**7-3 Project Two Defense**

Mohamed Saleh

Southern New Hampshire University

CS 370: Current/Emerging Trends in CS

Prof. Olga Mill

April 20,2025

Treasure hunt game is one of the most popular problems in the field of computer science. If a human begins to resolve the problem, he needs to keep exploring the valid cells/nodes to reach the goal node. However, that’s doesn’t guarantee that valid path is the shortest unless we mark used routes until we come to the shortest path. In computer science it’s a common path finding problem which can be solved using multiple algorithms like Dijkstra, A-Start, Bellman-Ford and BFS. Each algorithm works perfectly with different problem parameters, some are deterministic and some are heuristic. Furthermore, using advanced optimizing methods like Dynamic Programming and backtracking can help to resolve the problem.

Using reinforcement learning to apply Q-Learning allows the agent to act optimally in an environment. While the agent primarily learns by experience through exploitation, often, the agent can choose to explore the environment to find previously undiscovered paths. This is called "exploration" and is defined by epsilon. This value is typically a lower value such as 0.1, which means for every ten attempts, the agent will attempt to learn by experience nine times and will randomly explore a new path one time. The similarity between humans and the AI agent is both must explore the environment and learn based on that if the route going to lead to the goal.

In our problem it is important to maintain a ratio (Epsilon) between exploration and exploitation. Usually, we use exploration to try alternative ways and using exploitation to get the best optimal value or outcome as in greed approach. If we rely only on exploitation to get the best outcome, that might lead the agent to halt or stuck looking for optimal path, while relying only on exploration would waste time. Therefore, we need epsilon to switch between the two approaches.

In reinforcement learning works perfectly to find the path; using Q-table that estimates the value of taking a certain action from a certain state. Using rewards and penalties will guide the agent through navigating the map and keeping updating the policy table with the best expected outcomes.

Algorithms are typically used to solve a class of specific problems or to perform a computation. Algorithms are used as specifications for performing calculations and data processing. The implementation of Q-learning using neural networks for the game has multiple building blocks. First, we setup the environment, states and Actions. Second, create a sequential model (agent) which basically has one input (action) and one output (expected value) contains multiple layers. Third, create a dense network with grid size (columns\*rows), using PReLU as an activation function, new layer, activation function and finally an out layer with number of actions probabilities.

The second stage of the algorithm is the training process. We need to address the importance of epsilon value which determine two things as follow: success rate of the model, switch between exploration and exploitation approaches. Here is the psueocode of the algorithm

For each epoch:

Agent\_cell = randomly select a free cell

Reset the maze with agent set to above position

envstate = Environment.current\_state

While state is not game over:

previous\_envstate = envstate

Action = randomly choose action (left, right, up, down) either by exploration or by exploitation

envstate, reward, game\_status = qmaze.act(action)

episode = [previous\_envstate, action, reward, envstate, game\_status]

Store episode in Experience replay object

Train neural network model and evaluate loss

GameExperience.get\_data to retrieve training data (input and target) and pass to model.fit

call model.evaluate to determine loss.

If the win rate is above the threshold and your model passes the completion check, that would be your epoch.

**References**

*Dijkstra’s algorithm*. Wikipedia. <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>

*A\* search algorithm*. Wikipedia. [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

*Bellman–Ford algorithm*. Wikipedia. <https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm>

GeeksforGeeks. (2025a, February 25). *QLearning in reinforcement learning*. GeeksforGeeks. <https://www.geeksforgeeks.org/q-learning-in-python/>