Analyze the differences between human and machine approaches to solving problems.

1. **Describe the steps a human being would take to solve this maze.**
    a. A human would try to find the best navigation sequence that enables them find the treasure and maximize their reward. The human needs extra time after reading the instructions to analyze their strategy.
    b. The brute force approach for a human would be to look at the graph from the big picture. After placing a pencil in the starting cell, a human would attempt to draw the shortest path to the treasure that avoids occupied cells. This would be accomplished by using visual reference points to aid in solving the game. Additionally, the human may have to backtrack, do some math, write down the sum/difference, and then proceed.
    c. The game becomes more difficult when a human has to start the game from a random cell. A human can develop an algorithm that would conduct a BFS (exploitation) or a DFS (exploration) of the graph to find the treasure. Dijkstra's Algorithm "works harder but is guaranteed to find a shortest path" (Amit, 2023). The weights would have to be converted to non-negative values in a data set that contains the nodes and edges.
    d. Declare an adjacency list assigned to a collection list dictionary. Iterate over the edges to store the node, neighbor, and weights in the adjacency list. Declare a list variable (distance) with elements set to the max size to track distance from the current cell. Set distance at starting location to zero. A list will keep track of each cell visited. Lastly, a min heap is declared with the start location and weight.
    e. While there is something in the min heap, repeat the following. Two variables (cost, cell) will be assigned to the min value popped from the heap. A conditional statement will

check if the cell has been visited or if the current cell is the treasure cell. If false, then the cell is added to the visited list and we iterate over the neighbors of that cell.

   i. A conditional statement checks if the neighbor has been visited and check the distance list to see if the current cell value is greater than the node distance value plus the neighbor cost.
   ii. If true then the distance at index neighbor is assigned to the distance at index node plus the neighbor cost. A list that contains the weight and neighbor is pushed into the min heap.

f. When the while loop exits, the distance list will contain the shortest path.

2. **Describe the steps your intelligent agent is taking to solve this pathfinding problem.**
   a. The intelligent agent considers the graph as an 8 x 8 array. Each cell has a row and column value. A variable is declared and assigned to an object instance of Treasure Maze passed with the array. Building the neuro network required a member method call passed with the array. Both instances are passed in a member method call for training the model.
   b. Two hyperparameters are needed (epsilon, number of epochs). Epsilon will be used for the epsilon greedy algorithms in determining if the intelligent agent will explore or exploit from previous episodes. Number of epochs is used to glean insight into how many training episodes it took to train the model.
   c. An Experience Replay object instance is instantiated with the model and max memory size passed as arguments. The replay object list of episodes cannot exceed the max memory. The oldest episode is deleted when the maximum capacity is reached.

d. My code produced different total number of epochs. It returned 900 epochs on the low end and 1400+ on the top end. A safe estimate is chosen for the number of training episodes. A for loop is used to iterate over the estimated number of training episodes.

   i. At the beginning of the training episodes, a variable declared and assigned to a random choice from an array that contains a list of free cells. An object instance of Treasure Maze calls a member method to reset the maze with a pirate in the specified cell. Another variable is declared and assigned to the current state.

e. While the Treasure Maze object instance return value from the game status member method is equal to 'not over', the following will be repeated.

   i. The current state is saved and a conditional statement is used to determine if the intelligent agent will explore or exploit based on our epsilon value. If a random number is less than epsilon, then the intelligent agent will randomly choose a valid action. If the random number is greater, then the intelligent agent will predict the action from experience.

   ii. The Treasure Maze object instance makes a member method call with the chosen action and assigns the attributes to three variables (state, reward, game status). A list is declared and assigned with the three variables plus the previous state. The episode (list) is passed to the experience replay object for sampling during a later training runs.

   iii. We retrieve the training data from a Game Experience member method call. Then, the model is trained on the inputs and targets returned from the replay object.

f. The while loop will exit when the game status does not equal not over. This implies that upon completion of the while loop, the intelligent agent either won or lost. The training loops (epochs) increases until the agent achieves a 100%-win rate and pass a completion check.

g. The intelligent agent starts in random cells and primarily learns from exploitation. It is penalized for revisiting a cell, visiting an occupied cell, or moving outside the graph. Ultimately, the agent will try to find the shortest path to the treasure that maximizes the total reward.

3. **What are the similarities and differences between these two approaches?**

   a. **Similarities**: Both involve trying to find the shortest path that maximizes the reward. In the beginning of the game, both approaches require some trial and error. The movements and action will appear to be random but over time converge to a central limit. The two approaches involve remembering visited cells to prevent wondering.

   b. The intelligent agent and human start at the same place and choose from the same valid moves. The total reward will be considered at each cell before making an action. Both agents have to predict the best course to take based on previous experiences stored in memory. Movement from one cell to the next will be based on the lowest penalty or greatest reward. Thus, both agents are motivated to move from top left to bottom right.

   c. **Differences**: A human being may be able to determine the shortest path by viewing the graph. A neuro network algorithm needs the dataset in machine code or readable format (digitized). The human will use a pencil and paper or calculator to track the rewards or penalties associated with movement.

   d. An algorithm will have built-in math modules that can calculate to total sum of discount rewards for an action and store the reward associated with each action/episode/epoch when it updates using a Bellman equation.

e. A human being does not have an experience replay object to exploit previous experiences during learning. Memories have to move from short term to long term memory. A machine learning algorithm can implement Dynamic Programming to store the shortest path associated with a cell.

f. The human pseudocode algorithm does not store a path to the treasure from each cell. The algorithm would have to be ran 64 times from each cell in the graph before it would pass the completion check. The intelligent agent learns the shortest path from every cell during training.

g. Lastly, a human being's approach involves abstract conceptualization and concrete experimentation. A machine learning approach will be statistical and computational. Amit from Stanford stated that the "best results are archived by using both pathfinding and movement algorithms" (Amit, 2023).

Assess the purpose of the intelligent agent in pathfinding.

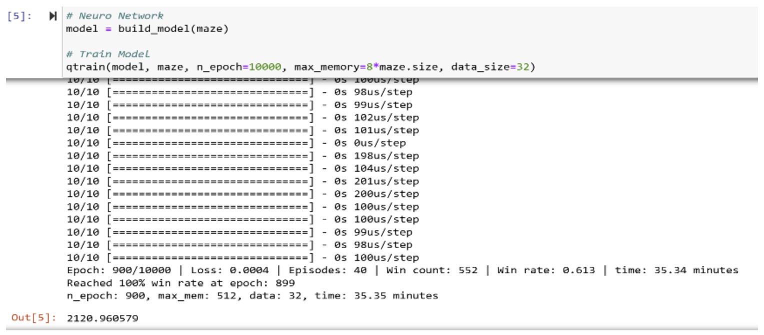
1. **What is the difference between exploitation and exploration? What is the ideal proportion of exploitation and exploration for this pathfinding problem? Explain your reasoning.**

   a. **Exploitation**: Considered the 'greedy' aspect in the epsilon-greedy algorithm (covered below). Our agent chooses the best action relative to an action-value estimate associated with each neighbor. Movement is predicted based on previous experience. The greedy aspect "may not actually get the most reward and lead to sub-optimal behavior" (GeeksForGeeks, 2023).

   b. **Exploration**: Allows our intelligent agent to improve on its knowledge by discovering the environment. This will lead to more accurate action-based value estimates. The pirate randomly chooses an action from the member method call to valid actions. Thus, the pirate is able to make "more informed decisions in the future" (GeeksForGeeks, 2023).

c. **Ideal Proportion**:

   i. The algorithm solved the completion check in 900 epochs with a 0.1 epsilon value.



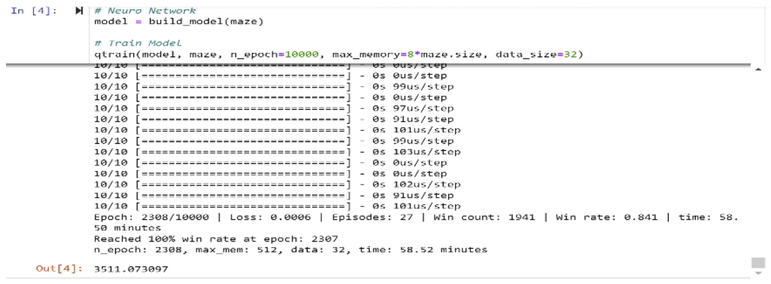
```
[5]:     # Neuro Network
         model = build_model(maze)

         # Train Model
         qtrain(model, maze, n_epoch=10000, max_memory=8*maze.size, data_size=32)
         10/10 [==============================] - 0s 98us/step
         10/10 [==============================] - 0s 99us/step
         10/10 [==============================] - 0s 102us/step
         10/10 [==============================] - 0s 101us/step
         10/10 [==============================] - 0s 0us/step
         10/10 [==============================] - 0s 198us/step
         10/10 [==============================] - 0s 104us/step
         10/10 [==============================] - 0s 201us/step
         10/10 [==============================] - 0s 200us/step
         10/10 [==============================] - 0s 100us/step
         10/10 [==============================] - 0s 100us/step
         10/10 [==============================] - 0s 99us/step
         10/10 [==============================] - 0s 98us/step
         10/10 [==============================] - 0s 100us/step
         Epoch: 900/10000 | Loss: 0.0004 | Episodes: 40 | Win count: 552 | Win rate: 0.613 | time: 35.34 minutes
         Reached 100% win rate at epoch: 899
         n_epoch: 900, max_mem: 512, data: 32, time: 35.35 minutes

Out[5]:  2120.960579
```

- Epsilon = 0.1

   ii. The algorithm solved the completion check in 2307 epochs with a 0.3 epsilon value.



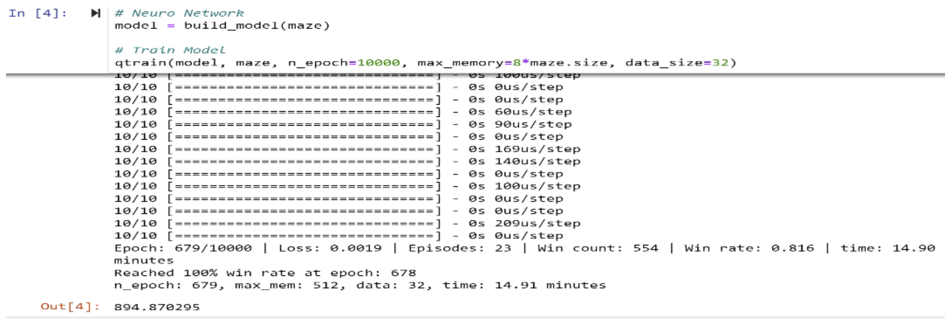
```
In [4]:     # Neuro Network
            model = build_model(maze)

            # Train Model
            qtrain(model, maze, n_epoch=10000, max_memory=8*maze.size, data_size=32)
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 99us/step
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 97us/step
            10/10 [==============================] - 0s 91us/step
            10/10 [==============================] - 0s 101us/step
            10/10 [==============================] - 0s 99us/step
            10/10 [==============================] - 0s 103us/step
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 102us/step
            10/10 [==============================] - 0s 91us/step
            10/10 [==============================] - 0s 101us/step
            Epoch: 2308/10000 | Loss: 0.0006 | Episodes: 27 | Win count: 1941 | Win rate: 0.841 | time: 58.
            50 minutes
            Reached 100% win rate at epoch: 2307
            n_epoch: 2308, max_mem: 512, data: 32, time: 58.52 minutes

   Out[4]:  3511.073097
```

- Epsilon = 0.3

   iii. The algorithm completed the check in 697 epochs with a 0.05 epsilon value.



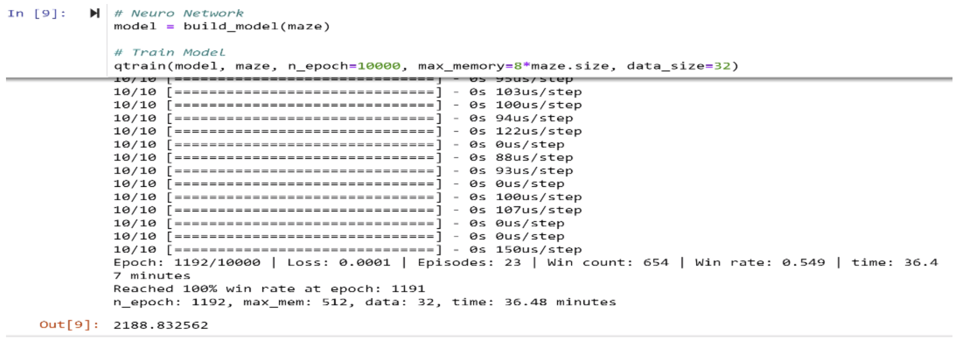
```
In [4]:     # Neuro Network
            model = build_model(maze)

            # Train Model
            qtrain(model, maze, n_epoch=10000, max_memory=8*maze.size, data_size=32)
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 60us/step
            10/10 [==============================] - 0s 90us/step
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 169us/step
            10/10 [==============================] - 0s 140us/step
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 100us/step
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 0us/step
            10/10 [==============================] - 0s 209us/step
            10/10 [==============================] - 0s 0us/step
            Epoch: 679/10000 | Loss: 0.0019 | Episodes: 23 | Win count: 554 | Win rate: 0.816 | time: 14.90
            minutes
            Reached 100% win rate at epoch: 678
            n_epoch: 679, max_mem: 512, data: 32, time: 14.91 minutes

   Out[4]:  894.870295
```

- Epsilon = 0.05

   iv. With an epsilon value of 0.01, the algorithm completed the check in 1192 epochs



```
In [9]:    # Neuro Network
           model = build_model(maze)

           # Train Model
           qtrain(model, maze, n_epoch=10000, max_memory=8*maze.size, data_size=32)
           10/10 [==============================] - 0s 95us/step
           10/10 [==============================] - 0s 103us/step
           10/10 [==============================] - 0s 100us/step
           10/10 [==============================] - 0s 94us/step
           10/10 [==============================] - 0s 122us/step
           10/10 [==============================] - 0s 0us/step
           10/10 [==============================] - 0s 88us/step
           10/10 [==============================] - 0s 93us/step
           10/10 [==============================] - 0s 0us/step
           10/10 [==============================] - 0s 100us/step
           10/10 [==============================] - 0s 107us/step
           10/10 [==============================] - 0s 0us/step
           10/10 [==============================] - 0s 0us/step
           10/10 [==============================] - 0s 150us/step
           Epoch: 1192/10000 | Loss: 0.0001 | Episodes: 23 | Win count: 654 | Win rate: 0.549 | time: 36.4
           7 minutes
           Reached 100% win rate at epoch: 1191
           n_epoch: 1192, max_mem: 512, data: 32, time: 36.48 minutes

Out[9]:    2188.832562
```

- Epsilon = 0.01

v. Ideal proportion appears to be with an epsilon value of 0.05.

d. **Reasoning**: The multi-armed bandit problem is a concept that formalizes decision-making. Our intelligent agent chooses between valid actions and receives a reward on the chosen action. The problem highlights the importance of rewards, timesteps, and values (GeeksForGeeks, 2023).

e. Above, I discussion exploration and exploitation. An algorithm has to choose to do one or the other. The Treasure Hunt Game implements an epsilon-greedy process to randomly choose between exploration and exploitation. Our intelligent agent is designed to exploit most of the time and explores sometimes.

f. Notable changes occurred when the epsilon value was manipulated. While the win rate never went higher than 81 percent, the lowest number of epochs was when the epsilon value was 0.05. The number of epochs were higher when the epsilon value was more or less than 0.05.

g. Additionally, it appears that when epsilon is above 0.1 and below 0.05, the intelligent agents does not find the ‘shortest path’ to the treasure that maximizes the reward (sub-optimal behavior). Based on the empirical evidence, I am led to theorize that an epsilon value between 0.05 – 0.1 is the ideal proportion for the greedy-epsilon algorithm in the Treasure Hunt Game (GeeksForGeeks, 2023)..

2. **How can reinforcement learning help to determine the path to the goal (the treasure) by the agent (the pirate)?**
    a. REINFORCE could be implemented to improve the accuracy of our intelligent agent's action-based estimates. A REINFORCE agent utilizes a parallel network created from the attributes of the observations and actions. The network can "learn to predict the action given an observation from the environment" (TF, 2023).
    b. We import actor distribution network from tf_agents to create the network.
    c. Next, a variable is created and assigned to an object instance of the network that is passed the observation spec, action spec, and layers in the network as arguments.
    d. An optimizer is needed to train the newly created network. The original multi-arm problem used Adam. Similarly, we can use Adam for our REINFORCE network passed with a specific learning rate as an argument. A counter will keep track of how many times the network is updated (TF, 2023).
    e. We can finally create our agent. Using the reinforce agent module, a variable is declared and initialized to an object instance passed with the following arguments.
        i. Time, Action, Optimizer, Counter
        ii. The agent contains evaluation/deployment and data collection policies.
    f. We can compute the average return over a few episodes and use the output to evaluate the policy. Additionally, we can use the experience replay object to keep track of the data collected. The replay buffer will contain episodes with the following attributes (TF, 2023).
        i. Previous State, Action, Rewards, Current State, Game Status.

g. The policy is evaluated once before training. Episodes are collected and stored in the replay buffer. Data is sampled from the buffer and the counter is incremented. The agent's network is updated after the average return is calculated (TF, 2023).

h. The accuracy of the calculated action-based estimate will be enhanced. In theory, the intelligent agent is more motivated to find the shortest-path.

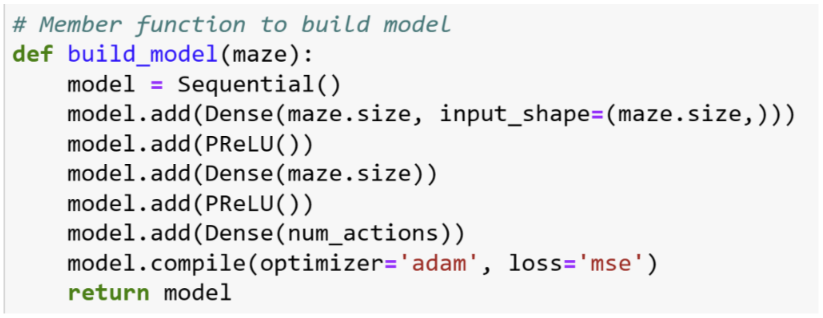
Evaluate the use of algorithms to solve complex problems.

1. **How did you implement deep Q-learning using neural networks for this game?**

   a. I created a model free algorithm that is able to make "predictions of future states and rewards". The Treasure Maze game is a multi-arm problem where an array is used as a maze and the pirate has to find the treasure while avoiding obstacles. The action space is described with 4 values (0 - 3) allowing the pirate to possibly move up, down, left, or right depending on the current cell (Wang, 2020).

   b. A normal Q-learning algorithm will map each "state-action pair to its corresponding Q-value". My code base replaces the Q-table with a "neural network that maps input states to (action/Q-value) pairs". The Q-values are predictions (Wang, 2020).

   c. The neuro network's architecture is defined below:

      i. 

```
# Member function to build model
def build_model(maze):
    model = Sequential()
    model.add(Dense(maze.size, input_shape=(maze.size,)))
    model.add(PReLU())
    model.add(Dense(maze.size))
    model.add(PReLU())
    model.add(Dense(num_actions))
    model.compile(optimizer='adam', loss='mse')
    return model
```

      ii. My Treasure Maze class member method takes in an array (maze) and instantiates a Sequential model instance that contains two networks. The target and main networks "consist of 3 densely connected neuron layers". First two layers with PReLu activation functions and the third is a dense layer equal to the size of the number of actions (Wang, 2020).

d. The decision to explore or exploit is based on the possibly that an exploration factor is greater than a randomly generated number. When exploiting, the algorithm will choose "the action that has the largest predicted value" between our two networks created earlier" (Wang, 2020).
e. Thus, the decision process under uncertainty (explore, exploit) is determined by using the epsilon-greedy algorithm. Each action has a reward and an associated q-value. The Treasure Hunt member method returns an action data structure with attributes passed to a Game Experience replay object (Wang, 2020).
f. My Deep Q-Learning algorithm enabled the pirate to learn from the replay object by "updating the algorithm's parameters using saved and stored information" from previous episodes. The inputs and targets from the Game Experience object instance member method call were used to train the model. The model is trained until it can find the treasure and pass the completion check (Wang, 2020).
g. The above demonstrates how I implemented deep Q-learning with neuro networks to solve the Treasure Hunt Game.

Resources

Tensor Flow Agent Authors. (2023). *REINFORCE agent*. [Blog Post]. Retrieved December 8, 2023 from https://www.tensorflow.org/agents/tutorials/6_reinforce_tutorial

Amit. (2023). Amit's Thoughts on Pathfinding: Introduction to A*. [Blog Post]. Retrieved December 9, 2023 from Introduction to A* (stanford.edu)

GeeksForGeeks. (n.d.). Epsilon-Greedy Algorithm in Reinforcement Learning. [Blog Post]. Retrieved December 9, 2023 from https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/

Wang, Mike. (2020). Deep Q-Learning Tutorial: minDQN: A Practical Gide to Deep Q-Networks. [Blog Post]. Medium. Retrieved December 10, 2023 from https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc