Jake Schinto
Ryan Papazoglou
Matthew Dolce

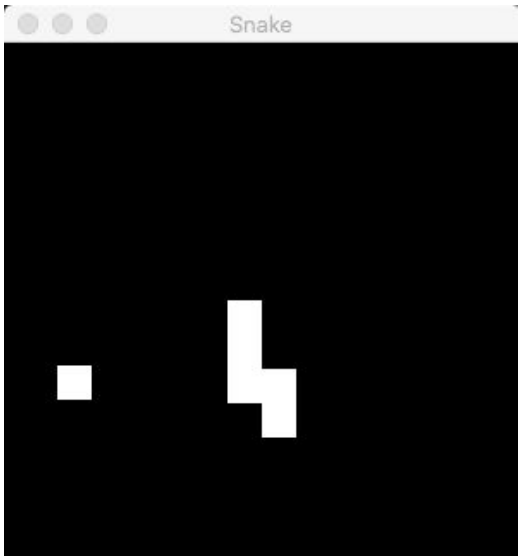
Snake with Deep Q Learning

Deep Q Learning is a method of reinforcement learning that trains an agent to perform a desired task based on certain inputs. We studied a learning algorithm that was training to play the game Snake and attempting to estimate the best move at any given turn. We started with source code that enabled us to run a base agent with no prior knowledge of the game and allowed it to play while learning which moves gave it rewards and which moves were negative in the overall performance of the algorithm. The essence of Deep Q learning is that not only does it evaluate reward of the previous states, it also evaluates the reward, or value, of the current state and decides its next action based on long term reward values.

An agent begins with no knowledge of its current environment and is unaware of what its potential reward is. Through rigorous trial and error, it is able to learn what the correct moves to make are, those that gain a reward, and those that are harmful to the overall algorithm. After a certain period of training time the agent is able to make educated decisions that will benefit its overall reward. It is then able to reinforce itself with the correct learning methods that will enable it to play the game as best as it can.

The initial state of the agent is barren, and it essentially has no knowledge of its surroundings. However, Deep Q Learning is special in the fact that it utilizes reinforcement learning methods to teach itself exactly what it needs to do. Given a desired goal by the user, us, it is able to interpret its surroundings and attempt to acquire as much of a reward as possible. In our Snake example, ‘eating’ the target block is granted with a reward, where as hitting a wall is

given a negative reward. This helped teach our algorithm what to and what not to do. The beauty of Deep Q learning is that it goes beyond the basics of reinforcement learning and is able to learn what action to perform based on its current state, not just the previous one.



When it came to implementing Deep Q Learning, we originally chose to try with the game tetris because the controls and rules of the game seemed simple enough. As we progressed, however, it was soon apparent that learning tetris was a much harder feat than initially predicted. The reward requirement in tetris is just too complex for a machine to randomly stumble upon. The program would have to get lucky enough to clear a line before losing the game. For this reason, tetris seemed to require some knowledge beforehand about its environment or be given a much more complex ruleset for telling when it's close to a reward in order to make it work in a Deep Q way. This is what led us to take on Snake (shown above).

Snake has basic controls, only 4 directions, and a much clearer objective that can be measured with great accuracy. The idea of snake is to maximize the length of the snake (the line) by locating and colliding with the apple (the dot). Each tick of time allows the snake to

move one unit in a chosen direction and have the rest of the snake follow in that same path. The game ends when either the snake collides with a wall or its tail (the trailing dots). These simple rules make Deep Q Learning a clear choice when it comes to teaching the machine to play.

When implementing from our chosen paper, they outlined a carefully designed reward mechanism for generating an immediate reward at each time step. We found a base Deep Q Learning algorithm that learns the game snake from GitHub and planned to add in this more complex reward mechanism from the paper to see how it would affect the performance of the base algorithm. The base algorithm only accounts for a punishment on death and a reward for eating an apple, which is sufficient to learn but not optimal. In the paper, four reward steps are discussed: distance reward, training gap, timeout strategy, and overall reward assignment.

$$\Delta r(L_t, D_t, D_{t+1}) = \log_{L_t} \frac{L_t + D_t}{L_t + D_{t+1}}. \qquad (1)$$

First, the distance reward is a new reward that can be calculated at any state of the game to determine if the snake is on the right track. This reward is originally initialized to zero and then calculated using information about the snakes current state and next state. In our code, we followed the paper's distance reward function (shown above) where Lt is the length of the snake and Dt is the distance between the snake head and apple at time t. The paper mentions that this equation takes on the form of the logarithmic scoring rule, which has the property of returning a negative when Dt+1 is greater than Dt (snake is moving away from the apple) or positive when the other way. This reward function is overwritten however when the snake has claimed an apple because Dt would be zero and our base algorithm gives a reward of 1.

$$M(L_t) = \begin{cases} 6, & L_t \leq k, \\ [p \times L_t + q], & L_t > k, \end{cases} \qquad (2)$$

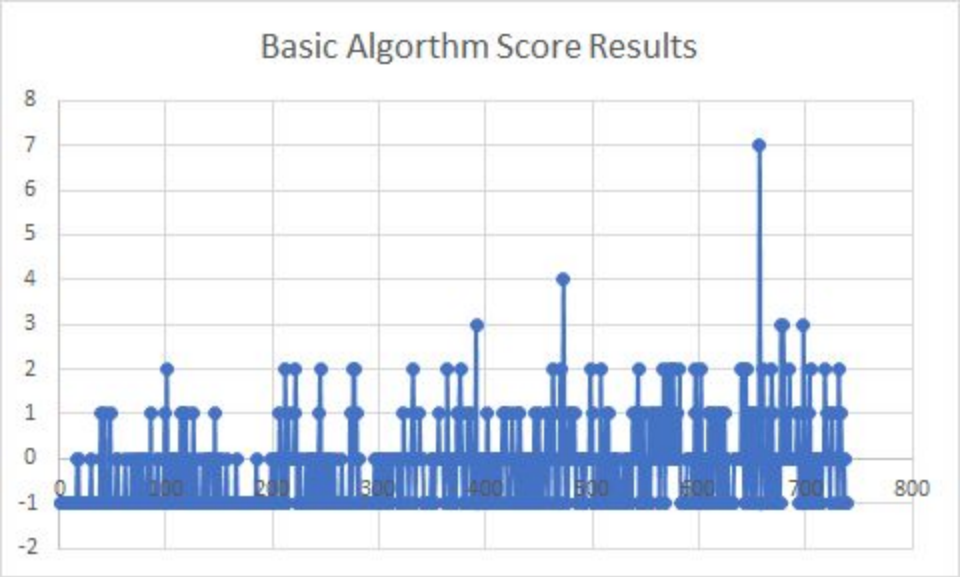
Second, the paper's algorithm called for a training gap to be implemented. This training gap refers to a period of M steps after receiving a reward for an apple where the environment is not suitable to be learned from. This is due to the fact that the snake has just completed its learned method to get the apple and is now faced with finding the new apple, which is spawned randomly on the board. Since the snake has no control over where this apple will spawn it would be unfair to punish it for these M steps while it is reorienting itself toward the new goal. The paper devised a function to find M as shown above. Lt is still the length of the snake and p, q, and k are all constants that satisfy the constraints of (6-q)/p = k so that M(Lt) is continuous at (k,M(k)). Whenever the length is less than k the gap is equal to 6 because this is half the width of the playable space. Otherwise the paper gives a linear function to determine a gap based on the length. The paper chose p, q, and k as 0.4, 2, and 10, respectively.

$$\Delta r(L_t) = -0.5/L_t. \qquad (3)$$

Third, a timeout strategy is necessary to be implemented. This is because sometimes the snake may learn to loop forever in order to prevent itself from dying. The paper calls for a timeout after P steps have occurred where P is ceil(0.7 * Lt) +10. After P steps, the algorithm applies the above function (3) to all rewards from the current game and forces the snake to die. This sufficiently punishes the snake for trying to use looping as a strategy.

Fourth, the overall reward assignment is made. The paper simply aggregates the three previous strategies to come up with a total reward for each step. This overall reward is then clipped to the interval [-1, 1] so that no reward or punishment is too large. Using this assignment and the three other strategies, we can create a single value to train the snake at each step of the game.

Tests were ran on many different variations of the algorithm in order to determine the overall performance of the system and to see which changes were more substantial than others. We first tested the results of a baseline algorithm without any of the systems in place recommended by the paper. This algorithm was capable of reading in the state of the system and learning on those states, but lacked any of the more dynamic scoring systems. This algorithm simply assigned a flat score of 1 for eating the apple, a score of -1 for dying, and a score of 0 for surviving. The results were as follows:



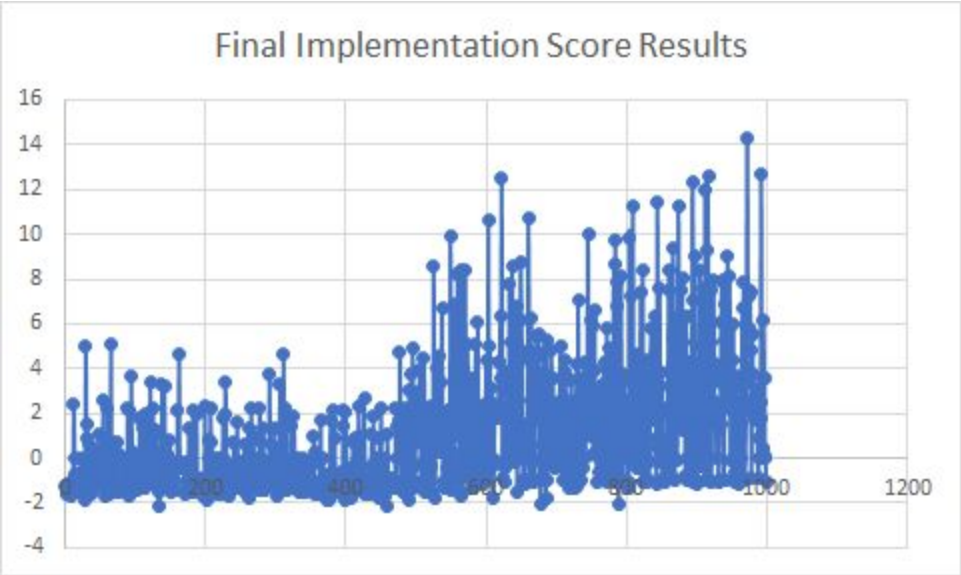
This graph shows the score that the algorithm earned while running test as it was training. As you can see by the results the algorithm was visibly learning but at a very slow pace, with it only able to achieve a score of 2 at a semi-reliable frequency. Overall it can be seen that the performance is very poor, as the rate of dying without collecting even a single apple is still very high. Another notable trend that cannot be seen in this data is that the snake tends to end up in an infinite loop and times out instead of dying. This is because there is no punishment for infinite loops and that infinite looping is better than dying in the system.

This led to our second variation of the algorithm, in which we now punish living with a score of -.01. This was done with intention to discourage infinite looping, as now moving without collecting apples would lead to a massive negative score. This was the only change we made to the system for this test. The results were as follows:

These results show that punishing movement itself is a poor choice when it comes to attempting to train the algorithm. It learned to eventually stop infinite looping, as you can see by the fact that the -5 case, which would result when the algorithm is stuck in an infinite loop, stopped happening as the algorithm trained. Unfortunately the algorithm only learned that death was better than infinite looping and not that scoring was better. This was overall the worse combination of inputs and one that was a very poor solution to the issue of infinite looping.

The final variation of our algorithm is where we implemented the papers recommendations in all of their details. Specifically, we implemented a system to reward surviving based on the direction and distance of the snake to the apple, punished infinite loops by adding a flat negative score to them, and implemented a system to make it so that the algorithm

stops learning after it collects ann apple to prevent the system learning from poor data. The results were as follows:



This algorithms performance was significantly better than the performance of any other algorithm tested. It was capable of achieving a score of up to 11 apples and was regularly able to hit numbers above 6, far exceeding the performance of the baseline algorithm's 2. It also suffered complete failures far less than previous algorithms, where it never collected a single apple. Notably, this algorithm also suffered from an infinite looping situation, but to a far tamer extent than the previous algorithms. These results show that implementing the papers methods for training the snake algorithm significantly improves the performance to an incredible degree. Specifically, rewarding the snake based on distance to the apple was the single largest improvement in the baseline algorithm, as it meant that the snake always received constant updates on its performance rather than only getting an update when it ran into an apple or died.

Matthew Dolce's major contributions to the overall development of the project was the implementation of multiple pieces of functionality to the source code we collected as well as

running the test cases for the code and compiling that data into the graphs in the testing section. He implemented the functionality to allow for the agent to be rewarded based on the distance and direction it was going relevant to the target apple and the functionality to allow for the system to punish infinite loops in a way that did not detrimentally affect the overall learning of the algorithm. Beyond that he ran other test cases to see how the algorithm might perform under certain situations, such as punishing every single movement in which the agent does not collect the apple or testing cases where only some of the systems improvements are implemented.

Jake Schinto's major contribution to the overall project development included testing on our previous game environment (tetris) as well as helping in the implementation and development of our current working code for snake. Most notably he implemented the training gap which is one of the important aspects involved in calculating the correct reward. He also ran the algorithm on his device to experiment with changing certain parameters however these results are not shown above because the computer was not strong enough to iterate through enough test cases to highlight significant trends. Finally, he helped in the final creation of this report by explaining the game snake and about the algorithm that we chose to implement from the paper and how it was to be implemented in our code.

Ryan Papazoglou contributed to the project in an essence that enabled the group to learn further about the overall essence of Deep Q Learning. I helped benefit the group in learning what exactly Deep Q Learning is and how it applied to our project. I heavily researched the topics and different applications of Deep Q Learning. In regards to our project I researched further about Deep Q Learning in respect to teaching an agent to play a game and was able to help my group decipher what attributes to manipulate to see different results in our output. Overall, I was able to

help my group become more familiar with the overall topic of Deep Q Learning and enabled us to complete our project in a smooth manner.

Source (The Paper):
http://www.ntulily.org/wp-content/uploads/conference/Autonomous_Agents_in_Snake_Game_via_Deep_Reinforcement_Learning_accepted.pdf