NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

School of Computer Science and Engineering

CZ4046 Machine Learning Assignment 1 Report

Yap Ming Chuen

Description of Implemented Solution.. 3
Answer to Part 1.. 4
Value Iteration Results...4
Policy Iteration Results.. 5
Answer to Part 2 (Bonus Question)... 7
Value Iteration Results...7
Policy Iteration Results.. 13
Answer to Question:...19

Description of Implemented Solution

This assignment aims to implement Value Iteration and Policy Iteration algorithms on a Markov's Decision Problem (with 0.99 as the discount factor) modeled by a 6 by 6 Grid World with walls, and each state has possible reward values of 1, -1, or -0.04, with no terminal states. Subsequently an optimal policy indicating the optimal action to be taken at each state, and a graph plotting the utility values of each state against the number of iterations are produced as results.

The implementation was done in Python. The Grid World was represented using a 2D list storing each state's reward value. Walls are represented by having a reward value of 0, which is used in the algorithm and various other places in the code to check for walls.

Below are brief descriptions of each file with the various functions and purpose:

utils/mdp_utils.py: utility functions to be used in MDP algorithms:

- **getAdjStates**: gets the 4 adjacent states of Left, Right, Up, Down, wall and out-of-bounds checking is done here. To be used in the below functions.
- **getBestAction**: given a state and current utilities, return the best action to be taken based on the highest utility outcome.
- **getNewUtility**: implements the Bellman's Equation, to calculate the new utility after taking the best action given by getBestAction().

utils/plot.py:

- **plot**: utility function to plot the results of state utilities against number of iterations, and store the plot as a png file in the /results directory.

utils/random_maze_generator.py:

- **random_maze**: utility function to produce a random maze of size X by X, where X is an input integer. Possible state reward values are the same as in the original question, and representation of maze is also a 2D list. To be used for the Bonus Question in Part 2.

algo/value_iteration.py:

- **value_iteration**: function which implements Value Iteration with the help of utility function in /utils directory. State utilities are initialized to 0. In each iteration, the utility of each state is updated based on the best action given current utilities, the reward of the current state, and the utilities of possible next states from taking the action. This is the Bellman's Equation. Additionally, the average absolute change in utility is calculated between the current iteration and the previous iteration. This is the convergence variable. If the average change in utility is less than a certain defined threshold (0.001 in implementation), the algorithm is deemed to have converged and the loop stops. Subsequently, the optimal actions at each state after the loop has terminated is returned as the optimal policy, and a dictionary containing the state utilities at each iteration is also returned.

algo/policy_iteration.py:

- **policy_iteration**: function which implements Policy Iteration with the help of utility function in /utils directory. State utilities are initialized to 0. Current policy is initialized to moving “Up” for all states. In each iteration, first the policy is performed once on each state and the resulting utilities updated. Next, the best action to take at each state is calculated based on the new utilities, and the policy is updated with these actions. The convergence variable is a counter which increases each iteration where there is no change in policy, and reset to 0 when there is a change in policy. In the implementation, this variable is set to 20. That is to say, once there are 20 consecutive iterations where the policy remains unchanged, the algorithm is deemed to have converged, and the loop ends. This policy is returned as the optimal policy, as long as a dictionary containing the state utilities at each iteration, similar to the return variables of Value Iteration above.

main.py: Main program to be run. Performs the tasks of Part 1 and Part 2, to run the Value and Policy Iteration algorithms and store the results as .txt and .png files in the /results directory.

Answer to Part 1

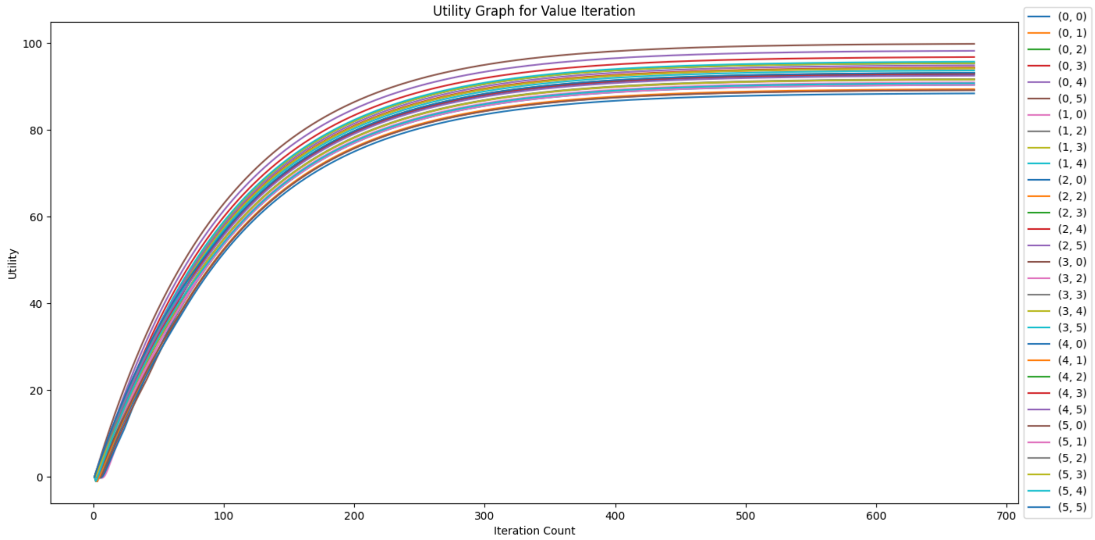
Value Iteration Results

value_iteration_optimal_policy.txt:

```
Optimal Policy obtained using Value Iteration:
(0, 0) : Up
(0, 1) : Up
(0, 2) : Up
(0, 3) : Up
(0, 4) : Up
(0, 5) : Up
(1, 0) : Left
(1, 1) : Wall
(1, 2) : Left
(1, 3) : Left
(1, 4) : Left
(1, 5) : Wall
```

(2, 0)	Left
(2, 1)	Wall
(2, 2)	Left
(2, 3)	Left
(2, 4)	Left
(2, 5)	Left
(3, 0)	Left
(3, 1)	Wall
(3, 2)	Up
(3, 3)	Up
(3, 4)	Left
(3, 5)	Left
(4, 0)	Up
(4, 1)	Up
(4, 2)	Up
(4, 3)	Left
(4, 4)	Wall
(4, 5)	Left
(5, 0)	Up
(5, 1)	Up
(5, 2)	Up
(5, 3)	Left
(5, 4)	Up
(5, 5)	Up

Value Iteration_utility_graph.png:

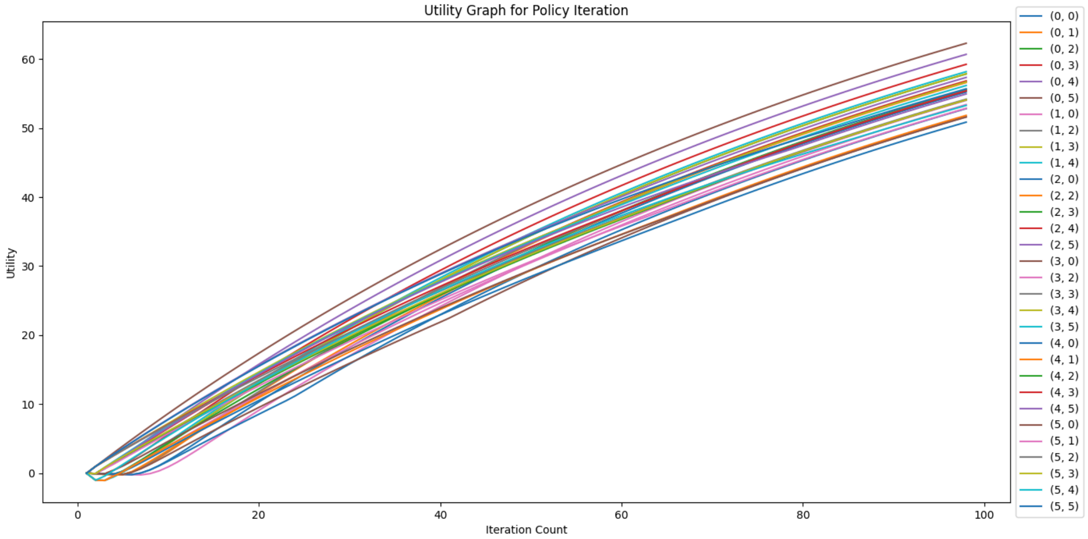


Policy Iteration Results

policy_iteration_optimal_policy.txt:

```
Optimal Policy obtained using Policy Iteration:
(0, 0) : Up
(0, 1) : Up
(0, 2) : Up
(0, 3) : Up
(0, 4) : Up
(0, 5) : Up
(1, 0) : Left
(1, 1) : Wall
(1, 2) : Left
(1, 3) : Left
(1, 4) : Left
(1, 5) : Wall
(2, 0) : Left
(2, 1) : Wall
(2, 2) : Left
(2, 3) : Left
(2, 4) : Left
(2, 5) : Left
(3, 0) : Left
(3, 1) : Wall
(3, 2) : Up
(3, 3) : Up
(3, 4) : Left
(3, 5) : Left
(4, 0) : Up
(4, 1) : Up
(4, 2) : Up
(4, 3) : Left
(4, 4) : Wall
(4, 5) : Left
(5, 0) : Up
(5, 1) : Up
(5, 2) : Up
(5, 3) : Left
(5, 4) : Up
(5, 5) : Up
```

Policy Iteration_utility_graph.png:



Answer to Part 2 (Bonus Question)

In this part, a 15 by 15 maze is generated with the same properties as the original maze, and the algorithms are the same (same reward, maze representation, discount factor etc.)

Value Iteration Results

value_iteration(bonus)_optimal_policy.txt:

```
Optimal Policy obtained using Value Iteration:
(0, 0) : Left
(0, 1) : Wall
(0, 2) : Wall
(0, 3) : Down
(0, 4) : Down
(0, 5) : Wall
(0, 6) : Right
(0, 7) : Right
(0, 8) : Right
(0, 9) : Wall
(0, 10) : Wall
(0, 11) : Down
(0, 12) : Right
(0, 13) : Right
```

```
(0, 14) : Right
(1, 0) : Left
(1, 1) : Down
(1, 2) : Down
(1, 3) : Wall
(1, 4) : Right
(1, 5) : Down
(1, 6) : Right
(1, 7) : Right
(1, 8) : Right
(1, 9) : Down
(1, 10) : Down
(1, 11) : Up
(1, 12) : Right
(1, 13) : Right
(1, 14) : Right
(2, 0) : Left
(2, 1) : Left
(2, 2) : Wall
(2, 3) : Wall
(2, 4) : Wall
(2, 5) : Left
(2, 6) : Right
(2, 7) : Right
(2, 8) : Down
(2, 9) : Down
(2, 10) : Down
(2, 11) : Up
(2, 12) : Up
(2, 13) : Right
(2, 14) : Wall
(3, 0) : Right
(3, 1) : Up
(3, 2) : Up
(3, 3) : Down
(3, 4) : Down
(3, 5) : Left
(3, 6) : Up
(3, 7) : Wall
(3, 8) : Left
(3, 9) : Wall
(3, 10) : Left
(3, 11) : Up
(3, 12) : Up
```

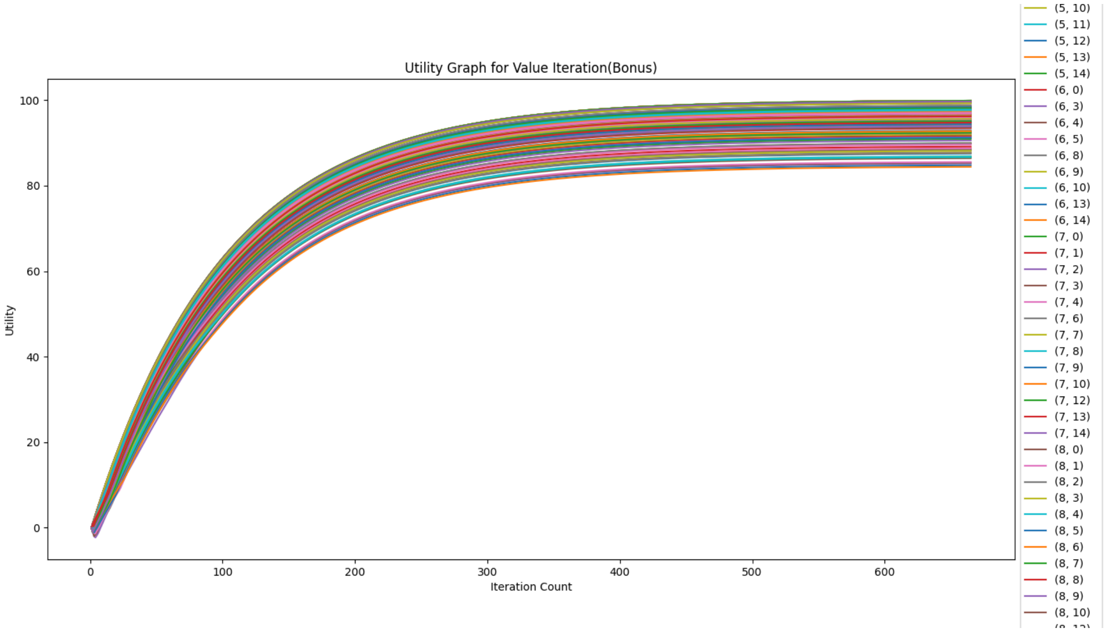
```
(3, 13) : Up
(3, 14) : Up
(4, 0) : Left
(4, 1) : Right
(4, 2) : Wall
(4, 3) : Wall
(4, 4) : Left
(4, 5) : Left
(4, 6) : Wall
(4, 7) : Up
(4, 8) : Left
(4, 9) : Down
(4, 10) : Down
(4, 11) : Down
(4, 12) : Left
(4, 13) : Left
(4, 14) : Wall
(5, 0) : Up
(5, 1) : Up
(5, 2) : Up
(5, 3) : Left
(5, 4) : Wall
(5, 5) : Left
(5, 6) : Up
(5, 7) : Left
(5, 8) : Left
(5, 9) : Left
(5, 10) : Left
(5, 11) : Left
(5, 12) : Left
(5, 13) : Right
(5, 14) : Right
(6, 0) : Left
(6, 1) : Wall
(6, 2) : Wall
(6, 3) : Left
(6, 4) : Down
(6, 5) : Down
(6, 6) : Wall
(6, 7) : Wall
(6, 8) : Left
(6, 9) : Left
(6, 10) : Left
(6, 11) : Wall
```

```
(6, 12) : Wall
(6, 13) : Right
(6, 14) : Right
(7, 0) : Up
(7, 1) : Up
(7, 2) : Up
(7, 3) : Left
(7, 4) : Left
(7, 5) : Wall
(7, 6) : Right
(7, 7) : Up
(7, 8) : Up
(7, 9) : Left
(7, 10) : Left
(7, 11) : Wall
(7, 12) : Up
(7, 13) : Up
(7, 14) : Right
(8, 0) : Left
(8, 1) : Left
(8, 2) : Left
(8, 3) : Left
(8, 4) : Left
(8, 5) : Left
(8, 6) : Down
(8, 7) : Down
(8, 8) : Up
(8, 9) : Left
(8, 10) : Down
(8, 11) : Wall
(8, 12) : Left
(8, 13) : Wall
(8, 14) : Wall
(9, 0) : Left
(9, 1) : Wall
(9, 2) : Left
(9, 3) : Wall
(9, 4) : Left
(9, 5) : Left
(9, 6) : Down
(9, 7) : Up
(9, 8) : Up
(9, 9) : Left
(9, 10) : Wall
```

```
(9, 11) : Wall
(9, 12) : Up
(9, 13) : Up
(9, 14) : Up
(10, 0) : Wall
(10, 1) : Up
(10, 2) : Left
(10, 3) : Wall
(10, 4) : Up
(10, 5) : Wall
(10, 6) : Up
(10, 7) : Wall
(10, 8) : Left
(10, 9) : Left
(10, 10) : Up
(10, 11) : Up
(10, 12) : Up
(10, 13) : Up
(10, 14) : Left
(11, 0) : Up
(11, 1) : Left
(11, 2) : Left
(11, 3) : Up
(11, 4) : Left
(11, 5) : Up
(11, 6) : Left
(11, 7) : Down
(11, 8) : Wall
(11, 9) : Wall
(11, 10) : Wall
(11, 11) : Wall
(11, 12) : Up
(11, 13) : Left
(11, 14) : Left
(12, 0) : Left
(12, 1) : Wall
(12, 2) : Wall
(12, 3) : Up
(12, 4) : Left
(12, 5) : Up
(12, 6) : Left
(12, 7) : Wall
(12, 8) : Wall
(12, 9) : Up
```

```
(12, 10) : Up
(12, 11) : Right
(12, 12) : Wall
(12, 13) : Up
(12, 14) : Left
(13, 0) : Left
(13, 1) : Left
(13, 2) : Up
(13, 3) : Up
(13, 4) : Left
(13, 5) : Down
(13, 6) : Left
(13, 7) : Down
(13, 8) : Down
(13, 9) : Down
(13, 10) : Wall
(13, 11) : Up
(13, 12) : Up
(13, 13) : Up
(13, 14) : Left
(14, 0) : Left
(14, 1) : Left
(14, 2) : Up
(14, 3) : Left
(14, 4) : Left
(14, 5) : Wall
(14, 6) : Wall
(14, 7) : Left
(14, 8) : Down
(14, 9) : Left
(14, 10) : Wall
(14, 11) : Wall
(14, 12) : Up
(14, 13) : Left
(14, 14) : Left
```

Value Iteration(Bonus)_utility_graph.png:



Policy Iteration Results

policy_iteration(bonus)_optimal_policy.txt:

```
Optimal Policy obtained using Policy Iteration:
(0, 0) : Left
(0, 1) : Wall
(0, 2) : Wall
(0, 3) : Down
(0, 4) : Down
(0, 5) : Wall
(0, 6) : Right
(0, 7) : Right
(0, 8) : Right
(0, 9) : Wall
(0, 10) : Wall
(0, 11) : Down
(0, 12) : Right
(0, 13) : Right
(0, 14) : Right
(1, 0) : Left
(1, 1) : Down
(1, 2) : Down
(1, 3) : Wall
```

```
(1, 4) : Right
(1, 5) : Down
(1, 6) : Right
(1, 7) : Right
(1, 8) : Right
(1, 9) : Down
(1, 10) : Down
(1, 11) : Up
(1, 12) : Right
(1, 13) : Right
(1, 14) : Right
(2, 0) : Left
(2, 1) : Left
(2, 2) : Wall
(2, 3) : Wall
(2, 4) : Wall
(2, 5) : Left
(2, 6) : Right
(2, 7) : Right
(2, 8) : Down
(2, 9) : Down
(2, 10) : Down
(2, 11) : Up
(2, 12) : Up
(2, 13) : Right
(2, 14) : Wall
(3, 0) : Right
(3, 1) : Up
(3, 2) : Up
(3, 3) : Down
(3, 4) : Down
(3, 5) : Left
(3, 6) : Up
(3, 7) : Wall
(3, 8) : Left
(3, 9) : Wall
(3, 10) : Left
(3, 11) : Up
(3, 12) : Up
(3, 13) : Up
(3, 14) : Up
(4, 0) : Left
(4, 1) : Right
(4, 2) : Wall
```

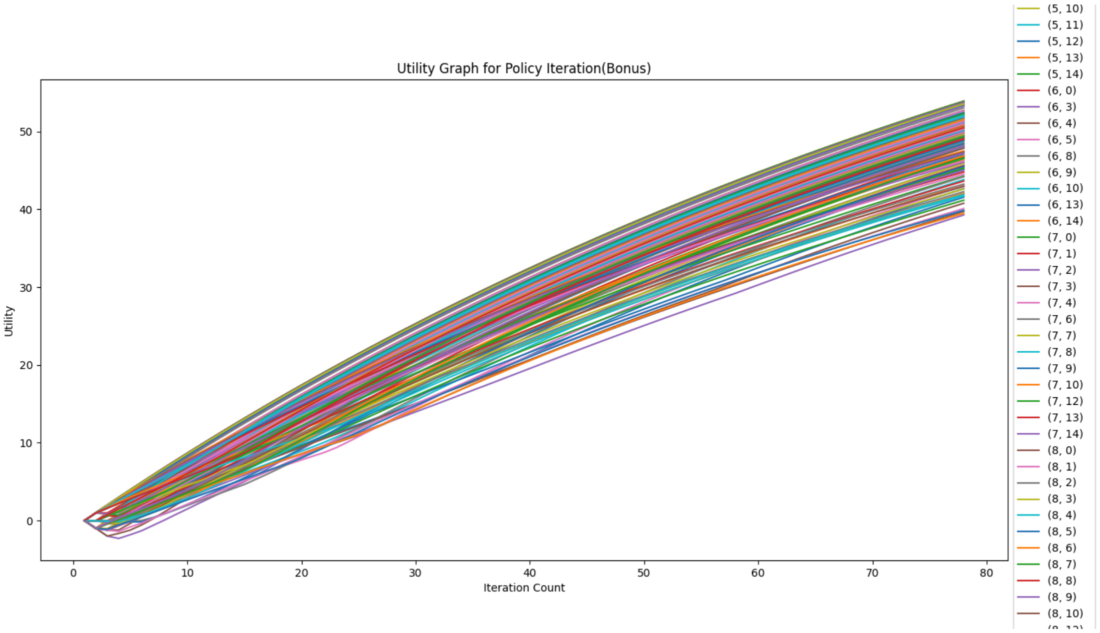
```
(4, 3) : Wall
(4, 4) : Left
(4, 5) : Left
(4, 6) : Wall
(4, 7) : Up
(4, 8) : Left
(4, 9) : Down
(4, 10) : Down
(4, 11) : Down
(4, 12) : Left
(4, 13) : Left
(4, 14) : Wall
(5, 0) : Up
(5, 1) : Up
(5, 2) : Up
(5, 3) : Left
(5, 4) : Wall
(5, 5) : Left
(5, 6) : Up
(5, 7) : Left
(5, 8) : Left
(5, 9) : Left
(5, 10) : Left
(5, 11) : Left
(5, 12) : Up
(5, 13) : Right
(5, 14) : Right
(6, 0) : Left
(6, 1) : Wall
(6, 2) : Wall
(6, 3) : Left
(6, 4) : Down
(6, 5) : Down
(6, 6) : Wall
(6, 7) : Wall
(6, 8) : Left
(6, 9) : Left
(6, 10) : Left
(6, 11) : Wall
(6, 12) : Wall
(6, 13) : Right
(6, 14) : Right
(7, 0) : Up
(7, 1) : Up
```

```
(7, 2) : Up
(7, 3) : Left
(7, 4) : Left
(7, 5) : Wall
(7, 6) : Right
(7, 7) : Up
(7, 8) : Up
(7, 9) : Left
(7, 10) : Left
(7, 11) : Wall
(7, 12) : Up
(7, 13) : Up
(7, 14) : Right
(8, 0) : Left
(8, 1) : Left
(8, 2) : Left
(8, 3) : Left
(8, 4) : Left
(8, 5) : Left
(8, 6) : Down
(8, 7) : Down
(8, 8) : Up
(8, 9) : Left
(8, 10) : Down
(8, 11) : Wall
(8, 12) : Left
(8, 13) : Wall
(8, 14) : Wall
(9, 0) : Left
(9, 1) : Wall
(9, 2) : Left
(9, 3) : Wall
(9, 4) : Left
(9, 5) : Left
(9, 6) : Down
(9, 7) : Up
(9, 8) : Up
(9, 9) : Left
(9, 10) : Wall
(9, 11) : Wall
(9, 12) : Up
(9, 13) : Up
(9, 14) : Up
(10, 0) : Wall
```

```
(10, 1) : Up
(10, 2) : Left
(10, 3) : Wall
(10, 4) : Up
(10, 5) : Wall
(10, 6) : Right
(10, 7) : Wall
(10, 8) : Left
(10, 9) : Left
(10, 10) : Up
(10, 11) : Up
(10, 12) : Up
(10, 13) : Up
(10, 14) : Left
(11, 0) : Up
(11, 1) : Left
(11, 2) : Left
(11, 3) : Up
(11, 4) : Left
(11, 5) : Up
(11, 6) : Left
(11, 7) : Down
(11, 8) : Wall
(11, 9) : Wall
(11, 10) : Wall
(11, 11) : Wall
(11, 12) : Up
(11, 13) : Left
(11, 14) : Left
(12, 0) : Left
(12, 1) : Wall
(12, 2) : Wall
(12, 3) : Up
(12, 4) : Left
(12, 5) : Up
(12, 6) : Left
(12, 7) : Wall
(12, 8) : Wall
(12, 9) : Up
(12, 10) : Up
(12, 11) : Right
(12, 12) : Wall
(12, 13) : Up
(12, 14) : Left
```

```
(13, 0) : Left
(13, 1) : Left
(13, 2) : Up
(13, 3) : Up
(13, 4) : Left
(13, 5) : Left
(13, 6) : Left
(13, 7) : Down
(13, 8) : Down
(13, 9) : Down
(13, 10) : Wall
(13, 11) : Up
(13, 12) : Up
(13, 13) : Up
(13, 14) : Left
(14, 0) : Left
(14, 1) : Left
(14, 2) : Up
(14, 3) : Left
(14, 4) : Left
(14, 5) : Wall
(14, 6) : Wall
(14, 7) : Left
(14, 8) : Down
(14, 9) : Left
(14, 10) : Wall
(14, 11) : Wall
(14, 12) : Up
(14, 13) : Left
(14, 14) : Left
```

Policy Iteration(Bonus)_utility_graph.png:



Answer to Question:

In general, value and policy iteration can find the optimal policy as the number of iterations approaches infinity. However, in practical implementations, the convergence condition is important. If the convergence condition is not strict enough with respect to the complexity of the environment, the algorithm may terminate prematurely and result in a non-optimal policy. In the case of the above implementation of Value Iteration, if the reward numbers of each state are very similar, within the margin of the “average change in utility” condition defined (as 0.001), then the algorithm can terminate prematurely. In the case of Policy Iteration, the convergence condition defined “number of consecutive iterations with no policy change” may also not be enough and cause the algorithm to terminate prematurely if it takes more iterations to change the policy than the defined threshold. In fact, when the above implementation is tested setting this parameter to “5 consecutive iterations”, in Part 1, the algorithm had a different optimal policy output from that of value iteration, and only after changing it to “20 consecutive iterations” did it produce the same optimal policy output. In general as complexity of the environment increases, the convergence condition has to be made stricter and scaled with respect to the environment complexity in order for both Value and Policy Iteration to learn the right policy.