Platforming AI using Genetic Algorithms

Joshua Bridges | 2203030@uad.ac.uk

Introduction

Since as early as 1992, Genetic algorithms (GA) have been used to create and design solutions that humans could not fathom. To this end, GAs excel at this area, specifically because they are a solution to the question, “How can computers be made to do what needs to be done, without being told exactly how to do it?” [1]. This project aims to apply a GA to one specific game domain. The GA was trained on a simple 2D platforming game titled “Block Party”, designed by Joshua Bridges. The goal was to develop artificial intelligence (AI) for bots that any player could compete against. GA was selected as the AI method, due to the unpredictable nature of the results, as well as the simplicity of implementation. However, a combination of the game design, as well as the GA implementation, contributed to significant training times. Consequently, the results used in this project are not from after convergence. They are taken from whatever generation the project completed in the allotted time frame.

Literature Review

GAs have been used to generate computer derived solutions to human problems for a long time now. Jumping from their earliest publications, in the early 1990’s, to the modern era, GAs can now commonly be seen tested on any and all fields of research. More importantly, GAs have become a common approach to creating AI in the game industry. They can be observed everywhere from popular online tutorials, from *Code Bullet,* to massive scale developments like *Quake 3* [2][3]. GA popularity is due to their ease of use and understanding. Massive libraries to assist the creation of GAs allow almost any user to generate AI in their game [4]. Similarly, the process of learning that GA’s use is fundamentally simpler to understand than more complex methods such as Deep Neural Networks.

In games, GAs have been traditionally used to fine tune parameters for a rule-based AI[4][5]. Now, because of GAs, game designers no longer need to tell bots explicitly how to play. The rule-based AI determines the best way to play using a cost function. By optimizing the cost function, the AI can theoretically play the game perfectly.

This project aims to explore GA based AI in a different direction. This project examines GA exclusive AI, thus moving past the previous hybrid rule-based/GA setup.

Implementation

The game application was developed by Joshua Bridges, for prior work, and adjusted to perform with GAs. The GA was implemented using the OpenGA library for C++ in Visual Studio. This library works by implementing GA, using a cost, rather than a score for each gene. Optimization required the GA to minimize the cost function. Other implementations may use maximizing a score function to achieve similar results.

The GA manipulates the game environment with a series of sequential actions. The game expects key inputs from the player to move around the screen. Instead, the GA has a list of these actions mapped to combinations of button inputs. There are actually two such lists in this implementation. The application has two alternating game modes where the action mapping differs. These two modes occur repeatedly, for three rounds. This results in every gene needing two sequences of actions for the corresponding periods for three periods. As a result, six lists represent the sequences of actions the AI improves by minimizing the cost function.

Running the game is the largest time cost of training. OpenGA normally operates with multithreading to accelerate computation. However, the game application was not properly configured for multithreading support. This meant that every generation, every gene had to run the game, sequentially. This process takes time. Nevertheless, it is a necessary process to calculate the cost function for each gene. Cost is how each gene is evaluated and is calculated as follows:

$$cost = distanceToGoal - score$$

After every gene's cost function is evaluated, a new generation is created with a 100% crossover rate. This means that every new gene is a hybrid of two genes from the prior generation. Genes are crossed by randomly selecting actions from either of the two parents. Mutations can also occur at a rate of one in every twenty genes. These mutations occur by flipping a coin to randomly change any of the actions in the sequences. These two features combine and gradually alter the action sequences to minimize the cost function.

Experimental Results

After every generation, the average cost function of the genes was recorded in a text file. The game's need of multiple players necessitated two bots be run, both simultaneously, and against each other. Consequently, results for two bots are generated while training.

The results displayed in figure 1, show results of training over 48 hours continuously. Only 56 generations were completed. This is a small quantity for evaluation but makes sense due to the complexity of the application. Consequently, speed would need to be the next area immediately improved upon for future work.

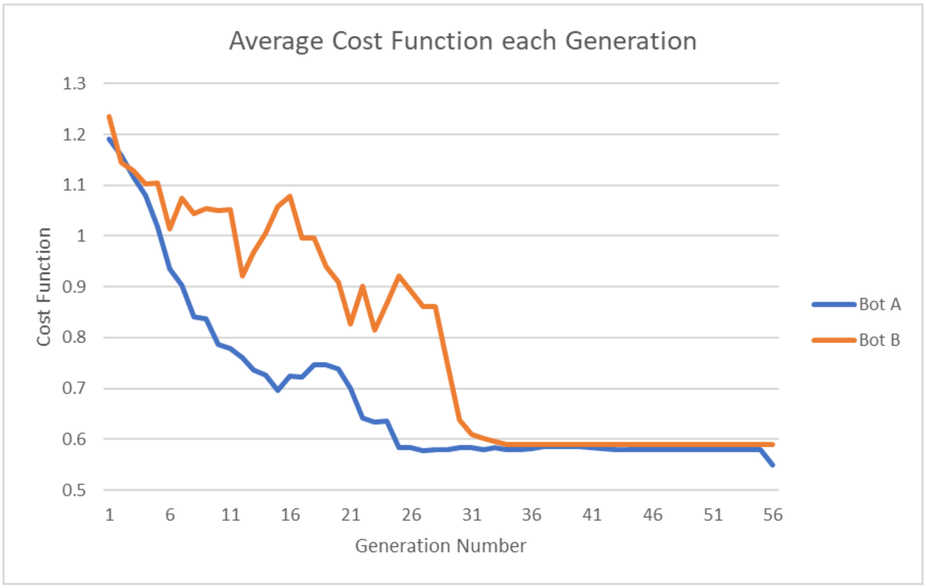

Figure 1: The average of the cost function each generation.

Nevertheless, the results display a substantial improvement over the initial generation within such a short timeframe. Seen in the cost function, the only values that could change are the distance to goal, or the score. The changes observed in this stage of learning are entirely improvements on the distance to goal. Despite the cost decreasing, this direction of improvement has gotten the algorithm stuck in a local minimum.

From generation 31 to generation 56, no significant improvements have occurred to the average cost. The method to escape this local minimum would be for a generation to randomly mutate into a better solution. However, this experiment is not working on a large enough timescale to observe these improvements. The leap from minimizing the distance to scoring multiple times per game, is a substantial change in the selection and sequencing of actions. As a result, these two bots show a large improvement in their ability to move right, across the screen. However, they are far from being able to platform to the finish line.

Conclusions

This algorithm is too computationally expensive to be fully realized in the timescale of this project. That being said, small improvements are already observed in the algorithm. In minimizing the distance between the player and goal, this algorithm is paving the way for more significant improvements to future generations.

The cost function was built on the idea that minimizing one area directly leads to minimizing the other. Scoring would imply that the distance to goal was minimized as best as possible. However, there is a massive jump in complexity between moving as far right as possible and getting on the scoreboard. This is an issue which a hybrid rule-based approach would mitigate. A hybrid system would lower the complexity of the algorithm by replacing a majority of the learning with developer provided macros.

The current algorithm can bridge this gap well. However, the time cost would be exorbitant and prohibitive. A possible solution could be adding the quantity of actions to the cost function. This feature would make the algorithm learn to take as few actions as possible to achieve the lowest cost function. Minimizing the number of actions would, in turn, directly reduce the training time for each generation as the game constantly accelerates in pace. An approach like this often ends with AI making strange leaps in logic. It might not try to win but instead end the game as quickly as possible. This effect could be mitigated by applying a precise weighting scheme to the cost function.

In conclusion, although improvements to the AI are seen almost immediately, GAs are far from completing the game. Improvements to the training time and the cost function may reduce the time it takes to see significant changes. However, the largest obstacle is the requirement for random mutation to generate a significant quantity of changes to the action sequences. The most substantial improvements would result from a change in mutation rate and the corresponding function.

Appendix

1. Koza, J.R., Streeter, M.J. and Keane, M. (2004) "Routine human-competitive machine intelligence by means of genetic programming," *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation VI* [Preprint]. Available at: https://doi.org/10.1117/12.512613.
2. Code Bullet (2018) *Ai learns to play google chrome dinosaur game || can you beat it??*, *YouTube*. YouTube. Available at: https://www.youtube.com/watch?v=sB_IGstiWlc (Accessed: January 9, 2023).
3. Champandard, A.J. (2004) in *Ai game development: Synthetic creatures with learning and reactive behaviors*. Indianapolis, Indiana: New Riders, pp. 467–467.
4. Mohammadi, A. *et al.* (2017) "OpenGA, a C++ genetic algorithm library," *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* [Preprint]. Available at: https://doi.org/10.1109/smc.2017.8122921.
5. Kim, H.-T. and Kim, K.-J. (2013) "Hybrid of rule-based systems using genetic algorithm to improve platform game performance," *Procedia Computer Science*, 24, pp. 114–120. Available at: https://doi.org/10.1016/j.procs.2013.10.033.
6. Cole, N., Louis, S.J. and Miles, C. (2004) "Using a genetic algorithm to tune first-person shooter bots," *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)* [Preprint]. Available at: https://doi.org/10.1109/cec.2004.1330849.