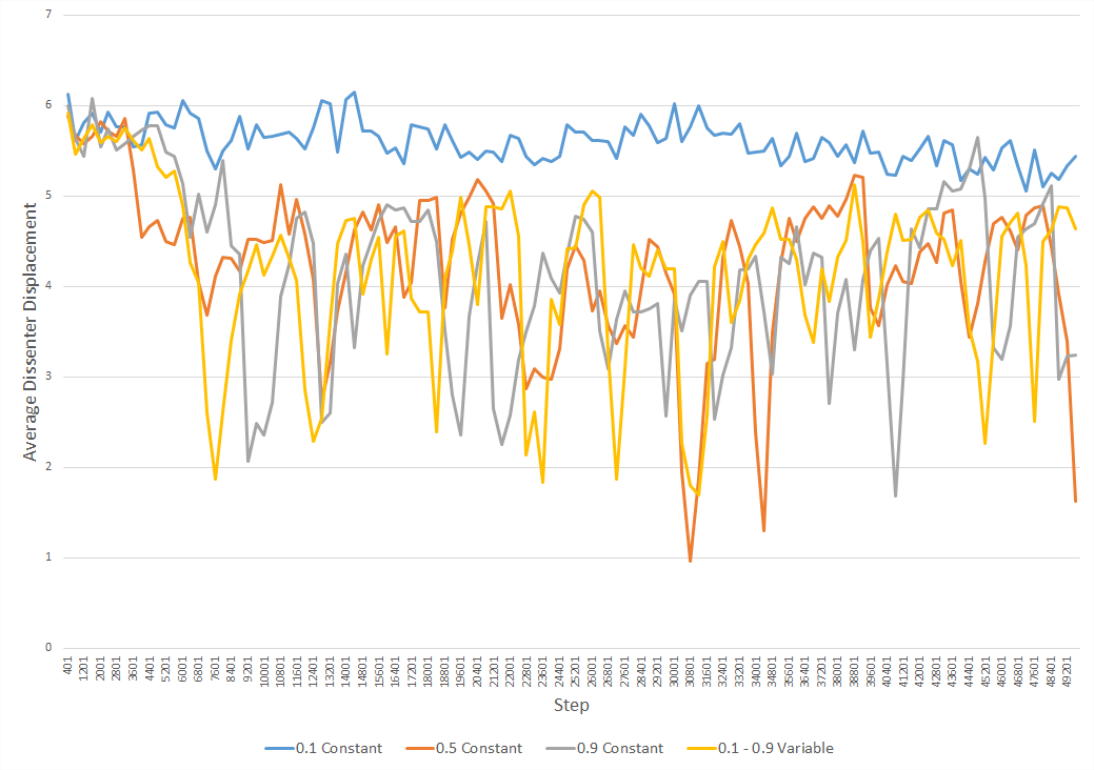
The same experiment was repeated using a structure with a doorway size of 3.



The results in general show a much quicker initial rush towards the goal. With wider doorways, this structure is much less suited for reliably defending against invading dissenters. However, what is interesting in contrast to the previous experiment is that the learning rate of the dissenters seems to have a much smaller effect on overall performance. The general trend of higher learning rates being less consistent is still observable but in terms of performance the rates are more evenly matched. The 'ranking' of effectiveness of the four learning rates is still the same as the previous test with constant 0.5 coming out on top again but the effect is less pronounced.

When observing the model visually, the reason behind these results is revealed. The overall 'flow' of the model is quite different when using a more open structure. Dissenter agents are able to consistently push through the enforcer defences from all sides and approach the goal before being repelled. This is in contrast to the tight environment, where dissenters typically had to find a 'route' to the goal, rather than simply moving in its general direction. Without the need for this more sophisticated knowledge of routes, learning rate becomes less important.

Enforcer Learning Rate in 'Tight' Environments



Changing the learning rate of the enforcers in a tight environment appears to have the opposite effect to that of changing the rate of the dissenters. Most noticeably, the employment of a low, constant learning rate is by far the most effective choice. The other three rates perform very similarly.

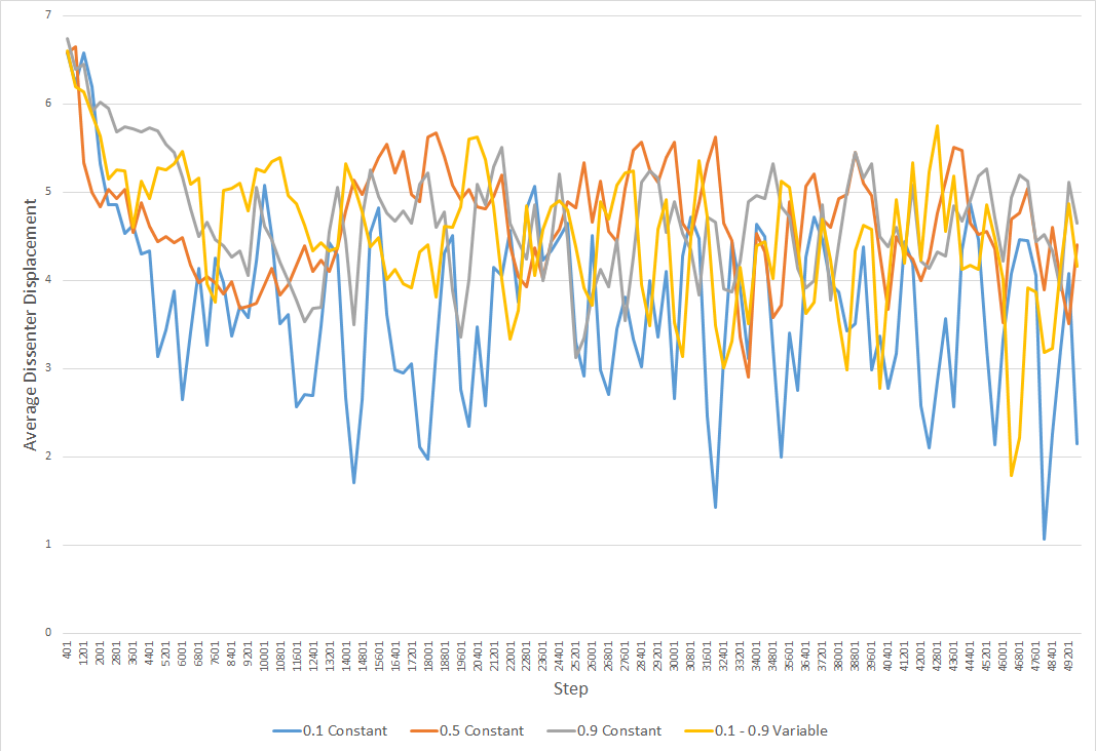
The graph paints a clear picture of the overall course of each run. It can be observed that at roughly 4500 steps into their respective runs, all of the rates except the lowest 'fail' and dissenter displacement drops rapidly. The lowest learning rate on the other hand performs consistently from start to finish. The reasoning behind this is that the only thing that the enforcers need to do to perform optimally in a tight environment like this is stand in the doorways and block the dissenters. There are enough enforcers in the default setup to make this a permanently viable tactic in this environment. Those enforcers employing a low learning rate are content to remain in place without altering their behaviour. Those with a higher rate are more susceptible to information communicated to them by others, which results in enforcers moving around more and leaving doorways unguarded.

The positive aspect of higher learning rates in this scenario is that when the dissenters do reduce their displacement significantly (i.e., they find an unguarded pathway and 'make a break' for the goal) the enforcers are very quick to react and push them back (as shown by

the many sharp spikes downwards on the graph). The graph shows that at no point did the dissenters manage to properly break through the defences of the enforcers using the low rate, but it is reasonable to assume that if this were to occur the response would be markedly more sluggish.

Enforcer Learning Rate in 'Open' Environments

This experiment used a doorway size of 2.

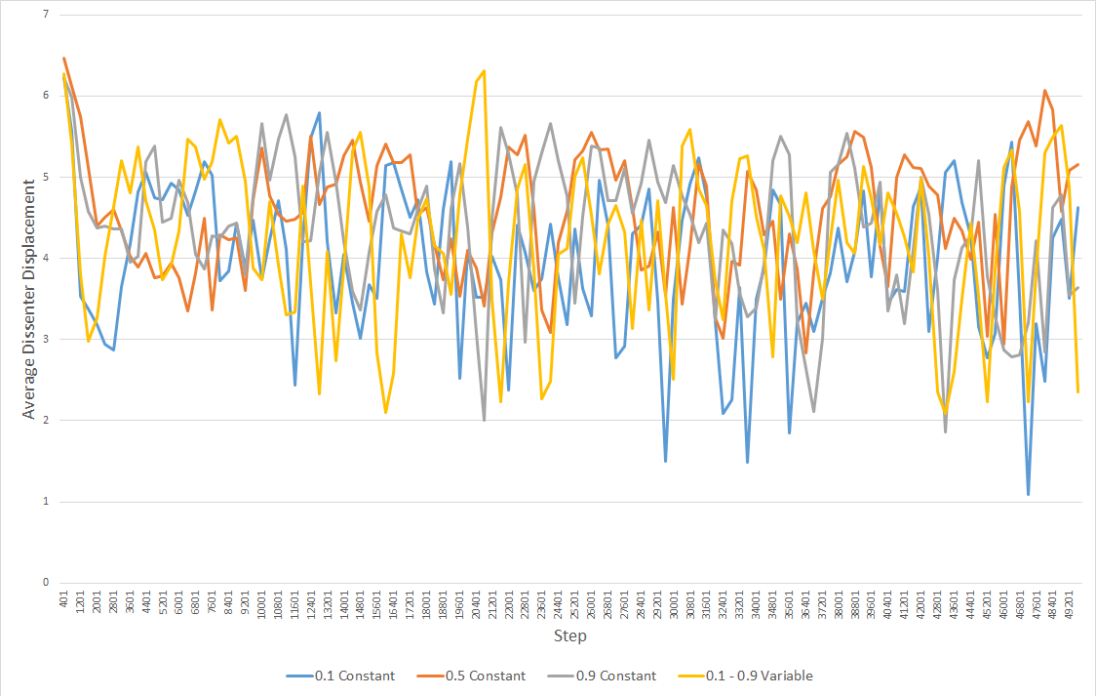


Widening the doorways of the structure by just one cell has a drastic effect on the relationship between learning rate and performance in enforcers. The constant low rate, which was previously optimal, now performs much worse than the other options. The other three options perform better than they had previously but are still quite similar.

By applying the conclusions made in the previous experiment to this set of results, we can conclude that the 'sit in the doorways' tactic does not work in this environment. This number of enforcers is evidently insufficient to completely block off the wider entrances. As a result, the dissenters are able to enter the structure no matter what the enforcers' learning rates are. In this scenario, the benefit of the higher learning rates becomes apparent. High-rate enforcers are able to respond to invading dissenters much more quickly than low-rate ones.

Further widening of the doorways all but erases the impact of learning rate on enforcer performance. They perform poorly in general no matter what rate they employ. This scenario

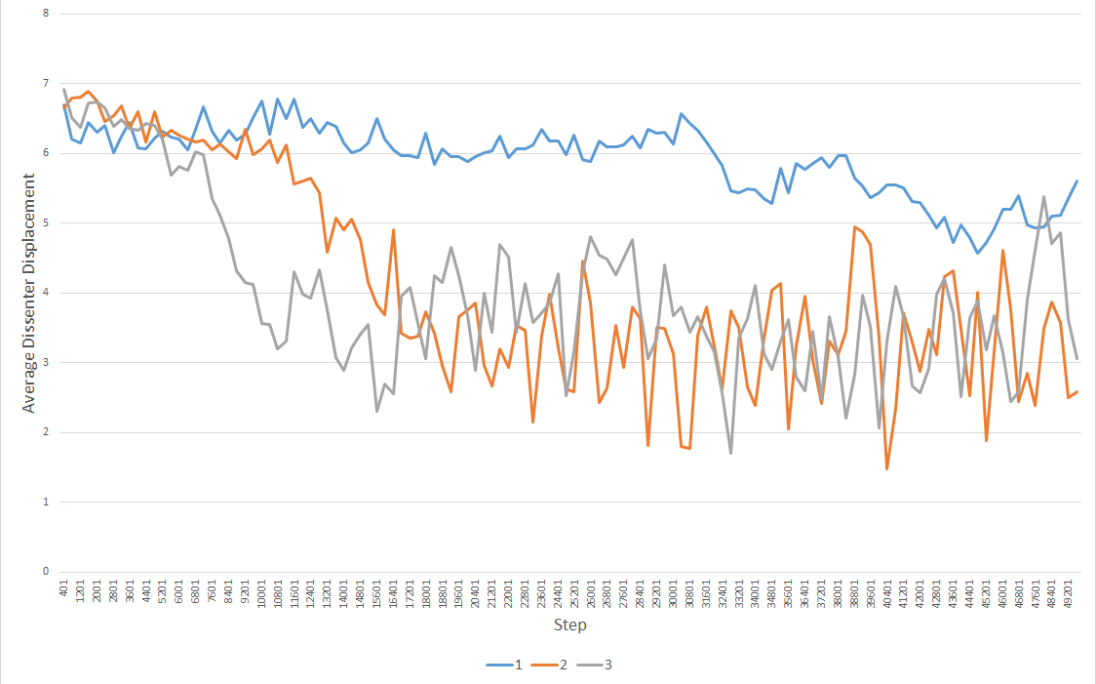
resembles more of a 'free for all' as the impact of the environment structure is lessened.



It's reassuring to observe in the enforcer experiments that a variable learning rate is comparable in performance to other, constant rates. This suggests that there may be some scenario in which a variable learning rate is optimal.

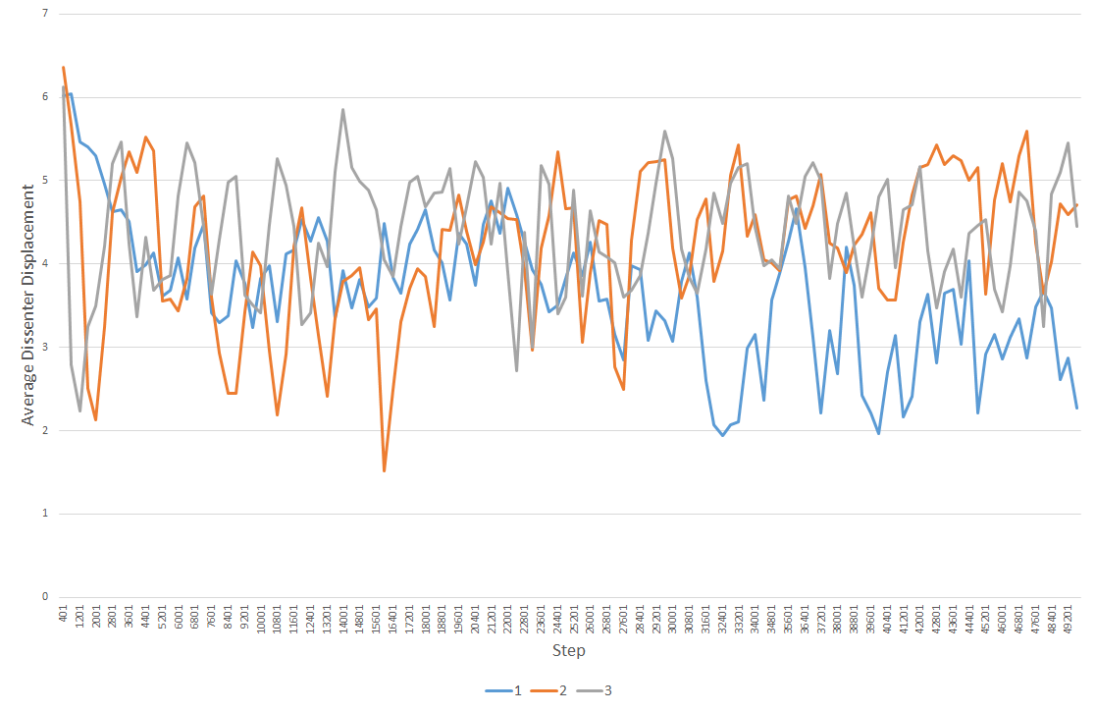
Dissenter Communication Range in 'Tight' Environments

Altering dissenter communication range in an environment with a doorway size of 1 has predictable results.



Increasing communication range has a drastic effect on how quickly dissenter agents are able to converge on the goal. This fits in with the current knowledge that dissenters in tight environments typically find, share and agree on specific routes to their goal. Increasing the agents' range has the obvious effect of ensuring that news of a viable pathway reaches others.

Dissenter Communication Range in 'Open' Environments



The effect of communication range on dissenter performance in an open environment (or lack thereof) further reinforces our knowledge about dissenter behaviour. As the structure opens up, 'routes' and sharing in general become less important and so the need for communication is reduced.

7.3 Example Scenario Experiment

This section will describe an experiment intended to replicate a hypothetical real-world application of the project model.

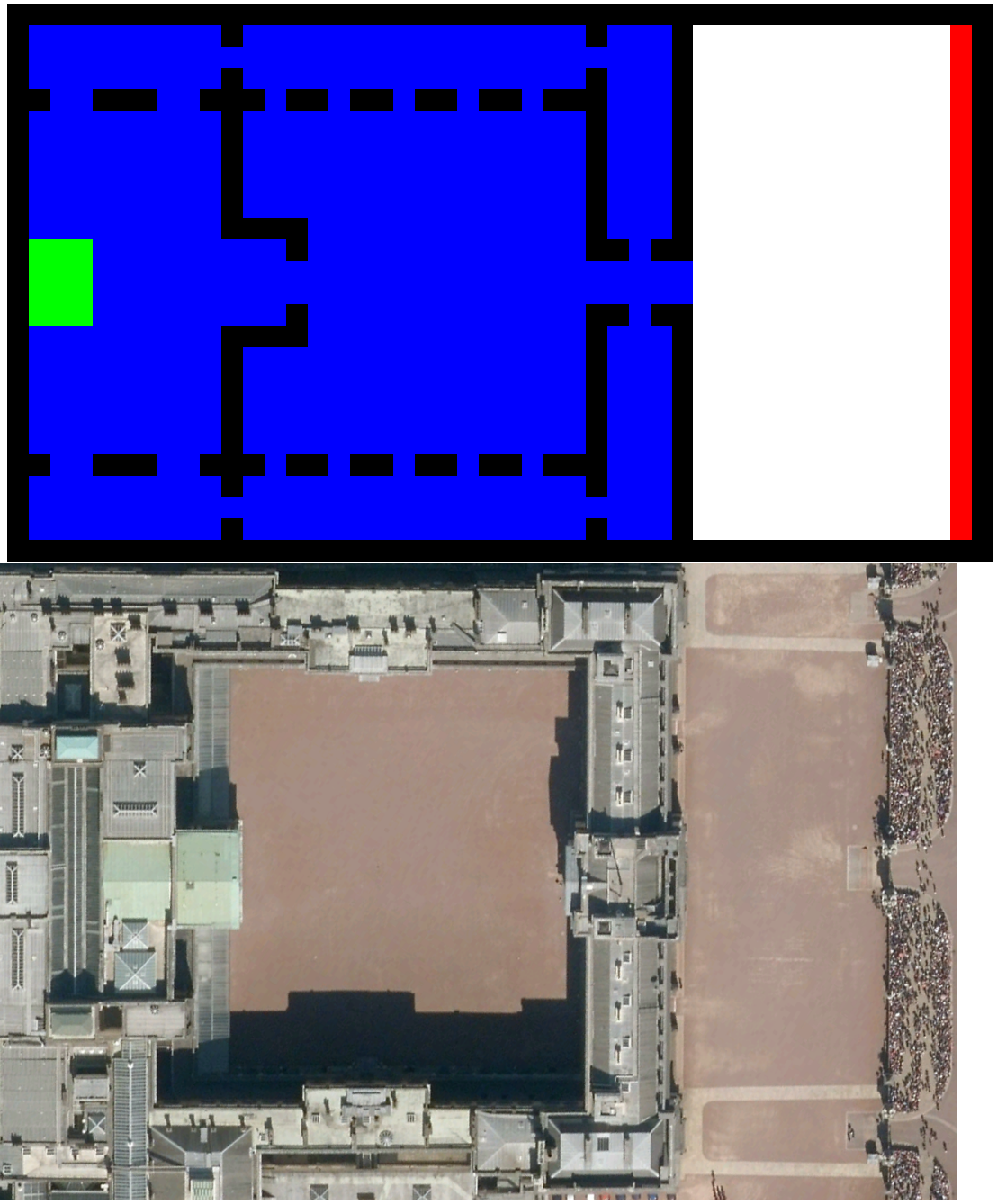
7.3.1 Hypothetical Scenario - Defence of Buckingham Palace

The UK government has received intelligence that warns of an imminent armed attack on Buckingham Palace. A reserve of 100 royal guards is available to defend the palace against an invasion force of 300 attackers. The goal of the guards is to delay the assault of the attackers for as long as possible to allow the royal family to escape by helicopter.

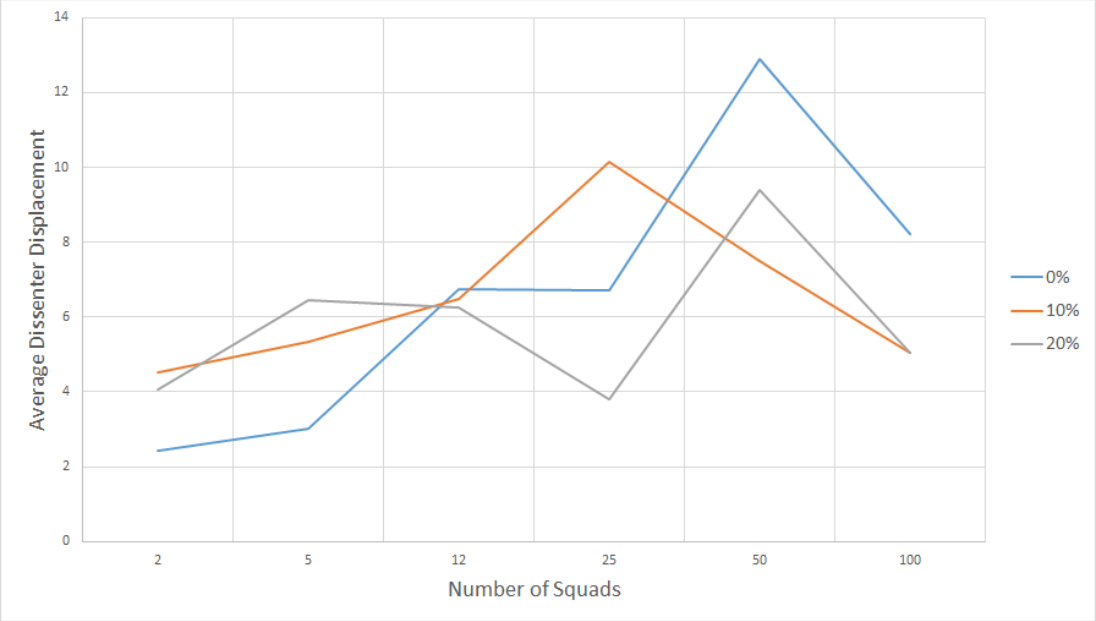
The number of available defending soldiers is fixed, but their commanders must decide on how to organise them: both the number of distinct squads to deploy and the proportion of the force that should be comprised of scouts.

7.3.2 Experiment

This experiment uses a custom Buckingham Palace map that recreates the general shape of the palace and the various entryways between parts of the building.



Each combination of values from the set 2, 5, 12, 25, 50, 100 for squad number and 0, 10, 20 for scout percentage were tested. Rather than display the average dissenter displacement over time like in previous experiments, this graph shows the average of this value over a whole run.



The graph shows that the optimal tactic in this instance is to use no scouts and 50 squads, meaning that enforcer agents operate in pairs. The generally poorer performance observed when utilising scouts may be due to the large size of the environment and relatively low density of agents to communicate information to. In such a large space, information shared by scouts is less often relevant to an agent than that shared in a tighter space. The large number of squads makes sense too, considering the complexity of the building and the large number of routes that could be taken by invaders. Having the enforcers behave as a group too much could cause them to bunch up too much and leave areas undefended.

It's interesting to note that the usefulness of scouts appears to increase when fewer squads are used. It would seem as though small squads act as their own scouts, since they don't follow the behaviour of a central crowd. When using fewer squads it's important to employ scouts to maintain a coverage of the entire patrol area, but this behaviour seems to arise naturally when using many squads.

7.4 Experiment Conclusion

The sample experiments performed in this section demonstrate just a small amount of the total functionality offered by the model. The results gathered are promising and suggest a greater level of depth that could be explored in future work.

Chapter 8

Evaluation

This section will evaluate the different aspects of this project and how they could be improved.

8.1 Software

The implementation of the model software itself is best evaluated by comparing it against the project requirements, which were split into several subsections.

8.1.1 Front End

The implementation of a suitable front end didn't end up being a major part of this project. After the decision was made to use Repast Simphony as a software platform there wasn't much that needed to be done to create a usable piece of software. Using the Repast wizards made the process of implementing graphical and graph views, as well as file streams and batch runs, very easy.

Using a pre-existing front end was definitely the right choice for this project, as it saved a significant amount of time that could be better used in development of the model itself.

8.1.2 Environment

The requirements for the model environment were not particularly demanding and were mostly fulfilled quite early in development. This was important considering how the structure of the environment underpins all agent behaviour. If the environment was not completed until later then any changes would have required a large amount of code to be rewritten. Thankfully, the choices made when designing the environment structure proved to be suitable throughout the

project and supported the sort of experimentation required from the final product well. The two-structure solution, creating the environment out of both a grid and a continuous space, ended up being the perfect choice for balancing realism and efficiency. After some optimisation in the later stages of development, the model came to be suitably efficient, so as to allow experiments utilising several hundred agents to be performed in a reasonable amount of time.

More could have been done to formally define the properties of the environment. Despite the fact that the model is intended to support environments which recreate real world scenarios, at no point were the in-model environment units equated with real world units of measure. If the model software were to be used to perform experiments in other fields of research this concrete definition would be highly desirable.

8.1.3 Agents

Agent behaviour was the central focus of this project, and so the majority of development time was spent adding to and building upon the basic functionality of the agents. The milestone-based development strategy proved to work well when developing the agents. After the basics of environment functionality and agent movement were introduced in the initial version, adding additional functionality was relatively easy, meaning that new ideas could be tested rapidly.

All of the desired behaviour laid out in the earlier chapters of the report was successfully implemented. The individual elements of agent behaviour are quite simple, but combining them together means that the final agents are quite complex, relative to the agents seen in the previous works cited in section. On one hand this is a good thing. Agent complexity allows for a large number of user-editable parameters for use in experimentation and also means that interesting and unexpected behaviour can occur within simulations. However, there are downsides to using more complex agents. Despite the fact that a large amount of time was spent throughout development carefully balancing the model parameters to create a good set of default values, it is possible that there is some set of values previously untested that could serve as a better basis for experimentation.

As with the environment, the agents are not defined in a way that concretely ties them to the real world. They do not have a 'size', with a physical form that is only implemented in abstraction by the model rather than being explicitly defined, i.e., agents move more slowly when in close proximity but only on a cell by cell basis. If this software were to be developed further, improving the 'physicality' of agents would be a high priority.

8.2 Experimentation

The experimentation chapter of this report demonstrates how the model software could be useful in research and real-world applications. The results of the experiments that were performed showed some promising results that demonstrated how altering agents and their environments could produce results that were interesting and could be explained intuitively. The Buckingham Palace example also gives an example of a theoretical real-world application of the model.

While the experimentation chapter does serve as an *example* of possible applications of the model, it is far from comprehensive. The model has far too many variables to perform a complete assessment of all of the functions available to users.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

This project relies on the intersection of several different fields of research and demonstrates just how much potential exists in each of them for future development.

The project aimed to model civil unrest in a more detailed way than had been done previously. In this way it has been quite successful, approaching the task of implementation from the perspective of a multi-agent system developer but treating the application of the software as a more traditional agent-based model.

The core of the project, the Q-learning algorithm, has shown itself to be a highly capable and flexible solution when developing agent learning. It adapted well to the needs of the project, combining static rewards, dynamic rewards and the Q-values of others to create a unified structure of learning in the agents.

9.2 Future Work

There are many improvements and additions that could be implemented in future version of the model software. Thanks to the module nature of the software, most additions would be reasonably easy to implement.

With regards to the environment, the most obvious improvement would be to add more cell types to increase the complexity and realism of the model. For example, areas could be designated as *partially obstructed* to impede, but not totally prevent, movement. If the mapping system was modified slightly, such that colour channel values were read as integer values rather

than as Booleans, this functionality could be extended further to allow each cell of the grid to have a *degree of obstruction*. Cells could also be designated as being not simply obstructed, but actively harmful to agents, to simulate obstructions like barbed wire or fire.

Another major improvement that could be made to the environment is the introduction of *dynamism*. Allowing the properties of the environment to change midway through the simulation would allow the recreation of many more real-world phenomena. For example, enforcer agents could 'construct barricades' by increasing the obstruction of a cell, and dissenters could destroy them, decreasing the obstruction. Cells could also become harmful dynamically to simulate a fire breaking out.

These additions to environment functionality are dependent somewhat on improvements to agent behaviour, most notably the addition of more actions that the agents can take. As the model software exists currently, the only action that agents can take is to move, which obviously doesn't cover the array of different actions a person could take in a real life civil unrest scenario. The first additional action to add would be some sort of 'attack' where agents could inflict damage on other agents or their environment. Later additions could be implemented to add any number of actions, such as the previously mentioned deployment of barricades. In order to make the attack actions and harmful environments meaningful, agents would need to be given a variable determining their 'health' or 'wellbeing'. This variable would be used to affect movement (weak agents move more slowly) and decisions (agents avoid choices which lead to loss of health, weak agents increase learning rate to change their situation).

A particularly interesting area of research that was considered early on in project development is the idea of simulating *emotion*. Humans, particularly when in violent or chaotic situations like that of civil unrest, do not act entirely rationally. They rely heavily on their emotions when making decisions, something that this project does not address. If realism was to be the primary goal of future work, then the addition of some sort of emotional simulation would be a very useful area to explore. There exist emotional simulations currently that vary greatly in complexity, from simple classifications of emotions such as the PAD model [13] (which has been used in human-interaction AI applications to improve realism [20]) to more complex solutions such as a more developed formalisation of the OCC model [1].

There are many more directions that this project could be taken in in future. The fields of agent-based modelling and machine learning are ripe with opportunity for development to achieve increasingly compelling results.

References

[1] Carole Adam. Occ's emotions: A formalization in a bdi logic. *Artificial Intelligence: Methodology, Systems, and Applications*, 2006.

[2] Thomas Booth. Modeling protest in lafayette square: An agent-based modeling approach.

[3] M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. *IJCAI'01 Proceedings of the 17th international joint conference on Artificial intelligence - Volume 2*, 2001.

[4] X. Chen. Grid world, 2006.

[5] S. Crompton. England riots: Why did youngsters who lack for nothing join the anarchy? *The Telegraph*, 2011.

[6] J. M. Epstein. Modeling civil violence: an agent-based computational approach. *Proceedings of the National Academy of Sciences of the United States of America, May 2002*, 2002.

[7] E. A. et al. Feinberg. *Handbook of Markov Decision Processes: Methods and Applications*. Kluwer, 2002.

[8] Department for Communities and Local Government. Government response to the riots, communities and victims panels final report, 2013.

[9] University of North Carolina GAMMA Research Group, 2014. http://gamma.cs.unc.edu/research/crowds/.

[10] M. Gardner. Mathematical games: The fantastic combinations of john conways new solitaire game "life". *Scientific American*, 1970.

[11] Hideaki Taguchi Akio Gofuku Kazuyuki Ito, Yoshiaki Imoto. A study of reinforcement learning with knowledge sharing -applications to real mobile robots-. *IEEE International Conference on Robotics and Biomimetics*, 2004.

[12] E. Mas, A. Suppasri, F. Imamura, and S. Koshimura. Agent-based simulation of the 2011 great east japan earthquake/tsunami evacuation: An integrated model of tsunami inundation and evacuation. *Journal of Natural Disaster Science 34 (1), 41-57*, 2012.

[13] A. Mehrabian. *Basic dimensions for a general psychological theory.* Oelgeschlager, Gunn & Hain, 1980.

[14] M. Nichols. U.n. chief calls for protection of rights in missouri protests. *Reuters*, 2014.

[15] D. O'Sullivan and M. Haklay. Agent-based models and individualism: is the world agent-based? *Environment and Planning A*, 2000.

[16] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987.

[17] Massive Software. Massive crowds in world war z, 2013. http://www.massivesoftware.com/wwz.html.

[18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[19] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

[20] S. Zhang, Z. Wu, H. M. Meng, and L. Cai. Facial expression synthesis using pad emotional parameters for a chinese expressive avatar. *Affective Computing and Intelligent Interaction, Second International Conference, ACII 2007 Proceedings*, 2007.