Literature Review – Alexander

## Current State of Evolution Simulation Games

# Simulation games that use evolution as a core premise are few and far between. A popular example of an old game that used evolution is Spore (Maxis, 2008). Spore used evolution to have the player change the characteristics and abilities of their character. Spore was one of the first of its kind to bring evolution games to a mass market. However, it did not provide the player with a simulation. The game was structured more as an RPG, one where a player can equip and change their body parts. A newer example of a game using evolution is The Sapling (Stoop, 2021). This game allows players to create plants and creatures and watch how they evolve in an ecosystem. The game has two modes for the player to interact with. The first is a puzzle mode where the player must design an ecosystem to pass specific criteria. The second is a pure simulation mode where the player can watch how the plants and animals mutate and evolve over time. For my purposes, I would like to try and improve on The Sapling, by giving the player agency on how their species evolve, whilst keeping the core to be a simulation game.

## Implementing Neat/RtNeat Algorithm

# Neural evolution algorithms are a reinforcement learning technique that allows an agent to evolve their weights and topology in an incremental process. This is done by running simulations consisting of populations of agents, and at the end of each simulation, the best to achieve a set task in the population are used to create a new generation with evolved neural nets. A simple way to think about this is natural selection, where the best in a population can breed and further the beneficial characteristics. The two algorithms I will be exploring are the NEAT algorithm and the rtNEAT algorithm.

# The NEAT algorithm was first introduced in 2001 to solve three problems with existing weight and topology evolution algorithms (KO Stanley, R Miikkulainen, 2002). The three key points I would like to examine from the report are: how does the network evolve and mutate; how does the algorithm protect innovation; how does the NEAT algorithm minimize topology to increase efficiency.

# The NEAT algorithm allows structural mutation to occur in two distinct ways. The network can add a connection between two existing nodes, or the network can add a new node in the middle of an existing connection. Each time a mutation is added it's given a global incrementing number called the innovation number, which allows you to keep track of the history of innovations.

# To protect newly innovated structures, the members of the population are speciated by the similarity of their genes. The members of the population only compete with the members of their species, and not the population at large. This allows for newly innovated structures to become more efficient without having to compete with the whole population. One of the key benefits of this is that it prevents a network that may evolve into a better solution than the current best from being destroyed before it gets a chance to evolve. This does bring a downside in that it can become expensive to organize each member of the species every time a new population is created.

The NEAT algorithm encourages a simple network over a complicated one by starting from a “minimal structure and [only growing] when necessary”. This is an improvement over older genetic algorithms that started a population with random topologies. (KO Stanley, R Miikkulainen, 2002)

# The NEAT algorithm proved to be very effective at solving simple game simulations, such as the beam balance, but it does have one drawback that makes it less effective for my game. The drawback is that it generates completely new populations after a set time, but I require an algorithm that works in real-time, having new members created within a population.

# The answer to this is an adaptation of the NEAT algorithm called the rtNEAT algorithm, or Real-Time NEAT (K. O. Stanley, et al., 2005). The rtNEAT algorithm was designed as an extension of the NEAT algorithm to allow for players to “interact with evolution during the game” (K. O. Stanley, B. D. Bryant and R. Miikkulainen, 2005). The rtNEAT algorithm posses the same basic structure as the NEAT algorithm in terms of how a network can mutate and how an offspring’s genes are merged from its two parents (K. O. Stanley, et al., 2005; KO Stanley, R Miikkulainen, 2002). The difference comes from two factors: the fitness score is calculated for each game tick for every agent; the worst agents are periodically removed and replaced with offspring from the best parents. This is beneficial as from the players' perspective the agents will start to perform more complex behaviour during gameplay with no jarring population wipes. It does come with an extra cost to perform the fitness calculation.

# The original rtNEAT algorithm chooses parents based on a probabilistic view of the top performers in each species. This leads to an agent that quickly diverges on the optimal solution. For my game, I would like to experiment with new agents being created via an attractiveness score, where agents are more likely to be attracted to high performers. In this way, I can still protect speciation and new networks by adjusting the attractiveness so that members of a species have a higher probability to produce offspring. (K. O. Stanley, et al., 2005)

# Victory fitness functions, where the agent is measured on if they have completed a task or not, have been shown to reach near-optimal gameplay for the game Galcon (A.Fernández-Ares, A.M.Mora, P.García-Sánchez, P.A.Castillo, J.J.Merelo, 2017). However, I would like my game to allow the player to set multiple goals for the player to aim for, and as such I will use a fitness based on slope method which allows me to convert an agent's performance in multiple areas to a single fitness score. (A.Fernández-Ares, A.M.Mora, P.García-Sánchez, P.A.Castillo, J.J.Merelo, 2017).

## References

# Bibliography

A.Fernández-Ares, A.M.Mora, P.García-Sánchez, P.A.Castillo, J.J.Merelo, 2017. Analysing the influence of the fitness function on genetically programmed bots for a real-time strategy game. *https://doi.org/10.1016/j.entcom.2016.08.001.,* 18(1875-9521), pp. 15-29.

K. O. Stanley, B. D. Bryant and R. Miikkulainen, 2005. *Real-time neuroevolution in the NERO video game,* s.l.: IEEE.

KO Stanley, R Miikkulainen, 2002. *Efficient Evolution of Neural Network Topologies,* Honolulu, HI, USA: IEEE.

Maxis, 2008. *Spore.* [Online]   
Available at: http://www.spore.com/getSpore  
[Accessed Feb 2022].

Stoop, W., 2021. *TheSaplingGame.* [Online]   
Available at: https://thesaplinggame.com/  
[Accessed Feb 2022].