Genetically Modified Agents that Slither: Applications of a Genetic Algorithm in Slither.io Bots

Sam Gilbert, Rae Hohle

December 17, 2016

AI - Susan Fox

Abstract- Implementing a genetic algorithm to improve results of an already existing parameterized intelligent agent that plays Slither.io reveals how advanced bot development has become. Agents' performances in Slither.io can improve significantly when generations are developed using information about which characteristics lead to an increase in score. In this project, nine generations of ten agents in Slither.io were created using a genetic algorithm that took into account past game performance as means of selection. It was found that some very poor traits were weeded out by the final generation, while other traits that led to a slow testing process were still present in the final generation due to the lack of accounting for these traits in the selection step of the genetic algorithm. The results have shown the efficacy of implementing genetic algorithms for intelligent agents in massively multiplayer online videogames, but further improvements will need to be made to increase the speed of testing.

I. INTRODUCTION

The use of artificially intelligent agents, has become a popular way to compete in online gaming. Slither.io is a massively multiplayer online game in which the player controls a snake that wanders a map eating glowing orbs while avoiding running into other snakes of any size. The bigger the player's snake is the easier it is to trap other snakes. The goal of the game is to become the biggest player on the map by controlling their snake with the mouse. As one would expect with any competitive online game, players have learned to develop bots to play the game for them.

In our project a current bot that was already created was used as a starting point, and a genetic algorithm was implemented to improve the bot's performance. Multiple generations of snakes were run with each snake in a generation being run in parallel in browsers. The genetic algorithm was run on a server we created allowing new generations to be created with traits from parents of the previous generation. We found that the use of a genetic algorithm can improve agents' performances after several generations.

II. RELATED WORKS

The use of multiple agents at once has been studied frequently. Jennings, Sycara, and Wooldridge (1998) provide a broad overview of how to make a multi-agent system. They explain how researchers can produce autonomous agents and how these can be helpful for study. They show the importance of current discoveries with autonomous agents by explaining the historical context of agent research and development. This paper shows how our improvements of an autonomous agent in a massively multiplayer online video game fits into the current research in the field.

Other researchers have studied using agents in video games. In one study (Hagelbäck, 2009), the researchers propose a multi-agent system that relies on potential fields in a full real-time strategy game online. The bots produced by this research team are able to compete in a competition against humans and perform well. The goal of the game the agents were playing was efficient navigation through varying environments. Instead of giving the agents specific paths they were made to react to their environments. The agents in their system were able to "communicate" to work towards their goal. This is a unique feature of their project. This paper is an interesting approach to a multi-agent system that exhitbits alternative implementations that could be applied to the production of a slither.io bot. This concept of agents assisting each other has been used in the past by some bot developers for slither.io, with the creation of many bots made to feed the main bot.

Others have looked at how multiple agents work against each other. Chen, Pen, and Gu (2016), studied simulations of predator-prey relationships. In their study, the predator was superior in speed. The researchers studied how evaders were captured in their simulation. The researchers mainly focused on three things: collision avoidance of predators, closing down distance between predator and prey during game, and improving the predator's path to prey. The researchers conducted simulations of their multi-pursuer multi-evader pursuit evasion game. This paper shows considerations of avoidance for an agent. This is a similar task for the snakes we generate, avoiding other snakes.

Genetic algorithms and their uses have been studied heavily. Hamblin (2012) explains how these naturally inspired algorithms can be used when looking to optimize characteristics of an agent. This is the idea that we are using as we are focused on optimizing the parameters of a snake. Hamblin's illustrations and explanations of the inner workings of genetic algorithms describe the major steps of genetic algorithms: fitness calculation, selection, crossover, mutation, and replacement. These steps are what we also followed in our project, and we also included elitism, the carrying over of top-scoring population members into the next generation without modifying their characteristics.

Additionally, some researchers have looked at the use of genetic algorithms in activities that are not only natural life simulations. Sholomon, David and Netanyahu (2014) studied the use of a genetic algorithm in a puzzle. The authors propose the use of a genetic algorithm to solve large jigsaw puzzles by applying the concepts of crossover and chromosome representation to the problem. The team used a crossover operator to verify that the pieces will create a geometrically feasible image. This means the edges of each piece are adjacent and matching in color and prospective characteristics discovered by the parents may be inherited by their offspring.

Many studies have looked into the use of artificial intelligence implementations in gaming, but it seems not many others have dealt with the same implementation as in this project. This project is similar to many other genetic algorithm implementations looking at natural life, with the snake requiring to eat and not be eaten, but this project differs in that the agent is in a live video game playing against other bots and humans from around the world.

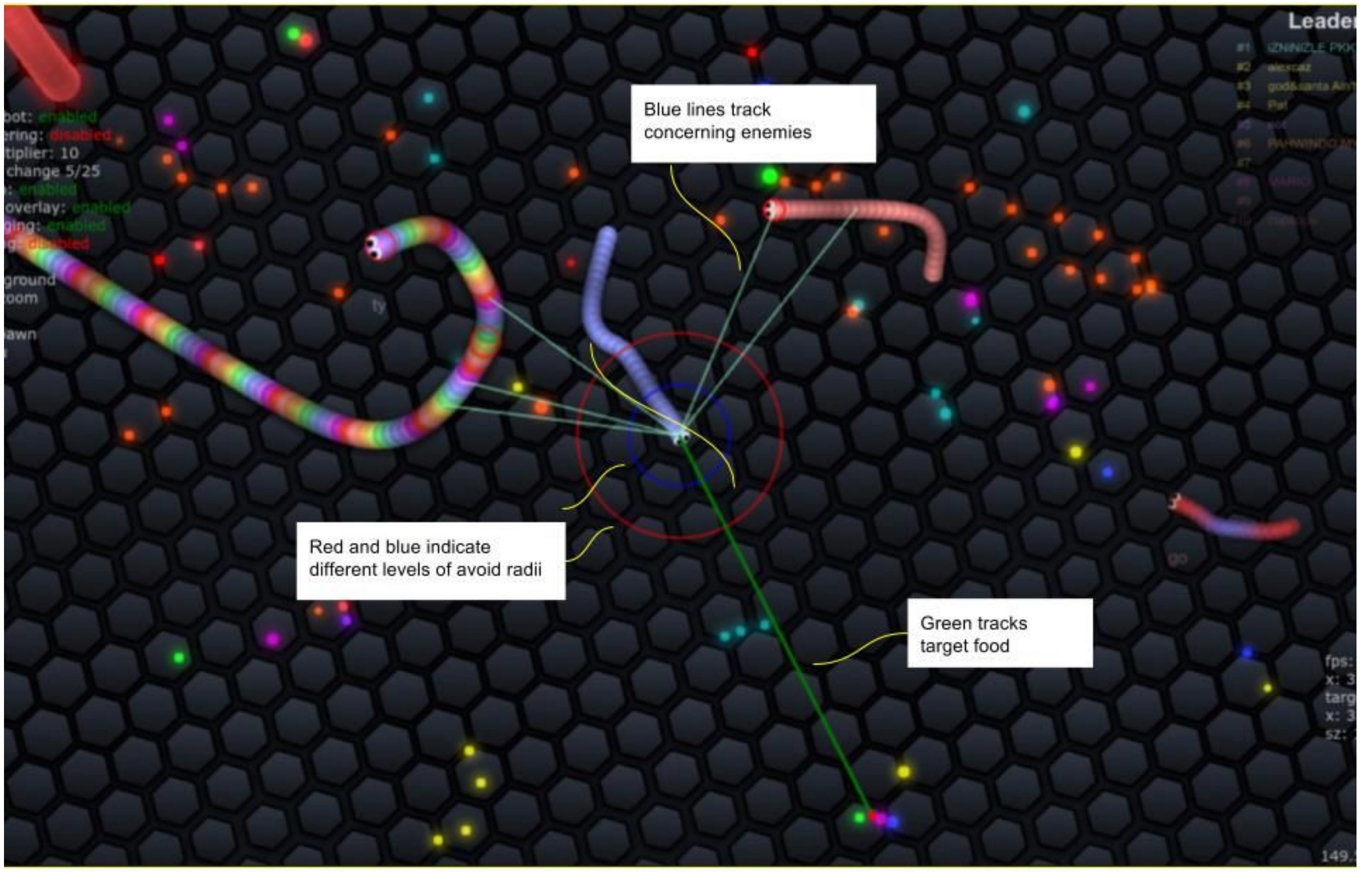
Blue lines track concerning enemies
Red and blue indicate different levels of avoid radii
Green tracks target food

Figure 1. The existing agent works by tracking enemies and by targeting food. The Red circle shown indicates the distance in which the agent will turn to avoid enemies.

III. IMPLEMENTATION

A. Architecture

We used a pre-existing agent known as a ‘bot’ (found at link in works cited) as the foundation for our experiment (See Figure 1). Given the nature of the experiment, we did not need to know the inner mechanics of the bot we were optimizing. The bot relies on a set of parameters to manage its behavior which were of importance in understanding the results of the experiment. These parameters for each agent are stored in an array (See Table 1). The script that runs the bot was modified to be able to accept an agent of a generation from our server and would transmit back data about the agent’s performance over five games. This bot, implemented in JavaScript, was deployed via a chrome extension called Tampermonkey, which allowed the bot to run over the JavaScript of the game website.

Due to the real-time nature of the agents, it would have taken impossibly long to run the agents serially through one browser. Therefore, the server was designed to allow the agents to be run in parallel as shown in Figure 2. When the bot dies and sends its data to the server, the server responds with the next agent to be run unless the server has no more agents for the current generation, meaning it is waiting for one of the bots to complete its run. In that case, the server sends a response to wait and try the request again. The server holds a queue of agents that are dispensed for running. Once the last agent of a generation submits its data, the server writes all of that generation's data to a file. This triggers the genetic algorithm to run in order to compute the next generation to fill the queue.

Attribute	Range
targetFps	30-30
arcSize	0-2*pi
radiusMult	0-50
foodAccelSize	2-100
foodAccelAngle	0-2*pi
foodFrames	0-15
foodRoundSize	1-20
foodRoundAngle	1-20
foodSmallSize	2-30
rearHeadAngle	0-2*pi
rearHeadAngle	0-2*pi
radiusApproachSize	1-40
radiusAvoidSize	5-100

Table 1. The attributes of the snakes that we focused on are included in this table. The most important attributes are *foodAccelSize*, which controls the size of the food the snake is willing to accelerate to get, *foodRoundSize*, which controls what the agent rounds size of the food clusters to, *radiusApproachSize*, which controls the size of a cluster that makes the agent willing to risk approach, and *radiusAvoidSize*, which controls the circle within which the agent will not allow other snakes to be regardless of the agent's goal.

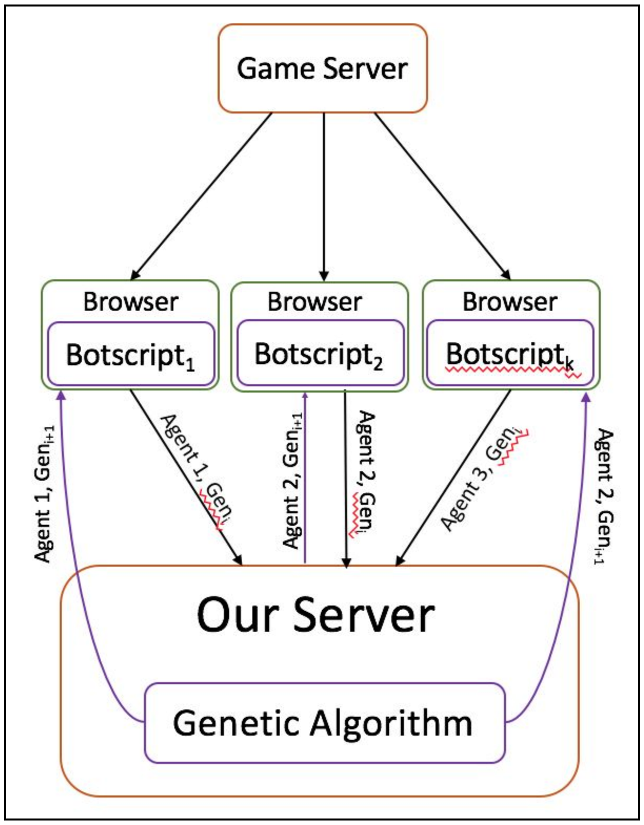

Figure 2. The architecture of our project shows how we rely on the game server, browsers, and our own server.

B. Genetic Algorithm

For this project ten agents were run in each generation. Once the data for all ten agents is in the file, the algorithm starts and completes the process shown in Figure 3 and described below. The first step is to assign a fitness value to each of the completed agents of the generation. This is done by averaging the scores over the five games played by each agent. The overall score for the generation is tracked and is a part of the performance data, and then each agent is given a weight for random selection based on its proportion of the overall score. Next, eight sets of parents are selected for the reproduction step. In the first trial of the experiment, both parents were selected based on random weighted selection, but this resulted in convergence too quickly, as agents were capable of being selected to be both parents. In the final implementation, the first agent was chosen based on random weighted selection, and the second was simply a random choice.

With the parent sets selected, they then go through reproduction. This is done via crossover. With each agent represented as an array, crossover is completed by selecting a random index of the array and copying the first part of one parent and the second part of the other parent to a new array to represent the offspring. All of the new child array's values for the parameters have a chance of mutation. There is a 10% chance of slight mutation for each attribute, and a 10% chance of major mutation. In a slight mutation, the parameter deviates slightly in a random direction, and in a major mutation, an entirely new value is selected from the range of possible values for that parameter. The final step to fill the remaining two spots of the generation is fulfilled by elitism, which is simply selecting the two best-performing agents from the generation and carrying them over without modification.

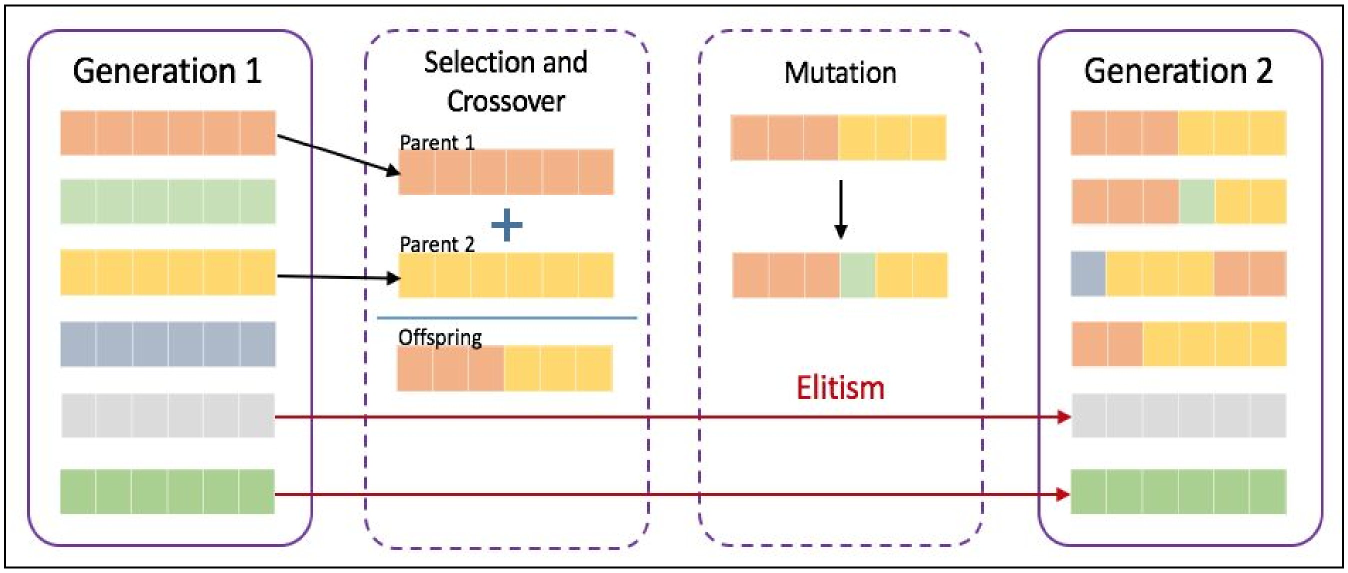

Figure 3. Our implementation of a genetic algorithm accounted for selection, crossover, mutation and elitism. How these processes affect snakes of a generation is shown. Colors represent different values but no particular numerical value.

IV. DISCUSSION

A. Experimental Results

We were capable of gathering 9 generations worth of data in nearly 48 hours. The slowness of the collection was largely due to a couple of behaviors observed in the evolved agents. Agents with a very small *foodAccelSize* would often move around the map very quickly and live a long time without growing very large. This type of agent was weeded out in the first couple generations but delayed the beginning of the experiment. The more troubling behavior that was observed was for snakes with a high level of avoidance. These snakes would tend to grow to a medium size quite slowly by taking no risks, and then they would gravitate toward the fringe of the map. There they would circle themselves and collect no further food until a larger snake happened upon them. This delayed the advance of the generation for hours. Overall we saw some trends of fitness increasing (See Figure 4). Due to mutations though, the fitness of a generation can suddenly decrease.

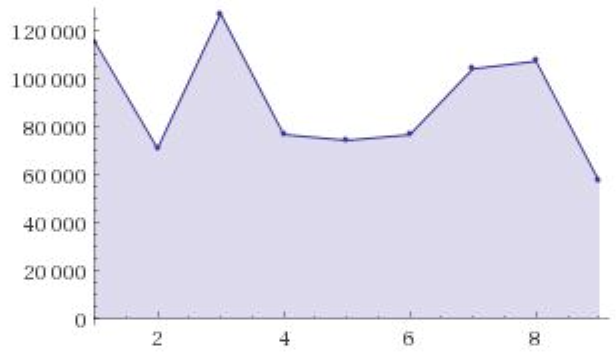

Figure 4. The graph shows the overall fitness of the generations. Nine generations were tested. A steady increase in fitness can be seen in generations four through eight. However, due to mutations there is a dramatic drop in the fitness of the ninth generation.

B. Future Work

In order to fix the problem of having agents take no risks and stagnate, two modifications might be made to further experiments. The first of which is to add a 'starve' feature. This would compute the score gained over a particular timeslice. If the score was not above some threshold, the agent would be terminated. In other words, this would lead to terminating agents that starved, or went too long of periods of time without eating.

The second enhancement would be to reduce the fitness value based on the amount of time taken to achieve the scores. This would be an attempt to penalize snakes who do not play with an acceptable amount of risk. It is possible for a snake to grow slowly on the outskirts of the map, but this behavior does not result in an agent capable of reaching the highest rankings, as to do that requires a snake that competes with other top snakes.

V. CONCLUSION

In this project we sought to improve an agent in Slither.io by using a genetic algorithm. We were able to improve the the performance of bots by optimizing their parameters using the genetic algorithm. Our genetic algorithm was implemented on a private server that collected game information. The server then sent back a generation of new agents to be run in the browser in parallel. The process led to the optimization of agent attributes. There is room for improvement in this implementation especially in terms of agents' lifespans. This project showed that the use of a genetic algorithm in a videogame can lead to an improvement in a bot's performance.

VI. WORKS CITED

Chen, J., Zha, W., Peng, Z., & Gu, D. (2016). Multi-player pursuit–evasion games with one superior evader. *Automatica, 71*, 24-32. doi:10.1016/j.automatica.2016.04.012

GitHub Repository with Functional Bot Used: https://github.com/ErmiyaEskandary/Slither.io-bot

Hagelbäck J, Johansson S J. A multi-agent potential field-based bot for a full RTS game scenario. In *Proc. the 5th Artificial Intelligence for Interactive Digital Entertainment Conference*, Oct. 2009.

Hamblin, S. (2013), On the practical usage of genetic algorithms in ecology and evolution. Methods Ecol Evol, 4: 184–194. doi:10.1111/2041-210X.12000

Jennings, N. R., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems, 1*, 275-306.

Sholomon, D., David, O. E., & Netanyahu, N. S. (2014, July). Genetic algorithm-based solver for very large multiple jigsaw puzzles of unknown dimensions and piece orientation. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (pp. 1191-1198). ACM.