Southern New Hampshire University

CS 370 Pirate Intelligent Agent Specifications

Agent Specifications

- You will use the Python programming language for this project, as well as the TensorFlow and Keras libraries. These have been pre-installed in the Virtual Lab (Apporto).
- The environment for your agent has already been designed as a maze (8x8 matrix), containing free (1), occupied (0), and target (1 at the bottom right) cells, as below:

```
[ 1.,  0.,  1.,  1.,  1.,  1.,  1.,  1.],
[ 1.,  0.,  1.,  1.,  1.,  0.,  1.,  1.],
[ 1.,  1.,  1.,  1.,  0.,  1.,  0.,  1.],
[ 1.,  1.,  1.,  0.,  1.,  1.,  1.,  1.],
[ 1.,  1.,  0.,  1.,  1.,  1.,  1.,  1.],
[ 1.,  1.,  1.,  0.,  1.,  0.,  0.,  0.],
[ 1.,  1.,  1.,  0.,  1.,  1.,  1.,  1.],
[ 1.,  1.,  1.,  1.,  0.,  1.,  1.,  1.]
```

- Your agent (pirate) should start at the top left. The agent can move in four directions: left, right, up, and down.
- The agent rewards vary from -1 point to 1 point. When the agent reaches the target, the reward will be 1 point. Moving to an occupied cell will result in a penalty of -0.75 points. Attempting to move outside the matrix boundary will result in a penalty of -0.8 points. Moving from a cell to an adjacent cell will result in a penalty of -0.04 points, primarily to avoid the agent wandering within the maze.
- A negative threshold has been defined for you in order to reduce training time, avoid infinite loops, and avoid unnecessary wandering.

Provided Elements

Below is a brief description of the different elements involved in the game. Several elements have already been given to you in the starter code. You will need to create the code for the **Q-Training Algorithm** section yourself.

Environment

(NOTE: You have been given this code)

TreasureEnvironment.py contains complete code for your environment. It includes a maze object defined as a matrix. The provided code supports methods for resetting the pirate position, updating the state based on pirate movement, returning rewards based on agent movement guidelines, keeping track of the state and total reward based on agent action, determining the current environment state and game status, listing the valid actions from the current cell, and a visualization method for graphical display of environment and agent action.

Experience for Replay

(NOTE: You have been given this code)

GameExperience.py contains complete code for experience replay. It stores the episodes, all the states that come in between the initial state and the terminal state. This is later used by the agent for learning by experience. The class supports methods for storing episodes in memory, predicting the next action

based on the current environment state, and returning input and targets from memory based on specified data size.

Build Model

(NOTE: You have been given this code)

You have been given a complete implementation to build a neural network model in the TreasureHuntGame Jupyter notebook. Make sure to review the code and note the number of layers, as well as the activation, optimizer, and loss functions that are used to train the model.

Q-Training Algorithm

(NOTE: You will need to *create* this code)

You have been given a skeleton implementation in the TreasureHuntGame Jupyter Notebook. Your task is to implement deep-Q learning. The goal of your deep Q-learning implementation is to find the best possible navigation sequence that results in reaching the treasure cell while maximizing the reward. In your implementation, you need to determine the optimal number of epochs to achieve a 100% win rate.

Play Game

(NOTE: You have been given this code)

You have been given a complete implementation of this function in the TreasureHuntGame Jupyter notebook. This function helps you to determine whether the pirate can win any game at all. If your maze is not well designed, the pirate may not be able to win, in which case your training may not yield any result. The provided maze in this notebook ensures that there is a path to win and you can run this method to check.