Treasure Hunt Game Notebook

Read and Review Your Starter Code

The theme of this project is a popular treasure hunt game in which the player needs to find the treasure before the pirate does. While you will not be developing the entire game, you will write the part of the game that represents the intelligent agent, which is a pirate in this case. The pirate will try to find the optimal path to the treasure using deep Q-learning.

You have been provided with two Python classes and this notebook to help you with this assignment. The first class, TreasureMaze.py, represents the environment, which includes a maze object defined as a matrix. The second class, GameExperience.py, stores the episodes – that is, 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, called "exploration". This notebook shows how to play a game. Your task is to complete the deep Q-learning implementation for which a skeleton implementation has been provided. The code blocs you will need to complete has #TODO as a header.

First, read and review the next few code and instruction blocks to understand the code that you have been given.

In [1]:
```
from __future__ import print_function
import os, sys, time, datetime, json, random
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD , Adam, RMSprop
from keras.layers.advanced_activations import PReLU
import matplotlib.pyplot as plt
from TreasureMaze import TreasureMaze
from GameExperience import GameExperience
%matplotlib inline
```

```
Using TensorFlow backend.
```

The following code block contains an 8x8 matrix that will be used as a maze object:

In [2]:
```
maze = np.array([
    [ 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.]
])
```

This helper function allows a visual representation of the maze object:

In [3]:
```
def show(qmaze):
    plt.grid('on')
    nrows, ncols = qmaze.maze.shape
    ax = plt.gca()
    ax.set_xticks(np.arange(0.5, nrows, 1))
    ax.set_yticks(np.arange(0.5, ncols, 1))
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    canvas = np.copy(qmaze.maze)
    for row,col in qmaze.visited:
        canvas[row,col] = 0.6
    pirate_row, pirate_col, _ = qmaze.state
    canvas[pirate_row, pirate_col] = 0.3   # pirate cell
    canvas[nrows-1, ncols-1] = 0.9 # treasure cell
    img = plt.imshow(canvas, interpolation='none', cmap='gray')
    return img
```

The pirate agent can move in four directions: left, right, up, and down.

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. You are encouraged to try various values for the exploration factor and see how the algorithm performs.

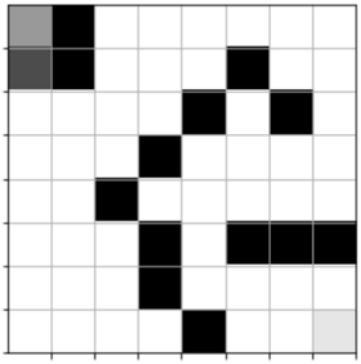
In [4]:
```
LEFT = 0
UP = 1
RIGHT = 2
DOWN = 3


# Exploration factor
epsilon = 0.1

# Actions dictionary
actions_dict = {
    LEFT: 'left',
    UP: 'up',
    RIGHT: 'right',
    DOWN: 'down',
}

num_actions = len(actions_dict)
```

The sample code block and output below show creating a maze object and performing one action (DOWN), which returns the reward. The resulting updated environment is visualized.

In [5]:
```
qmaze = TreasureMaze(maze)
canvas, reward, game_over = qmaze.act(DOWN)
print("reward=", reward)
show(qmaze)
```

```
reward= -0.04
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x28ad6945cc8>
```



This function simulates a full game based on the provided trained model. The other parameters include the TreasureMaze object and the starting position of the pirate.

In [6]:
```
def play_game(model, qmaze, pirate_cell):
    qmaze.reset(pirate_cell)
    envstate = qmaze.observe()
    while True:
        prev_envstate = envstate
        # get next action
        q = model.predict(prev_envstate)
        action = np.argmax(q[0])

        # apply action, get rewards and new state
        envstate, reward, game_status = qmaze.act(action)
        if game_status == 'win':
            return True
        elif game_status == 'lose':
            return False
```

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 win any game at all. In this case, your training would 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.

In [7]:
```
def completion_check(model, qmaze):
    for cell in qmaze.free_cells:
        if not qmaze.valid_actions(cell):
            return False
        if not play_game(model, qmaze, cell):
            return False
    return True
```

The code you have been given in this block will build the neural network model. 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.

In [8]:
```
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
```

#TODO: Complete the Q-Training Algorithm Code Block

This is your deep Q-learning implementation. 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.

You will need to complete the section starting with #pseudocode. The pseudocode has been included for you.

In [9]:

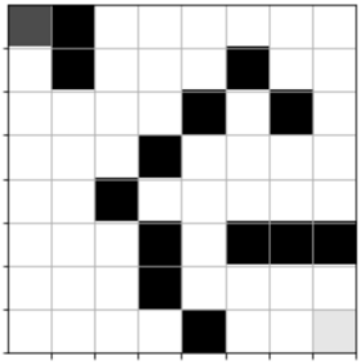
```
def qtrain(model, maze, **opt):

    # exploration factor
    global epsilon

    # number of epochs
    n_epoch = opt.get('n_epoch', 15000)

    # maximum memory to store episodes
    max_memory = opt.get('max_memory', 1000)

    # maximum data size for training
    data_size = opt.get('data_size', 50)

    # start time
    start_time = datetime.datetime.now()

    # Construct environment/game from numpy array: maze (see above)
    qmaze = TreasureMaze(maze)

    # Initialize experience replay object
    experience = GameExperience(model, max_memory=max_memory)

    win_history = []   # history of win/lose game
    hsize = qmaze.maze.size//2   # history window size
    win_rate = 0.0

    # pseudocode:
    # For each epoch:
    #    Agent_cell = randomly select a free cell
    #    Reset the maze with agent set to above position
    #    Hint: Review the reset method in the TreasureMaze.py class.
    #    envstate = Environment.current_state
    #    Hint: Review the observe method in the TreasureMaze.py class.
    #    While state is not game over:
    #        previous_envstate = envstate
    #        Action = randomly choose action (left, right, up, down) either by exploration o
r by exploitation
    #        envstate, reward, game_status = qmaze.act(action)
    #    Hint: Review the act method in the TreasureMaze.py class.
    #        episode = [previous_envstate, action, reward, envstate, game_status]
    #        Store episode in Experience replay object
    #    Hint: Review the remember method in the GameExperience.py class.
    #        Train neural network model and evaluate loss
    #    Hint: Call GameExperience.get_data to retrieve training data (input and target) and
pass to model.fit method
    #          to train the model. You can 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.

    #Print the epoch, loss, episodes, win count, and win rate for each epoch
        dt = datetime.datetime.now() - start_time
        t = format_time(dt.total_seconds())
        template = "Epoch: {:03d}/{:d} | Loss: {:.4f} | Episodes: {:d} | Win count: {:d} | W
in rate: {:.3f} | time: {}"
        print(template.format(epoch, n_epoch-1, loss, n_episodes, sum(win_history), win_rat
e, t))
        # We simply check if training has exhausted all free cells and if in all
        # cases the agent won.
        if win_rate > 0.9 : epsilon = 0.05
        if sum(win_history[-hsize:]) == hsize and completion_check(model, qmaze):
            print("Reached 100%% win rate at epoch: %d" % (epoch,))
            break


    # Determine the total time for training
    dt = datetime.datetime.now() - start_time
    seconds = dt.total_seconds()
    t = format_time(seconds)
```

Test Your Model

Now we will start testing the deep Q-learning implementation. To begin, select **Cell**, then **Run All** from the menu bar. This will run your notebook. As it runs, you should see output begin to appear beneath the next few cells. The code below creates an instance of TreasureMaze.

In [10]:
```
qmaze = TreasureMaze(maze)
show(qmaze)
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x28ad6a09548>
```



In the next code block, you will build your model and train it using deep Q-learning. Note: This step takes several minutes to fully run.

In [11]:
```
model = build_model(maze)
qtrain(model, maze, epochs=1000, max_memory=8*maze.size, data_size=32)
```

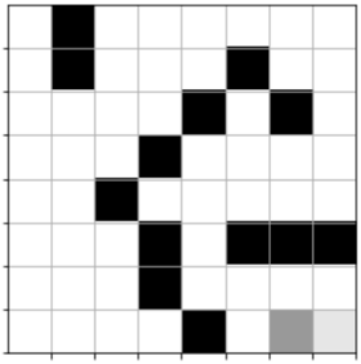
```
Epoch: 000/14999 | Loss: 0.0017 | Episodes: 148 | Win count: 0 | Win rate: 0.000 | time: 1
2.5 seconds
Epoch: 001/14999 | Loss: 0.0017 | Episodes: 145 | Win count: 0 | Win rate: 0.000 | time: 2
3.9 seconds
Epoch: 002/14999 | Loss: 0.0018 | Episodes: 142 | Win count: 0 | Win rate: 0.000 | time: 3
5.4 seconds
Epoch: 003/14999 | Loss: 0.0013 | Episodes: 7 | Win count: 1 | Win rate: 0.000 | time: 36.
0 seconds
Epoch: 004/14999 | Loss: 0.0013 | Episodes: 1 | Win count: 2 | Win rate: 0.000 | time: 36.
1 seconds
Epoch: 005/14999 | Loss: 0.0391 | Episodes: 134 | Win count: 2 | Win rate: 0.000 | time: 4
6.5 seconds
Epoch: 006/14999 | Loss: 0.0052 | Episodes: 139 | Win count: 2 | Win rate: 0.000 | time: 5
7.5 seconds
Epoch: 007/14999 | Loss: 0.0038 | Episodes: 144 | Win count: 2 | Win rate: 0.000 | time: 6
8.9 seconds
Epoch: 008/14999 | Loss: 0.0025 | Episodes: 73 | Win count: 3 | Win rate: 0.000 | time: 7
4.6 seconds
Epoch: 009/14999 | Loss: 0.0105 | Episodes: 11 | Win count: 4 | Win rate: 0.000 | time: 7
5.5 seconds
Epoch: 010/14999 | Loss: 0.0088 | Episodes: 10 | Win count: 5 | Win rate: 0.000 | time: 7
6.3 seconds
Epoch: 011/14999 | Loss: 0.0053 | Episodes: 139 | Win count: 5 | Win rate: 0.000 | time: 8
7.4 seconds
Epoch: 012/14999 | Loss: 0.0112 | Episodes: 137 | Win count: 5 | Win rate: 0.000 | time: 9
8.4 seconds
Epoch: 013/14999 | Loss: 0.0014 | Episodes: 142 | Win count: 5 | Win rate: 0.000 | time: 1
09.6 seconds
Epoch: 014/14999 | Loss: 0.0016 | Episodes: 146 | Win count: 5 | Win rate: 0.000 | time: 1
21.1 seconds
Epoch: 015/14999 | Loss: 0.0046 | Episodes: 46 | Win count: 6 | Win rate: 0.000 | time: 12
4.8 seconds
Epoch: 016/14999 | Loss: 0.0044 | Episodes: 15 | Win count: 7 | Win rate: 0.000 | time: 12
6.0 seconds
Epoch: 017/14999 | Loss: 0.0048 | Episodes: 7 | Win count: 8 | Win rate: 0.000 | time: 12
6.5 seconds
Epoch: 018/14999 | Loss: 0.0046 | Episodes: 5 | Win count: 9 | Win rate: 0.000 | time: 12
7.0 seconds
Epoch: 019/14999 | Loss: 0.0307 | Episodes: 143 | Win count: 9 | Win rate: 0.000 | time: 1
38.2 seconds
Epoch: 020/14999 | Loss: 0.0268 | Episodes: 2 | Win count: 10 | Win rate: 0.000 | time: 13
8.4 seconds
Epoch: 021/14999 | Loss: 0.0022 | Episodes: 12 | Win count: 11 | Win rate: 0.000 | time: 1
39.4 seconds
Epoch: 022/14999 | Loss: 0.0024 | Episodes: 144 | Win count: 11 | Win rate: 0.000 | time:
150.8 seconds
Epoch: 023/14999 | Loss: 0.0161 | Episodes: 142 | Win count: 11 | Win rate: 0.000 | time:
162.3 seconds
Epoch: 024/14999 | Loss: 0.0028 | Episodes: 143 | Win count: 11 | Win rate: 0.000 | time:
173.5 seconds
Epoch: 025/14999 | Loss: 0.0205 | Episodes: 7 | Win count: 12 | Win rate: 0.000 | time: 17
4.1 seconds
Epoch: 026/14999 | Loss: 0.0021 | Episodes: 143 | Win count: 12 | Win rate: 0.000 | time:
185.4 seconds
Epoch: 027/14999 | Loss: 0.0044 | Episodes: 1 | Win count: 13 | Win rate: 0.000 | time: 18
5.5 seconds
Epoch: 028/14999 | Loss: 0.0413 | Episodes: 141 | Win count: 13 | Win rate: 0.000 | time:
196.9 seconds
Epoch: 029/14999 | Loss: 0.0058 | Episodes: 4 | Win count: 14 | Win rate: 0.000 | time: 19
7.3 seconds
Epoch: 030/14999 | Loss: 0.0346 | Episodes: 140 | Win count: 14 | Win rate: 0.000 | time:
209.7 seconds
Epoch: 031/14999 | Loss: 0.0026 | Episodes: 3 | Win count: 15 | Win rate: 0.000 | time: 21
0.0 seconds
Epoch: 032/14999 | Loss: 0.0022 | Episodes: 144 | Win count: 15 | Win rate: 0.469 | time:
222.2 seconds
Epoch: 033/14999 | Loss: 0.0743 | Episodes: 15 | Win count: 16 | Win rate: 0.500 | time: 2
23.4 seconds
Epoch: 034/14999 | Loss: 0.0366 | Episodes: 5 | Win count: 17 | Win rate: 0.531 | time: 22
3.8 seconds
```

Out[11]: 631.285955

This cell will check to see if the model passes the completion check. Note: This could take several minutes.

In [12]:
```
completion_check(model, qmaze)
show(qmaze)
```

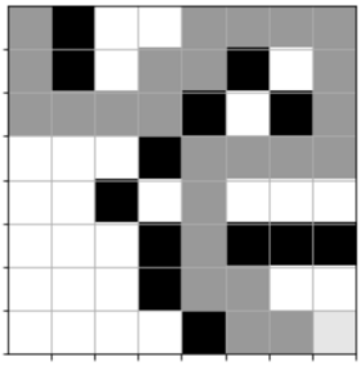
Out[12]: <matplotlib.image.AxesImage at 0x28add432c08>



This cell will test your model for one game. It will start the pirate at the top-left corner and run play_game. The agent should find a path from the starting position to the target (treasure). The treasure is located in the bottom-right corner.

In [13]:
```
pirate_start = (0, 0)
play_game(model, qmaze, pirate_start)
show(qmaze)
```

Out[13]: <matplotlib.image.AxesImage at 0x28add9a3688>



Save and Submit Your Work

After you have finished creating the code for your notebook, save your work. Make sure that your notebook contains your name in the filename (e.g. Doe_Jane_ProjectTwo.ipynb). This will help your instructor access and grade your work easily. Download a copy of your IPYNB file and submit it to Brightspace. Refer to the Jupyter Notebook in Apporto Tutorial if you need help with these tasks.