Assignment 2

Members: Blythe King, Phillip Smith

Parsing Overview:

1. The data for both of the methods can come from the same source file, as long as the lines in the source file are written as [start]/[action]/[end]/[probability] with no blank lines in between.
2. "/" is used to determine the "separation" of each element of the line.
3. The actions are added to the start location's list of possible actions, and the corresponding probabilities are stored in a dictionary that connects the start, action and end state (e.g., dictFairwayAt[states[endlocation.upper()].getNumber()] = prob).

Model-Based Method Overview:

1. After the parsing and "connecting" mechanisms, the initial step is to set the goal location, which is, in this case, in the hole.
2. A matrix M is created to connect the edges and nodes, with the edges being the actions that can be taken to get from one state to another.
3. The goal number is set to 100.
4. A Q matrix is then created, and the discount parameter (gamma), learning rate (alpha), and initial state (fairway) are all set.
5. Available actions are determined by retrieving the previously created list of actions that can be taken at each state. The initial available actions are from the fairway, as it is assumed any path will start there.
6. The action to be taken will be determined either through exploration or exploitation. A epsilon value will be chosen (in this case, epsilon is 0.9). A value will be randomly generated to determine whether exploration or exploitation will be done (if the random value is less than epsilon, exploration will be done, and if above, exploration will be done).
    a. Exploration = a random action will be chosen out of those available
    b. Exploitation = an action will be chosen by determining which action will lead to the highest score
7. The Q matrix will be updated after every step taken. The utility and score will be calculated and updated with each step.The calculation includes the models probabilities.
8. This "simulation" will be run a certain number of times (in the submitted code, 500 times) to best "train" the matrix.
9. The states, actions, and following states will be recorded in a trial matrix in order to calculate the transition probabilities.
    a. The number of times that each initial state and action combination occurs will be counted, and the number of times that each initial state, action, and end state combination occurs will be counted.
    b. Transition probability = count(initial state, action, end state) / count(initial state, action)

    c. A table showing the start state, action, end state, and transition probability will be printed. Each state state, action, and end state are represented with integers rather than with strings in order to best fill out this table.

10. The most efficient path is determined using the utility values. The max value (max utility) in the Q matrix is used to determine where the optimal step is when the current state is not 5 (closeToPin). If there are multiple utility values that are the same, then one is chosen randomly to be the next step. The most efficient path is then printed.

Model-Free Method Overview:

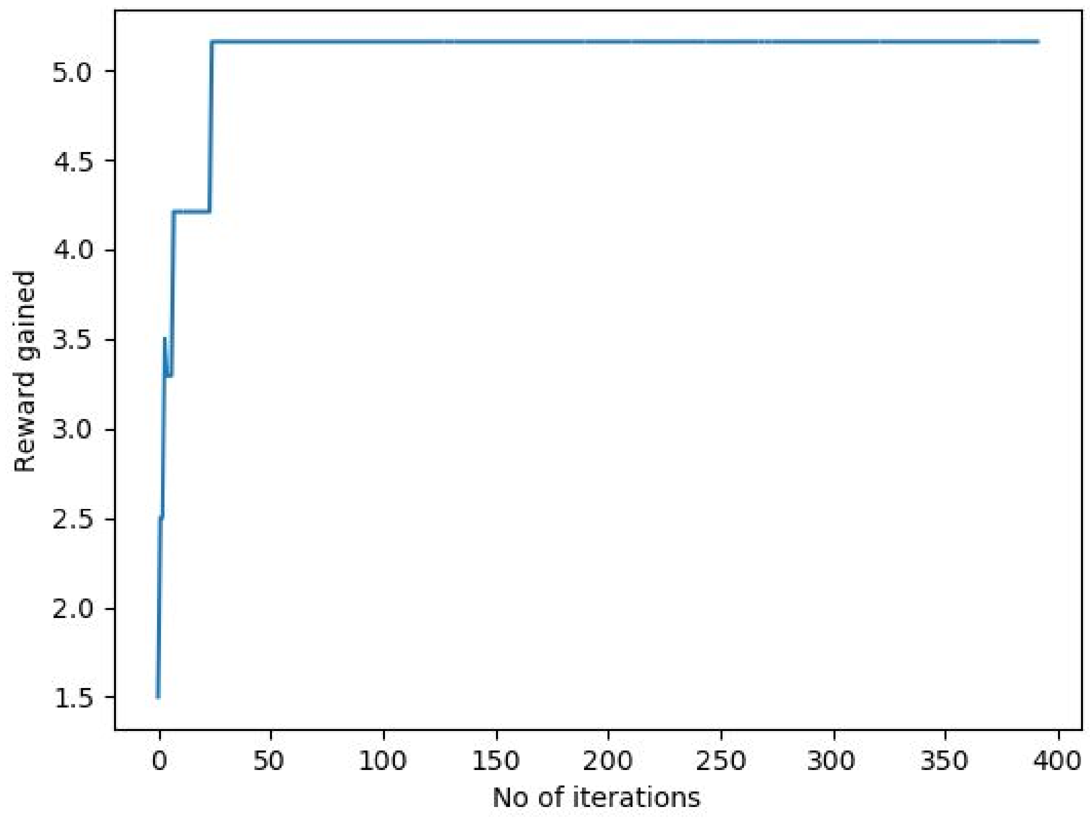
1. Steps 1-8 are the same as the model-based approach, except that the probabilities are no longer used to calculate the utility. The requirements for updating the matrix and determining whether or not exploration and exploitation are needed are the same for both algorithms.
2. The Q matrix for this method is printed out and contains the states, actions, utilities, and scores. The actions are the rows and the states are the columns. See each state/action object for its specific number.
3. The best path is calculated by finding the indexes of the best utilities. If there are multiple “best/max” utilities, then one is randomly chosen to be the index of the next state of the path.

<u>**Changing explore vs. exploit**</u>

Starting with a high exploitation value, the system no longer makes any mistakes while learning. Looking at the reward gained/iteration graph, the relationship is strictly positive, a higher iteration number always yields the same or greater reward. Generally, the optimal path the system takes is within reason, and has around three steps.

With an even split between exploring and exploiting, the system seems to take more steps to reach the hole than before. While it does not always take more steps, the optimal a larger percentage of the time has more steps in it. The optimal value for the cases where it takes more steps generally has a lower reward than the cases with fewer steps.

With a high exploration value, the system seems to more consistently produce low step, reasonable results, with rewards over 5. However, the system now makes mistakes, in that the relationship between iterations and reward gained is not strictly positive. An example of what I mean is pictured below. You can see the dip in reward around 20 iterations



Generally, a high exploration value seems to generate consistently better results than a high exploit value.

When to Stop Learning:

The system learns within a while loop limited by an iteration value. The iteration value was chosen to be low enough such that the code ran fast, but high enough the code could get to a stable level of reward, and produce reasonable answers. I would seem from the graph above that an epsilon value of around 75 would work just as well as a value of 500. This is not the case, as the 500 ensures consistent results that make sense. Essentially, with an iteration value between 75 and 100, there were cases where the system had not yet reached the optimal reward.

<u>Changing Gamma:</u>

With a gamma at .9, the code performed as described above. As the value of gamma fell, the reward at each state/action combination increased. With a gamma of .9, the values were in the .90 range. With a gamma of .5, the values moved to the .95 range, and with a gamma of .1 resulted with reward around .99. As a result, the reward gained for each policy increased. Unfortunately, this also generated some weird optimal paths. The system would find either that the ball should be hit directly into the hole, or that a weird combination of states were optimal. A gamma of .9 seems to be the best.

For the raw transition probabilities in the model-based algorithm, the value of gamma has no effect. However, a lower value of gamma would decrease the value of the utilities, as it is used as a multiplier in model-based learning.

<u>Constant gamma and alpha while increasing epsilon.</u>

As we learn longer, the result of the learning gets better. With very low level, like one iteration, the system learns virtually nothing and its results show that. It usually indicates that we should stop at almost every single state on the way to the goal. This is because it hasn't explored enough, and its knowledge of the states is basically zero. One humorous result was that it hit the ball back and forth between the ravine and fairway, before finally moving onto the green and into the hole.

With epsilon values around 30-50, the system starts to get better, but still is not great. It often recommends simply hitting the ball directly into the hole, or hits the ball into the fairway multiple times before moving on. Once the epsilon value gets to around 500, the system starts doing well. It no longer recommends just hitting the ball into the hole, but starts hitting the ball onto the green or near the green, and then putting the ball into the hole. Based on the probabilities given to us in the practice file, this behavior seems more likely to be the optimal behavior than simply just hitting the ball into the hole.

Contributions:

Blythe: Parsing, Model-Based, Report, README

Phillip: Parsing, Model-Free, Model-Based, Report