Using Evolution and Deep Learning to Generate Diverse Emergent Behaviour in Intelligent Agents

Marshall Joseph

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Department of Computer Science

Brock University

St. Catharines, Ontario

Abstract

Emergent behaviour is behaviour that arises from the interactions between the individual components of a system, rather than being explicitly programmed or designed. In this work, genetic programming is used to evolve an adaptive game AI, also known as an intelligent agent, whose job is to capture up to twenty-five prey agents in a simulated pursuit environment. For a pursuit game, the fitness score tallies each prey the predator captured during a run. The fitness is then used to evaluate each agent and choose who moves forward in the evolutionary process. A problem with only choosing the best performing agents is that genetic diversity becomes lost along the way, which can result in monotonous behaviour. Diverse behaviour helps agents perform under situations of uncertainty and creates more interesting computer opponents in video games, as it encourages the agent to explore different possibilities and adapt to changing circumstances. Inspired by the works of Cowan and Pozzuoli in diversifying agent behaviour, and Chen's work in L-system tree evaluation, a convolutional neural network is introduced to automatically model the behaviour of each agent, something previously done manually. This involves training the convolutional neural network on a large data set of behaviours exhibited by the agents, which take the form of image-based traces. The resulting model is then used to detect interesting emergent behaviour. In the first set of experiments, the convolutional neural network is trained and tested on several sets of traces, then the performance of each run is analyzed. Results show that the convolutional neural network is capable of identifying 6 emergent behaviours with 98% accuracy. The second set of experiments combine genetic programming and the convolutional neural network in order to produce unique and interesting intelligent agents, as well as target specific behaviours. Results show that the system is able to evolve more innovative and effective agents that can operate in complex environments and could be extended to perform a wide range of tasks.

Acknowledgements

I would first like to send a sincere thank you to my supervisor, Professor Brian Ross, for his immense amount of guidance, encouragement, and feedback over the past 2 years. I would also like to thank Cale Fairchild for his assistance with the computing cluster which allowed me to significantly speed up the execution of my experiments.

Next, I would like to thank my good friend Mathew Erwin for providing his expertise on data analysis, Andrew Pozzuoli for sharing his simulation code which saved me countless hours as I converted my system from Python to Java, and Tyler Cowan for providing insight on his past work and helping to improve my simulation's efficiency.

Lastly, I would like to give a special thanks to the rest of my friends, my family, and my partner for their encouragement and support over the years. Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, the problem description is described, highlighting the limitations of traditional measures of fitness, especially concerning the creation of diverse intelligent agents. Next, the key objectives and motivation behind the work will be presented. Four crucial research questions are posed that serve as motivation for the work. A brief discussion of the structure of the thesis concludes the chapter.

1.1 Problem Description

In a simulated pursuit domain (also known as *predator-prey*), an intelligent agent is given the task to capture a set number of prey agents scattered across the environment [46]. In this domain, evolutionary algorithms are commonly used to evolve the predator's controller [40]. A fitness score tallies the number of prey the predator caught during a run and is used to determine who moves forward in the evolutionary process. In essence, this is survival of the fittest.

A problem with only choosing the best performing agents is that genetic diversity becomes lost along the way, which can result in the evolution of common solutions having generic and monotonous behaviour. On the other hand, diverse behaviour helps agents perform under situations of uncertainty and creates more interesting computer opponents in video games, as it encourages the agent to explore different possibilities and adapt to changing circumstances.

One approach to addressing this issue is to use techniques that promote diversity in the population. These techniques encourage agents to explore different strategies, which can lead to an improvement in performance. For example, fitness sharing [8] is inspired by the idea of individuals in a population having to share their fitness score in a similar way to how species in nature occupying the same ecological environment

have to share resources [38]. Thus, individuals who are close to one another receive a lower fitness score, in hopes that the population spreads out into more genetically diverse areas. This method produces diverse individuals by making it difficult for a single dominant solution to emerge due to the incentives of exploring the search space. On the other hand, novelty search [29] rewards agents for discovering new and unexpected behaviours, regardless of their performance. Similarly, agents are encouraged to explore the environment and find novel solutions, which can help them adapt to changing circumstances and avoid local optima. Many-objective optimization [14] considers many fitness objectives simultaneously, such as maximizing performance and diversity. Techniques such as Pareto optimization are often used to find a set of solutions that trade-off between these objectives.

In the context of predator-prey, identifying agent behaviour can be a difficult task due to the complexity of the agents and their environment. Predator agents often exhibit a wide range of behaviours and hunting strategies such as ambushing or pursuit which make it difficult to develop a universal method for identifying and analyzing agent actions. In many cases, the predator's sensory information such as vision and hearing can vary which may further complicate the task of accurately identifying agent behaviour. Another problem with identification is that it can often be subjective and context-dependent which can make it challenging to be consistent when labeling data. It can also be time consuming and tedious to label thousands of agents [12]. To improve the identification process, a proposed solution is to automate the entire process using a convolutional neural network (CNN). The CNN can then be combined with genetic programming (GP) to help improve the agent generation process and create more unique and interesting agents.

1.2 Objectives & Motivation

The primary goal of this research is to use convolutional neural networks (CNNs) to model the diverse emergent behaviours of intelligent agents, and thus automate the recognition of agent behaviour. Previously, the act of behaviour recognition was completed manually by inputting a trace file into a viewer and analyzing the output image [12]. This method was shown to be inefficient, as it could take several minutes to generate a single trace file, view the output, and categorize the behaviour. This method was also prone to human error, as many of the behaviours are difficult to identify.

One interesting property of emergent behaviour is that a single agent can display

several classes of behaviours in a single run. Even in a simple pursuit domain environment, the predator agent could perform actions such as wall-scraping, ricocheting, and circling, among others. Agents that display such novel behaviours have shown to be high quality, effective, and therefore desirable [12].

In order to detect which agents are performing certain actions automatically, we can use the convolutional and pooling layers of a CNN to identify patterns in the data. Once recognized, these patterns can be used to efficiently score the diversity portion of a hybrid fitness-diversity score, allowing us to produce more interesting intelligent agents. Previously, pure optimization scoring using traditional GP fitness has resulted in effective agents, but with generic behaviours. Therefore, through the use of a CNN, the AI system will be able to evolve high-performing agents while focusing on those that have diverse and interesting behaviours.

This approach to behaviour recognition and intelligent agent creation has many potential applications, including in video games [52], robotics [7], and virtual assistants [34]. By using the increased accuracy and speed of the CNN model, it is possible to create more sophisticated video game characters, which enhance the user experience and overall interaction. Specifically in the domain of stealth-based video games, the model can be used to train opponents that recognize and adapt to their environment, allowing them to sneak around effortlessly, and making them more interesting to play against.

1.3 Research Questions

There are four primary research questions to be considered in this thesis:

1. **Can a CNN be trained to recognize classes of emergent behaviours?** In a predator versus prey simulation, training a CNN to recognize classes of emergent behaviours is essential to gaining a better understanding of the performance of the predator agent. This is because the predator's behaviour often determines how successful it will be during its simulated hunt. By recognizing emergent behaviours such as wall-scraping, ricocheting, and circling, among others, it is hoped that the AI model could accurately predict how successful the predator is by only looking at its trace file. This can also lead to the development of more effective predator agents that are adaptive and employ behaviours that have shown to be successful. Additionally, the CNN model can help us identify new emergent behaviours that weren't previously categorized, potentially leading to the discovery of new emergent behaviours.

2. **Can a CNN model be used to distinguish new emergent behaviours?**
   Some behaviours exhibited by predator agents can be monotonous, meaning that they are unsurprising, generic, and predictable. However, new behaviours that haven't previously been seen before have proven to be the most interesting and high-performing ones, as they can be more effective in different situations [12]. Therefore, recognizing novel patterns of behaviour that emerge as agents learn and adapt to their environment is crucial to improving the overall performance of the population. Through the use of a trained CNN model capable of detecting new emergent behaviours, we can make sure to incorporate these agents into the genetic pool for further iterations of the simulation.

3. **Can a CNN be used to score the diversity of an agent?**
   Various methods of scoring agent fitness exist, such as traditional fitness scoring that focuses on the performance of the agent or diversity-based fitness scoring that focuses on the novelty of the agent. Both these approaches help to assess the effectiveness of the predator agent, but with some caveats. As shown in [12], using a hybrid fitness-diversity score produces minimal harm to fitness and in many cases improves the overall performance of the population. Therefore, it should be possible to use a trained CNN model to help assess the effectiveness of different predator strategies by recognizing patterns of behaviour associated with successful outcomes. Specifically, unique and novel behaviours exhibited by an agent can help contribute to its diversity score. On the other hand, if the CNN model detects an agent using a strategy that has been seen before, such as one of the behaviours the CNN was trained to detect, it may indicate that the agent is less effective and needs to be modified. This type of feedback is valuable to improving the overall quality of the population, as it allows for monotonous agents with ineffective behaviours to be identified and eliminated.

4. **Can the entire process be automated?**
   Automating the entire process of creating intelligent agents with diverse emergent behaviour for a pursuit domain is important because it can significantly reduce the time and effort required to test different predator strategies. Furthermore, it would make way for significantly more interesting opponents, especially considering how long it takes modern video-games to develop opponents that mimic real-time adaption to the players [12]. This could ultimately lead to the development of more effective AI that is capable of adapting to new situations, operating in complex environments, and performing a wide range of tasks.

1.4 Thesis Structure

Chapter 2 presents a literature review of topics related to this work. Chapter 3 discusses the background reading required to understand core concepts of this research. Chapter 4 delves into the architecture and parameters of the GP system. Chapter 5 looks at the architecture and parameters of the CNN system. Chapter 6 presents the first series of experiments related to training and testing the CNN model. Chapter 7 looks at the architecture and parameters of the combined GP and CNN system. Chapter 8 presents the second series of experiments related to automatic agent generation. Chapter 9 concludes the work presented.

Chapter 2

Literature Review

This chapter provides an overview of research on intelligent agents and related areas. These topics are actively studied in the field of AI in order to improve the development of intelligent systems, especially those capable of adapting and learning from their environment. The intelligent agents section will outline various types of agents, along with some of the domains that agents may be suited for. In the pursuit domains section, applications of pursuit agents and their applications in computer games will be discussed. The section on novelty search will look into a relatively recent search and optimization approach which emphasizes the discovery of novel solutions. In the emergent behaviour section, information can be found about the type of behaviour that arises out of the interactions between parts of a system, which cannot be easily extrapolated from the behaviour of those individual parts. Lastly, the deep learning section will discuss various papers relating to classification problems and how researchers used CNNs to solve them.

2.1 Intelligent Agents

An intelligent agent is an agent that can autonomously solve a problem. In recent years, intelligent agents enable AI to help make decisions. The research area of intelligent agents is vast [22, 39, 51]. There are a multitude of search techniques that agents employ, such as A* [31, 45], minimax [15, 45], and Monte Carlo tree search (MCTS) [13, 32]. Intelligent agents can also be suited for a variety of domains, such as food gathering [27], shepherding [50], and predator-prey [12].

Intelligent agents have distinguishing characteristics which make them unique. They are able to interact with other agents (cooperatively or competitively), humans, and systems. It is also possible that goal-oriented habits may be exhibited by agents

in the form of emergent behaviour. This comes as a byproduct of agents having some level of autonomy that allows them to perform tasks and discover new information of their own.

When designing an intelligent agent, there are 5 primary categories to choose from, which are grouped based on their perceived level of intelligence [22].

- **Simple Reflex Agents** - These agents perform an action based on a simple condition-action rule. They do not look at the sensory inputs or percepts that an agent has stored in its memory over a period of time, but instead only focus on the current prompt.
- **Model-based Reflex Agents** - Model-based agents are an upgrade from simple reflex agents as they are able to look at the percept history of their actions. An internal model is used to determine an action to perform which reflects on the state of the observable environment.
- **Goal-based Agents** - Goal-based agents have a goal in mind when completing a task. These agents will use information about the goal to the best of their ability when choosing their next action.
- **Utility-based Agents** - These agents are more advanced than goal-based agents due to an extra utility measurement. The agent will work towards optimizing the expected utility before completing the goal, which in turn allows for more rational action selection.
- **Learning Agents** - Learning agents are the most complex type of agent and are built using the following elements: A learning element, a critic, a performance element, and a problem generator.

The agents to be used in this thesis are a special type of goal-based, embodied agent, that use an evolved GP controller for movement and rotation in a pursuit domain.

2.1.1 Pursuit Agents

Pursuit domains, specifically the predator versus prey environment, is a heavily researched domain that consists of two groups of agents. The predator, typically found in a smaller group and sometimes by itself, is tasked with capturing all of the prey during a simulated hunt. The prey on the other hand, try to escape the predators and

survive as long as possible. Much of the research in this field is devoted to exploring different behaviours that predator and prey agents develop in order to improve their hunting, or their survival skills.

Video games are a common place to find pursuit agents due to the high number of use cases that exist in that domain. One of the first pursuit-based video games created is Pac-Man, which is still being researched over 40 years after its creation. In [43], Pac-Man and its equally popular successor Ms. Pac-Man are referred to being the front-runners of the golden age of arcade video games. Moreover, the field of computational intelligence has seen many benefits from research into these games. The authors go into detail about different strategy guides that were published in the early days and the theoretical meaning behind the game. The authors also mention several spin-off implementations of the game which are widely used across academia and what makes Pac-Man such a target for research. An example of these traits is that Pac-Man simply poses a variety of challenges to overcome. The game field can be represented by using undirected connected graphs and therefore graph theorists began looking into how to solve the game. Another reason that Pac-Man is so difficult for an intelligent agent to learn is that the game is played in real-time. Lastly, the authors mention that even primates have been given the task to learn how to play Pac-Man due to its controller being a simple four-way joystick.

An example of some of the algorithms used in Pac-Man research include: rule-based algorithms, tree search and Monte Carlo, evolutionary algorithms, artificial neural networks, neuro-evolutionary approaches, reinforcement learning, and several other approaches such as ant colony optimization. Overall, this study highlights Pac-Man's significance in computational intelligence by detailing the wide variety of research that exists not only in computer science, but also in fields such as psychology, sociology, robotics, and biology. In conclusion, the most dominating forces to play Pac-Man are both rule-based approaches and MCTS.

Another domain in which pursuit agents can be found is in stealth-based games. These games involve scenarios such as navigating around buildings, walking through dense forests, and sneaking quietly behind enemy lines to avoid detection. Sometimes, the enemy gains sight of you for a brief moment, and you need to quickly retreat to a safe location before attempting your mission again. The non-playable character (NPC) enemies that observe you engage in a type of real-time pursuit, and they need to be effective. It's common practice for these enemies to start searching for you in your last known location, before looking in obvious hiding spots. In order to more efficiently search for opponents, researchers have developed a novel search method

called skeleton search [1], which is effective in hide-and-seek and pursuit/evasion domains.

The method of skeleton search involves using a navigation mesh for each agent to walk on. By doing so, agents are able to move around freely to avoid detection, rather than being limited to a set path. In order to calculate the position of an opponent who is freely moving around, a numerical value is propagated through the topology graph which represents the probability the opponent is at a certain position. The topology graph itself is based on a simplified representation of the map and is created using scale axis transform (SAT). The graph is then discretized by dividing each edge of the graph into smaller edges with a fixed length, referred to as segments.

Experiments were conducted by recreating maps from popular commercial games, with categories being first-person shooter, action/stealth, and role-playing games. Search behaviours were narrowed down into three categories: cheating (guards knowing positions at all time), probability propagation, and probability diffuse. Other parameters are measured such as the radius of the guards field of view, the number of guards actively searching for an opponent, and the opponent's speed. Overall search performance showed that the probability propagation method performed best in most maps, even exceeding their cheating upper-bound. Furthermore, this work showed that NPCs who show rational, intuitive behaviour can be more interesting to play against [1].

2.1.2 Emergent Behaviour

Emergent behaviour refers to behaviour of a system that arises out of the interactions between components of a system which cannot be easily predicted by the designer of those individual parts. In essence, the whole is greater than the sum of the parts.

In [26], Koza applies genetic programming to a problem of behaviour ecology in biology. In the simulation, a type of lizard known as the Caribbean Anolis lizard is required to forage for food. It is up to the lizard to determine the most optimal food foraging strategy. In the real world, these anoles are sit-and-wait predators which typically perch with their head down on tree trunks and scan the ground for desirable insects to eat [44]. The question arises as to what the most optimal way for the lizard to hunt insects is. If there are an abundance of insects, the lizard would have great success with hunting any nearby insects. However, if the lizard leaves its perch to hunt for distant insects, it may lose the possibility of hunting for a greater number of nearby insects. This suggests ignoring distant insects [26]. In the first version of the

problem, the lizard always catches the insect if the lizard decides to chase the insect. It was shown that an average of 132,000 individuals were evaluated in order to solve this problem with a 99% success rate. In the second version of the problem, the lizard does not necessarily catch the insect at the location where it saw the insect. It was shown that the lizard was able to learn to ignore insects it sees in the first region, and that it should chase insects it sees in the second region. This is an example of an emergent behaviour.

In [20], Grossi investigates how GP can be effectively used in a multi-agent system to enable agents to learn and communicate in pursuit domains. This study continues where previous studies have left off, by searching to evolve emergent behaviours in a multi-agent system. The pursuit domain uses multiple predators who are tasked with learning about a simple set of commands they are given, and using them to find and follow the prey. The predator agents were given limited knowledge of their environment in these experiments, this way their behaviour could be emerged from little to no information. Not only do they not know the location of the prey they are trying to capture, but the predators also do not know the location of other predators. The only information the predators have access to is the relative direction of other predators and the ability to send messages to each other. Results of her experiments determine that emergent behaviour occurs in both scenarios, enabling a high degree of communication. The most prevalent behaviour found was a message sending pattern which emerged from predators sending simple messages between one another. Eventually, this simple message sending became a synchronized alternative message sending pattern which enabled the predators to better track down the prey.

2.2 Novelty Search

Novelty search is an advanced search technique used to balance the perfect combination of exploring a search space to find new solutions, and exploiting a specific portion of the search space in order to improve upon already strong performing solutions [30]. This is known as the *exploration-exploitation trade-off*. The primary goal of novelty search is to find solutions that may not be found by traditional search techniques that focus heavily on fitness. In its purest form, novelty search uses novelty as 100% of the fitness metric, without caring about the exploitation aspect at all.

Lehman and Stanley show that novelty search has significantly outperformed objective-based search techniques in two problem tasks, without focusing on the objective at all [30]. The first domain explored was in a deceptive maze experiment. In

this type of domain, traditional fitness-based approaches have a difficult time escaping due to repeatedly being deceived by solutions that look like they are going to let them escape. With a novelty-based approach, new agents continuously explore the landscape until an exit is found.

The second domain explored was in a bipedal robot experiment. The purpose of this experiment was to use a well-known problem with high difficulty that is not suited towards novelty, but is rather challenging to solve in general. When averaged over 50 runs, the novelty-based controllers travelled 4.04 meters on average whereas the fitness-based approach only travelled 2.88 meters on average. As a conclusion, this research has shown that novelty search shows what is left if the pressure to achieve the objective is abandoned and that objective-driven search has its limits.

In [12], Cowan explores different methods of evolving predator behaviours through the use of diversity search, while maintaining a high level of efficacy. Cowan's motivations came from diversified behaviours having the potential to improve a wide range of applications, like art, architecture, virtual robotics, and so on [19]. The conclusions showed that the nature of fitness-based evaluation tends to exploit particular behaviours in monotonous and uninteresting ways, but with the addition of diversity-based evaluation, this is remedied [12]. The best results achieved in this paper were the k-nearest neighbours with archive (KNNA) approach with 50/50 split of fitness-diversity score (50F/50D) which was proved in more than one domain.

The Multi-dimensional Archive of Phentotypic Elites (MAP-Elites) algorithm [36] is an effective tool for finding elite solutions with diverse behaviour, but it has its limitations when exploring high-dimensional feature spaces. In Pozzuoli's work [41], an Age-Layered MAP-Elites (ALME) algorithm is proposed which addresses this limitation by separating the population into age layers with their own feature spaces, enabling the algorithm to increase the number of behaviours without exponential growth of the solution space. The algorithm was applied to a pursuit domain task, with results showing that ALME was able to find a high diversity of solutions with no fitness-diversity trade-off. Furthermore, the algorithm was capable of adding more features without adding more individuals. The primary drawback of this algorithm was that not all regions of the feature space were able to produce useful individuals, thus manually searching for those solutions was a time consuming task.

2.3 Deep Learning

Behaviour classification involves analyzing and categorizing the many behaviours that characterize intelligent agents. These behaviours can be classified manually by humans, or by using artificial intelligence to automate the process.

In [3], Asadi *et al.* use a supervised feed-forward neural network as an intelligent classifier for their system. By training the model using a supervised set of data, the framework is able to learn the information from the environment and its behaviour is recognized by the weights of the neural network. Due to the complexity of the supervised multi-layer neural network (SMNN), a suitable data pre-processing technique is required to find valid input values along with pre-training techniques to find desirable weights. By doing so, the training process can be reduced. In this case, potential weights linear analysis (PWLA) is used. PWLA is made up of three steps: (1) data pre-processing involving the normalization of input values, (2) pre-training by setting each weight equivalent to the sum of all absolute normal values in each instance, and (3) dimension reduction through hidden layer pruning. By using the SMNN model, it was shown that the classification system has capability of predicting future status and showing complex relations between components of systems.

Another type of deep learning algorithm that can be used for classification problems is the convolutional neural network (CNN). One example is a machine learning pipeline called DeepEthogram [5] that was developed by Bohnslav *et al.* to classify behaviour from raw pixels. Using this pipeline, videos of animal behaviour were used to study neural function, gene mutations, and pharmacological therapies. The overall architecture involves three stages: (1) estimating motion from a small snippet of video frames; (2) compressing a snippet of optic flow and individual still images into a lower dimensional set of features; and (3) using a sequence of the compressed features to estimate the probability of each behaviour at each frame in a video. To test the performance of their model, nine difference neuroscience research sets were used that include two species and present several challenges for computer vision algorithms. When trained with only 80 example frames of a given behaviour, the model performed with more than a 90% accuracy. Similarly, the model was able to predict with over 95% accuracy using only 100 example frames. Therefore, DeepEthogram models require little training to achieve high performance [5].

Chapter 3

Background Reading

This chapter provides the reader with background information required to understand the fundamentals of two machine learning algorithms used throughout this thesis. Genetic programming is an evolutionary algorithm based on Darwinian evolution that focuses on evolving a population of computer programs. A convolutional neural network is a deep learning algorithm commonly used to classify images. In this chapter, both algorithms will be described in detail, along with their components.

3.1 Genetic Programming

In 1988, nearly 30 years after researchers first experimented with machine evolution, John Koza patented his invention of a genetic algorithm (GA) for program evolution [24]. This patent would serve as the foundation for a new branch of evolutionary computation called genetic programming (GP), a process of evolving entire computer programs. In [25], Koza describes computer programs as some of the most complex structures created by man. The purpose of this literature was to answer an important question in computer science: *How can computers learn to solve problems without being explicitly programmed?*

3.1.1 Representation

To answer this question, we need to take a look at how GPs are represented. It is much the same as a traditional GA, except the flat data structure of a vector is replaced by a variable-length, tree-based structure. Another key difference is that instead of a node representing a data point, it represents a function. The functions could be mathematical operators such as add, subtract, or modulo. There are also

more complex functions that could be used, depending on the problem space. The children of these functions are then recursively computed until a terminal state is reached.

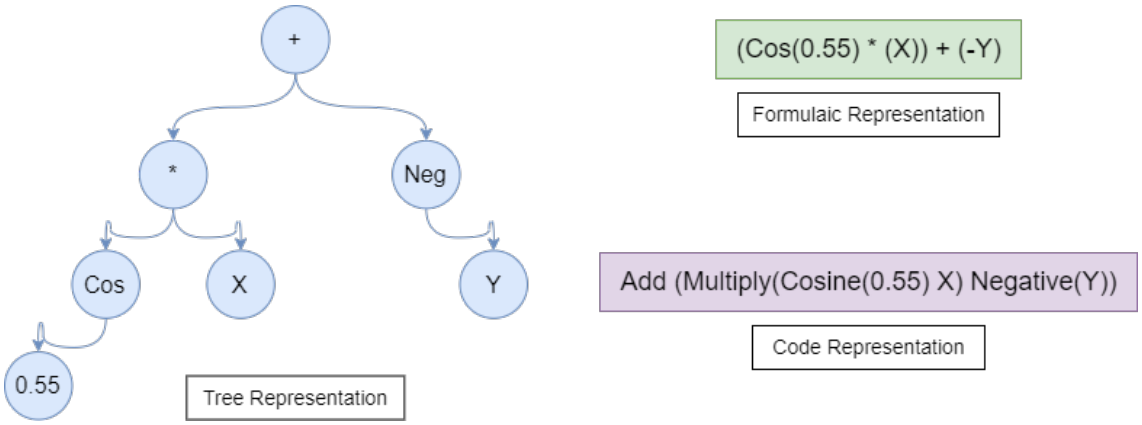


Figure 3.1: Genetic Program Representations

3.1.2 Initialization

The first parameter that must be determined before initialization is the population size. The population size refers to the number of individuals in the initial population, and can vary depending on the problem space. By choosing a large population size, the diversity of solutions will increase alongside computational costs. The inverse is also true, meaning a smaller population size will result in less diverse solutions and a decrease in computational costs.

During the initialization phase, the population is created by generating random individuals using a selection of randomly selected GP functions and terminals. Functions in GP represent operations that can be performed on input values, whereas terminals represent input values or constants that are combined with functions to create candidate solutions. These individuals are typically very poor performing due to the random nature of their actions and are not expected to generate optimal solutions. This randomness may continue into the first few iterations of evolution, but as the population is evaluated and the best individuals are selected for the next generation, their performance starts to improve.

3.1.3 Evaluation

The evaluation phase is a critical component of a GP, as it allows for the program to assess the quality of candidate solutions in the current generation. The goal of this

phase is to determine how well each individual solves the problem which the program is trying to address. To do this, a fitness function must first be defined.

A fitness function is created in order to quantify how well each individual solves the problem. Depending on the problem, the structure of a fitness function may vary significantly. Some fitness functions focus on minimizing their score, such as how long it took an individual to find the exit to a maze. Other fitness functions look to maximize their score, such as how many images it correctly identified in a classification problem.

It can be difficult to find the most optimal fitness function for given problem, which is why researchers go to great lengths to try different strategies of evaluating GPs. The most common method of evaluating individuals is to use a traditional fitness function based on how well the individual's solution represents a solution to the problem being addressed. In other words, the better the individual performs, the better the individual's solution is.

Other methods of fitness evaluation exist, such as novelty-based evaluation seen in [12] which measures the fitness of an individual based on a combination of traditional fitness and an individual's diversity when compared to its neighbouring solutions. Once the fitness function is defined, the evaluation process runs the fitness function over each individual in the population to determine their scores.

3.1.4 Selection

Once the individuals in the current population have been evaluated, the next step is to select individuals for the next generation based on their fitness scores. A variety of selection techniques can be chosen, such as tournament selection or roulette wheel selection [18]. The goal with the selection process is to select high performing individuals for the next population in order to gradually improve performance over time.

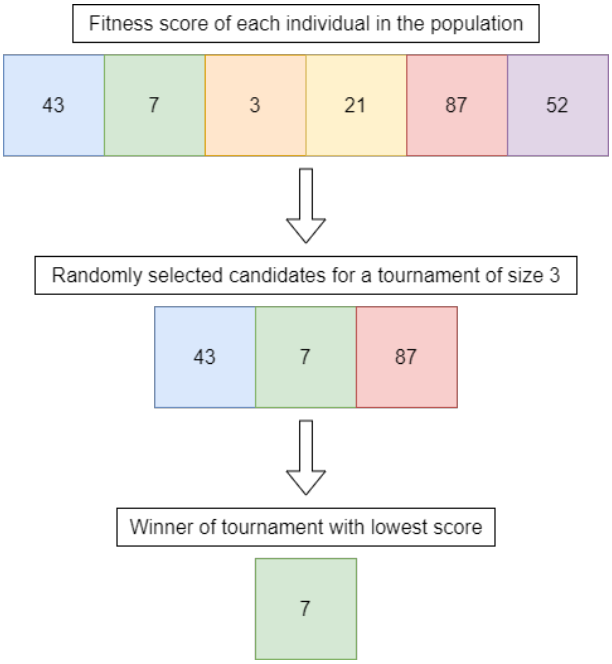


Figure 3.2: GP Tournament Selection

Tournament selection is the most popular selection method [18]. The first step is to select K random individuals from the current population. Once the competitors are selected, the individual with the best fitness score is selected for the reproduction stage as a parent.

3.1.5 Reproduction

The reproduction phase allows GP to create a new population of individuals based on those chosen during the selection process. Once the two parent solutions have been selected, the genetic material between them are combined using a method known as crossover.

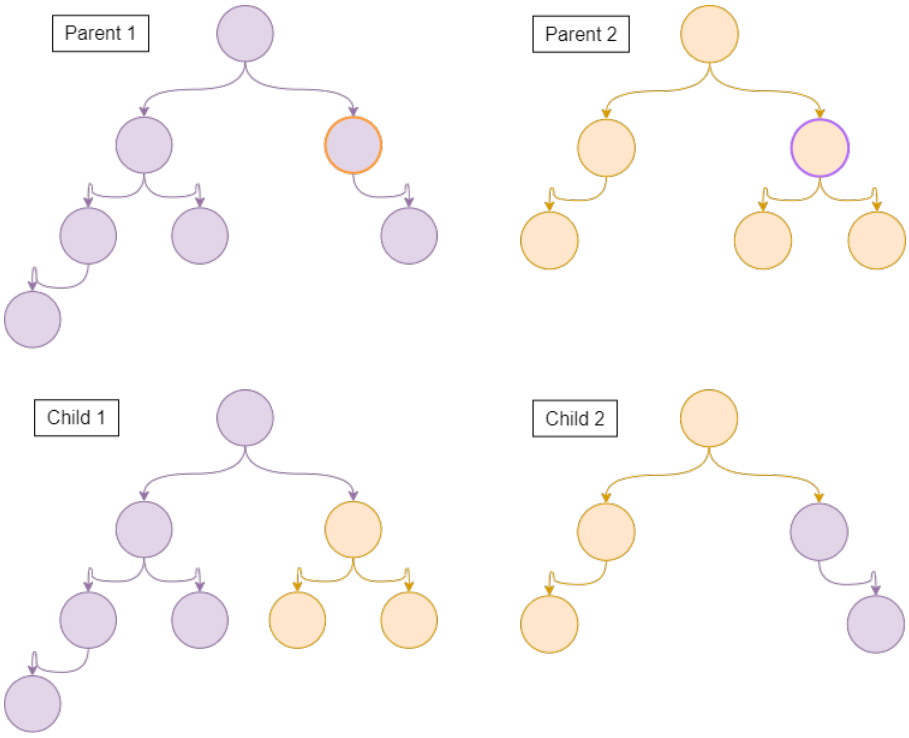


Figure 3.3: GP Crossover

Crossover involves choosing random branches from each of the parents trees and swapping them in order to create one or more offspring. Due to the nature of GP solutions, its important to make sure that the genetic material is compatible on both trees to prevent any type mismatches.

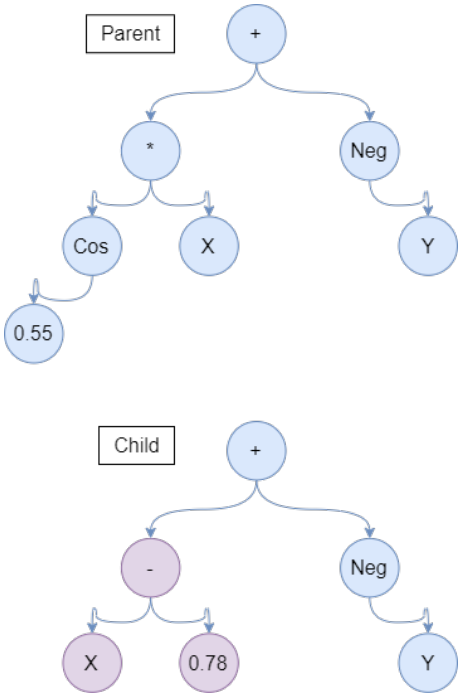


Figure 3.4: GP Mutation

Once crossover has been performed, there is a small chance that the offspring will be subjected to a mutation. The mutation operator introduces a random change to the genetic material of an individual, typically in the form of a removal, addition, or change of a single component. This process allows for new genetic material to enter the population and expand upon the search space. Mutation is typically applied with a low probability in order to avoid drastic changes to the offspring.

3.1.6 Termination

The termination condition is used to stop the evolutionary process when a solution that satisfies the problem has been found. Common termination conditions include reaching a maximum number of generations or when the fitness score of an individual reaches a threshold. Once the termination condition has been met, the GP will terminate.

3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of deep neural network made up of several interconnecting layers which differ from a traditional feed-forward neural network (FFNN) [17]. CNNs are widely used in computer vision tasks such as image

classification and recognition, due to their ability to learn and extract features from images.

3.2.1 Convolutional Layers

The core building block of a CNN is the convolutional layer. In this layer, a set of filters are applied to the input image. The filters work by performing a convolution operation which then produces a set of output feature maps. The filters are represented by a small matrix, typically between 2x2 and 5x5, which slides across the image and computes the dot product of the pixel values found in its location.

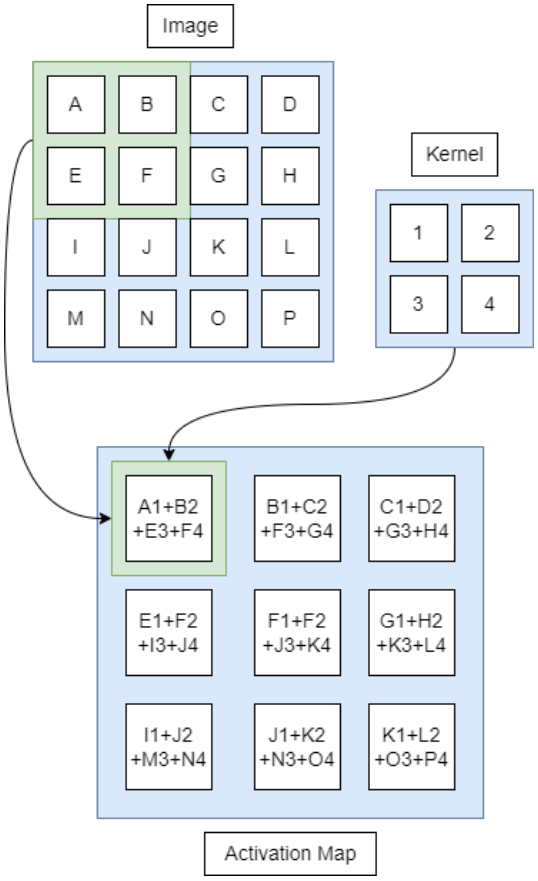


Figure 3.5: Convolutional Layer

Through the use of filters, different patterns and features in the image can be captured, such as edges and shapes. As more convolut ional layers are added, more detailed patterns can be captured, such as faces and textures. The number of filters can vary, depending on how many convolutional layers are needed. Other parameters include stride and padding, which work to control spacing between the filters and add extra pixels to the input image in order to maintain the output size.

3.2.2 Pooling Layers

A pooling layer is typically found after each convolutional layer. These layers work to reduce the spatial size of the convolved feature, while retaining the most important features. The two main types of pooling are max pooling and average pooling. Max pooling works by returning the maximum value from the portion of the image traversed by the kernel at each section, and discards the rest. This means that much of the valuable information is discarded during the early stages of the network [37]. On the other hand, average pooling returns the average of all values, which helps to give the average of all features present in a kernel.

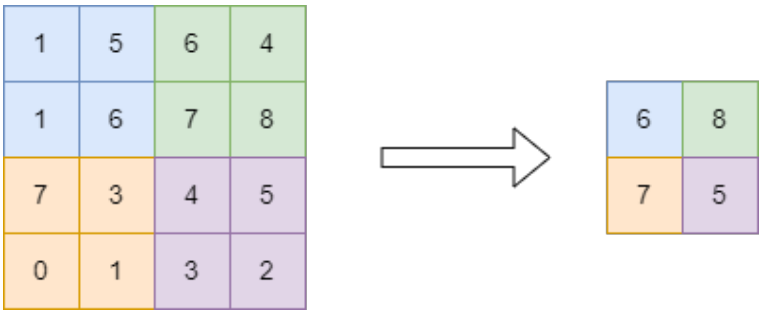


Figure 3.6: Max Pooling Layer (Filter = 2x2, Stride = 2)

3.2.3 Dense Layers

After the data has traversed through several convolutional and pooling layers, it will go through a series of fully-connected layers called dense layers. In this layer, every neuron is connected to all activations from the previous layer. Once the dense layers are traversed through, a final output layer allows the CNN to give its prediction. The number of neurons used in each dense layer depends on how complex the problem is, and what the size is of the previous layer. Optimization techniques such as backpropagation are used during training in order to reduce the error between the predicted and actual output. For classification tasks, a softmax activation function is typically used to give a prediction in the final dense layer.

3.2.4 Dropout Layers

In 2012, Geoffrey Hinton introduced dropout, an efficient way to prevent overfitting of neural networks [49]. Overfitting occurs when there is not enough training data for the CNN to learn from, and thus the model is unable to generalize and fits itself closely to the training data instead. By using dropout, the CNN is encouraged to

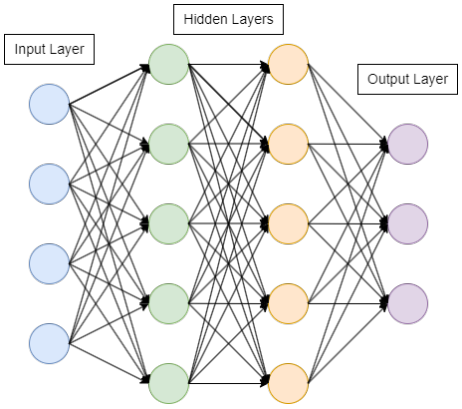


Figure 3.7: Fully-connected Layer

learn more about the features of the training data. During training, the dropout layer randomly drops out a portion of the inputs, forcing the network to adapt to different subsets of connections. When the CNN is finished training, the connections are turned back on for testing.

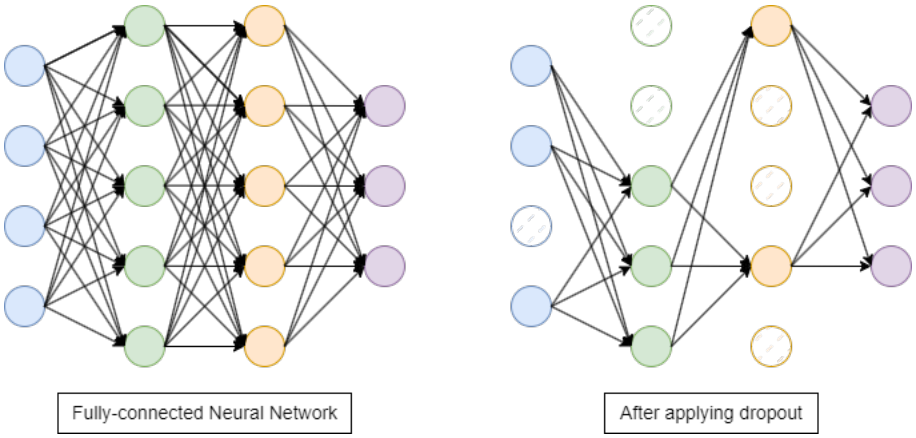


Figure 3.8: Dropout Layer (Left: All neurons are connected. Right: Crossed neurons have been dropped)

Chapter 4

Pursuit Domain Simulation

The first of 3 systems to be discussed in this thesis is the pursuit domain simulation. In this simulation, an intelligent predator agent is given the task of capturing a set number of prey agents randomly scattered across the environment. A fitness score tallies the number of prey the predator caught during its hunt, and is used to determine who moves forward in the evolutionary process. As shown in [42, 12, 20], the predator-prey pursuit problem is an appropriate test bed for multi-agent systems. The goal of the pursuit domain simulation is to generate thousands of trace files which depict the predator's movement and behaviour. These trace files will then be stored in a trace database and later used for CNN experiments in Chapter 6.

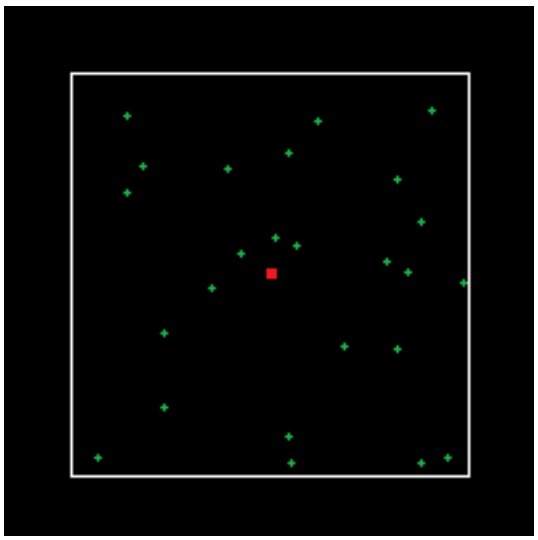


Figure 4.1: Simulation Overview (red square = predator, green crosses = prey)

The simulation is inspired by the work of Cowan [12] and Pozzuoli [41]. An image of the environment can be found in Figure 4.1, which depicts what a typical seed

looks like before any of the agents have moved. The environment is set to a size of 200 units by 200 units, with agents being able to freely move around in continuous, floating-point action space. The red square represents the predator agent, who always starts in the center of the environment and faces a random direction between -2π and 2π. At the start of the simulation, all 25 prey agents are randomly placed inside the environment, as indicated by green symbols. A boundary of 25 units is put in place during initialization phase, which prevents agents from being initialized too close to the walls of the environment. After all agents are initialized, they are free to move past the white boundary square towards the edges.

4.1 Predator & Prey Controllers

The predator and prey agents store 4 crucial pieces of information about themselves which are used to guide them through the simulation. The most important piece of information is the agent's (x, y) coordinates, which represents their real-time position in the simulation. The position is calculated at the beginning of each step and is saved after each move, allowing for the later recreation of text-based and image-based trace files. The next piece of information is the predator's orientation, which works in congruence with the predator's vision cone and sensing radius. The GP tree is evaluated at every step and returns a floating-point value. This value is then added to the predator's current orientation which rotates the predator in a new direction. Giving the predator a sense of direction allows for dynamic movement around the environment and an improved ability to locate prey. The final piece of information used to guide the agents through the simulation is their speed. This is where the predator has a major advantage over the prey, due to the fact that the prey have a locked speed at 0.7 units per move. When initialized, the predator has a speed of 1.0 units per move, but this can be increased up to 5.0 or decreased to 0.5 through use of the *SetSpeed* function.

Two common approaches were considered when designing the movement of each agent. The first approach was to limit the agents' movement to four cardinal directions on a grid, creating a discrete environment with strict instructions. This approach would be easy to implement, as it has similarities to the artificial ant problem [10]. However, this would limit the diversity in agent behaviour significantly as the agents would only be able to move in four directions. The second approach was to allow the agents to move in floating-point action space with limited restrictions. Due to this research focusing heavily on diverse emergent behaviours, choosing the continuous

action space with limited restrictions was the obvious choice. By doing so, agents possess the ability to rotate and move in any direction, allowing for patterns such as circling and ricocheting to become more abundant.

Once the agents are initialized, the predator moves in the direction they are facing by multiplying their orientation by speed, and adding it to their current position. The result is a new set of (x, y) coordinates which is where the predator will sit and wait while the prey make their moves. The prey agents use a predefined approach to movement, allowing for more consistent experimentation within the environment. Once the predator is sitting and waiting, each of the 25 prey move 0.7 units in the direction they are facing. Their orientation is fixed, but once each prey have moved in their current direction 5 times, they all rotate to a new, unique orientation between -2π and 2π.

If a predator finds themselves within 1 unit of a prey, the prey becomes captured and are added to the predator's fitness score. Once a prey is captured, it is no longer able to move or be re-captured by the predator. If any agent is found stuck in one of the four corners of the environment, they are rotated to a new direction between -2π and 2π. This cycle continues until 5000 steps are made by all agents.

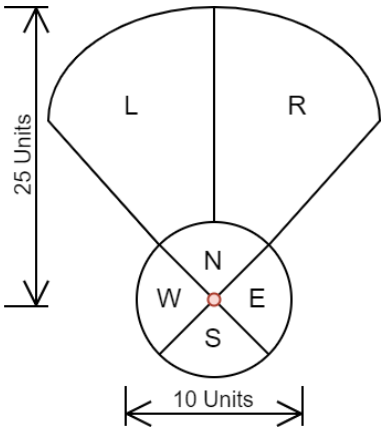


Figure 4.2: Predator Perception

Figure 4.2 shows the perception of a predator agent who has the ability to seek and sense in every direction. This combination of sensors is not always guaranteed, as predator agents can differ from one another in many ways depending on their GP controller. This is why as the predator evolves, so does their perception. A predator may not develop all of its functionality until later in the evolutionary process, or even at all, which allows for diverse and interesting behaviour to emerge out of uncertainty.

4.2 GP Architecture

A GP language is used to evolve the controller of the predator agent. ECJ 27 [33], a Java-based [2] evolutionary computation library is used for implementation of the GP system. Various parameters were tested during the discovery phase of this research, and the following were chosen due to their use in related literature. The primary criteria used to determine the GP parameters was to allow for diverse emergent behaviours to be abundant, and to limit the success of the predator within reason. This is why the predator's sensors and speed are clamped, as the predator must develop its own strategies for catching as many prey as possible.

Parameter	**Value**
Generations	50
Number of Steps	5000
Population Size	500
Tournament Size	2
Number of Elites	0
Maximum Tree Depth	15
Crossover Chance	90%
Mutation Chance	10%
Growth Algorithm	Half Builder
Growth Chance	50%
Initial Depth	2 to 6

Table 4.1: GP Parameters

Table 4.1 provides a breakdown of the various parameters used in the GP system. The number of generations is 50 which represents how many times the predator's tree has a chance to crossover or mutate with another parent. The number of steps is 5000 which represents how many moves a predator makes in its environment. During these moves, each predator is allocated a new (x, y) coordinate and orientation, as well as a potential update to its speed. The population size is 500 which represents how many predators are competing against one another. In order to compete, tournament selection chooses 2 random agents in the population and the highest fitness score is chosen for the next stage in the evolutionary process. After two agents are chosen, they have a 90% chance to crossover and a 10% chance to mutate.

GP Function	**Description**
Add (A, B)	Return A + B
ASin (A)	Return arcsin(A)
Divide (A, B)	If B $\neq$ 0 return A $\div$ B, else return 0
GreaterThan (A, B, C, D)	If A $<$ B return C, else return D
Multiply (A, B)	Return A $\cdot$ B
Negative (A)	Return $-$A
Sin (A)	Return sin(A)
Subtract (A, B)	Return A $-$ B

Table 4.2: Math Functions

The math functions in the GP language are summarized in Table 4.2. Functions with one child include: *ASin* which returns the arcsin of the value, *Negative* which flips the sign of a value, and *Sin* which returns the sin of the value. Functions with two children include: *Add* which adds the result of two children and returns the value, *Divide* which safely divides the result of two children and returns the value, *Multiply* which multiplies the result of two children and returns the value, and *Subtract* which subtracts the result of one child from the other and returns the value. Lastly, the *GreaterThan* function executes the first two children. If the result of executing the first child branch is greater than the result of executing the second child branch, then the third child branch is executed and the value is returned. If the result of executing the second child branch is greater than the result of executing the first child branch or they are equal, the fourth child branch is executed and the value is returned.

GP Function	**Description**
SetSpeed (A)	Set predator speed to [0.5 <= A <= 5.0] and return A

Table 4.3: Speed Function

The *SetSpeed* function shown in Table 4.3 is one of the key functions that allow the predator agent to gain an advantage over the prey agents. The prey agents have their speed locked at 0.7 units per move, whereas with the *SetSpeed* function, the predator is able to adjust their speed between a set of boundaries as needed. The function executes the child branch and receives the float value which is then converted into the predator's speed attribute. The speed value is locked between 0.5 units and 5.0 units, so as to not give the predator too much of an advantage. This function

allows the predator make dynamic movements, such as turning around corners slowly or chasing after prey.

Terminal	**Description**
MovesRemaining	The number of moves remaining in the current run
MovesTaken	The number of moves taken in the current run
Orientation	The current orientation of the predator
PreyCaptured	The number of prey captured in the current run
PreyRemaining	The number of prey remaining in the current run

Table 4.4: Terminals

Environment operands shown in Table 4.4 are terminals that change value as the simulation is running. These changing values allow the predator agent to behave differently as prey get captured and moves are taken. For instance, the first two terminals *MovesRemaining* and *MovesTaken* return a normalized value representing how many moves the predator has left, or has taken. Next, *Orientation* allows for the predator's orientation to be used in the GP language in other functions. Lastly, *PreyCaptured* and *PreyRemaining* return a normalized value representing how many prey are remaining, or have been captured.

Terminal	**Description**
Ephemeral	Random constant between [-1, 1]
Pi	The number π

Table 4.5: Constants

The next set of GP terminals are constant values seen in Table 4.5. These terminals store information like the environment variables but remain static across all tree executions after initialization. The *Ephemeral* operand generates a random number between -1 and 1 and remains constant throughout the evolutionary process. However, other individuals who have ephemeral random constants in their GP tree may have other static values they use instead. The second operand is *Pi* which represents the value of π in mathematics.

GP Function	Description
SeekPreyLeft (A, B)	If prey in left vision radius return A, else return B
SeekPreyRight (A, B)	If prey in right vision radius return A, else return B
SeekWall (A, B)	If wall in vision radius return A, else return B
SensePreyNorth (A, B)	If prey in north sensing radius return A, else return B
SensePreyEast (A, B)	If prey in east sensing radius return A, else return B
SensePreyWest (A, B)	If prey in west sensing radius return A, else return B
SensePreySouth (A, B)	If prey in south sensing radius return A, else return B
TouchingWall (A, B)	If prey touching wall return A, else return B

Table 4.6: If Statements

Table 4.6 outlines the various conditional statements that represent the predator's simulated vision and senses. Each of the functions have 2 children and work by executing the first branch if the conditional statement is true, or the second branch if the conditional statement is false. By using these conditional statements, a multitude of behaviours can emerge as the predator learns to adapt to seeking or sensing prey and walls.

4.3 Generating Training Data

The GP system generates data files and trace files after each run. The data files store information about each generation such as the population's average fitness, the best found agent's fitness, and the GP tree of the best found agent. The trace files on the other hand, are used to evaluate the emergent behaviour of the best found agent.

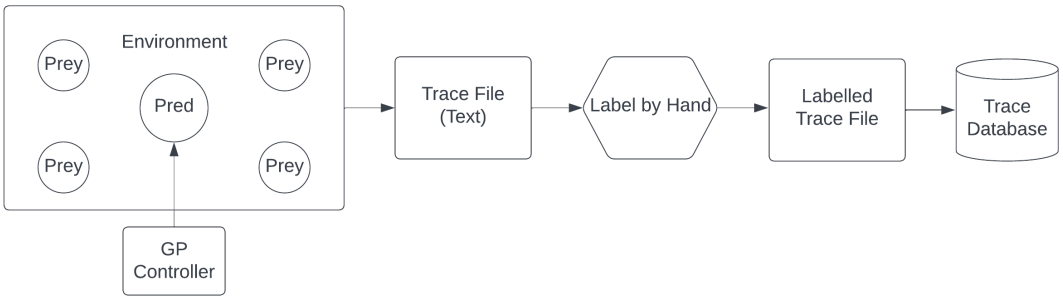


Figure 4.3: Simulation Flow Chart

Figure 4.3 outlines the flow of the GP-based trace generation system from left to right. The first step is to run the GP system until all 50 generations have completed.

At the end of execution, the best predator agent is isolated and performs 30 simulated hunts. Each of these simulated hunts will store the (x, y) coordinates of the predator at each step and output it as a text-based trace file. These trace files are then manually analyzed for behaviour qualities and labelled by hand. Once all 30 traces are labelled by their primary emergent behaviour, they are put into the trace database for later use. The trace database is simply a series of folders which store .txt based files.

```
Predator: -46.231747, 100.69174, 99.277855
Predator: -78.84808, 100.885155, 98.29674
Predator: -77.92871, 101.094284, 97.31885
Predator: -77.00933, 101.31908, 96.344444
Predator: -76.08995, 101.55947, 95.37377
Predator: -75.17058, 101.815414, 94.40708
Predator: -74.2512, 102.08684, 93.44462
Predator: -73.331825, 102.373665, 92.48663
Predator: -72.412445, 102.67583, 91.53338
Predator: -71.493065, 102.99325, 90.58509
...
```

Figure 4.4: A Text-based Trace File

Figure 4.4 shows what a text-based trace file looks like. The first numerical column represents the predator's orientation between -2π and 2π. The next two columns represent the (x, y) coordinate of the predator.

Chapter 5

CNN for Emergent Behaviour Classification

The second system to be discussed in this thesis is the emergent behaviour classifier. The CNN is trained on thousands of images depicting the predator agent's behaviour as it traverses through the simulation. In [12], the process of manually categorizing the behaviour of each agent individually was found to be highly monotonous and prone to human error. To remedy this, a CNN is trained to automate the classification of emergent behaviour with a low margin of error and limited training data.

5.1 Trace Files

A trace file refers to a type of file that is readable by humans or computers and contains representations of various events, such as the execution of actions and other relevant data points. Text-based trace files are easy for computers to interpret which makes them a popular choice for small to medium sized feed-forward neural networks when only a limited amount of data needs to be stored. On the other hand, image-based trace files are easy for humans to interpret and represent the data visually in the form of an image. These trace files can show a more detailed overview of a run which can include heat maps, gradients, and other simultaneous layers of data. Moreover, this format of trace is concise and allows for a larger amount of data to be stored in a compact visual form. It is also the input of choice for CNNs.

There were several considerations that had to be made when choosing the kind of trace to be used in each experiment. The first was to figure out which type allows for the easiest analysis and classification of each predator's behaviour. For complex models with large-scale data, visualizations can be more effective than inspecting raw

text. However, each predator only exhibits a few thousand steps per run, so it wasn't obvious if images were necessary during preliminary stages. The next consideration was to understand how a trace may be interacted with. For instance, text-based traces could technically be read by humans but it would make much more sense to feed it into a feed-forward neural network for analysis. On the other hand, the use of image-based traces allows for more interactive exploration of the data, such as zooming, panning, or even the possibility to interact with visual elements which can enhance the analysis process. Ultimately, it was found that image-based traces would allow for easier labelling of the data, saving countless hours. Lastly, it was important to determine which types of files would be most suitable for presentations and reports. For this the answer was also image-based trace files, as they are generally easier to understand and interpret compared to raw text. Even those not familiar with the system can still gauge how the predator is performing and what behaviours they exhibit.

In [12], a predator versus prey environment was used to conduct a series of experiments to determine which evolution strategies were best at evolving high-performing diverse predator agents. The method for analyzing the agents was a slow, manual process, and although the author found a high degree of success in their experiments, it should be possible to automatically categorize the behaviours of intelligent agents with no metrics or human interference using a CNN. To automatically categorize the behaviour of an intelligent predator agent, a trace file that models its movement over time is necessary. With a single trace file, insights about the agent's decision making process, behaviour, and performance can be found. The most important insight is the movement patterns of the agent. These patterns can be used to determine if the agent exhibits specific strategies, such as exploring the environment in a systematic manner or by favouring specific regions. They can also be used to determine the type of behaviour the predator is exhibiting, such as ricocheting off walls, rotating in a circle, or locking onto prey and pursuing them. The exploration versus exploitation tradeoff is another valuable insight which can be seen in a trace file. This tradeoff looks at balancing the need for the predator to discover new areas of its environment versus exploiting the knowledge it already has in order to capture more prey.

Learning and adaptation, which looks into how the predator adapts its movement based on information it gathers during its run, is also a valuable insight. The predator can use information about how many steps it has left or how many prey remain to dynamically change their approach. For instance, when the predator is low on moves, they may choose to increase their speed to further explore the environment with the

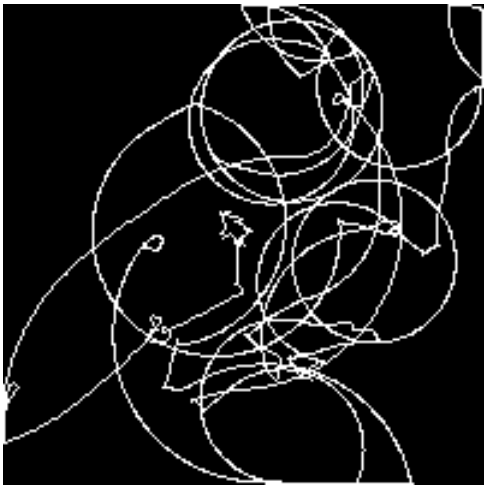
possibility of finding those last few hard to find prey. Obstacle avoidance is another crucial aspect of the agents performance. Although there are no obstacles in the middle of the environment, the predator will still need to avoid getting stuck on walls by using its perception to see when a wall is in their view. Efficiency is also a critical aspect to evaluate the agent's movement, especially due to their limited number of moves. If an agent is able to find optimal paths to each prey, there is a high chance that it will be selected for evolution in the next generation. Agents that continuously optimize their routes and improve are seen as the most valuable and intelligent type of agents when evaluating them using traditional fitness measurements. In order to generate traces once the predator has completed their simulated hunt, a trace image generator program is introduced.

5.2 Trace Image Generator

The trace image generator is a program that was developed in congruence with the emergent behaviour classifier program that converts text-based trace files into image-based trace files that are ready for CNN classification. The system was created using the Python 3.7 programming language [48] and Pillow 9.4 library [11].

```
Predator: -172.23602, 98.72479, 99.82613
Predator: -121.54169, 98.05154, 98.72927
Predator: -31.541685, 99.1484, 98.056015
Predator: 75.20681, 99.47701, 99.300354
Predator: 125.90114, 98.72233, 100.342865
Predator: 176.59547, 97.4376, 100.4193
Predator: 227.2898, 96.56464, 99.47362
Predator: 277.9841, 96.7434, 98.19909
Predator: 7.984119, 98.01793, 98.37785
Predator: 277.9841, 98.19669, 97.103325
...
```

(a) Text-based Trace (Excerpt)



(b) Image-based Trace

Figure 5.1: Text to Image Conversion

Figure 5.1 shows a portion of an original text-based trace file on the left, and its resulting image on the right. The program opens the text file as parses it line by line. Once a line is parsed, it is then split into four parts: the individual's name (in this

case, the predator), the orientation of the agent, the agent's x coordinate, and the agent's y coordinate.

A key feature of this system is that both types of trace files can be flipped along a horizontal or vertical axis and rotated in any direction, in order to create more entries for the data base. By doing so, a single text-based trace can now produce 8 different images. This means a single run of the GP system will generate 30 text-based traces and create up to 240 image-based traces.

5.3 CNN Architecture

The primary objective in this section is to create a CNN model that can accurately categorize the behaviours of intelligent predator agents during the pursuit of prey. By achieving this, the model will help with understanding the dynamics of the predators' decision making and adaptive strategies in their simulated environment. Successful implementation would mean that the model is capable of accurately detecting up to six predetermined emergent behaviours exhibited by the predator agents. Furthermore, the model could be extended to other applications where understanding predator-prey dynamics is crucial, such as biology or ecology. The system was created using the Python 3.7 programming language [48] and TensorFlow 2.12 library [16].

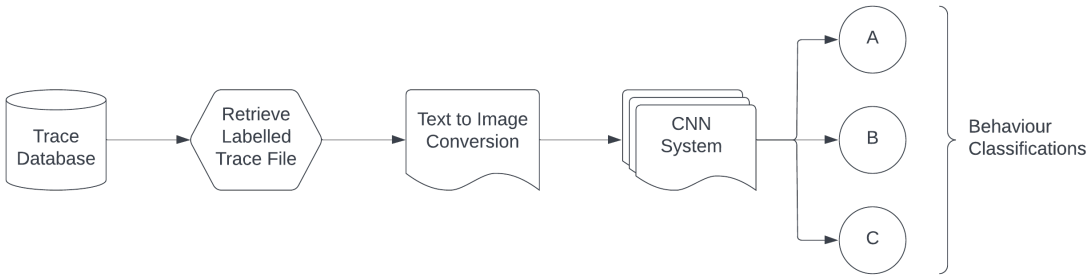
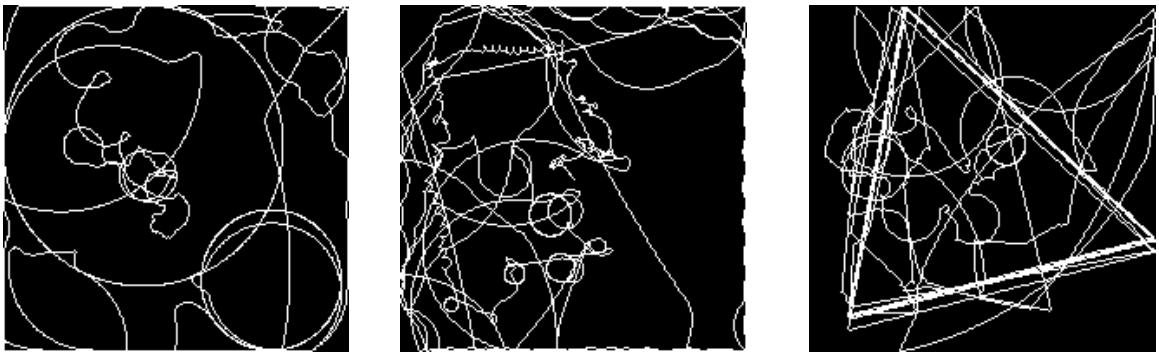


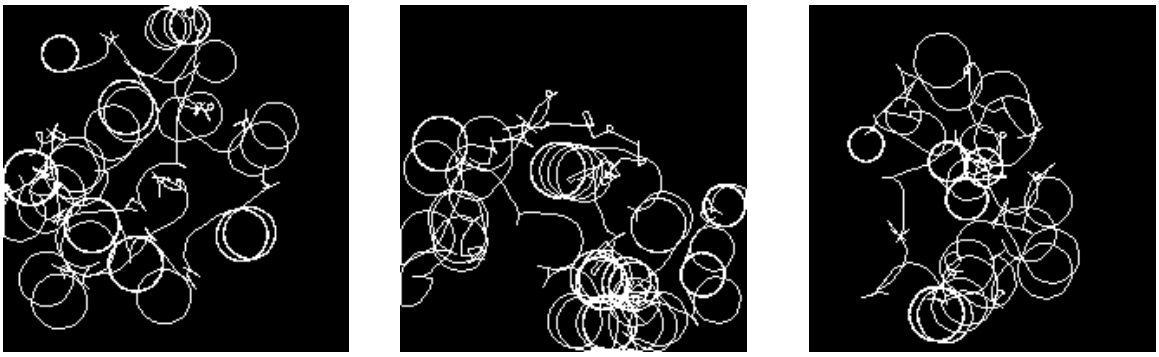
Figure 5.2: Classifier Flow Chart

Figure 5.2 outlines the flow of data from the moment it exits the trace database to the time its categorized using the CNN model. The first step is to obtain a labelled trace file from the trace database. The CNN uses a supervised approach for training, which means each trace has been manually labelled based on one of six emergent behaviours. The traces contained in the trace database were specifically chosen based on their easy to classify characteristics. For instance, an agent that exhibits a ricochet behaviour and a circling behaviour in one trace would not be easily classifiable, so

they are not added to the database. On the other hand, agents that emit only a single type of easily identifiable behaviour such as circling are added.



(a) Difficult to categorize due to multiple behaviours



(b) Easy to categorize due to simple circular behaviour

Figure 5.3: Comparison of Behaviours Based on Complexity

The next step in the classification pipeline is to convert the text-based trace file into an image-based trace file using the trace image generator program. Since all traces begin as text files, this stage is necessary in order to provide the CNN with proper input. Once completed, the image is fed into the CNN model for classification.

#	**Layer**	**Output Shape**	**Neurons**	**Activation Function**	**Note**
1	Rescaling	(200, 200)	1		Input
2	Conv2D	(198, 198)	32	ReLU	Kernel Size = (3, 3)
3	MaxPooling2D	(99, 99)	32		Pool Size = (2, 2)
4	Conv2D	(97, 97)	64	ReLU	Kernel Size = (3, 3)
5	MaxPooling2D	(48, 48)	64		Pool Size = (2, 2)
6	Conv2D	(46, 46)	128	ReLU	Kernel Size = (3, 3)
7	MaxPooling2D	(23, 23)	128		Pool Size = (2, 2)
8	Dropout	(23, 23)	128		Rate = 0.25
9	Dense	(256)	256	ReLU	Batch Normalization
10	Dropout	(256)	256		Rate = 0.5
11	Dense	(128)	128	ReLU	Batch Normalization
12	Dropout	(128)	128		Rate = 0.5
13	Dense	(6)	6	Softmax	Output

Table 5.1: CNN Architecture

A variety of CNN architectures were examined before settling on the architecture seen in Table 5.1. After trying different approaches with varying degrees of success, the architecture chosen was based on the work of Chen [9] due to using similarly sized black and white images containing thin line strokes. The model was also heavily inspired by VGGNet [47] and LeNet [28]. Some modifications were made to the overall architecture, such as the increasing the number of filters used in the 3 convolutional layers, as well as changing the number of outputs in the last dense layer to 6 in order to represent the number of emergent behaviour classes.

Table 5.1 shows that the CNN consists of 13 layers, each with different characteristics. The rescaling layer is the first layer in the CNN. It takes an image input of (200, 200, 1) which represents a 200 pixel by 200 pixel image with a single grayscale colour channel, and normalizes all pixel values between 0 and 1 by dividing them by 255. The following six layers consist of convolutional layers and max pooling layers. In each of the convolutional layers, a kernel size of (3, 3) is chosen alongside the ReLU activation function. The max pooling layers use a pool size of (2, 2). By combining these two layers and increasing the number of filters at each stage, various features can be extracted from the images. Layers 8, 10, and 12 are dropout layers which help to prevent overfitting by randomly dropping out some of the neurons. The first

dropout layer uses a rate of 0.25, whereas the final two dropout layers use a rate of 0.50. The dense layers are fully connected and use batch normalization which allow for faster and more stable training of the network. The last dense layer is the output layer which uses a softmax activation function to produce a probability distribution over the six output classes.

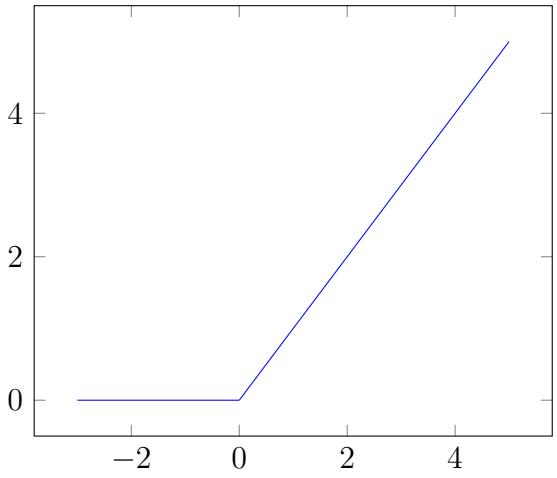


Figure 5.4: Graph of ReLU Activation Function

The ReLU activation function is a widely used activation function for training CNNs. Figure 5.4 shows what the ReLU function looks like when graphed. The function equates to $ReLU(z) = max(0, z)$ meaning that if the output of a neuron is less than 0, it will automatically round up to 0. If the output is greater than 0, it will use the original output instead.

The softmax activation function is commonly used in the final layer of a CNN for multi-class classification tasks. It works by taking the raw class scores (also referred to as *logits*) produced by the previous layers and converts them into probabilities that represent the likelihood of each class being the correct one. The softmax function is defined as:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{5.1}$$

where σ is the result from the softmax, $\vec{z}$ is the input vector, e^{z_i} is the standard exponential function for the input vector, e^{z_i} is the standard exponential function for the output vector, and K is the number of classes.

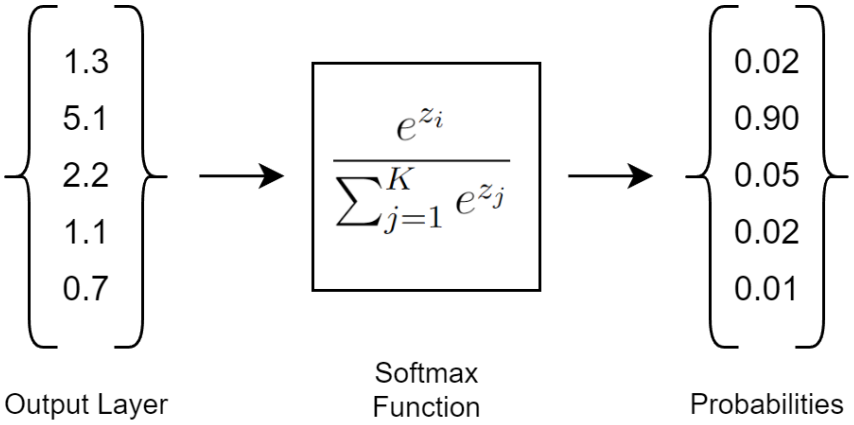


Figure 5.5: Softmax Function Example

Figure 5.5 shows what happens when a vector $\vec{z}$ consisting of $K = 5$ real numbers is fed into the softmax function and normalized into a probability distribution. Notice the probability distribution is proportional to each value and sums to 1. In this example, the highest input value from vector $\vec{z}$ results in the highest probability value in the output vector. Therefore, as the result of the classification, the value at index 1 ($5.1 \rightarrow 0.90$) represents the CNN's prediction output.

Chapter 6

Experiment Series I: Emergent Behaviour Classification

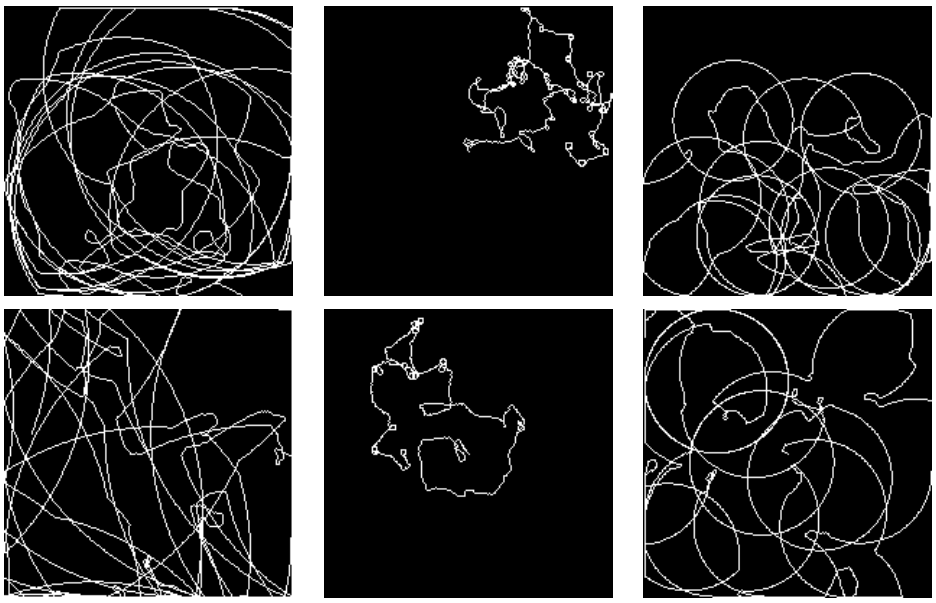
The goal of this series of experiments is to successfully train a CNN model to classify emergent behaviours of intelligent agents and test the model on a variety of traces.

6.1 Experiment A - Training the CNN Model

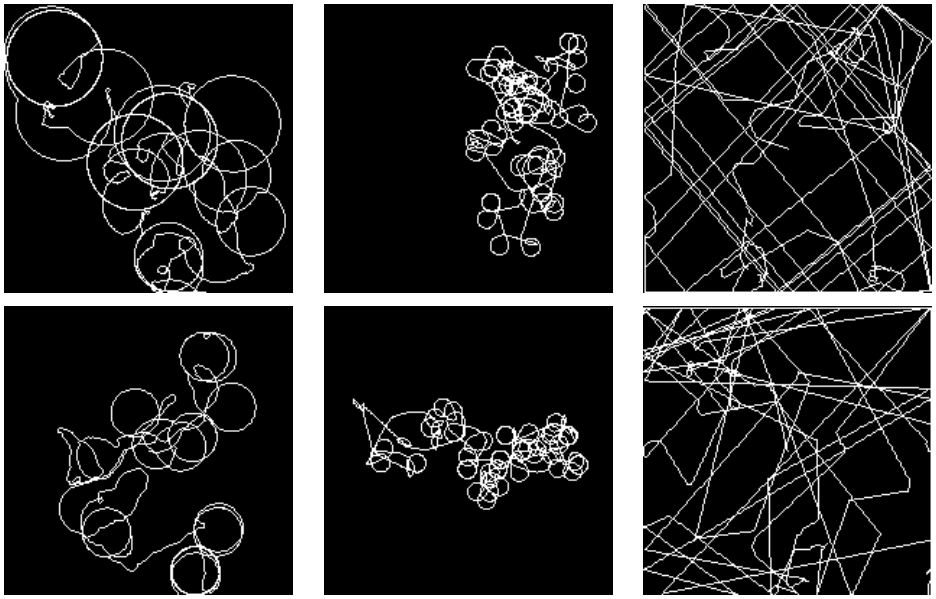
The purpose of the first experiment is to train the CNN model on a data set of emergent behaviours. These behaviours were chosen after running the GP system over 250 times and generating over 60,000 trace images. The 6 most prevalent categories of emergent behaviours were used to train the model: *arched line ricochet*, *classic pursuit*, *large circle pursuit*, *medium circle pursuit*, *small circle pursuit*, and *straight line ricochet*. 2,000 images from each category were selected to train and validate the model. Only trace images which clearly depict the behaviour and are easily distinguishable from other behaviours were chosen.

6.1.1 Training Dataset

Training the perfect CNN model to fit a specific set of data is not an easy task, especially when data can be difficult to fit into definitive categories. This is why the model chosen for this work will not resemble a perfect model but instead, a model that is deemed performant enough to categorize most emergent behaviours. After running the GP system over 250 times and generating over 60,000 hand-labelled training images, 6 categories of emergent behaviours were shown to be the most prevalent.



(a) Arched Ricochet (b) Classic Pursuit (c) Large Circle



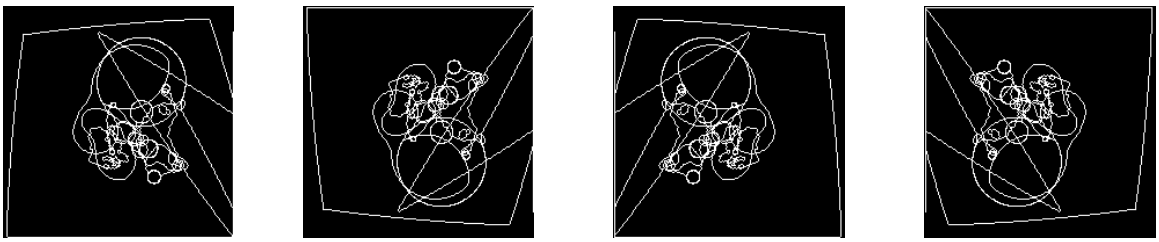
(d) Medium Circle (e) Small Circle (f) Straight Ricochet

Figure 6.1: Categories of Emergent Behaviours

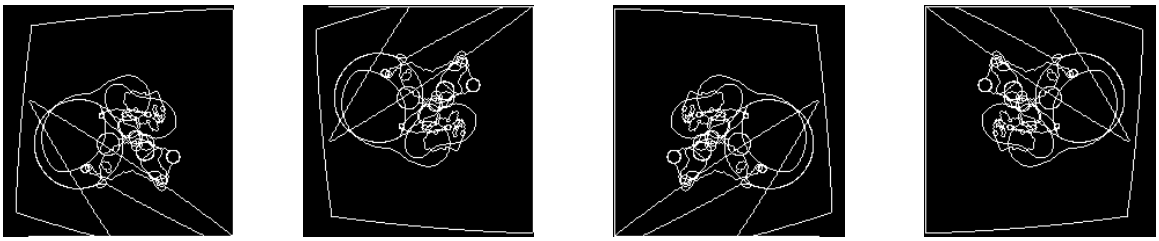
The first category is *arched-line ricochet* which represents lines that bounce off walls and are curved in nature. The second category is *classic pursuit* which represents typical cat and mouse behaviour. The third category is *large circle pursuit* which represents a large circling behaviour, often confused with arched-line ricochet. The fourth category is *medium circle pursuit* which represents medium circling behaviour and is often seen overlapping. The fifth category is *small circle pursuit* which are

sometimes difficult to distinguish from classic pursuit but easy to distinguish from medium circles. The sixth category is *straight line ricochet* which are straight lines at various angles that ricochet off the walls.

In order to generate a significant amount of training data for the CNN to train on, each trace is converted into 8 rotated and reflected versions. By doing so, a single GP run can generate 8 times as many training data images.



(a) From left to right: XY, X-Y, -XY, -X-Y



(b) From left to right: YX, Y-X, -YX, -Y-X

Figure 6.2: Comparison of Rotations and Reflections in Traces

Figure 6.2 depicts the 8 orientations that are possible from rotating and reflecting a trace file. Notice that there are no trace images where a translation has taken place. This is because images are known to be translation invariant when tested on CNNs, but variant to rotations and reflections. In (a), the first image is the original trace file, the second image is the result of using the original x values and subtracting the y values from 200, the third image is the result of subtracting the x values from 200 and using the original y values, and the fourth image is the result of subtracting the x and y values from 200. In (b), each image is reflected by switching the x and y values. The first image is the original trace file, the second image is the result of using the original y values and subtracting the x values from 200, the third image is the result of subtracting the y values from 200 and using the original x values, and the fourth image is the result of subtracting the y and x values from 200. The number 200 is chosen to subtract by because it is the height and width of the simulation environment as well as the size of the 200px by 200px image traces. If the simulation

environment was a different size, it would need to be accounted for when creating the rotated and reflected trace file variants.

6.1.2 Setup

To begin training the CNN for behaviour classification, several parameters were set, as shown in Table 6.1. All of the following values are widely accepted parameters used in various literature [9], [4], [6].

Parameter	**Value**
Validation Split	20%
Batch Size	32
Optimizer	Adam
Metrics	Accuracy, Loss
Loss Type	Sparse Categorical Cross-entropy
Epochs	20

Table 6.1: CNN Training Parameters

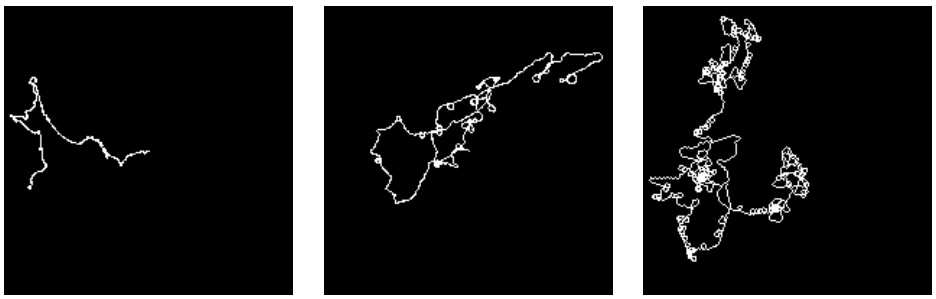
Validation split is an essential process that allows the model to evaluate its performance on data it has never seen before. By doing so, it will prove that the model is capable of generalizing, rather than memorizing, data. The validation split used in all experiments will be 0.2, meaning that 20% of the training examples will be withheld from training and used to validate the model as time goes on. Since there is a total of 12,000 images in the data set, 9,600 will be used to train the model and 2,400 will be used for validating. Another process used while training a CNN is to separate data into smaller subsets called batches. The batch size refers to the number of training examples a model will be fed before updating its parameters. Choosing an appropriate batch size will have a large effect on the model, as smaller batch sizes lead to more frequent updates which can help the model generalize well. On the other hand, large batch sizes speed up the training process but require more memory. The batch size chosen for this work is 32 and based on [9]. The adaptive moment estimation optimization algorithm, also known as Adam, is an optimization algorithm commonly used for training CNNs [23]. It works by maintaining a learning rate that adapts over time and applies itself to each parameter. Another key feature of Adam is that it keeps track of the momentum of past gradients which allows it to perform well on deep learning tasks and helps converge quickly.

The primary metrics used to evaluate the performance of the CNN model are accuracy and loss. Accuracy is defined as:

$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\% \tag{6.1}$$

The loss is a metric used to compare the predicted class probabilities with the actual target class. The sparse categorical cross-entropy loss function is used in multi-class classification problems where each input belongs to a single class. The primary difference between sparse categorical cross-entropy and categorical cross-entropy is that the labels are represented as integers rather than one-hot encoded vectors. The number of epochs used to train the model is 20. This value represents the number of complete passes the data makes through the CNN during the training phase.

One noticeable feature of the training data set is that there is a high degree of variation in each of the images. For each of the 6 emergent behaviours, approximately one third of the images depict a predator that traverses only a small portion of the arena and is typically low-performing. The next one third of images depict a predator who traverses a moderate portion of the arena and is typically mid-performing. The final one third of images depict a predator who traverses a large portion of the arena and is typically high-performing and reactive. In order to achieve a CNN model that is able to categorize a multitude of agents regardless of their performance, it is important to include a wide variety of examples to allow for potential unknown variations in predator movement to be predicted in the future. Examples of all classes of emergent behaviours will now be shown.

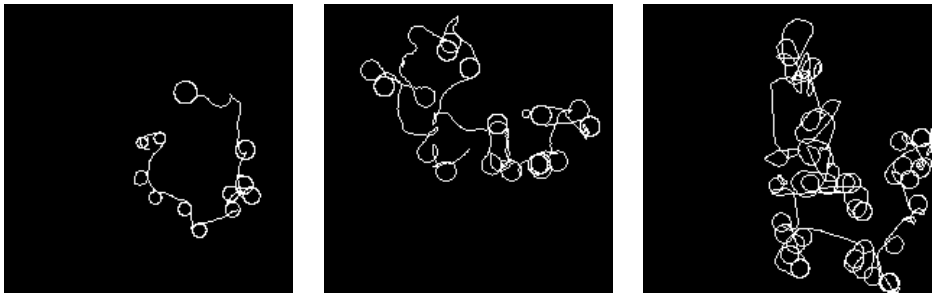


(a) Small Coverage (b) Medium Coverage (c) Large Coverage

Figure 6.3: Comparison of Classic Pursuit Arena Coverage

Figure 6.3 shows three examples of what we call "classic pursuit" image traces in order of their arena coverage from left to right. The first image depicts an agent that is low performing and traverses a very small amount of their environment. The middle image depicts an agent that performs well and makes adjustments based on

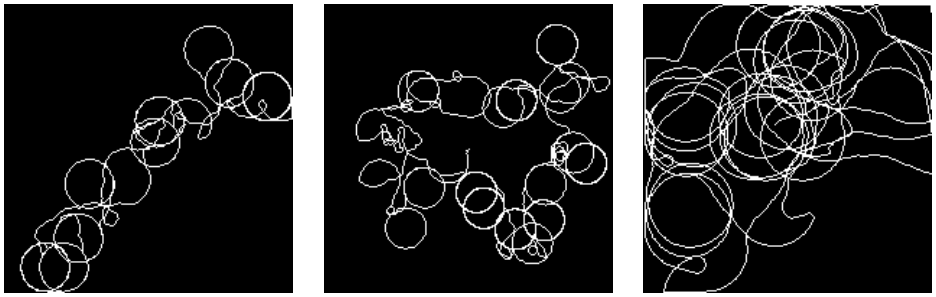
its surroundings. The final image on the right depicts an agent that performs well in all aspects, traversing a large portion of its environment and constantly making adjustments when needed.



(a) Small Coverage (b) Medium Coverage (c) Large Coverage

Figure 6.4: Comparison of Small Circle Arena Coverage

In Figure 6.4, the small circling behaviour seen in the left image depicts a predator that performs poorly and traverses in a small loop. This predator does not travel far from the center and therefore it is unlikely to catch many prey. Traces such as these can be hard for the CNN to correctly identify due to its similarities to a classic pursuit behaviour. The second and third images depict better performing agents that exhibit a pattern of small circles combined with chasing behaviour. They also explore close to 50% of the search space but leave the remaining 50% untouched.

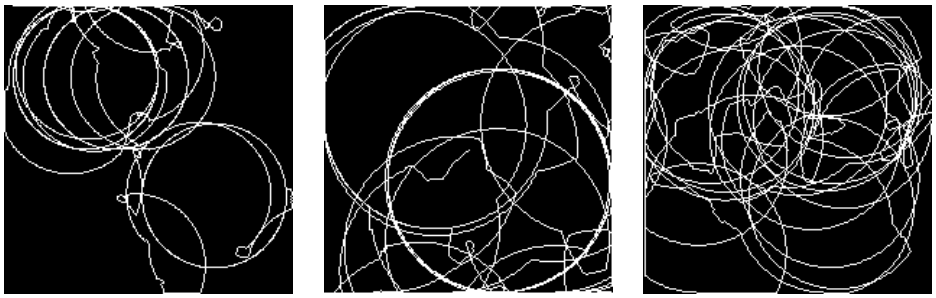


(a) Small Coverage (b) Medium Coverage (c) Large Coverage

Figure 6.5: Comparison of Medium Circle Arena Coverage

In Figure 6.5, the medium circle behaviour can be seen which allows for an improvement in arena coverage through use of tightly-coupled circling in multiple directions. These circles can range from being quite small as shown in the first two images, to growing quite large as seen in the right-most image. Due to the variance in the size of circles, it is possible that the CNN may classify these images as a small circle or large circle behaviour due to the subjective nature of data labelling. However,

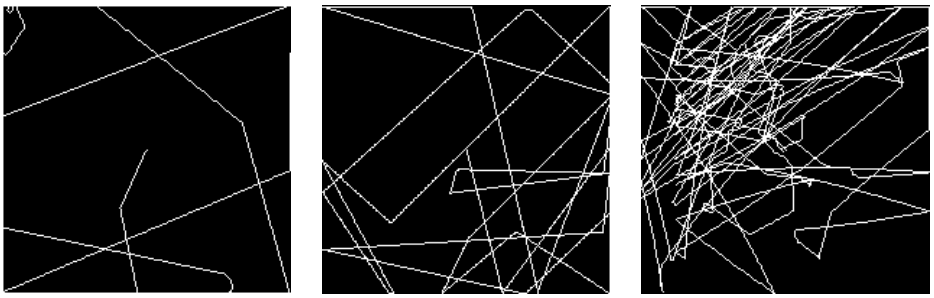
significant effort was made to keep the three kinds of circling behaviours as distinct as possible.



(a) Small Coverage (b) Medium Coverage (c) Large Coverage

Figure 6.6: Comparison of Large Circle Arena Coverage

The large circling behaviour shown in Figure 6.6 depicts agents that cover a significant portion of their environment when compared to small or medium circling behaviours. Starting at the left most image, the agent traverses a significant portion of the arena but stops to overlap itself. When looking at the middle image, there is a significant increase in circle diameter and many changes in the predator's positioning. The final image on the right is full of noise due to having so many large circles overlapping in such a small area. Due to the simplicity of the other trace files, this image may be difficult for the CNN to classify, but it is included to help improve generalization of the model.

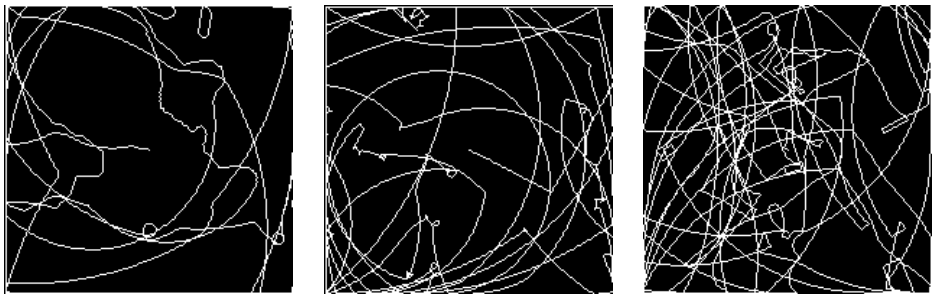


(a) Small Coverage (b) Medium Coverage (c) Large Coverage

Figure 6.7: Comparison of Straight Line Ricochet Arena Coverage

The easiest behaviour for humans to identify is straight line ricochet, as seen in Figure 6.7. This behaviour involves agents who very rarely traverse using curved lines, but instead choose to snap to their next orientation. Sometimes agents will only ricochet a handful of times, as seen in the first image on the left. Other times predators may ricochet dozens of times, exploring a significant portion of the search

space as seen on the right. Low performing agents that often get stuck after travelling a small distance, or only ricochet a handful of times, are included in the straight line ricochet data set.



(a) Small Coverage (b) Medium Coverage (c) Large Coverage

Figure 6.8: Comparison of Arched Line Ricochet Arena Coverage

Figure 6.8 outlines the most difficult to categorize behaviour: arched line ricochet. The image on the left depicts a predator who makes several turns and ricochets but does not represent a type of circling behaviour. The middle image depicts a predator who covers a significant portion of the arena by ricocheting and rotating. The final image on the right depicts a busy trace with lots of twists, turns and ricochets. The primary reason that makes arched line ricochet so difficult to classify is that many of the movements are closely related to that of large circling and straight line ricochet.

6.1.3 Training Results

The following subsection details the results obtained from training the CNN model. Matplotlib 3.5.3 [21] is used to visualize the data at the end of training.

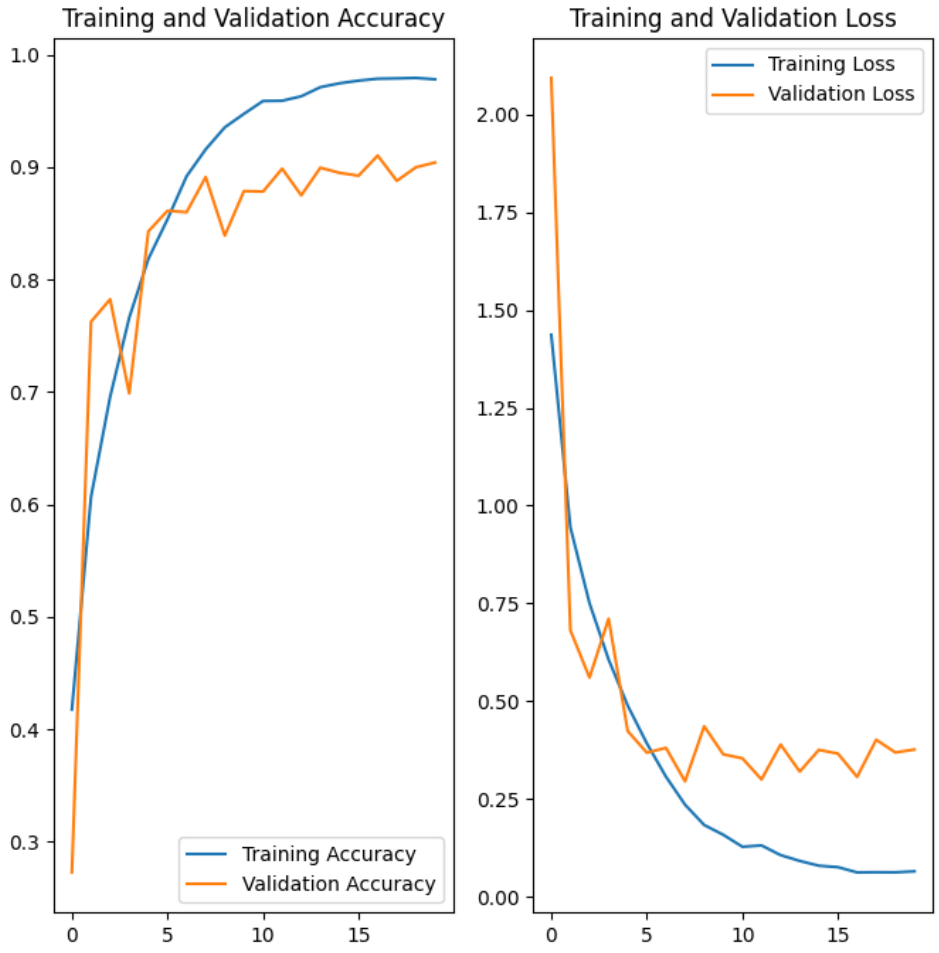


Figure 6.9: CNN Training & Validation Results

Figure 6.9 details the results from training and validating the CNN model on all 12,000 images from the trace data set. The graph on the left side represents the training and validation accuracy of the model, and the graph on the right represents the training and validation loss of the model. After training the model for 20 epochs, the training accuracy converges at 98% and the validation accuracy converges at 90%. The training loss converges at 0.065 and the validation loss converges at 0.37.

The model's training accuracy shown in Figure 6.9 shows a steady increase during the initial epochs. This increase is a positive indicator that the model is effectively learning from the data. The validation accuracy closely mirrors this trend until the 8^{th} epoch. From this point the training accuracy continues to climb but the validation

accuracy deviates from the trend and converges at 90%. After training for 20 epochs, a training accuracy of 98% is achieved, indicating a strong fit of the model to the data set and negating any concerns of under-fitting. In terms of validation, an accuracy of 90% is attained. Even though it falls slightly short of matching the training accuracy, the outcome is still strong enough to be considered for future experiments.

There are two potential reasons that the training and validation accuracy show some disparities. One likelihood is that the model is over-fitting to its data, meaning that it is memorizing the existing data set and having troubles categorizing novel images. A reason for this could be a lack of variety in training examples, or a lack of training examples overall. A second reason could be that the data itself may have some imperfections in the form of challenging to classify examples or potential labeling errors. Given the subjectivity of categorizing certain behaviours, these imperfections are the most likely of the two scenarios. Despite this, it is important to note that a validation accuracy of 90% still shows that the model is very strong at classifying novel data.

To enhance the model's performance in future iterations, several strategies could be employed. The first would be to refine the data set to mitigate any labeling concerns. Removing any examples from the data set that could be considered challenging to classify may result in some initial improvements. To further improve the labeling process, multiple people could be asked to classify the data and only traces that are agreed upon by the majority would be selected. Employing this strategy could help mitigate subjectivity. Another improvement could be to expand the data set by introducing additional training examples, particularly examples that introduce a greater diversity in traces. Furthermore, introducing new categories of behaviours to classify could allow for improvements to the model. Overall, terminating the training process at the 20^{th} epoch seems to be ideal, due to diminishing returns in terms of accuracy and loss improvement after only the 10^{th} epoch.

6.2 Experiment B - Testing the CNN Model

The following experiment is designed to assess the effectiveness of the CNN model through use of a carefully crafted testing data set consisting of 350 images. The data set consists of an array of traces spanning different categories in order to give a comprehensive evaluation of the model.

6.2.1 Setup

Now that the CNN model has been trained to recognize and classify traces depicting predator behaviours, the next step is to give the CNN new data that it has not seen before in order to gain a comprehensive understanding of which features it may be struggling with. To do so, 50 images have been selected to test the CNN model from each of the 6 categories. There is also a 7^{th} set of data consisting of 50 images that will challenge the CNN model on behaviours it has never seen before which can be found in Appendix B. These unique images can be classified two ways: either they are a combination of two or more known behaviours, or they depict an unknown behaviour that does not belong to one of the six categories the model was trained on. Despite its specialized training, the CNN model is expected to maintain a reasonably accurate classification score for traces it hasn't been exposed to. In cases where the CNN encounters unique traces, it should still be capable of making predictions by identifying the most similar behaviour class based on data it was trained on. By doing so, the model's adaptability and generalization capability will be highlighted.

To evaluate the model's performance on testing data, the primary metric to be used is accuracy. The values to be seen in the results section refer to the prediction generated by the model after it has been applied to all 50 individual traces. These traces represent distinct predators from each behaviour category that the model has not seen before. By averaging the predictions across all traces, an overall representation of the model's performance can be seen, reflecting its ability to make accurate predictions and determining which behaviours are easy or difficult for the model to classify. The overall objective is to achieve a 100% prediction accuracy within the category corresponding to the behaviour of each image. To better understand the dispersion of data in relation to its mean, the standard deviation of each set of data is measured. The standard deviation quantifies the amount of variability or spread within a data set. A higher standard deviation signifies a greater dispersion from the mean, while a lower standard deviation signifies that data points are close to the mean. The results from this experiment will give insights into multiple aspects of the CNN, one of which is to understand how the model averages its prediction per behaviour category. To better understand this, a visualization using stacked bar charts is created show the distribution of data from each behaviour category across all 50 traces. This allows for a comprehensive view of the model's predictive performance.

6.2.2 Testing Results

The following subsection details the results obtained from testing the CNN model. 7 data sets, each containing 50 trace images were used to evaluate the model.

Experiment / Class	ALR	SLR	CP	SCP	MCP	LCP
ALR	**35.1546**	30.8238	0.007	0.0196	0.0026	5.3338
SLR	32.2062	**68.9732**	0.007	0.015	0.0032	0.1524
CP	0.5358	0.0184	**97.3672**	0.4834	0.0028	0.0244
SCP	2.529	0.0184	0.9232	**83.0656**	0.8588	0.0258
MCP	0.3156	0.0548	1.6754	16.4	**98.9784**	5.135
LCP	29.2582	0.1112	0.0138	0.0134	0.1516	**89.327**

Table 6.2: Average Prediction per Category (50 Examples per Category)

Table 6.2 details the accuracy obtained from testing the CNN on each of the 6 known categories. The first row represents each experiment name, where ALR refers to *arched line ricochet*, SLR refers to *straight line ricochet*, CP refers to *classic pursuit*, SCP refers to *small circle pursuit*, MCP refers to *medium circle pursuit*, and LCP refers to *large circle pursuit*. The first column represents the name of each behaviour.

In the sets of experiments involving the ALR and SLR behaviours, the model encounters difficulties differentiating between them. The reason for this could be due to the behaviours being similar in terms of visual characteristics, such as their long lines and wide angles. The model seems to struggle the most with the ALR behaviour, and performs better at classifying the SLR behaviour, likely due to ALR being similar to SLR and LCP, whereas SLR is only similar to ALR. When looking at the circling behaviours such as SCP, MCP, and LCP, the model demonstrates very promising results. MCP stands out as the highest accuracy across all experiments which indicates that the model excels at distinguishing MCP from other behaviours. Despite the strong performance in the MCP runs, CP and LCP are occasionally classified as MCP. In summary, these experiments show that while the CNN exhibits strengths in classifying certain behaviours, it faces challenges when behaviours share similar attributes. However, these findings provide valuable insights that could be used to improve the model's performance in the future.

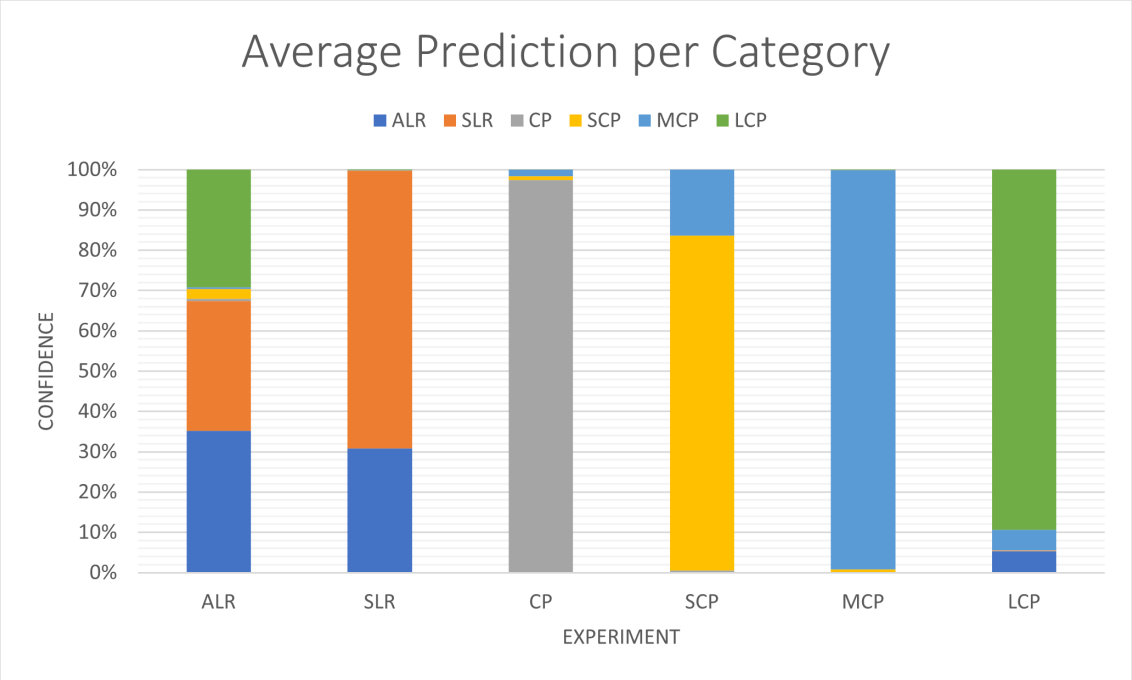


Figure 6.10: Average Prediction per Category Visualization

Figure 6.10 provides a visualization of the data presented in Table 6.2.

Experiment / Class	**ALR**	**ALR**	**CP**	**SCP**	**MCP**	**LCP**
ALR	40.22	36.36	0.02	0.08	0.01	18.35
SLR	40.37	36.53	0.02	0.05	0.01	0.60
CP	1.37	0.02	12.32	1.46	0.01	0.04
SCP	11.10	0.02	1.97	33.70	4.78	0.05
MCP	0.67	0.12	11.69	33.79	4.78	17.40
LCP	42.43	0.28	0.07	0.04	0.51	24.80

Table 6.3: Standard Deviation per Category

Table 6.3 details the standard deviation obtained from testing the CNN on each of the 6 known categories. In the first experiment involving the ALR data, the LCP category had the highest standard deviation, indicating a significant data spread. The SLR and ALR categories also showed similar standard deviations, while the SCP category contained some outliers indicated in Figure 6.14. In the SLR experiment, both the SLR and ALR categories exhibited high standard deviations which also indicate significant data spread. In the CP experiment, the CP and MCP categories

had comparatively higher standard deviations, while the other categories had data points that were closer to the mean. The CP, SCP, and MCP categories also had some outliers, as indicated in Figures 6.13, 6.14, and 6.15. In the SCP experiment, the SCP and MCP categories displayed high standard deviations and contained outliers as shown in Figures 6.14 and 6.15 which shows that making their distinction can be challenging for the CNN. In the MCP experiment, the SCP and MCP categories showed small standard deviations with only a few outliers. In the final experiment centered around LCP data, moderate standard deviations are seen across the ALR, MCP, and LCP categories along with several outliers as indicated in Figures 6.11, 6.15, and 6.16. This suggests potential confusion among these 3 categories, despite the CNN's reasonable accuracy when classifying them.

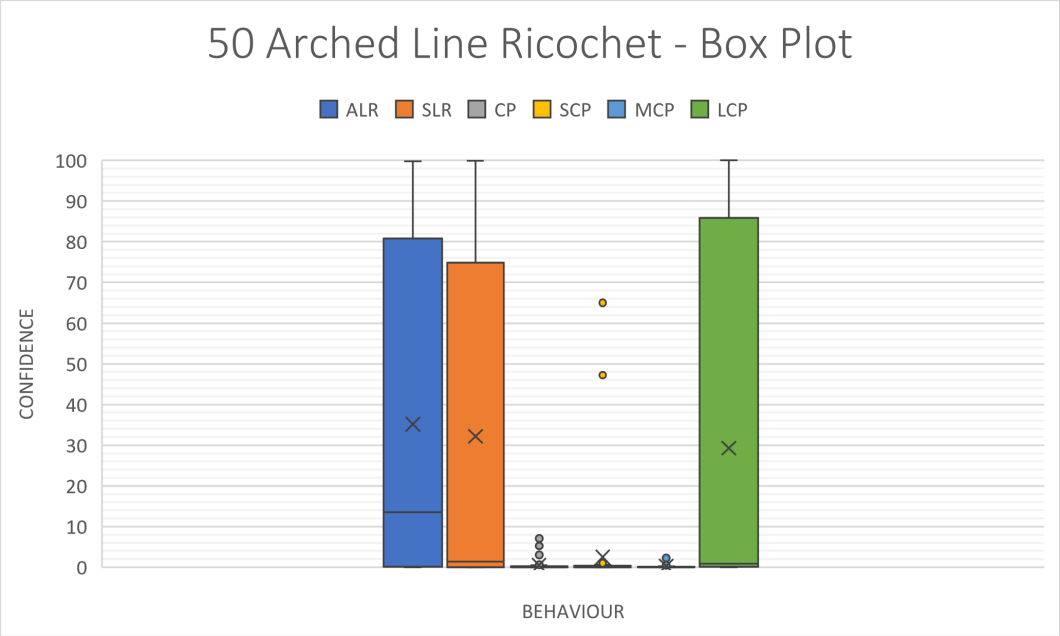


Figure 6.11: Arched Line Ricochet Box Plot

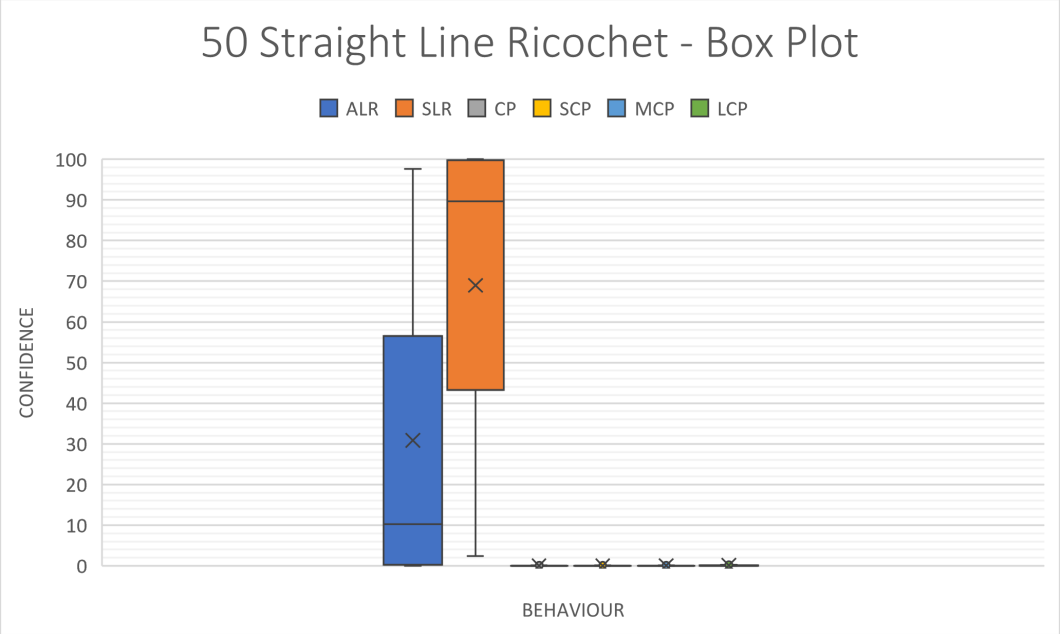


Figure 6.12: Straight Line Ricochet Box Plot

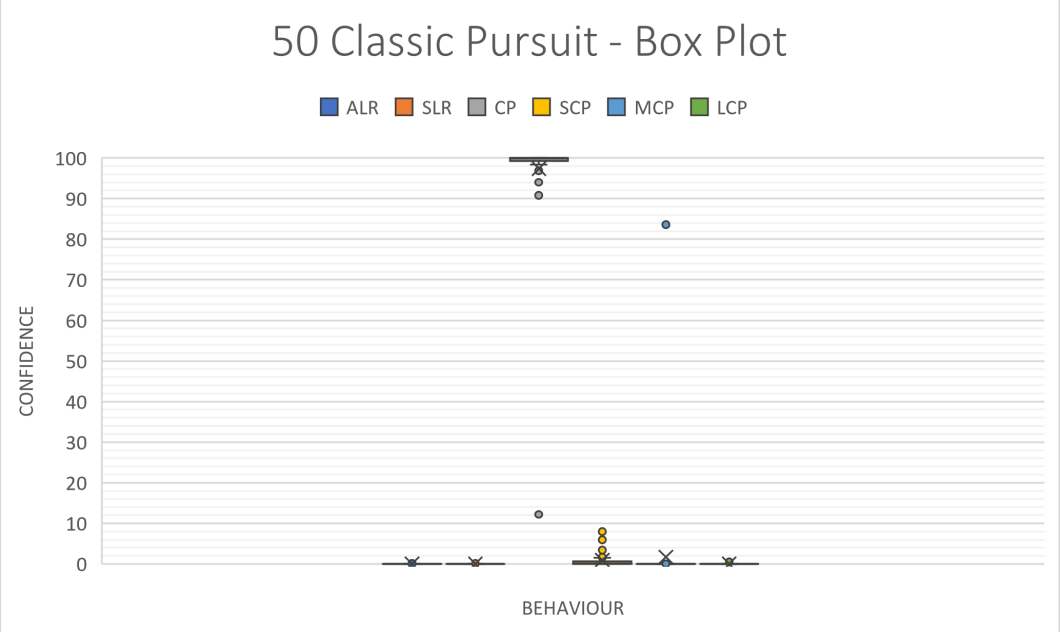


Figure 6.13: Classic Pursuit Box Plot

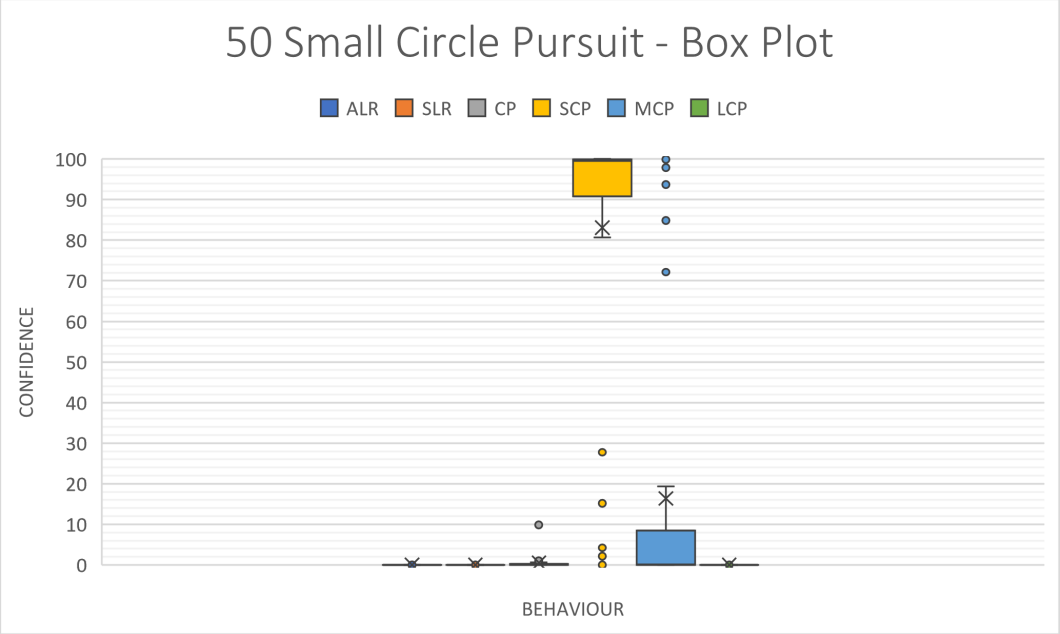


Figure 6.14: Small Circle Pursuit Box Plot

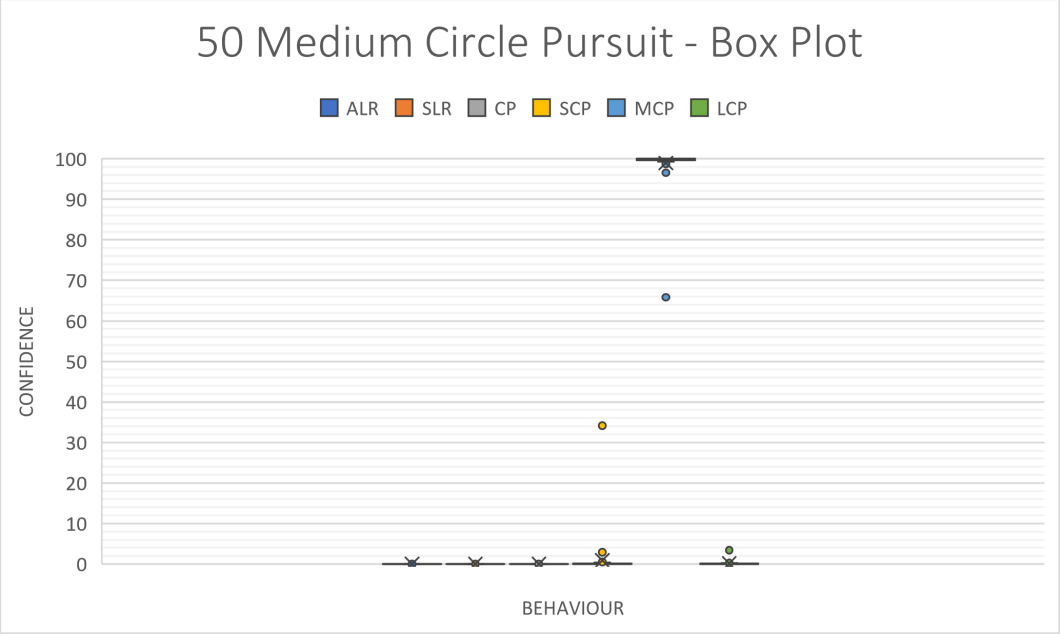


Figure 6.15: Medium Circle Pursuit Box Plot

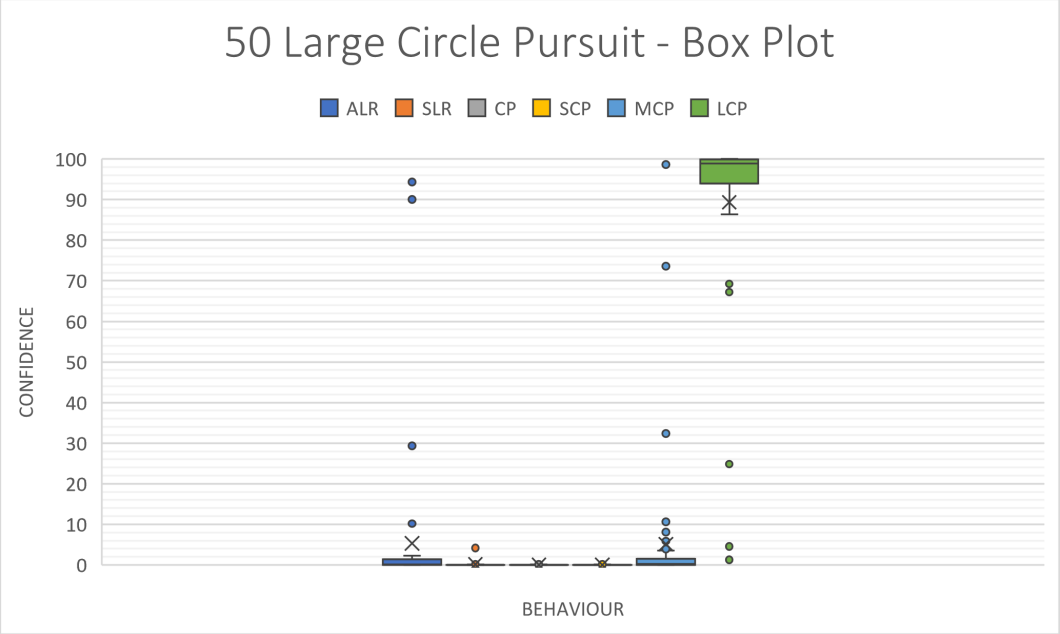


Figure 6.16: Large Circle Pursuit Box Plot

Figures 6.11 through 6.16 present a graphical representation of the spread of data across each category in the experiments. The dashed lines (whiskers) represent the minimum and maximum values in the set, the outer edges of each box represent Q_1 and Q_3 quartiles, and the center line represents the median of all entries. Any individual dots seen outside of the boxes represent outliers in the data.

	ALR	**CP**	**LCP**	**MCP**	**SCP**	**SLR**
Totals	622.76	403.19	134.17	561.68	553.25	556.99
5% Difference	30	34	43	42	27	34

Table 6.4: Unique Classification Differences

Table 6.4 shows the differences between the author's and CNN's classification of the 50 unique images in Appendix B. The first row is the total difference in classifications whereas the second row is the number of classifications out of 50 in which the author and CNN are within 5% of each other. Unsurprisingly, the ALR category has the highest total difference in classifications which shows that the author and CNN are in disagreement as to which traces depict this category. The best category in terms of total difference is LCP which is significantly lower than the rest of the categories. The number of 5% differences is 43 out of 50 which shows a very strong alignment in predictions. Overall it appears that the behaviours the CNN has most

trouble identifying (ALR and SLR) are the behaviours with the most disagreement between author and CNN. However, categories such as SCP show interesting results where only close to half (27 out of 50) traces are within 5% difference of each other. This could be due to the CP and SCP behaviours having similar characteristics. Lastly, the most surprising result is that the LCP behaviour is the most agreed upon category between the author and CNN, even though the CNN had trouble with it in previous experiments.

	ALR	**CP**	**LCP**	**MCP**	**SCP**	**SLR**
H	160.26	163.72	191.52	165.01	142.99	155.54
DF	5	5	5	5	5	5
P-Value	8.69E-33	1.59E-33	1.85E-39	8.47E-34	4.14E-29	8.81E-32
a	0.05	0.05	0.05	0.05	0.05	0.05
Sig	Yes	Yes	Yes	Yes	Yes	Yes

Table 6.5: Kruskal-Wallis Test - CNN Classification

The Kruskal-Wallis Test is used in this experiment to determine whether or not the group of populations have equal dominance. In order to test this, the CNN is given 50 traces from each behaviour category and the population of each behaviour prediction is compared across all runs (i.e., in the first column 6 sets of 50 ALR predictions are compared to each other, with each set coming from a different type behaviour trace). By doing so, we can ensure that the CNN is able to distinguish between different traces and make decisions based on the information it is given.

Looking at the results from the tests, the p-value is significantly lower than alpha in every run which shows that the population of predictions is very different from one another. This shows that out of the 6 populations, the CNN is giving one of them a significantly different set of values (the behaviour the trace belongs to) which shows that the CNN changes its predictions depending on the type of emergent behaviour the trace exhibits.

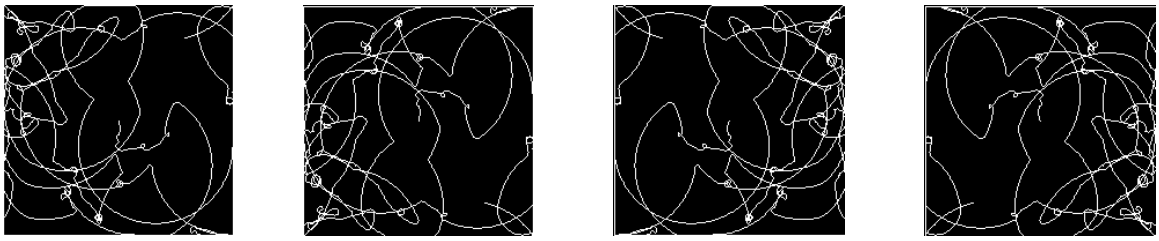
6.3 Experiment C - Rotations and Reflections

The purpose of the following experiment is to determine how the CNN reacts to traces that have been rotated and reflected. As stated in [35], a standard CNN model is not invariant to image rotations, and even a slight rotation of an input image can seriously degrade the performance of a model. In order to test this theory, a series of

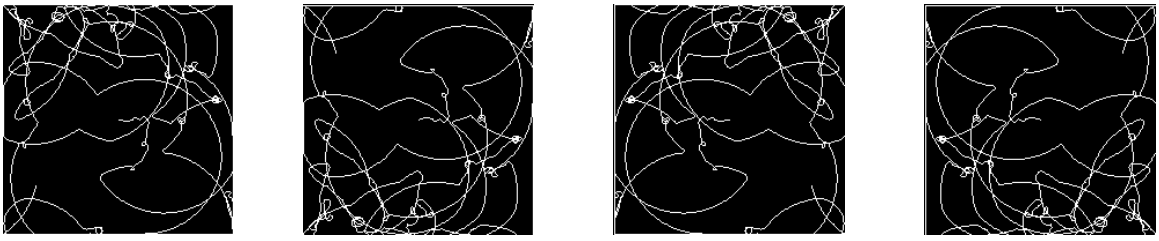
rotated and reflected images belonging to each of the 6 categories are classified by the model. The goal of this experiment is to show that using rotated and reflected traces is an efficient method of gathering additional training data, as the classification of each image is likely to differ.

6.3.1 Setup

In the following subsection, the CNN model will be tested on 8 images from each behaviour data set. The method of evaluating the model will be to compare the prediction output for each of the images and their respective rotations and reflections to see if there are any changes.

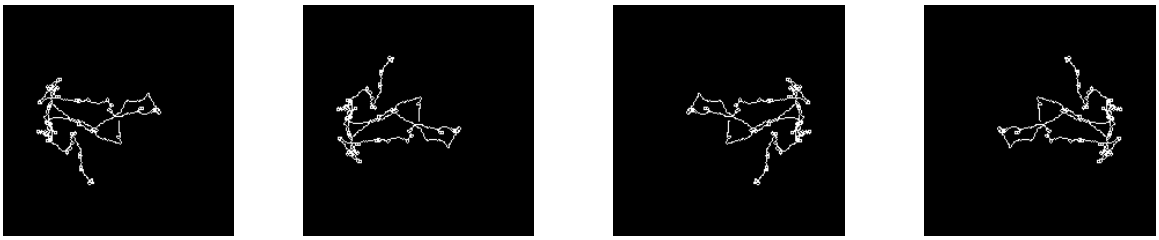


(a) From left to right: XY, X-Y, -XY, -X-Y

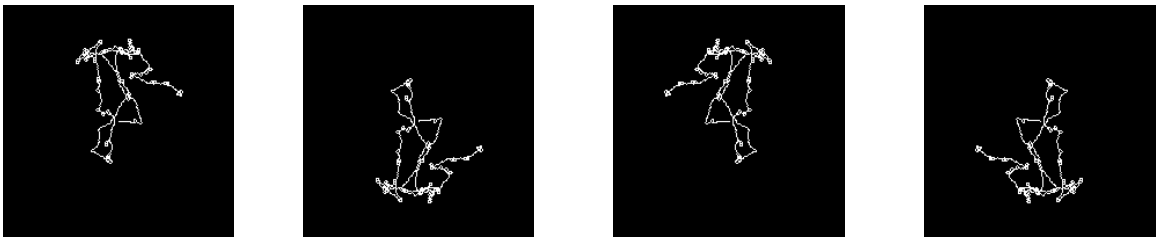


(b) From left to right: YX, Y-X, -YX, -Y-X

Figure 6.17: ALR Rotations and Reflections Test Data

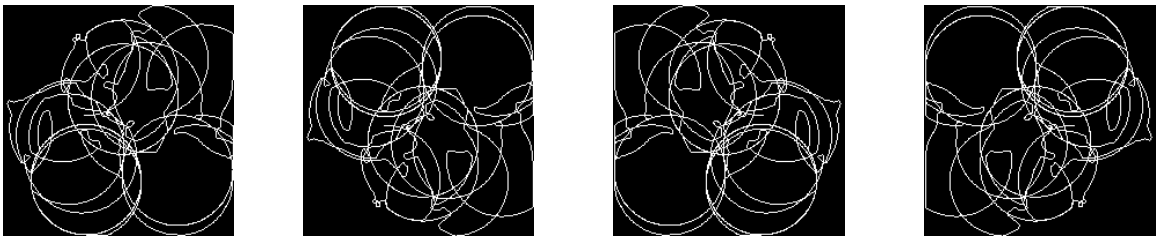


(a) From left to right: XY, X-Y, -XY, -X-Y

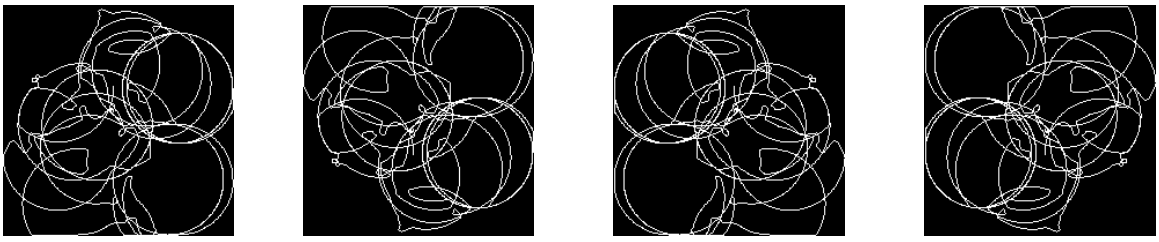


(b) From left to right: YX, Y-X, -YX, -Y-X

Figure 6.18: CP Rotations and Reflections Test Data

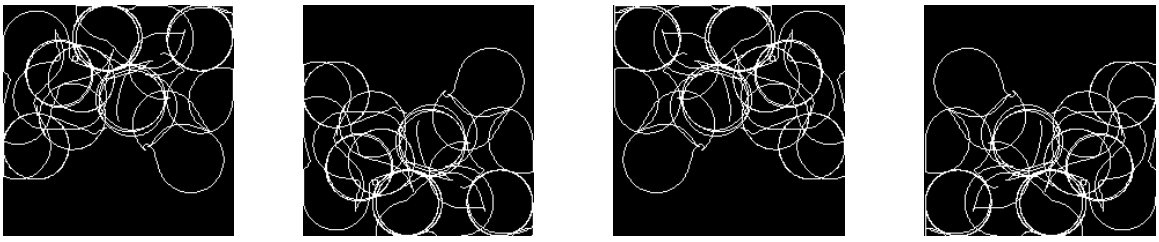


(a) From left to right: XY, X-Y, -XY, -X-Y

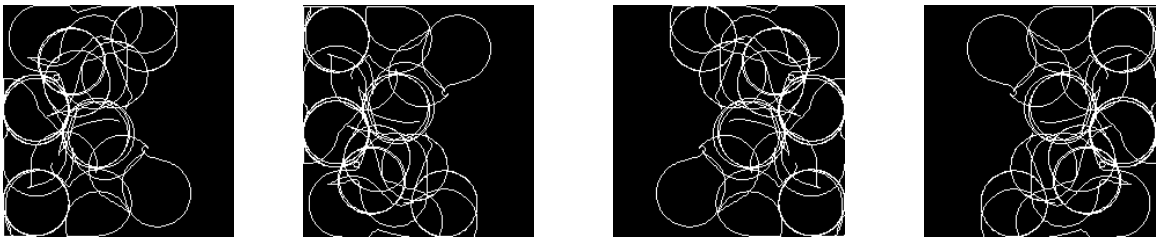


(b) From left to right: YX, Y-X, -YX, -Y-X

Figure 6.19: LCP Rotations and Reflections Test Data

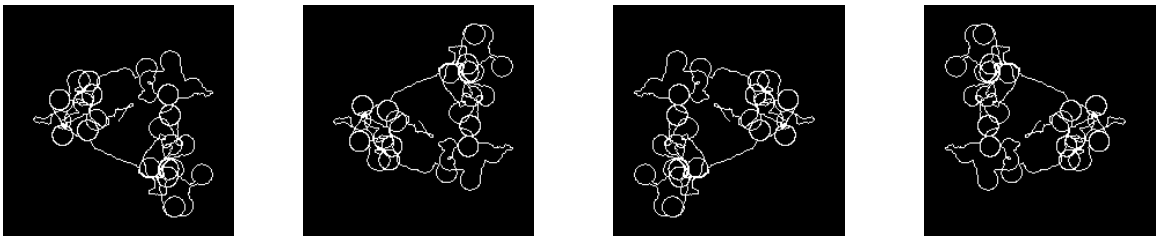


(a) From left to right: XY, X-Y, -XY, -X-Y

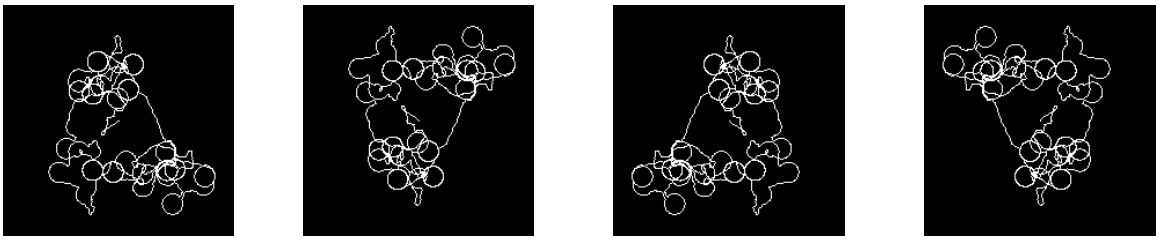


(b) From left to right: YX, Y-X, -YX, -Y-X

Figure 6.20: MCP Rotations and Reflections Test Data

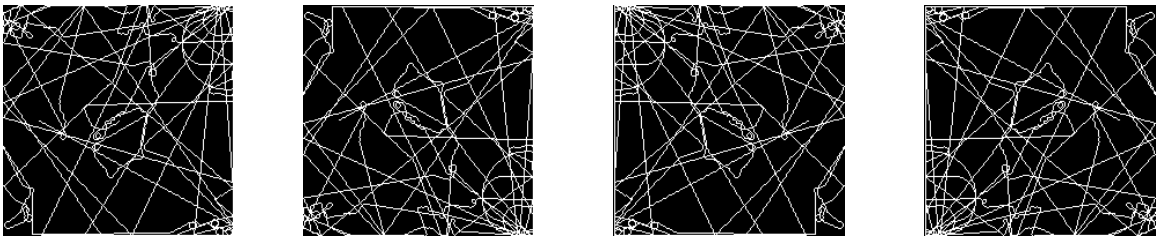


(a) From left to right: XY, X-Y, -XY, -X-Y

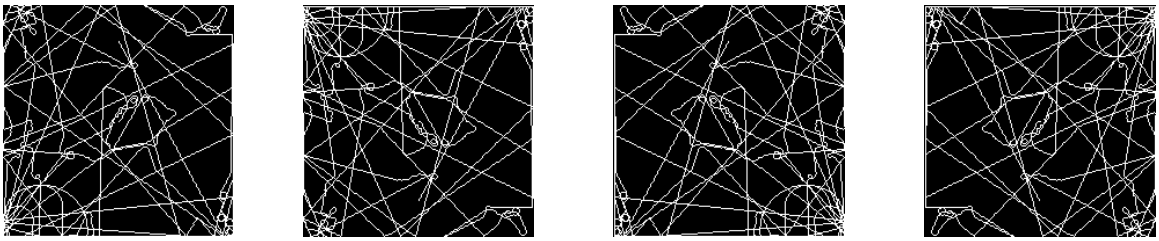


(b) From left to right: YX, Y-X, -YX, -Y-X

Figure 6.21: SCP Rotations and Reflections Test Data



(a) From left to right: XY, X-Y, -XY, -X-Y



(b) From left to right: YX, Y-X, -YX, -Y-X

Figure 6.22: SLR Rotations and Reflections Test Data

Figures 6.17 through 6.22 detail the sets of 8 images used to test the CNN on each behaviour.

6.3.2 Rotations & Reflections Results

	XY	X-Y	-XY	-X-Y	YX	Y-X	-YX	-Y-X
ALR	**79.62**	9.6	**81.62**	**69.22**	15.46	5.02	**94.18**	**96.79**
CP	0.1	1.01	0.04	0.05	2.47	0.17	0.04	0.01
LCP	0.08	0.13	0.12	0.09	0.59	0.07	0.2	0.05
MCP	0.08	0.13	0.04	0.05	0.42	0.07	0.04	0.01
SCP	0.08	0.13	0.04	0.05	2.48	0.07	0.04	0.01
SLR	20.05	**89.01**	18.13	30.53	**78.58**	**94.6**	5.49	3.11

Table 6.6: Rotations and Reflections Classification - Arched Line Ricochet

	XY	X-Y	-XY	-X-Y	YX	Y-X	-YX	-Y-X
ALR	0	0	0	0	0	0	0	0
CP	**99.98**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**
LCP	0	0	0	0	0	0	0	0
MCP	0	0	0	0	0	0	0	0
SCP	0.01	0	0	0	0	0.01	0.01	0
SLR	0	0	0	0	0	0	0	0

Table 6.7: Rotations and Reflections Classification - Classic Pursuit

	XY	X-Y	-XY	-X-Y	YX	Y-X	-YX	-Y-X
ALR	0	0	0	0.04	0.01	0.2	0	0.01
CP	0	0	0	0	0	0	0	0
LCP	**99.99**	**99.95**	**100**	**99.95**	**99.97**	**99.79**	**99.93**	**99.99**
MCP	0.01	0.04	0	0.01	0.01	0	0.07	0.01
SCP	0	0	0	0	0	0	0	0
SLR	0	0	0	0	0	0	0	0

Table 6.8: Rotations and Reflections Classification - Large Circle Pursuit

	XY	X-Y	-XY	-X-Y	YX	Y-X	-YX	-Y-X
ALR	0	0	0	0	0	0	0	0
CP	0	0	0	0	0	0	0	0
LCP	0	0	0	0.08	0	0	0.1	0.22
MCP	**99.99**	**99.92**	**99.83**	**99.86**	**99.99**	**99.97**	**99.9**	**99.78**
SCP	0.01	0.08	0.17	0.05	0.01	0.03	0	0
SLR	0	0	0	0	0	0	0	0

Table 6.9: Rotations and Reflections Classification - Medium Circle Pursuit

	XY	X-Y	-XY	-X-Y	YX	Y-X	-YX	-Y-X
ALR	0	0	0	0	0	0	0	0
CP	0	0	0	0	0	0	0	0
LCP	0	0	0	0	0	0	0	0
MCP	19.32	0	0.18	0.06	1.36	0.02	0.42	0.47
SCP	**80.67**	**100**	**99.82**	**99.94**	**98.64**	**99.98**	**99.57**	**99.53**
SLR	0	0	0	0	0	0	0	0

Table 6.10: Rotations and Reflections Classification - Small Circle Pursuit

	XY	X-Y	-XY	-X-Y	YX	Y-X	-YX	-Y-X
ALR	**87.55**	0.24	0.45	17.95	1.98	**61.03**	24.61	5.84
CP	0	0	0	0	0.01	0.01	0.01	0.01
LCP	0	0	0	0	0.01	0.01	0.01	0.01
MCP	0	0	0	0	0.01	0.01	0.01	0.01
SCP	0	0	0	0	0.01	0.01	0.01	0.01
SLR	12.43	**99.75**	**99.55**	**82.04**	**98**	38.94	**75.34**	**94.14**

Table 6.11: Rotations and Reflections Classification - Straight Line Ricochet

Tables 6.6 through 6.11 detail the results from testing the CNN on 8 images from each behaviour data set. In the case of ALR and SLR behaviours, the model's classifications differ when the images are rotated and reflected, indicating a challenge in distinguishing between them. In the case of CP, LCP, MCP, and SCP behaviours, the model consistently classifies them accurately. These results suggest that the model continues to struggle with classifying the ALR and SLR behaviours but performs well when classifying the CP, LCP, MCP, and SCP behaviours. Furthermore, the model's performance appears to be affected by image transformations, supporting the idea that it is variant to rotations and reflections. Overall, the use of a training dataset that includes various instances of rotated and reflected traces was beneficial in creating a versatile and well-trained model.

Chapter 7

Automatic Agent Generator

In this chapter, the automatic agent generator system will be discussed. The system allows for the GP predator-prey simulation and CNN emergent behaviour classifier to be combined in order to introduce a novel approach to intelligent agent design.

7.1 Combining CNN & GP

Previously, the GP system was used to evolve the predator's controller using a traditional fitness measurement that keeps track of the number of prey caught during a simulated hunt. While this method of evolution can produce high performing agents, their behaviours are quite repetitive as noted in [12]. The reason for this repetitive behaviour being seen among a population of agents is that the agents are not rewarded for diverse behaviour. Instead, they are rewarded for completing a single objective, which is to capture as many prey as possible within an allocated time window. So what if, instead of giving the agents a single objective to complete, they are also rewarded for diversifying their behaviours? The results of such an experiment can be seen in [12], where the agents are evaluated on not only how many prey were caught, but also on their wall impacts, time spent near prey, cells visited, and average speed. To summarize their findings, by using a combined fitness score that uses a traditional measurement and diversity measurement, the overall fitness of an individual could actually be *increased*. Agents that display new emergent behaviours were created, which behave in unique and interesting ways.

One problem that occurred when evaluating these agents is that classifying their behaviours was performed manually. This essentially meant that the author needed to watch the animations of agents to determine the emergent behaviour they were exhibiting. This process was shown to be highly monotonous, so only the best solu-

tion from each run was viewed this way. In [41], an improvement to the GP system was made which allowed for the creation of trace files for every agent in the population. This idea also served as the inspiration for the trace generation system used in this thesis. By converting the predator's behaviour into a viewable trace file then feeding it into a CNN, the act of categorizing their emergent behaviours is improved significantly.

As previously seen in Chapter 6, the idea of emergent behaviour classification can be taken one step further by automating the entire process using a CNN. By doing so, the need for additional data points and human classification is eliminated entirely. Instead, the CNN can be fed the trace file from each predator's simulated hunt in *real-time* and output an accurate depiction of their emergent behaviour. Since the entire process is automated and can be run in real-time, the CNN can be supplemented for up to 100% of the GP fitness function. The result is a cohesive system that allows for the automatic generation of intelligent agents that show unique emergent behaviours based on a given set of parameters. Furthermore, these parameters can be set to generate agents with specific desired behaviours.

The CNN and GP parameters have been defined in order to allow each system to perform strongly at their given tasks. The GP creates a variety of high performing agents over 50 generations with 500 individuals per population. Similarly, the CNN is able to classify emergent behaviour with over 90% validation accuracy after training for 20 epochs. It's safe to say that when combining the two systems, the parameters for each individual system are to remain unchanged in order to prevent any negative impacts to performance. There are also some new high-level parameters that are introduced, such as the type of fitness being used (CNN, GP, or Combined), the CNN model file, Python interpreter, and file system paths that need to be specified when running the program.

When choosing the type of fitness to be used, the first option is to use 100% of the GP fitness and 0% CNN, meaning that the predator's single objective is to catch as many prey as possible within the given time frame. This is simply generic fitness-guided evolution. The second option is to use 100% of the CNN fitness and 0% of the GP fitness. This means that the number of prey captured is irrelevant, and diverse but possibly ineffective behaviour is sought. Instead, a custom-made fitness function is introduced which uses the output of the CNN to determine the quality of each agent. For instance, if the goal of running the simulation is to create agents that primarily exhibit the medium circle pursuit behaviour, the CNN can determine whether or not this is the case. If it is, the predator will receive a high fitness score

based on how strongly the CNN classifies the behaviour. Conversely, if the agent is emitting a different behaviour, the CNN will give the agent a low fitness score. The third option for fitness scoring is to use a combined approach that portions a percentage of the fitness as traditional GP fitness and the remaining percentage as CNN fitness. For example, if the goal is to create a high performing agent that emits a certain kind of behaviour, a combined fitness of 50% GP and 50% CNN can be employed. This means that 50% of the fitness function comes from how many prey were caught by the predator, and the remaining 50% of the fitness is the result of the CNN's classification of behaviour.

A decision to make the CNN model exchangeable was made early on in order to keep the overall system modular. This means that the CNN model file can be swapped for different models at any point. In the experiments seen in Chapter 8, the CNN model comes from the 14^{th} epoch due to its similarities in its classifications when compared to the author's. In future experiments, the CNN model could be exchanged for another model that was trained on more than 6 behaviours, which could allow for more interesting variety in agent behaviour. The final parameters that were introduced are the Python interpreter and file system paths which are needed to combine both systems. Since the GP system was developed using Java and the CNN classifier was developed using Python, the Java ProcessBuilder API is used to connect the two using relative file paths.

7.2 System Architecture

In this section, the overall system architecture and its individual components will be discussed.

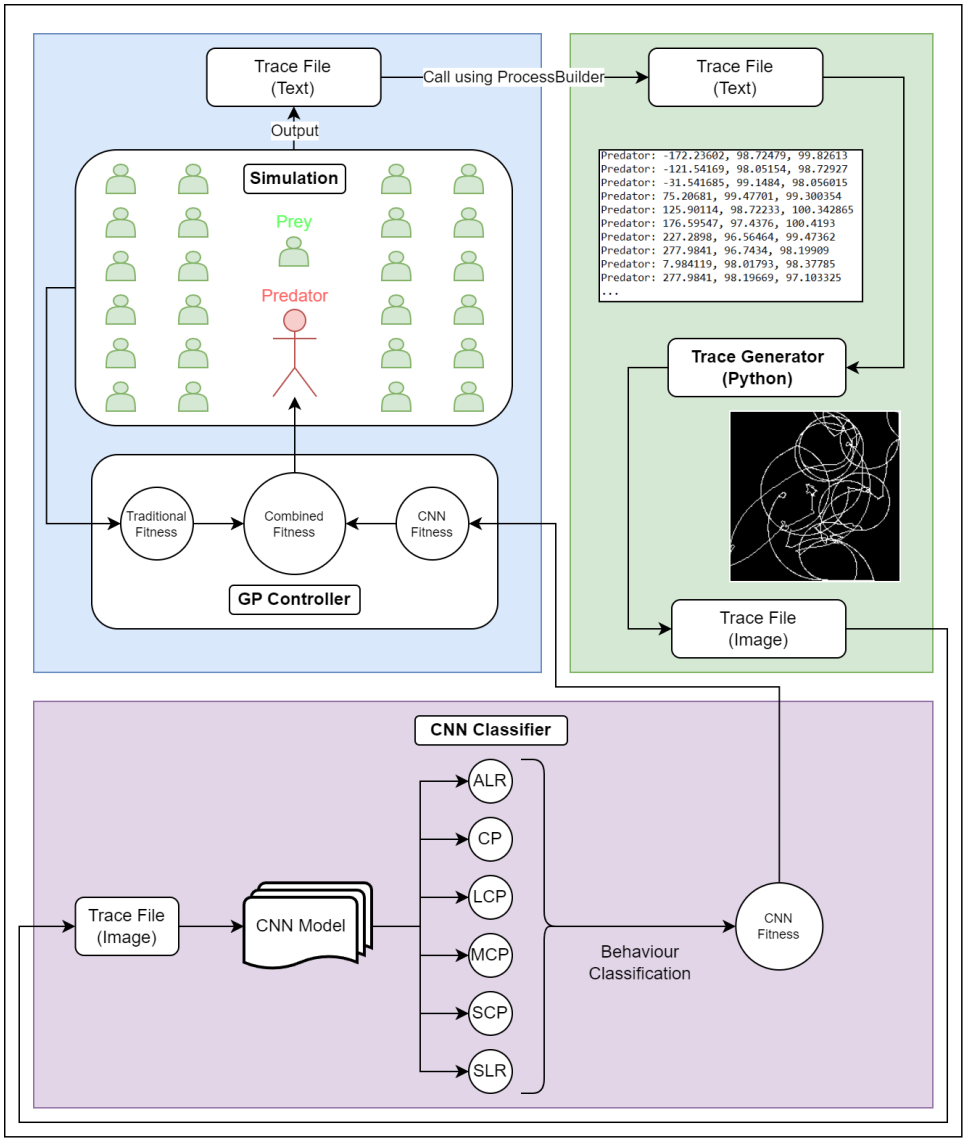


Figure 7.1: System Architecture (Blue: GP Simulation, Green: Trace Generator, Purple: CNN Classifier)

Figure 7.1 gives a high-level overview of the automatic agent generator system and its 3 key components. The first component is the GP simulation which is labelled in blue. This component consists of the simulation itself as well as the GP controller used to control the predator agent in its environment. Once the simulation has ran for 50 generations, a trace file is output which depicts the best performing predator's run. The second component is the trace generator program written in Python which is called using the Java ProcessBuilder API from the GP simulation. This allows for the text-based trace file to be given to the trace generator program as a parameter and used to create the image-based trace file. Once the image-based trace file is

created, the CNN model labelled in purple is used to classify the behaviour of the agent. This classification is then converted into the CNN portion of the fitness and passed to back to the GP simulation to be used for agent evaluation.

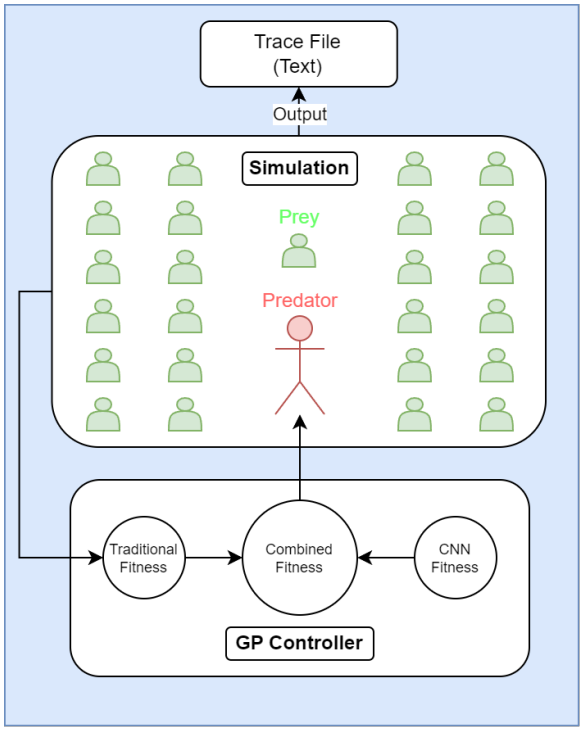


Figure 7.2: GP System Architecture

Figure 7.2 details the isolated system architecture of the GP simulation. The first step to running the simulation on its own is to set the combined fitness parameter to use 100% of the traditional GP fitness. Since the CNN is not in use during an isolated run, the CNN fitness portion of the combined fitness score must be set to 0%. Once the remaining parameters are set such as the population size and crossover/mutation rates, the simulation will run for a given number of generations. Inside the system, the evaluate function is run at the end of each predator's simulated hunt which determines the fitness of the individual based on how many prey they caught. After the final generation has run and all individuals have been evaluated, the best individual from all generations is selected and used in the describe function. The describe function runs the best performer through 30 new simulations in order to get an idea of their performance over multiple runs. Once the agent is finished their 30 runs, 34 text files are dumped as output which contains information about the agent's performance. The first 30 files are the text-based trace file from each run through the simulation. These 30 files can then be run through the trace generator individually to create the image-

based traces to view the type of behaviours the agent is exhibiting. The remaining 4 files are statistics files relating to the population of agents. These statistics include the total prey caught by the best predator, the best GP tree at the end of each generation, the average fitness of an individual across the entire population, and diversity scoring from the CNN if using CNN fitness.

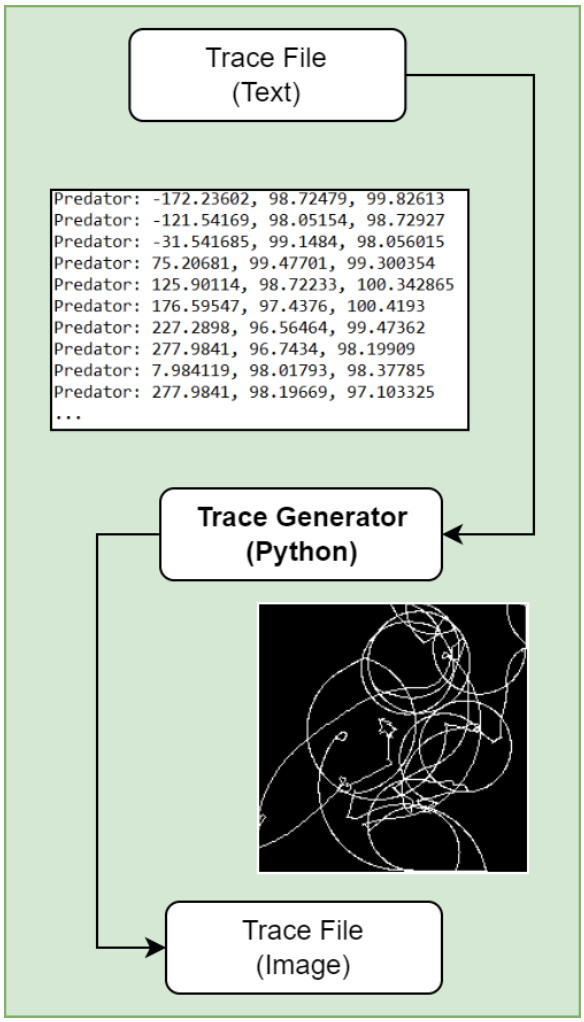


Figure 7.3: Trace System Architecture

Figure 7.3 details the isolated system architecture of the trace generator. In this system, a text-based trace file is required as input which is then converted into an image-based trace file. However, there are a few different methods available to generate traces depending on what is needed. The first method is to pass in a single text-based trace file which is converted into a single image-based trace file. This method is useful when the goal is to classify a single agent which is ideal for generating the input to the CNN. The second method is to take an array of text-based traces and

convert them into an array of image-based traces. This method is useful when the goal is to convert several trace files into images for experiments. The third method is to pass in a single text-based trace file as input, which is converted into 8 text-based trace files and 8 image-based trace files as seen in Figure 6.2. This method is especially useful for creating large amounts of training data. When using the trace generator in combination with the GP system and CNN classifier, the first method is chosen as it allows for the agents to be classified one at a time.

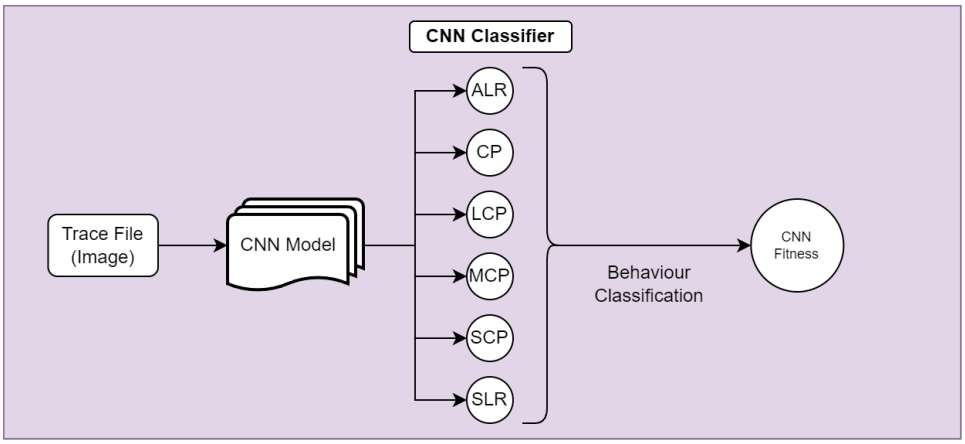


Figure 7.4: CNN Classifier System Architecture

Figure 7.4 details the isolated system architecture of the CNN classifier. The primary purpose of this system is to provide an evaluation of the CNN fitness portion of an agent's combined fitness score. Input to this system is in the form of an image passed in by the trace generator. The trained CNN model is then used to classify the behaviour of the agent using the 6 behaviour categories ALR, CP, LCP, MCP, SCP, and SLR. Once the image has been classified by the model, a customizable fitness function is ran which converts the classification into a CNN fitness score based on the goal of the overall system. For instance, the goal may be to generate agents that exhibit a specific behaviour such as medium circle pursuit. This means that the CNN fitness function would derive high fitness scores from traces that are high in MCP and low in the remaining behaviours. Likewise, it is possible to combine multiple behaviours into a single CNN fitness score. For example, if the goal was to generate agents that exhibit both the small circle pursuit behaviour and the straight line ricochet behaviour. The CNN fitness function could base its evaluation on having equal parts SCP and SLR in the behaviour classification. Overall there exist a multitude of uses for the CNN classifier, especially when combined with GP. In Chapter 8, various experiments will be conducted to test these methods and results are to be discussed.

Chapter 8

Experiment Series II: Automatic Agent Generation

The goal of this series of experiments is to successfully combine the GP simulation and trained CNN model in order to generate unique and interesting intelligent predator agents without sacrificing performance. The chapter is split into 3 primary experiments. The first experiment will determine the ideal combination of fitness and diversity, as well as target unique and interesting agent behaviours. To target unique and interesting behaviours, the diversity score will reward agents that equally split their behaviours closest to 16.67% of each of the 6 categories. The ideal agent would exhibit all 6 categories in some form, which in theory is an extremely complex and difficult task to achieve. The second experiment will use the ideal combination of fitness and diversity found in the first experiment in order to generate agents that exhibit combined behaviours. The diversity score in this experiment will reward agents that equally split their behaviours between two chosen categories closest to 50%. In the final experiment, the combined score will be used to target a specific behaviour, rewarding agents that exhibit 100% of the behaviour and penalizing agents that do not exhibit the behaviour at all.

8.1 Experiment D - Unique Behaviour Generation

In the following experiment, the GP and CNN will be combined in order to generate intelligent agents with the purpose of finding an ideal balance in combined score weightings. There are 5 sets of weightings to be tested: 0F100N, 25F75N, 50F50N, 75F25N, and 100F0N. The 0F100N weighting means that fitness is not being used at all, and the diversity score is being used for 100% of the evaluation process. On

the contrary, the 100F0N weighting means that fitness is being used for the 100% of the evaluation process, and diversity score is not used at all. The other 3 sets of weightings combine both fitness and diversity to create a combined score in order to evaluate the agents.

The section is split into three sets of evaluation techniques. The first set of evaluations will determine how the population of predator agents perform relative to fitness results across each of the 5 combined score weightings. This set of evaluations will then be repeated, but use the diversity score to compare the results across each of the 5 combined score weightings. Lastly, the categories of all agents will be evaluated by the CNN and discussed in the final section. It is important to note that each of the experiments were run 30 times and results are averaged to ensure statistical significance. Furthermore, the Kruskal-Wallis Test will be used in the final section to determine if there is a significant difference between the population of behaviours across the 5 combined score weightings.

8.1.1 Fitness Results

This section presents the fitness scores seen across each set of experiments. The fitness score is evaluated by dividing the number of prey caught by 25 and subtracting the result from 1. An agent with a perfect fitness score that has caught all 25 prey in their simulated hunt will have a fitness score of 0.

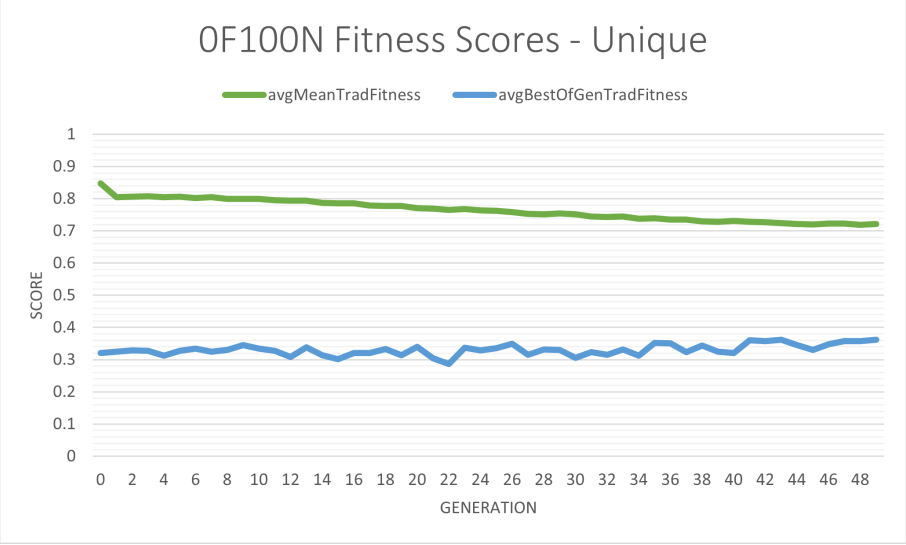


Figure 8.1: 0F100N Fitness Scores - Unique

Figure 8.1 shows the average mean fitness scores and average best fitness scores across the 30 runs of 0F100N combined score. In these runs, only diversity score is

used to evaluate the agents. As shown in the graph, the mean fitness and best fitness of the agents is very poor due to the lack of incentive for the predators to catch prey. Interestingly, the average best fitness gets slightly worse as time goes on. In order to fully understand how these agents perform, data about their diversity score and behaviours will need to be analyzed.

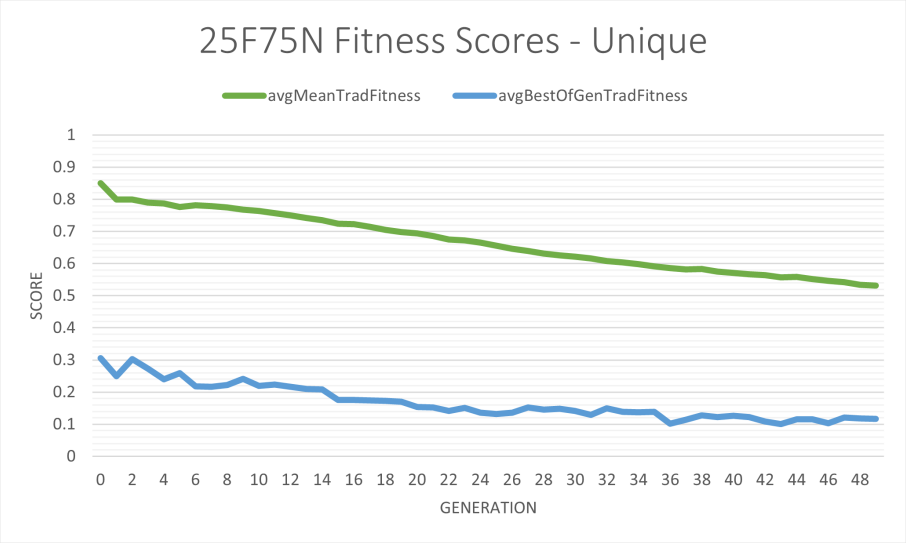


Figure 8.2: 25F75N Fitness Scores - Unique

Figure 8.2 shows the average mean fitness scores and average best fitness scores across the 30 runs of 25F75N combined score. As shown in the graph, the mean fitness and best fitness of the agents is improving, likely due to 25% of the combined score now being comprised of fitness. The improvement from 0% fitness to 25% fitness is quite substantial, so it will be interesting to see how the fitness changes as it is targeted more in future runs.

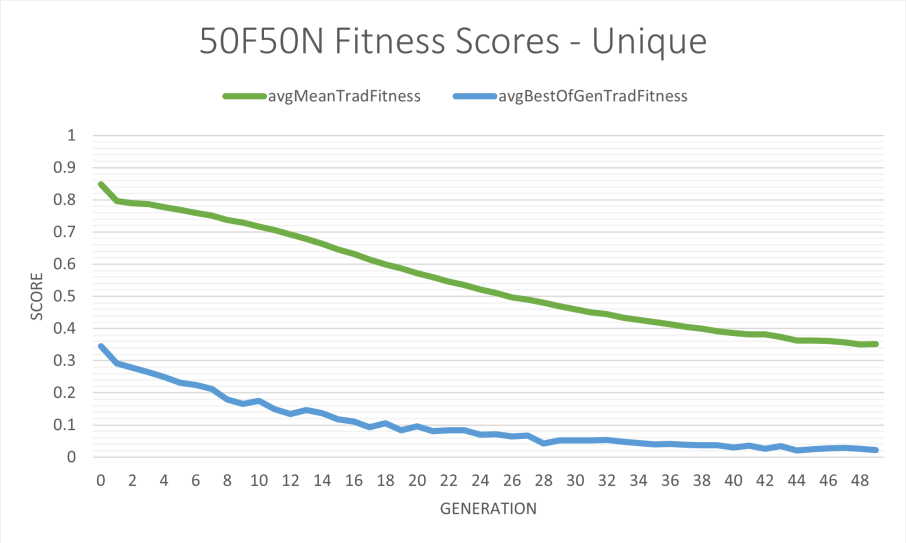


Figure 8.3: 50F50N Fitness Scores - Unique

Figure 8.3 shows the average mean fitness scores and average best fitness scores across the 30 runs of 50F50N combined score. As shown in the graph, the mean fitness and best fitness of the agents continues to improve, especially when compared to the 0F100N and 25F75N experiments. The graph now resembles a descending curve rather than a line and the average best individuals at generation 50 have near-perfect fitness. A comparison can be made that the average *mean* fitness of this experiment is similar to the average *best* fitness of the 0F100N experiment.

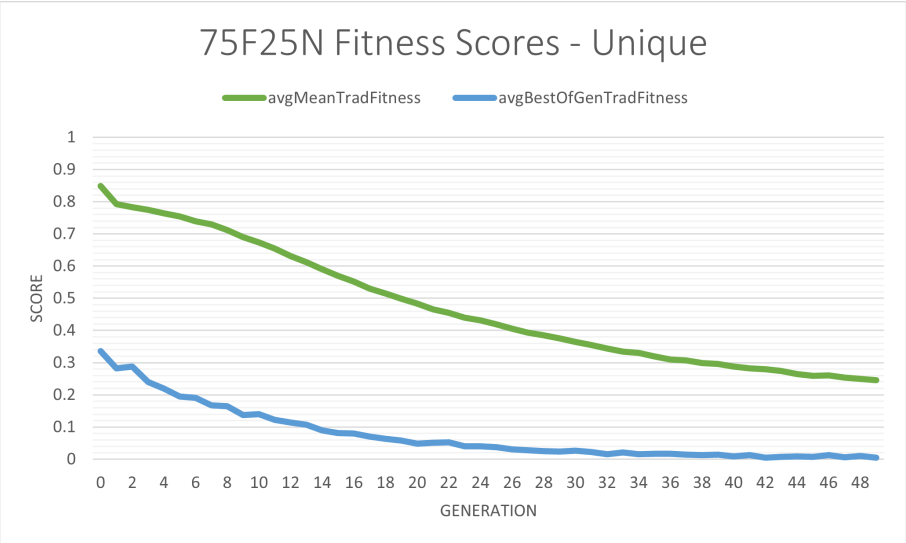


Figure 8.4: 75F25N Fitness Scores - Unique

Figure 8.4 shows the average mean fitness scores and average best fitness scores

across the 30 runs of 75F25N combined score. As shown in the graph, the mean fitness and best fitness continue to show a descending curve from their initial starting point to the end of the 50 generations. The average mean fitness has improved significantly, even when compared to the 50F50N experiment. Furthermore, the average best fitness converges at a near-perfect score after only 30 generations.

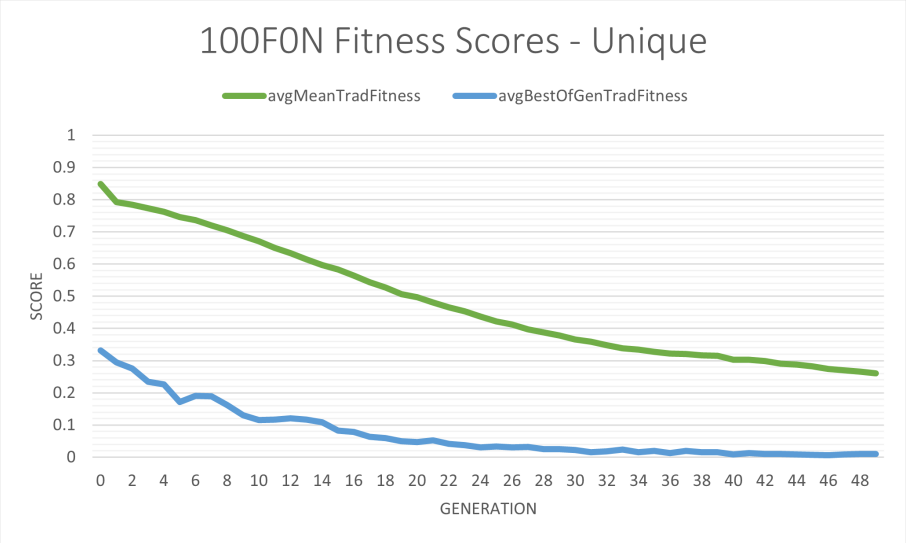


Figure 8.5: 100F0N Fitness Scores - Unique

Figure 8.5 shows the average mean fitness scores and average best fitness scores across the 30 runs of 100F0N combined score. As shown in the graph, the mean fitness and best fitness behave similarly to the 75F25N experiment which shows a descending curve from the initial starting point to the end of the 50 generations. The average best fitness score converges at a near-perfect score after only 28 generations. Overall it is unsurprising that the 100% fitness experiment has one of the highest average fitness scores, but it will be interesting to compare all of the experiments at once.

	Avg. Mean Fitness	**Avg. Best Fitness**	**Avg. Prey Caught**	**# Fit >10 Prey**	**# Unfit <=10 Prey**
0F100N	0.721976	0.361333	4.846667	123	777
25F75N	0.530902	0.117333	8.437778	333	567
50F50N	0.352421	0.022667	12.96889	679	221
75F25N	0.245768	0.005333	14.16111	713	187
100F0N	0.261294	0.010667	14.41	693	207

Table 8.1: Fitness Data - Unique

Table 8.1 outlines several key indicators of fitness across all 5 experiments. The first column represents the average mean fitness across all 30 experiments at the 50^{th} generation. The second column represents the average best fitness across all 30 experiments at the 50^{th} generation. For the final 3 columns, each of the best performing agents from the 30 experiments were ran through the simulation 30 times. The result of this is 900 runs that are used to determine the average prey caught, the number of fit individuals (>10 prey caught), and the number of unfit individuals (<=10 prey caught).

When looking at the average mean fitness and average best fitness columns, there is a clear trend that shows the combined fitness scores with low fitness performing significantly lower than the combined fitness scores with high fitness. There is improvements from 0F to 25F to 50F to 75F, but once the agents reach 100F there is an unexpected decline in mean fitness. This means that the 75F individuals outperform the 100F individuals, even though 25% of their fitness score comes from the CNN. On the contrary, the average number of prey caught in the 100F experiment seem to outperform the average number of prey caught in the 75F experiment. However, the number of fit individuals in the 75F experiment outperforms the number of fit individuals in the 100F experiment.

Overall, Table 8.1 concludes that increasing the amount of fitness used in a combined score improves the number of prey caught during a simulation until 75F. Once the individuals reach 75% fitness, there is no reason to continue adding fitness, and instead the agents can benefit from using the diversity score as part of their fitness instead.

8.1.2 Diversity Results

This section presents the diversity scores seen across each set of experiments. The diversity score is evaluated using the following formula before being normalized:

$$Diversity = (Prediction - 16.67)/(100 - 16.67) \tag{8.1}$$

where *Prediction* represents the CNN's highest categorical prediction from 16.67 to 100. When using this formula, the predator agent is encouraged to combine as many of the 6 behaviours as possible, thus creating an equal number of each behaviour and becoming a truly unique and interesting agent. If an agent exhibits only a single behaviour, they will be heavily punished by the fitness function. On the contrary, agents that exhibit a combination of all 6 behaviours should show a maximum CNN

classification of $100/6 = 16.67\%$. Overall, this is an extremely difficult task for the system to accomplish, and it will be interesting to see what kind of behaviours are created.

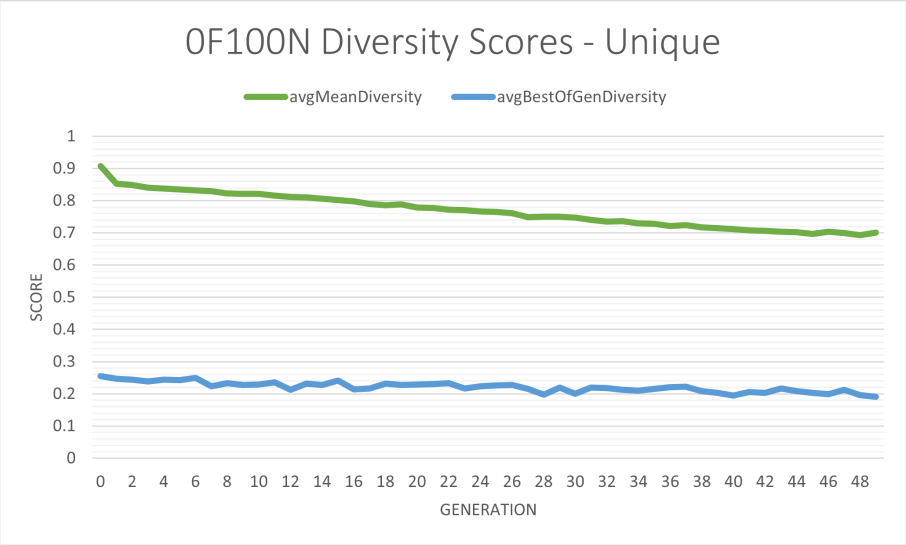


Figure 8.6: 0F100N Diversity Scores - Unique

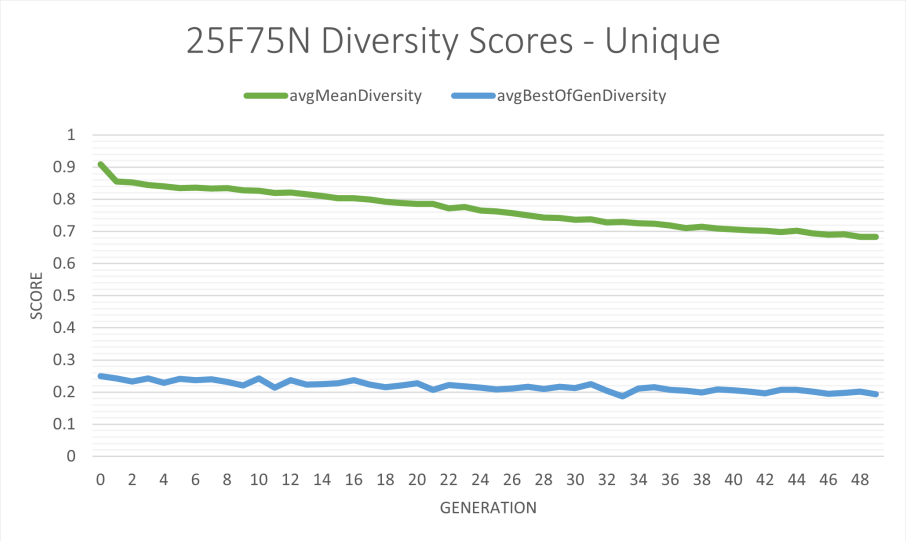


Figure 8.7: 25F75N Diversity Scores - Unique

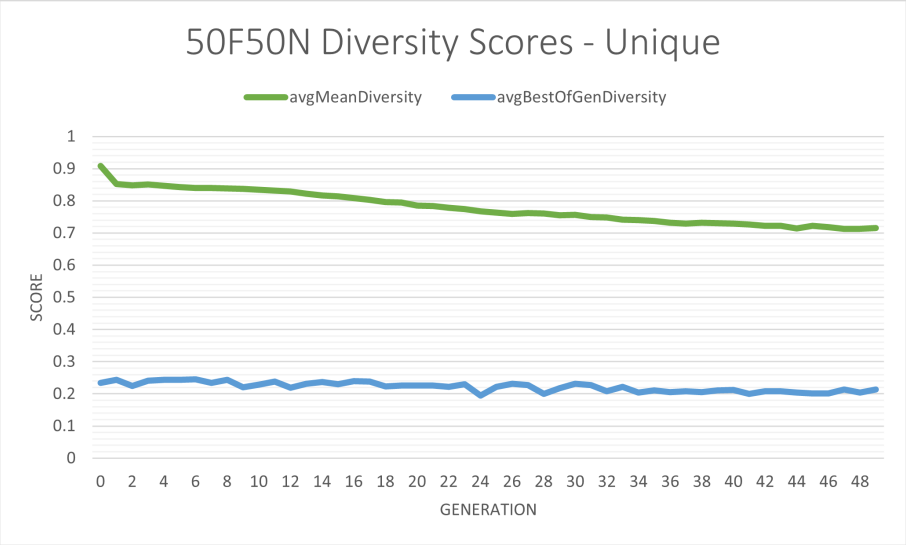


Figure 8.8: 50F50N Diversity Scores - Unique

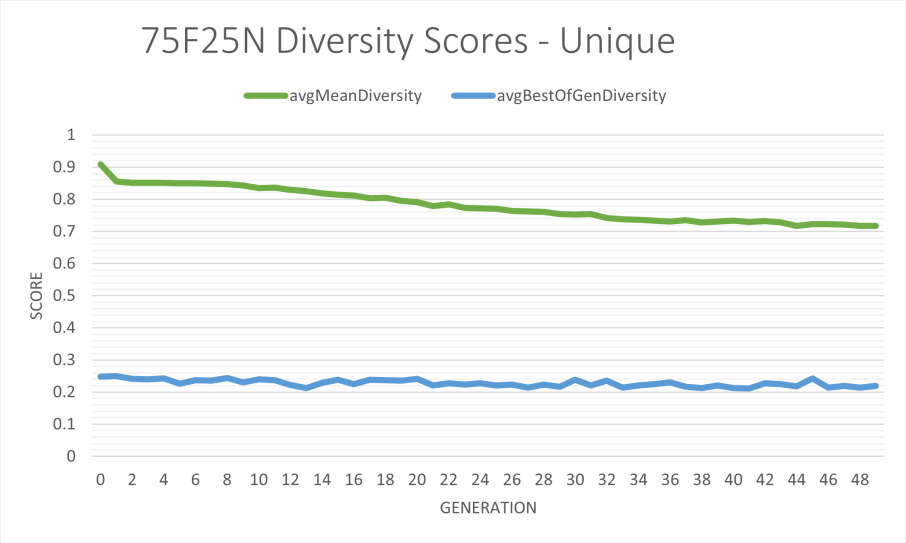


Figure 8.9: 75F25N Diversity Scores - Unique

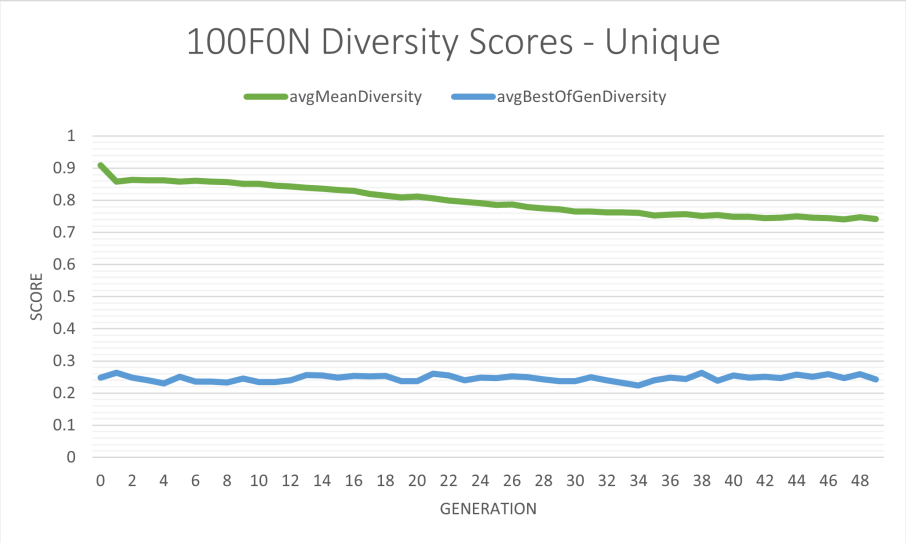


Figure 8.10: 100F0N Diversity Scores - Unique

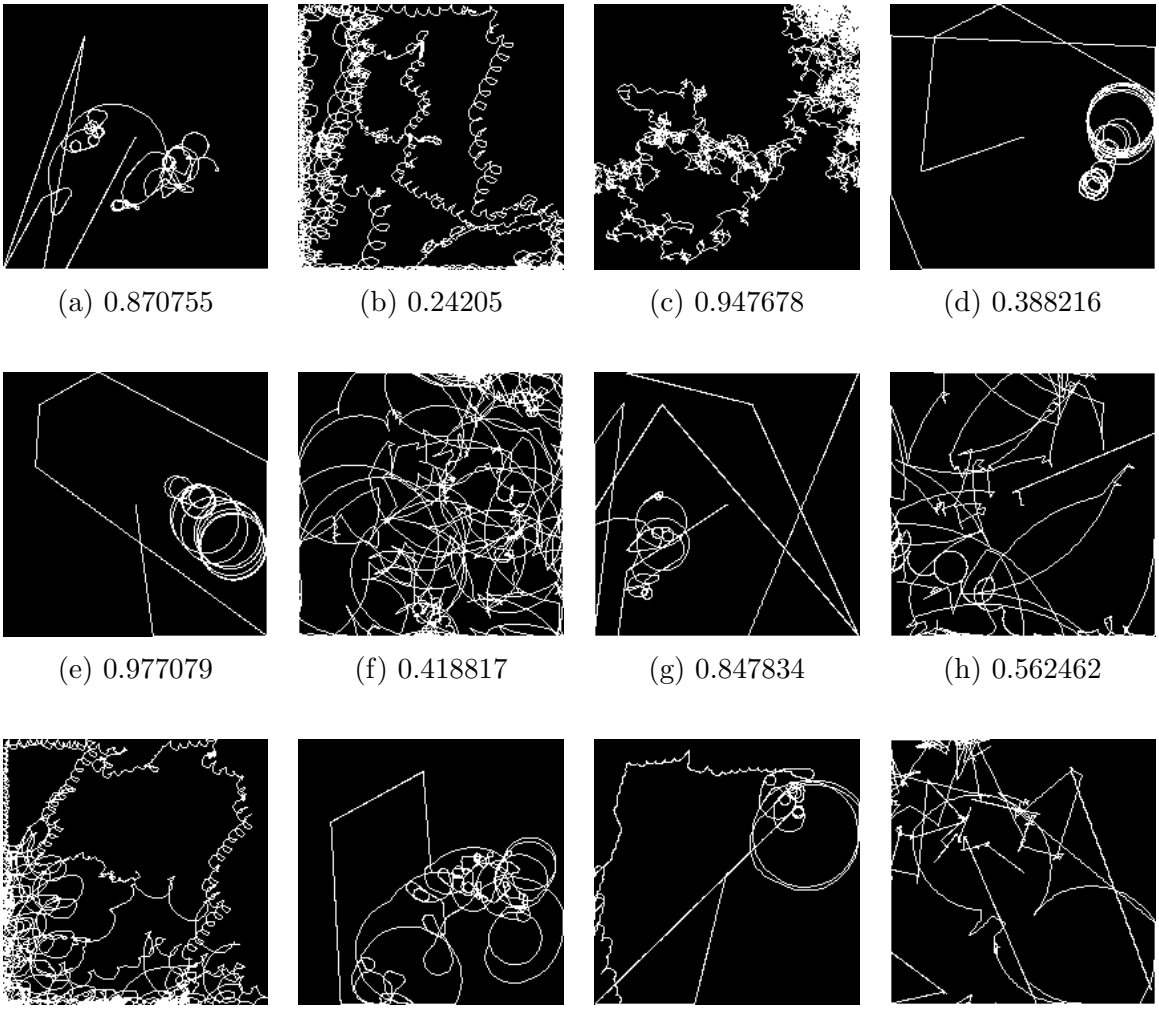
Figures 8.6, 8.7, 8.8, 8.9, 8.10 show the average mean diversity and average best diversity across the 30 runs of all 5 experiments. Surprisingly, there are no significant differences between any of the 5 graphs. They all begin with an average mean diversity of approximately 0.9 and steadily improve until approximately 0.7. Similarly, the average best diversity begins at approximately 0.25 and declines slightly to approximately 0.20.

	Avg. Mean Diversity	**Avg. Best Diversity**
0F100N	0.700692	0.191469
25F75N	0.683133	0.193751
50F50N	0.715925	0.214268
75F25N	0.717446	0.219904
0F100N	0.700692	0.191469

Table 8.2: Diversity Data - Unique

Table 8.2 outlines the average mean diversity and average best diversity of the 5 experiments at generation 50. As previously noted, there are no significant differences between the 5 experimental parameters which is surprising. The reason for this could be that the diversity function is too difficult for the system to optimize. Agents are often seen only exhibiting one or two behaviours, so having an agent exhibit all 6 behaviours at once would be an exceptional outlier. In the future, the diversity

function will be modified in ways that allow for targeted behaviours but with less combinations and the results will be discussed in order to see if there is a limit to what the system can produce.



(a) 0.870755 (b) 0.24205 (c) 0.947678 (d) 0.388216

(e) 0.977079 (f) 0.418817 (g) 0.847834 (h) 0.562462

(i) 0.964959 (j) 0.983439 (k) 0.74703 (l) 0.411496

Figure 8.11: Examples of Experiment D Behaviour Traces

Figure 8.11 gives an example of traces that depict interesting behaviours and some that were close to successfully accomplishing the desired results of this experiment by equally targeting all 6 behaviours in order to create unique and interesting high-performing agents. The values associated with each of the images represent their diversity scores, with values closest to 0 being best. The first thing noticeable about these traces is that they are all very different from one another. This alludes to the possibility of other behaviour types existing that have not been discovered yet. Another noticeable part of these traces is that they vary in terms of arena coverage

which could explain why the fitness scores are lower in the combined score experiments with 0F100N and 25F75N. The diversity function is working against the fitness function because the agent gets confused in its direction when trying to combine all 6 behaviours at once. Overall the experiment was somewhat successful at creating unique intelligent agents but lacking in agents that are high-performing in terms of fitness.

8.1.3 Behaviour Category Results

The following section presents the behaviour category results seen across each set of experiments. The behaviour category is calculated by running the image-based trace through the CNN for classification. Each experiment is run 30 times with the result being the best individual from all generations. This individual is then run through the simulation 30 more times for a combined total of 900 traces per experiment. The purpose of this experiment is to test if changing the ratio of the combined score produces a different number of each behaviour category.

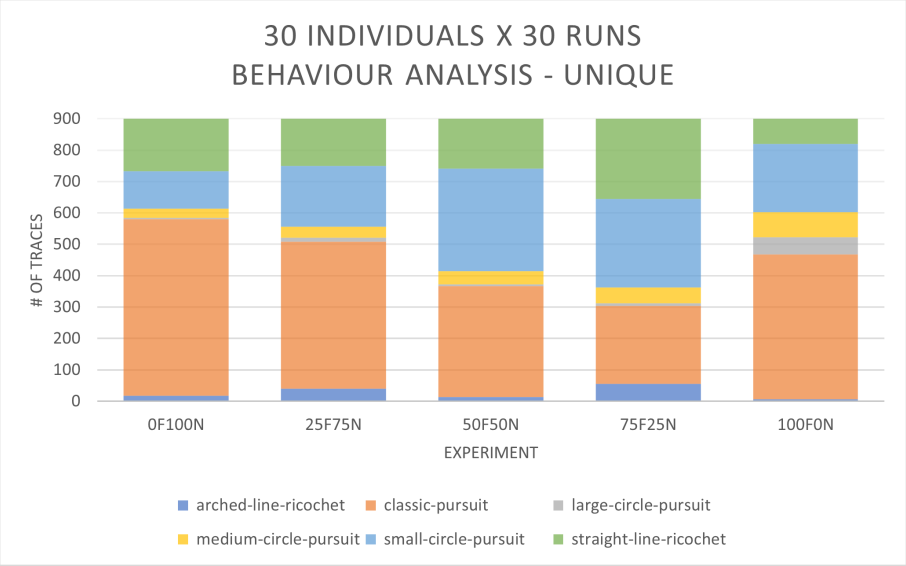


Figure 8.12: Behaviour Analysis - Unique

Figure 8.12 shows the total of all behaviours seen across all 900 runs in each experiment. At first glance there are noticeable differences between each of the experiments. Starting with 0F100N the primary behaviour being exhibited is the classic pursuit behaviour. This is unsurprising because the classic pursuit behaviour does explore the search space so it may be getting stuck at local optima and have trouble evolving into more advanced behaviours. As the amount of fitness increases across

each of the experiments, the number of classic pursuit behaviours decreases up until 100F. This is an indicator that as more fitness is added, the predator agents are able to find new behaviours that may increase their performance.

	ALR	CP	LCP	MCP	SCP	SLR
0F100N	17	563	4	30	119	167
25F75N	40	468	13	35	193	151
50F50N	13	354	5	42	327	159
75F25N	55	248	9	50	283	255
100F0N	6	461	55	81	217	80

Table 8.3: Categorical Data - Unique

Table 8.3 shows the number of each behaviour exhibited in each of the 900 traces per experiment. The most popular behaviours are classic pursuit, small circle pursuit, and straight line ricochet. The least popular behaviours are medium circle pursuit, large circle pursuit, and arched line ricochet. It appears that the 75F25N experiment produces the best spread of behaviours, with the amount of classic pursuit (248), small circle pursuit (283), and straight line ricochet (255) behaviours being nearly equal. However, the remaining behaviours are quite low. Overall the experiment did not seem to produce an equal number of behaviours as hypothesized, likely due to some behaviours being much more complex than others.

	ALR	CP	LCP	MCP	SCP	SLR
H	7.91	9.98	7.64	5.62	4.16	13.88
DF	4	4	4	4	4	4
P-Value	0.095	0.041	0.106	0.229	0.385	0.007
a	0.05	0.05	0.05	0.05	0.05	0.05
Sig	No	Yes	No	No	No	Yes

Table 8.4: Kruskal-Wallis Test - Unique

The Kruskal-Wallis Test is used in this experiment to determine whether or not the group populations have equal dominance. In order to test this, a sample is taken from each of the 30 runs from each of the 6 behaviours. The 6 behaviours are then compared based on their combined score ratio. For example, this test will prove if changing the combined score from 0F100 to 25F75N (or any of the other ratios) will produce a significantly different number of ALR (or any of the other behaviours).

Looking at the ALR, LCP, MCP, and SCP categories, the p-value is higher than alpha which proves that there are no significant differences between any of the combined score ratios when it comes to producing these behaviours. On the contrary, the number of CP and SLR categories are significantly different between the combined score ratios. Looking back at Figure 8.12 and Table 8.3 it can be seen that the CP category has a low of 248 and a high of 563 which is quite substantial. Similarly, the lowest number of SLR is 80 and the highest is 255 which is quite a large difference as well.

To summarize this experiment, it was successful at generating a variety of agents that exhibit different behaviours but it was unsuccessful at creating an equal number of them. It is shown that when changing the combined score ratio of fitness and diversity, it has a significant effect on performance (catching more prey), but does not have any effect on the diversity (for this specific task). In the next experiment, the diversity function will be simplified to allow for further testing.

8.2 Experiment E - Combined Behaviour Generation

In the following experiment, the GP and CNN will be combined in order to generate intelligent agents with two combined behaviours. There are 2 sets of weightings to be tested: 25F75N and 75F25N. The 75F25N weighting was chosen based on its performance in Experiment D and the 25F75N weighting was chosen due to its relationship to the first weighting. In the past experiment it was shown that the medium circle pursuit and arched line ricochet behaviours were two of the most difficult behaviours to create in an agent, most likely due to their high amount of arena coverage and overall complexity. For that reason, these are the two behaviours that are going to be targeted in the following experiment through use of the diversity function.

The section is split into three sets of evaluation techniques. The first set of evaluations will determine how the population of predator agents perform relative to fitness results across both of the combined score weightings. This set of evaluations will then be repeated, but use the diversity score to compare the results across both of the combined score weightings. Lastly, the categories of all agents will be evaluated by the CNN and discussed in the final section. It is important to note that each of the experiments were run 30 times and results are averaged to ensure statistical

significance. Furthermore, the Kruskal-Wallis Test will be used in the final section to determine if there is a significant difference between the population of behaviours across the combined fitness weightings.

8.2.1 Fitness Results

The following section presents the fitness scores seen across each set of experiments. The fitness score is evaluated by dividing the number of prey caught by 25 and subtracting the result from 1. An agent with a perfect fitness score that has caught all 25 prey in their simulated hunt will have a fitness score of 0.

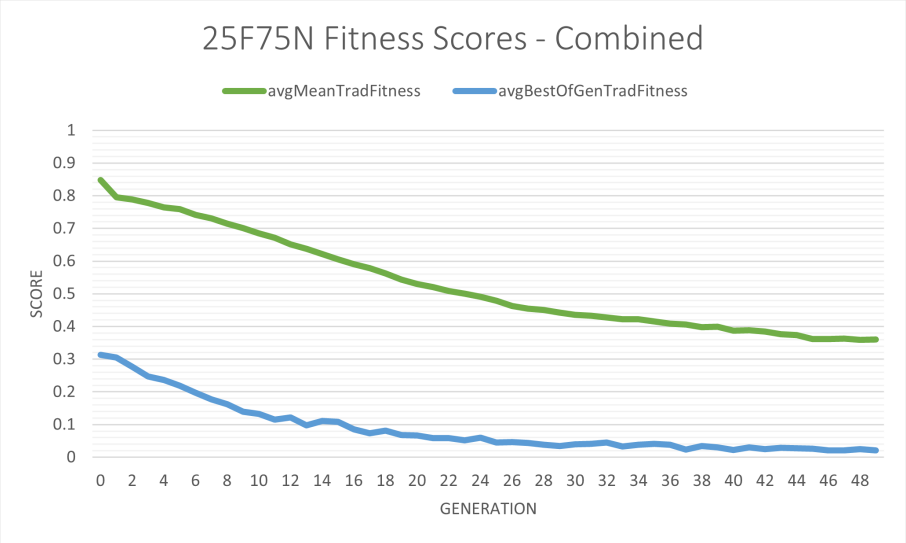


Figure 8.13: 25F75N Fitness Scores - Combined

Figure 8.13 shows the average mean fitness scores and average best fitness scores across the 30 runs of 25F75N combined score. As shown in the graph, the mean fitness and best fitness gradually improve over time and begin to flatten after the 40^{th} generation. The average fitness improves by approximately 0.5 and the best fitness improves by approximately 0.3, both of which are significant improvements and show that the agents are able learn over time, even with only 25% of their combined score being represented by fitness.

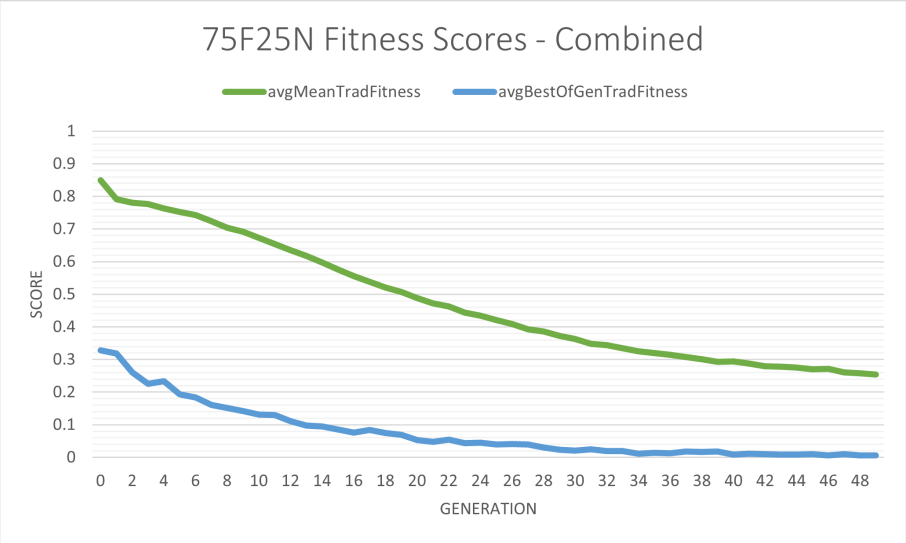


Figure 8.14: 75F25N Fitness Scores - Combined

Figure 8.14 shows the average mean fitness scores and average best fitness scores across the 30 runs of 25F75N combined fitness. As shown in the graph, the mean fitness and best fitness should a descending curve which continues to improve, even possibly past the 50^{th} generation. When compared to the 25F75N experiment, there are noticeable enhancements to the fitness of the agents. One improvement to note is that the best fitness score converges at 0 after 40 generations, meaning that the average best agent has captured all 25 prey.

	Avg. Mean Fitness	**Avg. Best Fitness**	**Avg. Prey Caught**	**# Fit >10 Prey**	**# Unfit <=10 Prey**
25F75N	0.360371	0.021333	11.45444	558	342
75F25N	0.254158	0.006667	14.50111	685	215

Table 8.5: Fitness Data - Combined

Table 8.5 outlines several key indicators of fitness across both experiments. The first column represents the average mean fitness across all 30 experiments at the 50^{th} generation. The second column represents the average best fitness across all 30 experiments at the 50^{th} generation. For the final 3 columns, each of the best performing agents from the 30 experiments were run through the simulation 30 times. The result of this is 900 runs that are used to determine the average prey caught, the number of fit individuals (>10 prey caught), and the number of unfit individuals (<=10 prey caught).

When compared to the first experiment, the 25F75N population seems to have improved significantly. The average mean fitness improved from 0.53 to 0.36, and the number of fit individuals improved from 333 to 558. This indicates that the new diversity function could be pushing the agents to improve. When comparing the 75F25N data to the first experiment, the results are quite similar, although the average number of prey caught increased slightly from 14.16 to 14.50.

Overall, Table 8.5 concludes that increasing the amount of fitness used in a combined score continues to improve the number of prey caught during a simulation. Furthermore, by changing the diversity function to combine 2 behaviours instead of 6, the predators are able to better accomplish their goal of catching prey.

8.2.2 Diversity Results

The following section presents the diversity scores seen across both experiments. The diversity score is evaluated using the following formula before being normalized:

$$Diversity = \sqrt{(ALR - 50)^2 + (MCP - 50)^2} \tag{8.2}$$

ALR represents the CNN's confidence level from 0 to 100 that the trace file is an arched line ricochet behaviour, and MCP represents the CNN's confidence level from 0 to 100 that the trace file is a medium circle pursuit behaviour. By using this formula, the predator is only rewarded when combining the ALR and MCP behaviours, and punished when this is not the case. A predator that exhibits a 100% ALR or 100% MCP prediction is given the same diversity score as a predator that exhibits a 0% ALR or 0% MCP prediction. Overall, this is a very difficult task for the system to accomplish, albeit slightly easier than the unique behaviour experiment, and it will be interesting to see what kind of behaviours are created.

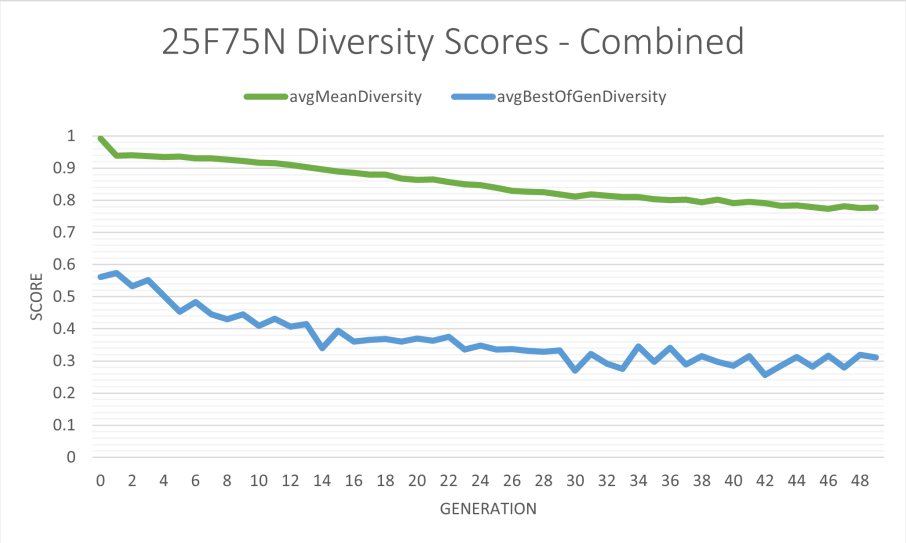


Figure 8.15: 25F75N Diversity Scores - Combined

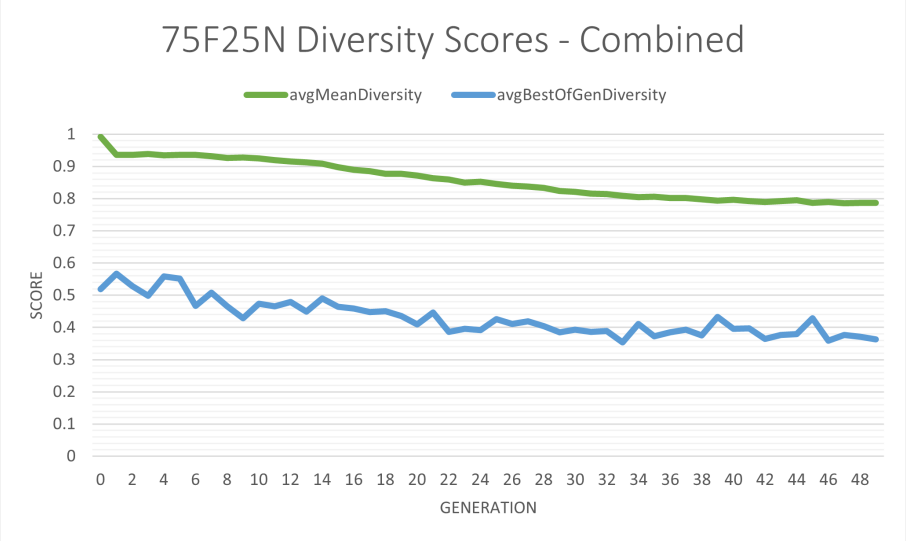


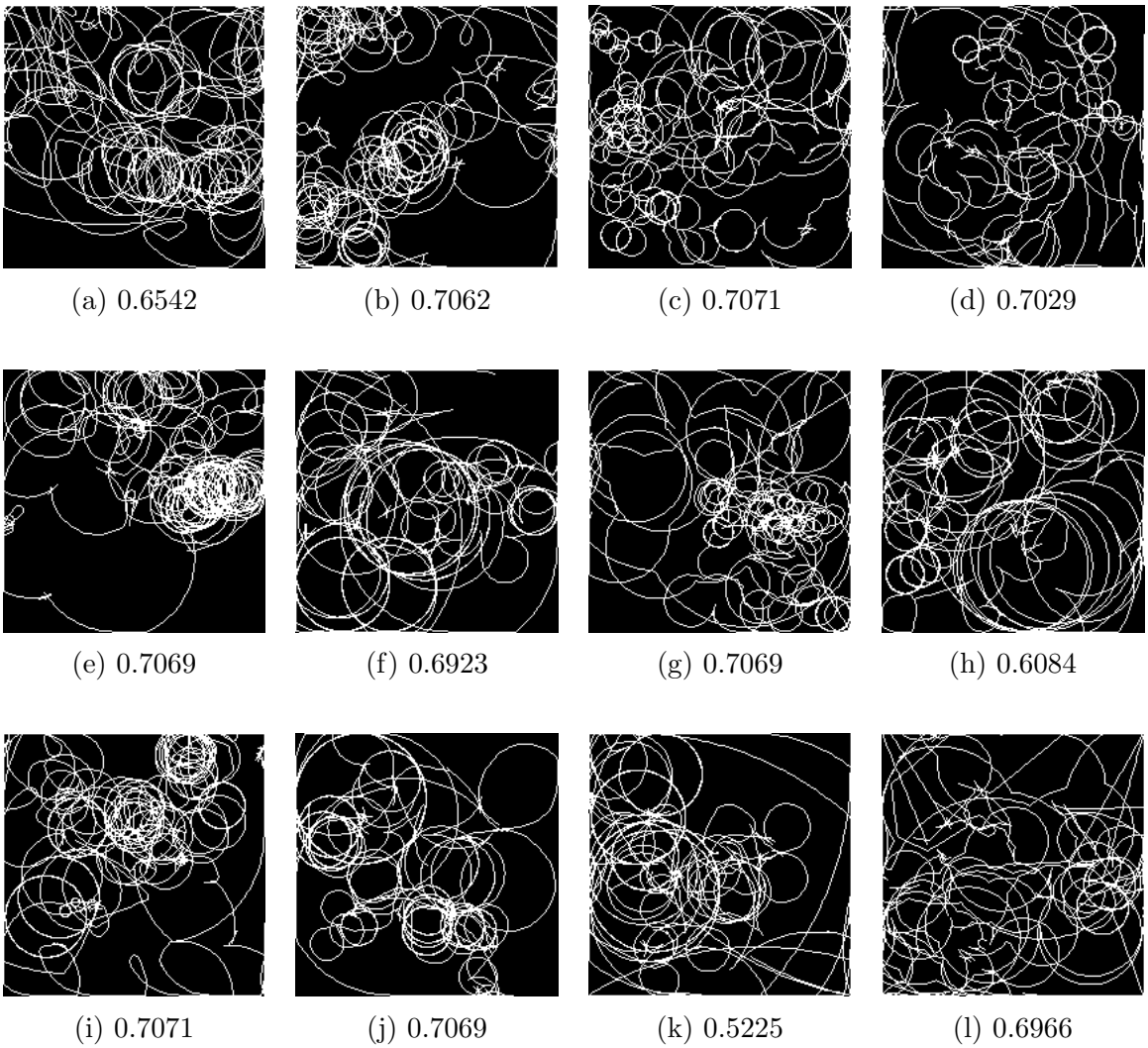
Figure 8.16: 75F25N Diversity Scores - Combined

Figures 8.15 and 8.16 show the average mean diversity and average best diversity across the 30 runs of both experiments. Again, there are no significant differences between either of the experiments, indicating that the diversity function may be too complex for the agents to learn from. Both graphs begin with an average diversity of 1 and end at approximately 0.78 which does show some improvement, but even when compared to the unique experiment, the agents seem to be having a difficult time improving.

	Avg. Mean Diversity	**Avg. Best Diversity**
25F75N	0.77777	0.310684
75F25N	0.786728	0.362529

Table 8.6: Diversity Data - Combined

Table 8.6 outlines the average mean CNN fitness and average best CNN fitness of both experiments at generation 50. As previously noted, there are no significant differences between the two experiments which is surprising. Overall, it seems that the CNN fitness function used is indicative of the performance of the agents.



(a) 0.6542 (b) 0.7062 (c) 0.7071 (d) 0.7029

(e) 0.7069 (f) 0.6923 (g) 0.7069 (h) 0.6084

(i) 0.7071 (j) 0.7069 (k) 0.5225 (l) 0.6966

Figure 8.17: Examples of Experiment E Behaviour Traces

Figure 8.17 gives an example of traces that were close to successfully accomplish-

ing the desired results of this experiment by combining the medium circle pursuit and arched line ricochet behaviour. The values associated with each of the images represent their diversity scores, with values closest to 0 being best. When visually inspecting these traces, it appears that the medium circle pursuit behaviour is much more dominant over the arched line ricochet behaviour which may be a reason why the agents have such a difficult time creating instances where both behaviours represent 50% of the trace. When compared to the unique experiment, these results are successful because the traces are more consistent with what was expected. In the next experiment, only a single behaviour will be targeted.

8.2.3 Behaviour Category Results

The following section presents the behaviour category results seen across each set of experiments. The behaviour category is calculated by running the image-based trace through the CNN for classification. Each experiment is run 30 times with the result being the best individual from all generations. This individual is then run through the simulation 30 more times for a combined total of 900 traces per experiment. The purpose of this experiment is to test if changing the ratio of the combined fitness score produces a different number of each category.

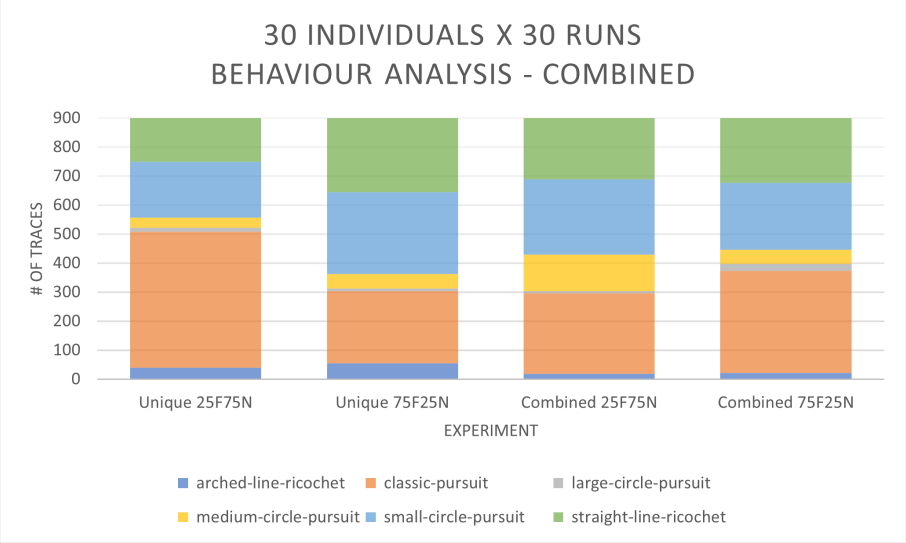


Figure 8.18: Behaviour Analysis - Combined

Figure 8.18 outlines the total of all behaviours seen across all 900 runs in each experiment, including the counterpart experiments from the unique runs. The first noticeable change is seen in the unique 25F75N experiment where a majority of traces

were categorized as the classic pursuit behaviour. However, after changing the CNN fitness function to target MCP and ALR, the number of classic pursuit traces has dwindled, as seen in the combined 25F75N chart. Another interesting feature is that the number of MCP traces has significantly improved in the combined 25F75N experiment which shows that targeting 50% MCP could be rewarding the agents who behave in this way. Unfortunately it does not appear that there was much improvement in the ALR behaviours but this does make sense due to the fact that the agents are punished if they are utilizing 100% ALR in their traces, so this benchmark may not be completely indicative of what is going on beneath the surface. Fortunately, there were plenty of successful traces seen previously that successfully combine the two behaviours.

	ALR	**CP**	**LCP**	**MCP**	**SCP**	**SLR**
25F75N Unique	40	468	13	35	193	151
75F25N Unique	55	248	9	50	283	255
25F75N Combined	19	277	7	126	259	212
75F25N Combined	21	352	25	48	230	224

Table 8.7: Categorical Data - Combined

Table 8.7 shows the number of each behaviour exhibited in each of the 900 traces per experiment. The most significant changes in this experiment are the increase in MCP behaviours from 35 to 126 and decrease in CP behaviours from 468 to 277. On the contrary, the number of ALR behaviours has decreased and the number of SLR behaviours has slightly increased in the 25F75N category. This is unexpected because the SLR category was not targeted, but it is similar to the targeted ALR behaviour. There is also an increase in LCP behaviours from 9 to 25 in the 75F25N category. What could be happening here is that by targeting behaviours with 50% MCP and 50% ALR, the result is that similar behaviours are created such as LCP due to the similarities in their features and arena coverage.

	ALR	MCP
H	1.932	1.006
DF	3	3
P-Value	0.587	0.790
a	0.05	0.05
Sig	No	No

Table 8.8: Kruskal-Wallis Test - Combined

The Kruskal-Wallis Test is used in this experiment to determine whether or not the group of populations have equal dominance. In order to test this, a sample is taken from each of the 30 runs from each of the 2 behaviours. The 2 behaviours are then compared based on their combined score ratio. For example, this test will prove if changing the diversity function or the combined score ratio will produce a significantly different number of ALR or MCP behaviours.

Looking at both categories, the p-value is higher than alpha which proves that there are no significant differences between the 4 sets of categorical data in both ALR and MCP categories. As shown in Table 8.8, the number of ALR categories ranges from 19 to 55 and the number of MCP categories ranges from 35 to 126. Based on the samples taken from this data, there are no significant changes across the two populations. Overall this experiment was successful in generating more MCP traces as well as combining the MCP and ALR behaviours. In the next experiment, the diversity function will be used to target a single behaviour.

8.3 Experiment F - Targeted Behaviour Generation

In the following experiment, the GP and CNN will be combined in order to generate intelligent agents with a targeted behaviour of straight line ricochet. There are 2 sets of weightings to be tested: 25F75N and 75F25N. The 75F25N weighting was chosen based on its performance in Experiments D and E and the 25F75N weighting was chosen due to its relationship to the first weighting. In the past experiments it was shown that the straight line ricochet behaviour was one of the most difficult behaviours to evolve, especially when it came to generating training data for the CNN. The reason for this is most likely due to the amount of GP operations that allow for smooth turning. For this reason, the only behaviour that is going to be

targeted in the following experiment is straight like ricochet, although it could be possible to target any of the 6 (or more) behaviours in the future.

The section is split into three sets of evaluation techniques. The first set of evaluations will determine how the population of predator agents perform relative to traditional results across both of the combined score weightings. This set of evaluations will then be repeated, but use the diversity score to compare the results across both of the combined score weightings. Lastly, the categories of all agents will be evaluated by the CNN and discussed in the final section. It is important to note that each of the experiments were run 30 times and results are averaged to ensure statistical significance. Furthermore, the Kruskal-Wallis Test will be used in the final section to determine if there is a significant difference between the population of behaviours across the combined fitness weightings.

8.3.1 Fitness Results

The following section presents the fitness scores seen across each set of experiments. The fitness score is evaluated by dividing the number of prey caught by 25 and subtracting the result by 1. An agent with a perfect fitness score that has caught all 25 prey in their simulated hunt will have a fitness score of 0.

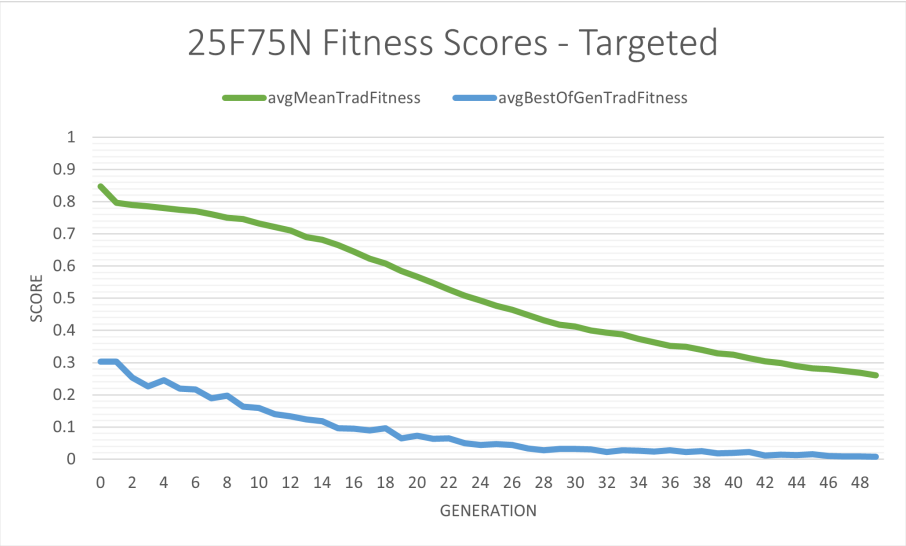


Figure 8.19: 25F75N Fitness Scores - Targeted

Figure 8.19 shows the average mean fitness scores and average best fitness scores across the 30 runs of 25F75N combined fitness. As shown in the graph, the mean fitness and best fitness gradually improve over time, with the best fitness flattening

after the 40^{th} generation. The average fitness improves from 0.85 to 0.26 after 50 generations which is very significant, especially for only 25% of the agents fitness coming from fitness.

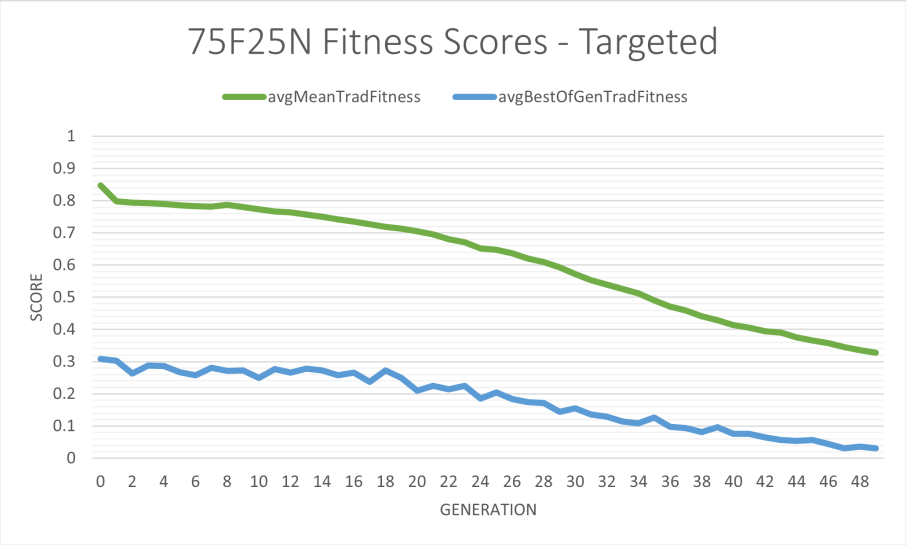


Figure 8.20: 75F25N Fitness Scores - Targeted

Figure 8.20 shows the average mean fitness scores and average best fitness scores across the 30 runs of 75F25N combined fitness. As shown in the graph, the mean fitness and best fitness continuously improve over time, with little to no convergence occurring by the 50^{th} generation. When compared to the 25F75N experiment, the shape of the graph's curve is less steep which may be indicative of lesser performing agents.

	Avg. Mean Fitness	**Avg. Best Fitness**	**Avg. Prey Caught**	**# Fit >10 Prey**	**# Unfit <=10 Prey**
25F75N	0.260933	0.008	13.95111	625	275
75F25N	0.328003	0.030667	13.47111	622	278

Table 8.9: Fitness Data - Targeted

Table 8.9 outlines several key indicators of fitness across both experiments. The first column represents the average mean fitness across all 30 experiments at the 50^{th} generation. The second column represents the average best fitness across all 30 experiments at the 50^{th} generation. For the final 3 columns, each of the best performing agents from the 30 experiments were run through the simulation 30 times. The result of this is 900 runs that are used to determine the average prey caught,

the number of fit individuals (>10 prey caught), and the number of unfit individuals (<=10 prey caught).

An interesting change in fitness is occurring in this experiment where the combined score with *less* fitness is producing agents with *better* fitness scores. This result is very surprising because the past experiments all had higher fitness scores up until the 75F25N experiments which means in this case, the diversity portion of the combined score is heavily influencing the agents. In Table 8.9 it can be seen that every column on the 25F75N experiment has better fitness values than its counterpart in the 75F25N column. However, the results are very similar to one another.

Overall, Table 8.9 concludes that when targeting the straight line ricochet behaviour, increasing the amount of fitness used in a combined score may not improve the number of prey caught and in some cases the number may even decrease. Furthermore, by changing the diversity function to target the straight line ricochet behaviour rather than 2 or 6 behaviours at a time, the predators are able to better accomplish their goal of catching prey.

8.3.2 Diversity Results

The following section presents the diversity scores seen across both experiments. The diversity score is evaluated using the CNN's ALR prediction for a given trace directly and normalizing the value. Overall, this task is not as difficult as the 6 behaviour unique task or the 2 behaviour combined task, and it will be interesting to see what kind of behaviours are created.

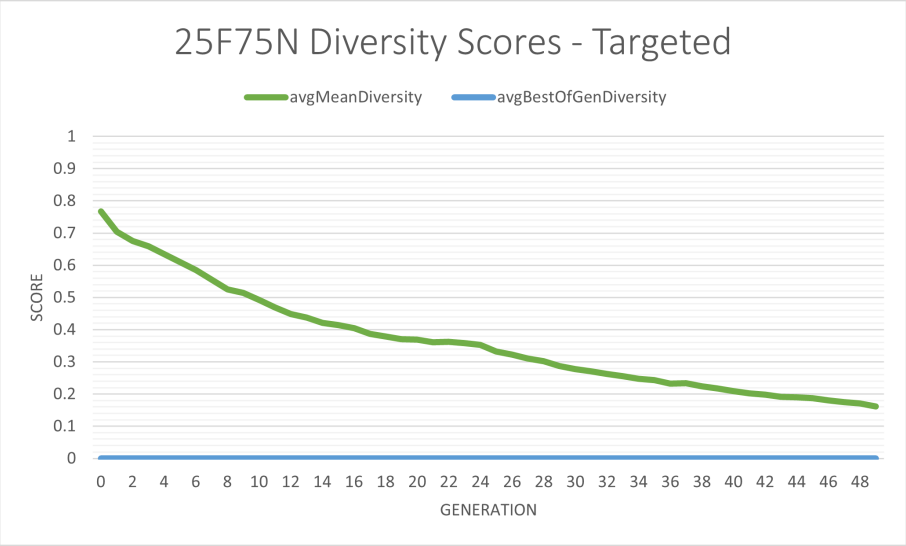


Figure 8.21: 25F75N Diversity Scores - Targeted

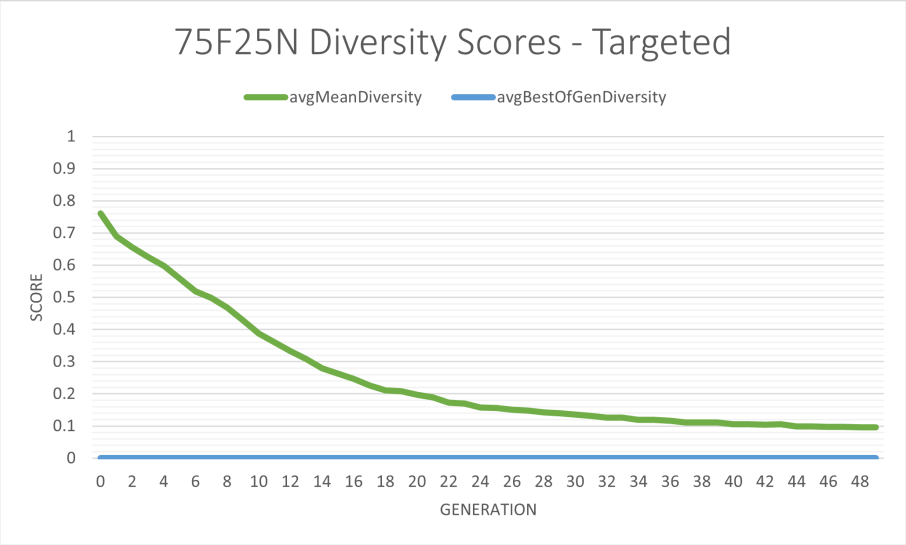


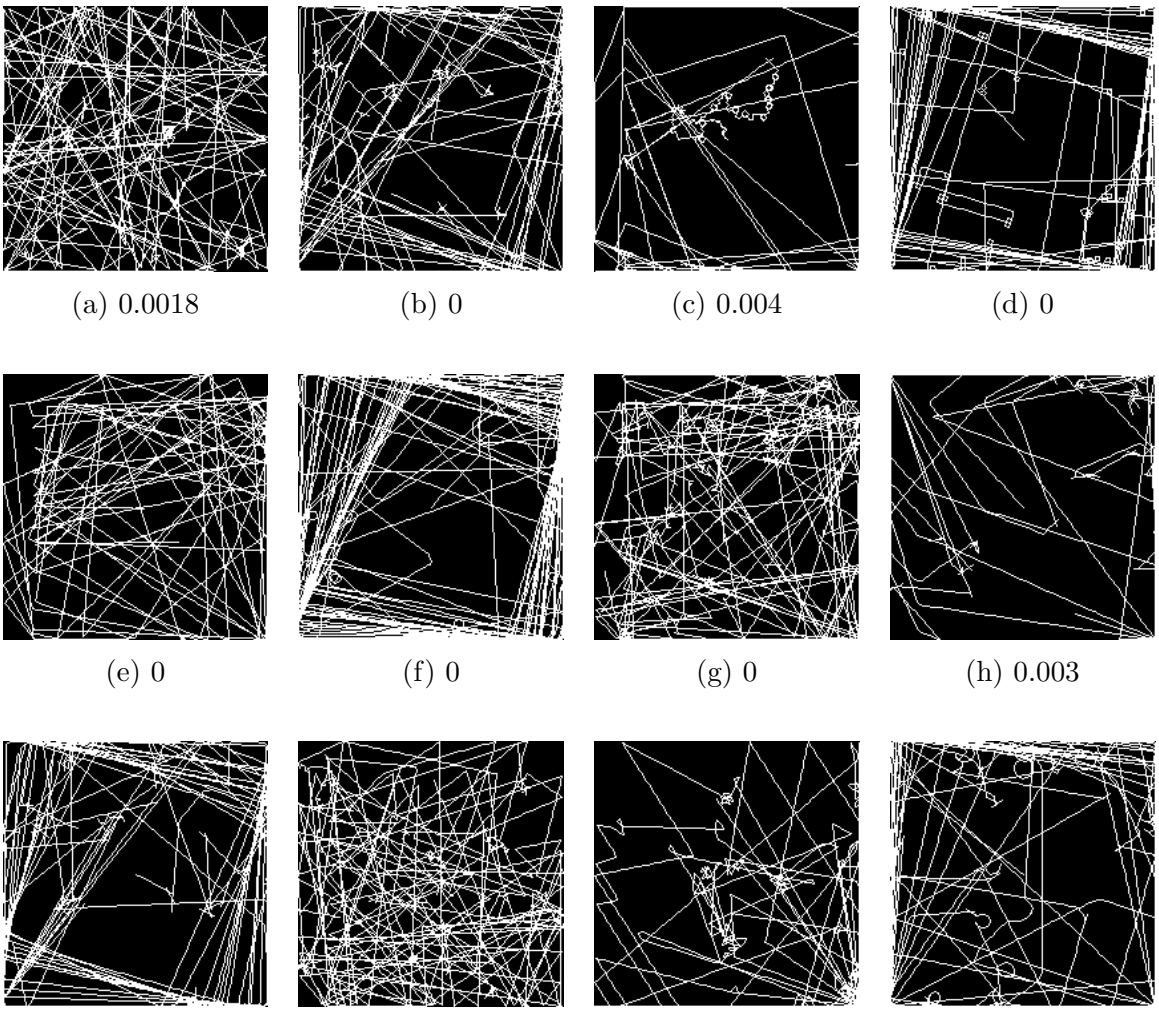
Figure 8.22: 75F25N Diversity Scores - Targeted

Figures 8.21 and 8.22 show the average mean diversity and average best diversity across the 30 runs of both experiments. One similarity between both graphs is that the best diversity scores are always 0 which means that there is at least one 100% categorized ALR trace in all of the populations across every generation. Interestingly, the diversity score in the 25F75N experiment seems to improve linearly whereas the diversity score in the 75F25N experiment seems to show a smooth curve as the population improves over time. Both graphs begin with an average diversity of 0.75 and end between 0.09 and 0.16 which indicates that the population is able to generate a significant amount of ALR agents.

	Avg. Mean CNN Fitness	**Avg. Best CNN Fitness**
25F75N	0.161707	0
75F25N	0.096449	0

Table 8.10: Diversity Data - Targeted

Table 8.10 outlines the average mean diversity and average best diversity of both experiments at generation 50. As previously noted, the average best diversity remains 0 across all generations in both experiments, indicating that there is at least one 100% categorized ALR trace to be found. Overall, it seems that the diversity function is performing better than the previous two experiments and the agents are able to complete their goals.



(a) 0.0018 (b) 0 (c) 0.004 (d) 0

(e) 0 (f) 0 (g) 0 (h) 0.003

(i) 0 (j) 0.0015 (k) 0.0791 (l) 0

Figure 8.23: Examples of Experiment E Behaviour Traces

Figure 8.23 gives an example of traces that successfully accomplish the desired results of this experiment by targeting the straight line ricochet behaviour. The values associated with each of the images represent their diversity scores, with values closest to 0 being best. When visually inspecting these traces, it appears that the SLR behaviour could possibly be broken down into various sub-behaviours such as spiraling ricochet, corner ricochet, or random ricochet. It is very interesting to see the amount of variety in traces, even though the CNN would classify the majority of these in the same way.

8.3.3 Behaviour Category Results

The following section presents the behaviour category results seen across each set of experiments. The behaviour category is calculated by running the image-based trace through the CNN for classification. Each experiment is ran 30 times with the result being the best individual from all generations. This individual is then ran through the simulation 30 more times for a combined total of 900 traces per experiment. The purpose of this experiment is to test if changing the ratio of the combined score produces a different number of SLR or other behaviours.

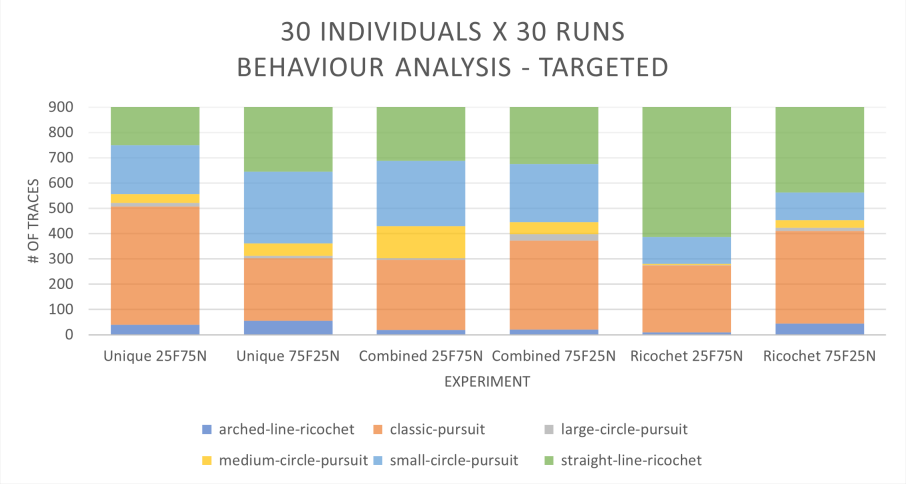


Figure 8.24: Behaviour Analysis - Targeted

Figure 8.24 outlines the total of all behaviours seen across all 900 runs in each experiment, including the counterpart experiments from the unique and combined runs. The first noticeable change is that the number of small circle pursuit behaviours has decreased significantly in the ricochet experiments, and the number of straight line ricochet behaviours has increased. This is highly indicative of a successful experiment because previous to targeting the straight line ricochet behaviour, the number of traces in this category were very low. In the 25F75N experiment over half of the traces are straight line ricochet which could be a reason why the fitness performance was better than in the 75F25N group. Since the straight line ricochet behaviour typically encourages the agent to travel across a large portion of the arena, it is possible that by targeting this behaviour specifically, the number of prey caught will increase.

	ALR	CP	LCP	MCP	SCP	SLR
25F75N Unique	40	468	13	35	193	151
75F25N Unique	55	248	9	50	283	255
25F75N Combined	19	277	7	126	259	212
75F25N Combined	21	352	25	48	230	224
25F75N Targeted	9	266	1	5	105	514
75F25N Targeted	44	366	14	30	109	337

Table 8.11: Categorical Data - Targeted

Table 8.11 shows the number of each behaviour exhibited in each of the 900 traces per experiment. The unique rows refer to the results from Section 8.1. The most significant changes occur in the SCP and SLR columns which show a significant decrease in SCP behaviours by approximately 50% and an increase in SLR behaviours by over 100%. The previous highest amount of SLR behaviours was in the 75F25N unique experiment which produced 255 SLR agents. In the 25F75N targeted experiment the number of SLR agents increased to 514 which is over 50% of the 900 runs. In terms of the remaining categories, their counts remain relatively stable which indicates that targeting a specific behaviour does not have negative effects on the rest of the population.

	SLR
H	11.524
DF	5
P-Value	0.042
a	0.05
Sig	Yes

Table 8.12: Kruskal-Wallis Test - Targeted

The Kruskal-Wallis Test is used in this experiment to determine whether or not the group of populations have equal dominance. In order to test this, a sample is

taken from each of the 30 runs from the SLR behaviour. The behaviours are then compared based on their combined fitness ratio. For example, this test will prove if changing the CNN fitness function or the combined fitness ratio will produce a significantly different number of SLR behaviours.

Looking at the results from the test, the p-value is lower than alpha which proves that the population of SLR solutions is significantly different across experiments. Looking back at Figure 8.24 and Table 8.11 it can be seen that the SLR category has a low of 151 and a high of 514 which is quite substantial, so it makes sense that the test confirms this hypothesis.

To summarize this experiment, it was successful at generating targeted behaviour of straight line ricochet and proven with statistical significance. It was also shown that increasing the fitness portion of the combined score does not always result in an increased number of prey caught. The reason for this is that increasing the diversity to target a specific high-performing behaviour can improve the number of prey caught just as well. In the future, it will be interesting to target other behaviours and see if the results are similar.

Chapter 9

Conclusion

This research explored pushing the boundaries of intelligent agent design by combining GP and CNN with the goal of creating intelligent agents capable of exhibiting a wide range of emergent behaviours while maintaining high performance. The GP played a role of controlling the adaptive game AI in a pursuit domain, with the primary task of capturing prey agents within their simulated environment. To assess their performance, a traditional fitness score counts the number of prey captured during a simulated hunt. This approach has its faults, particularly in terms of diversity, which led to the emergence of repetitive and monotonous behaviours among evolved agents. However, through the use of a CNN trained to recognize agent behaviours, this can be remedied.

Diversity in agent behaviour is an important factor when dealing with uncertainty and can be used to enhance the performance of computer opponents in video games. Drawing inspiration from the works of Cowan [12] and Pozzuoli [41], who delved into the realm of diversifying agent behaviour, and from Chen's work in L-system tree evaluation [9], this research introduced a CNN component. The CNN is used to automatically model the behaviour of each intelligent agent, a task that was previously performed manually. The data used to train the CNN consisted of 12,000 traces representing agent behaviours in the form of images. The resulting CNN model exhibited a high proficiency in detecting and categorizing emergent behaviours into 6 distinct categories.

The first set of experiments looked at the effectiveness of the CNN model. After training the model for 20 epochs, the training accuracy converges at 98% and validation accuracy at 90%. The training loss minimized to 0.065, and the validation loss to 0.37, indicating a robust and well-trained model. While the CNN is able to recognize several behaviour classes such as classic pursuit and three circling behaviours,

it did face challenges in distinguishing between the arched line ricochet and straight line ricochet classes due to their similar movement patterns. Lastly, comparing the CNN's classification of unique traces to the author's classification showed that the majority of traces had less than a 5% difference, reaffirming the CNN's accuracy in identifying behaviours.

One of the new features proposed in this research was to combine GP and CNN for intelligent agent design. In the past, GP had been used to evolve predator controllers, primarily guided by the traditional fitness measurement of prey capture or through novelty search. While these approaches yielded high-performing agents, they often exhibited repetitive and predictable behaviours. Guided by the need to increase diversity in agent behaviour, this thesis introduced the CNN component to the system which allowed for real-time analysis of each predator's simulated hunt, generating accurate depictions of their emergent behaviour. Through the combination of the two systems, an evolutionary pipeline capable of automatically generating intelligent agents with distinct and emergent behaviours is created, all based on a set of customizable user parameters.

The second set of experiments illustrated the potential of combining GP and CNN to produce unique and interesting intelligent predator agents without sacrificing performance. In the first experiment, the goal was to determine the ideal balance between GP and CNN fitness, while also targeting a balanced distribution of behaviours across the 6 categories. The 75F25N split emerged as the most successful approach as it focused primarily on the fitness of the agent, with just enough diversity to encourage the creation of unique and interesting agents without sacrificing performance.

Adding the CNN to the evolutionary process introduced some delay to the system, but as a result it could be used as a powerful tool for targeting specific behaviours. The most impressive results were achieved in the targeted experiment which focused on the straight line ricochet behaviour. Over half of the 900 traces generated in the 25F75N split were classified as straight line ricochet by the CNN which highlights the potential for tailored agent behaviour within this system.

To conclude, the unexpected findings and promising results presented in this research presents a step forward in the field of intelligent agent design. By combining GP and CNN, it not only addresses the concern of repetitive agent behaviour, but also opens up new methods of generating diverse, innovative, and high-performing agents.

9.1 Future Work

In this section, a comprehensive list of future research topics is outlined, aiming to enhance the capabilities and performance of intelligent agents. With these topics, a roadmap is provided for researchers and practitioners seeking to push the boundaries of intelligent agent design and behaviour modeling.

1. **Exploring Different CNN Output Functions:** The impact of various CNN output functions on agent behaviour and performance could be explored. A comparison of activation functions, such as ReLU, Sigmoid, and Tanh can determine the most suitable function for certain tasks or behaviours.

2. **Use of Feed-Forward Neural Networks:** Aside from CNNs, the use of feed-forward neural networks as an alternative architecture for emergent behaviour classification could be promising.

3. **Clustering for Trace Generation:** Clustering can help identify patterns and similarities within agent behaviours, potentially leading to more efficient training and behaviour classification. This approach opens doors to automated trace generation and behaviour categorization.

4. **Automatic Labeling Processes:** Streamlining the labeling process for agent behaviours is critical for efficient training. The development of automated methods such as machine learning-based labeling could significantly reduce human input and improve the scalability of intelligent agent design projects. Automation techniques may involve computer vision and natural language processing for label generation.

5. **Modifying the Agent's Environment:** Introducing environmental changes such as obstacles or removing walls can help to assess the adaptability of intelligent agents.

6. **Expanding into New Domains:** Expanding the research domain into similar tasks such as food gathering or shepherding could introduce fresh challenges and opportunities for intelligent agent design.

7. **Behaviour-Performance Relationship:** Analyzing the relationship between emergent behaviours and fitness could uncover trade-offs involved when selecting various strategies. This could result in the development of more effective intelligent agents.

8. **Sub-Behaviour Targeting:** Breaking down high-level agent behaviours into sub-behaviours could allow researchers to focus on optimizing specific aspects of agent performance. For example, targeting *spiraling* straight line ricochet behaviour separately from straight line ricochet could lead to more specialized and efficient agents.

9. **Dataset Expansion and Diversity:** Expanding the dataset to include more diverse training examples can enhance agent learning. Furthermore, introducing new categories of behaviours to classify can enable agents to adapt to a wide range of scenarios.

Bibliography

[1] Wael Al Enezi and Clark Verbrugge. Skeleton-based multi-agent opponent search. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021.

[2] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[3] Roya Asadi, Norwati Mustapha, and Nasir Sulaiman. A framework for intelligent multi agent system based neural network classification model. *International Journal of Computer Science and Information Security*, 5(1):168–174, Sep 2009.

[4] Yoshua Bengio. *Practical Recommendations for Gradient-Based Training of Deep Architectures*, pages 437–478. Springer Berlin Heidelberg, 2012.

[5] James P Bohnslav, Nivanthika K Wimalasena, Kelsey J Clausing, Yu Y Dai, David A Yarmolinsky, Tomás Cruz, Adam D Kashlan, M Eugenia Chiappe, Lauren L Orefice, Clifford J Woolf, and Christopher D Harvey. Deepethogram, a machine learning pipeline for supervised behavior classification from raw pixels. *eLife*, 10:e63377, Sep 2021.

[6] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[7] Sándor Tihamér Brassai and Attila Kovács. Intelligent agent based low level control of complex robotic systems. In *2021 IEEE 25th International Conference on Intelligent Engineering Systems (INES)*, volume 1, pages 61–66, Jul 2021.

[8] Edmund Burke, Steven Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions on Evolutionary Computation*, 8:47–62, Mar 2004.

[9] Eric Chen. Deep neural network guided evolution of l-system trees. Master's thesis, Brock University, 2021.

[10] Daniil S. Chivilikhin, Vladimir I. Ulyantsev, and Anatoly A. Shalyto. Solving five instances of the artificial ant problem with ant colony optimization. *7th IFAC Conference on Manufacturing Modelling, Management, and Control*, 46(9):1043–1048, 2013.

[11] Alex Clark. Pillow (pil fork). URL: https://pillow.readthedocs.io/en/stable/ (visited on May 6, 2022).

[12] Tyler Cowan. Strategies for evolving diverse and effective behaviours in pursuit domains. Master's thesis, Brock University, 2021.

[13] Harry Davis. Is Monte Carlo tree search machine learning. URL: https://quick-adviser.com/is-monte-carlo-tree-search-machine-learning. (visited on May 6, 2022).

[14] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[15] Elmer R. Escandon and Joseph Campion. Minimax checkers playing GUI: A foundation for AI applications. In *2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, pages 1–4, 2018.

[16] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[17] Lingling Ge, Renlong Hang, Yi Liu, and Qingshan Liu. Comparing the performance of neural network and deep convolutional neural network in estimating soil moisture from satellite observations. *Remote Sensing*, 10(9), 2018.

[18] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. volume 1 of *Foundations of Genetic Algorithms*, pages 69–93. Elsevier, 1991.

[19] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. Introducing novelty search in evolutionary swarm robotics. In *Proceedings of the 8th International Conference on Swarm Intelligence*, ANTS'12, page 85–96. Springer-Verlag, 2012.

[20] Gina Grossi and Brian Ross. Evolved communication strategies and emergent behaviour of multi-agents in pursuit domains. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 110–117, 2017.

[21] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[22] Dr. Rozita Jamili oskouei, Hamidreza Varzeghani, and Zahra Samadyar. Intelligent agents: A comprehensive survey. *International Journal of Electronics Communication and Computer Engineering (2278–4209)*, 5:790–798, Jun 2014.

[23] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, Dec 2014.

[24] John R Koza. Non-linear genetic algorithms for solving problems, June 19 1990. US Patent 4,935,877.

[25] John R. Koza. Genetic programming - on the programming of computers by means of natural selection. In *Complex Adaptive Systems*, 1993.

[26] John R. Koza. Evolution of emergent cooperative behavior using genetic programming. In Ray Paton, editor, *Computing with Biological Metaphors*, pages 280–297. Chapman & Hall, 1994.

[27] John R. Koza, Jonathan Roughgarden, and James P. Rice. Evolution of food-foraging strategies for the caribbean anolis lizard using genetic programming. *Adaptive Behavior*, 1(2):171–199, 1992.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[29] Joel Lehman and Kenneth Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19:189–223, Jun 2011.

[30] Joel Lehman and Kenneth O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 211—-218. Association for Computing Machinery, 2011.

[31] Xiang Liu and Daoxiong Gong. A comparative study of a-star algorithms for search and rescue in perfect maze. In *2011 International Conference on Electric Information and Control Engineering*, pages 24–27, 2011.

[32] Simon Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius. Artificial and computational intelligence in games (dagstuhl seminar 12191). *Dagstuhl Reports*, 2:43–70, 01 2012.

[33] Sean Luke. ECJ evolutionary computation library, 1998. URL: http://cs.gmu.edu/∼eclab/projects/ecj/ (visited on May 6, 2022).

[34] Celeste Mason and Frank Steinicke. Personalization of intelligent virtual agents for motion training in social settings. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 319–322, 2022.

[35] Hanlin Mo and Guoying Zhao. Ric-cnn: rotation-invariant coordinate convolutional neural network. arXiv preprint arXiv:2211.11812, 2022.

[36] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909, 2015.

[37] Rajendran Nirthika, Siyamalan Manivannan, and Amirthalingam Ramanan. An experimental study on convolutional neural network-based pooling techniques for the classification of hep-2 cell images. In *2021 10th International Conference on Information and Automation for Sustainability (ICIAfS)*, pages 281–286, 2021.

[38] Pietro S. Oliveto, Dirk Sudholt, and Christine Zarges. On the benefits and risks of using fitness sharing for multimodal optimisation. *Theoretical Computer Science*, 773:53–70, 2019.

[39] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:387–434, 2005.

[40] G. Parker and I. Parashkevov. Cyclic genetic algorithm with conditional branching in a predator-prey scenario. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2923–2928 Vol. 3, 2005.

[41] Andrew Pozzuoli. Diversifying emergent behaviours with age-layered map-elites. Master's thesis, Brock University, 2022.

[42] Juan Reverte, Francisco Gallego, Rosana Satorre, and Faraón Llorens. Cooperation strategies for pursuit games: From a greedy to an evolutive approach. In *MICAI 2008: Advances in Artificial Intelligence*, pages 806–815. Springer Berlin Heidelberg, 2008.

[43] Philipp Rohlfshagen, Jialin Liu, Diego Perez-Liebana, and Simon M. Lucas. Pacman conquers academia: Two decades of research using a classic arcade game. *IEEE Transactions on Games*, 10(3):233–256, 2018.

[44] Jonathan Roughgarden, Robert M May, and Simon A. Levin, editors. *Perspectives in Ecological Theory*. Princeton University Press, 1989.

[45] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[46] Yerkezhan Seitbekova and Timur Bakibayev. Predator-prey interaction multiagent modelling. In *2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–5, 2018.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*. 2015.

[48] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[49] Haohan Wang, Bhiksha Raj, and Eric P. Xing. On the origin of deep learning. *CoRR*, abs/1702.07800, 2017.

[50] Gregory M. Werner and Michael G. Dyer. Evolution of herding behavior in artificial animals. In *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*. The MIT Press, Apr 1993.

[51] Georgios N. Yannakakis and Julian Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):317–335, 2015.

[52] İhsan Şahin and Tufan Kumbasar. Catch me if you can: A pursuit-evasion game with intelligent agents in the unity 3d game environment. In *2020 International Conference on Electrical Engineering (ICEE)*, pages 1–6, 2020.

Appendix A

Additional Population Data

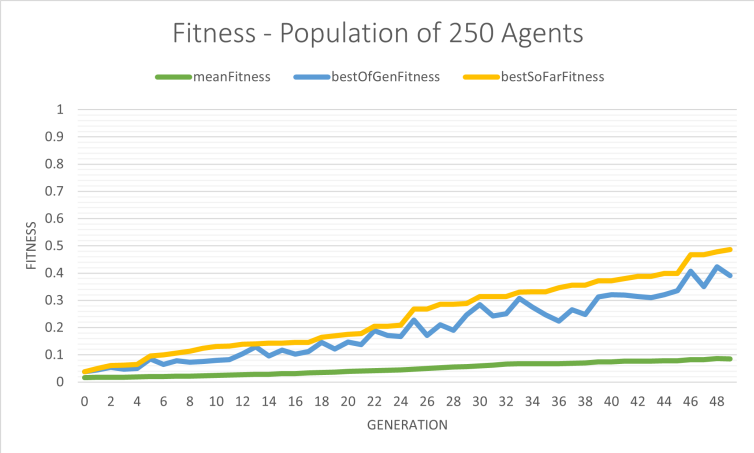


Figure A.1: Fitness - Population of 250 Agents

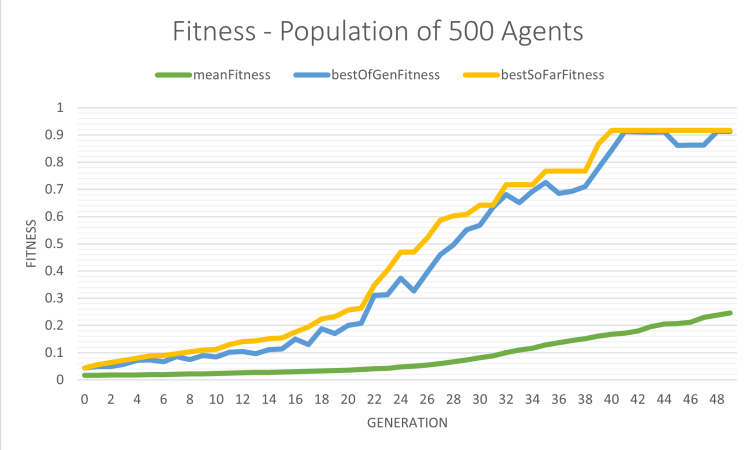


Figure A.2: Fitness - Population of 500 Agents

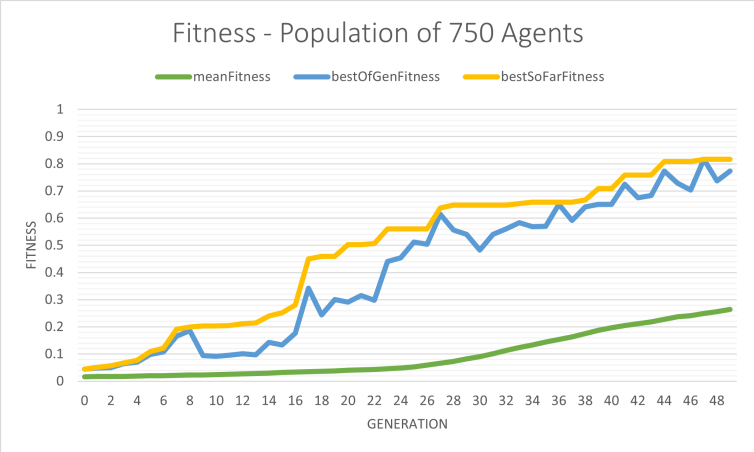


Figure A.3: Fitness - Population of 750 Agents

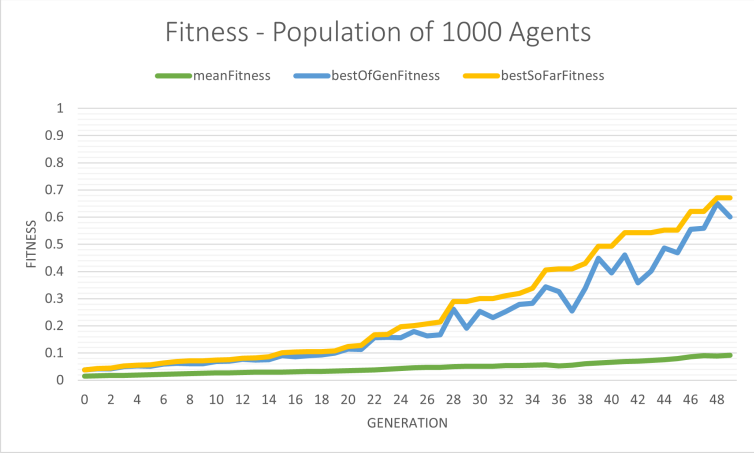
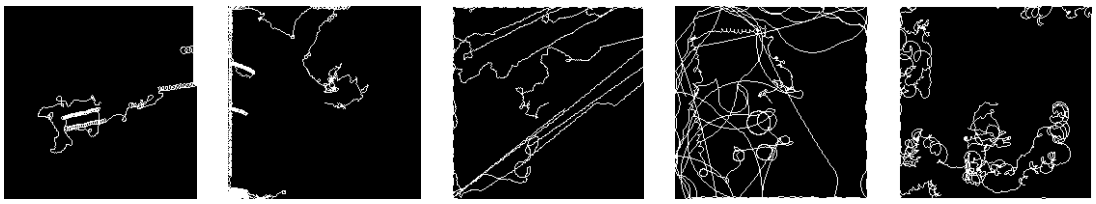


Figure A.4: Fitness - Population of 1000 Agents

Appendix B

Author & CNN Unique Traces Classification

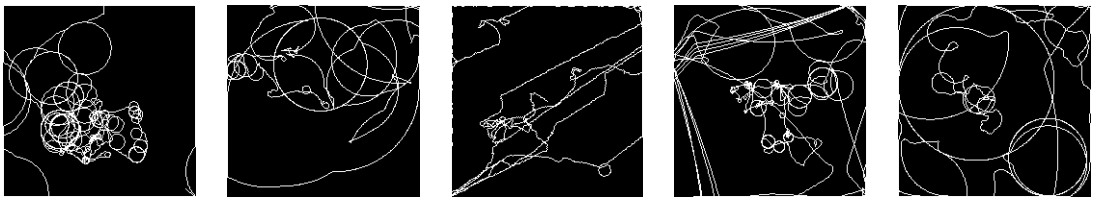


(a) Trace 1 (b) Trace 2 (c) Trace 3 (d) Trace 4 (e) Trace 5

Figure B.1: Unique Traces 1 - 5

	Trace 1		Trace 2		Trace 3		Trace 4		Trace 5	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	0	0	0.18	0	91.84	90	16.06	70	0.04	0
CP	99.9	90	74.75	80	0.05	5	1.11	10	3.38	70
LCP	0	0	0.18	0	0.26	0	5.23	0	0.04	0
MCP	0	0	0.18	0	0.07	0	0.87	0	0.09	0
SCP	0.08	10	16.75	20	0.05	0	5.3	5	96.41	30
SLR	0	0	7.97	0	7.72	5	71.42	15	0.04	0

Table B.1: CNN vs. Author Classification 1-5



(a) Trace 6 (b) Trace 7 (c) Trace 8 (d) Trace 9 (e) Trace 10

Figure B.2: Unique Traces 6 - 10

	Trace 6		Trace 7		Trace 8		Trace 9		Trace 10	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	0.01	0	62.63	75	96.43	40	1.22	10	16.94	10
CP	0.01	0	0.17	0	0.12	10	0.34	5	0.05	0
LCP	0.32	0	34.09	20	0.8	0	6.83	10	82.69	85
MCP	60.49	70	2.21	0	0.22	0	76.58	50	0.23	5
SCP	39.16	30	0.17	5	0.09	0	2.82	10	0.05	0
SLR	0.01	0	0.73	0	2.35	50	12.21	15	0.05	0

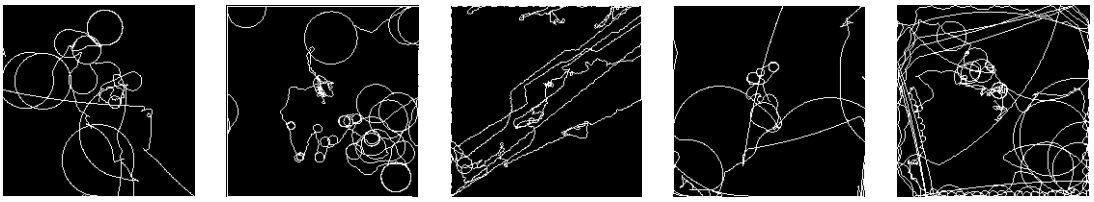
Table B.2: CNN vs. Author Classification 6-10

(a) Trace 11 (b) Trace 12 (c) Trace 13 (d) Trace 14 (e) Trace 15

Figure B.3: Unique Traces 11 - 15

	Trace 11		Trace 12		Trace 13		Trace 14		Trace 15	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	8.06	5	0.01	10	11.12	60	0.65	0	68.61	25
CP	0.2	5	0.01	20	0.35	10	0.03	0	1.17	5
LCP	0.08	0	0.5	20	1.99	0	2.86	5	3.3	0
MCP	0.08	0	99.4	30	76.65	20	2.31	5	3	5
SCP	0.08	0	0.08	20	1.52	0	0.03	0	0.35	0
SLR	91.49	90	0.01	0	8.37	10	94.12	90	23.57	65

Table B.3: CNN vs. Author Classification 11-15

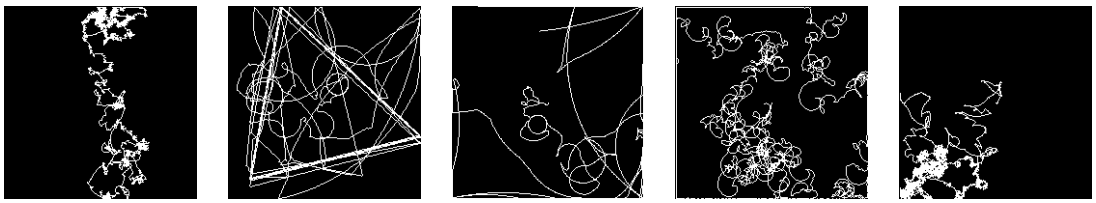


(a) Trace 16 (b) Trace 17 (c) Trace 18 (d) Trace 19 (e) Trace 20

Figure B.4: Unique Traces 15 - 20

	Trace 16		Trace 17		Trace 18		Trace 19		Trace 20	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	0.18	0	0.27	0	83.57	35	12.43	50	0.02	0
CP	0.17	0	0.27	0	0.95	0	1.45	0	0	10
LCP	0.6	5	0.34	0	1.52	0	34.15	0	0.02	10
MCP	98.57	90	4.84	50	2.3	0	2.76	0	0.01	20
SCP	0.38	0	87.04	50	0.25	5	19.22	30	0	10
SLR	0.1	5	7.25	0	11.42	60	29.98	20	99.95	50

Table B.4: CNN vs. Author Classification 16-20

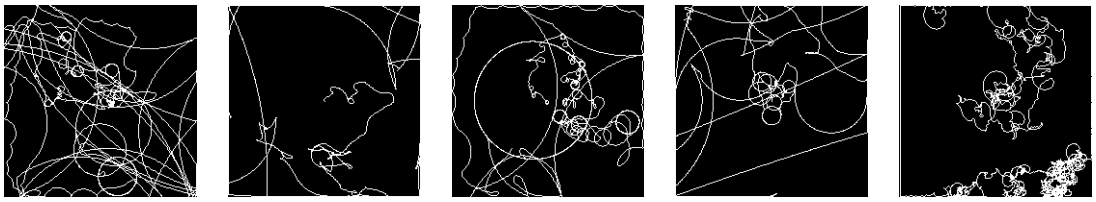


(a) Trace 21 (b) Trace 22 (c) Trace 23 (d) Trace 24 (e) Trace 25

Figure B.5: Unique Traces 21 - 25

	Trace 21		Trace 22		Trace 23		Trace 24		Trace 25	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	0.02	0	0.17	10	1.27	50	0	0	0.17	0
CP	89.58	90	0.02	0	0.24	0	0	30	47.62	50
LCP	0.02	0	2.3	0	0.05	0	0	0	0.17	0
MCP	0.02	0	2.97	10	0.05	20	0.01	0	0.69	0
SCP	10.32	10	0.02	0	0.09	0	99.99	70	51.17	50
SLR	0.02	0	94.52	80	98.3	30	0	0	0.17	0

Table B.5: CNN vs. Author Classification 21-25

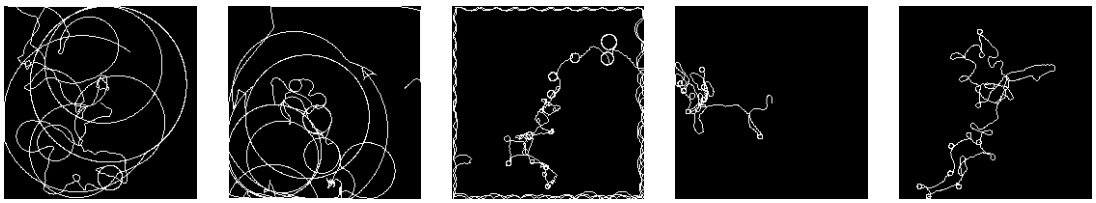


(a) Trace 26 (b) Trace 27 (c) Trace 28 (d) Trace 29 (e) Trace 30

Figure B.6: Unique Traces 26 - 30

	Trace 26		Trace 27		Trace 28		Trace 29		Trace 30	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	30.48	70	7.87	70	0.24	0	96.17	50	0	0
CP	0.14	0	6.94	10	0.29	0	0.09	0	0.01	10
LCP	0.4	0	0.53	0	7.47	10	1.18	0	0	0
MCP	1.36	10	0.53	0	58.81	60	0.08	20	0	0
SCP	0.14	10	9.88	0	32.95	30	0.08	10	99.97	90
SLR	67.47	10	74.24	20	0.24	0	2.39	30	0	0

Table B.6: CNN vs. Author Classification 26-30

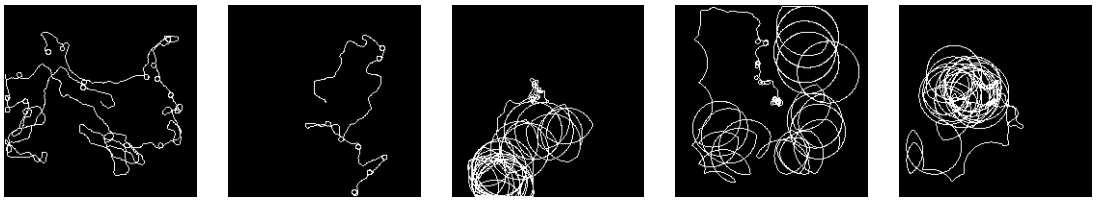


(a) Trace 31 (b) Trace 32 (c) Trace 33 (d) Trace 34 (e) Trace 35

Figure B.7: Unique Traces 31 - 35

	Trace 31		Trace 32		Trace 33		Trace 34		Trace 35	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	4.14	10	56	20	0.3	0	0.01	0	0.01	0
CP	0	0	0.03	0	7.28	20	93.39	50	98.35	50
LCP	95.83	80	43.88	50	0.3	0	0.01	0	0.01	0
MCP	0.02	10	0.03	30	0.63	0	0.01	0	0.01	0
SCP	0	0	0.03	0	88.37	80	6.55	50	1.61	50
SLR	0	0	0.03	0	3.12	0	0.01	0	0.01	0

Table B.7: CNN vs. Author Classification 31-35



(a) Trace 36 (b) Trace 37 (c) Trace 38 (d) Trace 39 (e) Trace 40

Figure B.8: Unique Traces 36 - 40

	Trace 36		Trace 37		Trace 38		Trace 39		Trace 40	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	0.1	0	0	0	0.02	0	0	0	0.01	0
CP	0.83	50	99.91	90	0.02	0	0	0	0.01	5
LCP	0.16	0	0	0	0.02	0	0.09	0	0.01	0
MCP	93.12	0	0	0	97.03	95	99.9	95	25.75	95
SCP	5.71	50	0.08	10	2.91	5	0.01	5	74.22	0
SLR	0.1	0	0	0	0.02	0	0	0	0.01	0

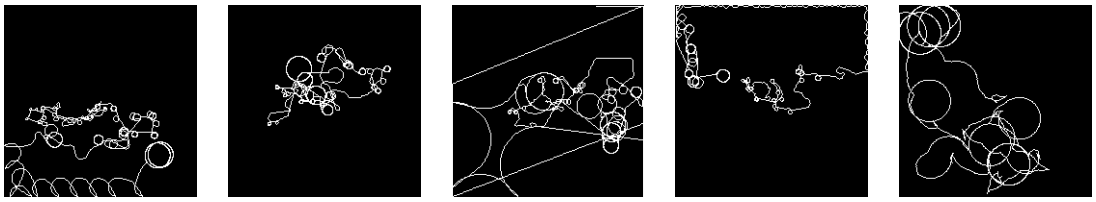
Table B.8: CNN vs. Author Classification 36-40

(a) Trace 41 (b) Trace 42 (c) Trace 43 (d) Trace 44 (e) Trace 45

Figure B.9: Unique Traces 41 - 45

	Trace 41		Trace 42		Trace 43		Trace 44		Trace 45	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	0.08	0	6.66	10	56.75	40	0.22	10	0.03	10
CP	0.08	0	0.09	0	0.08	0	0.01	0	0.02	0
LCP	0.08	0	0.07	0	0.08	0	0.01	0	0.01	0
MCP	78.92	95	0.07	0	0.08	0	0.01	0	0.01	0
SCP	20.58	5	0.07	0	0.08	0	0.01	0	0.02	0
SLR	0.27	0	93.06	90	42.94	60	99.74	90	99.91	90

Table B.9: CNN vs. Author Classification 41-45



(a) Trace 46 (b) Trace 47 (c) Trace 48 (d) Trace 49 (e) Trace 50

Figure B.10: Unique Traces 46 - 50

	Trace 46		Trace 47		Trace 48		Trace 49		Trace 50	
	CNN	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**	**CNN**	**A**
ALR	0	0	0	0	0.37	0	0.01	0	0.01	0
CP	0.01	0	0.1	0	1.73	0	0.65	20	0.01	5
LCP	0	0	0	0	0.37	0	0.01	0	0.01	0
MCP	0.02	10	0	5	65.05	70	0.01	0	98.51	95
SCP	99.97	90	99.9	95	25.82	20	99.33	80	1.46	0
SLR	0	0	0	0	6.66	10	0.01	0	0.01	0

Table B.10: CNN vs. Author Classification 46-50