Name: Timothy Johnson  
Date: June 18th, 2025  
Professor Balogun  
CS 370

Project Two: Design Defense

If I were trying to solve the treasure maze myself, I’d look at the whole map, try to plan out the easiest route, and remember which paths I’ve already tried. I’d use logic and memory to avoid dead ends. Which reminds me of a video game RuneScape, where I would learn the most efficient route by avoiding danger to reach my objective. Humans use common sense, past experiences, and even shortcuts based on how things "look."

The pirate agent I built doesn’t think like that. It doesn’t know anything about the maze at first. It learns by trial and error using reinforcement learning, trying moves, seeing what works, and updating its choices based on rewards. This is way more repetitive than how a person would solve the same problem. The cool part is that, after a lot of training, the agent finds the best path every time, without needing to "think" or hesitate (Valohai, n.d.). So, in short: I’d solve the maze by planning and adjusting, while the agent solves it by practicing over and over. I might get bored or make a mistake, but once the agent is trained, it doesn’t.

The main goal of my pirate agent is to beat the player to the treasure by finding the fastest path through the maze. It starts off randomly moving around but eventually figures out the smartest route. It learns this by balancing exploration (trying new paths) and exploitation (choosing the best-known path so far) (GeeksforGeeks, 2025).

At the beginning, I set epsilon (exploration rate) high, so the agent tries everything. Then it slowly decreases, letting the agent rely more on what it’s learned. When it hit a 100%-win rate around epoch 48, I knew it had basically mastered the maze.

Using deep Q-learning is smart here because it learns from scratch, there’s no need to hand-code rules or label data. That makes it flexible and powerful for games like this. I used deep Q-learning with a neural network to train the pirate agent. The network predicts Q-values, which tell the agent how good each move is in each state. It stores experiences like [state, action, reward, next state, done] and replays them during training to improve accuracy.

Once trained, the agent could reach the treasure consistently from any starting point. Using deep learning here was a good choice because the maze has a lot of possible states. A basic Q-table would’ve been too slow or too big. The neural network helped generalize decisions and train faster. Once trained, the agent didn’t just memorize paths, it understood how to handle any situation in the maze (TensorFlow, n.d.).

This project helped me understand how intelligent agents learn. I didn’t just follow steps; I saw the trial-and-error process in action. Watching my agent go from clueless to perfect was rewarding. I learned a lot about how machines can figure things out on their own with the right setup, which is different from the way I usually solve problems.

Resources

GeeksforGeeks. (2025, June 4). Deep Q-learning in reinforcement learning. <https://www.geeksforgeeks.org/deep-q-learning/>

TensorFlow. (n.d.). Playing Cartpole with the actor-critic method. <https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic>

Valohai. (n.d.). Reinforcement learning tutorial part 3: Basic deep Q-learning. <https://valohai.com/blog/reinforcement-learning-tutorial-basic-deep-q-learning/>