Brian Engel

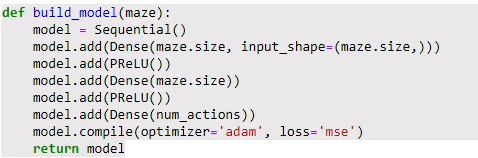
CS-370 Project 2

The very first path search algorithms students typically learn are depth-first search (DFS) and breadth-first search (BFS). Basically, the DFS rule is “always take the right-most path which you haven’t already explored”. Breadth-first search is like a plant: every time BFS hits an intersection, it splits and goes both ways. A human solving a maze usually tries to work their way closer to the end. If we’re not sure which way to go, we take the direction which points more toward the goal. Formalizing this approach leads to heuristic search algorithms (Johnswentworth 2018). These all use cognitive and spatial understanding and trial and error.

The algorithm that I used to solve the maze starts at a random location on the grid. It then uses the epsilon value to decide whether this turn is going to be a random action, or if it is going to be greedy and take the action with the most potential value. Once it makes this move, it determines whether a value for the move depending on if it is an available space, the goal, out of bounds, or blocked. Then it stores the experience (state, action, reward, next state) in the replay memory. If the move didn’t result in an invalid or game ending move, it repeats the previous steps. Once a move is invalid or game ending, it trains the neural network using experiences from the replay memory. It seems that the approaches are fairly similar between humans and machine learning. They both learn through experience and trial and error.

Exploitation in pathfinding is where the algorithm chooses the path with the highest known rewards. Exploration is when the algorithm chooses the direction at random to find out the possible rewards for a possibly unknown path. The primary purpose of exploration is to gather additional knowledge, discover more about the environment, and potentially find better long-term strategies. I think that the proportion that was set in this problem is probably pretty close to the ideal proportion, which is 90% exploitation and 10% exploration. The reason for this is if every square had a higher chance of picking a direction at random, It would be really hard to have the agent move in the right direction over a long time. There is only a 35% to stay on course in a move of 10 squares that the exploration is already done on, since it has a 10% chance to wonder off on any of those squares. One thing that might actually help this problem even more is to decay the exploration after a while, since there is a limited number of squares on the grid. This way it stops exploring as much and tries to get to the end more often. In doing the exploration, the pirate learns to navigate around the world, and will eventually find the treasure, provided that a route to the treasure actually exists. Another small addition in the algorithm is that it uses a penalty for every move. I thought this was pretty smart since it makes the pirate be more efficient and not wonder around the board.

To implement this as a deep Q-learning algorithm there are a couple of things you have to do to distinguish it from a normal Q-learning algorithm. The build\_model function defines the architecture of the neural network used for Q-value approximation and has several layers.



The GameExperience class is used to implement experience replay, which helps with stability in the training. In the training loop, the agent explores the environment, and then that is stored in GameExperience as experience replay. Then random samples of the experience replay are pulled to train the neural network and figure out the Q value. This is what the agent uses to determine what move it is going to make when it is exploiting the environment. For exploration it is truly just a random direction.

Johnswentworth. (2018, June 19). Problem Solving with Mazes and Crayon. LESSWRONG. <https://www.lesswrong.com/posts/CPBmbgYZpsGqkiz2R/problem-solving-with-mazes-and-crayon>