[NAME OMITTED]
CS6600
Dr. [NAME OMITTED]
12/12/2019

# Final Project - Sword Fighting AI

I really enjoyed working on this project. I didn't have to focus as much on learning a new language or framework since I developed it in the Roblox game engine. I've been developing on Roblox for roughly 12 years now so I was able to set up the project in a day and begin testing it and playing with it over an extended period.

The project was to design a Q-learning AI to sword fight. This was a success! Initially the state included a 30x30 grid world which both AIs would have their position on. In addition, each AI needed to represent their orientation, which I supposed 32 orientations would do fine. The actions available were moving in 8 directions or standing still, and turning left or right or not turning. The issue with this was that there were (30 * 30) positions times (30 * 30) enemy positions times 32 orientations times 32 enemy orientations to learn Q-values for. Since Q-values are per-action, there would be an additional 9 * 3 actions for each state to learn. That results in 22,394,880,000 states to learn!

To reduce the number of states and training time, I only included the distance between the AIs in the state which could be one of 30 magnitudes. The direction to the enemy was also broken down into 8 angles, along with the enemy's orientation into 8 angles. The actions remained the same. This resulted in only 51,840 Q-values to learn! The AI learns these fairly quickly.

I ran into some issues in training as well. First, since I couldn't step up the number of game physics steps per second, I had to introduce additional AIs. I decided that there was no reason all the AIs couldn't learn on the same Q-values at the same time. In the video I've included, there are 100 pairs of bots training so 200 bots using the same Q-values. This introduced a new issue. The bots would at times run away from each other and attack other pairs of bots this would cause bots to be unexpectedly rewarded for being far away from their opponent. I solved this in two ways. First, I introduced a hard boundary to keep bots in their general area. Second, I introduced a large cost if AI ever separate more than 30 studs and when they do I teleport them back to their starting positions. Finally, if bots are more than 15 studs away from each other, I subtract the (distance - 15)^2 from the reward to bring them back together. To ensure that bots were not waiting for long periods for an opportunity to strike, I introduced a constant -1 cost for each action.

As for the main rewards, each bot starts with 100 health points and are rewarded with the amount of damage they do to their opponent or punished by the number of health points they themselves lose. For a victory (opponent reaching 0 health) there is an additional 100 reward and the opposite for a loss.

To debug the implementation (I ran into some angle state issues) I developed a GUI shown in the later half of the video demonstration that shows the maximum Q-value for a magnitude-direction pair. There is another state variable, enemy orientation, so I find the maximum of all those states and display that. The angles on the left are actually forward and the right direction is clockwise. The results are as expected, when the enemy is in front of the AI there is a positive reward. The reward is also much higher on the front right than the front left since the front right is where the sword is. The debug GUI doesn't however show that there is a negative cost behind the AI. I assume this is because the AI has learned to avoid that position so there isn't much exploring going on down there and the GUI shows the maximum state so the AI would have to have seen the AI in every orientation and all have ended badly for the AI. This doesn't make sense because if both AIs are facing away, neither can damage each so the lowest the states can go is -1 for taking an action. Since there are rewarding actions nearby, the maximum of these states is likely to increase if trained longer.

Since this code only runs in Roblox Studio, I've included the Roblox place file in my submission and a video in case you do not have Roblox installed. Once the place is opened in Roblox, you can simply press play and the AI will begin training. In the combat script (opened by double clicking it) you can change the showDebugUI variable to true to enable the circular GUI showing the maximum Q-values.

The result after a couple hours of training is hundreds of very deadly AIs. I plan to continue building on this project and add player controls similar to the way the AI operates and allow players to combat the AI. I may also look into adding more controls like slash attacks with the Q-values initialized by the Q-values without slashing. To make a truly interesting game though I would need to add parameters for global position on an interesting map, jump actions, and falling states. The issue that I see with these things is that Q-learning doesn't capture the patterns - falling near an enemy is always going to be a bad idea no matter where you are at globally. Carefully adding states in the correct order and initializing them properly may give a good starting point for training, but isn't the best method. Really, there needs to be an underlying mechanism that can tell what pieces of the state are important contributors to reward. For this reason, I may end up switching to a neural network to estimate Q-values instead of explicitly learning every possible combination.