Project 2 CS - 370

Young children frequently love to complete mazes. Starting with a pencil, they trace through the path until hitting a wall or a block. Out comes an eraser, working backwards to the last fork in the maze before proceeding down another path. Larger mazes might have several eraser marks at numerous forks.

Instead of using a pencil or trying to figure it out ourselves, we can now use an AI to help us complete a puzzle. We set up a framework that consisted of an environment and an agent which acts in this environment. We have a blocked square, a treasure square, and a free square. This will be represented by a matrix that has 1s, and 0s. Our agent, a pirate, has only one goal in life and that is to get to the treasure by only using free squares and not going outside the maze. With this setup, we will reward the pirate for every move they make. These rewards are anywhere from 1 to -1. If the pirate steps outside the boundary of the maze they will receive a -1 which will cause the pirate to start over. If the pirate visits a blocked square, they will receive a -.75 or, if the pirate moves to a square they have already occupied, they receive -.25. Lastly, if the pirate moves to a square that is both free and new, they receive a -.04. If the pirate’s rewards equal or fall below -1, then the game is over. If they get to the treasure they win! We really want our pirate to take the shortest path with the best reward. Through this, our pirate will learn how to navigate the maze to get the treasure.

Both the typical child’s maze completion methods and the AI framework result in a successfully completed puzzle. However, if we continued using larger and more complex mazes than in this project, it would become more difficult for the human to find their way to the end. Using AI with this framework means we can place the pirate anywhere on the maze and he will always find the way out with an adjusted reward. Both methods, the child’s and the AI, use a type of reward system. While the pirates is getting a numerical value of some type, the human has to erase what they learned that was not useful but keep what was useful. Seeing their progress can be very rewarding. That being said, in a larger puzzle, the AI would most likely beat the human.

When we look at how our agent, the pirate, explored the area, we notice he used two different methods. One method was exploration. This allowed the agent to search the whole sample space based on the rules we already discussed. We set this to be done about ten percent of the time, also using the epsilon factor within the code. The other ninety percent of the time, the agent used exploitation. This method uses a different approach by exploiting the promising areas found when you did the exploration. These moves used experiences to understand what would help our agent get the best reward.

It is possible to change the percentage of exploration versus exploitation in our code. By doing this, we can allow our agent to explore more by increasing the epsilon value in the code. This increases the times our agent will search the whole sample space. If we decrease it to zero, we don’t give our agent enough feedback to help it make choices in the exploitation. We used the rewards gathered from the movements to provide reinforcement learning to our agent. While moving through the maze, they gather negative or positive rewards based on their movement, as mentioned earlier. We use these rewards to train how the agent should move in exploitation. This helps the agent get from the start to the finish in the least amount of negative rewards as possible.

This feedback is necessary as it helps build our Q-learning process. We will need to generate training samples from using the neural network by simulating examples. The neural network is trained after each maze example by injecting a random selection of the most recent training samples to the neural network. We also assume that in each maze example we are getting better and better. Because of this, we will forget older examples and continue to learn on with the newer examples. After each game move, the code will generate an episode and save it to memory. Each of these episodes contains a few inputs. The first input is the environment state, storing where on the maze the pirate is located. Next, we store the action the pirate makes, whether it is left, up, down, or right. The third input we store is the reward from the action followed by the new state the pirate is in. Lastly, we store the final result of the game, either the pirate won or the pirate lost. This data will be saved and used to train the neural network.