Intelligent Agents: Assignment 1

Author: Samarth Agarwal

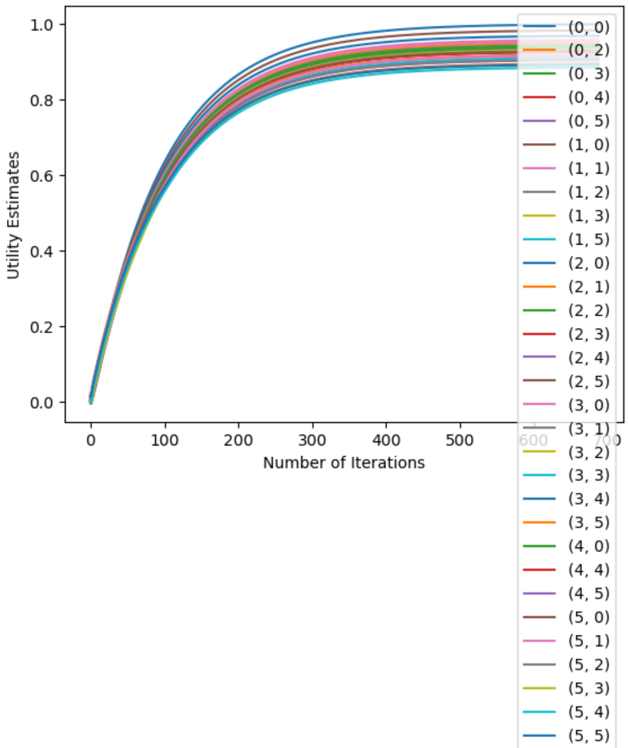
Part 1: Value Iteration

1.1) Brief description of implemented solution:

Note: Description omits information on the action array which saves the best policy and the dictionary dic which saves the Utility Values for each states and iteration, since those are just used for plotting purposes and not in the value iteration algorith.

For Value Iteration, we first create the reward grid and the utility grid. After this we iterate through the grid calculating the expected utility for that state (U[s']). This can be seen through **Fig 1.1**. After which we find the maximum utility from the possible moves that the agent can make. We use Bellman Equation to calculate the expected utility and save it in the Utility Grid. Before the end of the iteration we record the maximum change in utility for that iteration. After the end of the iteration we check if the Value Iteration is converging. We use an epsilon value of 0.1 when checking for convergence with the discount factor of 0.99 as given in the problem. This can be seen in **Fig 1.2**.

Lastly, we display the Utility of all the states and the Plot of the Utility Estimates and the Plot of utility estimates (right) as a function of the number of iterations (left) as shown below.




```
Coordinates in col, row
( 0 , 0 ) utility:  99.90068522040848 action:  up
( 0 , 1 ) utility:  98.29517945765397 action:  up
( 0 , 2 ) utility:  96.85143475909933 action:  up
( 0 , 3 ) utility:  95.4578641369429 action:  up
( 0 , 4 ) utility:  94.21750419704358 action:  up
( 0 , 5 ) utility:  92.84352668876943 action:  up
( 1 , 1 ) utility:  95.78595196220697 action:  left
( 1 , 2 ) utility:  95.49044080660859 action:  left
( 1 , 3 ) utility:  94.35747726945647 action:  left
( 1 , 5 ) utility:  91.63576947780736 action:  left
( 2 , 0 ) utility:  94.94959982740707 action:  left
( 2 , 1 ) utility:  94.44914096216962 action:  left
( 2 , 2 ) utility:  93.19951052802934 action:  left
( 2 , 3 ) utility:  93.13848999362497 action:  left
( 2 , 5 ) utility:  90.44307390309548 action:  left
( 3 , 0 ) utility:  93.78012700420777 action:  left
( 3 , 1 ) utility:  94.30303956313563 action:  left
( 3 , 2 ) utility:  93.08263620702817 action:  up
( 3 , 3 ) utility:  91.0226301768361 action:  up
( 3 , 5 ) utility:  89.26525214048898 action:  left
( 4 , 0 ) utility:  92.56068919803865 action:  left
( 4 , 2 ) utility:  93.00979699821708 action:  left
( 4 , 3 ) utility:  91.72286198573936 action:  up
( 4 , 4 ) utility:  89.45789032678049 action:  up
( 4 , 5 ) utility:  88.4795041522879 action:  up
( 5 , 0 ) utility:  93.23551703306048 action:  up
( 5 , 1 ) utility:  90.8258670547424 action:  up
( 5 , 2 ) utility:  91.70338243665208 action:  left
( 5 , 3 ) utility:  91.79750460206131 action:  up
( 5 , 4 ) utility:  90.47709780168087 action:  up
( 5 , 5 ) utility:  89.20892742157712 action:  up
up | Wall | left | left | left | up |
------------------------------
up | left | left | left | Wall | up |
------------------------------
up | left | left | up | left | left |
------------------------------
up | left | left | up | up | up |
------------------------------
up | Wall | Wall | Wall | up | up |
------------------------------
up | left | left | left | up | up |
------------------------------
```

```python
grid = [[1, None, 1, -0.04, -0.04, 1],
        [-0.04, -1, -0.04, 1, None, -1],
        [ -0.04, -0.04, -1, -0.04, 1, -0.04],
        [-0.04, -0.04, -0.04, -1, -0.04, 1],
        [-0.04, None, None, None, -1, -0.04],
        [-0.04, -0.04, -0.04, -0.04, -0.04, -0.04]]

# Value Iteration
def valueIteration():
    # Create the Utility Grid
    U =  [[0 for i in range(6)] for j in range(6)]
    # Create the Policy Grid
    action = [[0 for i in range(6)] for j in range(6)]
    # Create the dictionary for all the Iterations
    dic = {}
    maxChange = 0
    while True:
        maxChange = 0
        # Iterate through the Grid
        for r in range(0, 6):
            for c in range(0, 6):
                # Get U[s'] for UP
                if(grid[r][c] == None):
                    continue
                if r > 0 and r < 6:
                    if grid[r-1][c] == None:
                        up = U[r][c]
                    else:
                        up = U[r-1][c]
                else:
                    up = U[r][c]
                # down
                if r >= 0 and r < 5:
                    if grid[r+1][c] == None:
                        down = U[r][c]
                    else:
                        down = U[r+1][c]
                else:
                    down = U[r][c]
                # right
                if c < 5 and c >= 0 :
                    if grid[r][c+1] == None:
                        right = U[r][c]
                    else:
                        right = U[r][c+1]
                else:
                    right = U[r][c]
                # left
                if c > 0 and c < 6:
                    if grid[r][c-1] == None:
                        left = U[r][c]
                    else:
                        left = U[r][c-1]
                else:
                    left = U[r][c]
                # For each possible move calculate the Utility
                upValue = 0.8 * up + 0.1 * right + 0.1 * left
                downValue = 0.8 * down + 0.1 * right + 0.1 * left
                rightValue = 0.8*right + 0.1 * up + 0.1 * down
                leftValue = 0.8*left + 0.1 * up + 0.1 * down
```

Fig 1.1

```python
    left = U[r][c]
# For each possible move calculate the Utility
upValue = 0.8 * up + 0.1 * right + 0.1 * left
downValue = 0.8 * down + 0.1 * right + 0.1 * left
rightValue = 0.8*right + 0.1 * up + 0.1 * down
leftValue = 0.8*left + 0.1 * up + 0.1 * down
# Get the highest possible utility
maxValue = max(max(upValue, downValue), max(rightValue, leftValue))
actionString = "left"
if(maxValue == upValue):
    actionString = "up"
elif(maxValue == downValue):
    actionString = "down"
elif(maxValue == rightValue):
    actionString = "right"
else:
    actionString = "left"
# Save the Policy used to get this Utility Value
action[r][c] = actionString
oldValue = U[r][c]
# Calculate the Expected Utility using the Bellman Equation
val = grid[r][c] + 0.99 * maxValue
if dic.get((r,c)) == None:
    dic[(r,c)] = [val]
else:
    dic.get((r,c)).append(U[r][c])
# Update the Utility Grid
U[r][c] = val
# If the Change in Utility is Greater than the maximum change in utility for this iteration
# Then we update the maximum change in Utility to the current Change in Utility
if(abs(U[r][c]-oldValue)) > maxChange:
    maxChange = abs(U[r][c]-oldValue)
```

Fig 1.2

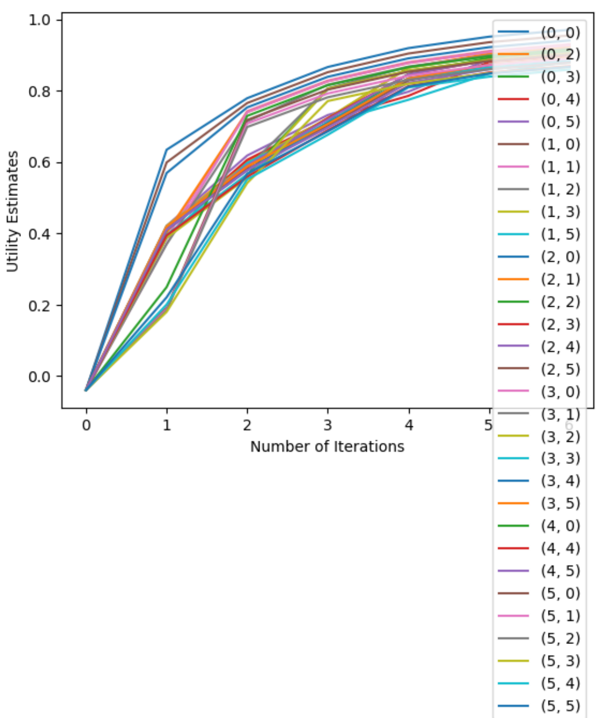
Part 1: Policy Iteration

1.2) Brief Description of implemented solution:

Note: Description omits information on dictionary dic which saves the Utility Values for each states and iteration, since those are just used for plotting purposes and not in the value iteration algorith. However, we do use the actions array for choosing the best policy in Policy Iteration.

We create the grid like before and then generate Utility grid, a grid of random initial actions. Following which we start our iterations. We have our Policy Evaluation for which we calculate the utility of the action in the optimal policy grid using bellman equation. We repeat the Policy Evaluation process for a k number of times (we use k = 50). This is to find the next utility estimate. This can be seen in **Fig 1.3**. Subsequently, after Policy Evaluation we run through the grid and calculate the expected utility of the State again and choose the maximum. If the maximum expected utility is greater than that of the utility optimal policy we update the optimal policy of that state and run policy evaluation again. This can be seen in **Fig 1.4**.

Lastly, we display the Utility of all the states and the Plot of the Utility Estimates and the Plot of utility estimates (right) as a function of the number of iterations (left) as shown below.




```
Coordinates in col, row
( 0 , 0 ) utility:  97.0329961549026 action:  up
( 0 , 1 ) utility:  95.46019758472289 action:  up
( 0 , 2 ) utility:  94.04869530152605 action:  up
( 0 , 3 ) utility:  92.68661138333694 action:  up
( 0 , 4 ) utility:  91.47396397097367 action:  up
( 0 , 5 ) utility:  90.13081275737447 action:  up
( 1 , 1 ) utility:  92.98321250463371 action:  left
( 1 , 2 ) utility:  92.71884212206348 action:  left
( 1 , 3 ) utility:  91.61389899098324 action:  left
( 1 , 5 ) utility:  88.95018268572633 action:  left
( 2 , 0 ) utility:  92.18171991016462 action:  left
( 2 , 1 ) utility:  91.68128049680881 action:  left
( 2 , 2 ) utility:  90.45880351791699 action:  left
( 2 , 3 ) utility:  90.42266333719112 action:  left
( 2 , 5 ) utility:  87.78434297893527 action:  left
( 3 , 0 ) utility:  91.04064479760886 action:  left
( 3 , 1 ) utility:  91.56931100272847 action:  left
( 3 , 2 ) utility:  90.37889323788343 action:  up
( 3 , 3 ) utility:  88.34806413713451 action:  up
( 3 , 5 ) utility:  86.63310852557039 action:  left
( 4 , 0 ) utility:  89.84860131822528 action:  left
( 4 , 2 ) utility:  90.33679784220132 action:  left
( 4 , 3 ) utility:  89.07951488114875 action:  up
( 4 , 4 ) utility:  86.84406042380847 action:  up
( 4 , 5 ) utility:  85.89246517321664 action:  up
( 5 , 0 ) utility:  90.55038851277692 action:  up
( 5 , 1 ) utility:  88.16758526519465 action:  up
( 5 , 2 ) utility:  89.0616474728267 action:  left
( 5 , 3 ) utility:  89.18200941226148 action:  up
( 5 , 4 ) utility:  87.88794052214917 action:  up
( 5 , 5 ) utility:  86.64589451921944 action:  up
up | Wall | left | left | left | up |
------------------------------
up | left | left | left | Wall | up |
------------------------------
up | left | left | up | left | left |
------------------------------
up | left | left | up | up | up |
------------------------------
up | Wall | Wall | Wall | up | up |
------------------------------
up | left | left | left | up | up |
------------------------------
```

```python
import random
#Grid
grid = [[1, None, 1, -0.04, -0.04, 1],
        [-0.04, -1, -0.04, 1, None, -1],
        [ -0.04, -0.04, -1, -0.04, 1, -0.04],
        [-0.04, -0.04, -0.04, -1, -0.04, 1],
        [-0.04, None, None, None, -1, -0.04],
        [-0.04, -0.04, -0.04, -0.04, -0.04, -0.04]]


def policyIteration():
    # Generate utility grid
    U =  [[0 for i in range(6)] for j in range(6)]
    # List of directions
    li = ["up", "down", "right", "left"]
    # Randomly Generate an action for each policy
    pi =  [[random.choice(li) for i in range(6)] for j in range(6)]
    # Create the Policy Grid
    action = [[0 for i in range(6)] for j in range(6)]
    # Create the dictionary to store all the Utility iterations
    dic = {}
    maxChange = 0
    # Initiate the unchanged Variable as false
    unchanged = False
    while unchanged == False:
        unchanged = True
        # Iterate through the possible coordinates
        # Policy Evaluation
        for x in range(50):
            for r in range(0, 6):
                for c in range(0, 6):
                    # Get U[s'] for UP
                    if(grid[r][c] == None):
                        continue

                    if r > 0 and r < 6:
                        if grid[r-1][c] == None:
                            up = U[r][c]
                        else:
                            up = U[r-1][c]
                    else:
                        up = U[r][c]
                    # down
                    if r >= 0 and r < 5:
                        if grid[r+1][c] == None:
                            down = U[r][c]
                        else:
                            down = U[r+1][c]
                    else:
                        down = U[r][c]
                    # right
                    if c < 5 and c >= 0 :
                        if grid[r][c+1] == None:
                            right = U[r][c]
                        else:
                            right = U[r][c+1]
                    else:
                        right = U[r][c]
                    # left
                    if c > 0 and c < 6:
                        if grid[r][c-1] == None:
                            left = U[r][c]
                        else:
                            left = U[r][c-1]
                    else:
                        left = U[r][c]

                    # Calculate Utility for each possible move
                    upValue = 0.8 * up + 0.1 * right + 0.1 * left
                    downValue = 0.8 * down + 0.1 * right + 0.1 * left
                    rightValue = 0.8*right + 0.1 * up + 0.1 * down
                    leftValue = 0.8*left + 0.1 * up + 0.1 * down
                    # Decide the Optimal Utility based on the Optimal Policy
                    val = 0
                    if(pi[r][c] == "up"):
                        val = upValue
                    elif(pi[r][c] == "down"):
                        val = downValue
                    elif(pi[r][c] == "left"):
                        val = leftValue
                    else:
                        val = rightValue
                    # Utility calculated based on Bellman Equation
                    val = grid[r][c] + (0.99 * val)
                    # Update Utility based on the Optimal Policy
                    U[r][c] = val
```

Fig 1.3

```python
for r in range(0, 6):
    for c in range(0, 6):
        # Get U[s'] for UP

        if(grid[r][c] == None):
            continue
        if r > 0 and r < 6:
            if grid[r-1][c] == None:
                up = U[r][c]
            else:
                up = U[r-1][c]
        else:
            up = U[r][c]
        # down
        if r >= 0 and r < 5:
            if grid[r+1][c] == None:
                down = U[r][c]
            else:
                down = U[r+1][c]
        else:
            down = U[r][c]
        # right
        if c < 5 and c >= 0 :
            if grid[r][c+1] == None:
                right = U[r][c]
            else:
                right = U[r][c+1]
        else:
            right = U[r][c]
        # left
        if c > 0 and c < 6:
            if grid[r][c-1] == None:
                left = U[r][c]
            else:
                left = U[r][c-1]
        else:
            left = U[r][c]
        upValue = 0.8 * up + 0.1 * right + 0.1 * left
        downValue = 0.8 * down + 0.1 * right + 0.1 * left
        rightValue = 0.8*right + 0.1 * up + 0.1 * down
        leftValue = 0.8*left + 0.1 * up + 0.1 * down
        realString = ""
        # Get the best expected utility
        maxValue = max(max(upValue, downValue), max(rightValue, leftValue))
        # Get the action of the best utility
        if(maxValue == upValue):
            realString = "up"
        elif(maxValue == downValue):
            realString = "down"
        elif(maxValue == rightValue):
            realString = "right"
        else:
            realString = "left"
        action[r][c] = realString
        # Get the Utility value of following the optimal policy
        oldAction = 0
        if(pi[r][c] == "up"):
            oldAction = upValue
        elif(pi[r][c] == "down"):
            oldAction = downValue
        elif(pi[r][c] == "left"):
            oldAction = leftValue
        else:
            oldAction = rightValue
        if dic.get((r,c)) == None:
            dic[(r,c)] = [val]
        else:
            dic.get((r,c)).append(U[r][c])
        if  maxValue > oldAction:
            pi[r][c] = realString
            unchanged = False
```

Fig 1.4

Part 2: Bonus Question

Answer: The effect of the more complicated maze environment on convergence seems to be rather negligible on convergence. This seems to be the case, since the number of iterations and the graphy of Utility Value against Iterations is very similar to the one in the previous environment. The graphs for Policy Iteration and Utility Iteration are shown below respectively. We can conclude that the environment should be able to handle environments of any complexity.