Reading Report

Intelligent Agents & Games

Marshall Joseph

January - April 2022

Contents

List of Figures

1 Introduction

The purpose of this report is to provide a summary of findings after taking a deep-dive into the field of AI and Games. This field is complex, and it became apparent from the start that it would not be possible to examine all corners of the landscape. Therefore, the main focus of this work resides in modern day, advanced search techniques, applied to a multitude of game environments. Intelligent agents are an element of a game environment that can use a search technique to solve a problem. As you'll see in the forthcoming sections, these techniques can be simple like A*, or a more complex evolutionary technique could be used instead.

When using an evolutionary technique such as a genetic algorithm, there are many considerations that need to be accounted for. The individual representation of each agent needs to be intuitive and the fitness evaluation of the agent's performance must be fair. One of the motivations behind this paper is to summarize the different types of representations of individuals that exist, alongside their respective fitness evaluation functions used. If done correctly, these techniques can be applied to a wide range of problems with a high degree of success.

With classic search techniques and evolutionary algorithms, the concept of diversity is not typically measured. Instead, we focus on finding individuals who have the best fitness score in the population, and choose them to reproduce in hopes of finding better candidates. The problem with this is that premature convergence may occur, preventing any individual in the population from reaching its theoretical limits. Interestingly, individuals with the highest fitness are not always the best performers. If the selection process operates in this manner, we consider it greedy. By using a concept of behavioural novelty, we can combine individual fitness with a novelty metric to find agents who behave differently, but still maintain a competitive level of fitness.

In this paper, the concept of novelty search will be explained, with its concept applied to several intelligent agent domains. Section 2 of this report will explore 4 types of evolution and search algorithms, beginning with classic search techniques and evolutionary algorithms, before moving on to diversity search techniques and related search techniques. Section 3 of this report will look

into some agent domains which these search techniques have been applied to, from simple agent domains such as food gathering, to advanced board games. Finally, in section 4, a conclusion will be given to finalize this literature.

2 Evolution & Search

2.1 Classic Search Techniques

Classical search techniques in computer science have been around for decades and are often derived from mathematical theorems. These techniques set the foundation for many modern algorithms, with several of them still being used today. In this section, three influential algorithms will be discussed. They are A* Search, Minimax Algorithm, and Monte Carlo Tree Search.

2.1.1 A* Search

A* is a popular best-first graph traversal and pathfinding algorithm published by three researchers in 1968. Though it is over 50 years old, A* is still able to find the best solution to path finding problems in many cases. The main concept of the algorithm is that it evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal [60]:

$$f(n) = g(n) + h(n) \tag{1}$$

A* search is both complete and optimal. However, one of its drawbacks is its complexity. In most problems, the search space is large, and the number of possible solutions grows exponentially. When compared to other search algorithms that pre-process the graph, A* is simply out-performed. Due to this, there exist many variants of A* that can find sub-optimal solutions quickly. In any case, the use of a good heuristic still provides enormous savings compared to the use of an uninformed search [60].

In their work titled *"A Comparative Study of A-star Algorithms for Search and Rescue in Perfect Maze"* [42], researchers put three different A* algorithms to the test by comparing their maze searching capacity and efficiency of their different heuristic functions. The first A* method uses the heuristic function described above. The second A* method uses Euclidean distance to measure how far the father point j is from the target point which in turn

reduces the number of nodes. The last A* method uses an estimated angle value from the current point to the target point.

After running each of the three A* algorithms 10 times on every maze, the average time it took to find the target is recorded. Based on the results, it was shown that all three A* algorithms outperformed the baseline depth-first algorithm in most mazes which validated that the heuristic function is beneficial in navigating the maze. It was also demonstrated that the third A* method which uses a heuristic function with angle and distance did not perform as well as the other algorithms.

2.1.2 Minimax Algorithm

Minimax is a function used as a decision rule in many Computer Science subject areas [60]. In particular, the minimax function can be found in the minimax algorithm; an AI technique used to find optimal decisions in zero-sum games such as chess and checkers. The minimax value of a player is the smallest value that the other players can force the player to receive, without knowing the player's actions; equivalently, it is the largest value the player can be sure to get when they know the actions of the other players [46]. The definition can be written as:

$$\bar{v_i} = \underset{a_{-i}}{min}\ \underset{a_i}{max}\ v_i(a_i, a_{-i}) \tag{2}$$

The algorithm itself uses a recursive call to perform a complete depth-first exploration of the game tree. As in typical depth-first fashion, the search will proceed to the bottom of the tree, then translate each minimax value up each node and back to the top. The result is computationally expensive, and for games like chess, it is not possible for modern computers to calculate all possible moves. The reason for this is that after a certain point, a computational depth limit is reached and therefore the search is unable to traverse deeper into the tree. However, for games such as tic-tac-toe, it is able to completely solve the game due to its smaller search space.

In order to reduce the number of bottom positions which must be examined in the search, alpha-beta pruning can be used [23]. Alpha-beta pruning works in conjunction with minimax by analyzing the min and max values from the set, and pruning branches that will never be traversed through if

played optimally. In a game of chess, pruning can occur on a tree layer that is played by black or by white. For a player seeking a minimum value score, high values of the tree have a chance to be completely ignored. For a player seeking a maximum value score, low values of the tree have a chance to be completely ignored. The way we can decide if we able to prune certain branches is if it is a sub-optimal route for either min or max to take when solving the game.

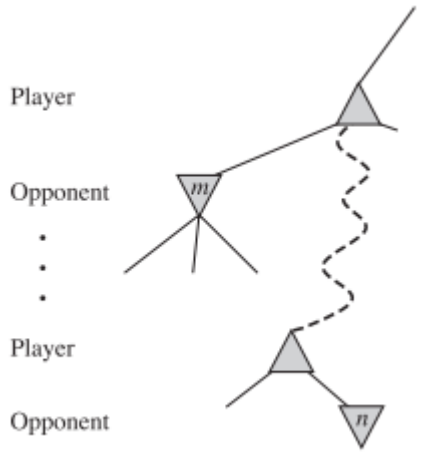


Figure 1: Alpha-Beta General Case [60]

Figure 1 shows a general case of alpha-beta pruning where if m is a better value for the player than n, then n will never get played. Authors of *"Minimax Checkers Playing GUI: A Foundation for AI Applications"* [21] show that minimax is only capable of delving to a depth of five levels in a game of checkers without using alpha-beta pruning. Even without pruning, the algorithm was able to defeat the average opponent 83% of times. In *"Move Generation and Search Strategy Research for Computer Game of Checkers"* [63], researchers implemented alpha-beta pruning and tested 200 games against freshmen students and 100 games against professors. The results were an astounding 100% win-rate for the computer, most likely due to the ability for the algorithm to more quickly decide on its moves and travel deeper in the search tree.

2.1.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a heuristic driven search algorithm which combines classic tree search implementations with machine learning principles of reinforcement learning [15]. When comparing to minimax tree search, rather than relying on a domain-dependent evaluation function, Monte Carlo simulations (roll-outs) are performed starting from each non-terminal game position to the end of the game, and statistics summarising the outcomes are used to estimate the strength of each position [44].

When it comes to tree search algorithms, the most simple version of these are uninformed searches such as breadth-first search (BFS) and depth-first search (BFS). These algorithms navigate each state in very strict order, whereas more intelligent algorithms such as A* use heuristics to help solve the problem. Meanwhile, there exists a branch of machine learning algorithms called reinforcement learning (RL) which can learn optimal strategies for the problem it is trying to solve. RL works to achieve an optimal strategy by repeatedly trying the best action it can find for a given problem. The issue with RL is the *exploration-exploitation* trade off. This occurs when the current action being repeated is not optimal, so we want to evaluate alternatives periodically during the learning phase. It can be difficult to find the correct balance between exploiting the actions and strategies that is found to be the best, and continuing to explore the local search space of alternative decisions which might replace the current best.

MCTS essentially merges both practices together. It is modelled after a tree structure of the world and searches for the best possible path by exploring specific subsections of the tree. It then calculates the effectiveness of these paths and updates the value of the states within them. After repeating these steps thousands of times, an optimal solution to a problem is likely to be found. This is very similar to how minimax operates, except minimax does not scale well at all, given that it must evaluate every opportunity quite thoroughly. MCTS proves its value by only searching a few layers deep into the tree, then prioritizing parts of the tree to explore. It will then simulate the outcome rather than exhaustively expanding the tree, and can be further limited to a certain depth.

A forward model is used in MCTS. It is an abstract approximation of the

game logic which can be summarized as playing action X in state Y resulting in outcome Z. The run will continue to play until a terminal state is reached. Then, the result is recorded and used to update a given state in the tree. Once the evaluations have been ran, the action that had the best rollout score is selected. Rollout is decided upon by four key steps: selection, expansion, simulation, and backpropagation. Through these four phases, we can take decisions to a fixed point in the tree, simulate their outcome, and propagate back the perceived value of it. This process is repeated thousands of times in order to balance which playouts to make. In order to speed up the process of calculating playouts, researchers have developed ways to parallelize MCTS [62]. It has also been shown that MCTS can be modified for general game playing [11].

2.2 Evolutionary Algorithms

Many evolutionary algorithms consist of a similar structure. Solutions are generic and population-based, heuristics are used to evaluate individuals, and the mechanisms used during reproduction are inspired by Darwin. These sets of algorithms begin with a random-like initialization phase. Once all candidates have been initialized, they begin to evolve through means of reproduction. This stage can involve genetic operators such as crossover, mutation, and a selection algorithm to choose the next candidates. In the following section, genetic algorithms and genetic programming will be briefly introduced.

2.2.1 Genetic Algorithms

In 1959, researchers experimented with machine evolution by making small mutations to a machine-code program. The idea, then, was to try random mutations with a selection process to preserve mutations that seemed useful [60]. The result was underwhelming, and it wasn't until modern genetic algorithms made their appearance that performance began to accelerate. Genetic Algorithms (GAs) are a type of evolutionary algorithm which improve upon the original idea of machine evolution [30]. One of these improvements is that solutions are represented as chromosomes in the form of a bit vector. By doing so, it is possible to apply GAs to a variety of problems such as puzzle solving, route-planning, optimization problems, and more.

GAs rely on biologically inspired operators called mutation, crossover and

selection. These operators allow for solutions to exchange genetic materials in order to diversify and improve over thousands of iterations. In order to be considered a GA, the algorithm must contain both a genetic representation of the solution, and a fitness function to evaluate the solution. As an example, we can consider a classic problem in computer science known as the travelling salesman problem (TSP). To solve this with a GA, we would need to save each location inside a chromosome vector which can be initialized in random order during the initialization phase. This would be our genetic representation. To understand how valuable this chromosome is, we can use a fitness function to evaluate it. One example of this could be to calculate the Euclidean distance to each destination in order, then sum them together. Chromosomes with a shorter distance (fitness) would be considered to have more valuable genes, and therefore have a higher chance to reproduce.

2.2.2 Genetic Programming

In 1988, nearly 30 years after researchers first experimented with machine evolution, John Koza patented his invention of a GA for program evolution [34]. 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 his work titled *"Genetic Programming: On the Programming of Computers by Means of Natural Selection"* [35], Koza goes on to describe computer programs as some of the most complex structures created by man. The purpose of this literature was to answer one of the most central questions in computer science: How can computers learn to solve problems without being explicitly programmed? [35]

To answer this question, we need to talk about 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 each node representing a data point, they represent a function. The functions could be mathematical operators such as add, sub, or mod. There are also more complex functions that could be used. The children of these functions can be recursively computed until a terminal state is reached. It is structured how you would typically solve an equation with nested statements using parentheses, and evolved with similar operations as a GA.

2.2.3 Advanced GP Techniques

GPs remain a steadily researched area in artificial intelligence with new and improved techniques appearing each year. Some of these techniques are custom made to solve unique and fascinating problems in computer science, whereas others are simply designed to improve upon the performance of the original GP algorithm. In their work titled *"A Field Guide to Genetic Programming"* [55], Poli and Mcphee describe a variety of alternative representations for programs and some advanced GP techniques. They begin by saying that the original GP system is the simplest thing that could possibly work, with most techniques dating back to late 1980's and early 1990's [55].

GP can be improved upon as follows:

- Tree initialization [36].
- ADF and other modules [55].
- Bloat control [61].
- Distributed GP [25].
- Strongly typed GP [45].
- GPU accelerating [3].
- Graph representation (Cartesian GP) [9].
- Alternate crossover [32]
- And more.

An example of such an improvement is Landon's *ramped uniform initialization* [38] which allows users to specify a range of initial tree sizes. By doing so, it was shown that certain problems obtain better results when tree sizes are more tightly controlled. On the same topic, one of the major issues with GP is bloat. Bloat occurs when trees become too large in size due to crossovers tendency to frequently rearrange the branches. Having larger trees isn't always an issue, but when trees can be simplified and provide the same or better performance, pruning is often used. Koza's *automatically defined functions* (ADFs) [36] remain the most widely used method of

evolving reusable components and have been used successfully in a variety of settings [55]. ADFs allow functions and components to be reused, and branches to be classified as function-defining or result-producing. The result of this is a more complex but less bloated tree.

2.3 Diversity Search Techniques

Diversity search techniques are different from traditional, fitness or heuristic-based search techniques. Instead of relying on a unit of measurement to gauge how close an agent is to an optimal solution, or evaluating likely scenarios and their outcomes in a tree or graph, diversity search looks for new and interesting solutions to common problems. It is effective and can outperform many algorithms in certain domains. In the following section, diversity search foundations will be discussed, starting with novelty search. Then, more advanced methods of searching for novelty will be analyzed, such as novelty and co-evolution, and two important algorithms will be explained in detail.

2.3.1 Novelty Search

Researchers have developed a way to search for behaviours while ignoring the objective altogether. This may seem counterintuitive, but in actuality it is competitive with many other advanced search techniques. In their work titled *"Abandoning Objectives: Evolution through the Search for Novelty Alone"* [39], Lehman and Stanley suggest a new search technique called Novelty Search (NS) that significantly outperforms objective-based search techniques in two problem tasks, without focusing on the objective at all. They describe the objective-based paradigm as limited, due to unexploited opportunities being missed thanks to premature convergence and deception.

Novelty isn't something that is completely new to search techniques. In fact, it has always been a challenge 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. This is known as the *exploration-exploitation trade-off*. With novelty search, the focus switches into full exploration mode. The goal of exploring more of the search space is to find solutions that may not be found by traditional search techniques that focus heavily on fitness. Since fitness influences evolu-

tion to exploit similar areas, it is likely to limit itself to a local optima. Due to how NS operates, the fitness does not influence the evolutionary process at all. Instead, we reward individuals who are unique and do not closely resemble others. This method improves upon many flaws of typical fitness-based search methods, especially in deceptive problems, by covering more ground in the landscape.

In order to search for novelty, it is important to define what novelty means. One approach could be to use novelty as a proxy to get closer and closer to the objective task. The reward would be assigned to agents that present significantly different discoveries than what has previously been found. Instead of a traditional function, a new measurement called the *novelty metric* is used as a numerical value to determine the behavioural novelty of individuals. This way, we can measure the different novelty scores across an entire population. An example of this metric could be in a Mario game playing domain. If the agent were to run into an enemy, it would be rewarded differently than if the agent were to jump off the edge of the screen, even though both yielded similar results and Mario did not reach the flag. In a traditional objective function, the Mario who made it the furthest in the level would score the highest fitness, regardless of how they reached the game over screen.

The first domain that novelty search was explored in was a deceptive maze experiment. In this type of domain, traditional fitness-based approaches would have a difficult time due to repeatedly being deceived by solutions that look like they are going to let them escape. With a novelty-based approach, new agents would continuously explore the landscape until an exit is found. They would do so without caring how close they get to the end, and therefore avoiding deception. In this domain, novelty search took on average 18,274 evaluations to reach the end, whereas the fitness-based NEAT approach took 56,334 evaluations to reach the end. NEAT with random selection was only successful in finding the exit in 21 out of 40 runs. Therefore, novelty search was three times faster than fitness-based NEAT in this problem.

The second domain that novelty search was explored in was a biped 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 challenging to solve in general. The objective is to walk as far as possible within the given time limit. To succeed, it is important to have coordination, balance, and

discover oscillatory patterns [39]. When averaged over 50 runs, the novelty-based evolved 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 does a great job at showing what is left if the pressure to achieve the objective is abandoned [39] and has shown that objective-driven search has its limits.

2.3.2 Novelty & Co-evolution

Two major kinds of novelty and co-evolutionary algorithms will be discussed in a later section. In this segment, a brief overview of both co-evolution methods will be described.

Cooperative Co-evolution

Cooperative co-evolutionary algorithms (CCEAs) are highly promising algorithms that are capable of decomposing complex problems into smaller sub-problems. These sub-problems can overlap and interact with each other, allowing individuals from separate populations to represent a complete solution to a given component. In previous works, CCEAs have shown to prematurely converge on stable but sub-optimal solutions during the evolutionary process. Co-evolutionary systems work by simultaneously evolving two or more populations, with at least one individual from each population being evaluated.

Competitive Co-evolution

In a competitive co-evolutionary system (CCES), each population competes against each other to see who can evolve the strongest. CCESs are appealing because (i) they are suited to domains where a notion of absolute fitness might not exist; (ii) each population acts as a progressively more challenging opponent to the other population, and (iii) a CCES may be less prone to stagnation than non-co-evolutionary methods, because of the ever-changing fitness landscape [64], [51].

2.3.3 MAP-Elites Algorithm

Quality-diversity (QD) optimization is a framework which searches for a large collection of solutions that are both high-performing and diverse. This is contrary to a typical optimization algorithm which will usually generate only a single high-performing solution. In their work titled *"Quality and Diversity Optimization: A Unifying Modular Framework"* [13], researchers present a framework to cover two major QD optimization algorithms: multidimensional archive of phenotypic elites (MAP-Elites) and novelty search with local competition (NSLC). They then propose new selection mechanisms for QD algorithms that show outstanding results. Finally, a new collection management system is presented in order to overcome erosion issues that occur when using unstructured collections.

QD algorithms can be used in a variety of domains. Most frequently, they are used in parallel with machine learning algorithms such as neural networks to optimize their weights and biases.

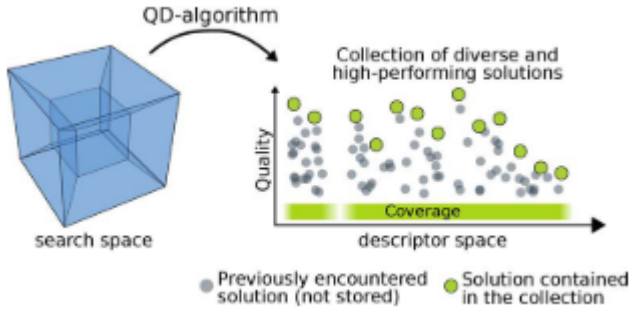


Figure 2: QD Algorithm [13]

Figure 2 gives and outline on how a QD-algorithm is used to generate solutions that are both diverse and high-performing. A high-dimensional search space seen on the left is converted into a lower-dimensional space on the right by use of solution descriptors. This collection measures the coverage of said descriptor space as well as the solutions in the collection. The grey dots represent solutions that are previously encountered and not stored, and the green dots represent solutions contained in the collection.

2.3.4 NSLC Algorithm

The second algorithm used in *"Quality and Diversity Optimization: A Unifying Modular Framework"* [13] is novelty search with local competition (NSLC) [39] which was developed by the same authors as novelty search. This algorithm focuses on solutions that are both novel (according to the novelty score) and locally high-performing [13]. When compared with vanilla novelty search, the main difference is that the performance of a solution is only compared with other local solutions. This is done through use of a local quality score which is calculated by finding the number of k-nearest neighbours with a lower performance in the novelty archive than the proposed solution. The final result of NSLC is the population of the optimization algorithm, which contains solutions that are both novel and high-performing compared to other local solutions [13].

The first research published on NSLC evolved morphology and behaviour of virtual creatures which generated entire populations of diverse species. The species could move in various directions using different limbs and maneuvers.

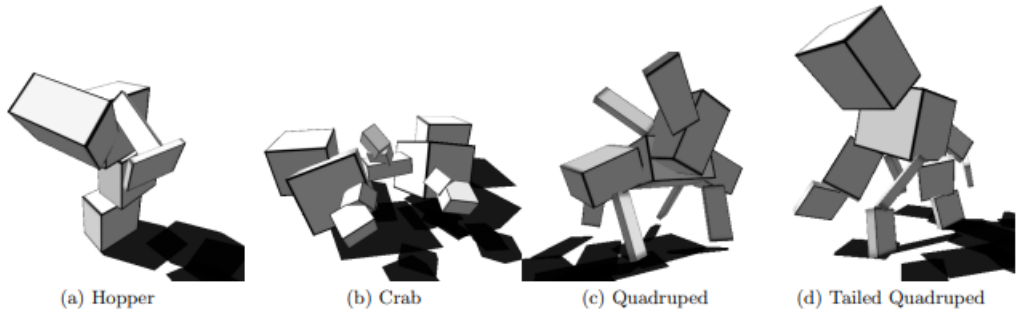


Figure 3: Diverse Creatures [39]

Figure 3 shows some interesting and diverse creatures generated during the evolutionary process. The descriptors used to generate these creatures are properties such as height, mass, number of joints. The quality of the solutions are measured based on their walking speed. If we combine both quality and diversity measurements, we could categorize the results together. For instance, height versus walking speed, mass versus walking speed, and so on. Another way of considering how to analyze the results of an NSLC run is described by Cully and Mouret [14] who suggest to consider the novelty

archive as the result, rather than the population. Since the goal is to develop novel behaviours and store them for later use, the novelty archive can be seen as a collection of solutions on their own.

2.4 Related Search Techniques

The following section outlines three interesting search techniques that can be used in a variety of game environments. These algorithms represent novel ideas or a combination of existing search algorithms that have had improvements made to them for use in specific domains. Skeleton search is a novel technique which can be used in stealth-based games. Lineage selection is a technique used to track the genes used in evolutionary algorithms in order to choose more diverse individuals to reproduce from. And portfolio search is used to address the complexity of the search space in real-time strategy games for game playing agents.

2.4.1 Lineage Selection

When using evolutionary algorithms, genetic material is passed down from parent individuals to children individuals each iteration. We don't typically care about the relationships between these individuals. Instead, the typical selection process is quite greedy, and individuals with a higher fitness score are the likely ones to be chosen for reproduction. But what if we chose individuals based on their genetic lineage instead? Genetic lineage is defined as the path from an individual to individuals which were created from its genetic material. In their work titled *"Is Increased Diversity in Genetic Programming Beneficial? An Analysis of Lineage Selection"* [10], researchers analyze the method of lineage selection to determine if increased diversity is beneficial in three different problem domains.

Previous work suggests genetic programming frequently gets stuck in a local optima due to a complete loss of diversity and the population converging towards one location [47]. Without any diversity, there is no room for the population to evolve new and exciting solutions. This enables genetic programming to behave like a set of parallel stochastic hill-climbers [54], where the population is chosen similar to how a blind random search would [24]. It's not always a bad thing though, as hill-climbers have shown to outperform genetic programming in some problem domains. Methods specific to

representation have been used to improve diversity as well. These include using Hamming distance to select diverse crossover partners in a GA [22] or edit-distance [16].

In the proposed method of lineage selection, the goal is to redirect selection pressure from the *fit*, to the *fit and diverse*. By doing so, the population should avoid premature convergence due to a higher amount of diversity.

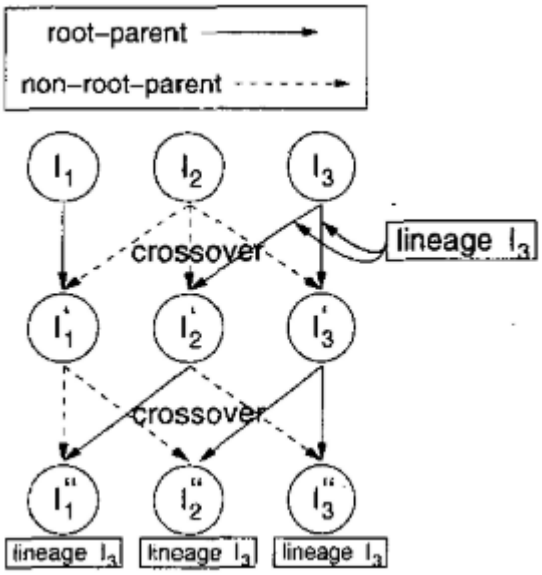


Figure 4: Lineage Recombination [10]

Figure 4 outlines what happens when three individuals begin to reproduce via crossover. The solid black line represents a root-parent and the dotted black line represents a non-root-parent. At the end of the cycle, all three children belong to the same lineage l_3. These different lineages are used during lineage selection, where individuals are selected based on common genetic lineages.

A modified version of tournament selection is used in this paper which selects individuals by choosing a random genetic lineage. From this lineage, a random individual is chosen. Once a select amount of individuals are chosen from different lineages, a tournament is held. The winner of the tournament is chosen at random to improve diversity. No elitism, measure of size, shape,

content, or fitness is considered.

$$-\sum_{k} p_k log p_k \tag{3}$$

The above equation is the measure of entropy, which measures the proportion of the population with the same fitness value. Entropy can be used to calculate the amount of chaos (unique individuals with fewer copies in the population) which in turn can tell us if new fitness values are being acquired. Lower entropy would mean that fitness classes could be used to group individuals for selection. Two measure of distances are used within the population. Both are edit-distance based and compare every member in the population to the current member with the highest fitness.

Two different experiments were run to examine the results of three different problem domains. The first experiment was a control experiment using tournament selection and the second was a real experiment which uses lineage selection. The three problems chosen for the study were Artificial Ant on the Sante Fe Trail, Even-5-Parity and symbolic regression of the Binomial-3 function [10]. Results showed that the only experiment where fitness improved was on the Ant problem. The comparison of genetic programming to hill-climbers makes it easy to understand why. When deception is embedded into the problem, improving diversity may help avoid local optima, or it may compound the deception [10]. In problems such as the Ant problem, the local optima was avoided in place of the global optima. Whereas in the Binomial-3 problem, the deception was compounded which made finding an optima difficult.

2.4.2 Portfolio Search

Real-time strategy games (RTS) are highly complex games which involve controlling large sets of units across a battlefield. In RTS games such as Starcraft, a single match can sometimes last several hours. As you may guess, the search space for such games can significantly exceed the complexity of chess and even Go. This genre proposes an exceptionally difficult challenge of developing artificial intelligence to play a game and make intelligent decisions. Authors of *"Portfolio Search and Optimization for General Strategy Game-Playing"* [17] develop a method for addressing this complexity through the use of action abstractions. The most basic abstraction type is a script

which selects an action based on a given game-state and unit.

The major contributions of this paper propose and compare different portfolio-based search algorithms for general strategy game playing. Their contributions can be summarized as:

- **Portfolio RHEA:** A new type of portfolio algorithm is proposed which combines portfolio online evolution with the rolling horizon evolutionary algorithm.
- **Optimization of portfolios and parameter sets:** Portfolio composition and parameter sets are tuned using the N-tuple bandit evolutionary algorithm.
- **Comparing performance and play-styles:** A round-robin tournament is used to compare the performance of previously reviewed and newly proposed portfolio methods.

The framework used to perform each experiment is called STRATEGA [41] which is used for studying AI development for general strategy games. It allows for the implementation of RTS games similar to Starcraft. Three pre-implemented game modes are used. They are Kings, Pushers, and Healers. Each of the games are played through a common interface for AI agent development, which provides access to the game's forward model [17]. A forward model allows agents to simulate the potential outcome of their decisions without having to make them. It does so by supplying a state-action pair in order to simulate a future state.

Three portfolio-based search methods were used in this paper. *Portfolio-based action abstractions* can be broken down into six simple scripts, which map a state and a unit to an action. These actions focus on different areas within the game, such as attacking the closest enemy or running away. *Portfolio greedy search* is an any-time greedy search algorithm that assigns a script to each player and the opponent's units [17]. It does not maximize the number of states explored. Instead, it finds a small subset of actions returned by scripts to by used. This process uses a hill-climber to optimize the script assignment to each player's units. Furthermore, the process is repeated many times in order to improve the quality of the agent's responses. *Portfolio online evolution* replaces the hill-climber used in portfolio greedy

search with evolutionary optimization. A heuristic is calculated during a game simulation which determines an individual's fitness.

The portfolio rolling horizon evolutionary algorithm (PRHEA) is an algorithm used for multi-action turn-based games [17] and can be specialized into two types: the multi-objective portfolio RHEA (MO-PRHEA) and the sparse portfolio RHEA (S-PRHEA). These are used to reevaluate the agent's unit-script assignment since RTS games typically allow users to observe what happens after a single event. The MO-PRHEA focuses on agents that require completion of multiple goals at once, such as killing the enemy units and keeping your team's units alive. The S-PRHEA can be used to plan long sequences instead. An example of this would be if attackers ran to attack, then back to defend, instead of using healers to help heal and attack. A longer, more careful planning would ensure all units are used and a higher level of efficiency is reached.

Results showed that the baseline methods PGS and POE outperformed the new proposed methods and variants. The PRHEA agent performed the best out of all newly proposed variants. A comparison of the agent's usages showed that different play-styles were preferred if they are provided the entire set of scripts to choose from, rather than narrowing their decisions into one type of play-style. The authors mentioned that this finding highlights specific optimization methods such as NTBEA which was used uncover new play-styles. This shows that the NTBEA had a large impact on the experiment as a whole.

3 Intelligent Agents & Games

3.1 Agent Domains

The following section outlines three interesting agent domains commonly used in intelligent agent research. Food gathering is a simple environment which is no longer heavily researched, but is important to mention due to its significance in the area of intelligent agents. Shepherding is a more advanced scenario which involves moving groups of agents towards a designated area and can yield interesting results. The final domain reviewed in this paper is the predator versus prey scenario which is by far the most complex. It

can involve individual predators and individual prey, or multiples of each, fighting to work together and survive or competing against one another. This domain is full of modern research which involves communication strategies and emergent behaviour.

3.1.1 Food Gathering

The food gathering domain is a simple domain with minimal criteria. In essence, the concept of food gathering (sometimes referred to as food foraging) involves an agent who searches through its environment for food, while avoiding danger along the way. The danger found in an environment could be in the form of obstacles or other agents. There also exist environments in which the food is hidden from the forager.

Evolving Food Foraging Strategies

Some of the first GP research published by Koza involved evolving strategies for real animals to gather food. In his paper titled *"Evolution of Food Foraging Strategies for the Caribbean Anolis Lizard using Genetic Programming"* [37], Koza uses his GP system find an optimal food foraging strategy for a type of lizard known as the Caribbean Anolis. Koza begins his paper by describing the behaviour of the lizard and the considerations it makes when searching for insects to consume. Some of these considerations include chasing every nearby insect if there are an abundance of insects, or when the lizard should leave its perch to chase after distant insects. The question arises as to what is the optimal tradeoff among these competing considerations [37].

The optimal strategy being evolved is a function which models the lizard and consists of four variables.

- Abundance α - the probability of the prey per square meter per second.
- Velocity v - the lizard's sprint speed.
- Coordinates x and y - the location of the insect within the lizard's view.

The traditional GP approach is used to solve this problem, with the terminal set T for this problem being:

$$T = \{X, Y, AB, VEL, \leftarrow\} \tag{4}$$

Where $\leftarrow$ represents the ephemeral random floating point constant in this case.

The conclusions drawn from this paper indicate that GP is well suited to the domain of behaviour ecology in biology, specifically in food foraging. In version 1 of the problem, the results indicate that if the program is run to at least generation 21, the yielded solution is sufficient with a 99% probability. In version 2 of the problem, the lizard does not always catch the insect at the location where it first saw it. The lizard's viewing area being divided into three regions depending on the probability that an insect will be present when the lizard arrives where it saw it. In the first region, the insect is never present when the lizard arrives. In the second region, the insect is always present. In the third region, there is a 100% probability along the X axis and a 50% probability along the Y axis. The probability varies linearly as the angle varies between 0° and 90° [37].

Food Gathering Game - "Snake"

The game Snake is a classic arcade computer game that has been around for decades. It involves a snake which is confined to a rectangular room and must eat food to grow and gain points. Furthermore, once a piece of food is eaten, a new piece of food is generated in a random location in the environment. What makes this game interesting is that as the snake grows, it becomes increasingly difficult to navigate the area. Once the snake collides with any of the four walls or itself, the game is over.

The paper *"Application of Genetic Programming to the Snake Game"* [20] describes a set of GP experiments which had a high success rate at beating the game Snake. The version used was a replica of the game which exists in Nokia cell phones from late 1990s era. This version involves a game board which consists of 220 squares in total, with 11 rows and 20 columns. The food always begins in a fixed position (11,6), and the snake is initialized at positions (1,11) to (9,11) with a size of 9 segments. The game is won once the snake consumes 211 pieces of food, leaving it with no space left to occupy.

Since there are only three moves a player can make when playing as the

snake, the terminal set T is:

$$T = \{\text{Forward}, \text{Left}, \text{Right}\} \tag{5}$$

The function set F is simple as well:

$$F = \{\text{ifFoodAhead}, \text{ifDangerAhead}, \text{ifDangerRight}, \text{ifDangerLeft}\} \tag{6}$$

The fitness used to score the snake is the raw score, i.e., how many pieces of food were eaten. Therefore, the best individual is the one who eats the most amount of food.

The researcher experimented with three types of runs. The first run was with the initial function set, the second was with the final evolved function set, and the third was with the final evolved function set and was primed. Results show that the primed runs with the final evolved function set outperformed both of the other runs by a considerable margin. In fact, the first set of runs would never improve after 100 food eaten. Therefore, using the primed runs would be the best option in order to beat the game of Snake.

3.1.2 Shepherding

The concept of herding and artificial animals has been around since at least 1993, where in their work titled *"Evolution of Herding Behaviour in Artificial Animals"* [65], researchers create a simulated world called "BioLand" to support experiments related to evolving cooperation, competition, and communication among agents. The particular experiments these researchers conducted explored evolving the herding behaviour in prey animals, then placing them in an environment with a population of their predators. The idea is that by herding together, prey animals would have a higher likelihood of being safe, finding food, and mating.

The goal of the experiment was to simulate the evolution of herding behaviour in animals that do not originally herd [65]. The 2 species of simulated animals were deer and wolves, along with two non-animal objects: plants and trees, which deer could increase their energy level by eating. On the contrary, wolves would increase their energy level by eating deer. An energy cost is associated to the actions of each animal, such as mating, eating, and producing sounds. The faster an animal moves, the more energy it uses. Due

to the multiple species of both animals and plants, this domain represents more of a hybrid predator versus prey and food gathering environment.

The perception of the simulated environment occurs differently for each species. For example, specific sensory neurons are used to smell each plant or to hear sound frequencies from other animals.

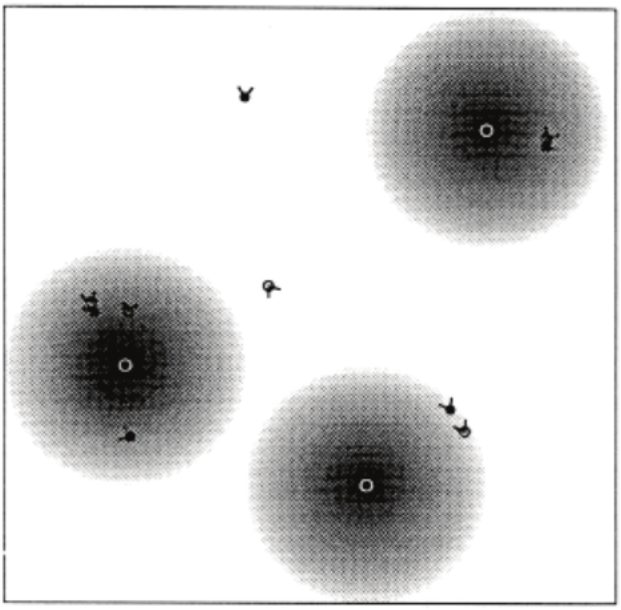


Figure 5: BioLand Objects [65]

Figure 5 shows only the gradients of the plant smell. The sensory neurons on each animal will fire proportional to the logarithmic gradient based on sensitivity. What causes these perceptions is called a biot. These biots have sensors on either side of them, and their behaviours are determined by a neural network.

The experiment conducted involved placing 8,000 prey and 8,000 predator animals into a 1000 by 1000 biot-length, toroidal environment [65]. Each animal had a randomly generated genome, and the behaviour of all animals was checked periodically throughout the run. Furthermore, a sample of the current genomes were taken every 20 generations in order to analyze how the animals were behaving. Results showed that after 30 generations, the

predators evolved in a way which was able to move closer to the prey due to their smell, and the prey evolved in a way that moved further away from the predators. After 80-100 generations, both species evolved the tendencies to move towards their own, eventually converging into small herds. These small herds continued to split apart and reform as they interacted.

In recent work, shepherding has been used for more advanced research topics. For example, *"Practical Applications of Multiagent Shepherding for Human-Machine Interaction"* [50] poses an interesting problem of trying to use multiagent systems (consisting of multiple actors) to investigate human-human or human-machine interactions.

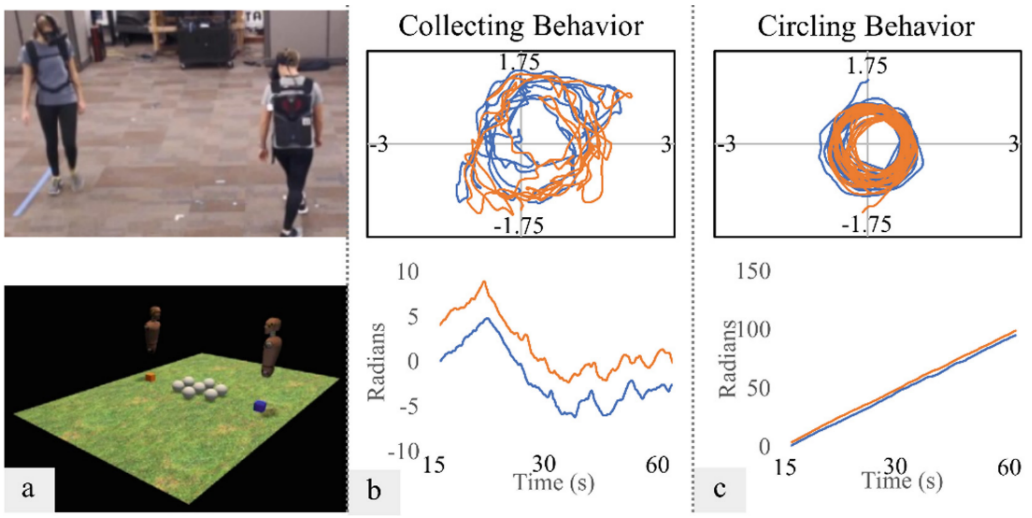


Figure 6: Locomotory Shepherding Task [50]

Shown in Figure 6, researchers equipped their participants with virtual reality headsets connected to portable backpack computers and had them complete shepherding tasks. The task shown in the bottom-left image *a* contains 7 agents (white spheres) which the participants were asked to contain. There was two major types of behaviours that the researchers noticed. The first was a collecting behaviour where the participants needed to return a sphere back to the centre, and the second was a circling behaviour which was imposed in order to keep the spheres from escaping. The gain from this study is that the researchers were able to better understand how to develop bio-inspired artificial actors through the use of human actors.

The same group of researchers of the previous paper were also authors of an earlier work titled *"Herd Those Sheep: Emergent Multiagent Coordination and Behavioural-Mode Switching"* [49]. In this work, 90 undergraduate students from the University of Cincinnati were selected as participants to learn to contain small herds of sheep within a circular region in the center of a game field.

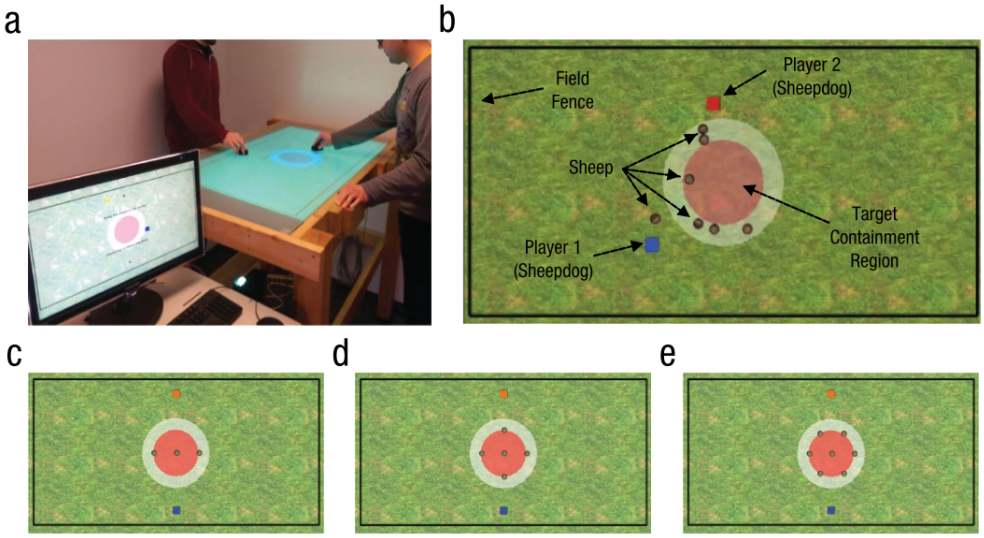


Figure 7: Shepherding Video Game [49]

Figure 7 shows what it looked like when participants were asked to play the game on a virtual tabletop display. In the images labelled *c*, *d*, and *e*, the game set up changes based on if there is 3, 5, or 7 sheep to herd. The game was developed using Unity Game Engine.

After analyzing each of the experiments, it was shown that there were two major coordination strategies adopted by the participants. The first behavioural mode was the *search-and-recover* (S&R) mode which involved recovering the furthest sheep from the participants side of the screen. The second behaviour mode was spontaneously transitioned to after a number of trials and called the *coupled-oscillatory-containment* (COC) strategy. In this strategy, a semi-circular movement was formed by each participant on either side of the circle which established a pseudo type of wall around the sheep. Interestingly, it

was noted that the S&R and COC behaviours observed were consistent with the behavioural patterns that define real-life sheepdog behaviour [7]

One last paper I'd like to briefly mention is called *"A Comprehensive Review of Shepherding as a Bio-inspired Swarm-Robotics Guidance Approach"* [43]. This is a particularly interesting paper due to its combination of shepherding and swarm intelligence which is referred to as *swarm shepherding*. It was shown in previous work that shepherding as a control technique can be applied to the control of autonomous vehicles and robots for a variety of applications. Alongside this, the addition of swarm intelligence and other AI techniques when applied to shepherding can behave in interesting ways, often improving the performance of the sheepdog.

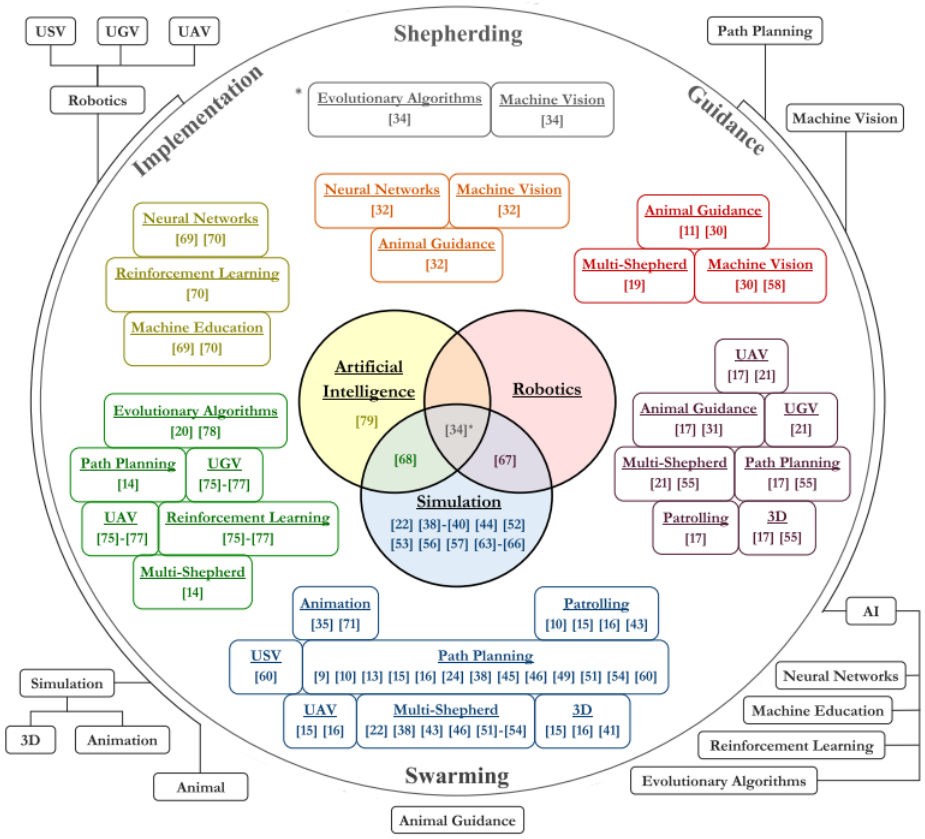


Figure 8: Shepherding Mind Map [43]

The research area of shepherding is large, as seen in the mind map.

3.1.3 Predator versus Prey

The predator versus prey domain is a heavily researched domain that consists of two groups of agents. The predators are typically a smaller group (sometimes only one), whose goal is to locate and eliminate the prey. The prey are typically a larger group, whose goal is to escape the predator and survive as long as possible. Much of the research in this field is devoted to exploring different behaviours that the predator and prey agents develop in order to survive. In his work titled *"Strategies for Evolving Diverse and Effective Behaviours in Pursuit Domains"* [12], Cowan explores different methods of evolving these behaviours through use of diversity search, while maintaining

a high level of efficacy.

Cowan's motivations come from the fact that diversified behaviours have the potential to improve a wide range of applications, like art, architecture, virtual robotics, and so on [27]. The main goals of his research are as follows:

- Combine fitness and diversity in pursuit domains.
- Produce diverse behaviours with minimal harm to fitness.
- Observe the behaviours quantitatively and qualitatively.

The first objective involves combining traditional fitness-based evaluation and diversity-based evaluation, as seen in previous work involving diversity search. Cowan employs four strategies in his paper and calculates diversity using the sum of ranks of these four behaviour measurements. Furthermore, the final fitness solution is calculated by combining this value with the original fitness value. The second objective is employed in order to ensure that solutions generated not only produce diverse behaviours, but also keep in mind that the fitness of the average individual needs to be adequate as well. The third and final objective allows Cowan to analyze data in a multitude of ways, such as by watching real-time replays of the solutions and using statistical analysis to observe behaviours in detail.

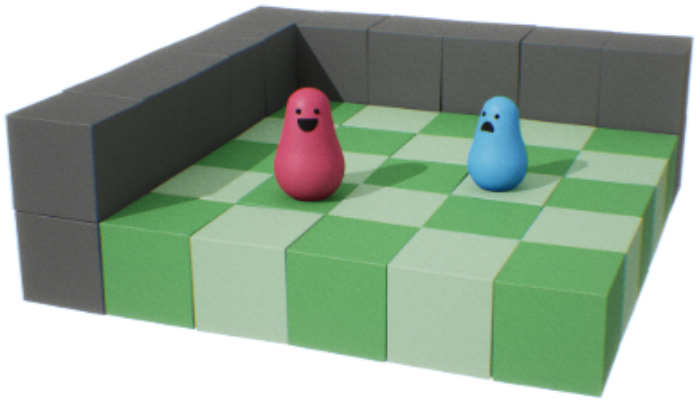


Figure 9: Portion of Predator-Prey Environment [12]

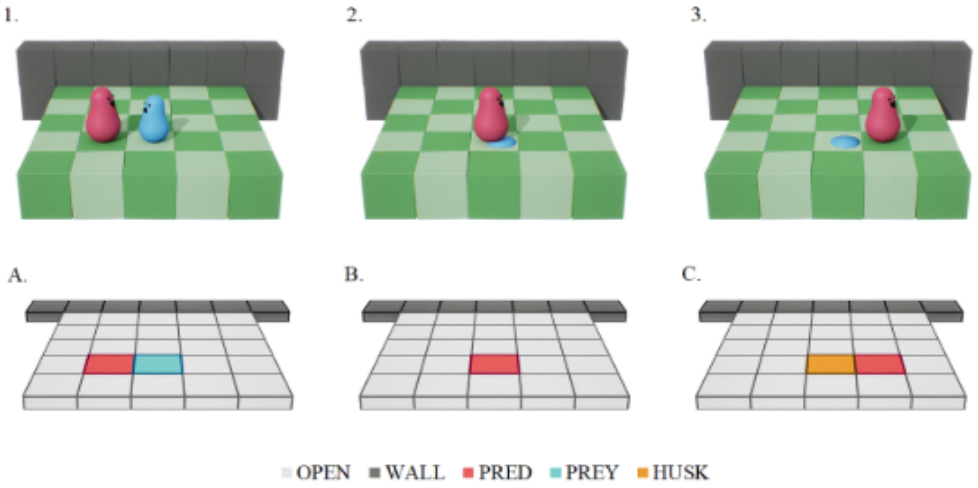


Figure 10: Continuous Environment with Discrete Overlay [12]

The environment used in the experiments was crafted from scratch and includes 5 different types of cells. Figure 9 shows a portion of the simulated environment used in [12], with a predator and prey agent staring at each other. In Figure 10, we can see the 5 different types of cells in detail. The open cells are simply cells that do not contain any objects or agents. The wall cells are an obstacle that stops agents from moving through it. The pred and prey cells are cells that contain the predator agent and prey agents in them respectively. Lastly, the husk cell are cells that include a prey agent that has been previously eaten by the predator.

The most intuitive approach for developing Cowan's diversity-based evaluation strategy was the all neighbours approach. The following four steps are used to calculate an individual's diversity score:

1. Calculate individual characterization vectors.
2. Create population average characterization vector.
3. Create individual distance vectors.
4. Rank individual distance vectors.

After the diversity score is calculated for the individual, both the diversity score and fitness-based score are multiplied by a set of weights and summed together to create the individuals overall fitness.

The conclusions drawn from this body of work show 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 research were the K-Nearest Neighbours with Archive (KNNA) using a 50/50 split of fitness score and diversity score (50F/50D) which was proved in more than one domain.

3.2 Cooperative Co-evolution

In order to combat premature convergence, authors of *"Novelty-Driven Cooperative Coevolution"* [26], propose and evaluate three novelty-based approaches which rely on (i) the novelty of the team as a whole, (ii) the novelty of the agents' individual behaviour, and (iii) a combination of the two. Convergence to stable states in CCEAs is substantially different from convergence to local optima in non-co-evolutionary algorithms [52]. Unfortunately, CCEAs are not always drawn to the global optimum solution due to deception. Deception can occur when the fitness gradient makes it seem like the population is improving when instead it has converged on a stable state. A second way deception occurs is when certain collaborators are chosen from other co-evolving populations [66].

In order to counteract deception, an evolutionary approach known as novelty search is employed which scores candidate solutions based on the novelty of their behaviour. Given the dynamic nature of the objective, novelty search has the potential to avoid deception and premature convergence [26]. The three new novelty driven co-evolution algorithms proposed are:

- **NS-Team:** Novelty search at the team behaviour level.
- **NS-Ind:** Novelty search at the individual agent behaviour level.
- **NS-Mix:** Novelty search as a combination of NS-Team and NS-Ind.

The NS-Team algorithm has its novelty score measured based on the behaviour of the team, with no regards to each individual score. The NS-Ind

algorithm has its novelty score based only on the behaviour of individual agents and their ability to collaborate with a team. Lastly, the NS-Mix algorithm has its novelty score measured as a combination of team behaviour and individual collaboration.

As the fitness landscape continues to change, so do the individuals from each population. Many strategies exist that measure the distance of individuals to other populations, and it has been shown that populations can sometimes be misled by each other. Due to this, a related problem known as *relative over-generalisation* occurs when populations in the system are attracted to regions of the search space in which there are many strategies that perform well with the individuals from the other populations [53], [67]. To measure the distance between individuals in the search space, the novelty metric relies on the average behaviour distance of that individual to the k-nearest neighbours.

In NS-Team, one team is represented by a composition of individuals from each population. A representative is chosen based on the best team fitness score in the previous generation, or a random representative is chosen in the first generation. In NS-Ind, individuals are evaluated based on how novel their behaviours are, not by how their team performs. Lastly, in NS-Mix the individuals are rewarded by a combination of their novel individual behaviours and their novel agent behaviours [26]. Specifically, the goal is to maximise team novelty, individual novelty, and team fitness score. The two sets of novelty scores are computed, which are then used to select and breed the individuals of each population.

Analysing the individuals in co-evolutionary algorithms can be beneficial due to providing insight into their behaviours. Three metrics are implemented in this paper to calculate the dispersion and exploration of best-of-generation (BoG), all team, and individual dispersion. The experiments take place in a popular pursuit scenario known as predator versus prey. In this paper, a particular version of the task is used where the predators can not communicate nor sense one another, based on [68], [56].
Figure 11 outlines the predator-prey task setup. The setup and parameters come from literature. In this experiment, the predators move at the same speed as they prey (1 unit/step). The results found that both NS-Team and NS-Mix overcame the issue of premature convergence to stable states. One

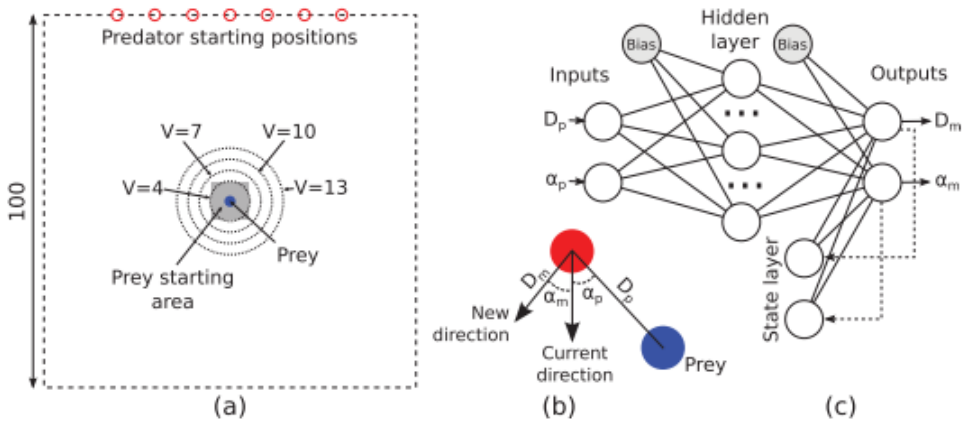


Figure 11: Cooperative Predator-Prey Task Setup [26]

interesting behaviour of NS-Team is that it still got attracted to low-quality regions of the collaboration space, however it was capable of escaping those regions and eventually reaching high-quality solutions. On the other hand, NS-Ind was a rather ineffective strategy to avoiding premature convergence. When it comes to the diversity of the solutions, NS-Team displayed much different behaviours and a wider range of collaborations than its opponents.

3.3 Competitive Co-evolution

The authors of *"Novelty Search in Competitive Co-evolution"* [28], work to investigate if novelty search, an evolutionary technique driven by behavioural novelty, can overcome convergence in co-evolution. Three new methods are proposed in their paper to apply novelty search to co-evolutionary systems with two species: (i) score both populations according to behavioural novelty; (ii) score one population according to novelty, and the other according to fitness; and (iii) score both populations with a combination of novelty and fitness [28].

One problem with co-evolutionary learning is that in most cases, a continuous arms race between opposing species is necessary in order to gain the advantages from using the algorithm. However, these arms races are not easy to establish, and they do not guarantee high quality solutions. A number of techniques to help sustain an arms race have been proposed [59],

but most of them are focused on maintaining performance against opponents from earlier generations [28]. In order to gather a wider range of solutions, the adaptation of behavioural diversity techniques are studied and applied to competitive co-evolution. This is a difficult task because populations need to be competitive at all times, and this conflicts with the major elements of diversity search where individuals are measured by their novel behaviours rather than their competitiveness. There are three dimensions used to assess individuals in this paper: (i) the quality of the best solutions achieved; (ii) the exploration of the behaviour space; and (iii) the diversity of high-quality solutions evolved [28].

Related work involving competitive co-evolution explores the property of premature convergence. This occurs when the same individuals or set of solutions are continuously selected by the population, and therefore never improve. This cycle continues until the end of the run and is not always easy to detect, as it can almost look as if there is competition occurring, when in reality there are no new solutions being explored. Many solutions to prompt a more diverse selection of individuals have been proposed. For instance, Rosin & Belew [59] propose three techniques that aim to select a diverse and challenging set of competitors to test the individuals: (i) competitive fitness sharing, (ii) shared sampling, and (iii) the hall of fame. The evolutionary search algorithm known as novelty search (NS) also has the potential to avoid premature convergence, and evolve a wide diversity of solutions in a single evolutionary run. Instead of scoring individuals using a fitness function related to the task at hand, individuals are measured on how novel their behaviour is compared to previously evaluated agents. The algorithm uses a measurement called *novelty metric* that quantifies how different an individual is from other, previously evaluated individuals [28]. The distance is given by comparing their *behaviour characterisation vectors* which are composed of the individuals traits which relate to the task they are trying to solve. When measuring distance between other individuals in the behaviour space, the average distance can be calculated by measuring the average behaviour distance to the nearest k neighbours.

The approach used in the paper involves applying novelty search to a competitive co-evolutionary system with two competing species. It can be difficult to decide on how heavily diversity should be prioritized when compared to traditional fitness. For instance, in [2], prioritizing diversity instead of

competitiveness has shown to compromise the effectiveness of co-evolution. However, in [19], it was shown that when a strategy is converged on by a population, it can be beneficial to reward these kinds of alternate strategies. Since it is not exactly clear which approach is best, several methods are evaluated in this paper [28]:

- **Fit:** Selection based exclusively on the fitness score, in both populations. This method is used as a baseline.
- **NS-Both:** Selection based exclusively on the novelty score in both populations.
- **NS-p:** Selection based on the novelty score in population p and based on the fitness score in the other population.
- **PMCNS:** Both populations are scored with PMCNS based on both novelty and fitness scores. The individuals are rewarded for displaying behaviour novelty while still meeting the minimal fitness criterion.

The experiments take place in a predator-prey pursuit environment with two co-evolved controllers: one for the predator and one for the prey. In this experiment, the predator needs to touch the prey to capture it, and the prey needs to run away.

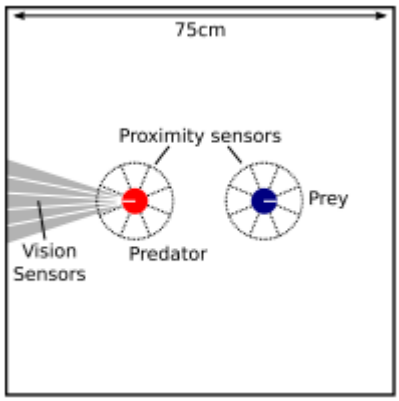


Figure 12: Competitive Predator-Prey Task Setup [28]

The experimental setup is based on [51] and an image can be seen in Figure 12, where both the predator and prey use eight evenly distributed proximity

sensors that are capable of reaching a range of 5cm. The agents are initially placed in fixed positions, facing opposing directions, and the predator has additional sensors that help detect the prey. The fitness function the prey uses is based on how many simulations it survived without being captured by the predator. The predator's fitness function is based on how many simulations it was able to successfully capture the prey. Other behavioural traits are characterised and saved into a four-element vector containing (i) the simulation length, (ii) the mean distance to the other agent throughout the simulation, (iii), the mean agent movement speed, and (iv) the mean distance between the agent and the closest wall [28].

The results show that each set of experimental parameters were able to generate high-quality predator controllers, with only minor differences. The only significant difference is between Fit and NS-Both, which was shown to have the lowest performing results in terms of quality of solutions. This was also the experiment in which novelty held the highest influence in decision making and selection. It was also shown that all high performing runs involved high fitness, and that none of the novelty-based experiments could outperform the purely fitness based Fit runs.

It was also shown that using novelty search to promote exploration of the behaviour space did not yield significant advantages regarding the achieved fitness scores [28]. NS did however have a large advantage over Fit in terms of behaviour space exploration. When coupled together, co-evolving a highly competitive, fitness-driven population with a novelty-driven population tends to generate the best solutions.

3.4 Emergent Behaviour

In their work titled *"Evolved Communication Strategies and Emergent Behaviour of Multi-Agents in Pursuit Domains"* [29], Grossi investigates how GPs can be effectively used in a multi-agent system to enable agents to learn to communicate in pursuit domains. This study continues where previous studies have left off, by searching to evolve emerged 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 nothing to determine how an agent in this situation may act. Not only do they not know the location of the prey they are trying to track down, but the predators also do not know the location of the 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 their experiments determine that emergent behaviour emerges in both experiments, enabling a high degree of communication. The most prevalent behaviour found was a message sending pattern which emerged from predators sending simple message between each other. Eventually, this simple message sending became a synchronized alternating message sending pattern which enabled the predators to better track down the prey. In addition to this, Grossi found that the behaviour closely resembled that of the guard and reinforcements found in popular stealth video games such as Metal Gear Solid [29].

3.5 Feature Extraction

The MAP-Elites algorithm started to investigate how these large collections of diverse and high-performing solutions can be generated [48]. It searches through a large portion of the search space which helps find a better overall solution than many other optimization and search algorithms. The main justification for using this algorithm is that a large group of high-performing and diverse solutions is better in most cases than a single high-performing solution. Several parameters are used in MAP-Elites which are input by the user. First, a performance measure is used to evaluate each solution. An example could be how many objects a self-driving car hits. Then, the user inputs the number of dimensions used to define the feature space. A feature space describes variations in behaviour. An example of this would be how aggressive or defensive the car AI is. If we put these parameters together, we could measure how many objects an aggressive self-driving car AI hits and how many objects a defensive self-driving car AI hits, for example.

Once the parameters have been defined, the MAP-Elites algorithm generates a number of genomes and outlines the performance and features of them. These genomes are then sorted into cells based on their feature space. The

algorithm is then initialized, and two crucial steps are repeated until a termination state is reached:

- A genome in a cell is randomly selected and produces an offspring using genetic operators such as crossover and/or mutation.
- The offspring is measured based on its features and performance. If the cell is empty, the offspring is placed there. If the cell is occupied, it is compared against the current occupant and placed there if its performance is better. In this case, the occupant would be discarded.

A termination state could be a set time limit being reached or a number of resources being consumed.

3.6 Game-playing Agents

Game-playing agents are agents that can be used to solve a particular problem or complete levels in a multitude of game domains. These domains can range from several century year old tabletop games like Chess, to modern video games such as Angry Birds. In the following section, game-playing agents capable of competing in board games, card games, and video games will be discussed. The distinction here is that game-playing agents are agents that can play a game for you, rather than play against you.

3.6.1 Board Games

In previous sections, simple board games such as Checkers were discussed by using move generation and simple search strategy algorithms as an example [63]. However, more recent work has shown that GP is capable of playing even more complex board games while producing human-competitive results. Specifically, in *"Towards Human-Competitive Game Playing for Complex Board Games with Genetic Programming"* [57], we can see that the gap between human players and GP programs is narrowing at an extraordinary rate.

The motivation behind this body of work was to determine if GP can evolve competitive AI to play against human-design AI or against MCTS in complex board games. The complex board game chosen for this experiment is called "7 Wonders". The game can be described as a fairly successful game,

selling 100,000 copies each year since its creation in 2011. Alongside its expansion packs, the game is similar to existing "civilization" games where a virtual country is developed and production of resources is managed. Unlike Checkers or Chess, 7 Wonders is a non-zero-sum game and belongs to the family of partially observable, multiplayer, stochastic games [57], making it exceptionally difficult to be solved by AI.

For games that are non-zero-sum, the fitness evaluation becomes difficult to determine. In this case, the GP needs to be opposed to other AIs. The researchers decided on using a rule-based AI and an MCTS player, rather than opposing the GP player to another GP. For the rule-based AI, a set of rules are followed which decrease in priority and are selected accordingly. This AI was not meant to perform at a high level, but it is put in place as a baseline. The MCTS player was selected due to its high performance in games when evaluation functions are hard to design. As previously mentioned, MCTS outperforms minimax with alpha-beta and is a more standard approach nowadays. The construction of the MCTS tree is incremental and consists of three different stages: *descent*, *growth*, and *evaluation*.

The conclusions drawn from their experiments show that GP can evolve very competitive players for complex board games even from basic inputs [57]. The main problem that the researchers faced was that a massive amount of computational time was needed to gather a significant fitness score. In fact, obtaining GP players trained against MCTS was completely impossible due to this restraint. However, GP trained against rule-based AI was able to beat MCTS with 1500 playouts on average and able to compete with a 3000 playout MCTS. Overall it was seen that the resulting player was several orders of magnitude faster than MCTS [57] and better performing.

3.6.2 Card Games

In their work titled *"On Novelty Driven Evolution in Poker"* [8], poker is used as an environment in order to test whether or not novelty as an objective is relevant enough to develop a wide range of NPC behaviours from a single EA run. A key difference while developing NPC strategies in poker is that much of the state information is hidden, such as the cards left in the deck or the cards in front of each player. The motivation behind this work is to investigate whether the explicit promotion of novelty (as a desirable evo-

lutionary trait) is still of fundamental relevance when attempting to evolve NPC behaviours under EC for a task with multiple sources of uncertainty [8].

Baker *et al.* used the NEAT framework for evolving neural network strategies for Poker NPCs and concentrate on the relative improvement when adopting Bayesian models to characterize opponent behaviour [4]. In their experiments, they use a simplified one-card game of Poker, and found that their opponent-model augmented NEAT networks were able to perform well against dynamic opponents. Other researchers have considered EA methods to construct strategic players for simplified forms of poker [5], [6].

The experiments in this paper were completed using a previous GP framework called Symbiotic Bid-based GP (SBB) [40] which provides a flexible architecture evolutionary classification tasks. Some properties of SBB include independent representation of team and program, separating context and action, and inter-host diversity. The fitness is calculated using a dual objective Pareto optimization (score and diversity). The score is determined by calculating the points of each team when compared to the hands of various opponents. Diversity is calculated using two mechanisms: team complement and behavioural diversity. Team complement expresses diversity in terms of a pair-wise comparison in program membership [33], whereas behavioural diversity assumes the concept of a normalized compression distance where unique traces per team are retained.

Opponents were drawn from groups using 1) a static pool, 2) trained Bayesian models, and 3) the result of self-play. The chosen rules of Poker was *Heads-up Limit Texas Hold'em*. Three types of opponents are used to play as SBB adversaries in the task: Static, Bayesian, and HoF. The classical opponent types are characterized as being loose or tight, and passive or aggressive. For example, a tight passive (TP) opponent will have a high starting hand strength but act more passive during betting rounds. On the other hand, tight aggressive (TA) opponents are seen as a dangerous combination of hand selection requirements and aggressiveness during betting rounds [8].

Testing of the experiments was performed against a balanced distribution of card hand strengths (weak, intermediate, strong) and against an unbalanced distribution of hands. The fitness function for each of the SBB teams is calculated by how many chips were won per hand. This value is normalized

based on the maximum number of chips that could be theoretically won or lost. A fitness of 0.5 means they did not win or lose chips, a fitness of 0 means they lost all chips, and a fitness of 1.0 means they raised their bet each round and won all rounds. Fitness and diversity measures are combined using a two step process [18]:

- Stochastically select one diversity measure at the beginning of each generation. This minimizes the computational overhead of supporting multiple diversity measures.
- Rank the combined fitness and diversity as two objectives using Pareto dominance. This implies that the Pareto rank of individuals in the objective space establishes their relative quality, hence fitness and diversity represent equally important objectives.

Overall, the strategies employed in this paper demonstrate that supporting both novelty and an underlying performance objective will yield a high performance when trying to maximize the number of chips won. The most used inputs to calculate the potential odds of winning were the pot odds, effective potential, and if the opponent was using a passive or aggressive play style.

3.6.3 Video Games

Video games are a heavily researched area when it comes to intelligent agents due to the amount of use cases that exist in the space. Plenty of games exist with different purposes. Online gaming can be seen as a competitive environment where cash prizes could be won, commercial games are developed in order to produce a profit, and on the other end of the spectrum, serious games are developed which can be used to improve ones cognitive ability for those with disabilities. When it comes to intelligent agents and video games, the space I am most interested in are agents that learn how to play casual platformer games independently. Second to this, I am interested in competitive online games as seen in the MOBA genre.

To give context to the casual platformer genre, let's take a trip back to classic arcade games such as Pac-Man. Even today, there are researchers still interested in Pac-Man and it truly seems to be that this game was ahead of its time when it was released. In their work titled *"Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game"* [58],

authors authors refer to Pac-Man and its equally popular successor Ms. Pac-Man as being the front-runners of the golden age of arcade video games. Moreover, the field of computational intelligence has seen many benefits from researchers who studied these games.

This paper does a great job at summarizing what made Pac-Man so unique and interesting. It goes 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 spinoff 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.

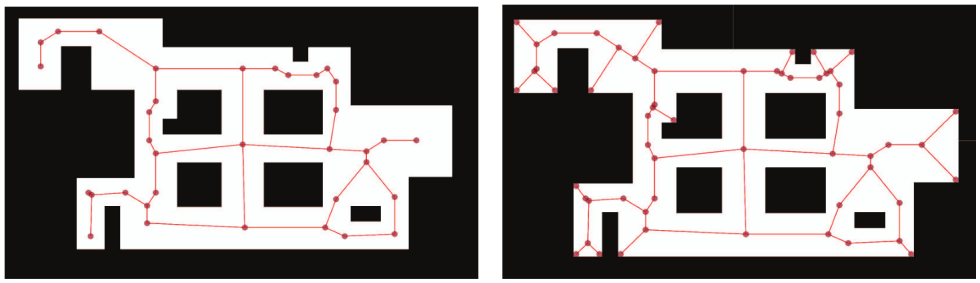
An example of some of the algorithms that researchers have used while playing Pac-Man include: rule-based algorithms, tree search and monte carlo, evolutionary algorithms, artificial neural networks, neuro-evolutionary approaches, reinforcement learning, and several other approaches such as ant colony optimization. Overall, this research 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. Lastly, they mention that the most dominating forces to play Pac-Man are both rule-based approaches and MCTS.

3.6.4 Stealth-based Games

Stealth-based games can 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 NPC enemies that observe you engage in a type of real-time pursuit, and they need to be smart. 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,

authors of *"Skeleton-based Multi-agent Opponent Search"* [1] develop a novel search method coined 'Skeleton Search' 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 [1].



(a) SAT (b) Modified SAT

Figure 13: SAT Graph Example [1]

Figure 13 outlines a) SAT and b) modified SAT used to describe the topology and segments of the docks level from the game Metal Gear Solid. Each segment has a numerical value associated with it which represents the probability an opponent is close by it. If both end-points and the middle of a segment are in a guard's field of view (FOV), then the segment is considered to be seen [1].

In the instance that a guard detects a player in a segment that is seen, all guards begin to navigate towards the player's position. Once the player leaves the FOV of the guards, the segments are assigned a probability of 1 based on which segment is closest to the direction of the opponent's velocity [1]. The remaining segments are then propagated through with lower probabilistic chance. This flow of probability is determined by two methods:

probability propagation and probability diffuse. Probability propagation determines that each segment neighbouring a probability of 1 segment is given a probability of lower value. Each neighbour of this segment is then given an even lower probability. The probabilities continue to increment so long that it has a nonzero probability, or it is adjacent to another segment with a nonzero probability. The method of probability diffusion used is similar to that used in Isla's implementation of occupancy maps [31]. The probabilities are normalized over each segment after they are diffused using a formula. In order to choose the next destination, a fitness function is used.

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 position 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. The opponent's movement speed is set to 150% faster than the guard's movement speed. Results compared the overall search performance, the search method performance, the number of guards vs their field of view, the opponent's behaviour, and the computational costs.

In terms of overall search performance, it was shown that none of the methods used could keep the opponent detected for more than 30% of an episode's time, most likely due to the opponent being 1.5 times faster than the guards. For search method performance, probability propagation performed the best is most maps due to the weights given to each segment which help guards converge effectively. The field of view radius was tested at values of 20% and 50% and there was no noticeable difference, most likely due to the structure of the maps. All 3 opponent behaviour types were able to hide effectively, though there was no significant impact between them. Finally, it was noticed that the computational costs were quite cheap and could be used in real-time for normal-sized maps. In larger maps, the performance decreases, but could be improved by increasing the segment length.

4 Conclusion

The research area of intelligent agents and games is more active than ever, and this paper has barely scratched the surface. Decades ago, classic search techniques such as minimax were developed and applied to games like chess that are centuries old. As the years progressed, video games became more prevalent and games like Pac-Man were seen as innovative due to their "advanced" AI. Fast forward to today, and there are more games that exist than any person could imagine, and area of computational intelligence has been growing year over year. Before, we would be lucky if we could apply a single algorithm to a game with the goal of being human-competitive. Nowadays, we have a plethora of algorithms to choose from to apply to thousands of games. Many of which are not only human-competitive, but magnitudes stronger than any human counterpart.

As the field of intelligent agents and games continues to grow, it is believed that further research will take us in new, unorthodox directions. Not only are agents able to play like humans, they are able to fool us into believing they are human and evolve diverse behaviours which have never been seen before. When it comes to video games, AI has gone from procedural generation of levels and enemies, to complete automatic generation of board games, card games, and more. In the future, it will be interesting to see where the advancements in technology lead us in the world of gaming. It is hoped that intelligence becomes more intelligent, and creativity becomes more creative.

References

[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] D. Ashlock, S. Willson, and N. Leahy. Coevolution and tartarus. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 2, pages 1618–1624 Vol.2, 2004.

[3] Douglas A. Augusto and Helio J. C. Barbosa. Accelerated parallel genetic programming tree evaluation with OpenCL. *Journal of Paral-*

lel and Distributed Computing, 73(1):86–100, 2013. Metaheuristics on GPUs.

[4] R.J.S. Baker, P.I. Cowling, T.W.G. Randall, and P. Jiang. Can opponent models aid poker player evolution? In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 23–30, 2008.

[5] L. Barone and L. While. Evolving adaptive play for simplified poker. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 108–113, 1998.

[6] L. Barone and L. While. An adaptive learning model for simplified poker using evolutionary algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 153–160 Vol. 1, 1999.

[7] Brandon Bennett and Matthew Trafankowski. A comparative investigation of herding algorithms. In *Proc. Symp. on Understanding and Modelling Collective Phenomena (UMoCoP)*, pages 33–38, 2012.

[8] Jessica P. C. Bonson, Andrew R. McIntyre, and Malcolm I. Heywood. On novelty driven evolution in poker. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.

[9] P. Burian. Fast detection of active genes in cartesian genetic programming. In *International Conference on Signals and Electronic Systems (ICSES 2014)*, September 2014.

[10] E.K. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Is increased diversity in genetic programming beneficial? an analysis of lineage selection. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 2, pages 1398–1405 Vol.2, 2003.

[11] Danil A. Chentsov and Sergey A. Belyaev. Monte carlo tree search modification for computer games. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 252–255, 2020.

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

[13] Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2018.

[14] Antoine Cully and Jean-Baptiste Mouret. Behavioral Repertoire Learning in Robotics. In *Genetic and Evolutionary Computation Conference*, pages 1–8, Amsterdam, Netherlands, July 2013.

[15] 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).

[16] Edwin D. de Jong, Richard A. Watson, and Jordan B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, page 11–18, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[17] Alexander Dockhorn, Jorge Hurtado-Grueso, Dominik Jeurissen, Linjie Xu, and Diego Perez-Liebana. Portfolio search and optimization for general strategy game-playing. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 2085–2092, 2021.

[18] Stephane Doncieux and Jean-Baptiste Mouret. Behavioral diversity with multiple behavioral distances. In *2013 IEEE Congress on Evolutionary Computation*, pages 1427–1434, 2013.

[19] Adam Dziuk and Risto Miikkulainen. Creating intelligent agents through shaping of coevolution. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 1077–1083, 2011.

[20] Tobin Ehlis. Application of genetic programming to the "snake game". *Gamedev.Net*, 1(175), 2000.

[21] 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.

[22] Larry J. Eshelman and J. David Schaffer. Crossover's niche. In *Proceedings of the 5th International Conference on Genetic Algorithms*, page 9–14, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[23] S.H. Fuller, J.G. Gaschnig, J.J. Gillogly, CARNEGIE-MELLON UNIV PITTSBURGH PA Dept. of COMPUTER SCIENCE., and Carnegie Mellon University. Computer Science Department. *Analysis of the Alpha-beta Pruning Algorithm*. Carnegie-Mellon University. Department of Computer Science, 1973.

[24] Chris Gathercole and Peter Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In *Proceedings of the 1st Annual Conference on Genetic Programming*, page 291–296, Cambridge, MA, USA, 1996. MIT Press.

[25] Wolfgang Golubski. Distributed genetic programming for regression analysis. In *WSEAS IMCCAS-ISA-SOSM and MEM-MCP*, Cancun, Mexico, May 12-16 2002.

[26] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. Novelty-driven cooperative coevolution. *Evolutionary Computation*, 25(2):275–307, 2017.

[27] 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, Berlin, Heidelberg, 2012. Springer-Verlag.

[28] Jorge C. Gomes, Pedro Mariano, and Anders Lyhne Christensen. Novelty search in competitive coevolution. *CoRR*, abs/1407.0576, 2014.

[29] 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.

[30] J.H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

[31] Damián Isla. Third eye crime: Building a stealth game around occupancy maps. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 9(1):206, Jun. 2021.

[32] Takuya Ito, Hitoshi Iba, and Satoshi Sato. Depth-dependent crossover for genetic programming. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 775–780, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.

[33] Stephen Kelly and Malcolm Heywood. Genotypic versus behavioural diversity for teams of programs under the 4-v-3 keepaway soccer task. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), Jun. 2014.

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

[35] John R. Koza. Genetic programming - on the programming of computers by means of natural selection. In *Complex adaptive systems*, 1993.

[36] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[37] 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.

[38] W. Langdon. Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 1:95–119, 01 2000.

[39] 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, page 211–218, New York, NY, USA, 2011. Association for Computing Machinery.

[40] Peter Lichodzijewski and Malcolm I. Heywood. Managing team-based problem solving with symbiotic bid-based genetic programming. In *GECCO '08*, 2008.

[41] Diego Perez Liebana, Alexander Dockhorn, Jorge Hurtado Grueso, and Dominik Jeurissen. The design of "stratega": A general strategy games framework. *CoRR*, abs/2009.05643, 2020.

[42] 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.

[43] Nathan K. Long, Karl Sammut, Daniel Sgarioto, Matthew Garratt, and Hussein A. Abbass. A comprehensive review of shepherding as a bio-inspired swarm-robotics guidance approach. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(4):523–537, 2020.

[44] 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.

[45] Viktor Manahov. The rise of the machines in commodities markets: new evidence obtained using strongly typed genetic programming. *Annals of Operations Research*, 260(1-2):321–352, 2018.

[46] M. Maschler, M. Borns, S. Zamir, Z. Hellman, and E. Solan. *Game Theory*. Cambridge University Press, 2013.

[47] Nicholas Freitag McPhee, Nicholas J Hopper, et al. Analysis of genetic diversity through population history. In *Proceedings of the genetic and evolutionary computation conference*, volume 2, pages 1112–1120. Citeseer, 1999.

[48] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites, 2015.

[49] Patrick Nalepka, Rachel Kallen, Anthony Chemero, Elliot Saltzman, and Michael Richardson. Herd those sheep: Emergent multiagent coordination and behavioral-mode switching. *Psychological science*, 28:956797617692107, 04 2017.

[50] Patrick Nalepka, Rachel Kallen, Anthony Chemero, Elliot Saltzman, and Michael Richardson. *Practical Applications of Multiagent Shepherding for Human-Machine Interaction*, pages 168–179. Springer, 06 2019.

[51] Stefano Nolfi and Dario Floreano. Coevolving predator and prey robots: Do "arms races" arise in artificial evolution? *Artificial Life*, 4(4):311–335, 1998.

[52] Liviu Panait, Sean Luke, and R. Paul Wiegand. Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, 10(6):629–645, 2006.

[53] Liviu Panait, R Paul Wiegand, and Sean Luke. A visual demonstration of convergence properties of cooperative coevolution. In *Parallel Problem Solving from Nature–PPSN VIII*, page 892. Springer, 2004.

[54] Riccardo Poli and W.B. Langdon. On the search properties of different crossover operators in genetic programming. In *University of Wisconsin*, pages 293–301. Morgan Kaufmann, 1998.

[55] Riccardo Poli, William Langdon, and Nicholas Mcphee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 01 2008.

[56] Aditya Rawal, Padmini Rajagopalan, and Risto Miikkulainen. Constructing competitive and cooperative agent behavior using coevolution. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 107–114, 2010.

[57] Denis Robilliard and Cyril Fonlupt. Towards human-competitive game playing forźcomplex board games with genetic programming. In *Revised Selected Papers of the 12th International Conference on Artificial Evolution - Volume 9554*, page 123–135, Berlin, Heidelberg, 2015. Springer-Verlag.

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

[59] Christopher D. Rosin and Richard K. Belew. New methods for competitive coevolution. *Evol. Comput.*, 5(1):1–29, Mar 1997.

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

[61] Sara Silva, Stephen Dignum, and Leonardo Vanneschi. Operator equalisation for bloat free genetic programming and a survey of bloat control methods. *Genetic Programming and Evolvable Machines*, 13(2):197–238, June 2012.

[62] Erik Steinmetz and Maria Gini. More trees or larger trees: Parallelizing monte carlo tree search. *IEEE Transactions on Games*, 13(3):315–320, 2021.

[63] Qi Wang, Chang Liu, Yue Wang, and Danyang Chen. Move generation and search strategy research for computer game of checkers. In *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pages 3727–3731, 2015.

[64] Richard A. Watson and Jordan B. Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, page 702–709, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[65] 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*. The MIT Press, 04 1993.

[66] R. Paul Wiegand, William C. Liles, and Kenneth A. De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, page 1235–1242, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[67] R. Paul Wiegand and Mitchell A. Potter. Robustness in cooperative coevolution. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, page 369–376, New York, NY, USA, 2006. Association for Computing Machinery.

[68] Chern Han Yong and Risto Miikkulainen. Coevolution of role-based cooperation in multiagent systems. *IEEE Transactions on Autonomous Mental Development*, 1(3):170–186, 2009.