Neuroevolution for RTS Micro

Aavaas Gajure*, Sushil J Louis†, Daniel J Mendoza‡ and Siming Liu§
Department of Computer Science and Engineering, University of Nevada Reno
Reno, Nevada
Email: *avvs@nevada.unr.edu, †sushil@ece.unr.edu, ‡dmendez@nevada.unr.edu, §simingl@unr.edu

Abstract—This paper uses neuroevolution of augmenting topologies to evolve control tactics for groups of units in real-time strategy games. In such games, players build economies to generate armies composed of multiple types of units with different attack and movement characteristics to combat each other. This paper evolves neural networks to control movement and attack commands for a group of ranged units skirmishing with a group of melee units. Our results show that neuroevolution of augmenting topologies can effectively generate neural networks capable of good micro control for our ranged units against a group of hand-coded melee units. The evolved neural networks lead to kiting behavior for the ranged units which is a common tactic used by professional players in ranged versus melee skirmishes in popular real-time strategy games like Starcraft. The evolved neural networks also generalized well to other starting positions and numbers of units. We believe these results indicate the potential of neuroevolution for generating effective micro in real-time strategy games.

Keywords—neural networks, evolution, NEAT, RTS micro

I. INTRODUCTION

Real Time Strategy (RTS) games are a genre of multi-player video games where players take actions concurrently and the underlying game world dynamically changes over time. The overarching objective of the the game is to establish a position capable of defending against and destroying opponents. Actions in the game can be largely divided into two modes: "macro" and "micro". Macro management relates to long term strategic decisions and is concerned with resource gathering, spending those resources on research, deciding on the type and number of units to build, and in building those units. Micro management is concerned with quick and short term tactical control of units usually during a skirmish between a group of friendly units against an opponents group. RTS game environment are a partially observable and imperfect information environment due to a restricted view through the camera on a part of the whole map and a "fog of war" which hides information form parts that have not been explored. Players have to control numbers of units ranging from tens to hundreds while simultaneously moving the camera around, deciding which units or unit factories to build, selecting units, scouting, and exploring. The state space of typical RTS game like Starcraft is estimated to be more than $10^{50}$ using a conservative branching factor of $10^{50}$ for each frame in a 25 minute game. Consequently, RTS games provide a challenging platform for testing machine learning approaches.

This paper focuses on generating artificial agents capable of good micro control in RTS games. Micro requires quick decision making and fast successive actions to control both movement and attack commands for units in a group. There are multiple types of units, each with its own advantages and disadvantages. Each unit has unique, well-defined characteristics regarding its capabilities, like weaponry, range, speed, maneuverability and others. Good micro can be a deciding factor in a skirmish between two groups with similar characteristics and the player has to consider the attributes of both friendly and enemy units to choose an effective tactic for the particular scenario. The complexity of the different ways in which any unit group can be controlled is as a result challenging, particularly since directives have to be provided in quick reactive time-frames.

RTS games have been used as an environment for AI research and various approaches towards automation of different aspects of RTS game playing have been explored. Approaches like reinforcement learning, scripting, and search, among others, have been used with the end goal of creating a fully automated, human-comparable RTS player. Previous work has explored using Genetic Algorithms (GA) to search for an optimal combination of parameters, which are then used in Potential Fields (PF) and Influence Maps (IM) equations to control the tactical actions of skirmishing units. Our research builds on this previous work in RTS game AI, but takes a different approach. Rather then having a set of parameterized control algorithms, or potential fields, for controlling movement, we explore the feasibility of evolving a neural network to perform good micro. In particular, we explore using Neuro-Evolution of Augmented Topologies (NEAT) to evolve neural networks to effectively and autonomously control units for skirmishes in RTS games.

NEAT evolves both the structure and connection weights of a neural network by utilizing genetic algorithm principles and applying them to a population whose chromosomes represent different instances of networks being explored. Thus, the NEAT approach encodes both the structure and weights of a neural network that tries solve a problem. NEAT incorporates historical markings, speciation and starts complexification from a minimal network. There is good empirical evidence that NEAT can evolve robust solutions for non-trivial problems.

We feed the evolving network with the relative positions of all units in the arena along with the unit’s internal state as inputs. There are three outputs. Two specify a relative 2D position $(dx, dy)$ to move towards and third represents a binary value that determines if we move or attack.

Preliminary results, on an underlying RTS-physics implementing simulation, show that NEAT can evolve networks for micro control of ranged units against a group of melee units. The evolved network generated kiting behaviour for ranged units (copied from vultures in Starcraft) which allowed five vultures to eliminate twenty-five zealots (a strong melee unit) without suffering any damage. We evolved our network on ten
different starting configurations differing in the initial positions and numbers of zealots, and tested the evolved network on configurations with varying numbers of zealots from one to thirty. Our results indicate that evolved networks generalize well to different starting configuration and varying numbers of vultures. We then moved to the recently released Starcraft II (SCII) game API and were able to show that NEAT can evolve good micro on a simple, flat Starcraft II map with no obstacles. Although the networks evolved in SCII are not as effective as those that evolve in our simulation, when pitting 5 hellions against 25 zealots, hellions learn to kite and can on average destroy close to a majority of zealots. As before, the NEAT networks generalized to different numbers of zealots and to different starting locations. We believe our method of network representation can be extended to incorporate new inputs to be applicable to more complex micro scenarios. In addition, we believe that we can significantly improve NEAT evolved micro in SCII with more computational resources.

The remainder of this paper is organized as follows. Section II describes previous approaches related to our current work, section III describes the neural network representation, evolution configurations and NEAT setup. In Section IV we describe our generalization results and lastly in section V we draw our conclusions and explain possible future approaches.

II. RELATED WORK

Significant work has been done over the years in the field of designing effective RTS AI players using different techniques [1]. Buckland et al. described a rule based approach in his book [6] and Rabin and Steve [9] explained scripted agents which is a general approach used by bots that play in starcraft AI ladder matches. Weber and Mateas [10] explored using data mining on gameplay logs to predict the strategy of an opponent. A tree based search approach was used by Churchill, Saffidine and Buro who utilized transition tables to generate trees of actions and performed alpha beta pruning to create agents for 8 vs 8 unit skirmish [11]. Others have also tried reinforcement Learning: Wender and Watson [12] used Q learning and Sarsa to develop a fight or retreat agent. Shantia, Begue and Wiering [13] applied reinforcement learning on neural networks by using neural-fitted and online versions of the Sarsa Algorithm where they implemented a state space representation similar to [12]. Vinyals et al. [14] applied deep reinforcement learning in a Starcraft II environment to train neural networks using gameplay data from expert players. Their representation used raw image features corresponding to the game display called feature maps and they provide baseline results for convolutional, longterm shortTerm memory and random policy based agents.

Potential Fields (PFs), which has been widely used for robot navigation and obstacle avoidance [21][22][23] have also been used for micro. Hagelback and Johansson [16] presented a multi-agent potential field based bot architecture for the RTS game ORTS [17] which incorporated PFs into their unit AI. More recent work has focused on combining PFs with Influence Maps (IM) to represent unit and terrain information. In this context, an influence map is a grid superimposed on the virtual world where each cell is assigned a value by an IM function, which is used by an AI to determine desired actions [18]. Coevolution was used by Avery and Louis in [19] to develop micro behaviours by coevolving influence maps for team tactics and in [20] where they coevolved influence map trees(IMTs) and show that evolved IMTs displayed similar behaviours to hand coded strategies.

NEAT has been applied to dynamic control tasks like double pole balancing without velocity information [5] where it could evolve a robust control policy. It has also been applied to evolving walking gaits for virtual creatures [21] and steering control for driving agents [22][23]. NEAT has also been applied to evolve video game playing agents for games like Ms. Pac-Man [24] and Tetris [25] and has been shown to be applicable to general Atari gameplaying [7]. Board games like 2048 [26] and Go [27] have also been shown to be within reach.

NEAT and its realtime variant rtNEAT have been used to tackle different aspects of RTS games. Olesen et al. [28] used NEAT and rtNEAT to control the macro aspects of the game to match the difficulty of the opponent. Gabriel et al. [29] used rtNEAT to evolve a multi-agent system for Brood war agents based on ontological templates, where they show that their hierarchical method could be used to evolve good micromanagement tactics. NEAT for RTS micro control was applied in [30] where the authors approached the problem by having a neural network control a combat unit’s fight or flee decision, based on various entity properties like weapon cooldown, remaining health, weapon range, enemy weapon range, number of allies in range and number of enemies in range. They used NEAT and rtNEAT and had the fight or flee logic hard-coded for the network to activate; which differs from the approach in this paper where we are directly trying to control unit movement based on the position of friendly and enemy units around the entity being controlled, without further structure.

III. METHODOLOGY

There are different aspects of micro game-play that can be controlled for an entity, such as movement, whether to attack, when to flee, and other such unit specific actions. Controlling all aspects of a micro engagement is therefore a complex endeavor. In this research, we focus on entity movement and firing control. Movement can be further classified into long and short range, based on the timescale within which the action must execute. Longer routes pertain to long distance movement of units, say from a player’s base to an enemy’s base, while short duration movement, like kiteing, are finer tactical movements done in shorter periods of time. Kiting is a strategy that is used by speedy ranged units against slower melee units, where the ranged units fire, run out of range, turn back, fire, and run back out of range again and again avoiding damage to themselves while damaging the enemy. We are trying to evolve networks which can perform similar tactics for ranged units against melee units.

We next describe the NEAT evolutionary algorithm, the neural network representation for NEAT, and the experimental setup used for evolution.
A. Neuro-Evolution of Augmenting Topologies

NEAT is a robust algorithm for evolving neural networks based on genetic algorithm principles. NEAT attributes its robustness to three aspects, specifically that it starts complexifying from minimal structure, that it leverages speciation and its use of historical markings in the genome for crossover and speciation [5]. NEAT allows for continuous complexification by allowing mating together with mutation to fully modify the resulting network and a number of different kinds of mutation are used [5].

Neat evolves a neural network from inputs and outputs specified by the problem domain. We specify the inputs and outputs in more detail below.

Our neural network inputs can be divided into two classes according to the type of information they represent. The first class deals with spatial information and describes the relative position of all entities on the map. In order to be able to represent units consistently and uniformly, regardless of the number, we followed an approach whereby we divide the visible world into regions relative to the current position of the unit being controlled. Figure [1] shows the spatial information being fed into the neural network. In our representation, the world around the entity is divided into 4 quadrants with the entity at its center. The four quadrants are further divided into eight regions separated by the attack range of the unit as shown by the labels R1 to R8 in Figure [1]. Each regions then corresponds to four inputs in the network:

1) the number of enemy units
2) average distance of enemy units
3) number of friendly units and
4) average distance of friendly units in the region

Next, we have four inputs indicating map boundaries. These inputs provide distance from the entity to north, south, east and west boundaries correspondingly. The second class of inputs, feeds the entity with some of the entity’s own internal state. The internal features that we have considered are

1) the current health
2) weapon cool-down
3) current fire or move state and
4) a recurrent input which is the previous attack/move output from the network

Thus, in total the neural network that controls the movement of friendly units in our environment has 40 inputs, 16 that are used to represent the position of all friendly units, 16 that are used to represent, the position of enemy units, 4 boundary sensors and 4 inputs for the internal state as shown in table I. The representation is constructed such that it can capture essential information from different map configurations and number of entities, without having to vary the number of inputs in the network. Once computed, all inputs are scaled between 0 to 1 by representing each value as percentage of a maximum possible value for that input.

| id | Type                                           |
|----|-----------------------------------------------|
| 1-8| enemy avg position per region                 |
| 9-16|friendly avg position per region               |
| 17-24|enemy units per region                         |
| 25-32|friendly units per region                      |
| 33-36|boundary sensors for 4 directions             |
| 37  | self cooldown                                 |
| 38  | self hitpoint                                 |
| 39  | current attack/move state                    |
| 40  | previous attack/move state                   |

1) Output representation: The output is represented by two scalars representing a desired $\delta x$ and $\delta y$ coordinate displacement, and one boolean for whether a unit should fire or move at that instant. $\delta x$ and $\delta y$ displacement output are scalar values from 0 to 1 from which we subtract 0.5 and then scale them to go the coordinate position relative to the unit’s current position. This allows the output to represent any coordinate around the entity in the region corresponding to the scaling factor. The outputs are then fed into the movement mechanism of the simulation or SCII in order to generate movement. The third output is a move or attack command which is a Boolean signal. If the output is greater than 0.5, the entity has to focus on attack and if the output is lower, the entity stops attacking and begins moving to the position signalled by $\delta x$ and $\delta y$ displacement output.

B. Experimental setup

Although our physics-simulation used for preliminary results and for experimentation to explore input and output representations runs fast, the simulated physics and entity properties cannot be easily made identical to SCII. This means that micro evolved in our simulation may not transfer well to SCII. In addition, there are differences in the properties of vultures in our simulation, vultures in Starcraft Brood Wars, and hellions in SCII. However, our simulation runs much faster and we can experimentally try multiple representations, input configurations, and NEAT parameters far more quickly than when using the SCII API. We can then start long evolutionary runs within the SCII environment with more confidence.

For our experiments, we choose the zealot as a representative melee unit and either the vulture or hellion as a representative ranged unit. More specifically, for our simulation environment, we copied zealot and vulture properties
from the Starcraft BW API [31]. When running in SCII, zealots and hellions use SCI properties. Vultures/hellions and zealots deal comparable damage in each attack but have different attack range and movement speeds. Table II shows the properties of the units considered in this paper. Hellions and vultures, when micro’d well can be strong against zealots because of their greater speed and attack range, which makes it possible for a small number of vultures/hellions to kite a bigger group of zealots to death. We expect our approach to evolve good tactical control that can exploit this strength of vultures/hellions against zealots.

**TABLE II. PROPERTIES OF ZEALOTS AND HELLIONS**

| Property          | Vulture | Hellion | Zealot |
|-------------------|---------|---------|--------|
| Hit-points        | 80      | 90      | 100    |
| Damage            | 20      | 13      | 16     |
| Attack Range      | 5       | 5       | 5.1    |
| Speed             | 4.96    | 5.95    | 3.15   |
| Cool-down         | 1.26    | 1.78    | 0.857  |

NEAT evolves the network across generations based on the fitness of the network. To evaluate the fitness of the neural network, we used two different environments: the Starcraft II game and our own simulation of the Starcraft environment which is tailored to capture the micro combat aspects of Starcraft and can be run without graphics for significant speedup.

Our experimental setup had three main components, the NEAT evolution module, the Simulation adapter and the game itself, which could either be Starcraft II or our own simulation of the game. Neat is concerned with running the evolution by assimilating the fitness results and generating networks. We used the SharpNeat implementation of NEAT by Colin Green [32] for the evolution module and adapted it for our purpose. A simulation adapter is the mediator between the evolutionary algorithm and the game which we implemented using sockets to be able to run the game simulations in parallel. It gets the configuration from the NEAT module and sets up the game, it also gets a neural network configuration from the evolution module and feeds inputs with the current game state into the network and uses the output from the network to feed the game and move entities. The adapter returns the final fitness after running the simulation which ends when one of the players has no units left or after a set number of frames. The game component can be either Starcraft or our combat simulation. The architecture diagram of the components is shown in figure 2.

These are diagonal, side by side, surround, surrounded and random. In diagonal, opposing sides spawn in groups diagonally on a square map. Similarly, in the side by side configuration, entity’s appear along the same y-coordinate separated by a distance. In surrounded, vultures or hellions appear at the center in a group surrounded by number of zealots and the opposite is true for surround - hellions or vultures surround zealots. We also experimented on random spawning locations for all units of both players. We kept the number of vultures or hellions constant at five (5) but randomly varied the number for zealots for different configurations. In the rest of the paper, we mean vultures or hellions when we use the term ranged units.

2) *Fitness Function:* We used a fitness function that considers both the damage received and the damage dealt by the our evolving ranged units (vultures and hellions). Over evolutionary time, fitness gradually grows as the ranged units get better at damaging zealots and at evading attacks. At the end of each game run, we sum the remaining hitpoints for both zealots \(H_z\) and ranged units \(H_h\) and subtract the remaining hitpoints of zealots from the remaining hitpoints of the ranged units. We add the maximum hitpoints for all zealot units so that the fitness function is always positive.

For number of starting zealots \(N_z\), remaining zealots \(R_z\), remaining hellions \(R_h\), and maximum hitpoint of zealot \(H_{zmax}\), fitness \(F\) is calculated as:

\[
F = \sum_{n=1}^{N_z} H_{zmax_n} + \sum_{n=1}^{R_h} H_{h_n} - \sum_{n=1}^{R_z} H_{z_n}
\]

We should note that as the hit points of both zealots and ranged units are similar, with increasing numbers of zealots and low numbers of ranged units, this fitness function leans towards giving more weight to damage done than damage received. This could be better balanced in various ways. For example, by multiplying the sum of hitpoints of hellions by a balancing factor. Nevertheless, we found that the current configuration performed well during our experimentation phase.

IV. RESULTS AND DISCUSSION

We experimented with two different RTS game environments. The Starcraft II game, and our simulation of the RTS environment particularly developed for fast simulation of skirmishes. Below, we describe our experiments and analyze the results for each.

A. Simulation Results

In our first set of experiments using our simulation environment, we evolved vultures against a larger group of zealots. Zealots in our simulation, use a hand coded AI which controls each unit as follows: pursue the nearest vulture and attack when it is in range. Both zealots and vultures were given complete map vision - there was no fog of war and thus they did not have to explore the map and could start pursuing their enemy right away. Note that this is a significant difference from SCII.

We ran NEAT on a population size of 50 individuals for 100 generations. The following results are average of 10 different runs of a complete evolutionary epoch, started with...
different random seeds. Various hyper-parameters that we used for the evolution are noted in Table III. Each genome was evaluated based on a complete run of a test configuration, which consisted of 10 different spawning locations with different number of zealots and vultures. We sum the fitness for each of the 10 different training configurations to get the final fitness, which is then forwarded to the NEAT module. We run each scenario until one of the player loses all its units or for a maximum number of frames. We had the option to run our simulation without the graphics rendering which significantly decreased running time compared to Starcraft II.

Initially, we tried to evolve agents only based on a single test configuration of the map, but results showed that they did not generalize well to new scenarios. Using the sum of fitnesses from different configurations led to good generalization across different map configurations and different numbers of units. The 10 different test cases are a sample from the the possible configuration space of different number of zealots and 5 different starting configuration. The training scenarios are listed below.

1) Diagonal, 25 zealots
2) Reversed Diagonal, 20 zealots
3) Side by Side, 10 zealots
4) Reversed Side by Side, 15 zealots
5) Surround, 20 zealots
6) Surround, 10 zealots
7) Surrounded, 20 zealots
8) Surrounded, 25 zealots
9) Random, 15 zealots
10) Random, 25 zealots

In our simulation environment, the average number of generations over ten runs, needed to find the best individual was 80 and the average best fitness was 96% of the maximum fitness possible. We found that the the evolved vultures learned kiting or to hit and run, against the group of zealots. Kiting is an effective tactic for speedy ranged units against slow melee units as explained earlier.

After evolving neural networks to control vultures with kiting ability against groups of zealots, we tested for the generalizability of our result against scenarios that the Vultures did not encounter during the training phase. For each possible starting configuration, we varied the number of starting zealots from 1 to 30 while the number of Vultures was always constant at 5. Here, we note that Vultures were only evolved against the group of 10, 15, 20 and 25 zealots thus, their performance against different number of zealots shows the robustness of the evolved network.

Results of our generalizability tests are shown in Figures 3 and 4. The vertical axis represents the number of units remaining at the end of each game run and the horizontal axis represents the number of zealots against which the five vultures skirmish. We present the results for 6 different starting positions, each starting position averaged over ten runs. Figure 3 shows the number of vultures remaining when times runs out while Figure 4 shows the number of zealots remaining. As shown in Figure 3 we see that vultures’ performance smoothly decreases as the number of zealots increases. The number of vultures never goes below two, a good indication of the robustness of evolved networks.

Generalization with respect to damage done is shown in figure 4 where we note that the zealots are completely destroyed by vultures till the number of starting zealots rises above 13. The number of surviving zealots then gradually increases across all our scenarios. The number of remaining zealots never rises above six another good indicator of micro quality and robustness.

We also found that we must provide a number of different training configurations in order to evolve robustness. Evolving

| Property                  | Simulation | Starcraft II |
|---------------------------|------------|--------------|
| Population                | 50         | 50           |
| Generations               | 100        | 100          |
| Species                   | 5          | 5            |
| Initial Conn Probability  | 0.2        | 0.1          |
| Elitism Proportion        | 0.2        | 0.3          |
| Selection Proportion      | 0.2        | 0.2          |
| Asexual Offspring Proportion | 0.5    | 0.8          |
| Sexual Offspring Proportion | 0.5      | 0.2          |
| Inter-species Mating      | 0.01       | 0.01         |
| Connection Weight Range   | 5          | 7            |
| Probability Weight Mutation | 0.95    | 0.95         |
| Probability Add Node      | 0.01       | 0.02         |
| Probability Add Connection | 0.025  | 0.04         |
| Probability Delete Connection | 0.025  | 0.025        |
with only one configuration, results in much less robust networks whose performance might jump form high to low or low to high when changing the number of zealots even by as little as one zealot. In one case removing a single zealot significantly decreased vulture performance. This result is not unexpected since much research in neural networks and other machine learning has shown similar effects.

B. Starcraft II Results

In Starcraft II, we evolved hellions which are a ranged units that can do splash damage against zealots which are strong melee units. We control the movement and attack commands for the hellions after translating the outputs from NEAT evolved neural networks to the commands for Starcraft II using the API. We ran the game at the top speed of 16. As Starcraft II runs relatively slow even at top speed, we ran on 15 machines for 24 hours and would have preferred to have a much larger cluster.

For Starcraft II experiments, we ran on a population size of 50 for 100 generations and the results averaged over 10 times for the final results. We use the same NEAT parameters as for our simulation and given in III. Unlike our simulation, we used the sum of fitness for only three different configurations to get the individual fitness. The three configurations and corresponding number of zealots are listed below; the number of hellions is always five.

1) Diagonal, 25 zealots
2) Random, 20 zealots
3) Side by Side, 15 zealots

Over 10 runs, the average number of generations needed to find the best individual was 85 and the average best fitness was 88% of the maximum fitness possible. The evolved network also displayed kiting behaviour against the zealots - similar to our findings from the simulation approach.

We tested for the generalizability of the evolved networks in similar fashion to the tests for then simulation environment. That is, we tested the best evolved network against new configurations and with varying number of zealots. Here, we note that hellions only evolved against groups of 15, 20 and 25 zealots, and on only three configurations. Generalization results are shown in Figure 5 and 6. The vertical axis represents the number of units remaining at the end of each game run and the horizontal axis represents the number of zealots remaining when skirmishing with five hellions. We present the results for six different starting positions with the number of zealots varying from 1 to 30. We ran the simulation for five runs on the same starting configuration to get the average number of remaining units. We expect to do more runs as computational resources allow.

As shown in figure 5, we see a downward trend for then number of remaining hellions starting from 5. However, unlike in our simulation, the trends are different for different starting configurations. Diagonal and side by side show good performance across different numbers of zealots while random and surrounded perform comparatively lower. The gradual decrease is expected as the hellions get overwhelmed by the increasing number of zealots. Still, we can see from the graph that hellions are generalizing well with respect to damage received against different number of zealots and different starting positions.

Generalization with respect to damage done is shown in figure 6 where we again note that the hellions perform well for diagonal and side by side scenarios while performing comparatively lower in random and surrounded scenarios. The number of remaining zealots for each scenario only gradually increases on these previously unseen scenarios and indicates generalizability of our evolved result.

C. Comparison of Results

We have shown that the NEAT generated neural networks from from Starcraft II and our simulation were able to generalize with respect to different starting positions and different numbers of zealots. Ranged units performed well against smaller numbers of zealots and performance decreased with increase in number of zealots for both environments. The gradual progression of values for different series without major deviance indicates that the evolved network is robust against changes in both starting position and number of zealots. Videos at [https://www.cse.unr.edu/~aavaas/Micro.html](https://www.cse.unr.edu/~aavaas/Micro.html) serve well to indicate the quality and robustness of the evolved micro.
Neat was able to evolve kiting behaviour in both our simulation and in the Starcraft II environments but there are some differences between the results from two environments. The evolved network in Starcraft II seemed to perform comparatively worse than our simulation. We believe the fewer training configurations, the increased complexity of SCII, and differences in the AI we evolved against, account for these differences.

V. Conclusion and Future Work

Our research focused on using NEAT to evolve neural networks that could provide robust control of a squad in an RTS game skirmish. We showed that our representation of the game state provided to NEAT sufficed to evolve high performing micro, while training on a variety of scenarios leads to more robust and high performing micro. The evolved networks generalized well to different numbers of opposing units and different starting configurations.

We used our own simulation environment for initial experimentation and exploration. Because our simulation runs much faster than Starcraft II, we were able to explore multiple representations and evolutionary parameters to hone in good representations and parameters. We then used this experience to try reproduce our results in the popular RTS game. Starcraft II. Here, we ranged hellions evolved kiting behaviour against melee zealouts - like in our simulation environment and meeting our expectations. We believe these results show that NEAT holds promise as a potential approach to evolving RTS game micro.

With a general neural network representation and with NEAT, we think that our approach can be effectively extended to approach more complex scenarios and group configurations. We are working on probabilistic activation model for outputs: we can consider the output of the neural network as the probability of it being active rather than comparing it against the threshold. Using recurrent neural networks would enable incorporating sequential strategies spanning multiple time frames. In addition, we will be adding more game state information about opponents, considering a multi-objective formulation of the fitness function, and obtaining and using much larger computational resources.

REFERENCES

[1] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, “A survey of real-time strategy game ai research and competition in starcraft,” IEEE Transactions on Computational Intelligence and AI in games, vol. 5, no. 4, pp. 293–311, 2013.

[2] M. Buro, “Call for ai research in rts games,” in Proceedings of the AAAI-04 Workshop on Challenges in Game AI, 2004, pp. 139–142.

[3] G. Robertson and I. Watson, “A review of real-time strategy game ai,” AI Magazine, vol. 35, no. 4, pp. 75–104, 2014.

[4] S. Liu, S. J. Louis, and C. A. Ballinger, “Evolving effective microbehaviors in real-time strategy games,” IEEE Transactions on Computational Intelligence and AI in Games, vol. 8, no. 4, pp. 351–362, 2016.

[5] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” Evolutionary computation, vol. 10, no. 2, pp. 99–127, 2002.

[6] J. H. Holland, Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. MIT press, 1992.

[7] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, “A neuroevolution approach to general atari game playing,” IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, no. 4, pp. 355–366, 2014.

[8] M. Buckland and M. Collins, Ai techniques for game programming. Premier press, 2002.

[9] S. Rabin, Ai game programming wisdom 4. Nelson Education, 2014, vol. 4.

[10] B. G. Weber and M. Mateas, “A data mining approach to strategy prediction,” in Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on, IEEE, 2009, pp. 140–147.

[11] D. Churchill, A. Saffidine, and M. Buro, “Fast heuristic search for rts game combat scenarios,” in AHIDE, 2012, pp. 112–117.

[12] S. Wender and I. Watson, “Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: Broodwar,” in Computational Intelligence and Games (CIG), 2012 IEEE Conference on, IEEE, 2012, pp. 402–408.

[13] A. Shantia, E. Begue, and M. Wiering, “Connectionist reinforcement learning for intelligent unit micro management in starcraft,” in Neural Networks (IJCNN), The 2011 International Joint Conference on, IEEE, 2011, pp. 1794–1801.

[14] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al., “Starcraft ii: A new challenge for reinforcement learning,” ArXiv preprint arXiv:1708.04782, 2017.

[15] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” The international journal of robotics research, vol. 5, no. 1, pp. 90–98, 1986.

[16] J. Hagelback and S. J. Johansson, “Using multi-agent potential fields in real-time strategy games,” in Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 631–638.

[17] M. Buro, “Orts - a free software rts game engine,” Accessed March, vol. 20, p. 2007, 2007.

[18] A. Uriarte and S. Ontanón, “Kiting in rts games using influence maps,” in Eighth Artificial Intelligence and Interactive Digital Entertainment Conference, 2012.

[19] P. Avery and S. Louis, “Coevolving influence maps for spatial team tactics in a rts game,” in Proceedings of the 12th annual conference on Genetic and evolutionary computation, ACM, 2010, pp. 783–790.

[20] C. Miles and S. J. Louis, “Co-evolving real-time strategy game playing influence map trees with genetic algorithms,” in Proceedings of the International Congress on Evolutionary Computation, Portland, Oregon, IEEE Press, 2006, pp. 0–999.
[21] B. Allen and P. Faloutsos, “Complex networks of simple neurons for bipedal locomotion,” in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, IEEE, 2009, pp. 4457–4462.

[22] L. Cardamone, D. Loiacono, and P. L. Lanzi, “Evolving competitive car controllers for racing games with neuroevolution,” in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ACM, 2009, pp. 1179–1186.

[23] F. G. Durán, F. L. Largo, M. P. López, and R. R. Aldeguer, “Driving bots with a neuroevolved brain: Screaming racers,” *INTELIGENCIA ARTIFICIAL*, 2005.

[24] J. Schrum and R. Miikkulainen, “Evolving multimodal behavior with modular neural networks in ms. pac-man,” in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ACM, 2014, pp. 325–332.

[25] L. E. Gillespie, G. R. Gonzalez, and J. Schrum, “Comparing direct and indirect encodings using both raw and hand-designed features in tetris,” 2017.

[26] T. Boris and Š. Goran, “Evolving neural network to play game 2048,” in *Telecommunications Forum (TELFOR), 2016 24th*, IEEE, 2016, pp. 1–3.

[27] K. O. Stanley and R. Miikkulainen, “Evolving a roving eye for go,” in *Genetic and Evolutionary Computation Conference*, Springer, 2004, pp. 1226–1238.

[28] J. K. Olesen, G. N. Yannakakis, and J. Hallam, “Real-time challenge balance in an rts game using rtneat,” in *Computational Intelligence and Games, 2008. CIG’08. IEEE Symposium On*, IEEE, 2008, pp. 87–94.

[29] I. Gabriel, V. Negru, and D. Zaharie, “Neuroevolution based multi-agent system with ontology based template creation for micromanagement in real-time strategy games,” *Information Technology and Control*, vol. 43, no. 1, pp. 98–109, 2014.

[30] J. S. Zhen and I. D. Watson, “Neuroevolution for micromanagement in the real-time strategy game starcraft: Brood war,” in *Australasian Conference on Artificial Intelligence*, Springer, 2013, pp. 259–270.

[31] (2018). Starcraft vulture, [Online]. Available: [http://liquipedia.net/starcraft/Vulture](http://liquipedia.net/starcraft/Vulture)

[32] C. Green. (2018). Sharpneat, [Online]. Available: [http://sharpneat.sourceforge.net/](http://sharpneat.sourceforge.net/)