Agent Pathfinding using A* and Q-Learning

Joseph Sarabia
Yuksel Hocalar

University of Central Florida

JSarabia@knights.ucf.edu
Yuksel_Hocalar@knights.ucf.edu

Abstract

This paper will begin with an overview of how both the A pathfinding algorithm, and the Q-Learning algorithm when applied to pathfinding. We will then discuss how we implemented these algorithms in python. Finally we will discuss the results of running this implementation on different maps and provide a short video of the A* algorithm guiding an agent in a continuous world.*

I. Introduction

The A* algorithm is one of the most simple, yet robust algorithms for pathfinding. Given a starting location and a goal location that is reachable, the A* algorithm will return a shortest path from that start to the goal. The underlying idea of the A* algorithm is to utilize both a current cost function, and a projected cost function to calculate the shortest paths. The current cost function is the cost it took to reach the current location from the beginning location and the projected cost function is the cost we predict that remains until we reach the goal location from the current location. One important aspect of the projected cost function is that it cannot overestimate cost to the goal, because then the actual shortest path can be overlooked due to the high projected cost. Each unique location is given a current cost and projected cost, we then add those values together and use that as a total cost for that location.

Beginning at the starting location, we branch out towards the destination using the total cost as the value in a priority queue of locations to be visited. In the algorithm we simply select the location at the front of the queue, the lowest total cost value, and visit that location and enter its neighbors into the queue. Continuing this process we eventually reach the destination and we retrace our steps to create a shortest path.

The other algorithm we will look at it Q-Learning. Q-Learning is a reinforcement-learning algorithm that tries to model a world using a series of exploration episodes to allow for educated decisions later on. This is done by the use of a reward function, in which reward values are given to the algorithm for reaching desired states. As the algorithm continues to learn, rewards are propagated to states leading to desired states, ultimately providing the algorithm with a map of how to get to desired states given a current state.

The performance of the Q-Learning algorithm can depend heavily on input parameters given to the algorithm that control learning rate, how rewards diminish, and how rewards are given, as well as the number of episodes run during the learning phase and the exploration policy used for learning.

II. Implementation

Our implementation of the A* algorithm was done using Python. We used the standard implementation described above. I will provide the psuedo code below, along with comments on additional data structures used.

*Figure 1: Psuedo Code of A**

```
--------------------------------------------------------------------------------
    Create a Priority queue of open_nodes initialized with the start location
    Create a dictionary (similar to the java treemap) of return paths initialized with the
start location
    create a dictionary of gcosts (current cost) initialized with the start location
    create a dictionary of fcosts (total cost) initialized with the start location
    create a dictionary of hcosts (projected cost) initialized with the start location

    while the open_nodes queue isnt empty
        take the first location in the queue

        if the current node is the goal
            return the dictionaries created

        for every adjacent location to the current locations
            calculate the current cost
            if current cost is less than the previous current cost, or if it hasn't been visited
                store the new current cost for the adjacent location
                store the new projected cost for the adjacent location
                store the new total cost for the adjacent location
                store the location into the open_list for the adjacent location
                store the current location as the return path for the adjacent location

    return failure because we have not reached the goal and exhausted all possible paths
--------------------------------------------------------------------------------
```

Our implementation included extra dictionaries for the gcost, fcost, and hcost so we could more easily report those values after having executed the algorithm. Overall, the algorithm performed very well in both the grid environment and the continuous world. We will provide both a printout of the various costs stated above for one game world along with a link to a short video demonstrating the algorithms success at guiding an agent in the continuous world.

For our implementation of Q-Learning, we use a random exploration policy for learning. As such, a sufficient number of episodes need to be ran in order to better guarantee that all state-action pairs are explored. In our experiments we used an exploration depth of 150 for each episode; this is the max amount of state transitions an individual episode is allowed to take in order to reach the goal.

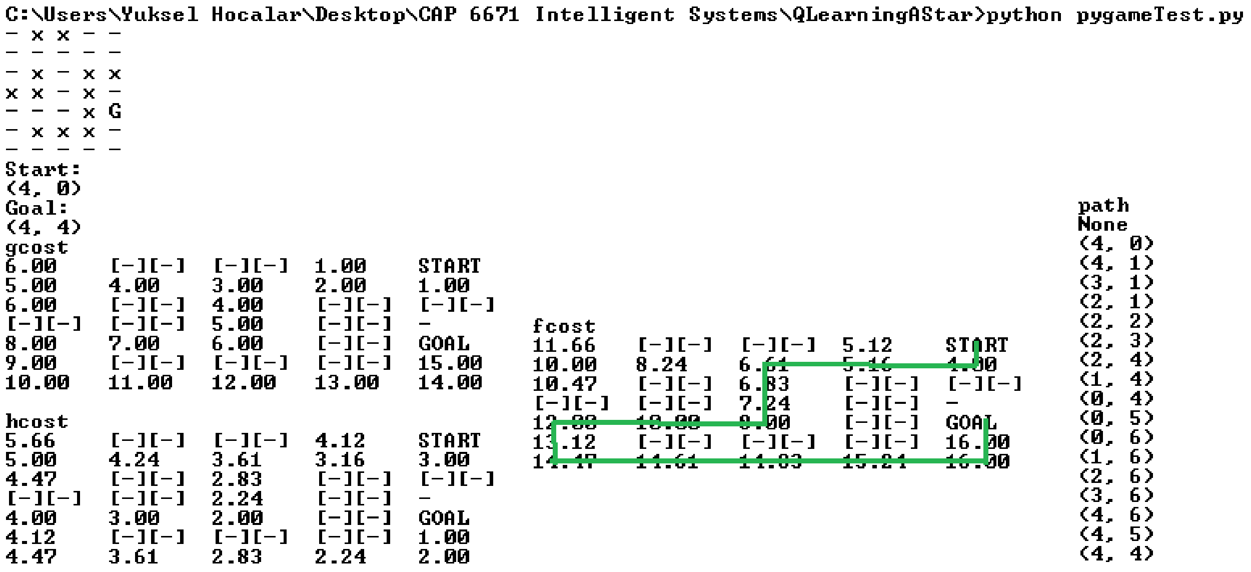
When the learning phase is completed, the agent uses the QTable to pick the best possible state to move to (which is determined by the highest reward yielding state-action pair available from the current state). In the event of a tie the first state-action pair in the list of available pairs is selected. This serves as the state-action pair that the agent would like to take, but, since the world is stochastic, the agent must pass a success check to determine if that action is actually taken. The success check is a randomly generated number between 0 and 9 is selected; and values below 6 are considered a success. If the check is failed, one of the remaining possible states is randomly selected (with evenly distributed odds determined by the number of remaining states). In our policy the agent will not have the choice of staying in the current state, and invalid states (such as those occupied by obstacles) are not provided in the list of available state-action pairs.

In Q-Learning, Our Learning rule is defined by Q(s,a) = Q(s,a) + alpha*(r(s') + gammaQ*(s',a') – Q(s,a)). R(s') is our reward function, which returns 100 if the state s' is the goal state, and 0 otherwise. S is the current state, S' is the state the agent is moving to (selected from the exploration policy), a is the action taken to get to S' from S, and a' is the action in S' that yields the highest reward value.

III. Results and Conclusion

Our implementation of the A* algorithm provided a printout of the various costs for each state. I will include an image below. Our implementation does not draw the green line showing the path. I simply included it for ease of visualization of the path printout on the right hand side.

Figure 2: A cost printout*

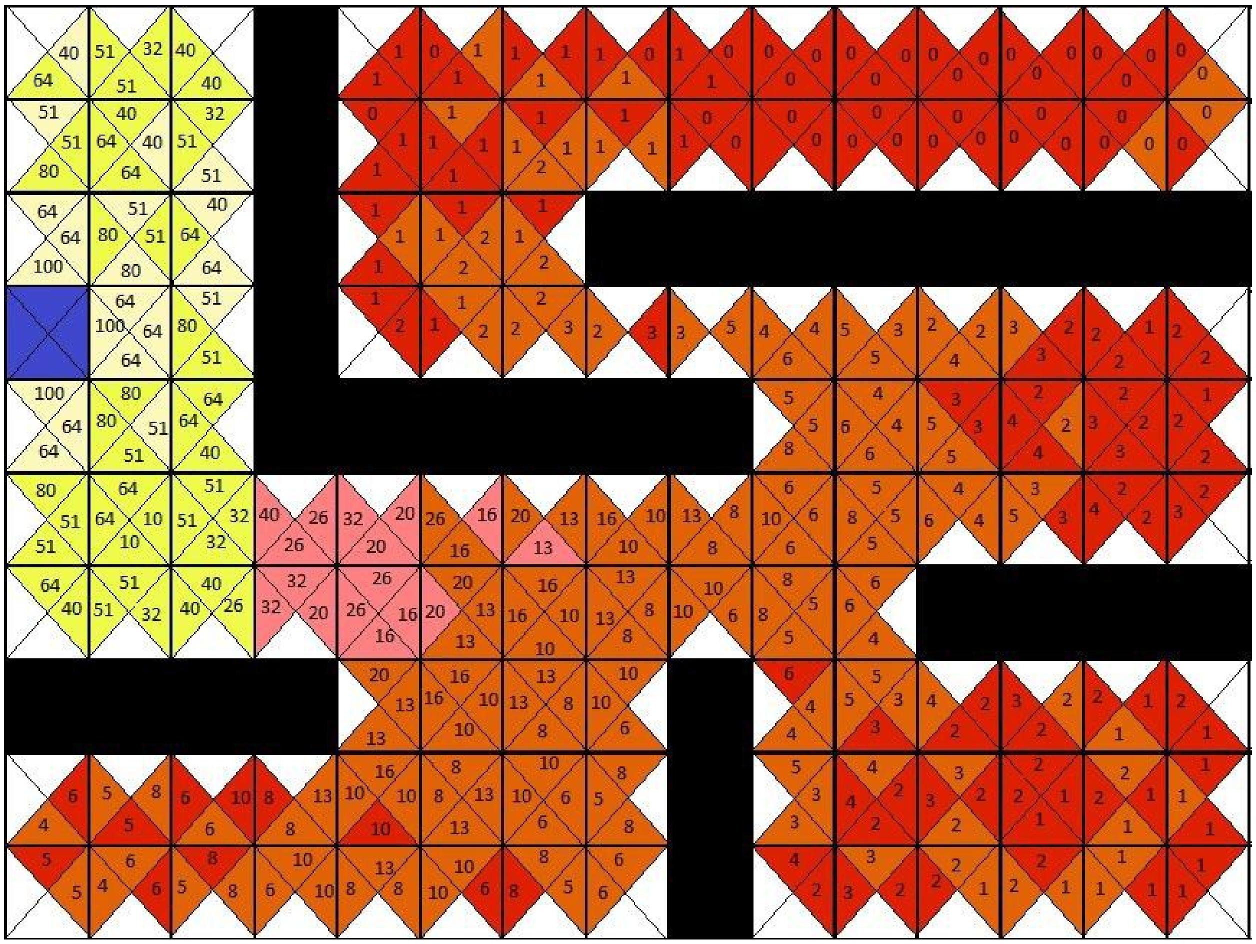


```
C:\Users\Yuksel Hocalar\Desktop\CAP 6671 Intelligent Systems\QLearningAStar>python pygameTest.py
- x x - -
- - - - -
- x - x x
x x - x -
- - - x G
- x x x -
- - - - -
Start:
(4, 0)
Goal:                                                                              path
(4, 4)                                                                             None
gcost                                                                              (4, 0)
6.00    [-][-]  [-][-]  1.00    START                                              (4, 1)
5.00    4.00    3.00    2.00    1.00                                               (3, 1)
6.00    [-][-]  4.00    [-][-]  [-][-]                                             (2, 1)
[-][-]  [-][-]  5.00    [-][-]  -        fcost                                     (2, 2)
8.00    7.00    6.00    [-][-]  GOAL     11.66   [-][-]  [-][-]  5.12    START     (2, 3)
9.00    [-][-]  [-][-]  [-][-]  15.00    10.00   8.24    6.61    5.16    4.00      (2, 4)
10.00   11.00   12.00   13.00   14.00    10.47   [-][-]  6.83    [-][-]  [-][-]    (1, 4)
                                         [-][-]  [-][-]  7.24    [-][-]  -         (0, 4)
hcost                                    12.00   10.00   8.00    [-][-]  GOAL      (0, 5)
5.66    [-][-]  [-][-]  4.12    START    13.12   [-][-]  [-][-]  [-][-]  16.00     (0, 6)
5.00    4.24    3.61    3.16    3.00     14.47   14.61   14.83   15.24   16.00     (1, 6)
4.47    [-][-]  2.83    [-][-]  [-][-]                                             (2, 6)
[-][-]  [-][-]  2.24    [-][-]  -                                                  (3, 6)
4.00    3.00    2.00    [-][-]  GOAL                                               (4, 6)
4.12    [-][-]  [-][-]  [-][-]  1.00                                               (4, 5)
4.47    3.61    2.83    2.24    2.00                                               (4, 4)
```

Here is a video link to the A* implementation guiding an agent in a continuous world: https://www.youtube.com/watch?v=nQ43wiHZ9-Y

The Heatmap was generated using 1000 episodes using an alpha of 1, and a gamma of .8. Pale yellow actions were taken less than 100 times, bright yellow actions were taken less than 200 times, salmon actions were taken less than 300 times, orange actions were taken less than 400 times, and red actions were taken more than 500 times. The image is included below:

Figure 3: Q-Learning Heatmap



----

100 episodes, 1a, .8g = converges
100 episodes .5a .8g= converges
100 episodes .2a .8g= converges
100 episodes 1a .5g = converges
100 episodes .5a .5g = converges
100 episodes .2a .5g = converges
100 episodes 1a .2g = converges
100 episodes .5a .2g = converges
100 episodes .2a .2g = converges
50 episodes 1a .8g = converges
50 episodes 1a .5g = converges
50 episodes 1a .2g = converges
50 episodes .5a .8g = converges

Varying the alpha or gamma values did not seem to make much of a difference on the convergence rate of the algorithm. Varying alpha between .2, .5, and 1 , and gamma between .2, .5, and .8 seemed to only affect how small or large the reward values are. Even with low values or alpha and gamma, the algorithm is still able to distinguish good actions from bad actions though low values can result in very small floating-point numbers, which could be of concern for a large enough map. In fact, with all combinations of the above alpha and gamma values, the algorithm was able to converge for 100 and 50 episodes. At 25 iterations the algorithm did not converge, with most elements of the QTable remaining zero. As such the agent is only able to travel to the goal because of the stochastic nature of the world.

Two paths from 0,0 to 0,5

```
Agent selects direction:  down
Agent moves in direction:  down
(0, 1)
Agent selects direction:  down
Agent moves in direction:  up
(0, 0)
Agent selects direction:  down
Agent moves in direction:  down
(0, 1)
Agent selects direction:  down
Agent moves in direction:  up
(0, 0)
Agent selects direction:  down
Agent moves in direction:  down
(0, 1)
Agent selects direction:  down
Agent moves in direction:  down
(0, 2)
Agent selects direction:  down
Agent moves in direction:  right
(1, 2)
Agent selects direction:  left
Agent moves in direction:  up
(1, 1)
Agent selects direction:  left
Agent moves in direction:  left
(0, 1)
Agent selects direction:  down
Agent moves in direction:  down
(0, 2)
Agent selects direction:  down
Agent moves in direction:  down
(0, 3)
Agent selects direction:  down
Agent moves in direction:  down
```

```
(0, 4)
Agent selects direction:  down
Agent moves in direction:  right
(1, 4)
Agent selects direction:  left
Agent moves in direction:  up
(1, 3)
Agent selects direction:  left
Agent moves in direction:  right
(2, 3)
Agent selects direction:  left
Agent moves in direction:  up
(2, 2)
Agent selects direction:  left
Agent moves in direction:  down
(2, 3)
Agent selects direction:  left
Agent moves in direction:  down
(2, 4)
Agent selects direction:  left
Agent moves in direction:  down
(2, 5)
Agent selects direction:  left
Agent moves in direction:  up
(2, 4)
Agent selects direction:  left
Agent moves in direction:  left
(1, 4)
Agent selects direction:  left
Agent moves in direction:  left
(0, 4)
Agent selects direction:  down
Agent moves in direction:  up
(0, 3)
Agent selects direction:  down
Agent moves in direction:  down
(0, 4)
Agent selects direction:  down
Agent moves in direction:  right
(1, 4)
Agent selects direction:  left
Agent moves in direction:  right
(2, 4)
Agent selects direction:  left
Agent moves in direction:  up
(2, 3)
```

```
Agent selects direction:  left
Agent moves in direction:  left
(1, 3)
Agent selects direction:  left
Agent moves in direction:  left
(0, 3)
Agent selects direction:  down
Agent moves in direction:  down
(0, 4)
Agent selects direction:  down
Agent moves in direction:  down
(0, 5)
```

Path 2

```
Agent selects direction:  down
Agent moves in direction:  right
(1, 0)
Agent selects direction:  left
Agent moves in direction:  down
(1, 1)
Agent selects direction:  left
Agent moves in direction:  up
(1, 0)
Agent selects direction:  left
Agent moves in direction:  down
(1, 1)
Agent selects direction:  left
Agent moves in direction:  right
(2, 1)
Agent selects direction:  left
Agent moves in direction:  up
(2, 0)
Agent selects direction:  left
Agent moves in direction:  down
(2, 1)
Agent selects direction:  left
Agent moves in direction:  down
(2, 2)
Agent selects direction:  left
Agent moves in direction:  down
(2, 3)
Agent selects direction:  left
Agent moves in direction:  left
(1, 3)
Agent selects direction:  left
Agent moves in direction:  left
```

```
(0, 3)
Agent selects direction:  down
Agent moves in direction:  down
(0, 4)
Agent selects direction:  down
Agent moves in direction:  down
(0, 5)
```

In conclusion, Both the A* algorithm and Q-Learning were effective in guiding an agent throughout their respective environments. The agent was able to successfully avoid obstacles and navigate to its goal destination in a timely manner.