School of Electronic Engineering
and Computer Science

Final Report

Programme of study:
BSc FT Computer Science 4 year with IE

Project Title:

APPLYING NEAT IN A ROLLING HORIZON WAY

Supervisor:
Dr Diego Perez Liebana

Student Name:
Muhammad Sajid Alam

Final Year
Undergraduate Project 2019/20

Date: 11/05/2020

Queen Mary
University of London

Acknowledgements

I would like to express my deep gratitude to Dr Diego Perez-Liebana for his patient guidance, enthusiastic encouragement and useful critiques of this research work. I would also like to thank Mr. Finn Eggers, for his advice and permission on building upon his NEAT framework, and Miss Raluca Gaina for her invaluable support on the submission of a condensed version of this paper for the 2020 IEEE Conference on Games. I would also like to extend a warm thanks to my family and friends for helping me through these tough times, as the pandemic spread, they kept me level headed and focussed on what was in my control.

This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT. http://doi.org/10.5281/zenodo.438045

Abstract

An area of interest is the development of AI agents that are able to improve and adapt as a video game is played. This paper presents a new methodology for Evolutionary Algorithms (EA), specifically Rolling Horizon NeuroEvolution of Augmenting Topologies (rhNEAT). This algorithm works by evolving weights, connections and nodes in real-time instead of sequence of actions like in traditional Statistical Forward Planning (SFP) methods. This in conjunction with using Forward Models for planning several moves ahead, before returning an action, forms the basis of this new approach. We explore several variations and parameter changes of the algorithm across 20 GVGAI (General Video Game AI) games and compare it against state-of-the-art methods. We find that changing rhNEAT specific parameters such as mutations, coefficients from the distance function and speciation thresholds have marginal effects on performance of the algorithm, where at most results tend to perform slightly worse than the baseline parameters. Furthermore, we also found that rhNEAT in general is not better than other SFP methods it has, in some games, been able to set new records that other popular methods struggle with. This algorithm is general and opens up new ways of conducting rolling horizon techniques, providing an inspiration for future work based on these algorithms.

Contents

Introduction

From as early as 1951 video games, such as *Nim,* and artificial intelligence have worked together to create entertainment that is both interesting and enjoyable. Many modern video games still rely on “scripting” for their behaviour, this can result in the AI being easily exploited ruining immersion and enjoyability of the game (Lara-Cabrera et al 2015). Thus, research into General Video Game Playing (GVGP) has become very popular in recent years, some of which can be attributed to the availability of frameworks such as General Video Game AI (GVGAI) (Perez et al 2015). An important area of research is creating AI agents that are general i.e. they are able to play multiple types of games. GVGAI provides more than 180 single and two-player games where each game can have multiple levels. This framework has been used in many previous studies to evaluate Statistical Forward Planning (SFP) methods such as Rolling Horizon Evolutionary Algorithms (RHEA) (Perez et al, 2013).

The conceptualised rhNEAT algorithm aims to introduce a new SFP method by taking concepts from both NeuroEvolution of Augmenting Topologies (NEAT) and RHEA. NEAT is not an SFP method as it was designed to train AI agents offline i.e. individuals are compared one or two at a time until the entire population has been tested and a new population is created to form the next generation.

rhNEAT evolves neural network weights, connections and nodes. The network is then only used to generate sequences of actions from which an individual’s fitness score is obtained.

This paper will explore the Background concepts and Literature Review in Chapters 1 and 2. We will also discuss the implementation and experimentation in Chapters 3 and 4. Finally, we will conclude the paper in Chapter 5 and discuss further work in Chapter 6.

Aims:

The main aims of this project include; creating an AI Agent that successfully implements the rhNEAT algorithm and evaluate its performance compared to other SFP methods such as Monte Carlo Tree Search and Rolling Horizon Evolutionary Algorithm. Furthermore, our goals include developing and testing several variants of rhNEAT where its specific parameters are changed and compared against a baseline.

Research Question:

Can rhNEAT be a new SFP method with performance similar to or greater than other popular SFP methods such as RHEA?

Accepted for Publication:

As part of the 2020 IEEE Conference on Games a condensed version of this report was submitted for peer review. The paper was accepted and it will be published in the conference and on IEEE Xplore.

Chapter 1: **Background**

1.1 Statistical Forward Planning Methods

Statistical Forward Planning (SFP) methods are algorithms that use a Forward Model to simulate future states from corresponding state-action pairs (Perez et al, 2019). Examples of SFP methods include Monte Carlo Tree Search (MCTS) and Rolling Horizon Evolutionary Algorithms (RHEA).

1.1.1 Monte Carlo Tree Search

MCTS is a popular game playing algorithm which employs best-first search techniques (Chaslot et al, 2008). The basic process for MCTS follows a tree that is incrementally built upon in an asymmetric manner. The algorithm must balance the exploration of different nodes while also exploiting those that have the most potential. The tree is used to simulate possible actions an agent can take by pseudo-randomly selecting moves. Thus, simulating many random games enables winning strategies to be found. The algorithm builds the tree by repeating four steps until some decision budget is reached. These steps include:

- **Selection**: While the state is in the tree the next action is decided by balancing exploration and exploitation.
- **Expansion**: When a state cannot be found in the tree a new node is added, in this way the tree is expanded for each simulation.
- **Simulation**: During simulation actions are picked at random and simulated until a certain depth or game over is reached.
- **Backpropagation**: Once reaching the end of a simulated game each tree node that was traversed is updated with a value assigned to the final state of the simulation.

The action that is actually played in the game is the node that was most frequently explored.

1.1.2 Rolling Horizon Evolutionary Algorithms

In RHEA individuals are represented by a sequence of actions. A forward model is used to simulate several moves ahead for each individual and evaluate how well they perform until some decision budget like a time limit is reached. The first action of the individual with the highest evaluation is played in the actual game tick. RHEA has been extensively documented in the field for General Video Game playing and has been found to, in some places, outperform MCTS (Perez et al, 2020). Section 1.4 will go into further detail about the algorithm.

1.2 NeuroEvolution

NeuroEvolution is a deeply studied field of computer science. It falls under the umbrella of evolutionary computation in respect of its work on evolving artificial neural networks (ANN). It was initially coined almost three decades ago where work was mainly focused on, “describing techniques for learning connection weights,” (Baldominos et al, 2019). This resulted in most of the early work of NeuroEvolution being bound to simple ANN. Baldominos notes that it wasn’t until the emergence of the backpropagation algorithm and its limitation of not knowing the most optimal way of connecting neurons (topologies) within a network that pushed for the popularity of evolutionary algorithms (EAs).

Gnana and Deepa (2013) note, that optimal topologies for backpropagation were found through trial-and-error and following guidelines, which they say in truth were not sufficient in finding optimal solutions. It wasn’t until the late 1980s when researchers finally began using evolutionary algorithms as opposed to backpropagation “for evolving weights and determining the optimal topologies,” (Baldominos et al, 2019).

Smith (1974) explains that evolution in biology can be seen as ‘*an optimisation process’*. With this thought, Baldominos states; it can be seen that NeuroEvolution is the cross between optimisation problems and the “Darwinian theory of natural selection and evolution.” So, to summarise NeuroEvolution or evolutionary computation can be seen as solving optimisation problems using real life biological mechanisms as analogues. As such EAs can be seen implementing operators such as cross-over, selection, speciation and mutations. Effectively taking the principles of evolution from the real world and applying them to optimisation algorithms.

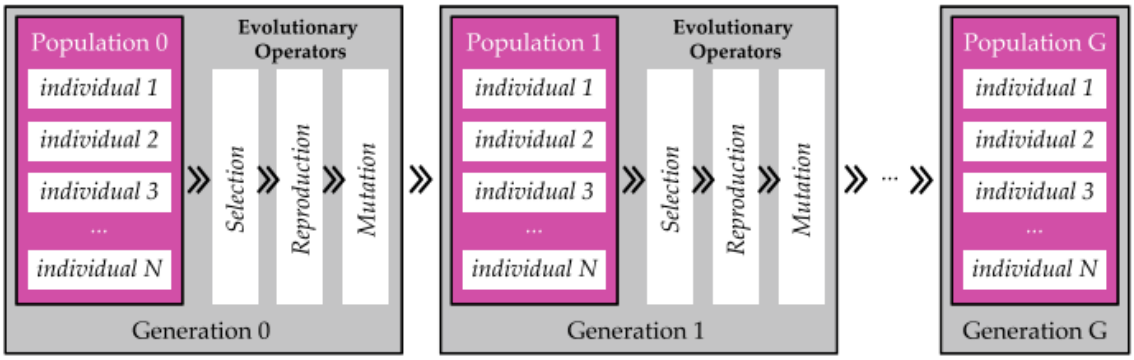

Figure 1 General framework for evolutionary algorithms. (Baldominos et al 2019, p.5)

Figure 1 shows the general framework for most evolutionary algorithms where it involves a defined population of N which undergoes several evolutionary operators such as selection, reproduction and mutation and then iteratively repeat the step for the newly generated population in hopes that each subsequent population is better than the last in terms of an average fitness score.

1.3 NEAT

NEAT was introduced by Stanley and Miikkulainen in 2002 in their paper, *Evolving Neural Networks through Augmenting Topologies,* (Stanley and Miikkulainen, 2002). Baldominos has stated that it had, “become one of the most cited and used systems in NeuroEvolution”. The NEAT algorithm evolves both weights and topologies of a neural network. As explained in 2.1 genetic algorithms implement the crossover operator to generate offspring from two parents. This means that EAs must take this functionality into account for encoding and make it as seamless as possible. Stanley and Miikkulainen have stated, “NEAT’s genetic encoding scheme is designed to allow corresponding genes to be easily lined up when two genomes cross over during mating.” They have achieved this by introducing the concept of *innovation numbers.* These numbers act as ‘historical markings’ as to determine which genes match up with which during crossover, we will go over this in section 1.3.2.

1.3.1 Genetic Encoding

The NEAT method involves three fundamental solutions to evolving artificial neural networks. These include the usage of historical markings to allow for more meaningful crossover, the idea of speciation so that generations are protected from being removed too early and finally, the idea of starting with the simplest network first and incrementally making it more complex to find the most efficient solution (Stanley et al, 2005).

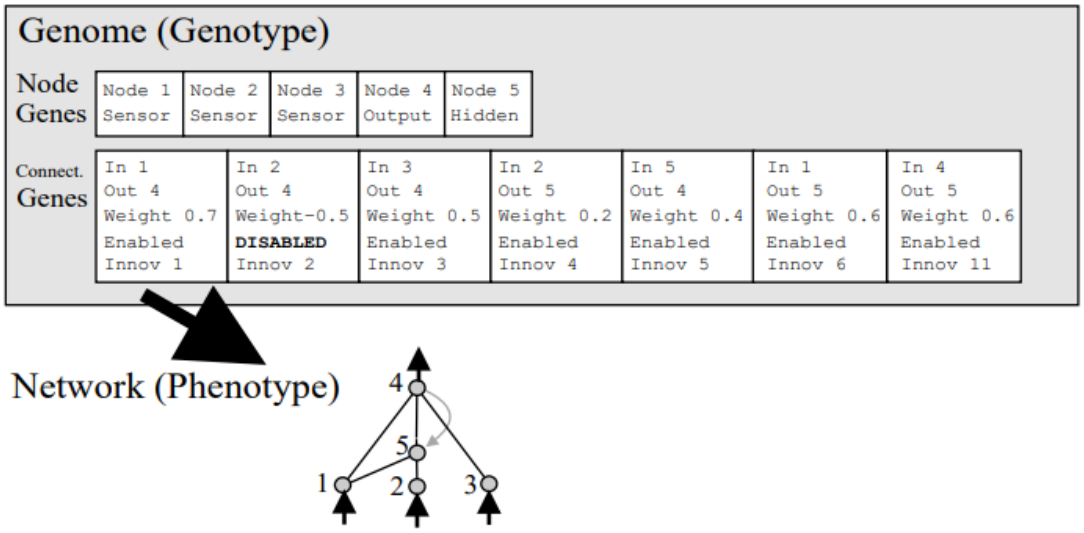

Figure 2 Genotype to Phenotype mapping example a high-level overview of the genetic encoding of NEAT (Stanley and Miikkulainen, 2002, p.106)

Figure 2 shows an abstract depiction of the NEAT genome encoding. Each genome in NEAT would have a corresponding list of *node genes* and *connection genes*. The *node genes* include a list of input, hidden and output nodes, that can be connected (Baldominos et al, 2019). Stanley also mentions that each connection gene contains an *in-node*, *out-node*, *weight of connection*, a Boolean

variable describing if its enabled or not, and an *innovation number.* Mutations in NEAT can affect weights and topology of a network.

Weights evolve following the established conventions in NE. However, topological mutations can involve adding new connections or nodes, forming the basis for complexification of networks in NEAT. Stanley describes the *add connection* mutation, as a singular new connection gene being added to two previously unconnected nodes. The *add node* mutation adds a node to an already existing connection effectively splitting them and disabling the original connection. As a result, two new connections are made and the connection between the first and new node is given a weight of one while the connection from the new node to the second node is given the original weight. This is illustrated in figure 3 below.

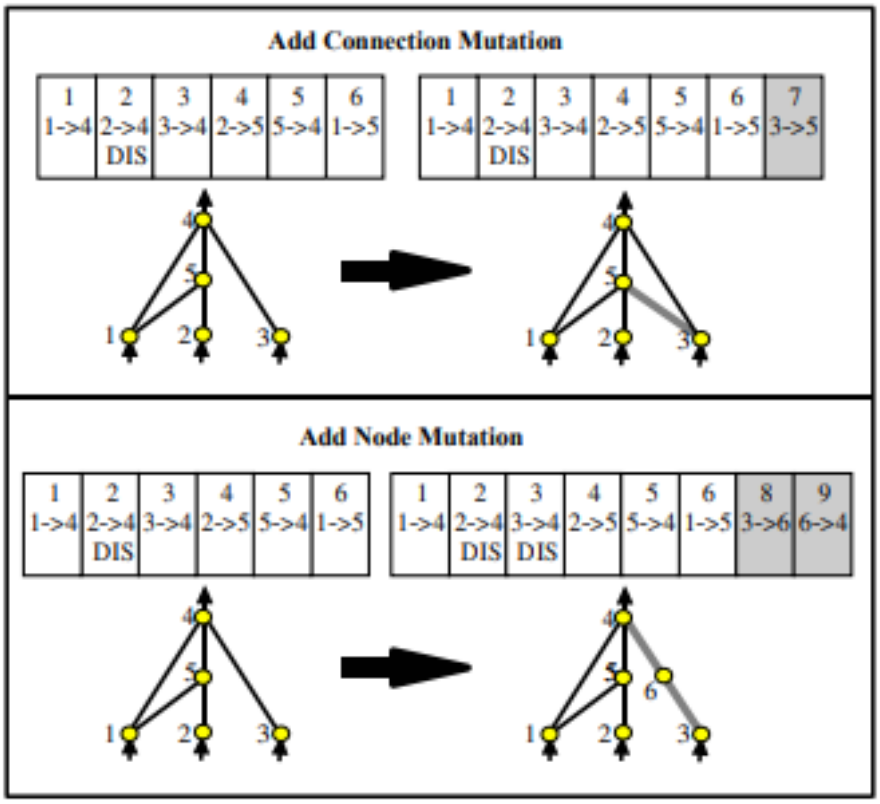

Figure 3 example illustration of the two genetic mutations in NEAT. (Stanley et al, 2005, p.9). The numbers at the top of the boxes in figure 3 represent the innovation numbers, the arrows show a connection between two nodes and the last line represents if that connection is enabled or not. In the add connection mutation step we can see a random connection has been made between nodes 3 and 5. This is represented in the grey box which shows the new connection with an incremented innovation number. The add node mutation steps illustrate what happens when a random node is added between an existing connection. In this case a random node, 6, is added between the connection of nodes 3 and 4. This has the effect of disabling the connection 3->4 and creating two new connections, 3->6 and 6->4.

1.3.2 Innovation Number

Baldominos describes innovation numbers as incremental values that is added to each connection that is created. This acts as indicator to when the gene appeared in the evolution process. Innovation numbers play a crucial role in the crossover operator as they need to be able to recombine genes with different

topologies, something which has been stated as being difficult (Radcliffe, 1993). Two genes are lined up based on their innovation numbers, gaps may appear when innovation numbers are present in only one parent. The offspring is formed either through uniform crossover, by randomly choosing matching genes, or through blended crossover, where the weights of two matching genes are averaged (Wright, 1991). Figure 4 shows an illustration from Stanley's paper of uniform crossover.

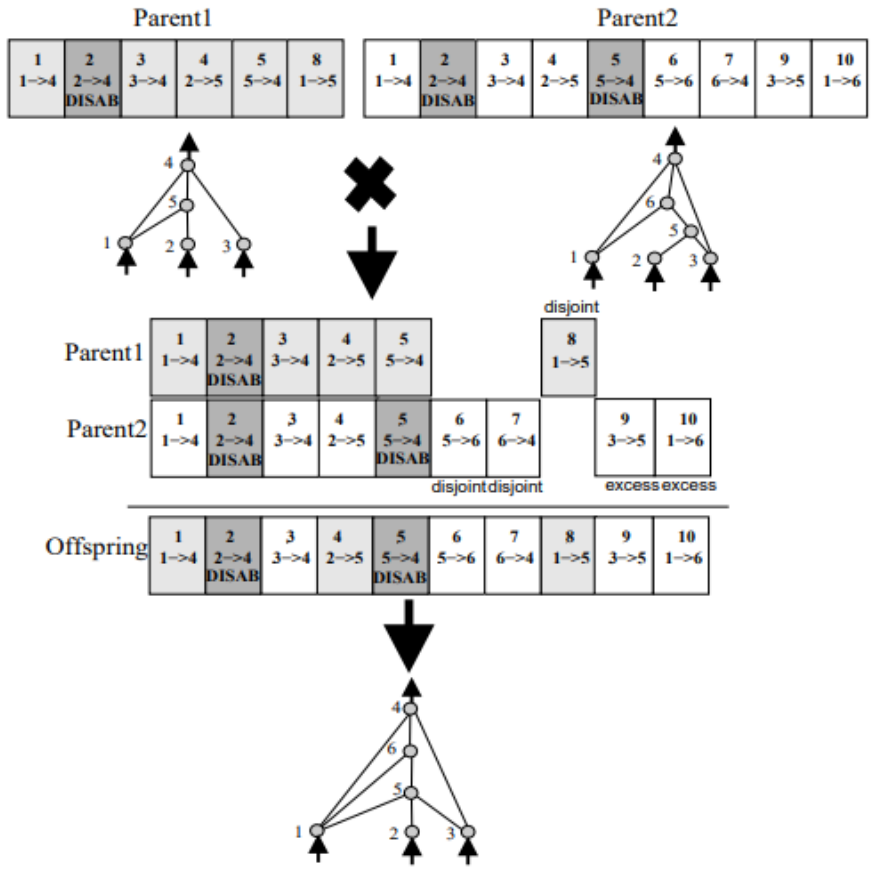

Figure 4 Crossover of two parents forming an offspring (Stanley and Miikkulainen, 2002, p.109).

In Figure 4 Parent 1 and 2 are lined up according to their innovation numbers. Where there are genes that do not match, they are labelled as *disjoint* or *excess* genes dependent on whether or not the gene is within the range of the opposite parent's innovation number. *Disjoint* and *excess* genes are taken from the more fit parent or from both if they are equally fit. In Figure 4 we assume that both parent's fitness is equal so we can see that the offspring has inherited genes from both parents. This operator also provides the opportunity for disabled genes to be enabled again. To summarise the use of innovation numbers allows NEAT to perform crossover without doing expensive analysis on topologies as genomes regardless of size or species stay compatible throughout the evolution process.

1.3.3 Speciation

Stanley has noted that, since smaller networks optimise faster and adding new mutations to genes can initially result in lower fitness scores, it can result in newer topologies having a small chance of surviving more than one generation. This can lead to losing innovations that could prove to be important in the future. Therefore, Stanley and Miikkulainen have incorporated the idea of speciation into NEAT. This relies on the principle of populations within specific species competing against each other over competing against the entire population as a whole. This provides time for different *topological innovations* to optimise before they compete against other species (Stanley et al, 2005).

1.4 RHEA

Rolling Horizon Evolutionary Algorithms (RHEA) is a subset of EAs discussed in section 1.2. Unlike conventional EAs that train a controller with an off-line simulator, RHEA works by evolving an individual plan in an *imaginary model* for some time in milliseconds to reach an imaginary state several times. This is repeated for all individuals in a population so that the first action of the best individual is chosen. This behaviour, of looking ahead, is what gave rise to the nomenclature “Rolling Horizon” in RHEA (Perez, 2013).

An in-depth analysis of the vanilla version of RHEA was done on the General Video Game Artificial Intelligence (GVGAI) framework on a subset of 20 games by Perez et al. They modified several parameters such as population size and individual length and measured the performance of each configuration. Performances were then compared against other algorithms such as the Open Loop Monte Carlo Tree Search (OLMCTS) and Random Search (RS). Some of their findings concluded that RHEA outperformed OLMCTS when number of individuals per population was greater than five thus making it an alternative to OLMCTS in GVGAI competition (discussed in section 2.1). They also found that the RS algorithm was able to find better solutions than RHEA which they believe is due to the vanilla RHEA not being able to, “explore the search space quickly enough given the limited budget,” (Perez et al, 2013). This leaves room to improve RHEA such as to combine RHEA with other techniques to form hybrid GAs.

1.4.1 Algorithm

As the algorithm has been used extensively on GVGAI I will be discussing it with the assumption it is playing one of the single player games of the framework. The algorithm begins at the start of each game tick by initialising a new set of action plans, represented by populations of individuals (Perez et al, 2013). Various methods of population seeding have been analysed by Gaina et al, but this paper will only be looking at vanilla RHEA where the population is randomly initialised. Traditional EA operators, such as mutation and crossover discussed earlier, are

used to obtain new individuals for the next generation. These new individuals are evaluated using a Forward Model, a system to simulate several moves ahead, and are then assigned a fitness value, via a heuristic function, and sorted accordingly to it (Gaina et al, 2017). Figure 5 shows an overview of the RHEA cycle. The best individuals according to their fitness scores are carried through subsequent generations. This process is repeated until a set number of iterations or an end condition is met, such as the allotted time or memory limit being reached. At this stage the algorithm chooses the first action from the best individual at the end of the process. Once the action is played, a new state is obtained and the entire process is repeated.

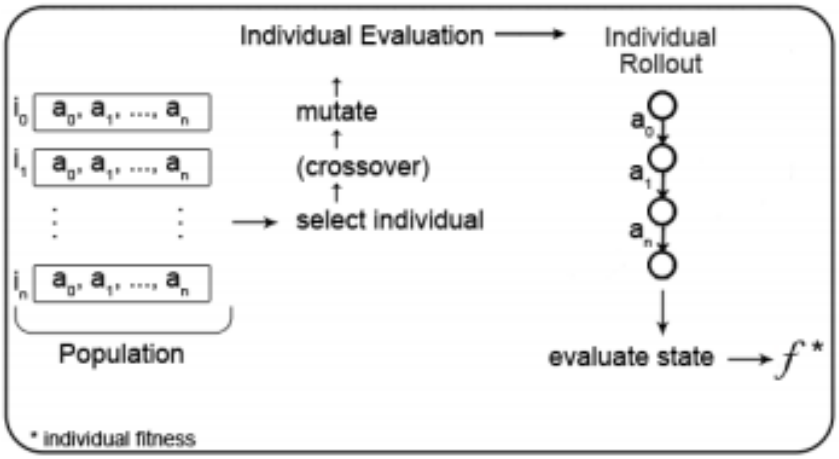

Figure 5 RHEA cycle overview (Perez et al, 2013, p.5)

Chapter 2: **Literature Review**

2.1 GVGAI

Benchmarking AI algorithms using game-based AI competitions has proven to be popular and successful in raising awareness of AI research. However, Perez et al describe that most existing competitions focus on one game. This leads to competitors tailoring their AI algorithm specifically to that game rather than improving the quality of their algorithm. They argue that an algorithm that is able play “any number of unseen games” is what truly constitutes to an artificial *general* intelligence. Thus, the General Video Game Artificial Intelligence (GVGAI) competition is one the first created that does not focus on playing one game, but rather on a “number of unseen games,” (Perez et al 2015).

The GVGAI framework is a Java port of Py-Video Game Design Language (py-vgdl) redesigned by Perez et al. The framework introduces an interface in which a controller can make decisions on which actions the player in the game should make. Perez et al believe, researchers need to develop their algorithms without knowing which games the agent would play. The GVGAI competition has made available a dedicated webserver that allows competitors to upload their AI agents to be evaluated in a set of unknown games (Perez et al, 2019). During the competition all agents have only 40ms of *decision time.*

The GVGAI framework also has made improvements to the sprites from the py-vgdl. This to make the games more appealing for human players. Figure 6 shows the evolution of graphical quality in the Pac-Man game from py-vgdl to the most recent version of GVGAI framework.

Figure 6 Graphical improvements of Pac-Man game from py-vgdl to the most recent GVGAI framework (left to right). The middle image is an early version of Pac-Man on the GVGAI framework (Perez et al, 2019, p.7)

The GVGAI framework contains a Forward Model which is able to model the environment and simulate future states when an action and current state is given (Perez et al 2019). Perez et al put forward that allowing AI players to simulate

and create plans to approximate the best course of action is something “inherent to human intuition and processing.” They argue since humans use this decision-making process for a wide range of scenarios an AI having this ability would make it a step closer to *general intelligence*.

A controller class named Agent.java must inherit the abstract class in the framework must inherit the abstract class `AbstractPlayer` and implement the methods the constructor `Agent` and `act`. The constructor is called once per game and needs to finish before 1 second of CPU time (Perez et al, 2015). The second function is called every game cycle and is responsible for returning an action before 40 milliseconds. If an agent takes between 40 and 50ms the action returned is `NIL` i.e. no action is returned. If the agent goes over 50ms it will be disqualified from that run.

Two arguments of the same type are given to both `Agent` and `act` those being:

- `ElapsedCpuTimer:` a timer to tell the method when the call will occur.
- `StateObservation:` this is an object which represents the current state of the game and provides a FM.

Therefore, the `StateObservation` object acts as the medium in which states are advanced to the next state after an action. The object allows for copying so AI agents can use the FM and plan actions taken in the game. The stochastic nature of the games found in the framework should typically be dealt by the AI agent (Perez et al, 2015). The `StateObservation` object also provides a wide range of information including a score, the victory state and the current time step. The agent is provided a list of actions by the `StateObservation` object each game within the framework supplies the available actions. Perez et al also mention that the object keeps a list of *observations* some of which include, each sprite in a game being identified by a unique integer and its corresponding behaviour i.e. if it’s stationary or non-stationary, Non-playable characters, collectables and doors. An observation grid is also accessible through the object which relays all observations on a 2-dimensional array. Finally, a *history of avatar events* is provided by the object to access details of collisions between the player and any sprite in the game.

2.2 Real Time NEAT (NERO)

Real Time Neat (rtNEAT), was introduced by Stanley et al in 2005. It removes the general concept of EAs being trained offline. They introduce an algorithm that allows agents to adapt and change in real time while keeping core elements of NEAT i.e. rtNEAT is able to complexify ANN as the game is being played allowing complex behaviour in real time (Stanley et al, 2005). To showcase rtNEAT they came up with the idea of creating the game called NeuroEvolving Robotic Operatives (NERO). The idea involved creating a game where learning is the key

to win, this is emphasised by Stanley et al saying, “without learning NERO could not exist as a game.”

NERO is played by both a human player and an AI agent in this case rtNEAT. The human player acts as a *trainer* who teaches a team of agents in military combat (Stanley et al, 2005). The trainers must design a series of exercises and goals with increasing difficulty so that their team can begin learning. Once the trainer feels ready, they can deploy their team to fight another team trained by another trainer. Stanley et al have found that this made for a “captivating and exciting culmination of training.”

rtNEAT assumes that the entire population is playing at the same time, so fitness scores are constantly being collected as the game progresses. The key question that was posed was how could agents be constantly replaced so that offspring could be evaluated. Stanley et al solution included replacing a single individual every several game ticks. The agent with the worst adjusted fitness is removed and replaced with a child of parents chosen from among the best adjusted fitness values.

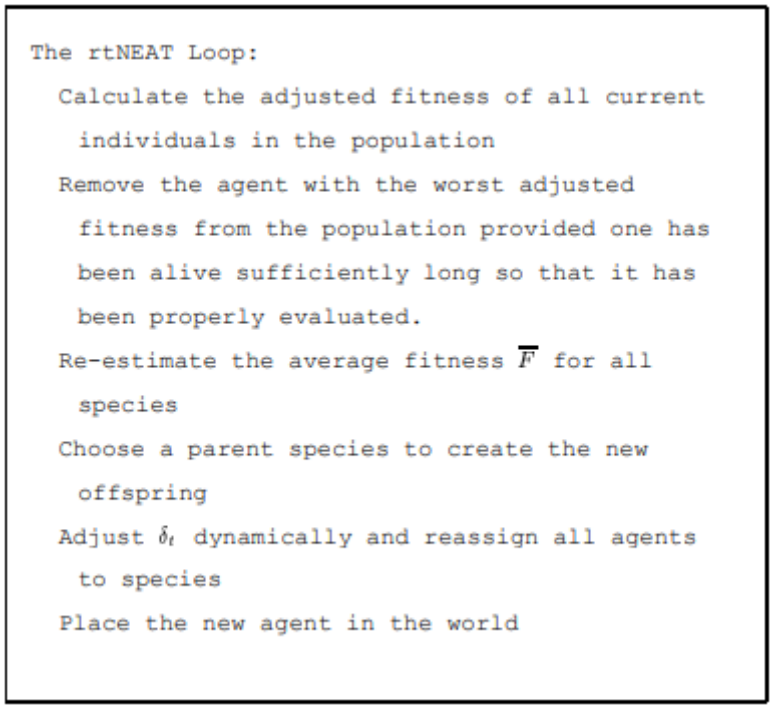

Figure 7 rtNEAT loop performed every tick. (Stanley et al, 2005, p.16)

Figure 7 shows the rtNEAT loop that is performed every ‘n’ tick. Stanley et al states that, as rtNEAT produces one new offspring at a time it cannot replicate NEATs conventional method of speciating as NEAT assigns species to every individual each new generation. Therefore, rtNEAT must speciate in real-time.

2.3 HyperNEAT

Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) is a method introduced by Gauci et al as a way of evolving indirectly-encoded neural networks by using a compositional patten producing network (CPPN) (Gauci et al, 2010). This allows for neural networks to be described in terms of its geometry to view any geometric regularities. Gauci et al used HyperNEAT on checkers as a benchmark. They found that the algorithm was able to extract geometric information from the board providing the agent with an advantage as it was able to generalise better than other approaches. They concluded that the algorithm was able to create smoother and more regular neural networks.

2.4 Applications in General Video Game Playing

Some of the existing applications of NE in general video game playing include Hausknecht et al. introducing HyperNEAT-GGP to play Asterix and Freeway on the Atari platform. The algorithm worked by taking in the raw game screen and analysing it to detect objects that were used as input features for HyperNEAT, using the game score as fitness (Hausknecht et al, 2012). Hausknecht et al in 2014 later extended the algorithm to work on a set of 61 Atari games, where he compared HyperNEAT and NEAT methods (Hausknecht et al, 2014). Finally, Samothrakis et al. used Separable Natural Evolution Strategies to evolve a neural network that learns to play 10 of the GVGAI games. They showed that the methods proposed were able to learn most of the games played (Samothrakis et al, 2015).

2.5 Conclusion

To conclude, the purpose of Background and Literature review was to understand key aspects relating to NeuroEvolution, understand in detail how the NEAT and RHEA algorithms work, analyse and understand the GVGAI framework and to find related work already done in this field. The paper by Stanley et al on rtNEAT provided an existing implementation of a real-time NEAT algorithm, this allows for a more defined research question and an idea of where rhNEAT fits within the scope of NE. rhNEAT differs from rtNEAT with respect to the idea of using a forward model to plan actions to take in game whereas rtNEAT requires a real-time trainer to train the network before it can be tested. rhNEAT should be able to solve simple problems without needing training. Analysing the GVGAI framework has proven that the included forward model and implementation of games would be the perfect test for the rhNEAT algorithm as it will heavily rely on concepts of RHEA which the GVGAI framework has extensive support for.

Chapter 3: **Rolling Horizon NEAT (rhNEAT)**

3.1 The NEAT part

Rolling Horizon NEAT (rhNEAT) is configured to have a fixed population size of P individuals. Each one of them represents the configuration of a neural network, which is initialized with a fixed number of inputs and outputs and no connections. As seen with NEAT, the genotype of rhNEAT is formed by node genes and connections genes, which are initialized through a list and a map respectively. This helps keep a record of all the nodes and connections between those nodes. Since NEAT starts with the simplest network first and incrementally makes it more complex through each evolution, rhNEAT begins with an empty connection gene map i.e. there are no connections between any nodes. However, there are input and output nodes that are added to the node gene list at the start. A list of individuals with a fixed size is populated where each of them contain an empty genome with only the input and output nodes (Perez et al, 2020).

As explored in Section 1.3, mutations in NEAT can affect weights and the topology of the network in different ways, each one of them under certain probabilities:

- **Mutate link** $(\boldsymbol{\mu_t})$**:** creates a new link between two nodes.
- **Mutate node** $(\boldsymbol{\mu_n})$**:** picks an existing random connection and splits it into two new connections with a new node in the middle; the original connection is disabled and the weight from the first node to the new middle node is set to 1.0, while the connection from the middle to the second node is set to the original connection weight.
- **Mutate weight shift probability** $(\boldsymbol{\mu_{ws}})$**:** alters the weight of a connection, shifted by a value picked uniformly at random factor in the range $[-W_s, W_s]$.
- **Mutate weight random probability** $(\boldsymbol{\mu_{wr}})$**:** replaces the weight of a connection by a value picked uniformly at random, in the range $[-W_r, W_r]$.
- **Mutate toggle link probability** $(\boldsymbol{\mu_{tl}})$**:** toggles a connection from enabled to disabled, and vice versa.

As can be seen, there are several types of mutations that can occur to the individuals. NEAT mutations, particularly the topological ones, form the basis for complexification of the evolved networks. Since smaller networks optimise faster and adding new mutations to genes can initially result in lower fitness scores, it can result in newer topologies having a small chance of surviving more than one generation (Stanley and Miikkulainen, 2002). This can lead to losing innovation that could prove to be important in the future. In order to prevent this, NEAT uses speciation, which relies on the principle of populations within specific species competing against each other instead of competing against the entire population

as a whole. This provides time for different topological innovations to optimise before they compete against other species. On each generation individuals are firstly organized into species according to a distance function. This function, shown in Equation 1, is a simple linear combination of the number of excess, disjoint genes and the average weight difference $\bar{W}$ of matching genes (Stanley and Miikkulainen, 2002).

$$\delta = \frac{c_1 Excess}{N} + \frac{c_2 Disjoint}{N} + c_3 \bar{W} \quad (1)$$

The coefficients c_1, c_2 and c_3 can be used to tune the impact of these 3 variables. N is the number of connections from the larger genome, which is normalized to 1 if there are less than 20 connections. Each species is represented by a randomly chosen genome from that species in the previous generation. Individuals are compared to these representatives in the distance function. A new individual is placed in the first species with a distance as determined by Equation 1, respecting a maximum threshold CP. If the genome is not able to find a suitable species, a new species is created with this new individual as its representative.

Furthermore, during evolution, a percentage R of individuals in each species are removed but not deleted. The individuals in each species are sorted such that the lowest scoring members will always be removed first. If a species has no individuals or only the representative left, said species is removed from the algorithm.

All tune-able parameters mentioned in section 1.3, NEAT, are present in rhNEAT. The base values used for the experiments in this paper, are included below in Table 1.

Table 1 Base rhNEAT Parameters and their Values (Perez et al 2020)

Parameter	Name	Value
P	Population Size	10
L	Rollout Length	15
R	Individuals discarded per generation	20%
CP	Speciation Threshold	4
c_1	Excess coefficient	1.0
c_2	Disjoint coefficient	1.0
c_3	Weight difference coefficient	1.0
μ_l	Mutate Link Probability	0.5
μ_n	Mutate Node Probability	0.3
μ_{ws}	Mutate Weight Shift Probability	0.5
W_s	Weight Shift Strength	0.4
μ_{wr}	Mutate Weight Random Probability	0.6
W_r	Weight Random Strength	1.0
μ_{tl}	Mutate Toggle Link Probability	0.05
FM_b	Forward Model calls budget	1000

3.2 Rollout

A crucial aspect of rhNEAT is the use of a forward model. An individual is evaluated by first producing the phenotypical NN it encodes, to then roll the state forward from the current game state until a state in the future, L steps ahead. On each one of these steps, features are extracted from the game state and used as input for the NN the individual encodes.

The action suggested by the NN is then applied to that intermediate state, S_n so S_{n+1} is reached, repeating the process until L steps have been given. Figure 3 describes the rollout function in rhNEAT.

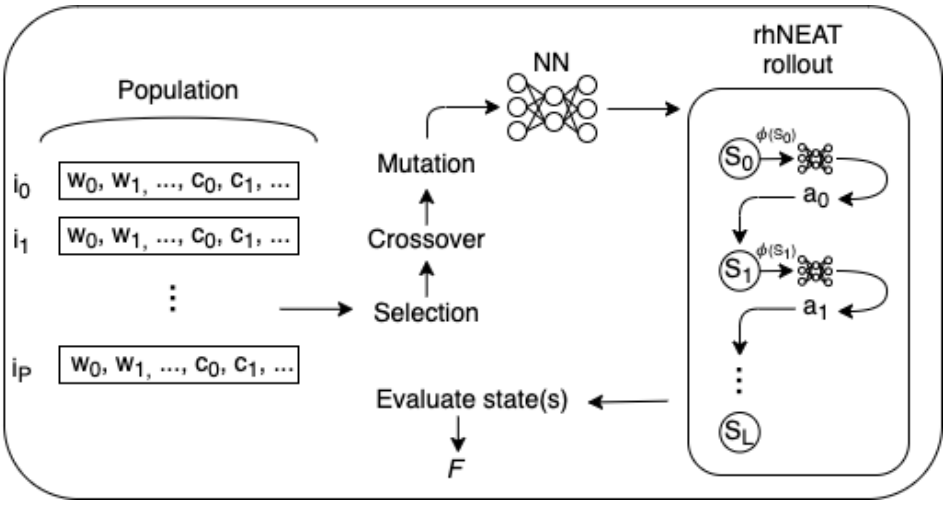

Figure 8 This figure shows the summary of what occurs in rhNEAT after genetic operators are applied. A single individual's Neural Network (NN) is evaluated by performing rhNEAT rollout on it. The features are extracted from each state and is used as the input for the NN. The NN returns an action to pass to the Forward Model. This is repeated rolling the state forward L times. The individual fitness can be evaluated from one or more states visited in the rollout (Perez et al, 2020).

Each visited state can be evaluated using a simple heuristic function, described below in Equation 2. The fitness of an individual can be computed by evaluating either the last state, or a combination of the observed in the states visited. Perez et al have experimented with different configurations we will discuss their results in Chapter 4.

$$h(s) = \begin{cases} 10^6 & \text{win} = \text{True} \\ -10^6 & \text{win} = \text{False} \\ \text{game score} & \text{otherwise} \end{cases} \quad (2)$$

3.2.1 Evaluating rhNEAT's Neural Network Output

When evaluating the output of the NN we receive values for each output node in order. We take the highest value output node and translate that it into the correct action. This process occurs throughout the rollout and at the final end of the game tick to get the best possible actions.

3.3 Rolling Horizon NEAT and GVGAI

Rolling Horizon NEAT (rhNEAT) is the combination of NEAT's ability to procure complex neural networks (NN) that are formed through evolutionary operators (selection, crossover, mutation and speciation) to give an output action and the statistical forward planning features found in RHEA to evaluate an individual's NN.

3.3.1 Overview of the rhNEAT algorithm

Population of individuals are evolved through NEAT genetic operators to encode nodes and connection into a NN. Inputs for the NN are derived from the current game state features while the outputs are fixed nodes representing all actions that can be applied in a given game. Each individuals NN is put through the rollout where it is evaluated by seeing an L number of steps ahead using the output of the NN as actions. Once a computational budget threshold is reached, the algorithm selects the individual with the highest fitness and runs its network forward using features from the current game state as inputs. The output action is then returned to be played in the game. rhNEAT will then again be called in the next frame to select an action and continue playing. At this stage, the population evolved in the previous game tick is initialized again to start a fresh evolutionary process. However, the algorithm can continue evolution from the previous population with only some K percentage of the population being reinitialised. This approach is known as population carrying and is one variants of rhNEAT that has been explored by Perez et al.

3.3.2 Input and Outputs for the GVGAI Framework

rhNEAT requires game state inputs to evaluate an individual. The following game features for all tests carried out in this paper (Perez et al, 2020):

- Avatar x, y position, normalized between [0, 1].
- Avatar x, y orientation.
- Avatar's health points, in [0, MAX_h], where MAX_h is the maximum health points achievable in each game.
- Proportion of up to three resources r_1, r_2, r_3 gathered by the avatar, where each r_i is normalized in [0, 20].
- Distance d and orientation o to the closest instance of a sprite of the following categories:
    - NPC sprite.
    - Immovable sprite.
    - Movable sprite.
    - Resource sprite.
    - Portal sprite.
    - Sprite produced by the avatar.

Distances are normalized in [0, MAX_d], where MAX_d is the maximum possible distance in a game level. Orientation is normalized in [−1, 1]. A zero value shows that the direction of the distance vector to the sprite is parallel with the avatar's orientation i.e. they are both facing in the same direction. When the distance vector to the sprite is pointing in opposite directions the orientation is normalised to −1. Clockwise orientation gradually progresses to 1 while anti clockwise tends to −1. This means that a 90-degree clockwise rotation corresponds to a value of $o = 0.5$.

Health points, resources, distances and orientations to the different sprites are only considered if such features exist in a given game, this to ensure a minimal input size of the network. However, in some GVGAI games, it is possible that some observations do not appear before a certain frame for example, in the game Aliens enemies that are spawned are not visible for the first few frames therefore carrying the population from one tick to the next will lead to complications as two consecutive ticks have differing input sizes. To rectify this in the implementation we reinitialize the whole population when this occurs.

For most games on GVGAI framework, especially the ones used in this paper, the network output nodes can be kept constant, as explained in section 3.2.1 these output nodes correspond to the number of actions available in the game and most them only have a fixed number of actions you can take during it. For those GVGAI games that may change the number of available actions mid-game reinitializing the population would be prudent.

Chapter 4: **Experiments**

Experiments in this paper will be split into two parts. The first will focus on how changing certain parameters in rhNEAT affects the performance of the algorithm. These parameters include the various mutation probabilities (μ), speciation threshold value (CP) and finally the coefficients of the distance function shown in equation 1 (c_1, c_2, c_3).

In the second part of the experiments we will discuss the several variants of rhNEAT conducted by Perez et al. The rhNEAT variants are split into three experimental studies the first study includes differing components in the algorithms through toggling speciation and population carrying. The second looks at the alternative calculation methods. The final, using the best variant from the first set as a baseline, will compare the rhNEAT algorithm against other SFP methods (such as MCTS and vanilla RHEA) and RHEA state of the art.

4.1 Setup

The setup for both parts of the experiments discussed in this paper are the same. The setup is as follows; all experiments are run on 20 games from the GVGAI framework, the same ones used by Gaina et al, 2017. These games provide a wide variety of environments, difficulty and game types. Each of the games have 5 levels, during the experiments we repeat each level 20 times. This means that every game is played 100 times.

A decision budget is given to rhNEAT that determines when evolution should stop before providing an action to be played in the game. The decision budget chosen for this paper is limiting the number of times the Forward Model can be used. However, the decision budget can be other things such as, the number of generations or some time limit. A decision budget ensures that results are uniform across differing machine specifications. Furthermore, in order to provide a fair comparison with other SFP methods, rhNEAT is configured to run with a population size of 10 individuals, a rollout length of 15 and a budget of 1000 Forward Model calls.

4.2 Experiment Part 1 rhNEAT Parameters

In the first part of the experiments we want to determine how changing certain parameters effects the overall performance of the algorithm. In order to do this, we must start out with a baseline rhNEAT version of the algorithm to compare against. Table 1 in section 3.1 outlines all the baseline parameters used. For this baseline version we used speciation, population carrying and used only the final states score as the fitness value for each individual.

4.2.1 Mutation Parameters

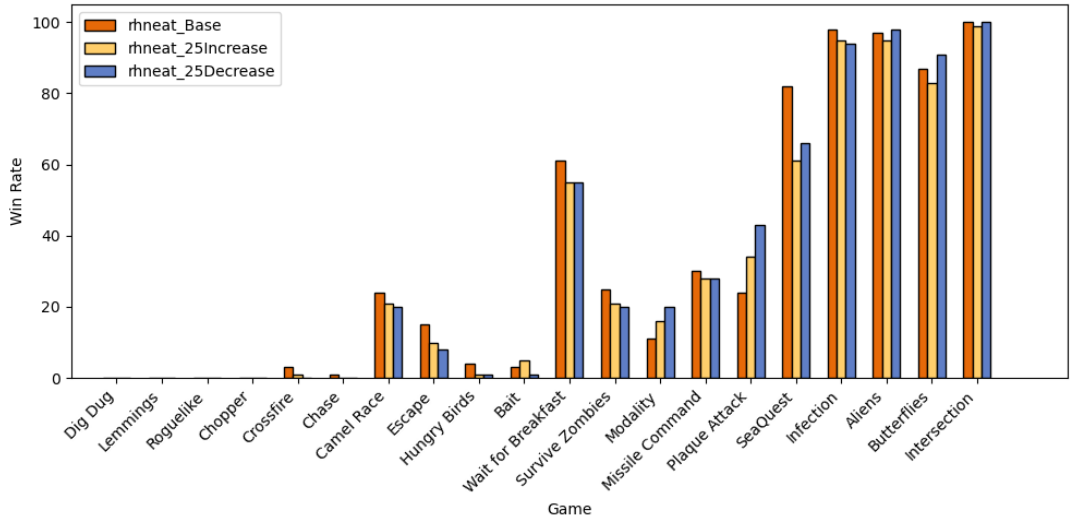

Figure 9 Win rates per game for different rhNEAT mutation parameters. rhNEAT_25increase are results for increasing all mutation probabilities by 25% while 25_decerease is the opposite.

Table 2 shows the win rate and scores of the different mutation parameters for rhNEAT.

$Algorithm$	$Win\ Rate$	$Scores$
$rhneat\_Base$	33.25%	9
$rhneat\_25Decrease$	32.25%	8
$rhneat\_25Increase$	31.25%	3

Two variations of rhNEAT were tested for the mutation parameters. We reduced all mutation probability values by 25% for one test and vice versa for the other. See Appendix A for their tables and their specific values. We note that the baseline variant retains a higher win rate of 33.25% compared to the other two variants as shown in Table 2. In fact, we also see a higher score, 9, compared to the alternatives meaning that the baseline had the highest score in 9 out of the 20 games. This could suggest that a middle ground approach to mutation probabilities is desirable.

$rhneat\_25Decrease$ has a higher win rate of 32.5% compared to $rhneat\_25Increase$ with a value of 31.25% similarly their scores follow the same pattern, but here we notice a much sharper fall in score between them going from 8 to 3 compared to the former test. Overall $rhneat\_25Decrease$ performs worse than the baseline except in a few games namely, Modality and Plaque Attack, where it outperforms the baseline, this could be due to a number of reasons some of which could include that these games are more complex meaning the NN

needs more time to play out the level which a lower mutation rate allows for. Attaining scores in some games may take a while, in this case higher mutation probabilities could result in NN skipping over useful networks that will eventually work well in the game this could be a reason for why $rhneat\_25Increase$ performed the worst compared to its counterpart. If rollout length were increased, we may see even better results for $rhneat\_25Decrease$ but for fairness we have kept the rollout length limited to 15.

Furthermore, we see that the performance across the different games is quite varied. Some of them are close to achieving 100%-win rate such as Infection, Aliens and Intersection while others are close or equal to 0% such as Dig Dug, Lemmings, Roguelike, Chopper and Crossfire. Most other GVGAI studies also experience this. As explained before this could be due to the former list of games rewarding agents more often while the latter do not reward as often or are harder to learn.

4.2.2 Coefficient Parameters

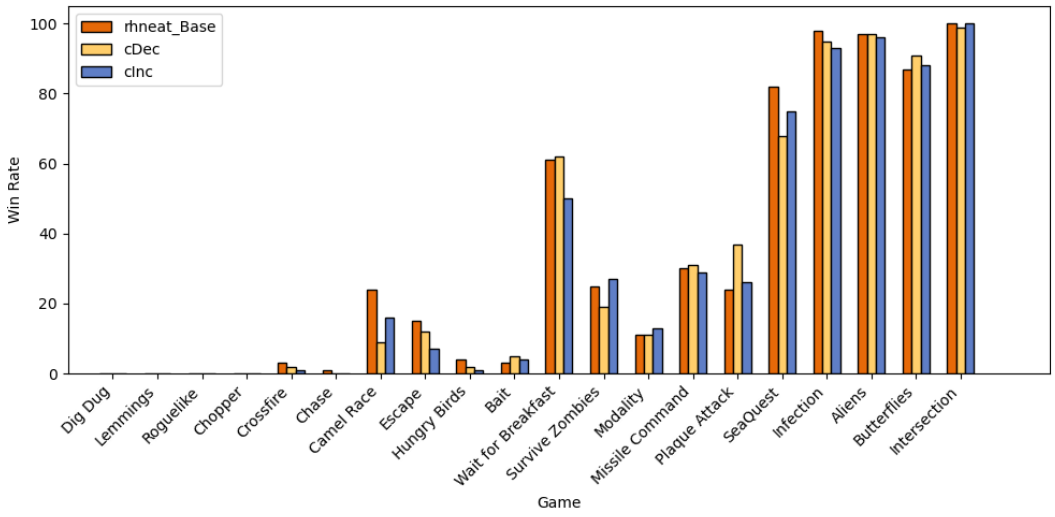

Figure 10 shows the Win Rate for different Coefficient parameters of rhNEAT.

Table 3 shows the win rates and scores of different coefficient parameters for rhNEAT.

$Algorithm$	$Win\ Rate$	$Scores$
$rhneat\_Base$	33.25%	7
$cDec$	32.00%	8
$cInc$	31.30%	5

The coefficients c_1, c_2 and c_3 are used to tune the importance of three factors for the distance function shown in equation 1. These are: the number of excess, E, and disjoint, D, genes and the average weight difference of matching genes, $\bar{W}$. Here we test how increasing and decreasing all these coefficient values by 50% effects the performance of our algorithm. $cDec$ is the one where all coefficient values are reduced by 50% while $cInc$ is the opposite. See Appendix B for the table of changes for both tests. We start by noting that the baseline, rhneat_Base, outperformed in win rate against the other two variants. However, we see the baseline score falls just short of the score for $cDec$ by just a value of 1. It is also

interesting to note that $cDec$ outperformed $cInc$ both in terms of scores and win rates. $cDec$ performed 1.25% worse compared to the baseline but did attain a higher score. The coefficients are proportional to the compatibility distance, δ, therefore decreasing the coefficient values would result in a lower compatibility distance value. This in turn would have the effect of there being less types of species since more individuals would be grouped into larger species as they meet the speciation threshold limit. This may have resulted in some useful innovations being able to live longer as otherwise if they formed a small species they likely would have been removed early on. This also explains why $cInc$ performs the worst with a win rate of 31.30% and only a score of 5. Increasing the coefficients meant that δ became higher resulting in more species forming increasing the likelihood of future species that may perform well being removed early from the population.

4.2.3 Speciation Threshold Parameter

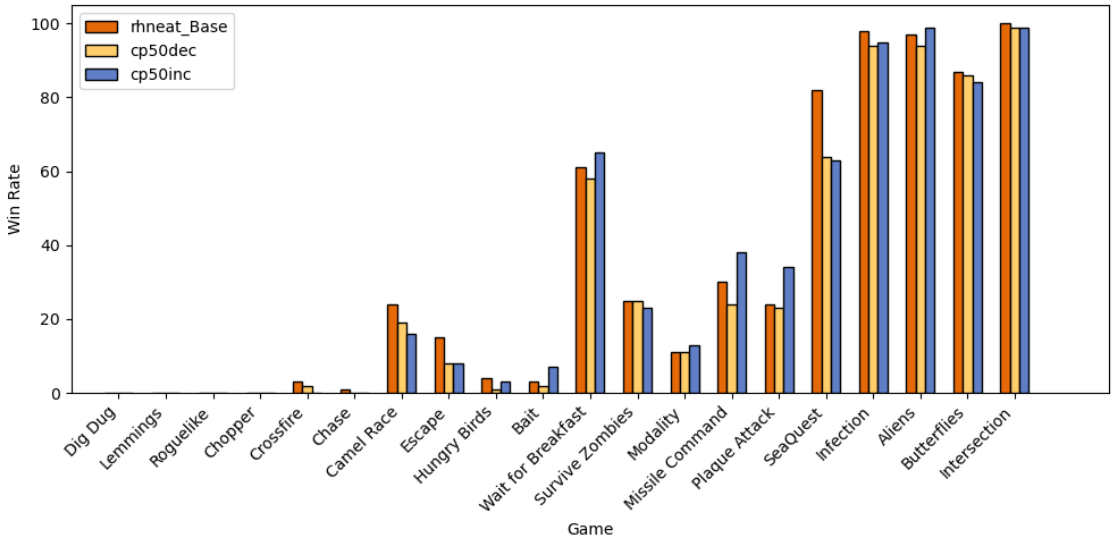

Figure 11 shows the Win Rate for 20 of the GVAGI games for two variations of rhNEAT method for speciation threshold value.

Table 4 shows the win rates and scores of different speciation threshold parameters for rhNEAT.

$Algorithm$	$Win\ Rate$	$Scores$
$rhneat\_Base$	33.25%	5
$cp50Dec$	30.50%	3
$cp50Inc$	32.35%	13

In section 4.2.2 we were tuning compatibility distance, δ, between individuals, however, this value is compared to another value known as speciation threshold, CP. It is this value that decides whether a specific compatibility distance results in individuals being grouped into the same species. CP and δ are used in conjunction to determine which individuals are grouped together as a species. δ values that are less than CP are grouped into the same species. In our results we see that the increased variant, $cp50Inc$, of CP has a higher win rate of 32.50% compared to the lowered version, $cp50Dec$, with only 30.50%. We also note that the increased variant has a much higher score of 13 higher than its counterpart

and the baseline. This means it had the highest score in 13 out of the 20 games available. Having a higher speciation threshold means that more individuals are grouped into the same species as there will be a higher probability that δ values lower than CP will occur. As explained in 4.2.2 this means that there will be a smaller number of species, increasing the likelihood of a useful innovation being able to live longer as smaller species tend to be removed from the population after a short while. This may explain why decreasing CP, lowering the range of acceptance of δ values, means a higher number of small species form, increasing the probability of losing useful innovations early. The baseline does have the highest win rate of 33.25% suggesting that a balanced approach may be the most desirable for the value of CP. See Appendix C for the specific speciation threshold values used in each of the tests.

4.3 Experiment Part 2 rhNEAT Variants

The experiments presented here are the same ones presented in a condensed version of this report that has been accepted for publication by the 2020 IEEE Conference on Games carried out by Perez et al. The objectives for these experiments are to evaluate the performance of the different variants of rhNEAT.

4.3.1 rhNEAT Variants

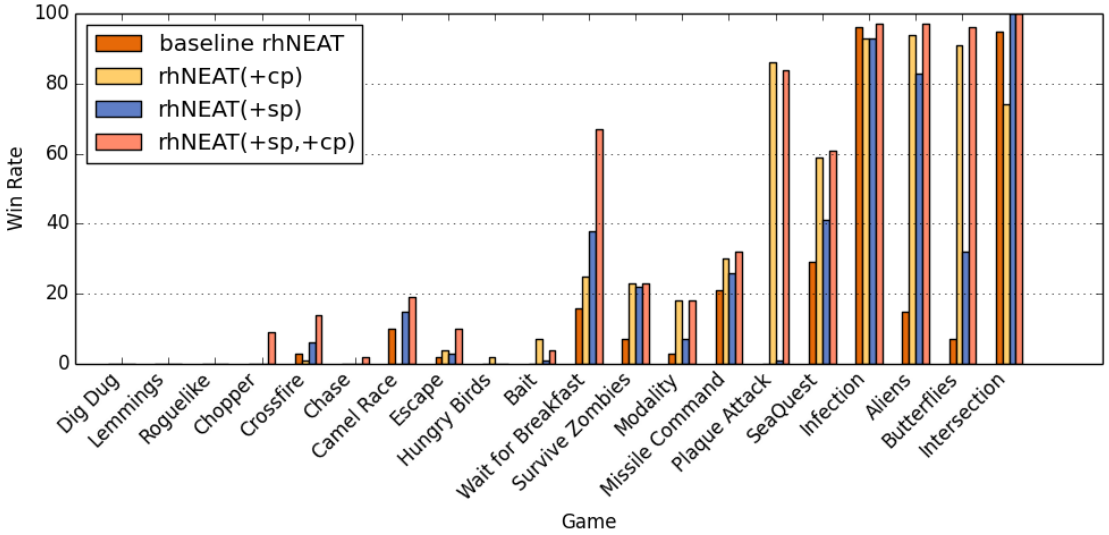

Figure 12 shows the Win Rate for 4 variants of rhNEAT algorithm where each variant has some component like speciation or carry population or both.

Table 5 this table shows the win rate and scores of the different variants of rhNEAT.

$Algorithm$	$Win\ Rate$	$Scores$
$baseline\ rhNEAT$	15.54%	0
$rhNEAT(+cp)$	22.69%	2
$rhNEAT(+sp)$	30.45%	6
$rhNEAT(+sp,+cp)$	36.50%	12

In these set of experiments, we compare the different variants of rhNEAT by incorporating various components into them. Some of these components include:

- Speciation: Individuals in a population are grouped into species according to how related they are to one another. Section 1.3.3 has more details.
- Population Carrying: at each frame some percentage of the population is carried over from the frame. Section 3.3.1 has more details.

We chose these two components as they have the greatest influence on the performance of the algorithm. We use a baseline rhNEAT as the version that does not use any of these components.

- $rhNEAT(+cp)$: Algorithm uses only population carrying
- $rhNEAT(+sp)$: Algorithm uses only speciation
- $rhNEAT(+sp,+cp)$: Algorithm incorporates both speciation and population carrying.

All variants use only the final state value at the end of rollout as their individual fitness. From the results we clearly see that using speciation and population carrying increases both win rate and game scores. The baseline rhNEAT performs the worst with only a 15.54%-win rate. We note that there is an increase in performance to 22.69% when speciation is added, while adding population carrying increases the win rate to 30.45%. Furthermore, the final variant with both components reaches a 36.5%-win rate, this being the highest number of best win rate across games while also obtaining the best score in 12 out of the 20 games tested. The results suggest that the algorithm benefits from having different niches of weights in the population of rhNEAT, and even more so when the population is kept between frames (Perez et al, 2020). Agents can adapt to drastic changes in game mechanics or the removal of features used as input for rhNEAT by reinitialising the entire population, this also may be a reason for the increased performance when adding population carrying.

4.3.2 rhNEAT Reward

This section presents alternatives to how fitness is assigned to individuals. All variants use the best performing algorithm from the previous tests which was found to be rhNEAT with speciation and population carrying.

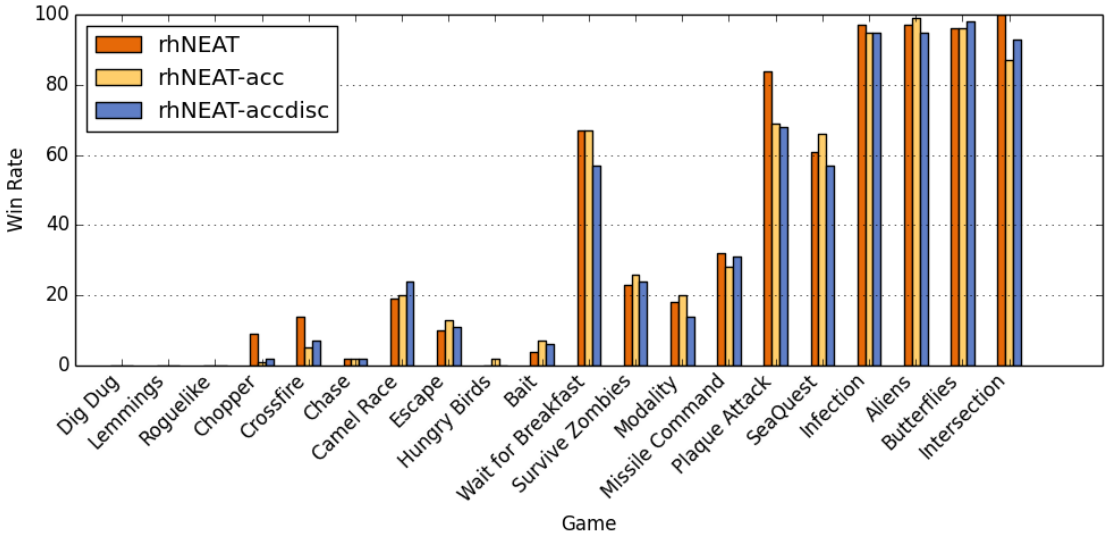

Figure 13 shows the Win Rates per game of different rhNEAT variants: using the evaluation of the last state of the rollout (rhNEAT), accumulated through all states visited (-acc) and accumulated and discounted (-accdisc).

Table 6 shows the Win Rate and Scores of the different reward and fitness alternatives of rhNEAT.

$Algorithm$	$Win\ Rate$	$Scores$
$rhNEAT$	36.50%	12
$rhNEAT - acc$	35.15%	5
$rhNEAT - accdisc$	34.20%	4

In this section we aim to analyse three procedures to compute the reward of a given rollout. These versions include:

- rhNEAT: uses the final game state at the end of rollout value as the reward.

- rhNEAT-acc: provides an accumulated sum of the values of all states visited during the rollout
- rhNEAT-accdisc: provide the same accumulated sum as above but also includes a discount factor $\gamma = 0.9$.

From the results we can see that the win rate between rhNEAT and rhNEAT-acc is not that high with only a 1.35% difference. This suggests that considering all states and considering only the last state has little impact on our win rate. However, we see a bigger fall in win rate from the baseline if the accumulated sum is discounted. From Figure 13 we see that rhNEAT-accdisc tends to achieve marginally worse results per game, while rhNEAT and rhNEAT-acc achieve higher win rates. We also see that rhNEAT obtains the highest scores in most games, 12 compared to the other two variants. Therefore, we can conclude that the version of the algorithm that only uses the final state's evaluation is the best hence the most appropriate to use when comparing against the alternative versions of rhNEAT and other approaches.

4.3.3 rhNEAT Comparison with Other Algorithms

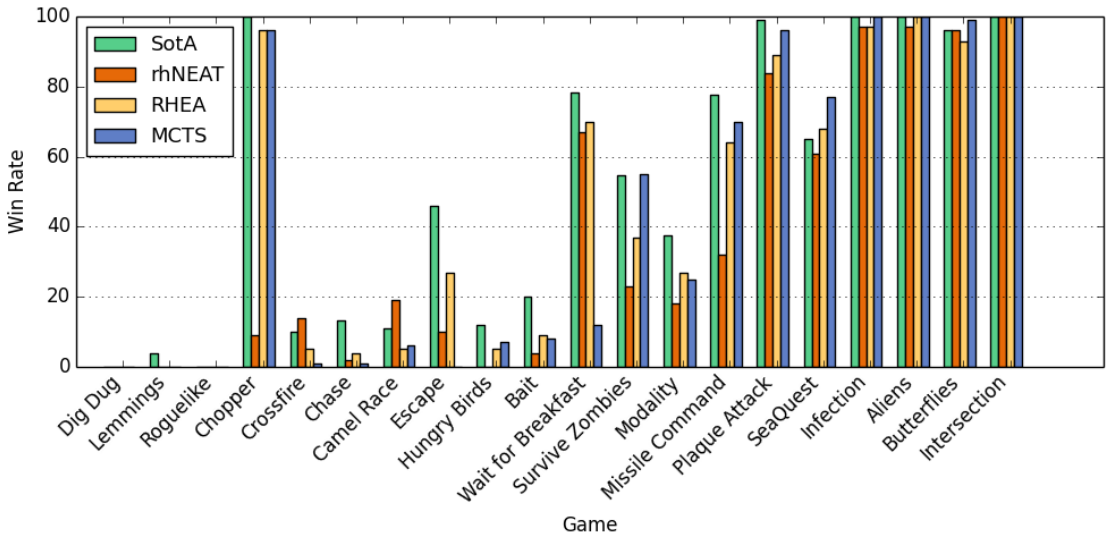

Figure 14 Win rates per game for rhNEAT, MCTS, RHEA and state of the art results RHEA methods.

Table 7 shows the win rate and highest scores achieved by different SFP methods including rhNEAT.

$Algorithm$	$Win\ Rate$	$Scores$
$rhNEAT$	36.50%	1
$RHEA$	44.80%	0
$MCTS$	42.65%	4
$SotA$	51.21%	16

In this last test we compare rhNEAT to other methods used in previous studies. Namely we will compare rhNEAT against RHEA and MCTS specifically we will be using their sample versions from GVGAI. We will also compare it against RHEA state of the art (SotA) refer to the paper "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing" for more information (Gaina et al, 2020). The tests will run on the same 20 games from the GVGAI framework with the same number of repetitions and levels. Parameters such as

population size, individual/rollout length, budget and state evaluation functions will be kept the same. The exploration constant, K, for MCTS is set to $\sqrt{2}$. We will compare the best version of rhNEAT found using previous tests, rhNEAT with base values from Table 1, speciation and population carrying, with MCTS, RHEA and state-of-the-art results obtained by RHEA.

From the results we find that rhNEAT performs below that of the other methods with only a 36.50%-win rate compared to RHEA and MCTS with 44.80% and 42.65% respectively. As expected SotA achieves the highest win rate but it is to be noted that SotA compiles results from multiple configurations of RHEA to represent a goal that other single configured SFP methods can aim for. Nonetheless we see some strengths of rhNEAT where the win rate in several games surpasses that of RHEA, MCTS and SotA. These games include Crossfire and Camel Race. rhNEAT also achieves a 100%-win rate in Intersection the same as the other methods.

From this we see that rhNEAT can play better than some agents when it comes to games with sparse rewards. SFP methods traditionally struggle playing well in these types of games as little information is provided to agents so very little guidance is given to search for optimal solutions (Gaina et al, 2019).

Finally, we see that rhNEAT played Infection, Aliens, Butterflies and Intersection to a degree that is comparable to the other methods where they achieved similar high win rates.

Chapter 5: **Conclusion**

This paper sets out to implement and introduce rhNEAT, a new Statistical Forward Planning (SFP) algorithm that combines the concepts of Rolling Horizon Evolutionary Algorithms (RHEA) and of NeuroEvolution of Augmented Topologies (NEAT), and evaluate it in a variety of games from the General Video Game AI (GVGAI) framework. We compared rhNEAT to other variants of itself to determine the most optimal version. We compared the final version of rhNEAT's performance against other SFP methods including a state-of-the-art method.

The algorithm works by receiving game features as input for a neural network (NN) from which it outputs one of the possible game actions. The topology and weights of the NN are evolved using an evolutionary algorithm. Each individual's NN is assigned a fitness by rolling the game forward, applying the actions specified by the NN given the input features of each state, until the end of the rollout is reached and the final game state is evaluated.

From the comparison of the different parameters of rhNEAT and by analysing their performance across 20 GVGAI games we find that the base rhNEAT values outlined in Table 1 offers the best performance. We also find that the best rhNEAT variant from those that were explored was the one using speciation, population carrying and using the last game state reached in rollout as the fitness of the individual.

We can conclude our research question:

Can rhNEAT be a new SFP method with performance similar to or greater than other popular SFP methods such as RHEA?

Results show that rhNEAT achieves a lower overall win rate than other SFP methods like Monte Carlo Tree Search and RHEA. However, it is able to obtain better results than the state-of-the-art RHEA in two sparse reward type games.

From this we believe that these results display the potential of this method as a new SFP method, even though it did not perform better than other methods it was still able to compete closely to them and even outperform state of the art in two of the games. For these reasons we believe further investigation would be interesting and prudent for this new algorithm.

Chapter 6: **Further Work**

The literature review on NEAT variants is vast see sections 2 for ones on HyperNEAT and NERO. Further work could be done on combining the other NEAT variants with Rolling Horizon methods. One such example could be to explore how indirect representations such as Compositional Pattern Producing Networks (CCPNs) via HyperNEAT effects performance when introduced to rhNEAT. Further work can also be carried on rhNEAT specific parameters as this paper hasn't tested singular variable changes i.e. this paper tested all mutation parameters at once instead of individually measuring each mutation probability. Further tests may bring light to a more optimal rhNEAT setting. An additional possibility could be to look into dynamically alternating between different rhNEAT settings depending on the game state and the perceived reward landscape, as results showed that different variants seem to perform differently depending on these factors. Another interesting idea to explore is the usage of convolutional layers to extract features from the game screen. Finally, the idea of using general game features with forward planning can be extended to other problems, such as Grammatical Evolution (O'Neill et al, 2001), Tangled Program Graphs (Kelly et al, 2017) or different variants of Genetic Programming (Perez et al, 2020).

References

Baldominos, Alejandro & Sáez, Yago & Isasi, Pedro. (2019). On the Automated, Evolutionary Design of Neural Networks-Past, Present, and Future. Neural Computing and Applications. 10.1007/s00521-019-04160-6.

Diego Perez-Liebana and Simon M. Lucas and Raluca D. Gaina and Julian Togelius and Ahmed Khalifa and Jialin Liu. (2019). General Video Game Artificial Intelligence. Morgan & Claypool Publishers. London.

Gaina R.D., Liu J., Lucas S.M., Pérez-Liébana D. (2017). Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing. In: Squillero G., Sim K. (eds) Applications of Evolutionary Computation. EvoApplications 2017. Lecture Notes in Computer Science, vol 10199. Springer, Cham

Gaina R.D., S. M. Lucas and D. Perez-Liebana. (2017). "Rolling horizon evolution enhancements in general video game playing," 2017 IEEE Conference on Computational Intelligence and Games (CIG), New York, NY, 2017, pp. 88-95.

Gaina, Raluca & Lucas, Simon & Perez Liebana, Diego. (2017). Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing.

Gaina, Raluca & Couëtoux, Adrien & Soemers, Dennis & Winands, Mark & Vodopivec, Tom & Kirchgebner, Florian & Liu, Jialin & Lucas, Simon & Perez Liebana, Diego. (2017). The 2016 Two-Player GVGAI Competition. IEEE Transactions on Computational Intelligence and AI in Games. PP. 1-1. 10.1109/TCIAIG.2017.2771241.

Gnana Sheela K, Deepa SN (2013). Review on methods to fix number of hidden neurons in neural networks. Math Probl Eng 2013

K. O. Stanley and R. Miikkulainen. (2002). "Evolving Neural Networks through Augmenting Topologies," in *Evolutionary Computation*, vol. 10, no. 2, pp. 99-127, June 2002.

K. O. Stanley, B. D. Bryant and R. Miikkulainen, (2005). "Real-time neuroevolution in the NERO video game," in *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 653-668, Dec. 2005.

Lara-Cabrera, Raul & Nogueira-Collazo, M. & Cotta, Carlos & Fernández-Leiva, Antonio. (2015). Game artificial intelligence: Challenges for the scientific community. CEUR Workshop Proceedings. 1394. 1-12.

Perez-Liebana, D, Samothrakis, S., Lucas, S.M., & Rohlfshagen, P. (2013). Rolling horizon evolution versus tree search for navigation in single-player real-time games. GECCO.

Perez-Liebana, D, Samothrakis, S, Togelius, J, Lucas, SM & Schaul, T. (2016). General video game AI: Competition, challenges, and opportunities. in 30th AAAI

Conference on Artificial Intelligence, AAAI 2016. AAAI press, pp. 4335-4337, 30th AAAI Conference on Artificial Intelligence, AAAI 2016, Phoenix, United States, 2/12/16.

Perez Liebana, Diego & Samothrakis, Spyridon & Togelius, Julian & Schaul, Tom & Lucas, Simon & Couëtoux, Adrien & Lee, Chen-Yu & Lim, Chong-U & Thompson, Tommy. (2015). The 2014 General Video Game Playing Competition. IEEE Transactions on Computational Intelligence and AI in Games. 8. 1-1. 10.1109/TCIAIG.2015.2402393.

Radcliffe, N. J. (1993). Genetic set recombination and its application to neural network topology optimization. Neural computing and applications, 1, 67–90.

Samothrakis, Spyridon & Perez Liebana, Diego & Lucas, Simon & Fasli, Maria. (2015). Neuroevolution for General Video Game Playing. 200-207. 10.1109/CIG.2015.7317943.

Smith, Maynard J (1978) Optimization theory in evolution. Ann Rev Ecol Syst 9:31–56

Wright, A.H. (1991). Genetic Algorithms for Real Parameter Optimization. FOGA. 205–218.

Yannakakis, Georgios N., and Julian Togelius. (2018). *Artificial Intelligence and Games*. Springer

Perez-Liebana, D., Gaina, R.D., Drageset, O., Ilhan, E., Balla, M. and Lucas, S.M., 2019, October. Analysis of Statistical Forward Planning Methods in Pommerman. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (Vol. 15, No. 1, pp. 66-72).

Diego Perez-Liebana & Muhammad Sajid Alam & Raluca D. Gaina. (2020). Rolling Horizon NEAT for General Video Game Playing. Accepted in IEEE Conference of Games

Chaslot, G., Bakkes, S., Szita, I. and Spronck, P., 2008, October. Monte-Carlo Tree Search: A New Framework for Game AI. In AIIDE.

Gauci, J. and Stanley, K.O., 2010. Autonomous evolution of topographic regularities in artificial neural networks. Neural computation, 22(7), pp.1860-1898.

M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone, “Hyperneat-ggp: A hyperneat-based atari general game player,” in Proceedings of the 14th annual conference on Genetic and evolutionary computation, 2012, pp. 217–224.

M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, “A neuroevolution approach to general atari game playing,” IEEE Trans. on CI and AI in Games, vol. 6:4, pp. 355–366, 2014.

Gaina, R.D., Lucas, S.M. and Pérez-Liébana, D., 2019, July. Tackling sparse rewards in real-time games with statistical forward planning methods. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, pp. 1691-1698).

R. D. Gaina, S. Devlin, S. M. Lucas, and D. Perez-Liebana, “Rolling Horizon Evolutionary Algorithms for General Video Game Playing,” arXiv:2003.12331, 2020.

M. O’Neill and C. Ryan, “Grammatical evolution,” IEEE Transactions on Evolutionary Computation, vol. 5, no. 4, pp. 349–358, 2001.

S. Kelly and M. I. Heywood, “Multi-task learning in atari video games with emergent tangled program graphs,” in Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 195–202.

Appendix A – Mutation Probability Changes

Parameter	Name	Value
μ_l	Mutate Link Probability	0.375
μ_n	Mutate Node Probability	0.225
μ_{ws}	Mutate Weight Shift Probability	0.375
μ_{wr}	Mutate Weight Random Probability	0.45
μ_{tl}	Mutate Toggle Link Probability	0.0375

The table above shows all mutation probabilities reduced by 25% from the baseline values show in Table 1.

Parameter	Name	Value
μ_l	Mutate Link Probability	0.625
μ_n	Mutate Node Probability	0.375
μ_{ws}	Mutate Weight Shift Probability	0.625
μ_{wr}	Mutate Weight Random Probability	0.75
μ_{tl}	Mutate Toggle Link Probability	0.0625

The table above shows all mutation probabilities increased by 25% from the baseline values show in Table 1.

Appendix B – Coefficient Changes

Parameter	Name	Value
c_1	Excess Coefficient	0.5
c_2	Disjoint Coefficient	0.5
c_3	Weight difference coefficient	0.5

The table above shows all the coefficients for the distance function reduced by 50% from the baseline values show in Table 1.

Parameter	Name	Value
c_1	Excess Coefficient	1.5
c_2	Disjoint Coefficient	1.5
c_3	Weight difference coefficient	1.5

The table above shows all coefficients for the distance function increased by 50% from the baseline values show in Table 1.

Appendix C – Speciation Threshold Changes

Parameter	Name	Value
CP	Speciation Threshold	0.2

The table above shows the speciation threshold decreased by 50% from the baseline values show in Table 1.

Parameter	Name	Value
CP	Speciation Threshold	0.6

The table above shows the speciation threshold increased by 50% from the baseline values show in Table 1.