NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CE/CZ4046 INTELLGENT AGENTS

Assignment 1

Lim Song Wei, Greg U2120517G

Set Up

Maze Implementation

The maze can be represented with 2 components, the “abstract” maze, containing walls and layout, and the rewards. The maze is generated with a default empty cell reward of -0.04.

```
def makeGridReward(width, hight, plusOne, minusOne, wall):
    grid = np.zeros((hight, width))
    reward = np.full((hight, width),-0.04)
    U = grid.copy()

    for i in plusOne:
        grid[i] = 1
        reward[i] = 1
    for i in minusOne:
        grid[i] = -1
        reward[i] = -1
    for i in wall:
        grid[i[0]][i[1]] = np.nan

    return (grid,reward)
```

Code 1.1: Function to generate Grid and reward arrays

```
WIDTH = 6
HIGHT = 6
START = (3,2)
WALLS = ((0,1), (1,4), (4,1), (4,2), (4,3),)
PLUSONE = ((0,0), (0,2), (0,5), (1,3), (2,4), (3,5),)
MINUSONE = ((1,1), (1,5), (2,2), (3,3), (4,4),)
(grid,reward) = makeGridReward(WIDTH, HIGHT, PLUSONE, MINUSONE, WALLS)

prettyPrint(grid)
```

Code 1.2: Setting up grid given in assignment 1

```
+----+----+----+----+----+----+
| +1 | [] | +1 |    |    | +1 |
+----+----+----+----+----+----+
|    | -1 |    | +1 | [] | -1 |
+----+----+----+----+----+----+
|    |    | -1 |    | +1 |    |
+----+----+----+----+----+----+
|    |    |    | -1 |    | +1 |
+----+----+----+----+----+----+
|    | [] | [] | [] | -1 |    |
+----+----+----+----+----+----+
|    |    |    |    |    |    |
+----+----+----+----+----+----+
```

Figure 1.1: Grid generated

Value and Policy Iteration Implementation

To implement value Iteration, there are 2 2d matrix to hold utility (U) and next step utility (Uprime). The following loop will run until the average absolute change in U matrix is less than 10^-5.

1. For all cells in the grid:
    a. All actions are tested, and the max utility and action is returned
2. The utility is stored in Uprime until all cells are tested.
3. U = Uprime

```
def getRelVal(x,y,U,dir,walls):
    """
    Return the expected utility for a certain direction relative to a cell
    if the revertive direction is a wall, return its own utility
    """
    hight = len(U)
    width = len(U[0])
    def isValid(x,y):
        if(x<0 or x>=hight): return False
        if(y<0 or y>=width): return False
        if (x,y) in walls: return False
        return True

    if dir == "^":
        # x,y = x,y-1
        return U[x-1][y] if isValid(x-1,y) else U[x][y]
    elif dir == ">":
        # x,y = x+1,y
        return U[x][y+1] if isValid(x,y+1) else U[x][y]
    elif dir == "v":
        # x,y = x,y+1
        return U[x+1][y] if isValid(x+1,y) else U[x][y]
    elif dir == "<":
        # x,y = x-1,y
        return U[x][y-1] if isValid(x,y-1) else U[x][y]
```

Code 2.1: getRelVal() helper function

The helper function in code 2.1 is to gets the expected utility of a cell given its direction of its real move. This value returned the reward of the cell in that direction or its own cell reward if the move is not valid.

```python
def PI(reward, pos, U, w):
    """
    Return a tuple (BestAction, ActionExpectedUtility)
    Best action 0: Up, 1: Right, 2: Down, 3: Left
    ActionExpectedUtility is the utility expected for taking best action
    """

    # PI main
    expected = [-999,-999,-999,-999]
    x,y = pos

    expected[0] = reward[x][y] + gamma*(getRelVal(x,y,U,"<",w)*0.1 + getRelVal(x,y,U,"^",w)*0.8 +
getRelVal(x,y,U,">",w)*0.1)
    expected[1] = reward[x][y] + gamma*(getRelVal(x,y,U,"^",w)*0.1 + getRelVal(x,y,U,">",w)*0.8 +
getRelVal(x,y,U,"v",w)*0.1)
    expected[2] = reward[x][y] + gamma*(getRelVal(x,y,U,">",w)*0.1 + getRelVal(x,y,U,"v",w)*0.8 +
getRelVal(x,y,U,"<",w)*0.1)
    expected[3] = reward[x][y] + gamma*(getRelVal(x,y,U,"v",w)*0.1 + getRelVal(x,y,U,"<",w)*0.8 +
getRelVal(x,y,U,"^",w)*0.1)

    return (np.argmax(expected),max(expected))
```
Code 2.2: Value and Policy Iteration

Code 2.2, PI() performs a single step on a single cell in value iteration and policy iteration. It takes in the current utility and reward function (a 2d array) and gets the expected utility of moving all 4 directions. It returns both the highest expected utility and its corresponding policy. PI() performs both value iteration and policy iteration. "`np.argmax(expected)`" is the best policy for a cell and "`max(expected)`" is the expected utility for that action.

```python
U = np.zeros((HIGHT, WIDTH))
action = np.zeros((HIGHT,WIDTH))
deltaChange = 100
deltaHistory = []
while deltaChange >pow(10,-5): # converges when average absolute difference is < 10^-5
    Uprime = np.zeros((HIGHT, WIDTH))
    # for all cells in the grid
    for x in range(HIGHT):
        for y in range(WIDTH):
            if (x,y) in WALLS: continue
            action[x][y], Uprime[x][y] = PI(reward, (x,y), U, WALLS)


    deltaChange = np.average(np.absolute(np.array(Uprime) - np.array(U)))
    deltaHistory.append(float(deltaChange))
    U = Uprime
```
Code 2.3: Value and Policy Iteration over every cells

In code 2.3, Value and Policy iteration is done over every cell in. The average change is determined by the average of absolute difference between U and Uprime, deltaChange. U converges when deltaChange < 10^-5.

```
100.00          95.04   93.87   92.65   93.33

 98.39   95.88  94.54   94.40           90.92

 96.95   95.59  93.29   93.18   93.10   91.79

 95.55   94.45  93.23   91.11   91.81   91.89

 94.31                          89.55   90.57

 92.94   91.73  90.53   89.36   88.57   89.30
```

```
+---+---+---+---+---+---+
| ^ |   | < | < | < | ^ |
+---+---+---+---+---+---+
| ^ | < | < | < |   | ^ |
+---+---+---+---+---+---+
| ^ | < | < | ^ | < | < |
+---+---+---+---+---+---+
| ^ | < | < | ^ | ^ | ^ |
+---+---+---+---+---+---+
| ^ |   |   |   | ^ | ^ |
+---+---+---+---+---+---+
| ^ | < | < | < | ^ | ^ |
+---+---+---+---+---+---+
```

Figure 2.4 a: Expected utility of each cell in the grid.

Figure 2.4 b: Policy of each cell in the grid.

Figure 2.4 is trained with:

Gamma	:	0.99
Dimensions	:	6 x 6
Convergence Threshold	:	10^-5

The 6 x 6 maze took 1132 iterations to converge.

Figure 2.4 a show that cell 0, 0 has an expected utility of 100.00. We can validate this answer with the policy diagram in Figure2.4 b showing always going up at cell 0, 0.

$$E(0,0) = 1 + 0.99 + 0.99^2 + 0.99^3 + \cdots$$

$$E(0,\ 0) = \frac{1}{1-\gamma} = \frac{1}{1-0.99} = 100$$ (Infinite Geometric Series)

This aligns with the results we observed.

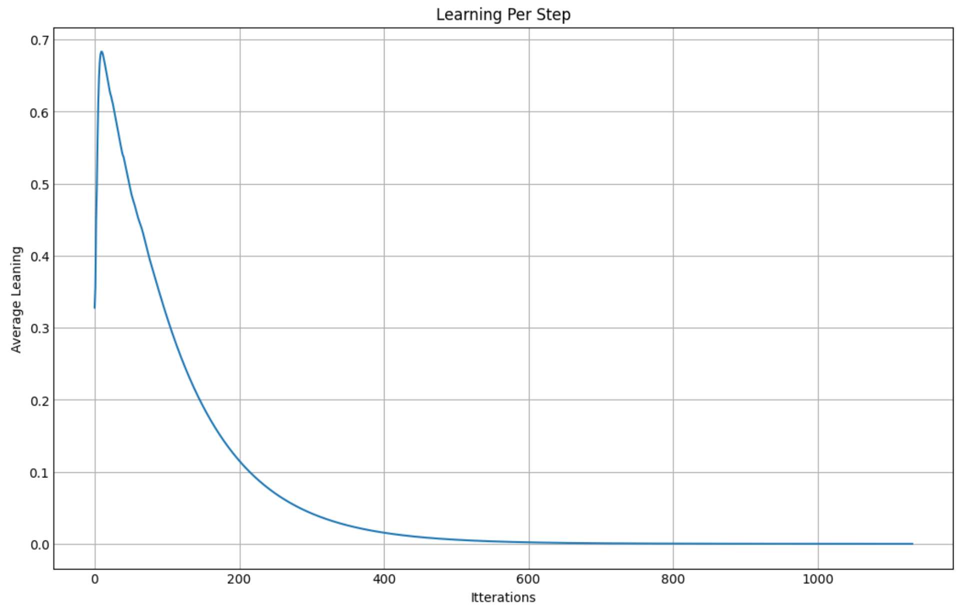

Figure 2.5: Average change in U over many iterations.

From Figure 2.5, it is shown that initially, the change in expected utility is large. However, as the reinforcement learning model trains, its change expected utility per iteration converges to 0.

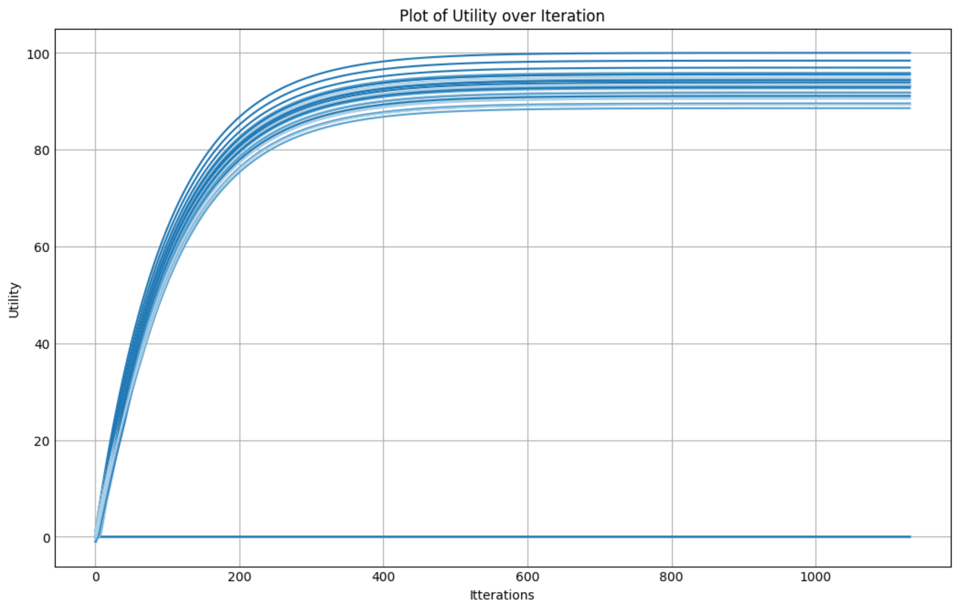

Figure 2.6: Change in Expected Utility of each cell with iterations

Figure 2.6 shows that every cell in the grid eventually converges.

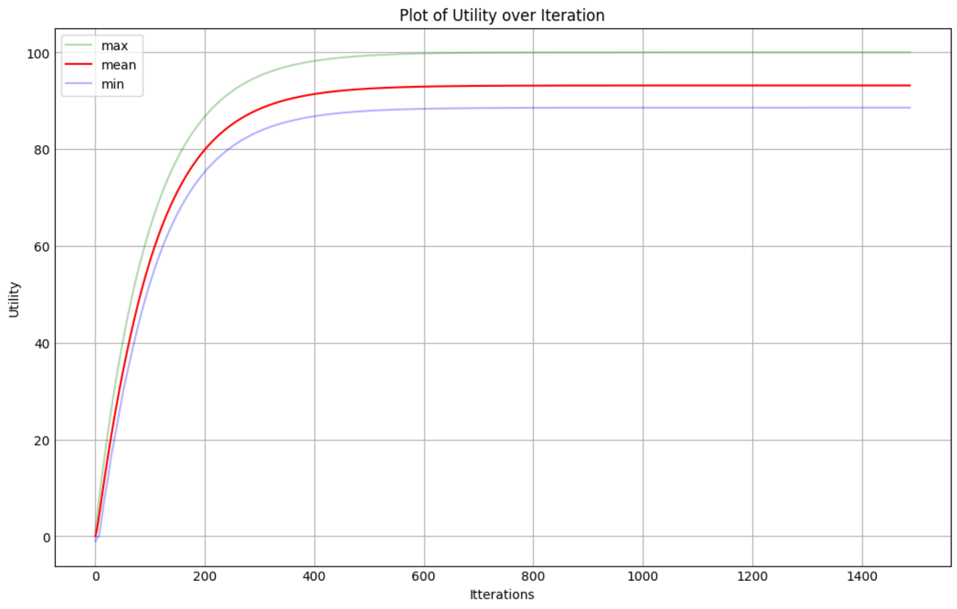

Figure 2.7: Same as 2.6 but only shows max, mean and min

Part 2 (Bonus Questions)

Making a More Complex maze

```
TOTALWALLS = 20
TOTALPLUSONE = 20
TOTALMINUSONE = 20

random.seed("A for SC4003")

def gencoordinates(w, h):
    w=w-1
    h=h-1
    seen = set()
    x, y = random.randint(0, h), random.randint(0, w)
    while True:
        seen.add((x, y))
        yield (x, y)
        x, y = random.randint(0, h), random.randint(0, w)
        while (x, y) in seen:
            x, y = random.randint(0, h), random.randint(0, w)

WIDTH = 16
HIGHT = 12

WALLS = []
PLUSONE = []
MINUSONE = []
# START = (3,2)

if (TOTALWALLS+TOTALMINUSONE+TOTALPLUSONE)*2> (WIDTH*HIGHT):
    print("Not enought blank tiles")
else:
    g = gencoordinates(WIDTH, HIGHT)

    for i in range(TOTALWALLS):
        WALLS.append(next(g))
    for i in range(TOTALPLUSONE):
        PLUSONE.append(next(g))
    for i in range(TOTALMINUSONE):
        MINUSONE.append(next(g))

    (grid,reward) = makeGridReward(WIDTH, HIGHT, PLUSONE, MINUSONE, WALLS)

    prettyPrint(grid)
```

Code 3.1: Generating a 16x12 grid

In this part, we generate a random maze (with preselected seed) of dimensions 16 x 12 with 20 “+1”, 20 “-1” and 20 walls.

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| +1 |    |    |    |    |    |    | +1 |    |    |    |    | +1 |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| +1 | +1 |    |    | +1 | -1 |    | -1 |    | [] |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    | +1 | -1 | -1 |    |    |    |    |    |    |    | -1 |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| [] | +1 |    |    | -1 |    | [] |    |    |    | +1 | +1 | [] |    | -1 |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    | -1 |    |    |    | [] | [] |    |    |    |    |    |    |    |    | +1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    | [] |    | +1 | -1 | +1 | -1 |    | [] |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    | -1 | [] | [] |    |    |    |    | -1 | +1 | +1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| [] | +1 |    |    |    | -1 |    | -1 |    |    |    |    |    | -1 |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    | [] |    |    | [] |    | -1 |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    | [] |    |    |    |    | -1 | -1 |    | [] |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    | -1 |    |    | [] | [] | +1 |    |    |    | +1 |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    | +1 |    | +1 | [] |    | [] |    | [] |    |    |    |    |    |    | -1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

Figure 3.1: 16x12 grid generated

```
99.70   98.51   97.31   96.10   95.01   93.76   92.51   92.33   90.99   89.80   88.52   87.39   87.76   86.89   87.82   88.75
99.67   99.39   97.90   96.58   96.23   93.61   92.27   90.16   89.84           87.58   87.58   86.86   87.81   88.98   90.04
98.33   98.00   96.82   96.67   94.30   92.04   91.03   89.74   88.69   87.76   88.74   88.83   87.58   88.69   90.16   91.36
        97.85   96.46   95.28   92.97   91.79           88.48   87.74   88.68   90.02   90.16           91.03   91.26   92.72
93.83   95.20   95.09   94.01   92.68                   87.31   87.17   88.25   89.12   90.06   91.22   92.20   93.52   94.12
92.74   93.82   93.74   92.76   91.55   90.24   89.06   87.84           89.44   90.13   90.14   92.45   92.52   94.85
91.64   92.49   92.42   91.52   90.36   89.27   89.09                   90.53   90.02   90.96   92.34   93.66   96.39   96.72
        92.44   91.25   90.32   89.28   88.97   91.46   92.96   93.15   91.79   91.03   90.28   91.42   92.57   94.91   95.40
89.65   90.96   90.06   89.14   88.19   87.85           95.56   94.57           92.23   90.03   90.96   92.23   93.57   94.10
88.60   89.57   88.86   87.98   87.10   86.75           96.94   95.70   94.56   93.72   91.40   89.91   90.91           92.87
87.42   87.27   87.65   87.54                   100.00  98.37   96.94   95.42   95.13   93.53   92.00   90.77   90.21   91.50
86.68   87.73   87.54   88.69           -4.00           97.09           94.14   93.80   92.69   91.54   90.39   89.33   89.14
```

Figure 3.2a: Expected Utility for 16x12 grid

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| < | < | < | < | < | < | < | < | < | < | < | < | ^ | v | v | v |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ^ | < | < | < | < | < | < | ^ | ^ |   | v | v | > | > | v | v |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ^ | ^ | < | < | < | < | ^ | < | ^ | v | v | v | < | v | > | v |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   | ^ | < | ^ | < | < |   | ^ | ^ | > | > | v |   | v | v | v |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| > | ^ | ^ | ^ | < |   |   | ^ | > | v | > | > | v | > | v | v |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ^ | ^ | ^ | ^ | < | < | < | < |   | v | > | > | > | > | v |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ^ | ^ | ^ | ^ | ^ | < | v |   |   | v | v | > | > | > | > | ^ |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   | ^ | ^ | ^ | ^ | > | > | v | v | < | v | > | > | > | ^ | ^ |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| > | ^ | ^ | ^ | ^ | ^ |   | v | v |   | v | < | > | > | ^ | ^ |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ^ | ^ | ^ | ^ | ^ | ^ |   | v | < | < | v | < | v | ^ |   | ^ |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ^ | ^ | ^ | v |   |   | < | < | < | < | < | < | < | < | > | ^ |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| > | v | > | > |   | ^ |   | ^ |   | ^ | ^ | < | < | < | < | ^ |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 3.2b: Policy for 16x12 grid

Figure 3.2a and 3.2b is trained with:

Gamma	:	0.99
Dimensions	:	16 x 12
Convergence Threshold	:	10^-5

From figure 3.1a and 3.1b, we can validate the results by observing the 2 highlighted cells.

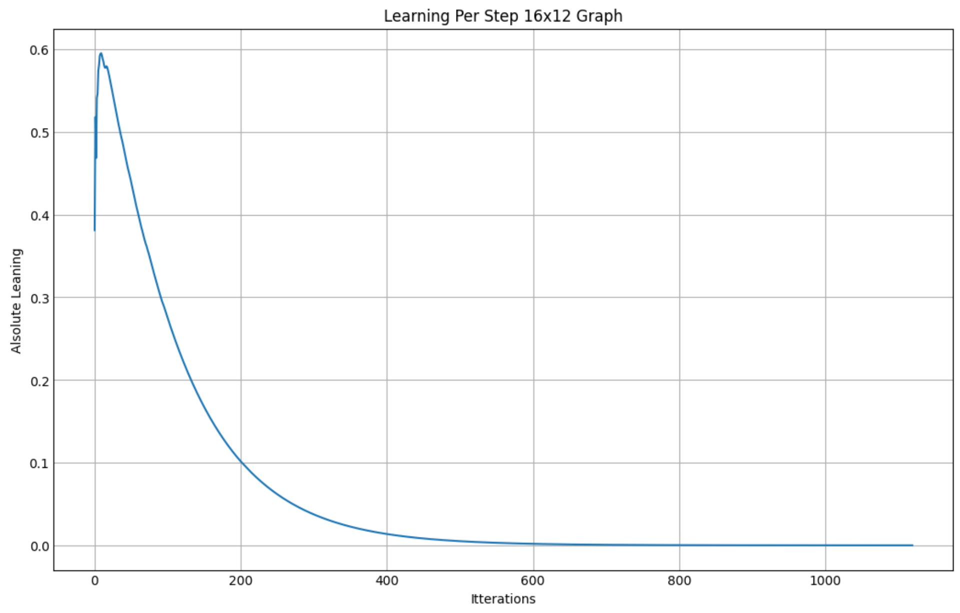

Figure 3.3: Average change in U over many iterations.

Though the larger graph has about 5 times more states, it still converges in 1134 iteration, very similar to the 6 x 6 maze convergence of 1132. This suggests that grid size does not affect the number of iterations to converge.

States, Complexity, and Convergence

In this chapter, we would analyze how gird size and complexity affects convergence.

Set-up for auto generative mazes

Mazes are made with 25% walls, 25% “-1”, 25% “+1” and 25% empty spaces (with a reward of –0.04).

Implementation

```
def makeGrid(width):
    def gencoordinates(w, h):
        w=w-1
        h=h-1
        seen = set()
        x, y = random.randint(0, h), random.randint(0, w)
        while True:
            seen.add((x, y))
            yield (x, y)
            x, y = random.randint(0, h), random.randint(0, w)
            while (x, y) in seen:
                x, y = random.randint(0, h), random.randint(0, w)

    WALLS = []
    PLUSONE = []
    MINUSONE = []

    numCells = width*width
    g = gencoordinates(width, width)

    for i in range(numCells//4):
        WALLS.append(next(g))
    for i in range(numCells//4):
        PLUSONE.append(next(g))
    for i in range(numCells//4):
        MINUSONE.append(next(g))

    (grid,reward) = makeGridReward(width, width, PLUSONE, MINUSONE, WALLS)
    return (grid, reward, WALLS)
```

Code 4.1: Generates a maze with a given width (maze is square)

```
def solveGrid(grid, reward, walls):
    width = len(grid)
    U = np.zeros((width, width))
    action = np.zeros((width,width))
    deltaChange = 100
    deltaHistory = []
    while deltaChange >pow(10,-3): # converges when absolute sum of difference is > 10^-3
        Uprime = np.zeros((width, width))
        for x in range(width):
            for y in range(width):
                if (x,y) in walls: continue
                action[x][y], Uprime[x][y] = PI(reward, (x,y), U, WALLS)

        deltaChange = np.mean(np.absolute(np.array(Uprime) - np.array(U)))
        deltaHistory.append(float(deltaChange))
        U = Uprime

    return deltaHistory
```

Code 4.2: Solves a given maze and return history of its leaning rate (converges when less than 10^-3 for faster running)

```
TESTGRIDWIDTH = [6,8,10,12,14,16,20,24,28,32]
allDeltaChange = []
for i in TESTGRIDWIDTH:
    grid , reward, walls = makeGrid(i)
    deltaChange = solveGrid(grid , reward, walls)

    allDeltaChange.append(deltaChange)
```

Code 4.3: Train for different widths

Findings

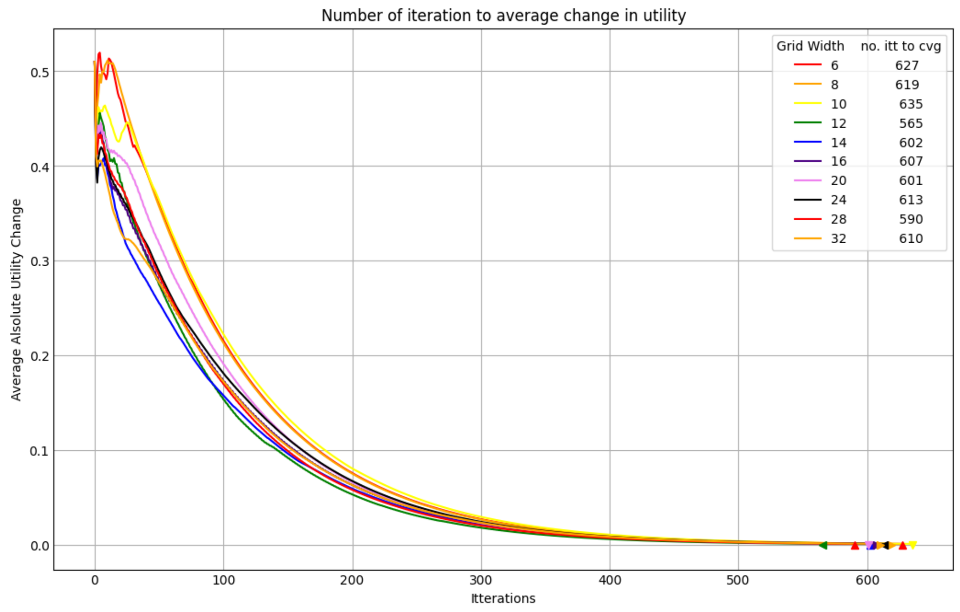

Figure 4.4: Average change in learning rate with iterations

Figure 4.4 is trained with:

Gamma	:	0.99
Dimensions	:	Square [6,8,10,12,14,16,20,24,28,32]
Convergence Threshold	:	10^-3

From Figure 4.4, all grids, 6x6 to 32x32, eventually converged. Additionally, they all converge after a similar number of iterations. The data from this experiment aligns with the RL model eventually converging.

Bibliography

Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence: A modern approach*. Prentice-Hall.