Exploring Inter-Agent Communication for the Emergence of Collective Intelligence in Team-Based Strategic Environments

Matthew A. Ford

Candidate Number: 291086

Supervised by:

Dr. Chris A. Johnson

Master of Science

Artificial Intelligence and Adaptive Systems

Department of Informatics

University of Sussex

Summer 2025

Statement of Originality and Intellectual property Rights

This report is submitted as part requirement for the degree of Artificial Intelligence and Adaptive Systems MSc at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged. I hereby give / withhold permission for a copy of this report to be loaned out to students in future years.

Signed: [illegible] Date: 19th August 2025

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Dr. Chris Johnson, for his guidance throughout this project, and the entirety of the taught course. It was his modules during the course that first inspired me to delve deeper into the world of AI and research, opening my mind to the limitless applications and research opportunities that AI can be harnessed for. His feedback, discussions, and constant encouragement throughout this project were instrumental to the completion of this work.

I would also like to extend my heartfelt thanks to my family for their endless support and encouragement. A special thank you to my parents, Emma and Les, whose patience during lengthy and seemingly nonsensical conversations worked often for helping me clear my thoughts, work through complex stages, and determine the next steps in my research. Finally, I am also deeply grateful for my grandad, Bill, and my grandma, Maureen, for their support and understanding, especially during the periods where this project demanded so much of my time and attention.

ABSTRACT

Collective intelligence in multi-agent systems, is an important challenge in artificial intelligence. Applying this problem to team-based strategic environments has proven to be computationally expensive and hard to achieve. Although some solutions have successfully demonstrated collective intelligence emergence, these rely on running the simulation on massive scale simulations, running in parallel. This report investigates a more sample-efficient and accelerated approach, exploring the role of direct inter-agent communication as an enhancement mechanism for encouraging the emergence of collective intelligence.

To create a suitable test for this, a Capture the Flag simulation was developed, where teams of agents compete and collaborate within a shared environment. Each agent's decision making is controlled by convolutional neural network, and a duelling deep Q-network, where the learning process is optimised by using a prioritised experience replat buffer. For population management, a genetic algorithm is used to evolve the agent's individual hyperparameter values and encourage team-based strategies over the generations.

The results show that this approach shows some signs of producing emergent, cooperative behaviours. Tournaments are conducted to show that the communication mechanism increases the average and consistency of agent performances. Furthermore, other analysis shows that the emergence of sophisticated tactics, such as escorting agent formations and protocols, were learnt with no explicit implementation, other than the reward functions provided as a standard of reinforcement learning. This report concludes that inter-agent communication and learning efficiency techniques was an effective methodology for accelerating the signs of emergent collective intelligence in these multi-agent systems.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

TABLE OF EQUATIONS

ACRONYMS

Abbreviation	Definition
AI	Artificial Intelligence
RL	Reinforcement Learning
ALE	Arcade Learning Environment
CI	Collective Intelligence
MAS	Multi-Agent System
ABM	Agent-Based Modelling
DQL	Deep Q-Learning
GA	Genetic Algorithm
MARL	Multi-Agent Reinforcement Learning
ML	Machine Learning
Deep RL	Deep Reinforcement Learning
DQN	Deep Q Network
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
EA	Evolutionary Algorithm
SI	Swarm Intelligence
CTF	Capture the Flag
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
TD	Temporal Difference
PBT	Population-Based Training
UAV	Unmanned Aerial Vehicle

1 | INTRODUCTION

Artificial Intelligence (AI) has undergone a metaphorical explosion in research and application in recent years. This trend is reinforced by increased investment and rapid performance increases in AI models *(Stanford University 2025)*. One prominent area of AI is Reinforcement Learning (RL), which is particularly popular for use in game environments as they serve as valuable test beds *(Hu, C., et al. 2024)* for the development of complex and sophisticated agents. The process of reinforcement learning allows for agents to learn complex behaviours through trial-and-error interaction with their environment and other agents. Unlike other traditional Machine Learning (ML) approaches, reinforcement learning is not reliant on pre-labelled data but instead uses a feedback signal, known as a reward, to guide the agent towards a desired goal or behaviour. This has led to human-level control in challenging game domains *(Mnih, V., et al. 2015*).

There are many well-known examples of reinforcement learning in games such as Chess or Go, an example of this being AlphaGo, created by DeepMind in 2017 *(Silver, D., et al. 2017)* that ultimately beat some of the best players of Go, and translated it's learning to play chess at superhuman levels without the need for parameter changes. *(Sejnowski T. J. 2018),* highlighting the potential of reinforcement learning but also outlining some significant challenges for developing these agents – the first being generalisation.

General competency in a variety of tasks and domains, without the need for specific tailoring, is a common goal of AI. To achieve this, platforms with different challenges and environments must be provided. The Arcade Learning Environment (ALE) is one such platform. *(Bellemare, M. G., et al. 2013).* This provided an interface to Atari 2600 game environments, each with unique challenges, allowing for a general model performance to be outlined. This demonstrated that having agents learn in a single environment prohibits their performance in those that may vary and therefore is bad practice. This issue is also referred to as overfitting. *(Whiteson, S., et al. 2011).*

The second major challenge is encouraging the emergence of Collective Intelligence (CI), where multiple agents interact with little to no centralized control *(Wolpert, D. H., Turner, K. 1999).* This problem is central to team-based Multi-Agent Reinforcement Learning (MARL), an extension to reinforcement learning where multiple agents learn and adapt within a shared environment. Due to the nature of MARL, the environment is inherently non-stationary for each individual agent, as its own teammates and opponents are constantly learning and adapting to their environment and each other simultaneously. This makes the process of learning effective individual policies, and cooperative behaviours a significant challenge.

This report addresses these challenges through the development of a multi-agent simulation of the game Capture the Flag (CTF). The CTF environment provides an ideal testing ground, requiring both strategic decisions, and a high level of cooperation to achieve the team-based objective. To create the agents with the intelligence to handle these tasks a hybrid approach is employed, consisting of Deep Reinforcement Learning (DRL) for decision-making, and biologically inspired Genetic Algorithms (GAs) to evolve team-wide and individual strategies and hyperparameter values over many generations. The primary objective of this research is to discover the extent that inter-agent communication can affect the emergence and effectiveness of collective intelligence in team-based strategic, and competitive environments.

The following chapter will provide a critical analysis of related research to the concepts introduced. First, discussing the concept of Agent Based Modelling (ABM), before exploring how biological observations have inspired systems used for creating sophisticated agents. Then, the key principles of reinforcement learning are examined, and the historical advancements made are discussed. Subsequently, the challenges posed by generalization and the role of environments in creating generalised agents, such as the ALE, is analysed to determine its role in addressing overfitting. Following this, collective intelligence id explored, specifically in relation to multi-agent reinforcement learning (MARL), to outline the key challenges and research opportunities this report aims to address. Finally, the problem of Capture the Flag (CTF) is outlined, and existing solutions are discussed to highlight the gap in research that this report targets.

2 | RELATED RESEARCH

This chapter provides a critical review of the research that relates to the major components that are the foundations of this report. It begins by discussing agent-based modelling and its use in studying emergence. Moving onwards, the biological systems that inspire genetic algorithms, and network architectures for managing agent populations, and processing observed inputs are compared. Following this, the principles of reinforcement learning are presented, along with its history with tabular methods, to the significant advancement with deep learning. Then, this chapter combines these concepts by exploring collective intelligence, looking into the challenges and opportunities that are created when working with Multi-Agent Reinforcement Learning (MARL) that also are the primary motivation for the research presented before discussing the solutions found in creating agents that perform collectively in the Capture the Flag environment.

2.1 | AGENT-BASED MODELLING

Agent-Based Modelling (ABM) is a computational method that allows for complex systems to be simulated, focusing on the individual actions and interactions of autonomous individuals, or agents. A typical agent-based model includes agents with individual attributes and behaviours, agent relationships and interaction methods, and an environment where said agents can interact *(Macal, C.M., & North, M.J., 2005).* ABM employs a bottom-up approach, where system-wide patterns emerge from smaller, local interactions *(Epstein, J. M., & Axtell, R., 1996).* The fundamental concept of ABM is that simple, individually based rules can lead to the emergence of complex patterns. This makes ABM useful for studying behaviours that are not explicitly programmed, but emerge from the interactions over time *(Bonabeau, E., 2002).*

The bottom-up approach is particularly powerful in exploring complex cooperative and competitive environments. Arryo-Kalin *(2017)* describes ABM as a “virtual laboratory” that allows for researchers to experiment on systems that cannot, or are harder to, replicate outside of a computational environment. This proves to be true for tactical simulations, where ABM can help to show the contribution of different agent strategies to the emergence of group-level outcomes *(Ilachinski, A., 2004).*

MESA is a powerful framework used for building, analysing, and visualising agent-based models in Python *(Masad, D. & Kazil, J., 2015)*, providing a modular and accessible platform for researchers and developers to simulate complex systems from the bottom up. Using this framework gives the user many functionalities for analysis, including recording, storing and exporting data from the model *(Antelmi, A., Caramante, P., et al., 2024)* and a wide range of visualisation tools, making it a popular choice for ABM.

While ABM provides the framework for simulating complex systems and interactions, it introduces the question of how the rules of the agents are designed to complete a task. These rules be virtually impossible to design, and therefore that's where machine learning and AI are employed to provide a solution. To create agents to both, adapt and learn the behaviours that complete the task in the most efficient way, it is common for researchers to turn to those methods inspired by the most advanced adaptive system – that system being nature.

2.2 | BIOLOGICALLY INSPIRED ALGORITHMS AND ARCHITECTURES

The primary concept of Artificial Neural Networks (ANNs) is to create computing models inspired by biological neural networks to solve problems in natural tasks *(Yegnanarayana, B., 2003).* The first mathematical concept of an artificial neuron was proposed by McCulloch and Pitts (1943), creating a simple unit that had an on or off state based on a threshold of inputs. This foundational idea of biologically inspired computing is most explicit in CNNs, whose architecture is inspired by the structure of visual cortices in mammals *(Hubel, D. H. & Weisel, T. N., 1962).* The research by Hubel and Weisel solidified the foundations of the modern CNN architecture and was later used to achieve practical success in document recognition *(Lecun, Y. et al. 1998).*

Visual experiences are based on information processed by the neural circuit in the eye. Once the set of features is created using the visual data, the output is then conveyed to the brain *(Kandel, E. R., et al. 2000).* This is due to the visual cortex not being responsible for decision-making tasks. In artificial neural networks, the role of the brain is performed by a fully connected network. Figure 1 illustrates the two-part process, showing the parallels between biological visual and mental decision-making processes with their computational counterparts.

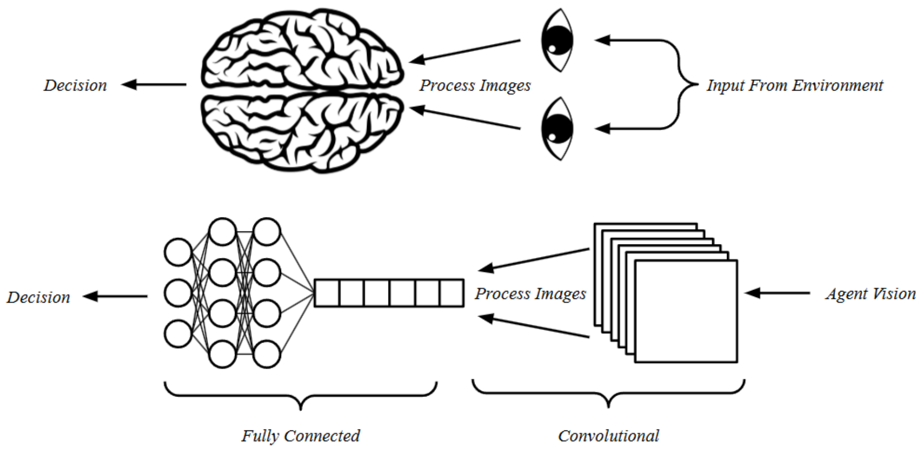


Figure 1*: A conceptual diagram illustrating the parallels between human biological vision and the architecture of a Deep Q-Network (DQN). The top diagram shows the human brain processing visual input to decide. The bottom diagram shows the corresponding network architecture.*

Genetic Algorithms (GAs) *(Holland, J.H. 1992)* are used in population dynamics with evolution. They are a form of Evolutionary Algorithms (EAs) which are heuristic optimisation algorithms, inspired by the biological principles of evolution *(Kramer, O., 2017).* The suitability of a solution is formulated using a fitness function. This assigns a score, judging by how well an individual solves a target problem. Those with a higher fitness score are therefore considered to be more adapted to the environment of the problem.

After fitness values are assigned, a selection process takes place to create the next 'generation' of solutions – referring to a complete team of agents, each with their own unique set of gene values. Most commonly, Darwin's theory of evolution *(1859)* is used – assigning a higher probability of selection to those with a greater fitness. New solutions are then generated using two primary genetic operators: crossover and mutation. *(Goldberg, D. E., 1989).* As Goldberg describes, crossover facilitates a "structured, though randomized, information exchange" *(p.9)* between parent solutions. Mutation is described as the "occasional random alteration of the value of a string position" *(p.11)* with the role of reintroducing genetic diversity within the population. These two methods both help traverse the fitness landscape with the potential to find a new optimal solution.

In terms of ABM, each individual agent has a set of values, known as a genome. This genome encodes the values that change the agent's strategy or behaviour. An agent's performance within the ABM environment directly determines the fitness, and by using the evolutionary methods of crossover, mutation and selection as discussed, an optimal strategy can be found. This approach, using a GA to evolve strategies for agents based on their performance, was most notably demonstrated in sim rating the Prisoner's Dilemma *(Axelrod, R., 1987).*

The main reason for using biological systems as inspiration is to emulate their most powerful characteristics. The brain has an ability to generalise, learning patterns and principles to adapt and react appropriately when new situations and challenges present themselves *(Lake, B. M., et al. 2017).* However, where artificial neural networks are involved, this ability does not emerge naturally but instead poses the critical challenge of ensuring an agent can perform well in varied and unseen environments *(Goodfellow, I. et al. 2016).* This is not only a report-specific problem, but a central problem in modern AI, unifying both neuroscience and AI research *(Hassabis, D., et al. 2017).* As discussed in chapter 1, the Arcade Learning Environment (ALE) *(Bellemare, M. G., et al. 2013)* was created to promote such characteristics.

2.3 | REINFORCMENT LEARNING

Reinforcement Learning (RL) is a general term in Machine Learning (ML) which represents the idea of learning through experience. The main concept of reinforcement learning is to create an environment where applied actions produce an immediate reward determined by the current state *(Jonothan H. Connell, et al., 1999).* The actions of the agents within the environment aim to choose actions that tend to increase the average reward. This is learnt over time through systematic trial and error with the guidance of algorithms *(Kaelbling, L. P., et al., 1996)*.

There are many algorithms that can be used for reinforcement learning, but one of the most prominent is Q-Learning *(Watkins, C., 1989)*. This method works by learning an action-value function, Q, which gives the expected utility of taking a given action, in a given state *(Watkins, C. & Dayan, P., 1992)*. This forms the basis of all reinforcement learning algorithms but has one limitation: Q is stored in a lookup table indexed by state and action. This is a plausible approach when there are few actions to be taken but quickly falls apart when the problems are more complex. In games, the environment is constantly changing and evolving. This means that the number of possible states becomes extremely large. This therefore makes it computationally impossible to store each state, and lookup said state each time an agent needs to decide on its next action. This problem is otherwise known as the 'curse of dimensionality' *(Bellman, R. 1966),* but there have since been solutions that have emerged.

The combination of deep neural networks and reinforcement learning, otherwise known as Deep Reinforcement Learning (Deep RL) *(Li, Y., 2017)* has become increasingly powerful as it enables agents to learn effective strategies directly through sensory inputs, such as images, or game state information. This led to DeepMind creating a network capable of human-level performance across a variety of Atari 2600 games using no prior knowledge, but instead using data gathered directly by the agent in its immediate environment *(Mnih, V., et al. 2015).* This specific agent worked using a deep Q network (DQN), which combines deep neural networks with the Q—learning algorithm previously discussed.

Convolutional Neural Networks (CNNs) play a vital role in processing visual information, such as map layouts, and location information, such as those seen in Atari games. The CNN architecture, outlined by Mnih et al. (2015), was designed specifically for this purpose, taking the raw pixel data from the in-game screen and using convolutional layers to identify relevant features.

While the CNN addresses the curse of dimensionality, the stability of the learning process is managed by another mechanism known as an experience replay. This concept was first introduced in early machine learning research *(Lin, L-J. 1992)* before becoming a benchmark of the DQN *framework (Mnih, V., et al. 2015).* Using an experience replay involves storing the agents' most recent experiences in a replay buffer. These experiences are randomly sampled during training to help break any temporal correlations that may emerge as a result of sequential data. Further research showed that prioritising more 'surprising' experiences to be replayed more frequently led to even greater efficiency *(Schaul, T. et al. 2015).*

To further increase stability, the DQN uses a second neural network, known as the fixed target network *(Mnih, V. et al., 2013; Mnih, V. et al., 2015)*. This network is a copy of the main network, but its weights are not updated every step. This provides a stable target for the Q-value update calculations rather than having to calculate these errors based on the networks own rapidly fluctuating estimates *(Arulkumaran, K. et al. 2017)* as this would be equivalent to "chasing a moving target", making the network highly unstable.

While these techniques create a robust agent, they were designed for a single agent in a static environment. This is something that becomes more complex when multiple agents are required, each making their own moves and decisions according to the current state of those that share the same environment and therefore creating a non-stationary environment.

2.4 | COLLECTIVE INTELLIGENCE

Collective intelligence refers to the emergence of intelligent group behaviours from the interactions of multiple agents in a system where the agents follow simple, local rules without any kind of central controller. This is usually identified when the capabilities of a group exceed that of the individuals whilst remaining decentralised. *(Wooley, A.W., et al., 2010; Malone, T.W. & Bernstein, M.S., 2015).* Collective intelligence is both a key subject in AI and social studies. Surowiecki was one of the first to display this concept, observed in human intelligence in his work *"The Wisdom of Crowds" (2004),* arguing that groups can exhibit a higher intelligence than the smartest individual, given the correct conditions. The conditions outlined were diversity of opinion *(ch.2),* independence of members *(ch.3)*, decentralisation *(ch.4)*, and aggregation *(p.xv).* Once met, groups can excel in the given task.

Swarm Intelligence (SI) is a form of collective intelligence that develops computational models inspired by the collective behaviour of natural swarms, such as colonies of ants *(Bonabeau, E., et al. 1999)*. The main concept driving swarm intelligence is that complex behaviours can emerge from simple, decentralised agents following local rules. While some swarm systems rely on indirect communication mediated by modifications of the environment, otherwise known as stigmergy *(Dorigo, M., et al. 2000),* many use direct local communication. An individual in the swarm interacts only with its close neighbours to share information about the local environment. Reynolds' 'Boids' algorithm *(1987)* is an example of complex flocking behaviour emerging from simple rules based only on the local neighbourhood. This can be applied to cause a cascade of local information exchange throughout an entire swarm, dependent on their positioning.

This swarm-based approach of emergent complex behaviours through simple, local interactions provides the theoretical foundation for achieving collective intelligence. A modern model for this is Multi-Agent Reinforcement Learning (MARL), which serves as an incubator for these behaviours to emerge through learning. In MARL, agents do not need pre-defined rules but instead discover their own optimal rules – also referred to as learning a policy – through trial and error *(Busoniu, L., et al. 2008)*. As each agent leans its own policy, there is potential for collective behaviour to emerge through communication and interaction, allowing the group to collaborate and complete a specified goal without any centralised control. Therefore, following the conditions discussed earlier, by Surowiecki *(2004)*.

Creating the necessary conditions for collective intelligence to successfully emerge is a significant challenge in MARL. The main concern being that multiple agents must learn simultaneously in a shared environment which, from the perspective of each agent, is non-stationary. This is due to other agents are learning and changing their policies. Furthermore, the complexity of MARL systems causes a scalability issue. As more agents are added into the environment, the joint state-action space increases exponentially, again falling subject to the curse of dimensionality *(Bellman, R. 1966; Hernandez-Leal, P. et al., 2017)*. Having such a large area to explore makes it computationally harder to learn effective and co-ordinated strategies to complete their task, hence technique such as action masking are employed to reduce the search space and increase the learning efficiency.

The challenges outlined in the previous sections help to highlight the requirement for a complex, dynamic environment to test and develop collective intelligence and MARL solutions. The game of Capture the Flag (CTF) serves as an ideal testbed for this purpose, providing a multi-agent environment with cooperation and competition to force complex, emergent strategies and collective intelligence under the correct conditions.

2.5 | SOLVING CAPTURE THE FLAG

The core premise behind Capture the Flag is simple: starting from your team's base, retrieve the enemies' flag before they retrieve yours *(Jacob, T., et al, 2023)*. This already provides a mixed cooperative-competitive environment, making it an exemplary test bed for training teams of reinforcement learning agents, along with having the potential to guide the emergence of collective intelligence. However, the challenge of creating such agents is a complex task, featuring a large action space and requiring a balance of long-term, and short-term goals. The environment itself is inherently non-stationary due to ever-adapting opponents and teammates, aligning with the core challenge of MARL.

The abstract layout of a typical play area is illustrated in Figure 2, showing that the flags and jails are in opposite corners to each other for each team.

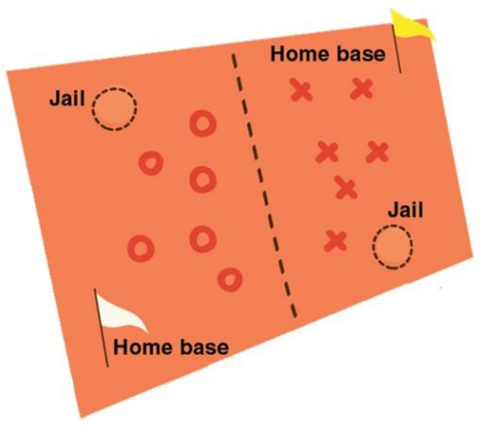


Figure 2: *By Swartz, C., An illustration of the basic setup for the game of Capture the Flag. Available at: scoutlife.org/hobbies-projects/funstuff/160235/how-to-play-capture-the-flag/*

An article from Swartz, C. outlines the key aspects of playing Capture the Flag, including:

- The goal of each team is to capture the other team's flag and return it to its zone.
- Players can be defensive and aggressive
- Players within their opposing team's territory can be tagged and sent to jail.
- A jailed player must be released by one of their teammates.

Similar experiments have recently seen agents performing at a human level in the Quake III Arena Capture the Flag team game mode *(Jaderberg, M., et al. 2019).* The approach is particularly relevant as the agents achieved a high level of cooperation and coordination without any direct communication between agents. Instead, collective intelligence was entirely emergent and learned over the course of 450,000 matches. Here the agents had only partial observations of their environment, being bound to a first-person perspective. Agent's actions were reinforced if they contributed to the success of the team, even if it was indirect. Through this, agents learnt to estimate their teammates' intentions from their actions in the previous game states.

To ensure the strategies were generalised, the agents in this experiment were trained using procedurally generated maps that changed every game. This was combined with a Population-Based Training (PBT) system that worked similarly to a genetic algorithm. While this research demonstrated the potential of purely emergent, non-communicative team-based strategies, it also required a massive number of computational resources as large populations were run in parallel.

There have also been hardware solutions using Unmanned Areial Vehicles (UAVs) in a centralised training, and decentralized execution framework. This system, created by Jacob, T. *(et al, 2023)* included a centralised commanding agent to efficiently analyse the global state, and make strategic decisions for the team. The other agents were then responsible for acting on these orders to complete the game. This structure allowed for complex, strategic computation to be handled separate from the agents themselves. Although, this work would not be able to exhibit any form of collective intelligence due to there being a centralised controller that decides the appropriate moves of all other agents, rather than each agent deciding individually to collectively learn new cooperative strategies that aid their team in completing the task.

To summarise, the research presents highlights a key gap for experimentation. While collective behaviours can emerge without inter-agent communication, it often requires extensive computational power. Centralized controllers, on the other hand, can be effective but do not allow for the study of emergent complex behaviours. This leads to the primary research question of this report: to explore whether direct inter-agent communication can provide a more efficient route to the complex, team-based strategies, without requiring a massive computational scale, whilst remaining emergent by staying decentralised. The following chapter provides a detailed description of the system implemented to investigate this question.

3 | METHODS

This chapter discusses the methods used for the implementation of the system, used to gather the results seen in chapter 4. The main components of the system are individually outlined, and the reasoning behind the choices made at each step are discussed.

The structure of this chapter starts with the agents and their environment along with the task they are challenged with and the rules of the game. Following this, the network architecture is presented and reasoned. Then, the reinforcement learning algorithm, deep Q-learning is explained, including its reward function to guide the desired behaviour, and the strengths and weaknesses with respect to the project. After, the techniques used to manage the agent population are presented, and the fitness function used for the genetic algorithm are explained, relating to individual success and the encouragement of collective intelligence. Finally, the simulation visualisation and runtime stability techniques are shown, along with the hyperparameters that could be used to change modes for research efficiency.

3.1 | AGENTS AND ENVIRONMENT

To investigate the emergence of collective intelligence, an Agent based model was created. The environment takes the form of a two-dimensional grid divided into two halves – one for each team of agents. The simulation is built using the MESA framework, one that is commonly employed for ABM in Python. MESA manages the grid, agent scheduling, and overall simulation state. The environment is populated by two teams of agents, static obstacle agents, and two flag agents (one for each respective team). The main layout of the standard environment created for this experiment is captured and annotated in Figure 3.

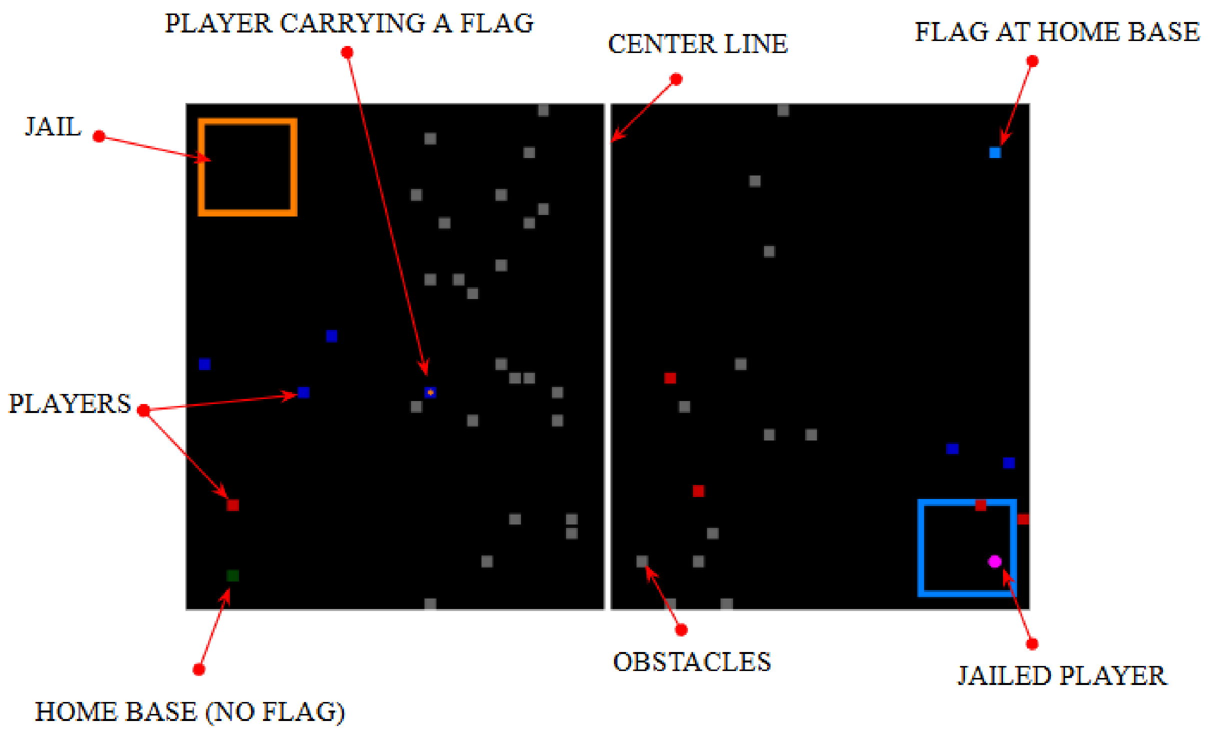


Figure 3: *An annotated capture from a standard simulation run, showing the agents playing in the set-out environment.*

Every player agent shares the same action space, outlined in Table 1, consisting of possible actions, and their respective IDs. While this action space defines the set of possible moves, the main challenge is selecting the optimal action at any given moment. To make these decisions, an agent must perceive its surroundings, gathering and sharing information about the positions of teammates, enemies, obstacles and flags. This is constructed as a 'mental map' of the entire environment, being used alongside a state representation that can be processed by its neutral network.

ID	ACTION
0	MOVE NORTH
1	MOVE NOTH-EAST
2	MOVE EAST
3	MOVE SOUTH-EAST
4	MOVE SOUTH
5	MOVE SOUTH-WEST
6	MOVE WEST
7	MOVE NORTH-WEST
8	STAY IN PLACE
9	TAG ENEMY
10	PICKUP FLAG
11	RELEASE TEAMMATES IN JAIL

Table 1: *The action space of any player agent, showing the action and respective ID.*

3.1.1 | AGENT PERCEPTION AND STATE REPRESENTATION

For each individual agent to decide an appropriate action to take, decisions are made based on two distinct inputs: a state vector of the current state of the overall game, and the agent's own local perception of the immediate environment alongside positional information shared by teammates. In essence, the state vector provides the strategic context for the tactical information from its perceived map, providing the full set of information needed, and enabling the agent to decide what to do given what it sees.

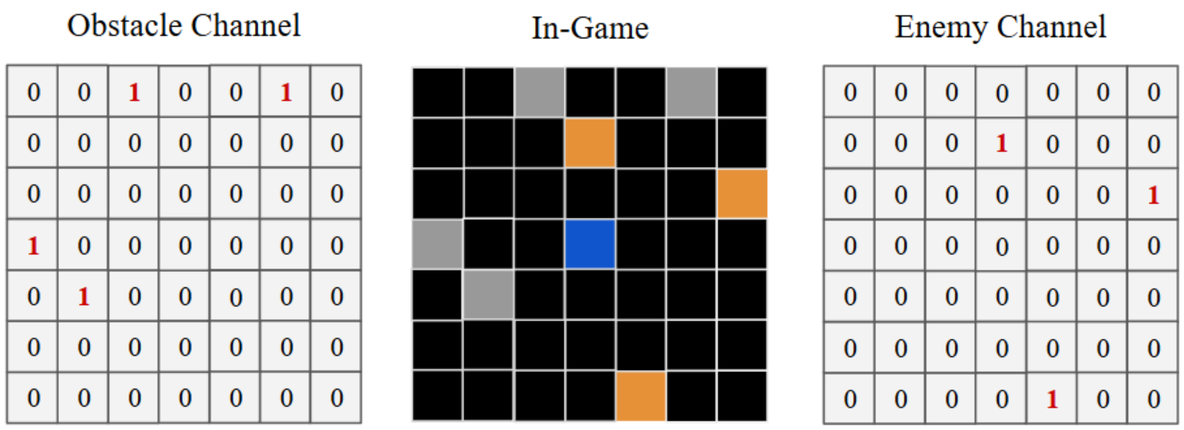
The complete representation of the agent's own "mental map" is captured as a multi-channel grid where each channel is dedicated to a separate feature (similar to RGB images); these are shown in Table 2.

CHANNEL INDEX	FEATURE
0	OBSTACLES
1	TEAM PLAYERS
2	ENEMY PLAYERS
3	TEAM FLAG
4	ENEMY FLAG
5	TEAM JAIL AREA
6	ENEMY JAIL AREA

Table 2*: Channel indexes alongside their respective feature represented when creating the local perception representation.*

To identify the locations of the specific features within each channel, all empty spaces are assigned a zero value, and those positions that contain the feature are marked with a 1. This is illustrated in Figure 4, which shows a local snippet of how obstacles and enemies are marked in their respective channels. However, it is important to note that the network processes a representation of the full map.

The dimensions of this full- map representation are (N, W, H), where N is the number of channels (7), W is the total width of the environment (40), and H is the total height (24). This binary, multi-channel representation is an ideal format for processing through the convolutional layers of the agent's neural network, enabling the agent to react to spatial patterns and relationships.



Obstacle Channel

0	0	1	0	0	1	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Enemy Channel

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	1	0	0

Figure 4*: An illustration of an agent's local perception representation, showing the binary obstacle (left) and enemy (right) channel grids, reflecting the in-game example (middle).*

To gather the data for the perception, and to outline the directions available for map traversal, the agents use a Moore neighbourhood *(ter Hoeven, E., et al., 2025).* As shown in Figure 5, this allows for the agent to gather a larger area of data and allows for movement in any of the eight adjacent cells. Having a wider range of motion provides greater flexibility than a restrictive four-dimensional Von-Neumann neighbourhood *(ter Hoeven, E., et al., 2025).* In addition to movement, any other action in Table 2 can be performed within this immediate Moore neighbourhood.

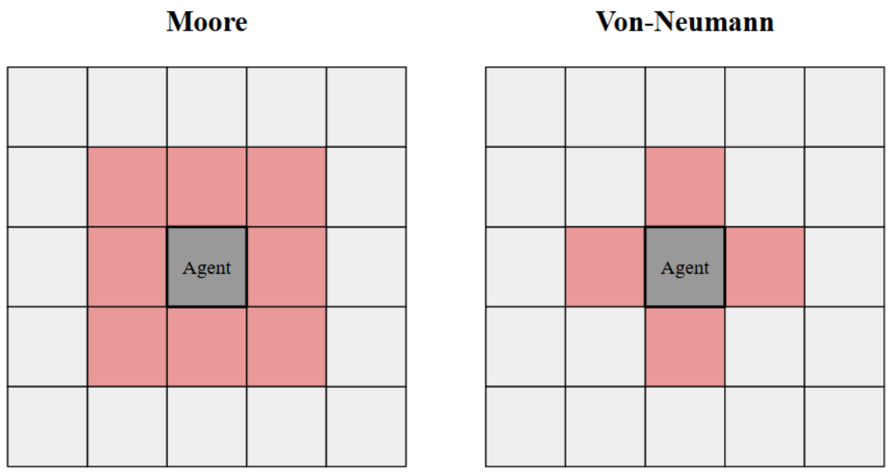


Figure 5: *An illustration of a Moore neighbourhood compared to a Von-Neumann neighbourhood. Each with a size of 1.*

To complement the spatial information gathered from the local perception, each agent is also provided with a vector space representation of the overall game state. This vector has a fixed size of 6 and contains the critical, high-level context that isn't available from the local perception. The contents of the vector are shown in Table 3, along with a description of what each value represents.

DATA	DESCRIPTION
AGENT FLAG STATUS	A binary value indicating if the agent is carrying a flag.
AGENT JAILED STATUS	A binary value indicating if the agent is currently jailed.
OWN FLAG AT HOME	A binary value indicating if the team's flag is in its home position.
ENEMY FLAG AT HOME	A binary value indicating if the enemy's flag is in its home position.
JAILED TEAMMATES	An integer value of currently jailed teammates.
JAILED ENEMIES	An integer value of currently jailed enemies.

***Table 3**: The contents of the vector space representation, determined by the current game state, and their respective descriptions.*

This state vector is ultimately concatenated with the processed and flattened features from the agent's "mental map" grid, creating a complete state representation that is fed into the fully connected neural network layers, allowing the agent to determine the most appropriate action.

While the full state representation and action space provide a single agent with the ability to make intelligent and well-informed decisions, success in Capture the Flag is dependent on teamwork. An agent operating completely independently is still flawed as their knowledge is massively restricted to its own sight and the high-level information given by the game state vector. To exceed these individual limitations and enable true teamwork, a system of cross-agent communication is implemented.

3.1.2 | CROSS-AGENT COMMUNICATION

Having a means of communication between agent teams is key for collective intelligence in this experiment. To enable this, the system implements a local, timestamped messaging function that allows for tactical intelligence to be shared.

As an agent finds an obstacle or observes critical events such as an enemy, or a flag away from its home base, a message is generated containing all the information needed for other agents to respond. This information includes the event type, location, and the current timestep to show when it was observed. This message is then broadcast, but only to those teammates that are within a fixed communication distance of 3 units. This range is intentionally less than the spotting distance of 5, compelling agents to maintain closer formations to ensure effective information sharing. This process of an agent sighting an enemy, gathering the respective information, and broadcasting it to a nearby teammate is illustrated in Figure 6.

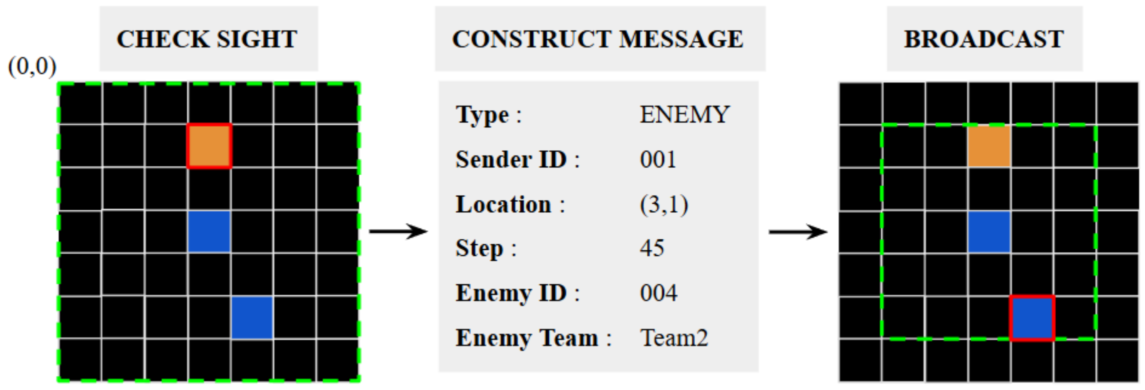


***Figure 6**: An illustration of the agent messaging system for "spotting" an enemy. The process begins with a sender agent observing its surroundings and identifying an enemy (left). Then, a data message containing the event type, location, and current timestep is.*

Each agent maintains and internal knowledge base. This stores the information along with the timestep that each entry was directly observed. Upon receiving a message, an agent updates its knowledge but crucially ensuring the data is newer than their current entries of the same nature. Furthermore, to prevent agents from acting on outdated enemy positions, information regarding enemy locations automatically expires, and is deleted from the agent's knowledge if it isn't updated within 30 timesteps. This ensures the team is acting upon the most up-to-date and relevant information as the observes and received information is used to construct the mental map for the convolutional layers of the network. Additionally, agents that are carrying a flag are always self-aware. In this situation, the agent will continuously update their own internal knowledge base with its current position on every step to guarantee the flag location is always accurate. Any agents that are out-of-range from the flag bearer, and any teammates will maintain the last location that theirs, or their enemies' flag was seen.

The challenge of processing this large dataset to make intelligent decisions is the role of the agent's Deep Q-Network (DQN), whose architecture is specifically tailored to interpret this sensory input and translate it into a single, optimal action. The following section details the architecture and discusses the decisions made.

3.2 | NETWORK ARCHITECTURE

The agent's ability to make appropriate decisions throughout a match is guided by a hybrid network, using a convolutional, and duelling deep-Q network *(Wang, Z. et al., 2016)*. This is illustrated at a high-level in Figure 7. This architecture was chosen to handle the complexity of the input that consists of the visual data, and the global state vector (discussed in Chapter 3.1.1). The visual perception data is processed through a series of convolutional layers, which notably include a skip connection block to improve gradient flow and an adaptive max pooling layer to standardise the feature size. The output from the convolutional layers is then concatenated with the game state vector and passed to the fully connected network. The output from this final architecture are the final action values, each representing how appropriate an action is at the current time.

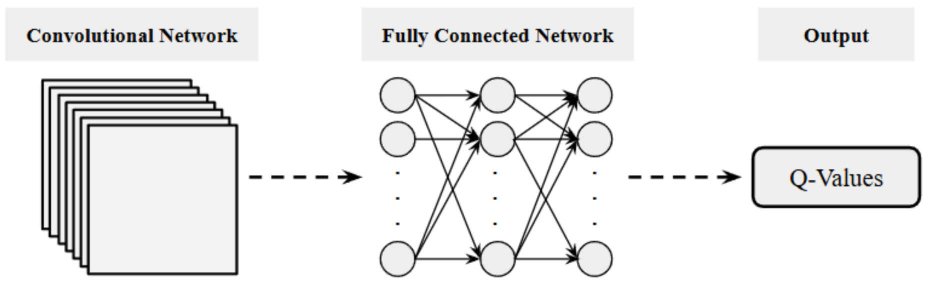


Figure 7: *A high-level diagram of the agent's network architecture. Consisting of a convolutional network, fully connected network, and an output for the Q-values.*

A key component that is consistently used throughout the architecture is the Rectified Linear Unit (ReLU) activation function *(Krizhevsky, A., et al., 2010)*, which is applied after most convolutional and linear layers. The equation (1) that defines this function is:

$$f(x) = \max(0, x) \tag{1}$$

The purpose of ReLU is to replace all negative values in the current tensor with zeroes, whilst maintaining any positive values. This function is standard for most deep learning applications due to its ability to mitigate the 'vanishing gradient problem'. This problem is related to the backpropagation of the network where gradients become extremely small as they are passed backwards through the layers.

As the gradient travels through many layers of deep networks, this value can become so small that it effectively vanishes and causing the initial layers of the network to never be updated. As a result, the training process can be hindered, preventing convergence on an optimal solution.

As illustrated in Figure 8, the ReLU activation function's key difference is its derivative. For any positive input, the function's gradient is a constant value of 1. This directly benefits the training process as the error signal is backpropagated through the network's layers, ReLU does not diminish the value. This flow of the gradient signal mitigates the vanishing gradient problem that is common in deep networks using sigmoid or tanh functions.

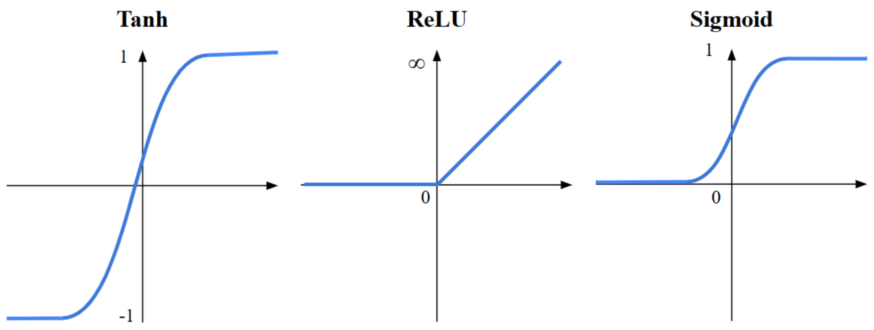


Figure 8: *Graphs of the Tanh (left), ReLU (middle) and Sigmoid (right) activation functions. Each illustrating how a neurons input (x-axis) is transformed to its output (y-axis).*

3.2.1 | CONVOLUTIONAL LAYERS

The function of the convolutional layers within the network is to process the raw, grid-based map representations constructed using the agents own stored information to create a high-level feature vector. This section of the architecture is illustrated in Figure 9.

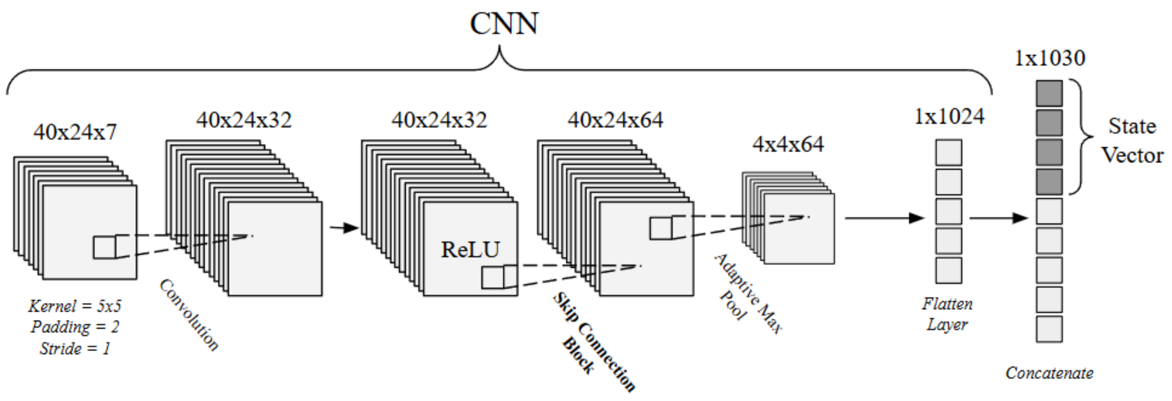


Figure 9*: the architecture of the convolutional section of the agent's network. Processing the raw perceived map layout data into a flattened vector before concatenating with the game state vector.*

The input to this architecture is a tensor of size 40x24x7 due to the chosen map size being 40x24, and there being 7 channels for each of the unique features in Table 2. This input is processed by a 2-dimensional convolutional layer with a 5x5 kernel, transforming the 7-channel input into a 32-channel feature map. After a ReLU activation to negate any negative values and increase the sparsity, this feature data is then passed through a skip connection block (otherwise known as a residual block) – illustrated in Figure 10.

This skip connection is designed to enable the training of deeper networks whilst maintaining stability as it improves the flow of gradient during backpropagation. This is achieved using two paths for the data to flow through, each running in parallel.

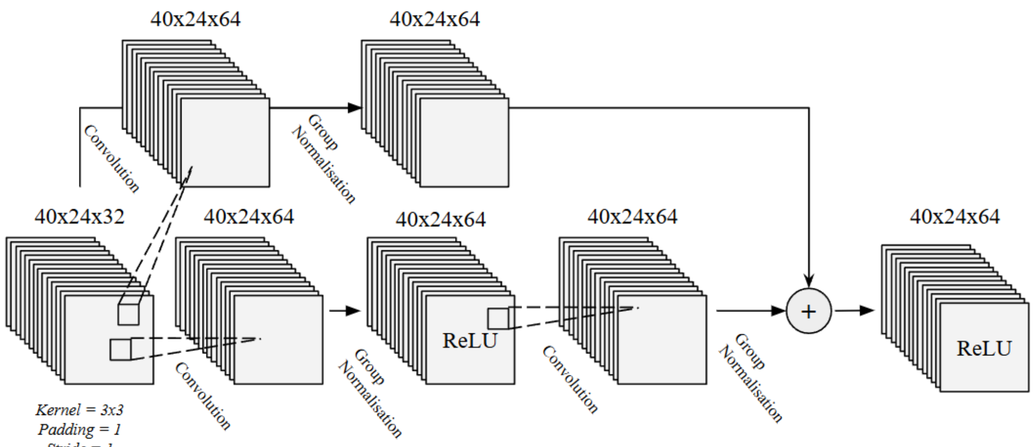


Figure 10: *The architecture of the skip connection block. Processing the data in parallel using one main path (bottom), and a shortcut (top) before adding the results and applying ReLU.*

The main path processes the data to a higher extent, first performing a 3x3 convolution to expand the feature depth from 32 to 64, followed by a group normalisation layer and ReLU activation. This feature data is then passed through a second 3x3 convolution and another group normalisation layer. The purpose of group normalisation is to help stabilise the training of agents by ensuring that the activation distribution is consistent.

While the ReLU function introduces non-linearity, it does not act as a normaliser for the overall distribution of values that it receives due to its constant gradient of 1. The purpose of the normalization layer is to stabilise this distribution before the activation is applied by normalising the features by the mean and variance computed within a mini batch, leading to a stable and efficient training process and benefiting generalisation *(Wu, Y. & he, K., 2018)*. For this purpose, group normalization was deemed as the most appropriate choice in the context of the given architecture, due to its independency in relation to the batch size, making it much more robust for a dynamic reinforcement learning environment.

The shortcut path, in parallel, processes the original tensor, allowing information to bypass the main processing layers. In the case that the spatial dimensions or the channel number changes between the input and output to this skip connection block, this path uses a 1x1 convolutional layer to ensure the dimensions are equal for next step. The outputs from the two paths are then added together elementwise before a final ReLU activation is applied to produce the block's final output.

Following the skip connection, the produced data goes through a final processing stage. First, the 40x24x64 feature map is passed through an adaptive max pooling layer. This layer calculates the necessary stride and kernel size to achieve a desired output size, providing much more flexibility than regular max pooling layers. By representing local regions of the feature map in terms of their maximum values, it down-samples the spatial dimensions to a fixed size of 4x4, resulting in a 4x4x64 tensor. Finally, to put the data into the correct format for the fully connected stage, this tensor is flattened into a one-dimensional vector consisting of 1024 features

Between the two network architectures, the 1x6 game state vector is concatenated with the 1x1024 feature vector outputted by the CNN, forming the complete input for the fully connected network layers.

3.2.2 | FULLY CONNECTED LAYERS

The fully connected network layers use the information in the vector created from the CNN output alongside the state vector to approximate the action-value function. For this purpose, a Duelling Deep Q-Network (Duelling DQN) is implemented. This architecture consists of two streams – value and advantage – and helps in learning state values more efficiently and robustly which is especially important as the input vector is large. The architecture for this section can be seen in Figure 11.

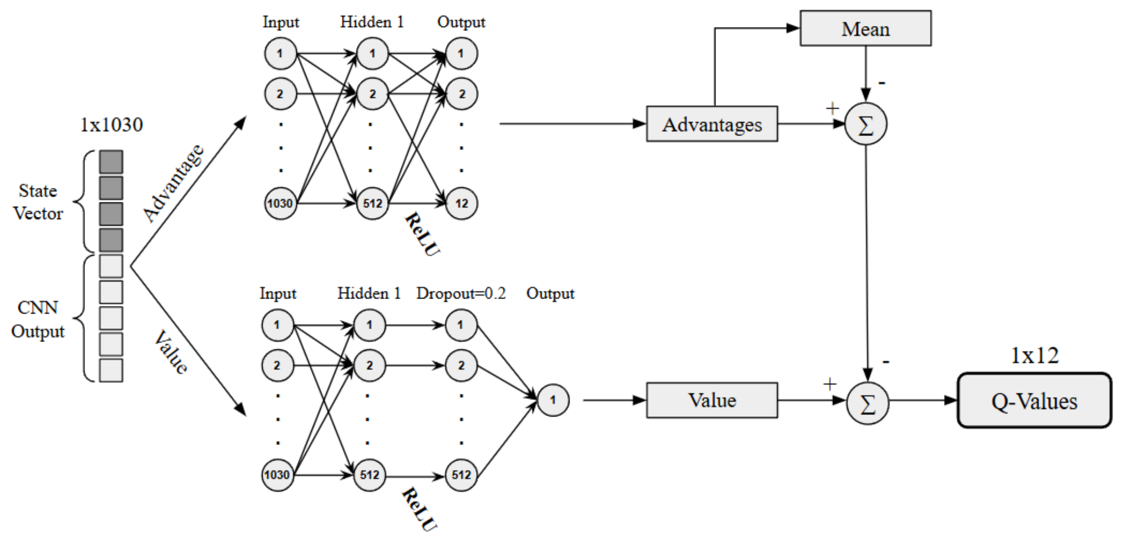


Figure 11: *The architecture of the fully connected layers. Showing a duelling architecture, and aggregation layer to produce the final Q-values.*

The output to the value stream produces a single scalar estimate of the state value, $V(s)$. This value is representative of the benefit of being in the current state before any action is taken. This stream is constructed of a linear layer that maps the 1030-feature inputs to 512 features, followed by a ReLU activation. A dropout layer, with a rate of 0.2, is then applied to prevent overfitting by randomly disabling a fraction of neurons and their connections. Then, a final layer maps the 512 features to a single scalar value, $V(s)$.

The advantage stream, in parallel, calculates the advantage, $A(s, a)$, for each possible action. This represents the benefit of a specific action, a, is compared to the other available actions in the current state, s. This stream also begins with a linear layer, to map the 1030-feature input to 512 features with a ReLU activation. However, there is no dropout in this stream, and the final linear layer maps the 512 features to a vector of size 12, equivalent to the size of the action space, and produces the advantage for each of these actions.

Once the outputs are produced by each of the streams, both undergo a final aggregation layer to produce the Q-values. To strengthen the stability of the learning process, and to correct the identifiability problem, the mean of the advantages are subtracted from each individual action's advantage. Then the final Q-values are calculated with the equation (2):

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|A|} \sum_{a' \in A} A(s, a') \right) \tag{2}$$

The identifiability problem is where the individual state, $V(s)$, and action-advantage, $A(s,a)$, cannot be uniquely recovered from the final Q-value, $Q(s,a)$. *(Wang, Z., et al. 2016)* This arises if the two streams are combined using the naïve summation shown in equation (3):

$$Q(s,a) = V(s) + A(s,a) \tag{3}$$

Therefore, using the equation (2) ensures that the value stream learns the true value of the state, while the advantage stream only learns the relative preference of each action, leading to more effective training.

The final aggregation of these streams produces the vector of Q-values, each element representing the predicted suitability of a specific action from the agent's action space. From here, an action mask is created and used to specify what moves are legal for the agent to perform in the given state, taking the form of a binary vector.

With the Q-values calculated, the mask is then used to set any illegal actions to an extremely small value, essentially making them impossible to be selected. For instance, as shown in Figure 12, if an agent is not near an enemy, then tagging would not be a legal move. This system helps to improve the efficiency of learning by limiting the waste steps needed if an agent was to learn these actions without the mask through trial and error. Also, as the Q-values are only estimations, the action masking helps to avoid any mistaken Q-value assignments that may be a product of a policy error.

After masking, the agent selects its next move by choosing the action that has the highest corresponding Q-value in the remaining vector.

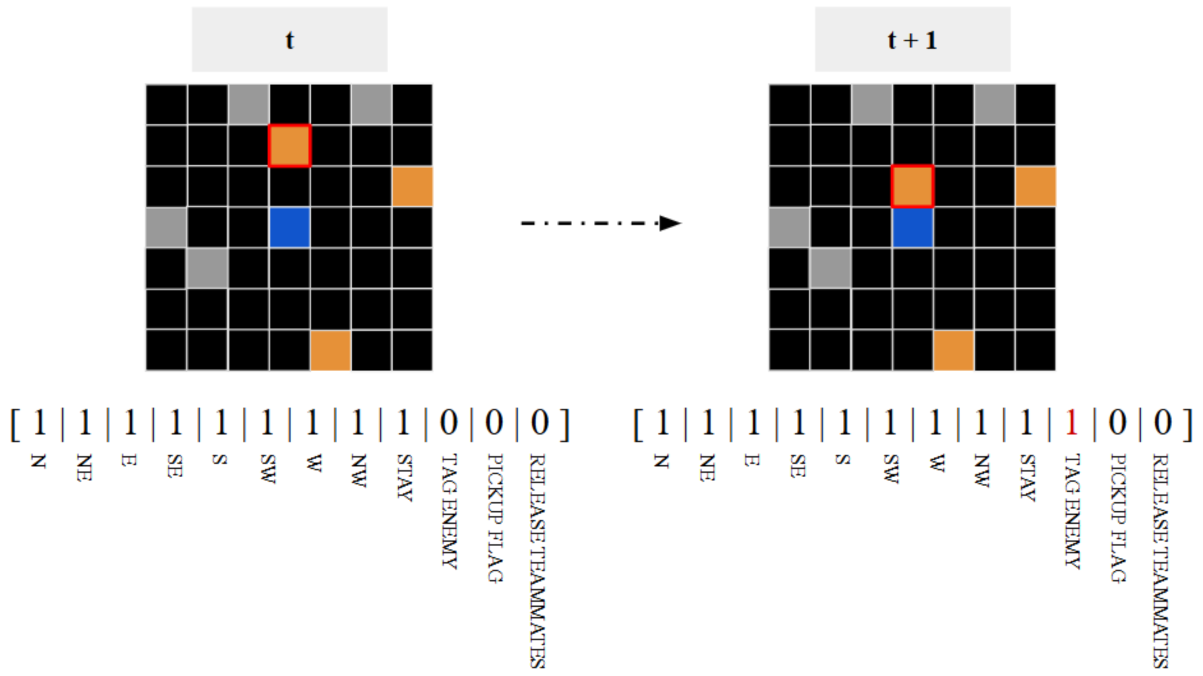


Figure 12: *An illustration of the action mask in the context of tagging enemies. If the player is not next to an enemy, then the "TAG ENEMY" action is 0, but when the enemy is closer, the action is 1.*

3.3 | REINFORCMENT LEARNING

Reinforcement learning allows each agent to operate and learn within the complex environment of Capture the Flag. To create a system that works independently of hard-coded behaviours, the network architecture discussed in Section 3.2 must be used alongside the deep Q-learning algorithm to create a customised policy. This policy is guided by a reward function, incentivising the agent to perform specific actions in any given state. For this, a prioritised experience replay is used to enable more efficient learning by filtering information that is considered not useful, to focus more on the experiences that warranted a higher reward. Together, through the agent setup, network architecture, and reinforcement learning components, a framework is developed for developing sophisticated and adaptive strategies.

3.3.1 | LEARNING ALGORITHM

The algorithm at the core of the agent's learning process in Deep Q-Learning (DQL). As discussed in Chapter 1, DQL was first introduced by DeepMind in 2015, representing a landmark advancement in reinforcement learning AI, combining deep neural networks and traditional Q-learning, with the primary objective of is to learn an optimal policy to help dictate the best next action to take in any given state.

Unlike traditional Q-learning that relies on predetermined tables of state-action pairs, DQL uses a deep neural network to approximate the action value function, $Q(s,a)$. This approach allows the agents to generalize using past and similar experiences to handle high-dimensional and complex state spaces that are inherent from environments such as Capture the Flag. The learning process is driven by minimizing the Mean Squared Error (MSE) between the network's predicted Q-values and the target Q-value. This loss function is derived from the Bellman equation *(1966)*, as shown in Equation (4):

$$L(\theta) = \mathrm{E}\left[\left(r + \gamma \max Q(s', a'; \theta^{-}) - Q(s, a; \theta)\right)^2\right] \tag{4}$$

where:

$L(\theta)$	is the loss
r	is the immediate reward
γ	is the discount factor
$max\, Q(s', a'; \theta^{-})$	is the maximum achievable Q-value
$r + \gamma\, max\, Q(s', a'; \theta^{-})$	is the target Q-value
$Q(s, a; \theta)$	is the predicted Q-value

In the context of the system implemented, the discount factor, γ, is not a fixed variable for each agent, but instead is one of the many genes that are optimised using the Genetic Algorithm (GA), discussed in Section 3.4, with values ranging from 0.999 to 0.995. This allows agents to either value future rewards similarly to the immediate ones (far-sighted), or to prioritise immediate rewards, making quick solutions more desirable (short-sighted). The reward, r, is generated at each step, carefully shaped to guide the agent's behaviour for better performance in the specific environment, and the learning step approximates the expected value, E, by sampling a small batch of experiences, stored in the Prioritized Experience Replay (PER) that weights experiences based upon their individual informativity.

3.3.2 | REWARD FUNCTION

To guide the agents towards complex individual and cooperative behaviours, required for completing the task of Capture the Flag, a multi-part reward function was implemented. The total reward an agent receives after each step is an accumulative total of general incentives, state dependent rewards, and a genetically influenced combination of attacking and defensive actions.

Firstly, a set of general rewards and penalties are applied to encourage the core behaviour and discourage actions that may reflect negatively on the outcome of the game. These include smaller incentives to encourage exploration, competition, and cooperation alongside penalties for repeated movements / actions, inactivity, or being jailed. These fundamental rewards are outlined in Table 4.

ACTION TAKEN	REWARD VALUE
AGENT IN JAIL	- 0.05 → RETURN
AGENT REVISITED RECENT POSITION	- 0.25
IN ENEMY TERRITORY	+ 0.05
TEAMMATES JAILED	- TEAMMATES IN JAIL / TEAM SIZE
TAGGED ENEMY	+ 10.0
NEAR ENEMY TAGGING	+ 5.0
STOLEN FLAG FROM ENEMY	+ 25.0
RESCUED TEAMMATES	+ 10 * TEAMMATES IN JAIL
PICKUP FLAG	+ 20.0
STAY	- 0.1
NEW POSITION VISITED	+ 0.01

Table 4: *The contents of the vector space representation, determined by the current game state, and their respective descriptions.*

Beside the general rewards, the agent's primary behaviour is controlled by a state-dependent function, capable of shifting their priorities based on the game's context in the current state. This reward function is shaped to promote complex, goal-oriented actions that reward agents for recovering their flag, escorting a flag-bearing teammate, or chasing a flag-bearing enemy. These state-dependent rewards are shown in Table 5.

GAME STATE	ACTION TAKEN	REWARD VALUE
AGENT HAS FLAG	MOVE TOWARDS HOME	+ 5.0
TEAMMATE HAS FLAG	STAY IN PROXIMITY	+ 2.0
ENEMY HAS FLAG	MOVE TOWARDS ENEMY	+ 3.0
TEAM IN DESPERATION	MOVE TOWARDS JAIL	+ 3.0
NO FLAGS OBTAINED	DEFENSIVE	+ DEFENSIVE REWARD
NO FLAGS OBTAINED	ATTACKING	+ ATTACKING REWARD

Table 5: *The state-dependent reward function, showing the rewards given in specific scenarios to encourage a shift in gameplay.*

In the case where both flags are in their home positions, there is a second layer of agent incentives that consists of a blend of two separate reward functions. The first function rewards offensive behaviours, and the other that rewards defensive behaviours. The final reward is calculated as a weighted sum of the results of both functions determined by the agent's "aggressiveness" gene that is genetically evolved, allowing the simulation to explore the trade-off between offensive and defensive strategies.

$$r_{blended} = \left(r_{offensive} * aggressiveness\right) + \left(r_{defensive} * (1 - aggressiveness)\right) \tag{5}$$

The specific actions rewarded by the offensive and defensive functions are outlined in Table 6 and Table 7, respectively.

ACTION TAKEN	REWARD VALUE
MOVING TOWARDS ENEMY HOME	+ 1.0
MOVING TOWARDS ENEMY	+ 1.2
ESCORTING A TEAMMATE WITH FLAG	+ 1.5
ESCORTING MOVE TOWARDS HOME	+ 1.5

Table 6: *The offensive reward function used to reward more aggressive agents for performing actions that move towards capturing the enemy's flag.*

ACTION TAKEN	REWARD VALUE
MOVING TO OWN FLAG ON FIELD	+ 3.0
GUARDING FLAG AT HOME BASE	+ 1.5
CHASING ENEMY IN TERRITORY	+ 3.0
MOVING BETWEEN ENEMY AND FLAG	+ 1.5

Table 7: *The defensive reward function used to reward less aggressive agents for performing actions which stop the enemies from capturing the flag.*

These rewards are designed to both guide the individual agents and create an environment where complex and emergent cooperative strategies can be learned and evolved throughout the training that spans many generations.

3.3.3 | PRIORITISED EXPERIENCE REPLAY

To improve the efficiency of the agent's learning process, a Prioritised Experience Replay (PER) *(Schaul, T., et al. 2015)* is implemented. This ensures that the agent's network is focused to a greater extent on those experiences that provide the most information.

An experience that is inputted into the replay takes the following format:

(state, action, reward, next state, game over)

A standard experience replay buffer stores these experiences and samples them uniformly at random. This provides the benefit of breaking any temporal correlations by treating all experiences equally. However, some experiences are much rarer than others and yield much greater rewards, providing an important learning opportunity than other commonly seen experiences. This is the disadvantage of a standard replay buffer, causing inefficiency whilst wasting training cycles on redundant information.

PER addresses this issue by replaying important experiences more frequently. The importance of any individual experience is determined by its Temporal Difference (TD) error, $|\delta|$, calculated using the Equation (6):

$$|\delta| = |r + \gamma \max Q(s', a') - Q(s, a)| \tag{6}$$

where:

• r	is the immediate reward
• γ	is the discount factor
• $max\, Q(s', a'; \theta^-)$	is the maximum achievable Q-value
• $r + \gamma\, max\, Q(s', a')$	is the target Q-value
• $Q(s, a)$	is the predicted Q-value

Experiences which result in a higher TD error have a higher priority and therefore are weighted to be more likely chosen for replaying. This priority value is calculated using the formula:

$$p = (|\delta| + \epsilon)^{\alpha} \tag{7}$$

where:

- $|\delta|$ is the TD error
- ϵ is a small constant to ensure non-zero priorities
- α is the degree of prioritisation

To implement the weighted sampling a sum tree data structure is used. As illustrated in Figure 13, the value of a node in a sum tree is equivalent to the sum it's direct children, meaning that the root node holds the total value of every leaf node.

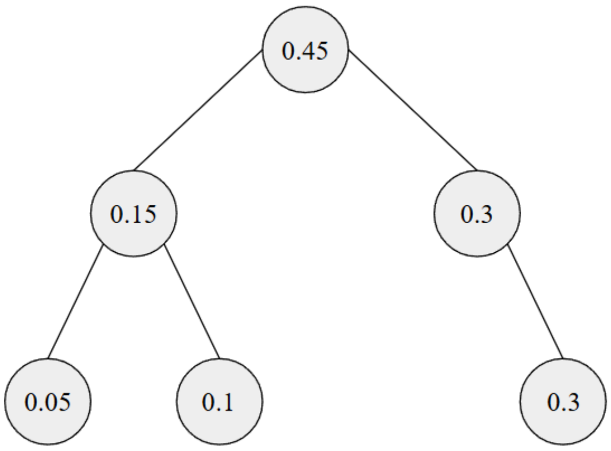


Figure 13: *An example illustration of a sum tree: A binary tree where the sum of the nodes in the left and right subtrees equal the value of any node.*

To sample an experience, a random value between 0 and the total sum is generated, before traversing from the root down toa leaf node, the path of which is determined by the generated number. This guarantees that processes with higher priorities are more likely to be selected. However, this also introduces a bias in training the network more often on higher-error samples. To solve this issue, importance sampling is used, assigning each experience with another weight, w_i, that controls its contribution to the loss function update and ensuring that the training process remains unbiased. This is calculated using the Equation (8):

$$w_i = \left(N \cdot P(i)\right)^{-\beta} \tag{8}$$

where:

- N is the size of the replay buffer
- $P(i)$ is the probability of sampling an experience, i
- β controls the weight correction for the bias

The value of β is annealed from 0.4 to 1 during training to slowly transition the agent's focus from prioritising fast learning earlier in training, to then prioritising unbiased and stable learning as the simulation progresses.

3.4 | POPULATION MANAGEMENT

With the agents implemented, the populations of agents are then managed and evolved using a Genetic Algorithm (GA), helping to refine their strategies and converge on an optimal solution. The genetic algorithm itself controls how the agents change over time, while the population management system determines which of the agents survive and reproduce to create the next generation.

3.4.1 | GENETIC ALGORITHM

To naturally evolve the genomes of each agent, a genetic algorithm is employed. An agent's genome consists of multiple hyperparameters, and their respective ranges, [min, max]. Specifically, this includes:

• Learning Rate (LR)	[0.0001, 0.01]
• Discount Factor (γ)	[0.95, 0.999]
• Decay	[0.995, 0.99985]
• Aggressiveness	[0.0, 1.0]
• Awareness	[4, 6]

The learning rate controls the weight adjustment steps as the neural network learns. The discount factor, as discussed in section 3.3.1, determines if the agent is short-sighted or far-sighted by controlling the importance of future rewards over immediate ones. The decay manages decay the epsilon value, ε, this controls the agent's exploration by controlling the balance between random, new actions and known good actions. The aggressiveness factor determines the playstyle of the agents, managing the blend of rewards obtained by the reward functions discussed in Section 3.3.2. Finally, the awareness determines the size of the agent's local environment that can be "seen".

The genetic algorithm implements two operations that work to evolve these gene values and converge onto an optimal solution. As parents are selected for reproduction, the crossover function is used to create a new child genome using Darwin's theory of pangenesis *(2010)*, this is detailed in Algorithm 1. For each gene, a new value is randomly assigned, chosen uniformly from the range between the two parents' corresponding gene values. This function occurs with a given 'crossover rate' probability that is set to a constant value of 0.8.

ALGORITHM 1: CROSSOVER

```
    Inputs: p1 & p2, two parent genomes selected from the population
    Output: a child genome, created from the two parent genomes
1   IF (random[0,1] ≥ crossover rate) THEN
2       RETURN p1
3   END IF
4
5   child genome = {}
6   FOR EACH gene IN p1
7       min val = min(p1[gene], p2[gene])
8       max val = max(p1[gene], p2[gene])
9       child genome[gene] = random[min val, max val]
10  END FOR
11  RETURN child genome
```

The second function is mutation, which helps to introduce genetic diversity by randomly shifting the gene values within an existing agent's genome, this is shown in Algorithm 2 and occurs with a given probability set by the 'mutation rate' which is set to a constant value of 0.2. When an agent is selected for mutation, a small, random value is added to each of its genes. This value is selected using a Gaussian distribution, making larger shifts less likely, and as a result helping the stability of the population convergence whilst giving an opportunity to escape the trap of a local optimum. After the value is added, it is ensured that the value still falls within the predefined ranges.

ALGORITHM 2: MUTATE GENOME

```
    Input: a genome to be mutated
    Output: the mutated genome
1   mutation rate ← a predefined value to determine how often a genome is mutated
2   IF (random[0, 1] ≥ mutation rate) THEN
3       RETURN genome
4   END IF
5
6   mutated genome = genome
7   gene ranges ← the predefined gene names and ranges: 'gene' : [min:max]
8   FOR EACH gene, range IN gene ranges
9       sigma = (range max – range min) * 0.1
10      mutation value = random_gaussian[mean=0, std=sigma]
11      mutated genome[gene] = mutated genome[gene] + mutation value
12      mutated genome [gene] = clamp(mutated genome[gene], range min, range max)
13  END FOR
14  RETURN mutated genome
```

The evolution and replacement of the agent population happens cyclically. The frequency is managed by the 'population replacement frequency' that replaces and evolves the entire population after a set number of matches, this is set to 10. When replacing a population, the selection for parents is weighted by their fitness value, therefore those which achieve a higher fitness are more likely to be preserved for the next generation. To ensure the best-found solution is maintained, a system of elitism is also employed, this is used to preserve the single best solution by copying it into the succeeding generation *(Tang, K.S., et al. 1996).*

To define fitness, the running score of the agent over the 10 games could not be used as a direct comparison. This is due to the nature of the reward system having multiple strategies, therefore offensive rewards could be more favourable than defensive rewards. Therefore, a more sophisticated measure, seen in Algorithm 3, was created that layered the base score with a behavioural fitness bonus that rewards agents for performing team-oriented tasks, even when those actions did not directly lead to a win. These included spending time in enemy territory, rescuing teammates and tagging opponents.

ALGORITHM 3: AGENT FITNESS CALCULATION

```
   Input: population,  a full list of agents for a single team
   Output: the fitness values for each agent in the given population
1  ATET ← individual agent's time in enemy territory
2  fitness values = [ ]
3  FOR EACH agent IN population:
4      | behavioural bonus = (0.1 * ATET) + 5.0 * (teammates rescued) +( 2.0 * enemies tagged)
5      | diversity bonus = |agent aggressiveness – team avg aggressiveness| * diversity multiplier
6      | agent fitness = score + behavioural bonus + diversity bonus
7      | APPEND agent fitness TO fitness values
8  END FOR
9  RETURN fitness values
```

Finally, to align with the principles given by Surowiecki for collective intelligence, a maximum diversity bonus of 75.0 is awarded as an incentive to avoid the team converging on a single strategy. This feature provides a reward to agents whose aggressiveness gene value differs from the team's average, encouraging strategic diversity. By combining the raw performance, behavioural incentives, and a push for genetic diversity, the fitness score creates a way of guiding the evolution towards a more effective solution.

Before the parents are selected for crossover, the fitness values for each agent in the population must be converted to a probability proportionate to its fitness. However, this requires all fitness values to be positive, but since the reward function consists of both positive and negative rewards, it is possible for a negative fitness to be achieved. To solve this, a fitness scaling step is applied, shifting all values up by the minimum fitness score found (plus one), providing this score is negative. This shifts the entire fitness distribution into a positive range, without changing the relative differences. After this, the probabilities of each agent can be calculated using Algorithm 4, and the agents can be selected.

Once the two parents are selected, they are then given to the crossover function discussed earlier. If they fall outside of the 0.8 probability, the first parent selected will be duplicated, and potentially mutated. If a child is created, they inherit the trained neural network from one parent, but only if the awareness is identical. This is due to the network structure being created with the specific input size, controlled by the awareness. Therefore, having a different awareness value would mean that the parent's network weights are incompatible and, in this case, a new network is created.

Overall, the population management framework (see Appendix A) provides a structure for simulated natural selection, with the genetic algorithm supplying the core functionality for variation. The success of the evolution is dependent on the fitness function, which is a complex mix of performance, behaviour, and diversity to encourage the conditions needed for collective intelligence to emerge. To add, the inheritance of neural network weights ensures that the knowledge is preserved through generations whenever possible to help accelerate training. Therefore, these elements create an evolutionary loop capable of finding optimal solutions in a complex space, encouraging the emergence of complex cooperative and adapted teams.

ALGORITHM 4: CALCULATE PROBABILITIES

```
Input: a list of fitness values for a single team
Output: the probabilities for the selection process
min score = min[fitness values]
IF min score < 0 THEN
    offset = absolute(min score) + 1
    adjusted values = ADD offset TO ALL fitness values
ELSE
    adjusted values = fitness values
END IF

total fitness = SUM adjusted values
probabilities = [ ]
IF total fitness > 0 THEN
    FOR EACH score IN adjusted scores
        probability = score / total fitness
        ADD probability TO probabilities
    END FOR
ELSE
    num players = LENGTH fitness values
    uniform probability = 1 / num players
    probabilities = [uniform probability FOR i FROM 1 TO num agents]
END IF
RETURN probabilities
```

4 | RESULTS AND ANALYSIS

This chapter presents the results from the training process and the evaluation tournaments. It first outlines the two-tournament structure used to identify the top-performing teams and to analyse the impact of different communication strategies to identify the potential of collective intelligence withing the trained system. Following this, an analysis of the quantitative data is presented, focusing on agent performance, team behaviour, and strategic outcomes.

It was necessary to treat the two sets of teams (Team 1 and Team 2) independently, as their learned policies were specific to their starting side of the map. During the training process, agents belonging to "Team 1" always started on the left side of the arena, while "Team 2" agents always started on the right. Consequently, agents learned strategies that were tailored to their starting conditions, including attacking and defending movement directions to take at the beginning of the match. If these teams were treated as interchangeable, the results would have been invalid as their policies would no longer be effective when started on the opposite side of the arena.

4.1 | SIMULATION RUNS

In total, ten independent simulation runs were conducted, each resulting in a distinct "best performing" team for both Team 1 and Team 2. Each run consisted of 1000 matches, and the best team from each was identified by tracking the cumulative fitness scores of the agents throughout the training process, calculated using a summation of Equation (3).

From running these simulations, all the data gathered was largely averaged to give the results of gene distances of individual agents from the average of their team, the impact of elitism on the population, and the aggressiveness evolution over the generations to show the impact of diversity encouragement on the system.

Figure 14 shows the average gene distance of the agent's current genes to their team's average – used to determine the amount of diversity within either team. As seen, both team's average gene distances plumet in the early generations, before stabilising below 0.1. This indicates that the agent's gene values rapidly converge on a single strategy. Although mutation and the diversity bonus had attempted to find alternate strategies, from this plot, it is evident that no alternative optimal solution could be found whilst traversing the fitness landscape.

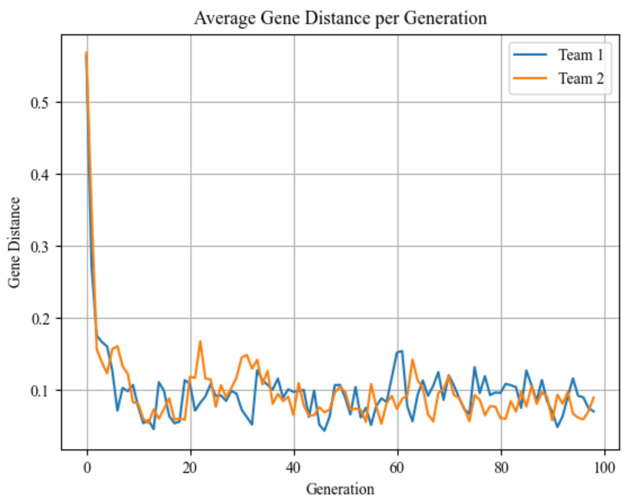


***Figure 14**: a graph showing the average distance of each agent's gene from their respective team average.*

However, when training teams together, Figure 15 shows that the overall average value for the “aggressiveness” gene increasers to ~0.4, and the standard deviation converges to a value just above 0.5, showing that there is a large diversity in the gene value. This is a direct contradiction to the gene distance results per team, but a closer inspection shows that the best-found teams for each run always co-evolved to adopt opposing strategies. Most commonly, one team would become almost entirely aggressive (aggressiveness ≈ 1.0), and the other would become almost entirely defensive (aggressiveness ≈ 0.0).

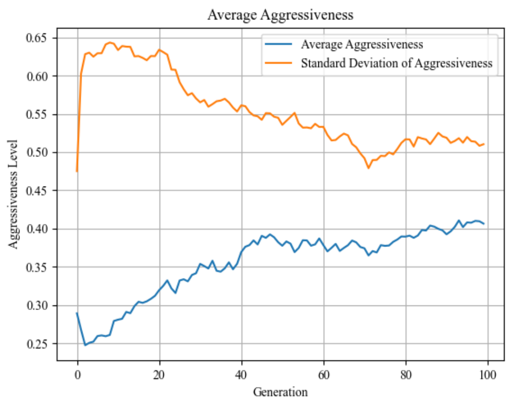


Figure 15: *A graph showing the average and standard deviation of the aggressiveness gene value for ALL player agents in the simulation over the generations.*

The mechanism of elitism used in the genetic algorithm preserves the best-performing agents from each generation, guaranteeing that successful genetic combinations were not lost during the creation of offspring through crossover. To assess the impact of this mechanism, the average fitness of elite agents was compared to the average fitness of the new “child” agents. These results can be seen in Figure 16.

The results help to confirm that the mechanism is working as intended, as the average elite fitness is consistently higher than that of the children. This indicated that the top performing agents are correctly being preserved and copied to the succeeding generation, following the defined by Tang, K.S., et al. *(1996)*. However, there is also some noise in the elite graphs as the measure of performance is done so for each generation independently, therefore the elite agent that was preserved had the possibility of performing worse with the new set of agents. As the fitness is dependent on multiple factors, some of which tied to the team, it could be the case that the previous best-performing agents no longer fit into the current team, creating a selection pressure that favours both individual ability and contribution to a team-wide solution.

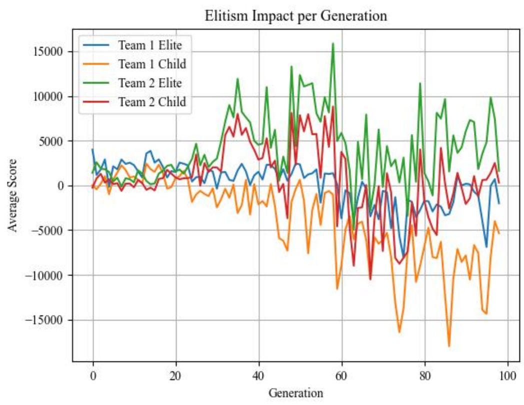


Figure 16: *A graph showing the elitism impact for both Team 1 and Team 2. Plotting both the average scores of the elite agents, and the "child" agents.*

4.2 | TOURNAMENTS

To determine the single best team for each side, two round-robin tournaments were conducted. In the first tournament, the best team from each of the ten training runs was matched against every opposing team for five matches, meaning that each team played a total of 50 games. This provided a stable measure of performance, providing the win rates for each team. The results of this tournament are shown in Table 8.

Team 1 Win Rate	Run Number	Team 2 Win Rate
46.0 %	**1**	20.0 %
74.0 %	**2**	48.0 %
46.0 %	**3**	40.0 %
56.0 %	**4**	30.0 %
28.0 %	**5**	44.0 %
64.0 %	**6**	36.0 %
28.0 %	**7**	**60.0 %**
48.0 %	**8**	40.0 %
68.0 %	**9**	26.0 %
26.0 %	**10**	48.0 %

Table 8: *The results of running the round-robin tournament, with the best Team 1, and Team 2 (determined by win rate) highlighted in green.*

Based on these results the top-performing teams were identified as Team 1 from run 2 with a 74% win rate, and Team 2 from run 7 with 60%. From assessing the final genes of the top performing agents, all the 5 top performing teams converged on the same, highly aggressive strategy with an aggressiveness ≈1.0.

Using these two teams, the main results could be produced. For a large majority of the results, a second tournament was conducted to investigate the possible emergence of collective intelligence, and to evaluate the agent's performance using the inter-agent communication. The experiment tested three distinct communication settings:

1. **Communicating**: The default setting where agents share information regarding enemy, obstacle, and flag locations with nearby teammates at every timestep.

2. **Non-Communicating**: Communication is disabled, forcing agents to act independently on their own observations.

3. **Hallucinating**: Agents generate and shared random information regarding enemy, obstacle, and flag locations, simulating noisy communication.

To ensure the experiment was controlled, and each setting is tested fairly in isolation, one team's communication was altered, while it competed against a standard, communicating opponent. This process was repeated for each team, providing a stable baseline for the experiment and analysis. Each communication strategy was tested for 500 matches, and the resulting data was averaged to ensure that any trends could be identified.

The data gathered from communicating and non-communicating agents help to assess if the performance is higher when working collectively, rather than independently to assess if collective intelligence has potentially emerged. Whereas the data from hallucinating agents help to validate the communication pipeline, ensuring that agents depend and act upon the contents of the shared information, rather than just requiring the signal given from receiving random data.

Figure 17 shows the average cumulative fitness distribution of each team and strategy, resulting from the second tournament.

Immediately it is evident that the hallucinating teams, generating random and false information, perform considerably worse than either of the communicating, or non-communicating agents, having a large, inconsistent range, and low mean score. This outcome provides a strong indication that the agents are not reacting only to the signal given from communication, but are actively interpreting the content of the messages, confirming that the CNN is successfully extracting the relevant features from the agent's shared "mental map" to inform their strategic decisions.

When comparing the communicating and non-communicating strategies, a trend is revealed for both teams where the mean fitness score is higher for communicating teams, compared to teams of individual agents. However, the magnitude of this improvement differs between the two teams.

For Team 1, the range of the finesses values is slightly lower, and the median is slightly higher, showing that on average, the performance of agents is generally better whilst communicating. Although, this is much better demonstrated with Team 2, as the mean is considerably higher, and the range is much lower when the agents are communicating. This helps to reinforce the point that the agents perform consistently better whilst communicating, providing some indication towards collective intelligence.

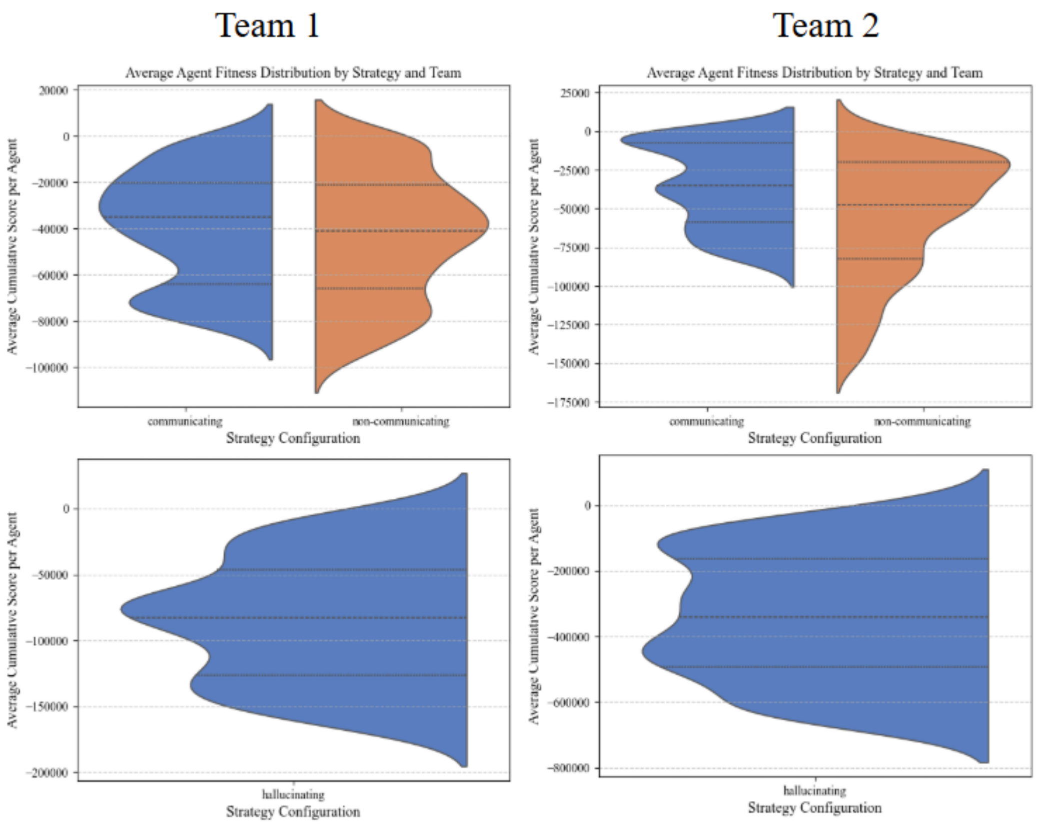


Figure 17: Violin plots for the scores of agents using different communication strategies for both Team 1 (left) and Team 2 (right). These include communicating and non-communicating (top), and hallucinating (bottom).

The initial run of this experiment included an error where some of the agent's networks would be replaced randomly every generation. This was when another gene was created to control the distance that each agent can "see". Here, if the parents' network sizes did not match the child's, the network would be created from scratch, negating any previous learning. However, the results from this communication tournament, seen in Appendix B, proved to be insightful. The disruption to these agents caused the mean of the communicating teams to be much lower than the teams that consistently evolved. This finding helps to solidify the fact that the higher mean of the communicating teams in Figure 17 was a direct result of the policies that were learned slowly over time. This shows that the cooperative behaviours were not simply random, or hardcoded, but were forged from the changing weights of the neural networks, evolved over many generations. When disrupted, the team's ability to work cooperatively collapsed.

4.3 | TEAM STRATEGIES AND COHESION

To analyse any emergent cooperative behaviours of the evolved teams, a series of 100 matches was played between the two top teams found in the tournament. This experiment tracked team, cohesion, calculated as the average distance between all agents on a team. Over the course of each match, and key events such as flag captures, agent jailing, team releases, and flag tackling were timestamped to show how a team's formation changes in response to these events.

From the results generated, it was clear that there was a common occurrence where distance would spike as a flag is captured. This can be seen in all the Figures 18, 19, and 20. However, from analysing the gameplay captures in the following steps after these flag captures, there were some sighs of team-based strategy.

Shown in the timestep captures in Appendix C.1, the flag carrier immediately changes its direction to take the flag back to its own home base. Although not surprising as a standalone observation, it can also be seen that a teammate that was close to the flag bearer also changes its trajectory to provide an escort towards completing the goal. This same system is shown again in Appendix C.2, mapping the timesteps after the flag capture highlighted in Figure 19. Again, the carrier changes its trajectory upon retrieving the enemy flag, but this time there are multiple escorting agents, building up a defensive formation to ensure the goal is completed.

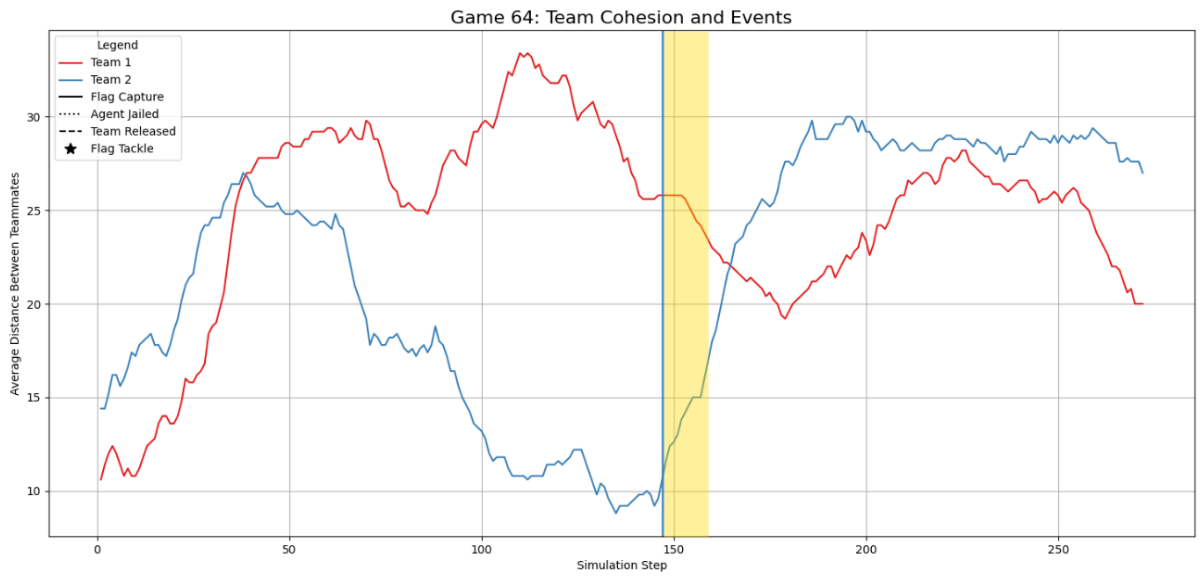


Figure 18*: A graph for game #64, plotting the average distance between teammates over time, with key events marked, and the area of interest mapped in yellow (see Appendix C.1 for sequence images)*

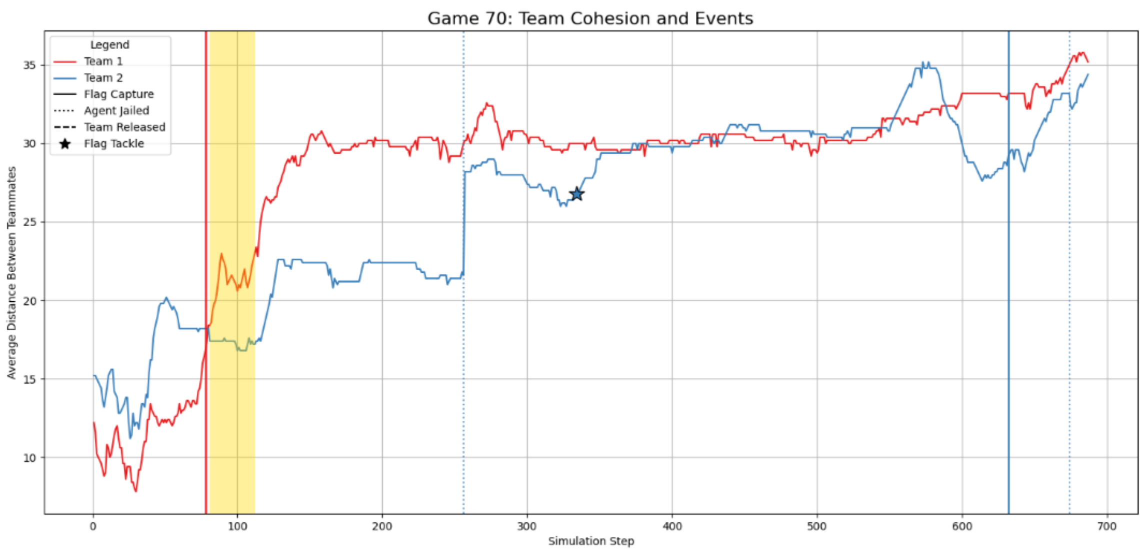


Figure 19*: A graph for game #70, plotting the average distance between teammates over time, with key events marked, and the area of interest mapped in yellow (see Appendix C.2 for sequence images)*

The match plotted in Figure 20 show the defensive formation acting appropriately to protect the flag. The timesteps shown in Appendix C.3 show that the escorting agents attacking the enemies that are in proximity to the flag-bearer and subsequently jailing them whilst the flag-bearer itself retreats away from the danger to allow for the other agents to make the area safer. This behaviour, although rarer than desired, still shows the potential of team-based behaviours through emergent collective intelligence.

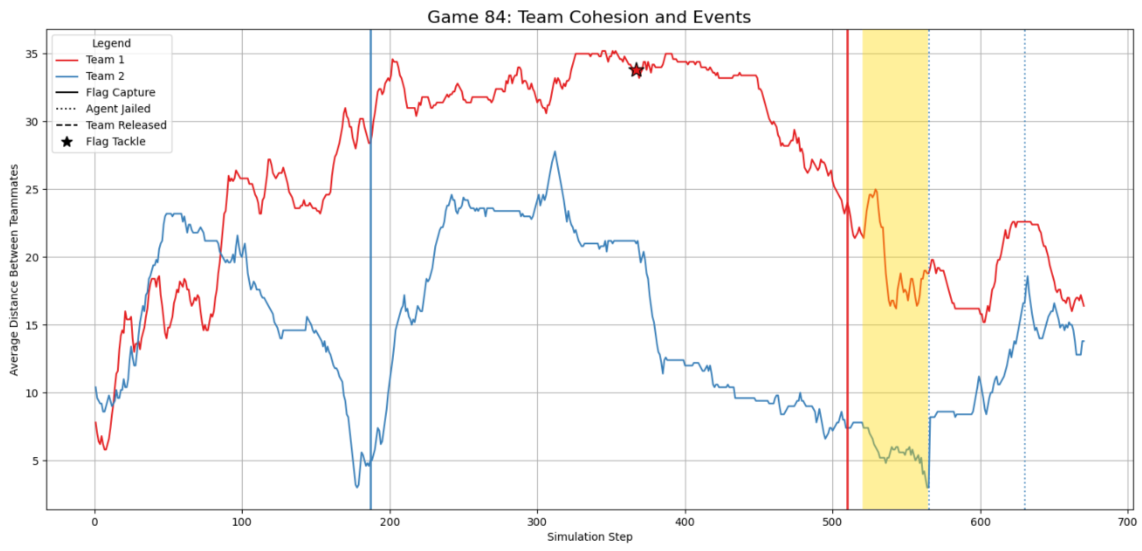


Figure 20*: A graph for game #84, plotting the average distance between teammates over time, with key events marked, and the area of interest mapped in yellow (see Appendix C.3 for sequence images).*

Overall, the results presented show a basis of some collective behaviours being exhibited. This was a positive outcome, showing that the averages of communicating agents was higher than those acting independently. However, some key issues with the shaping of the reward function, and the encouragement of diversity were identified. These issues are inherent with reinforcement learning, which is a reason why these complex behaviours can be extremely difficult to emerge. The following chapter will; discuss the main outcomes, how the system encouraged collective intelligence, the improvements that could be made, and any further research following on from this experiment.

5 | Discussion

The primary goal of this research was to investigate whether a system of direct inter-agent communication, alongside the use of deep reinforcement learning and a genetic algorithm could encourage the emergence of collective intelligence in a competitive, multi-agent environment, and to what extent it's effective. The results from the simulation and tournaments demonstrate that this approach has the potential to be a viable solution, showing signs of collective intelligence and complex team-based strategies.

This chapter will first discuss the initial research question, analysing how the communication between agents acted as a performance multiplier, showing potential signs of collective intelligence emergence. Then, the system will be analysed to show how collective intelligence was encouraged, and what the key findings were that resulted in the conclusion of potential or early emergence. Finally, the system will be discussed, given areas that would require improvement, and where this research could be taken to in future projects.

5.1 | MAIN RESEARCH CONCLUSIONS

From the results, it can be concluded that the inter-agent communication encouraged the emergence of collective intelligence. The main indicator being the increased average performance and the consistency of the communicating teams. This aligns with the principle outlined by Malone and Bernstein *(2015),* where a key sign of collective intelligence is a group's ability to exceed the performance of individuals.

However, this evidence should be considered carefully as Surowiecki *(2004)* had stated that a truly intelligent group can outperform even its smartest individual members. In this experiment, seen in Figure 17, the top-performing non-communicating agents still achieved a higher fitness than the average of the communicating team. While the communication mechanism allowed for the average performance to be increased, it did not elevate every agent above the absolute best individual agents. Therefore, while the current results suggest that emergent cooperative behaviours are possible, a result where the performance range of the communicating team is entirely above that of the non-communicating team would be more definitive.

Beyond the performance averages, the observed behaviours of agents was a second indicator of the emergence of some collective intelligence. As detailed in the previous chapter, the simulation showed some sophisticated, multi-agent strategy to ensure the goal is being met. This involved a flag carrier immediately retreating to its home base while its nearby teammates follow and spread out to create a protective "escort", and in some cases, these agents pursued enemies that posed a threat to the flag bearer. These emergent behaviours show a form of decentralised coordination, where individuals perform separate tasks to serve a group objective, providing another characteristic of collective intelligence. As this was not explicitly programmed, it shows that the framework provided the conditions for the emergence of intelligent, team-based behaviours.

The agents themselves effectively learned to play the game of Capture the Flag and were successfully competing against each other. This shows that the reward function was fit-for-purpose in guiding the main strategies and actions needed to complete any given match. However, as is natural with reinforcement learning, it would have been beneficial to run more tests in order to fine-tune and balance the rewards further to make a defensive playstyle more suitable and encourage more diversity within a single team, something that the available time and resources didn't allow dor.

5.2 | ENCOURAGMENT OF COLLECTIVE INTELLIGENCE

The principles set out by Surowiecki *(2004)* were a primary source of guidance for the system implementation to ensure the best chance of emerging collective intelligence in such a complex environment. Here, the "diversity of opinion" was encouraged through the aggressiveness gene, where a diversity bonus was added to the score for outliers from the average. Although this encourages different strategies, this did not stop the convergence on a single, highly aggressive approach.

The "independence of members" was implemented using a separate network for each agent, therefore ensuring each makes their own decisions and policies. Although local communication shared some information, this being limited by a distance still ensured there was no "hive mind" for each team, where information would be shared globally, as this would negate the independence factor.

As discussed, the system had no direct implementation or controller for specific behaviours – this fact being reinforced by the results of the error test. Instead, the system was decentralised, relying on emergent strategies as a sum of these decentralised decisions. The reward function used encourages team-based behaviours, alongside the independent gameplay strategies to create well-rounded agents.

Finally, the aggregation of the system was implicit through the nature of capture the flag. Here, the outcome of the match is a combination of the effects of all agent's actions. Although a single agent can perform the task of capturing the flag and returning it to the base, success is often determined by the defensive actions and strategic positioning of its teammates. The teams long-term win rate therefore acts as the main aggregation of these coordinated contributions, rather than the actions of a single player.

5.3 | SYSTEM IMPROVEMENTS

While the experiment demonstrated that collective intelligence is likely able to emerge using a system of inter-agent communication, there were also several improvements that would aid in improving the current results.

Firstly, as the agents currently are specialised to their starting sides of the map, making Team 1 and Team 2 independent groups. Although the obstacles themselves randomise to provide different paths, the generalisation could be improved by assigning teams to random sides each match and even rotating the map itself. This would help for agents to learn more robust policies that are no longer dependent on a fixed starting area.

Second, the diversity of the agents could be improved by increasing the bonus value, given to outliers. As the current system still converges, increasing this value will better encourage the "diversity of opinion" principle. To also encourage a multi-strategy convergence, the reward function could require further experimentation to make sure the defensive and offensive rewards are perfectly balance, therefore providing no incentives towards a certain playstyle. It is assumed that the current system has a bias towards aggressive playstyles due to those defensive teams mostly yielding the lowest win-rate in the tournament.

Apart from the system, it would have been beneficial to train the agents in parallel, and for much longer. Due to the limitations in time and computational power, this was not possible for this experiment, but as the mean performance was already improved for communicating agents, this could indicate that with further training, the gap will widen, and collective intelligence will fully emerge.

Another possibility that would have been beneficial to training would have been a stage-based learning. This would allow agents to learn the game in smaller stages (e.g. collecting flags, tagging enemies, returning flags, etc.). In hindsight, this could have led to even more effective agents, although again, this would have taken an overwhelming amount of time and resources.

5.4 | CONCLUSION AND FURTHER RESARCH

This report set out to investigate the application of inter-agent communication for the emergence of collective intelligence within a competitive, team-based environment. By developing an advanced capture the flag simulation, this experiment demonstrated that this system, combined with deep reinforcement learning, and genetic algorithms, can produce some instances of complex, cooperative team-based strategies.

The key findings of this report confirm that a communication pipeline can play an important role in effective teamwork. Through the tournaments, it showed that communication acts as a performance multiplier, increasing the average scores and performance consistency of teams. The analysis of agent behaviour provided evidence of some emergent strategies, such as a defensive escorting technique, which shows some signs of collective intelligence.

From running the simulation for 1000 matches per pair of teams, the results shown were surprising for the relatively short training process. Compared to Jaderberg's model (2019), that ran for 450,000 matches, the results of this report highlighted a more sample-efficient path to complex, team-based strategies through action masking and providing a direct channel for coordination through communication. This allowed the agents to show signs of learnt strategies in a fraction of the training time and demonstrating that these mechanisms can act as an agitator for the emergence of collective intelligence.

While the results are generally positive, the discussion acknowledged the limitations of the current system in-place, such as the tendency to converge on single strategies, and the need for a more robust form of generalisation.

For further research, the most logical step is to explore advanced communication strategies. As the current system sends constant messages to all neighbouring teammates, a system of targeted messaging may be beneficial. This would involve allowing agents to learn when top send messages, and to what teammates that are in the communication range. To add, the content of the messages could be advanced further, allowing agents to learn how to construct their own messages, and develop their own team language to better coordinate their actions. However, these suggestions will require an extensive amount of training as with new abilities, the agents have more to learn.

In conclusion, this project contributes to the field of Multi-Agent Reinforcement Learning (MARL) by providing a framework for evolving and analysing team-based behaviours. It demonstrates the complexities of designing learning algorithms, reward functions and evolutionary methods, and how it is possible to maintain the conditions necessary for decentralized intelligence to simultaneously learn and cooperate to solve complex, strategic, and competitive problems.

REFERENCES

Abar, S., Theodoropoulos, G. K., Lemarinier, P., & O'Hare, G. M. (2017). Agent Based Modelling and Simulation tools: A review of the state-of-art software. *Computer Science Review*, *24*, 13-33. doi:10.1016/j.cosrev.2017.03.001

Antelmi, A., Caramante, P., Cordasco, G., D'Ambrosio, G., De Vinco, D., Foglia, F., ... & Spagnuolo, C. (2024). Reliable and efficient agent-based modelling and simulation. JASSS, 27(2), 1-28. doi:10.18564/jasss.5300

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. *"Deep Reinforcement Learning: A Brief Survey"* in IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26-38, Nov. 2017, doi: 10.1109/MSP.2017.2743240

Axelrod, R. (1987). The evolution of strategies in the iterated prisoner's dilemma. *The dynamics of norms*, *1*(1).

Bellemare, M. G.; Naddaf, Y.; Veness, J.; & Bowling, M. 2013. *The arcade learning environment: an evaluation platform for general agents.* Journal of Artificial Intelligence Research 47(1):253–279, doi:10.1613/jair.3912

Bellman, R. E. (1966). *Dynamic Programming*. Princeton University Press. doi:10.1126/science.153.3731.34

Bonabeau, E. (2002). Agent-based modelling: Methods and techniques for simulating human systems. *Proceedings of the national academy of sciences*, *99*(suppl_3), 7280-7287. doi:10.1073/pnas.082080899

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems* (No. 1). Oxford university press.

Busoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *38*(2), 156-172. doi:10.1109/TSMCC.2007.913919

C. Hu, Zhao, Y., Wang, Z., Du, H., & Liu, J., *"Games for Artificial Intelligence Research: A Review and Perspectives,"* in IEEE Transactions on Artificial Intelligence, vol. 5, no. 12, pp. 5949-5968, Dec. 2024, doi:10.1109/TAI.2024.3410935

Darwin, C. On the Origin of Species by Means of Natural Selection. London: Murray, 1859.

Darwin, C., & Charles, D. (2010). The variation of animals and plants under domestication (Vol. 2). Cambridge University Press.

Dorigo, M., Bonabeau, E., & Theraulaz, G. (2000). Ant algorithms and stigmergy. *Future generation computer systems*, *16*(8), 851-871. doi:10.1016/S0167-739X(00)00042-X

Epstein, J. M., & Axtell, R. (1996). *Growing artificial societies: social science from the bottom up*. Brookings Institution Press.

Golberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, *1989*(102), 36.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press, doi:10.4258/hir.2016.22.4.351

Hassabis, D., Kumaran, D., Summerfield, C., & Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, *95*(2), 245-258, doi:10.1016/j.neuron.2017.06.011

Hernandez-Leal, P., Kaisers, M., Baarslag, T., & De Cote, E. M. (2017). A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*. doi:10.48550/arXiv.1707.09183

Holland, J. H., "Genetic Algorithms." *Scientific American*, vol. 267, no. 1, 1992, pp. 66–73. *JSTOR*,

Hubel D. H., Wiesel T, N., Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J Physiol. 1962 Jan;160(1):106-54. doi:10.1113/jphysiol.1962.sp006837

Ilachinski, A. (2004). Artificial War: Multiagent-based Simulation of Combat. Singapore: World Scientific Pub.

Jacob, T., Duran, D., Pfeiffer, T., Vignati, M., & Johnson, M. (2023, September). Multi-agent Reinforcement Learning for Unmanned Aerial Vehicle Capture-the-Flag Game Behavior. In *Intelligent Systems Conference* (pp. 174-186). Cham: Springer Nature Switzerland.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., ... & Graepel, T. (2019). Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, *364*(6443), 859-865.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., Siegelbaum, S., Hudspeth, A. J., & Mack, S. (Eds.). (2000). *Principles of neural science* (Vol. 4, pp. 1227-1246). New York: McGraw-hill.

Kramer, O. (2017). Genetic Algorithms. In: Genetic Algorithm Essentials. Studies in Computational Intelligence, vol 679. Springer, Cham. doi:10.1007/978-3-319-52156-5_2

Krizhevsky, A., Nair, V., & Hinton, G. (2010). Cifar-10 (Canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html*, *5*(4), 1.

Reynolds, C. W. (1987, August). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (pp. 25-34). doi:10.1145/37401.37406

'ROBOT LEARNING, edited by Jonathan H. Connell and Sridhar Mahadevan, Kluwer, Boston, 1993/1997, xii+240 pp., ISBN 0-7923-9365-1 (1999) *Robotica*, 17(2), pp. 229–235. doi:10.1017/S0263574799271172

Kaelbling, L.P., Littman, M.L. & Moore, A.W., *"Reinforcement Learning: A Survey",* Vol. 4, 1996, doi:10.1613/jair.301

Lake, B.M., Ullman, T.D., Tenenbaum, & J.B, Gershman, S.J. (2017) 'Building machines that learn and think like people', *Behavioral and Brain Sciences*, 40, p. e253. doi:10.1017/S0140525X16001837

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P., *"Gradient-based learning applied to document recognition"* in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791

Li, Y., "Deep Reinforcment Learning: An Overview", 2017, doi:10.48550/arXiv.1701.07274

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3), 293-321, doi: 10.1007/BF00992699

Macal, C. M., & North, M. J. (2005, December). Tutorial on agent-based modelling and simulation. In *Proceedings of the Winter Simulation Conference, 2005.* (pp. 14-pp). IEEE. doi:10.1109/WSC.2005.1574234

Malone, T. W., & Bernstein, M. (Eds.). (2015). *Handbook of collective intelligence*. MIT press.

Masad, D. & Kazil, J. (2015). *MESA: An Agent-Based Modelling Framework*. 51-58. doi:10.25080/Majora-7b98e3ed-009

McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5**, 115–133 (1943), doi:10.1007/BF02478259

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Reidmiller, M. *Playing Atari with deep Reinforcement Learning*. (2013) doi:10.48550/arXiv.1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D. *et al. Human-level control through deep reinforcement learning*. Nature 518, 529–533 (2015). doi:10.1038/nature14236

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). *Prioritized Experience Replay*, doi:10.48550/arXiv.1511.05952

Sejnowski, T. J. *The Deep Learning Revolution,* Cambridge, MA: MIT Press, 2018, doi:10.7551/mitpress/11474.001.0001

Silver, D., Schrittwieser, J., Simonyan, K. et al. *Mastering the game of Go without human knowledge.* Nature 550, 354–359 (2017), doi:10.1038/nature24270

Stanford University. *AI Index Report.* 2025. Accessible via: hai.stanford.edu/ai-index/2025-ai-index-report

Surowiecki J. *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies and nations*. Doubleday. 2004.

Swartz, C., *How to Play Capture the Flag*, Scout Life URL:scoutlife.org/hobbies-projects/funstuff/160235/how-to-play-capture-the-flag/

Whiteson, S., Tanner, B., Taylor, M.E., and Stone, P., *"Protecting against evaluation overfitting in empirical reinforcement learning,"* 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning *(ADPRL)*, Paris, France, 2011, pp. 120-127, doi:10.1109/ADPRL.2011.5967363

Tang, K. S., Man, K. F., Kwong, S., & He, Q. (1996). Genetic algorithms and their applications. *IEEE signal processing magazine*, *13*(6), 22-37. doi:10.1109/79.543973

ter Hoeven, E., Kwakkel, J., Hess, V., Pike, T., Wang, B., rht, & Kazil, J.*, Mesa 3: Agent-Based modelling with Python* (2025) Delft University of Technology. doi:10.21105/joss.07668

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016, June). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning* (pp. 1995-2003). PMLR. doi:10.48550/arXiv.1511.06581

Watkins, C. “Learning from Delayed Rewards”, 1989. King’s College.

Watkins, C.J.C.H., Dayan, P. *Q*-learning. *Mach Learn* **8**, 279–292 (1992). doi:10.1007/BF00992698

Woolley, A. W., Chabris, C. F., Pentland, A., Hashmi, N., & Malone, T. W. (2010). Evidence for a collective intelligence factor in the performance of human groups. *science*, *330*(6004), 686-688. doi:10.1126/science.1193147

Wolpert, D. H., & Tumer, K. (1999). *An introduction to collective intelligence,* doi:10.48550/arXiv.cs/9908014

Wu, Y., & He, K. (2018). Group normalization. In Proceedings of the European conference on computer vision (ECCV) (pp. 3-19). doi:10.48550/arXiv.1803.08494

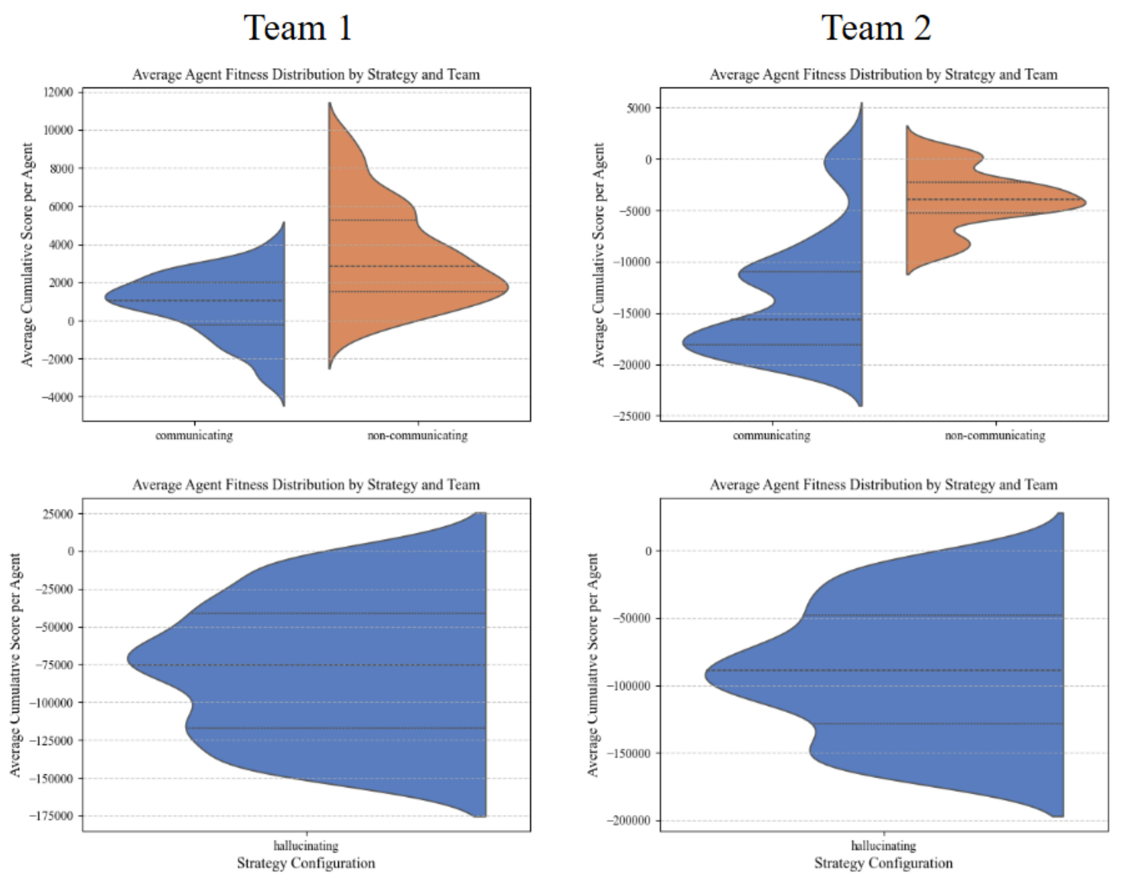
Yegnanarayana, B. *"Artificial Neural Networks"*, India: PHI Learning, 2003

APPENDIX A – GENETIC ALGORITHM FRAMEWORK

ALGORITHM 5: FULL GENETIC ALGORITHM

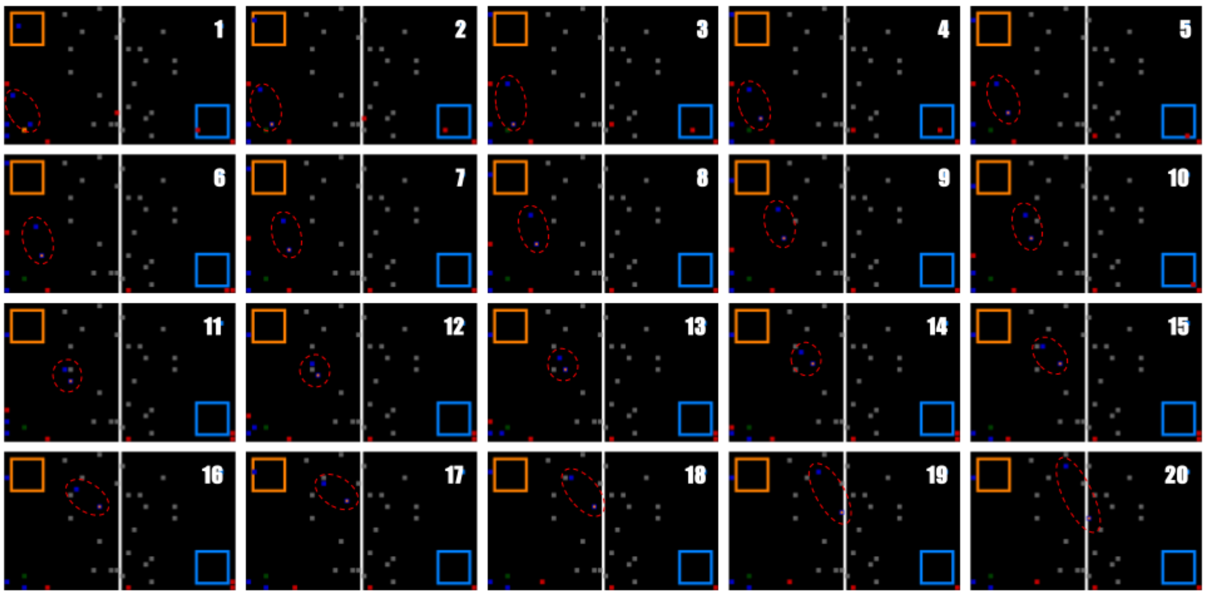
```
population replacement frequency ← a value for the agent replacement frequency
IF (match number % population replacement frequency = 0) THEN
    team fitness values = Agent Fitness Calculation(team agents) ←[ALGORITHM 3]
    SORT team agents BY team fitness values IN DECENDING ORDER

    new generation = [ ]
    elitism count ← the number of agents with the highest fitness to be preserved
    FOR i FROM 1 TO elitism count
        elite agent = team agents[i]
        APPEND elite agent TO new generation
    END FOR

    num children = team size – elitism count
    selection probs = Calculate Probabilities(team fitness values) ← [ALGORITHM 4]
    FOR i FROM 1 TO num children
        p1 = SELECT agent FROM population USING selection probs
        p2 = SELECT agent FROM population USING selection probs WHILE p2 != p1
        child genome = Crossover(p1, p2) ← [ALGORITHM 1]
        mutated genome = Mutate Genome(child genome) ← [ALGORITHM 2]
        APPEND mutated genome TO new generation
    END FOR
END IF
```

APPENDIX B – NETWORK REPLACMENT RESULTS

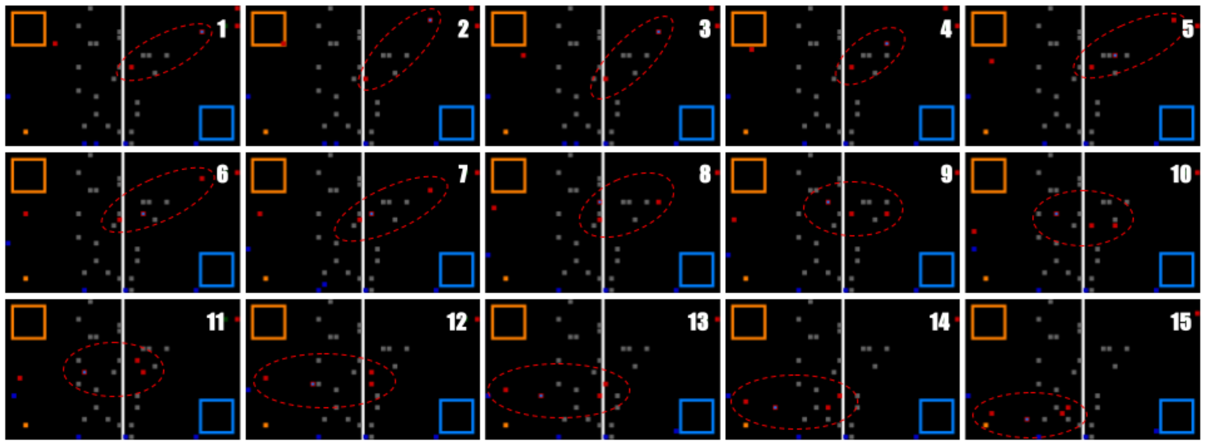


Appendix B 1: *The communication tournament results, found from a mistaken run that replaced agents' networks randomly every generation.*

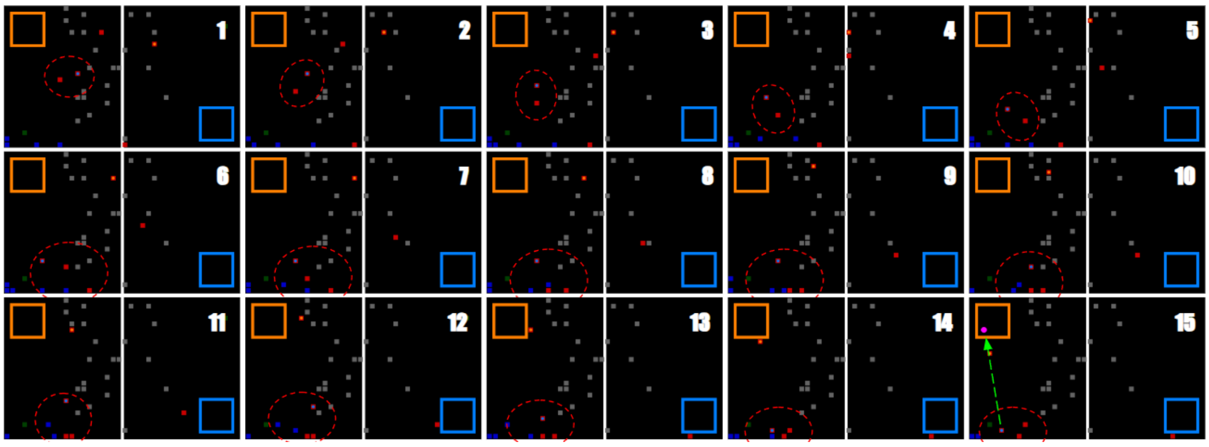
APPENDIX C – BEHAVIOUR GAMEPLAY SEQUENCES



Appendix C 2: *The gameplay sequences from the highlighted area of match #64, with the agents of interest circled in red.*



Appendix C 3: *The gameplay sequences from the highlighted area of match #70, with the agents of interest circled in red.*



Appendix C 4: *The gameplay sequences from the highlighted area of match #84, with the agents of interest circled in red.*