Teaching an AI to Play Video Games

Brian Temple, Gianni Gabriel, Giuseppe Celentano

Abstract

The problem we are trying to solve is how to implement artificial intelligence to play, learn, and perfect playing the video game Snake. Upon successful completion of this problem, we would have developed an AI that is capable of learning the rules of Snake and establishing its own optimized way to play the game effectively. A practical application of this project would be to provide an automated solution for game testing with regards to the game development industry. An automated AI that could test a game to discover any game-breaking issues would highly benefit game developers.

Our approach to build and train the AI was to utilize methods such as deep learning, convolutional neural networks, recurrent neural networks, long short-term memory, reinforcement learning, and neuroevolution. We also researched a few already existing implementations of similar AI as a reference to build a more optimal algorithm. We will evaluate our results by using the following procedures. To quantify our results, we evaluated the effectiveness of our AI by comparing the scores it achieved with our own performance of playing Snake a few times. After playing a couple games, I managed to get my own high score of 24 in the game. Our AI's top score was 52 so far. Given that the AI hasn't received too much training and already surpassed double my high score proves that our AI was a success.

Introduction

In the game development industry, developers are plagued with the difficult task of fully testing their games to find any bugs or game-breaking gameplay mechanics. This is often costly and time-consuming, specifically when attempting to do this manually. There are possibly an infinite number of ways a player could navigate themselves in a game, which makes testing significantly more difficult even with a large team of multiple hundreds of developers. There are many examples of games produced by large studios that suffered from bugs and game-breaking issues. *Fallout 76* (2018), *Anthem* (2019), Cyberpunk 2077 (2020) are all examples of games produced by AAA studios but still suffered from an array of issues, bugs, and broken features. Exploits and bugs are naturally bound to be discovered in a game, especially considering the fact that a released video game is meant to be played by potentially millions of players. It’s vital that a game studio reduces the number of ways players could break their game to allow for maximum enjoyability. Our neuroevolution, reinforcement learning AI algorithm provides a solution that automates the testing process and allows game developers to discover broken features in their game that they otherwise would have never found. Our AI is evidently very competent at playing Snake at an optimal level, thus with a few modifications it seems that the agent could provide insight to game developers of how a skilled player could play their game and possibly discover unforeseen issues as well.

Related Work

There is a research paper titled “Improving Playtesting Coverage via Curiosity Driven Reinforcement Learning Agents”. The researchers of this paper recognized an issue in game development where when one change is made in a certain part of the game, it also causes major problems in a completely different area of the game. These types of issues are incredibly difficult to predict and it’s completely impractical to perform manual tests every time a change is implemented. Their solution was a reinforcement learning algorithm that guides an AI agent through a premade level in a game to find all possible paths to traverse through an entire level. The paths can then be visualized and be analyzed by a game designer to check if any illegal paths were discovered by the AI agent. The approach discussed in this research paper allows an agent to explore 50% of a level in about 1-2 hours, 90% in 28 hours, and 95% after around 50 hours. These times correlate to our AI agent in regards to how long it takes to train, but our approaches differ in some ways as well. For instance, our AI algorithm uses neuroevolution to get better at playing a game, to the point where it can play a specific game, Snake in our case, very optimally. As opposed to this other research paper, the AI agent isn’t reinforced to get better at the game, but is instead reinforced to explore more of a particular game.

Data

There are three main types of data in our project: input data for the AI to understand its environment, the weights used to determine the impact of different environmental variables on the AI’s actions, and the AI’s performance at the game Snake.

The input data consists of the game's coordinate bounds, the location of the snake, and the location of the snake’s food. This data is generated at runtime per frame and is supplied to the AI so that it can compute what actions to take based on the input data in combination with its weights. The input data is discarded when the next frame is drawn. The size of this data is small due to it being discarded after each run.

The weights used by the AI are a series of values used to decide how the data will react based on environmental inputs. The weights are generated each run by duplicating and altering the previously used weights based on statistical probabilities as well as random chance in a way that promotes weights that increase the performance of the AI. Additionally, the quantity of changes made is controlled by a value epsilon, which is constantly decreasing as the number of runs increases. This means that many changes are made to the weights during initial runs and fewer changes are made to the weights in later runs. This is done because the weights early on need to be heavily changed in order to successfully control the snake, thus more changes on early runs allow the algorithm to quickly obtain a set of weights that are decent at playing snake. However, as the number of runs increases, altering the weights more frequently would result in loss of weights that contribute to the AI’s success, thus fewer changes on later runs allow the algorithm to fine-tune the set of weights. After all of the runs have concluded all the weights are stored as an *HDF5* file. We have generated many weights with over around 500+ iterations.

The performance of the AI is measured as the total amount of items eaten by the snake, represented as a whole number. This value is used to determine the success of the weights and subsequently each weight's likelihood to be passed on. The performance of the AI is synthesized after each run concludes and can be saved after all runs have been concluded. We have generated 500+ performance measurements.

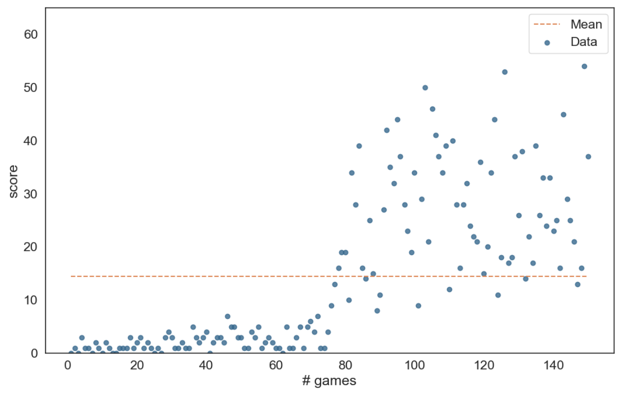
Methods

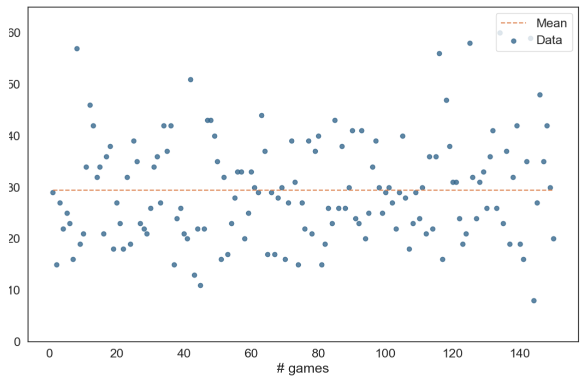
Before we determined an approach to our project, our team had to decide what game to implement the AI into. We ultimately decided on Snake due to its limited number of controls and simple game objective. These aspects of Snake proved it to be a good option since we wanted our AI agent to be put into an environment with a very limited number of variables to ensure a speedy training process. In regards to how we would exactly implement our AI algorithm, we considered a few options. Some approaches we evaluated required training the algorithm using sets of pre-recorded game playthroughs and allowing the AI to analyze these datasets. We avoided these kinds of approaches because we didn't want our AI to adopt any biases from the pre-recorded data. Instead, we were looking for an approach that would allow the AI to establish its own optimized way of playing the game. That is why we ultimately chose to implement our algorithm using a Deep-Q Network. A DQN would allow our AI to start playing and learning immediately instead of trying to feed it playthroughs to learn from. There are various different play styles in Snake with some debate on which is the most optimal way to play the game. So to avoid further complications and arguments we want to remove any sources of bias that could be introduced into the implementation of our algorithm, and this is why we are mostly confident in choosing a Deep-Q Network for our project.

We knew we wanted to use python to write our algorithm since it's what this course has best prepared us for. The various assignments involving python have broadened our knowledge of the language and how to use it. Researching and doing examples of sentiment analysis has also helped us complete our project as it helped us understand the thought process of training a model to predict an outcome, similar to training an AI to become better at playing Snake.

Experiments

The experiments we performed consisted of training the AI using the Deep-Q Network method as mentioned above. The AI plays Snake and the weights and score for each run are recorded. (if viewing MD, look for JPG's in docs folder)





The first chart displays the performance of 150 runs of training the algorithm. The initial performance of the AI is lower because initially the weights have not been trained and the AI is simply moving randomly. However, after about 80 runs the AI discovers a certain combination of weights that allows the AI to navigate to the food. The second chart displays 150 runs of testing and displays the performance of a partially trained AI. It is clear that the performance of the AI increases as the number of runs increases. The mean score of all runs is about 25 and the highest score achieved is 63. When examining the AI controlling the snake, the AI exhibits the ability to navigate to the food, avoiding the walls, avoiding collisions with itself when not trapped, and minor navigation techniques such as “coiling” itself up so as to decrease its surface area, preventing circumstances where it has to cross itself to eat food. One problem it seems to be running into is that of navigating around itself. In later runs, the most frequent cause of collision is when the snake has to cross itself to eat the food. In these interactions the snake seemingly makes a guess as to which direction it will turn, either left or right, resulting in a 50% chance of boxing itself in and colliding with itself. Additionally there seem to be frequent mutations of the weights that result in the snake chasing its own tail and getting caught in an infinite loop.

Conclusion

It’s evident that our AI can competently play Snake at an optimal level. As discussed earlier, our AI beat the top score in our group after playing Snake for a few rounds over two times. Our success can be measured by the fact that with just minimal training the AI significantly surpassed our high score. And with a few modifications, our agent could prove to be a valuable asset for game developers to reveal any unforeseen issues in a particular game.

Furthermore, there is some room for future extensions to help improve the results of this project. For instance, if our AI is meant to be used to test games then technically there is no reason to have the visuals of a game displayed, including unnecessary animations and any other visual components. Removing these aspects during testing would allow the AI to play games at a much faster pace and thus make testing more efficient.