STEP 1:

Question 1:

First, when gridworld.py -h is executed to understand the different options, the following output is obtained:

```
aahana@MacBook-Pro reinforcement % python gridworld.py -h
Usage: gridworld.py [options]

Options:
  -h, --help            show this help message and exit
  -d DISCOUNT, --discount=DISCOUNT
                        Discount on future (default 0.9)
  -r R, --livingReward=R
                        Reward for living for a time step (default 0.0)
  -n P, --noise=P       How often action results in unintended direction
                        (default 0.2)
  -e E, --epsilon=E     Chance of taking a random action in q-learning
                        (default 0.3)
  -l P, --learningRate=P
                        TD learning rate (default 0.5)
  -i K, --iterations=K  Number of rounds of value iteration (default 10)
  -k K, --episodes=K    Number of epsiodes of the MDP to run (default 1)
  -g G, --grid=G        Grid to use (case sensitive; options are BookGrid,
                        BridgeGrid, CliffGrid, MazeGrid, default BookGrid)
  -w X, --windowSize=X  Request a window width of X pixels *per grid cell*
                        (default 150)
  -a A, --agent=A       Agent type (options are 'random', 'value' and 'q',
                        default random)
  -t, --text            Use text-only ASCII display
  -p, --pause           Pause GUI after each time step when running the MDP
  -q, --quiet           Skip display of any learning episodes
  -s S, --speed=S       Speed of animation, S > 1.0 is faster, 0.0 < S < 1.0
                        is slower (default 1.0)
  -m, --manual          Manually control agent
  -v, --valueSteps      Display each step of value iteration
aahana@MacBook-Pro reinforcement %
```

Then, valueIterationAgents.py is edited to perform value iteration. I start by pulling all the MDP states and printing them. I got an error for taking a range on a list.

```
# Write value iteration code here
"*** YOUR CODE HERE ***"
for i in range(self.iterations):
    updated_values = self.values.copy()
    mdp_states = self.mdp.getStates()
    for j in range(0, mdp_states):
        print("Current state: ", mdp_states)
```

```
inforcement/valueIterationAgents.py", line 52, in __init__
    for j in range(0, mdp_states):
TypeError: range() integer end argument expected, got list.
```

Hence, the range function is taken away and all the MDP states are printed and all the MDP states are printed out.

```
aahana@MacBook-Pro reinforcement % python gridworld.py -a value -i 5
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future rel
ease. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
('Current state: ', ['TERMINAL_STATE', (0, 0), (0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1),
 (2, 2), (3, 0), (3, 1), (3, 2)])
*** Method not implemented: computeActionFromValues at line 83 of /Users/aahana/Desktop/School
/CAP5636 - Advanced AI/Assignment_1/reinforcement/valueIterationAgents.py
```

I wanted the max_value of a state to be initialized to negative infinity. I tried to use the math library for it. However, I ran into errors. I fixed it by using float('-inf'). This seemed to work.

```
Traceback (most recent call last):
  File "gridworld.py", line 485, in <module>
    a = valueIterationAgents.ValueIterationAgent(mdp, opts.discount, opts.iters)
  File "/Users/aahana/Desktop/School/CAP5636 - Advanced AI/Assignment_1/reinforcement/valueIte
rationAgents.py", line 54, in __init__
    maxVal = -math.inf
AttributeError: 'module' object has no attribute 'inf'
aahana@MacBook-Pro reinforcement %
```

Next, I wanted to print all the actions for a given state. My code at this stage looks as follows:

```
# Write value iteration code here
"*** YOUR CODE HERE ***"
for i in range(self.iterations):
    updated_values = self.values.copy()
    mdp_states = self.mdp.getStates()
    for state in mdp_states:
        # print("Current state: ", state)
        maxVal = float('-inf')
        if mdp.isTerminal(state):
            continue
        else:
            actions = self.mdp.getPossibleActions(state)
            print(actions)
```

Output obtained for one iteration is as follows:

```
aahana@MacBook-Pro reinforcement % clear && python gridworld.py -a value -i 1

DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future rel
ease. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
('Current state: ', 'TERMINAL_STATE')
('Current state: ', (0, 0))
('north', 'west', 'south', 'east')
('Current state: ', (0, 1))
('north', 'west', 'south', 'east')
('Current state: ', (0, 2))
('north', 'west', 'south', 'east')
('Current state: ', (1, 0))
('north', 'west', 'south', 'east')
('Current state: ', (1, 2))
('north', 'west', 'south', 'east')
('Current state: ', (2, 0))
('north', 'west', 'south', 'east')
('Current state: ', (2, 1))
('north', 'west', 'south', 'east')
('Current state: ', (2, 2))
('north', 'west', 'south', 'east')
('Current state: ', (3, 0))
('north', 'west', 'south', 'east')
('Current state: ', (3, 1))
('exit',)
('Current state: ', (3, 2))
('exit',)
*** Method not implemented: computeActionFromValues at line 86 of /Users/aahana/Desktop/School
/CAP5636 - Advanced AI/Assignment_1/reinforcement/valueIterationAgents.py
```

Once all possible actions for a state are found out, the next step is the calculate Q values for all these actions available from a state. For this, I just implemented the Q-value formula in the function computeQValuefromValues and passed the present state and action as parameters to it. To get the Q-value, I first need transition functions (probabilities) and the next states. My code snippet and the corresponding output looks as follows:

```
"*** YOUR CODE HERE ***"
states_and_probs= self.mdp.getTransitionStatesAndProbs(state,action)
print("Transition states and probabilities: ", states_and_probs)
```

```
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future rel
ease. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
('Transition states and probabilities: ', [((0, 1), 0.8), ((1, 0), 0.1), ((0, 0), 0.1)])
*** Method not implemented: computeQValueFromValues at line 83 of /Users/aahana/Desktop/School
/CAP5636 - Advanced AI/Assignment_1/reinforcement/valueIterationAgents.py
aahana@MacBook-Pro reinforcement %
```

Since, states_and_probs is a list with tuple values within it, the transition state and the transition probabilities for each state and action can be printed out.

```
"*** YOUR CODE HERE ***"
states_and_probs = self.mdp.getTransitionStatesAndProbs(state,action)
print("Transition states and probabilities: ", states_and_probs)
for trans_state_and_prob in states_and_probs:
    print("Transition state: ", trans_state_and_prob[0])
    print("Transition probability: ", trans_state_and_prob[1])
```

Output looks as follows:

```
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future rel
ease. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
('Transition states and probabilities: ', [((0, 1), 0.8), ((1, 0), 0.1), ((0, 0), 0.1)])
('Transition state: ', (0, 1))
('Transition probability: ', 0.8)
('Transition state: ', (1, 0))
('Transition probability: ', 0.1)
('Transition state: ', (0, 0))
('Transition probability: ', 0.1)
*** Method not implemented: computeQValueFromValues at line 88 of /Users/aahana/Desktop/School
/CAP5636 - Advanced AI/Assignment_1/reinforcement/valueIterationAgents.py
aahana@MacBook-Pro reinforcement %
```

Next step is to calculate the reward for taking an action from a state and moving to the next state. This can be calculated using the function self.mdp.getReward(). The output after 1 iteration is shown below.

```
"*** YOUR CODE HERE ***"
states_and_probs = self.mdp.getTransitionStatesAndProbs(state,action)
print("Transition states and probabilities: ", states_and_probs)
for trans_state_and_prob in states_and_probs:
    transition_state = trans_state_and_prob[0]
    transition_prob = trans_state_and_prob[1]
    print("Transition state: ", transition_state)
    print("Transition probability: ", transition_prob)
    reward = self.mdp.getReward(state,action,transition_state)
    print("Reward obtained: ", reward)
    print("\n")
```

```
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future
release. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
('Transition states and probabilities: ', [((0, 1), 0.8), ((1, 0), 0.1), ((0, 0), 0.1)])
('Transition state: ', (0, 1))
('Transition probability: ', 0.8)
('Reward obtained: ', 0.0)


('Transition state: ', (1, 0))
('Transition probability: ', 0.1)
('Reward obtained: ', 0.0)


('Transition state: ', (0, 0))
('Transition probability: ', 0.1)
('Reward obtained: ', 0.0)


*** Method not implemented: computeQValueFromValues at line 91 of /Users/aahana/Desktop/Sch
ool/CAP5636 - Advanced AI/Assignment_1/reinforcement/valueIterationAgents.py
```

Using the formula for Q-Value: Q-value = summation over s'{T(s,a,s') [R(s,a,s') + discount * V(s')]}. However, this gave me an error for not initializing Q-value = 0 in the beginning. Once, done, the output for Q-value given a state and action was calculated.

```
"*** YOUR CODE HERE ***"
states_and_probs = self.mdp.getTransitionStatesAndProbs(state,action)
# print("Transition states and probabilities: ", states_and_probs)
Q_value = 0
for trans_state_and_prob in states_and_probs:
    transition_state = trans_state_and_prob[0]
    transition_prob = trans_state_and_prob[1]
    # print("Transition state: ", transition_state)
    # print("Transition probability: ", transition_prob)
    reward = self.mdp.getReward(state, action, transition_state)
    # print("Reward obtained: ", reward)
    # print("\n")
    ## Q - Value formula: summation over s'[T(s,a,s'){R(s,a,s') + discount * V(s')}]
    Q_value += transition_prob * (reward + self.discount * self.getValue(transition_state))
    print("Q value for the state", transition_state, " and action ", action, "is: ", Q_value)
```

```
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future
release. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
('Q value for the state', (0, 1), ' and action ', 'north', 'is: ', 0.0)
('Q value for the state', (1, 0), ' and action ', 'north', 'is: ', 0.0)
('Q value for the state', (0, 0), ' and action ', 'north', 'is: ', 0.0)
*** Method not implemented: computeQValueFromValues at line 96 of /Users/aahana/Desktop/Sch
ool/CAP5636 - Advanced AI/Assignment_1/reinforcement/valueIterationAgents.py
aahana@MacBook-Pro reinforcement %
```

Once Q-values are calculated, we find the max Q-value. However, the output did not look correct.

```
    "*** YOUR CODE HERE ***"
    for i in range(self.iterations):
        updated_value = self.values.copy()
        mdp_states = self.mdp.getStates()
        for state in mdp_states:
            # print("Current state: ", state)
            maxVal = float('-inf')
            if mdp.isTerminal(state):
                continue
            else:
                actions = self.mdp.getPossibleActions(state)
                # print(actions)
                for action in actions:
                    value = self.computeQValueFromValues(state, action)
                    # print("Action and value from state :", action, value)
                    if value > maxVal:
                        maxVal = value
                        updated_value[state] = maxVal
                        print("updated_value[state]: ", updated_value[state])
                        # self.actions[state] = action
                        # print("action[state]: ", action)
    self.values = updated_value
    print("Self.values: ", self.values)
```

```
('Self.values: ', {(0, 1): 0.0, (1, 2): 0.0, (3, 2): 1.0, (0, 0): 0.0, (3, 0): 0.0, (3, 1):
-1.0, (2, 1): 0.0, (2, 0): 0.0, (2, 2): 0.0, (1, 0): 0.0, 'TERMINAL_STATE': 0, (0, 2): 0.0})
```

I realized that the indentation was wrong and that I had to update the value of the state only after checking all possible actions from that state. Also, the state values have to be updated at the end of every iteration. Once these errors were fixed, the code and the outputs are as follows:

```
        "*** YOUR CODE HERE ***"
        for i in range(self.iterations):
            updated_value = self.values.copy()
            mdp_states = self.mdp.getStates()
            for state in mdp_states:
                # print("Current state: ", state)
                maxVal = float('-inf')
                if mdp.isTerminal(state):
                    continue
                else:
                    actions = self.mdp.getPossibleActions(state)
                    # print(actions)
                    for action in actions:
                        value = self.computeQValueFromValues(state, action)
                        # print("Action and value from state :", action, value)
                        if value > maxVal:
                            maxVal = value
                    updated_value[state] = maxVal
                    print("updated_value[state]: ", updated_value[state])
            self.values = updated_value
            print("States and the corresponding values: ", self.values)
```

```
('States and the corresponding values: ', {(0, 1): 0.26873856000000007, (1, 2): 0.71
55216000000002, (3, 2): 1.0, (0, 0): 0.0, (3, 0): 0.13208256000000002, (3, 1): -1.0,
 (2, 1): 0.5532404400000002, (2, 0): 0.36980064000000007, (2, 2): 0.8408520000000002
, (1, 0): 0.22208256000000004, (0, 2): 0.5076172800000002})
```

As seen above, the values obtained match the values that need to be obtained by the 5th iteration.

Next, I need to be able to generate the gridworld to show the values that I have generated. So, I need to work on the last function computeActionFromValues(). For this, I do the same process that I did for finding the largest value of a state. However, this time, I want to record the action that gives the largest value for a state.

```
"*** YOUR CODE HERE ***"
maxVal = float('-inf')
if not self.mdp.isTerminal(state):
    actions = self.mdp.getPossibleActions(state)
    # print(actions)
    for action in actions:
        value = self.computeQValueFromValues(state, action)
        # print("Action and value from state :", action, value)
        if value > maxVal:
            maxVal = value
            state_action = action
return state_action
```

I got an error for not defining ‘state_action’, hence I had to state_action to a default value as “north”.

```
  File "/Users/aahana/Desktop/School/CAP5636 - Advanced AI/Assignment_1/reinforce
ment/valueIterationAgents.py", line 121, in computeActionFromValues
    return state_action
UnboundLocalError: local variable 'state_action' referenced before assignment
```

At the end of this, the output is obtained:

Question 2:

The purpose of this question is to modify either the discount or the noise to be able to send the agent across the bring to the +10 reward. Currently, with a noise of 0.1 and discount of 0.9, the output looks as follows:

I changed the discount values a couple times to see if I could get the agent across the bridge. Some of the outputs look as follows:

When noise is 0.1 and discount is 0.8 and discount = 0.6 is shown below:

Then, noise = 0.1 and I reduced discount even further to 0.5. The outputs look as below:

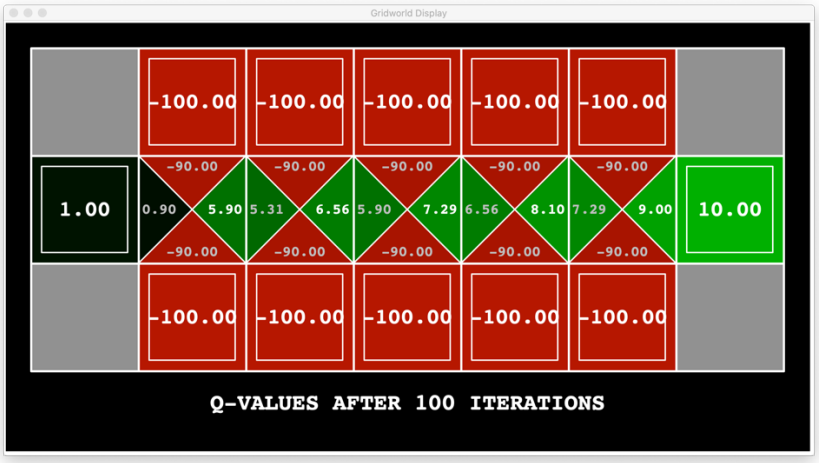
Once I noticed that changing discount is not allowing the agent to get across the bridge, I decided to change the noise. We want the noise to be as low as possible to have the agent get across the bridge. I started reducing noise by small numbers. Output when discount = 0.9 and noise is 0.07 and noise = 0.05 is as follows:

Output when discount = 0.9 and noise 0.01 is as shown below:

Ideally, we could just set noise to 0.0 and discount to 0.9 and have the agent cross the bridge. The output would look as shown below:


```
def question2():
    answerDiscount = 0.9
    answerNoise = 0.0
    return answerDiscount, answerNoise
```

The corresponding Q – values for noise = 0.0 and discount = 0.9 is as shown below:



For all values of noise below 0.016, the agent moves across the bridge if it were to follow the optimal policy indicated by the arrow marks. The output when **discount = 0.9 and noise = 0.016** is as shown below:

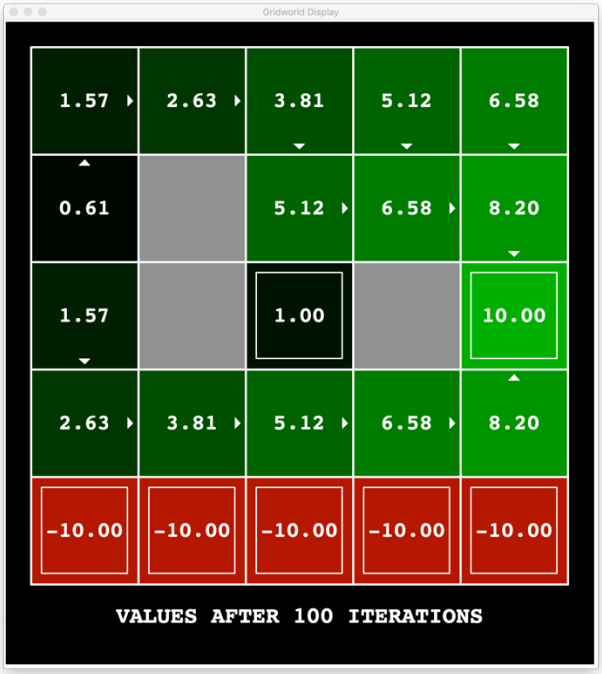
Question 3:

For this question, I randomly started giving values for discount, noise and living reward. Some of the different combination of values for the parameters that I tried, and the corresponding outputs are below.

Try 1: discount = 0.9, noise = 0.0 and living reward = 0.0

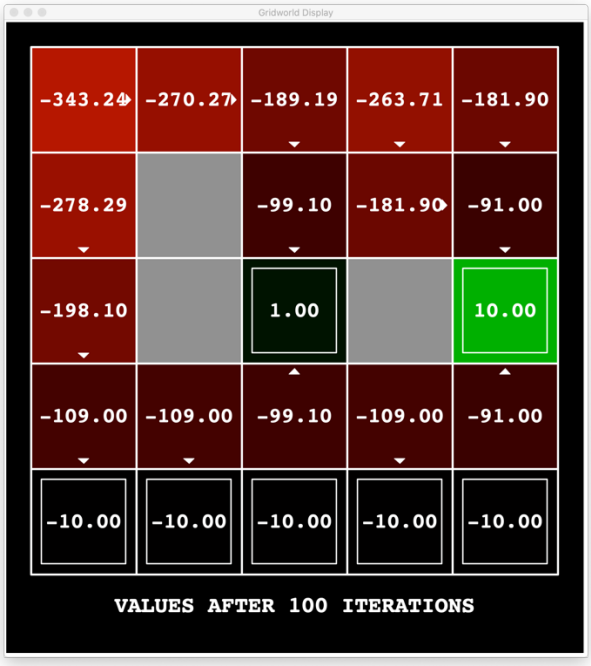
If I want the agent to take the shorter path along the firepits, then I want to make the living reward negative, in a way that the agent is motivated to take the shorter path.

Try 2: discount = 0.9, noise = 0.0 and living reward = -0.8



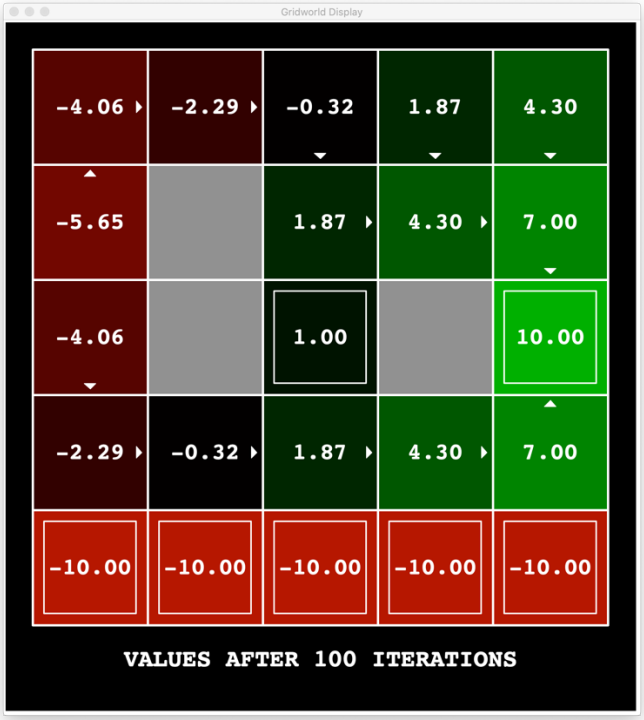
Clearly, I am on the right track, I just have to decrease the living reward even further.

Try 3: discount = 0.9, noise = 0.0 and living reward = -100



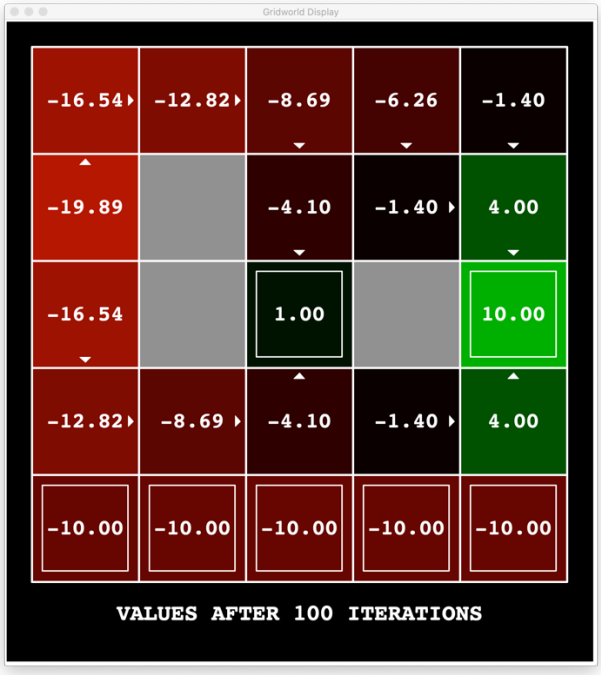
That was way too low of a living reward.

Try 4: discount = 0.9, noise = 0.0 and living reward = -2



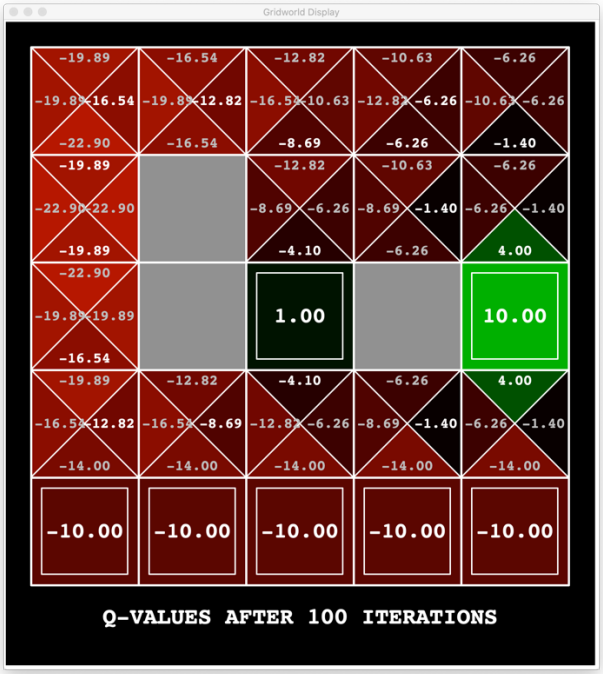
Question 3a:

Try 5: **discount = 0.9, noise = 0.0 and the living reward = -5**



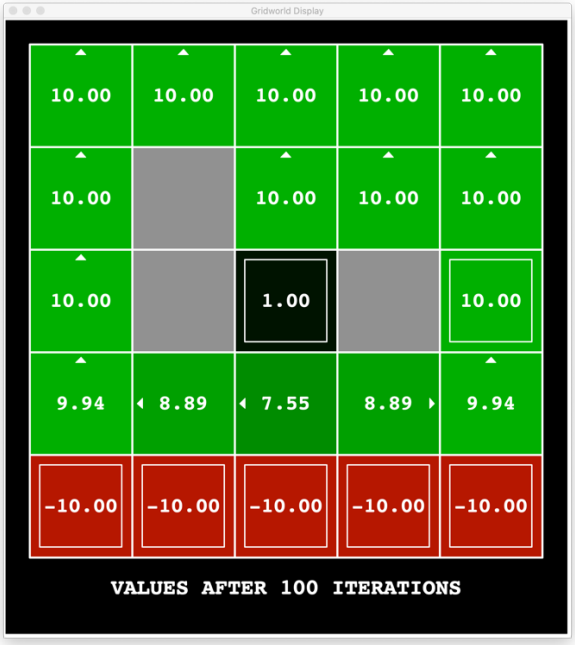
The living reward is still too low but the desired policy for question 3a is achieved. The agent moves along the firepits and takes the reward +1, when discount was set to 0.9, noise = 0.0 and

living reward = -5.0. Similar output was obtained when living reward was set -4.0. The Q-values for this are shown below:



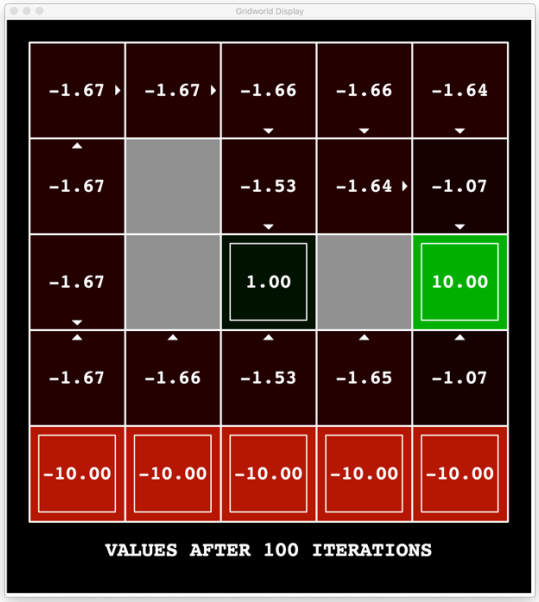
To find the policy that takes us away from the cliff, I set the living reward low as it doesn't matter how many steps the agent takes to reach the final reward. Here are some tries that I made to achieve the desired policy.

Try 1: discount = 1.0, noise = 0.1 and living reward = 0.0



This kind of achieves the purpose by moving away from the fire pits and taking the longer path. However, it does not take the agent to reward = +1.

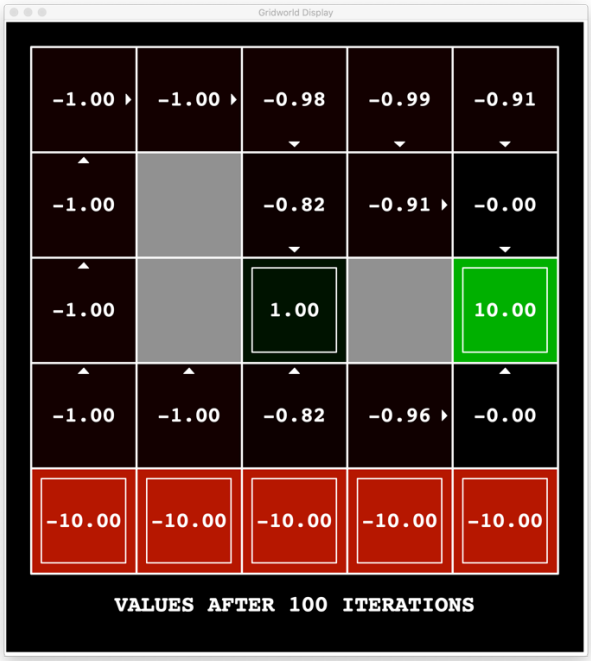
Try 2: discount = 0.1, noise = 0.5 and living reward = -1.5



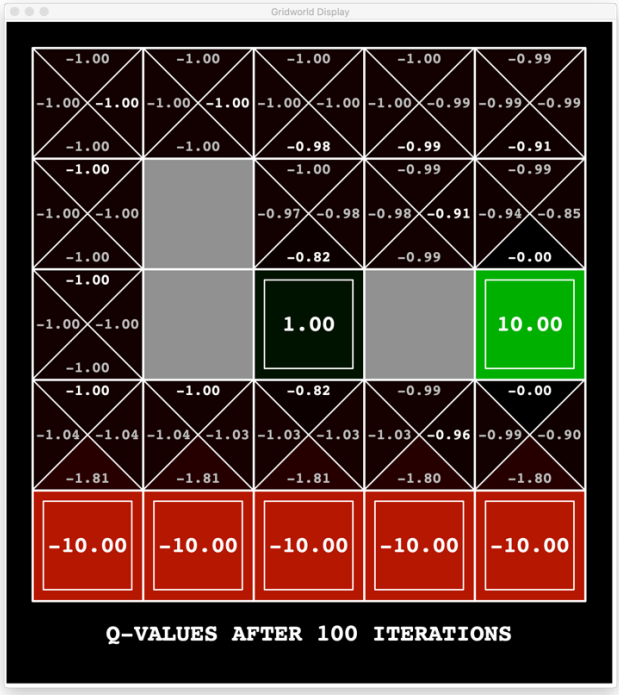
I then tried to reduce the noise and the living reward a bit and I ended up with the policy we were looking for.

Question 3b:

Try 3: **discount = 0.1, noise = 0.1 and living reward = -0.9**



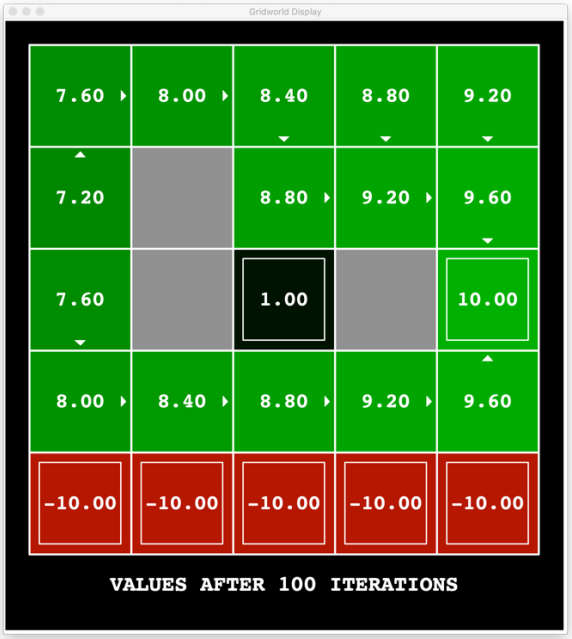
The corresponding Q-values are shown below:



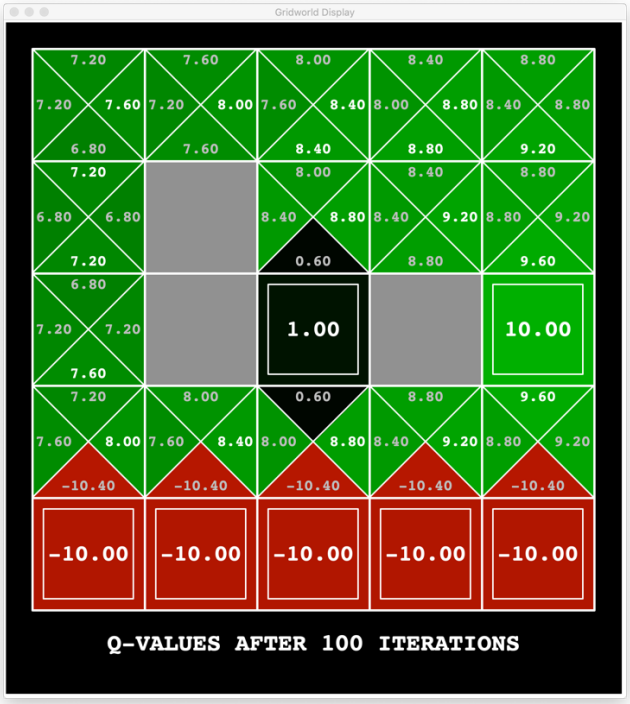
During one of my previous tries, I accidently found the values for discount, noise and reward that would have the agent to the reward +10 by moving along the firepits. I immediately made a note of the parameters.

Question 3c:

Try 1: **discount = 1.0, noise = 0.0, living reward = -0.4**

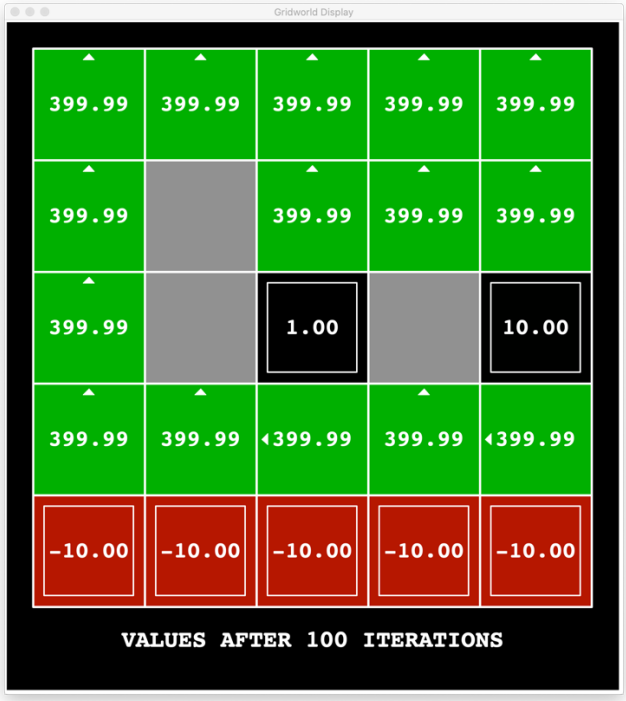


The corresponding Q-values are:



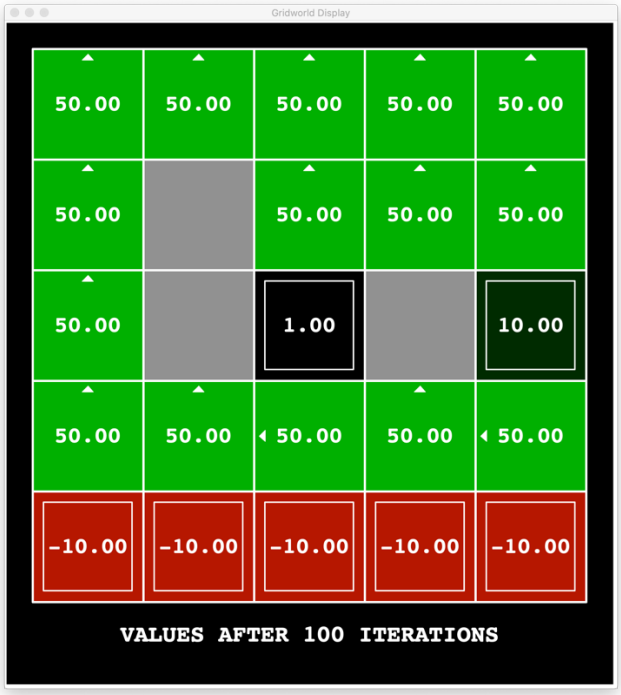
Next, to get to the furthest reward and to avoid the firepits to take the longer route, I decided to set a high positive living reward.

Try 1: discount = 0.9, noise = 0.0, living reward = 40

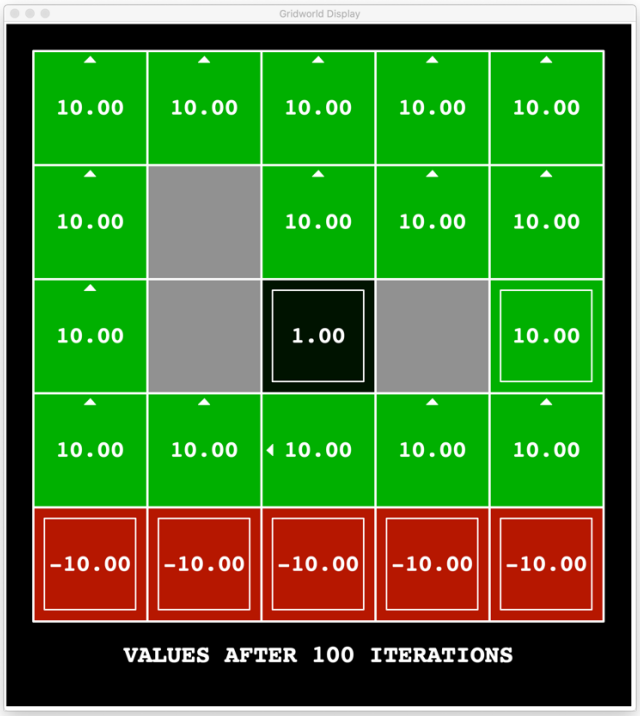


I figured I was in the right direction but just had a high living reward. Hence, I reduced the living reward down.

Try 2: discount = 0.9, noise = 0.0, living reward = 5

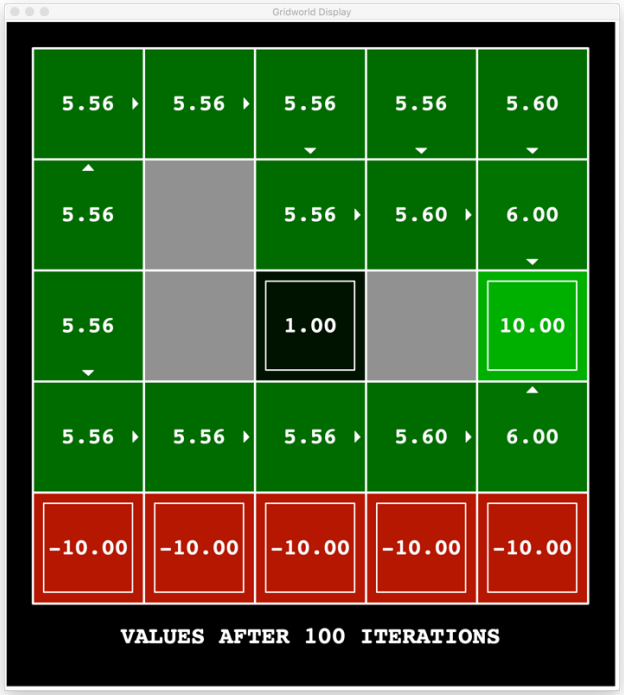


Try 3: discount = 0.5, noise = 0.0, living reward = 5

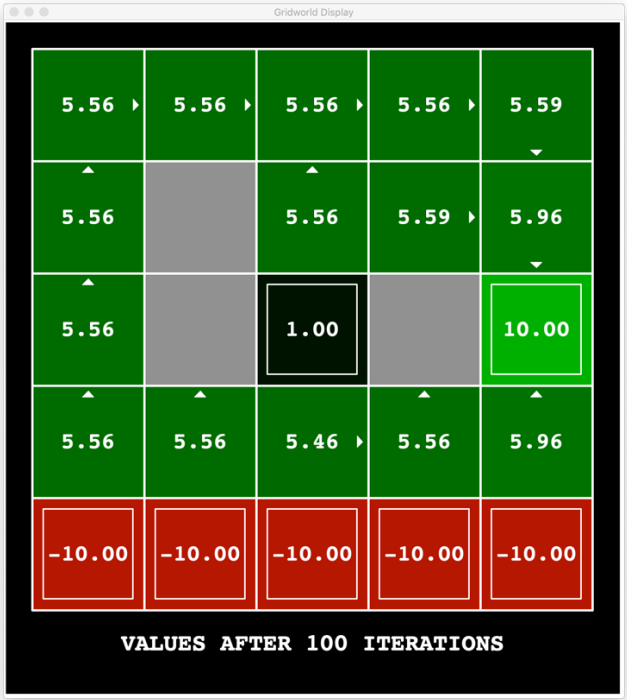


Next, I decided to change the discount and noise a bit to get the desired policy.

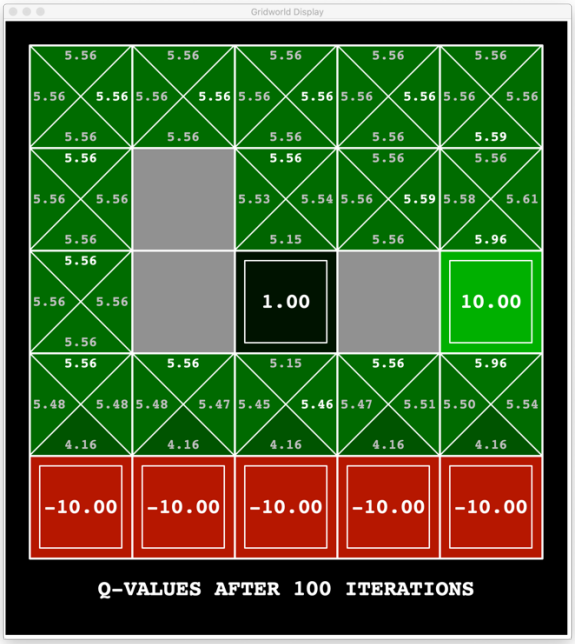
Try 4: discount = 0.1, noise = 0.0, living reward = 5



Question 3d:
Try 5: **discount = 0.1, noise = 0.1, living reward = 5.0**

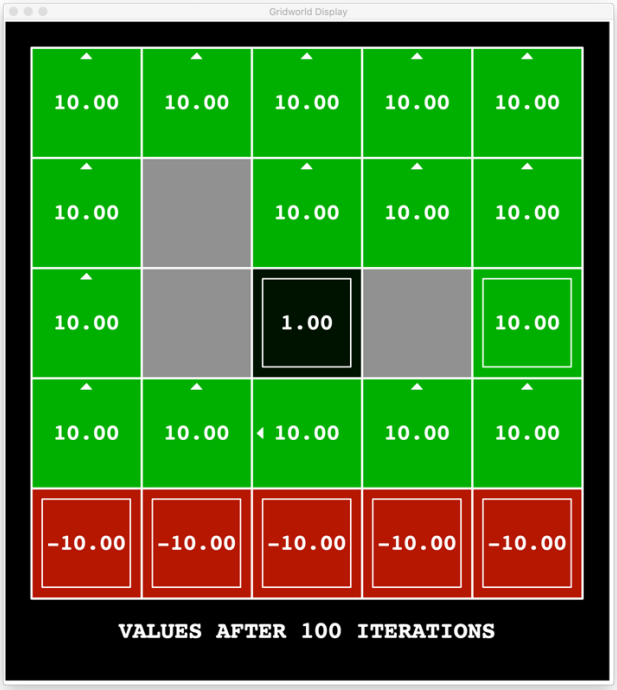


The corresponding Q-values for these parameters are as follows:

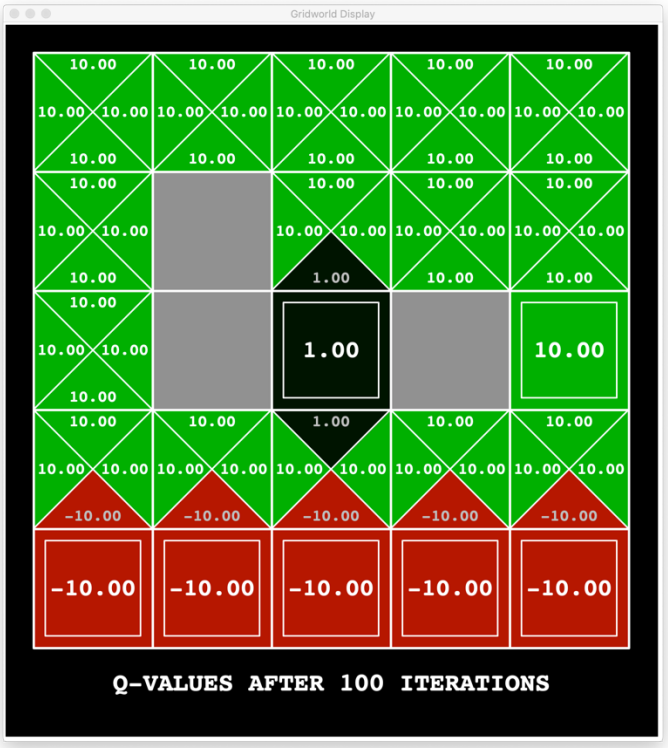


I was able to send the agent on an infinite loop by setting discount to 1.0, noise to 0.0 and living reward to 0.0. These parameters were found during one of the several combinations that I tried on the *discountgrid*. For these parameters, the agent does not get any reward.

Question 3e:
Try 1: **discount = 1.0, noise = 0.0 and living reward = 0.0**



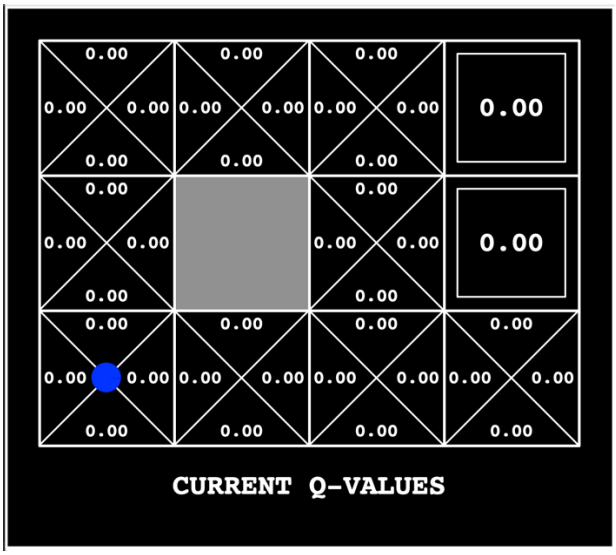
The corresponding Q-values for this grid is as follows:




```
aahana@MacBook-Pro reinforcement % python analysis.py
Answers to analysis questions:
  Question question2:   (0.9, 0.0)
  Question question3a:  (0.9, 0.0, -5.0)
  Question question3b:  (0.1, 0.1, -0.9)
  Question question3c:  (1.0, 0.0, -0.4)
  Question question3d:  (0.1, 0.1, 5.0)
  Question question3e:  (1.0, 0.0, 0.0)
  Question question6:   (None, None)
```

Question 4:

Initially, Q-values look as follows:



Similar to question 1, first I calculate the maximum value for every action in a given state.

```
def computeValueFromQValues(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"
    maxVal = float('-inf')
    actions = self.getLegalActions(state)
    # print("Actions: ", actions)
    for action in actions:
      q_value = self.getQValue(state, action)
      if q_value > maxVal:
        maxVal = q_value
    return maxVal

    util.raiseNotDefined()
```

Similarly, I got the action that gives the maximum Q value.

```
def computeActionFromQValues(self, state):
    """
      Compute the best action to take in a state.  Note that if there
      are no legal actions, which is the case at the terminal state,
      you should return None.
    """
    "*** YOUR CODE HERE ***"
    maxVal = float('-inf') # set default values
    maxAct = "north"
    for action in actions:
      q_value = self.getQValue(state, action)
      if q_value > maxVal:
        maxAct = action
    return maxAct

    util.raiseNotDefined()
```

Next, we have to update the values of the Q-learner. For this, I used the formula:

Q value = (1 - alpha)*current_Qvalue + alpha*[reward + discount*Qvalue(next_state)]

```
def update(self, state, action, nextState, reward):
    """
      The parent class calls this to observe a
      state = action => nextState and reward transition.
      You should do your Q-Value update here

      NOTE: You should never call this function,
      it will be called on your behalf
    """
    "*** YOUR CODE HERE ***"
    # Q_value formula:
    # (1-alpha)*current_Qvalue + alpha*[reward + discount*Qvalue(next_state)]
    current_Qvalue = self.getQValue(state, action)
    next_state_Qvalue = self.computeValueFromQValues(nextState)
    updated_Qvalue = (1-self.alpha)*current_Qvalue + self.alpha(reward + self.discount * next_state_Qvalue)
    self.q_val[(state,action)] = updated_Qvalue

    util.raiseNotDefined()
```

Next, to get random actions for a state, I used random.choice:

```
def getAction(self, state):
    """
      Compute the action to take in the current state.  With
      probability self.epsilon, we should take a random action and
      take the best policy action otherwise.  Note that if there are
      no legal actions, which is the case at the terminal state, you
      should choose None as the action.

      HINT: You might want to use util.flipCoin(prob)
      HINT: To pick randomly from a list, use random.choice(list)
    """
    # Pick Action
    legalActions = self.getLegalActions(state)
    action = None
    "*** YOUR CODE HERE ***"
    random_action = util.flipCoin(self.epsilon)
    if random_action:
      # pick a random action from the set of actions.
      action = random.choice(legalActions)
      print("The randomly picked action is: ", action)
    else:
      action = self.computeActionFromQValues(state)

    return action

    util.raiseNotDefined()
```

Once done, I initialize Q value and find Q values within getQValue function.

```
def __init__(self, **args):
    "You can initialize Q-values here..."
    ReinforcementAgent.__init__(self, **args)

    "*** YOUR CODE HERE ***"
    self.q_val = util.Counter() # dictionary with values for key = 0


def getQValue(self, state, action):
    """
      Returns Q(state,action)
      Should return 0.0 if we have never seen a state
      or the Q node value otherwise
    """
    "*** YOUR CODE HERE ***"
    return self.q_val[(state,action)] # list of tuples

    util.raiseNotDefined()
```

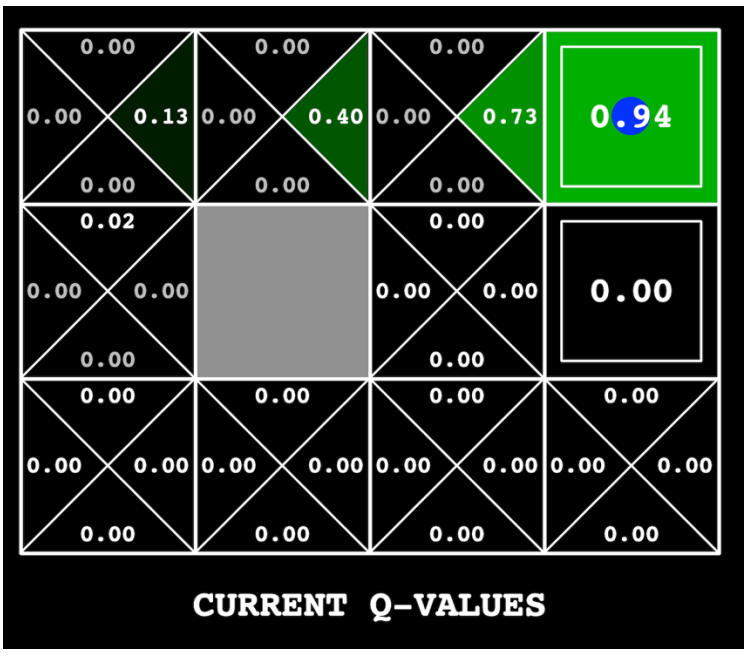
Once all of this was done, I executed the code and got the following error:

```
  File "/Users/aahana/Desktop/School/CAP5636 - Advanced AI/Assignment_1
/reinforcement/qlearningAgents.py", line 138, in update
    updated_Qvalue = (1-self.alpha)*current_Qvalue + self.alpha(reward
+ self.discount * next_state_Qvalue)
TypeError: 'float' object is not callable
aahana@MacBook-Pro reinforcement %
```

The error was fixed by putting a multiplication symbol after self.alpha in the formula implementation. However, I had another error that I got right after the first iteration.

```
Traceback (most recent call last):
  File "gridworld.py", line 570, in <module>
    returns += runEpisode(a, env, opts.discount, decisionCallback, displayCallback, messageCallback, pauseCallback, episode)
  File "gridworld.py", line 350, in runEpisode
    display(state)
  File "gridworld.py", line 546, in <lambda>
    if opts.agent == 'q': displayCallback = lambda state: display.displayQValues(a, state, "CURRENT Q-VALUES")
  File "/Users/aahana/Desktop/School/CAP5636 - Advanced AI/Assignment_1/reinforcement/graphicsGridworldDisplay.py", line 58, in displayQValues
    drawQValues(self.gridworld, qValues, currentState, message)
  File "/Users/aahana/Desktop/School/CAP5636 - Advanced AI/Assignment_1/reinforcement/graphicsGridworldDisplay.py", line 163, in drawQValues
    drawSquare(x, y, value, minValue, maxValue, valString, action, False, isExit, isCurrent)
  File "/Users/aahana/Desktop/School/CAP5636 - Advanced AI/Assignment_1/reinforcement/graphicsGridworldDisplay.py", line 215, in drawSquare
    square_color = getColor(val, min, max)
  File "/Users/aahana/Desktop/School/CAP5636 - Advanced AI/Assignment_1/reinforcement/graphicsGridworldDisplay.py", line 328, in getColor
    return formatColor(r,g,0.0)
  File "/Users/aahana/Desktop/School/CAP5636 - Advanced AI/Assignment_1/reinforcement/graphicsUtils.py", line 36, in formatColor
    return '#%02x%02x%02x' % (int(r * 255), int(g * 255), int(b * 255))
ValueError: cannot convert float NaN to integer
aahana@MacBook-Pro reinforcement %
```

To fix this, I had to check the length of actions inside computeValueFromQValues function. Once, that was fixed, I got the output I was looking for. My output just before exiting the 5th iteration looks as below. Upon exit, the exit state get 0.97.

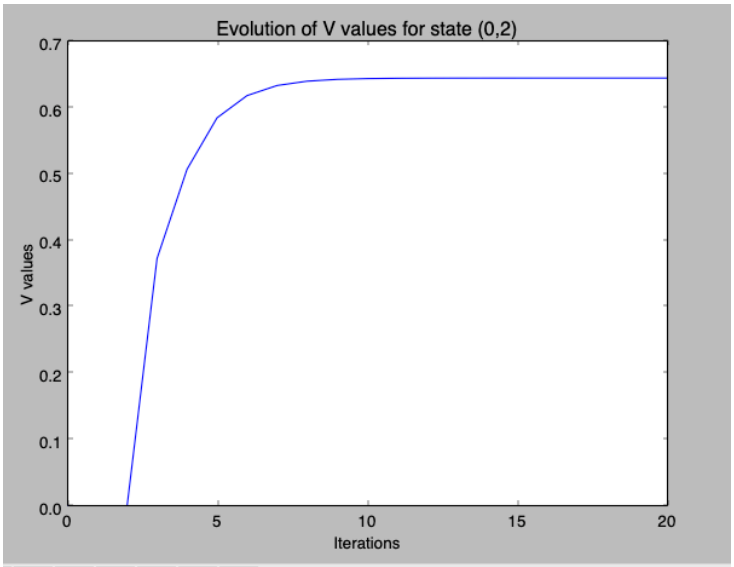


STEP 2:

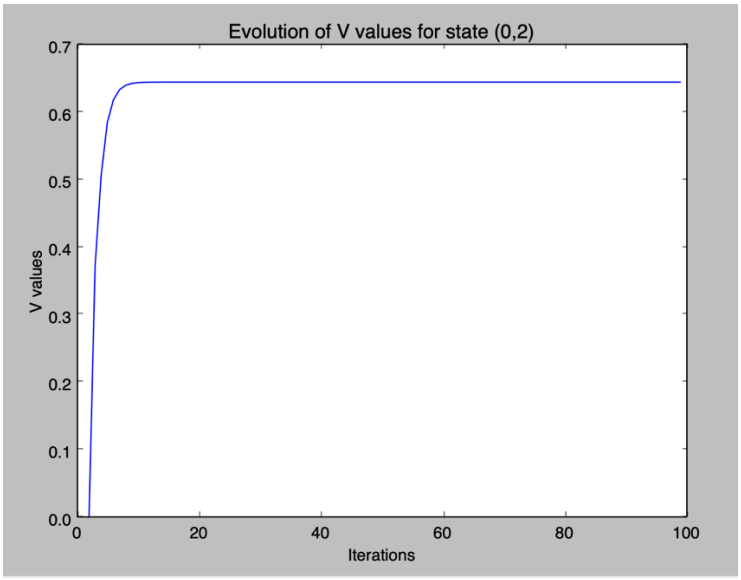
To plot the V-value evolution for the state (0,2), the valueIterationAgents.py was modified to store V-values in a list and plot them. The code snippet is shown below:

```
# Write value iteration code here
"*** YOUR CODE HERE ***"
V_values = []
from matplotlib import pyplot as plt
for i in range(self.iterations):
    updated_value = self.values.copy()
    mdp_states = self.mdp.getStates()
    for state in mdp_states:
        # print("Current state: ", state)
        maxVal = float('-inf')
        if self.mdp.isTerminal(state):
            continue
        else:
            actions = self.mdp.getPossibleActions(state)
            for action in actions:
                value = self.computeQValueFromValues(state, action)
                # print("Action and value from state :", action, value)
                if value > maxVal:
                    maxVal = value
            updated_value[state] = maxVal
            # print("updated_value[state]: ", updated_value[state])
        if state == (0,2):
            V_values.append(updated_value[state])
            print("The values of the state (0,2) are: ", V_values)
    self.values = updated_value
    # print("States and the corresponding values: ", self.values)
plt.plot(V_values)
plt.xlim(0, 20)
plt.xlabel("Iterations")
plt.ylabel("V values")
plt.title("Evolution of V values for state (0,2)")
plt.show()
```

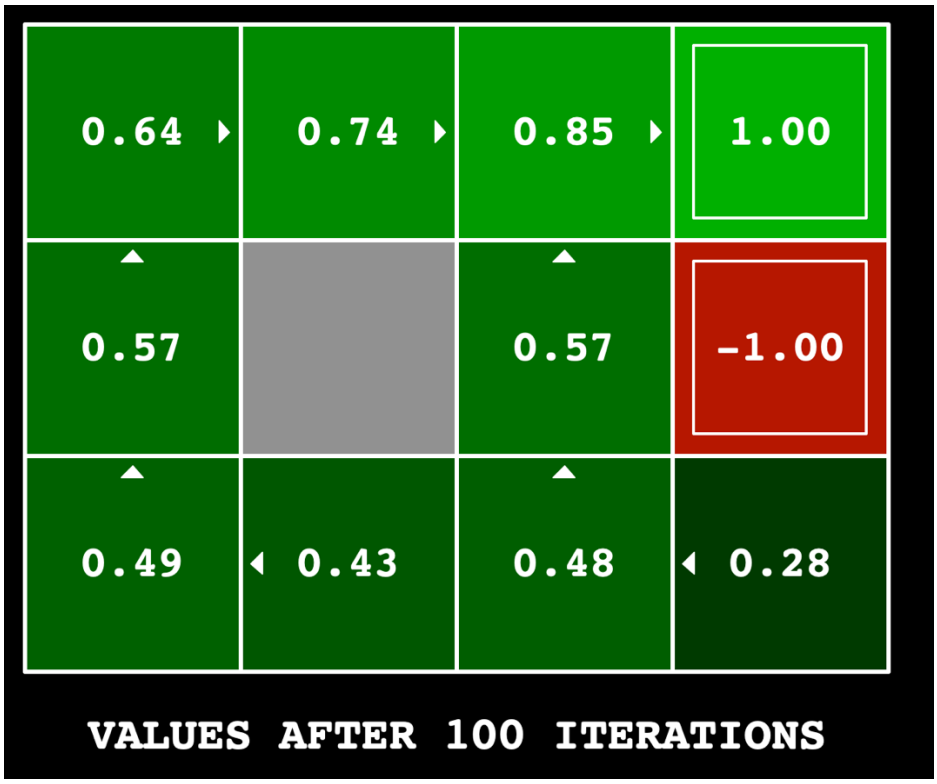
When X-axis is limited to 20, the output looks as follows:



When X-axis goes until 100, the output looks as follows:



The values after 100 iterations is as follows:

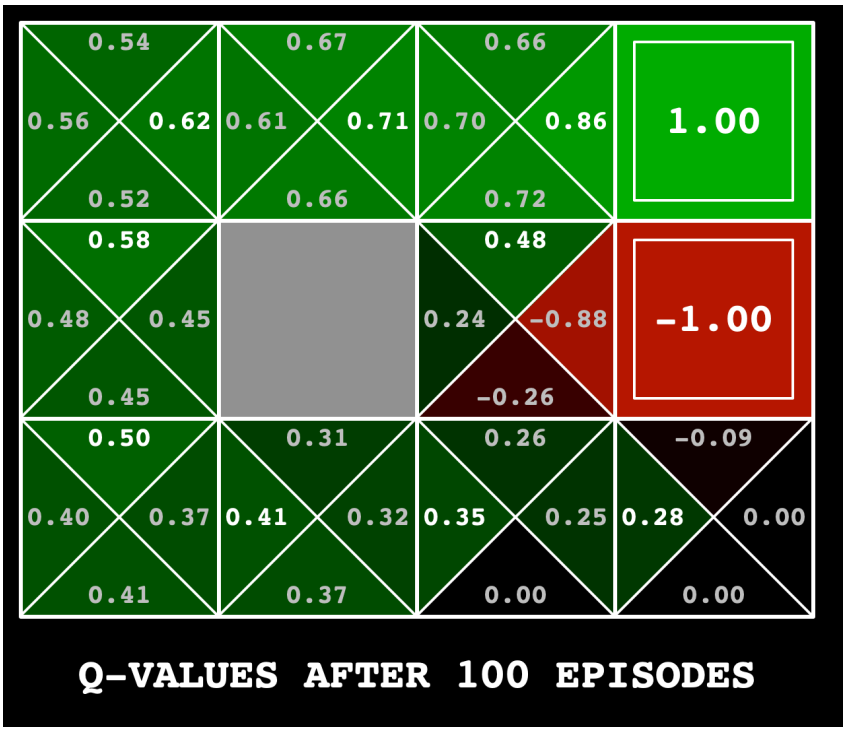


STEP 3:

To understand the evolution of Q-values of a state, I modified the update function in the qlearningAgents.py file. The code snippet looks as follows:

```
current_Qvalue = self.getQValue(state, action)
next_state_value = self.computeValueFromQValues(nextState)
updated_Qvalue = (1-self.alpha)*current_Qvalue + self.alpha * (reward + self.discount * next_state_value)
self.q_val[(state,action)] = updated_Qvalue
# print("final answer to the update: ", self.q_val[(state,action)])

if state == (1,2):
  self.q_actions[action].append(self.q_val[state,action])

  for other_action in self.q_actions.keys():
    if other_action != action:
      if self.q_actions[other_action] == []:
        # append 0 if the list is empty
        self.q_actions[other_action].append(0.0)
      else:
        # append the last value for that action that is already in the list
        self.q_actions[other_action].append(self.q_actions[other_action][-1])
  print("The dictionary values are: ", self.q_actions)

  # creating a 2D list for excel data dump
  csv_output = [["episode", "north", "south", "east", "west"]] # first row of the csv
  for row in range(len(self.q_actions['east'])):
    # append value for each action based on the current iteration (row)
    csv_output.append([row+1, self.q_actions['north'][row], self.q_actions['south'][row], self.q_actions['east'][row], self.q_actions['west'][row]])
  with open("step3_output.csv","w+") as my_csv:
    csvWriter = csv.writer(my_csv,delimiter=',')
    csvWriter.writerows(csv_output)
```

I dump the Q-values for all four actions (North, South, East and West) for the state (1,2) into a csv file. The output with this code after 100 iterations is as follows. However, I noticed that the csv file has 200 values (rows) for each action. This is not right as I should just have 100 values



Last few rows of the step3_output.csv file for one the iterations is as follows:

186	0.66771861	0.64474781	0.6758157	0.6147803
187	0.66771861	0.64474781	0.70006618	0.6147803
188	0.66771861	0.64474781	0.73361225	0.6147803
189	0.66771861	0.65249942	0.73361225	0.6147803
190	0.66771861	0.65249942	0.76109571	0.6147803
191	0.66771861	0.65249942	0.77483744	0.6147803
192	0.66771861	0.65249942	0.78974111	0.6147803
193	0.66771861	0.72991091	0.78974111	0.6147803
194	0.66771861	0.72991091	0.7110699	0.6147803
195	0.66771861	0.69341536	0.7110699	0.6147803
196	0.66771861	0.69341536	0.7110699	0.61108563
197	0.66771861	0.69341536	0.68029893	0.61108563
198	0.66771861	0.65874459	0.68029893	0.61108563
199	0.66771861	0.65874459	0.67676464	0.61108563
200	0.66771861	0.65874459	0.70918991	0.61108563

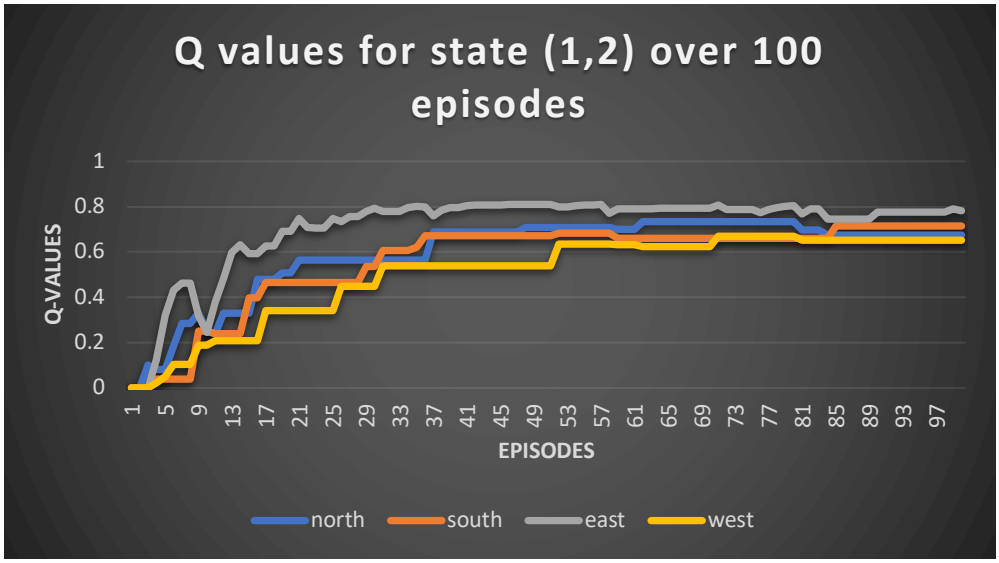
I had to change the entire code and the way I was recording the action values for the state. I decided to save the Q-values only when the agent reached the terminal state. That way, I would get the Q-values for state (1,2) only at the end of each episode. The modified code is below.

```
"*** YOUR CODE HERE ***"
# Q_value formula:
# (1-alpha)*current_Qvalue + alpha*[reward + discount*Qvalue(next_state)]

current_Qvalue = self.getQValue(state, action)
next_state_value = self.computeValueFromQValues(nextState)
updated_Qvalue = (1-self.alpha)*current_Qvalue + self.alpha * (reward + self.discount * next_state_value)
self.q_val[(state,action)] = updated_Qvalue
# print("final answer to the update: ", self.q_val[(state,action)])

if state == (3,2) or state == (3,1):
  self.q_actions['north'].append(self.q_val[((1,2),'north')])
  self.q_actions['south'].append(self.q_val[((1,2),'south')])
  self.q_actions['east'].append(self.q_val[((1,2),'east')])
  self.q_actions['west'].append(self.q_val[((1,2), 'west')])

  # creating a 2D list for excel data dump
csv_output = [["episode", "north", "south", "east", "west"]] # first row of the csv
for row in range(len(self.q_actions['east'])):
  # append value for each action based on the current iteration (row)
  csv_output.append([row+1, self.q_actions['north'][row], self.q_actions['south'][row], self.q_actions['east'][row], self.q_actions['west'][row]])
with open("step3_output.csv","w+") as my_csv:
  csvWriter = csv.writer(my_csv,delimiter=',')
  csvWriter.writerows(csv_output)
```

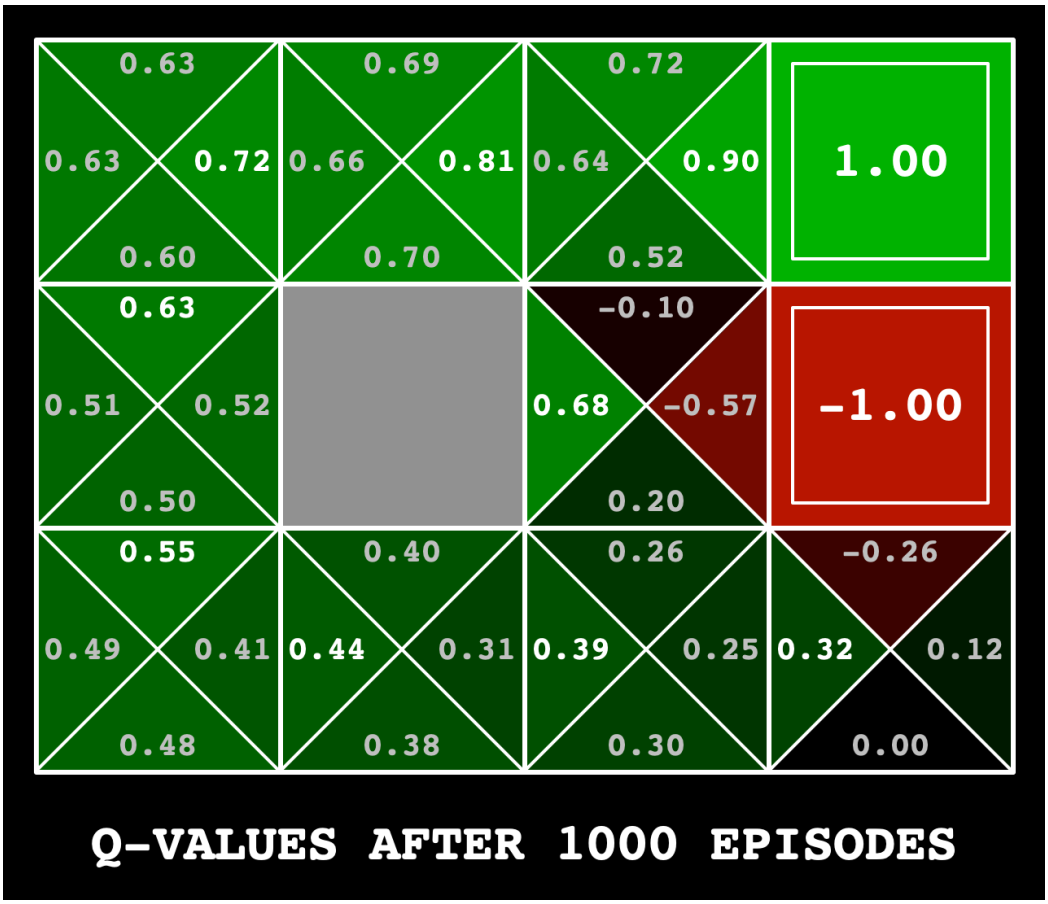
With this, I was able to get Q-values for the state (1,2) at the of each episode. The last few rows of step3_output.csv looks as shown below:

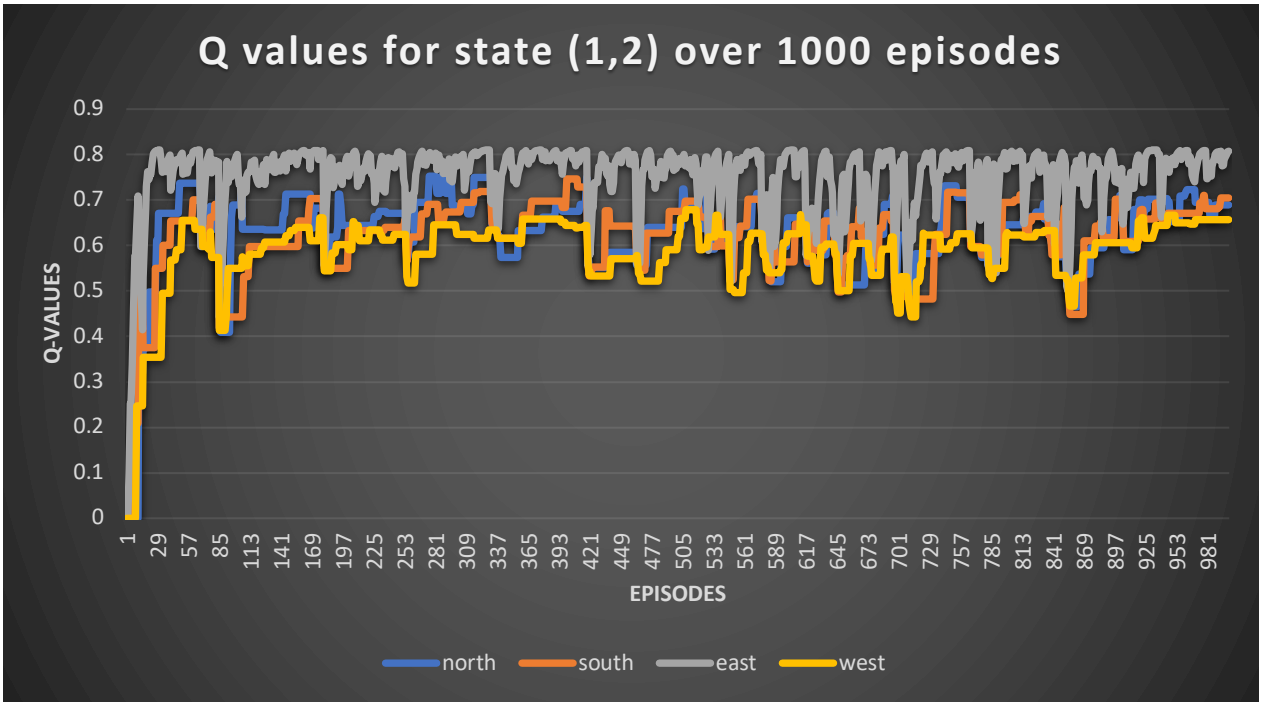
96	0.67334165	0.71477996	0.77653015	0.65088762
97	0.67334165	0.71477996	0.77653015	0.65088762
98	0.67334165	0.71477996	0.77653015	0.65088762
99	0.67334165	0.71477996	0.79069762	0.65088762
100	0.67334165	0.71477996	0.78327219	0.65088762

Now that I have the right values in the excel, I was able to plot them. The plot showing Q-values for 100 episodes is shown below:



Below is the plot showing Q-values for 1000 episodes:





GitHub link: Reinforcement_Learning

https://github.com/monicabernard/CAP5636_Advanced_AI.git