# CS370 Project Two

Chuke Rupert

CS-370

23 June 2024

Humans and computers (or intelligent agents) tend to approach problems in fairly distinct ways, though similarities between the approaches is not in short supply. Machines leverage a tool called a neural network to interpret an environment or problem and produce a solution/action which is, at its core, designed to mimic the functioning of the human brain. Humans too receive input through our senses which trigger a series of interconnected neurons to help us process and identify an environment or problem and come up with a solution in an outwardly similar fashion. Where this process begins to differ is typically in what logic agents use to arrive at a conclusion, which often appears vague and untraceable to humans despite the resulting efficacy (*But what is a neural network* 2017).

With respect to the game at issue, humans and machines will likely approach the problem very differently. Foremost among the differences is how the two groups perceive the game board. Humans will assess at a glance (and based on the rules of the game) that they can’t move toward an immediately proximate obstacle or board edge and simply choose not to. It is also fairly common practice with mazes and pathfinding problems for humans to start from the objective and reverse engineer functional paths despite the logic of the practice not withstanding critical scrutiny. Lastly, humans will view the task as having only two outcomes, either pass or fail, mapping to whether or not they reached the objective from the starting location or not.

Machine agents on the other hand don’t perceive the game in quite the same way. Firstly, while obstacles are declared and visible to the machine, it is more accurate to say that agents only know they shouldn’t move toward them or that it is less efficient. The agent may still try depending on exploration settings and will simply be “punished” more harshly for doing so, especially during training, so the agent doesn’t “know” an action can’t be taken until either it tries or it samples a set of valid actions explicitly. Secondly, and informing further on the previous example, agents do not only perceive the game as a pass/fail paradigm, but rather as an optimization problem on a continuous spectrum. Win or lose, or pass and fail, are just conditions which indicate to the agent that the game is over. The neural network’s goal in this game is to achieve the highest score possible according to a reward system established for its training, which is done by reaching an ending condition in the fewest number of actions. We can infer from this that the agent will want to win while running into the fewest obstacles and without using unnecessary actions, as all three of these represent inefficiencies in its reward system. Notice, however, that this last description aligns exactly with how a human will best play the game as-well, minus the justifications for the restrictions.

Let’s explore even a little further just how the agent does this. The AI in this game uses a Deep Q-learning algorithm to train its network. An essential element in effectively training a network this way is that it must be allowed to both explore new paths and leverage what it already knows to be “good” decision-making (Beyslow 2019). These two options are called exploration and exploitation, exploring new or random options and exploiting the current Q-table. This Q-table or Q-matrix is the object that the agent’s neural network is trying to directly optimize, and is a representation of the scores the agent could expect to receive for taking a given action. When choosing to exploit the network, the agent examines the vector within the Q-table that represents the current state or position the agent or pirate is in, and chooses the action from that vector that has the highest likely score. Exploration in this game is done by simply looking at the set of valid actions for the agent to take given its current position and selects one at random.

The agent chooses when to explore or exploit uniformly according to a ratio typically denoted as “epsilon”. For this model, I decided to update the epsilon value during training such that the model would explore 35% of the time (“epsilon = 0.35”) initially. However, a model that maintains a constant epsilon will become unstable and inefficient as training goes on as the model continues to choose random actions despite already having a pretty good idea what the best action will be. This is usually addressed by including an epsilon-decay factor in the model which reduces the exploration rate as the model becomes more efficient. For this model, I set epsilon to decay according to the model’s win rate over it’s most recent 32 rounds. This value can continue to be used in production as well as a way to adjust the difficulty of the game for users. Those that wish to compete against the most efficient bot will have epsilon set to near 0 so that it very rarely selects a sub-optimal path.

References

Beysolow, T. (2019). *Applied Reinforcement Learning with python: With Openai Gym, tensorflow and keras*. Apress.

YouTube. (2017, October 5). *But what is a neural network? | Chapter 1, Deep Learning*. YouTube. https://www.youtube.com/watch?v=aircAruvnKk