A7 – CS4300 Assignment A7 Lab Report

Value Iteration

By Haochen Zhang & Tim Wei
11/22/2017

1. Introduction

In this assignment, we were to find out the behaviours of a value iteration algorithm on a fully observable Wumpus World. We implemented such a function, *CS4300_MDP_value_iterationI*, and *CS4300_MDP_policy* to study about the policy evaluated under the utilities learnt using *CS4300_MDP_value_iterationI*. In this report, we want to answer the following points of interests:

- What are the policies calculated under different reward values?
- How are the utilities calculated over the iterations with different γ values?

2. Method

We have implement 3 main functions in this lab which is *CS4300_MDP_value_iteration*, *CS4300_MDP_policy* and *transition_probability_table*. The agent is to learn a policy for the following Wumpus world:

0 0 0 G
0 0 P 0
0 0 W 0
0 0 P 0

In *transition_probability_table*, we create a 16 by 4 matrix holding a structure in each of the cell representing initial state and action taken. Each cell is a 1 by 16 matrix hold probabilities representing final state. Thus, the neighboring cells of the current cell will have values vary from 0 to 1 and other cells will only have 0 as their value. For this lab, we put cells with pits, Wumpus and gold into terminal states.

For *CS4300_MDP_value_iteration*, we implement the value iteration algorithm given on p. 653 of the textbook, which is modified to suit the needs of Wampus World. We choose max iteration to be 1000 and eta to be 0.1.

We use the equation, choosing the action that maximizes the expected utility of the subsequent state, on p. 651 of the textbook to implement our *CS4300_MDP_policy* function. We simple use the dot product of the utility and the transition probability for each cells and find the maximum utility to determine the best action for that cell.

We ran *CS4300_MDP_value_iteration* with the reward value of a clear cell ranging from -2000 to 2000 to find how many different policies will be generated. We also ran *CS4300_MDP_value_iteration* with γ values of 0.9, 0.99, 0.999, 0.9999, 0.99999 and 0.999999.

3. Verification of Program

For *transition_probability_table*, we checked some coner cell with action that will lead agent to stand at put. For example, if agent are at cell (1,1) and take action 1, the probabilities will be 0.1 at cell (1,1). 0.1 at cell (2,1) and 0.8 at cell (1,2). The followings are some other data we checked:

Initial cell	Action taken	Probabilities
1,1	Up	0.1 at Cell (1.1), 0.1 at Cell (2,1), 0.8 at Cell(1,2)
1,1	Left	0.9 at Cell (1,1), 0.1 at Cell(1,2)
4,1	Down	0.9 at cell(4,1), 0.1 at Cell(3,1)
1,4	Right	0.1 at Cell (1,4), 0.1 at Cell (1,3), 0.8 at Cell(2,4)

The result from the last page matches our hand developed solutions.

For *CS4300_MDP_value_iteration*, as you can see in the following table with max iteration to be 1000 and eta to be 0.1, the closer the cells is to the gold, the higher the utility it will have. On contrary, the closer the cell is to the pits or Wumpus, the lower the utility it will have.

505.08	585.04	691.21	1000
437.77	396.83	-1000	691.21
378.22	333.17	-1000	446.89
326.88	290.47	-1000	253.58

For *CS4300_MDP_policy*, we have generated a policy base on the original rules of Wumpus world. We set the gold cell have reward 1000; pit cells and Wumpus cell have reward -1000; each move will cost the agent -1 points. The result is as following when we run on the broad given to us:

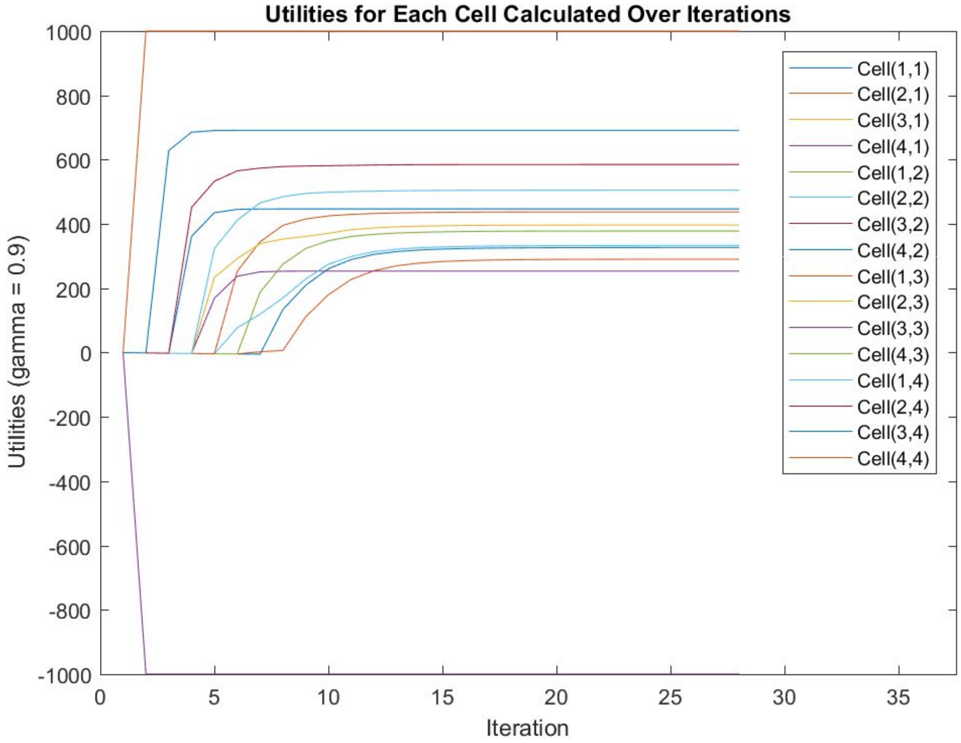
```
    ">"   ">"   "^"   "G"
    "^"   "<"   "x"   ">"
    "^"   "<"   "x"   ">"
    "^"   "<"   "x"   ">"
```

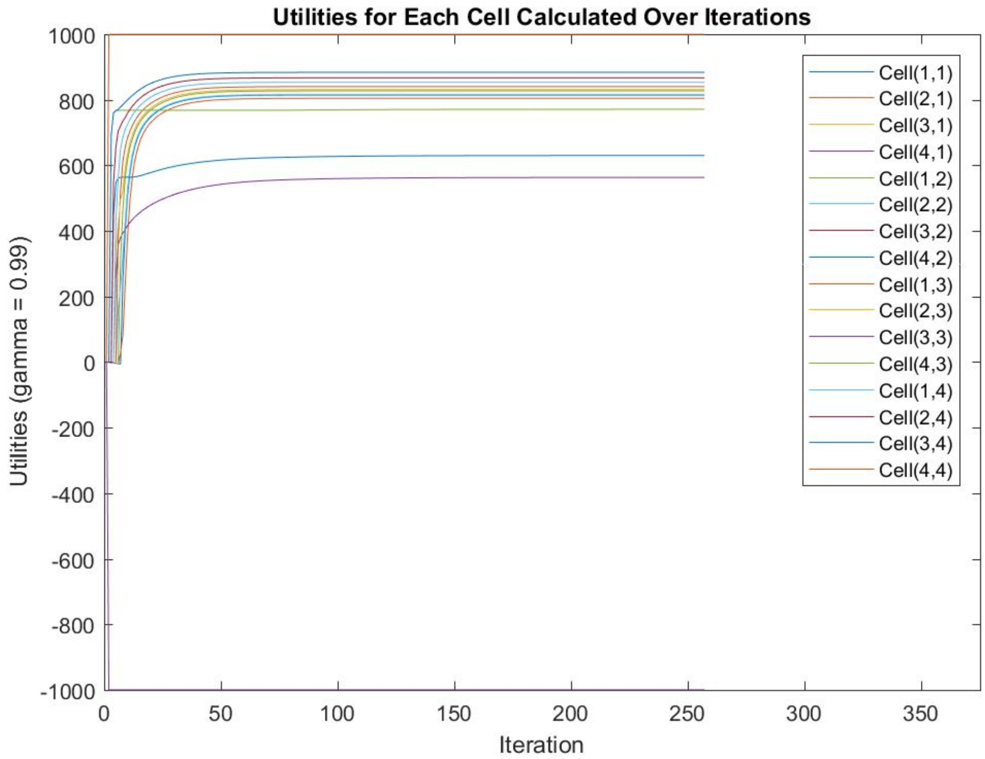
Arrows indicate actions and the letter “G” indicate gold position. Since we don’t have shot action in this lab, we use letter “x” to indicate both Wumpus and pits. It also matches our hand developed solution.

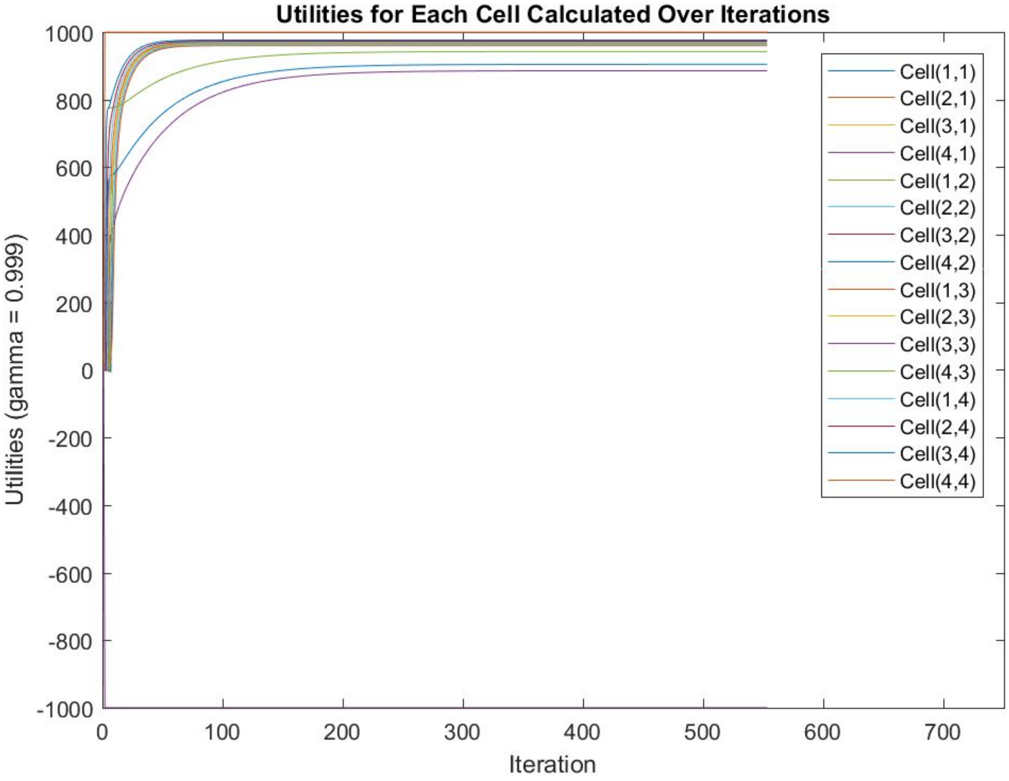
4. Data and Analysis

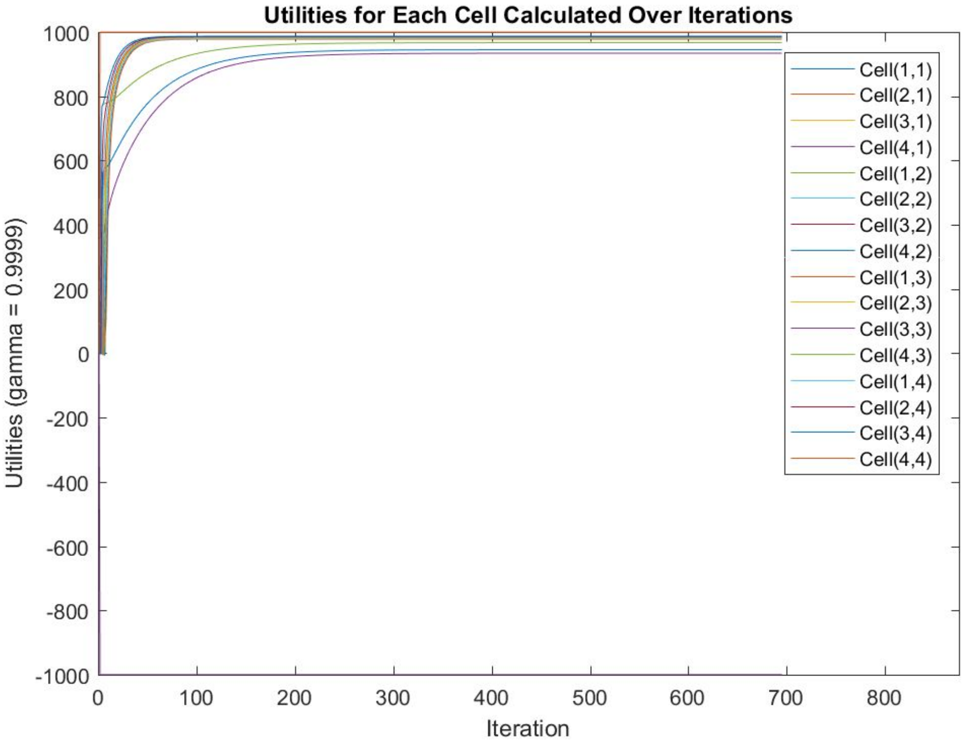
With the reward value of a clear cell from -2000 to 2000, *CS4300_MDP_policy* generated 92 different policy sets. Due to the amount of the policy sets we will not show them in the report. The human readable form of the policy sets is *policies_s* generated by *CS4300_policies_with_different_rewards*.

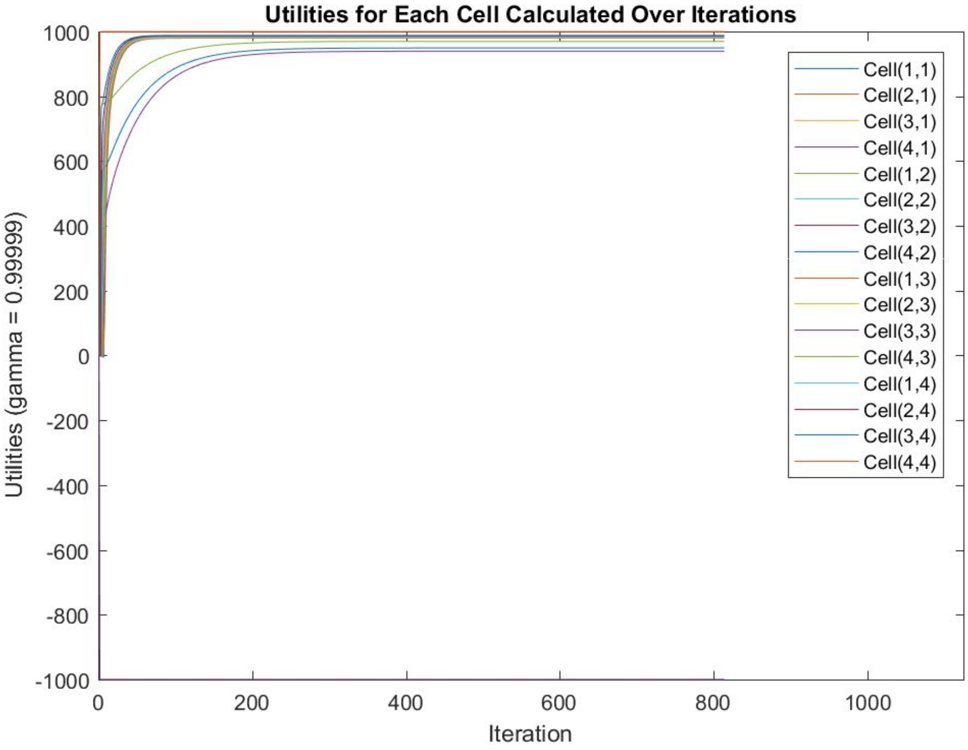
The following plots shows our results for γ values of 0.9, 0.99, 0.999, 0.9999, 0.99999 and 0.999999.

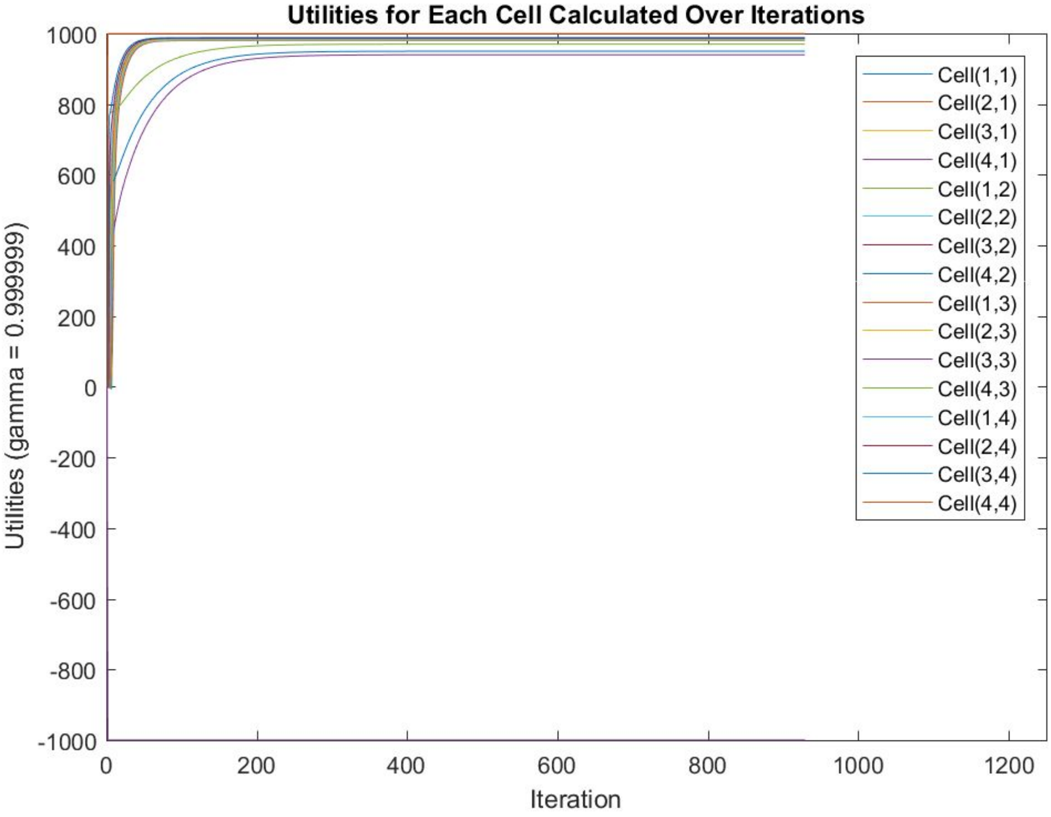



Utilities for Each Cell Calculated Over Iterations
Utilities (gamma = 0.99)
1000
800
600
400
200
0
-200
-400
-600
-800
-1000
0
50
100
150
200
250
300
350
Iteration
Cell(1,1)
Cell(2,1)
Cell(3,1)
Cell(4,1)
Cell(1,2)
Cell(2,2)
Cell(3,2)
Cell(4,2)
Cell(1,3)
Cell(2,3)
Cell(3,3)
Cell(4,3)
Cell(1,4)
Cell(2,4)
Cell(3,4)
Cell(4,4)


Utilities for Each Cell Calculated Over Iterations
Utilities (gamma = 0.999)
1000
800
600
400
200
0
-200
-400
-600
-800
-1000
0
100
200
300
400
500
600
700
Iteration
Cell(1,1)
Cell(2,1)
Cell(3,1)
Cell(4,1)
Cell(1,2)
Cell(2,2)
Cell(3,2)
Cell(4,2)
Cell(1,3)
Cell(2,3)
Cell(3,3)
Cell(4,3)
Cell(1,4)
Cell(2,4)
Cell(3,4)
Cell(4,4)


Utilities for Each Cell Calculated Over Iterations
Utilities (gamma = 0.9999)
1000
800
600
400
200
0
-200
-400
-600
-800
-1000
0
100
200
300
400
500
600
700
800
Iteration
Cell(1,1)
Cell(2,1)
Cell(3,1)
Cell(4,1)
Cell(1,2)
Cell(2,2)
Cell(3,2)
Cell(4,2)
Cell(1,3)
Cell(2,3)
Cell(3,3)
Cell(4,3)
Cell(1,4)
Cell(2,4)
Cell(3,4)
Cell(4,4)


Utilities for Each Cell Calculated Over Iterations
Utilities (gamma = 0.99999)
1000
800
600
400
200
0
-200
-400
-600
-800
-1000
0
200
400
600
800
1000
Iteration
Cell(1,1)
Cell(2,1)
Cell(3,1)
Cell(4,1)
Cell(1,2)
Cell(2,2)
Cell(3,2)
Cell(4,2)
Cell(1,3)
Cell(2,3)
Cell(3,3)
Cell(4,3)
Cell(1,4)
Cell(2,4)
Cell(3,4)
Cell(4,4)



5. Interpretation

In this section, we will answer the question we posted for this study:

- What are the policies calculated under different reward values?

There turns out to be 92 different sets of policies, we will list some sets with interesting qualities.

a. R(s) < -1623

The world is so painful that the policy is to get to a terminal state as soon as possible, even if it means death.

```
">"   ">"   ">"   "G"
">"   ">"   "x"   "^"
">"   ">"   "x"   "<"
">"   ">"   "x"   "<"
```

b. -7 < R(s) < 0

This set of policy is probably ideal for an agent to work with. The world is bearable so it does not take any risk, and will try to get to the gold.

```
">"   ">"   "^"   "G"
"^"   "<"   "x"   ">"
"^"   "<"   "x"   ">"
"^"   "<"   "x"   ">"
```

c. 2 < R(s) < 16

There are some variation of this set of policy, where the policies when x = 1 and (2,4) can be any of the 4 directions, we marked them as + here. As the reward becomes positive, the policy start to aim to stay in the world as long as possible, even risk a chance of dying in (3,4).

```
"+"  "+"  "<"  "G"
"+"  "<"  "x"  ">"
"+"  "<"  "x"  ">"
"+"  "<"  "x"  ">"
```

d. 17 < R(s)

The world is so enjoyable that the agent will try to avoid gold even more, risking more at (4,3) so it have more chance to stay on the board even longer.

```
"+"  "+"  "<"  "G"
"+"  "<"  "x"  "v"
"+"  "<"  "x"  ">"
"+"  "<"  "x"  ">"
```

- How are the utilities calculated over the iterations with different γ values?

From the data we can see that when the γ value increases, it requires more iterations for *CS4300_MDP_value_iteration* to reach the equilibrium of utility values, and the utility values also becomes larger. We can also see that terminal states' utilities are calculated to be its reward value on the second iteration. If we were to design an agent with *CS4300_MDP_value_iteration*, it seems like γ = 0.9 is a better choice out of the six, since the utility values are distributed more evenly so the agent would know better about the priority of each cell.

6. Critique

We put a lot of effort in analyzing how many policies our program will generation by changing the reward value in clear cells in *CS4300_MDP_value_iteration*. The result is reasonable but we don't have a way to recognize that random action could be taken in cells that is not next to terminal cells if the rewards is larger than 1.

7. Log

Haochen Zhang(Section 2, 4, 6)
A total of 4 hours was spent performing the experiment in Matlab.
A total of 3 hours was spent performing the experiment in writing the report.

Tim Wei (Section 1, 3, 5)
A total of 4 hours was spent performing the experiment in Matlab.
A total of 3 hours was spent performing the experiment in writing the report.