**Automatic Character (for Pathfinder)**

Xavier Humberg & Nate Rogers

Final Project Report | CS5650

**Problem definition and motivation – what problem did you choose? Why is it important or interesting? Why did you use machine learning techniques to solve it?**

The program we made for this project was an automatic fighter class character for the tabletop roleplaying game Pathfinder. We decided to make an automatic character to fill a role for times there are not enough players to go through a campaign. The idea is that the automatic character would be able to fill a role for a missing player. We decided to start with a fighter class for a few different reasons. The main reason is that there are fewer players that play the fighter class role and would more likely be in need of filling. Another reason is that the fighter class is a little more straight forward with actions to take each turn. The fighter class, in Pathfinder, serves as a kind of tank role, and should be mostly focused on getting up to enemies and attacking in order to keep the enemies attention off the other characters. To this end, our program keeps track of how many enemies there are, how big they are, the distance to the closest enemy and the biggest enemy, how much health the character currently has, the highest amount of damage an enemy has done, and the hit rates for both the biggest enemy and the character. This would also serve as a good starting point for future implementations of other character classes as they would also need this information plus some additional information important for each specific class.

Pathfinder is a tabletop role playing game and is really only limited by the imagination of the dungeon master. We wanted to use machine learning techniques here instead of traditional techniques so that the program can be presented with any scenario and make a prediction for the best course of action. It would be difficult to use traditional techniques for this, as we would need to program each exact situation. With nearly infinite possibilities within even a single campaign, we needed a program that could learn what actions to take with just the information available to players on any given turn. This would allow the program to be used whenever and wherever needed.

**Your solution – the details of the machine learning models/algorithms you chose/developed (or proofs for theoretical projects)**

Our solution involves two layers. The first layer uses decision trees to narrow all possible actions down to the three actions that make the most sense with the available information. For example, we don’t want the program to try to attack when the enemy is too far to hit. For this layer, we use three decision trees, one for each of the three possible actions. We trained each tree a little differently in order to get three different actions. The first tree was trained as the most aggressive choice, with most of it’s actions being to charge, or to do a full attack. The second tree was trained with more strategic choices, like flanking, fighting defensively, or just defending if health is low. The third tree was trained with more passive actions, like step away and heal, or even retreating if needed. The three choices would then be fed to the second layer of our program, which would then make a prediction of which would be the best action to take in each scenario.

**Experimental evaluation**

**Future plan**