

Value Iteration Report

Question 1 – Write a value iteration agent in ValueIterationAgent.java which has been partially specified for you. Here you need to implement the iterate() & extractPolicy() methods. The former should perform value iteration for several steps (k steps – this is one of the fields of the class) and the latter should extract the policy from the computed values. **(6 points)**

Answer 1 –

Summary for Value Iteration Agent Implementation

1. Functions Modified:

- a. `iterate()`
- b. `extractPolicy()`

2. Detailed Implementation Analysis:

a. `iterate()` Method

Function:

- i. Perform value iteration to estimate state values
- ii. Runs for a fixed number of iterations (k)
- iii. Computes the maximum expected value for each state
- iv. Updates the value function based on the Bellman optimality equation

3. `extractPolicy()` Method

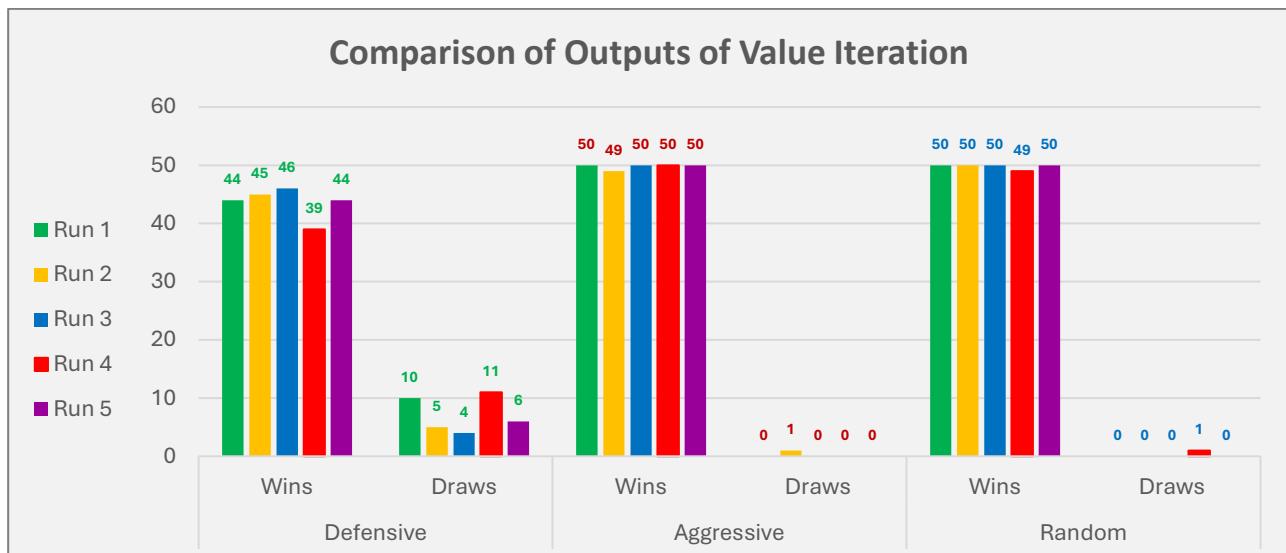
Function:

- i. Extract the optimal policy based on the computed value function
- ii. Uses one-step look-ahead to determine the best move for each state
- iii. Creates a policy map with the best move for each non-terminal state

Question 2 – Test your Value Iteration Agent against each of the provided agents 50 times and report on the results – how many games they won, lost & drew against each of the other rule-based agents. The rule-based agents are random, aggressive, defensive. (1 point)

Answer 2 –

Iteration/ Agent	Defensive			Aggressive			Random		
	Wins	Losses	Draws	Wins	Losses	Draws	Wins	Losses	Draws
Run 1	44	0	10	50	0	0	50	0	0
Run 2	45	0	5	49	0	1	50	0	0
Run 3	46	0	4	50	0	0	50	0	0
Run 4	39	0	11	50	0	0	49	0	1
Run 5	44	0	6	50	0	0	50	0	0



The program was run for a total of 250 iterations split into 5 runs of 50 iterations each. As we can see from the tabular column, we obtain the best result during iteration 3 of the Value-Iteration Agent.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure.
- Test ValueIteration:** The active Java test class.
- Run:** The test has run successfully with 3/3 passes.
- Failure Trace:** No failures are present.
- Output:** Displays the execution trace of the test methods, showing the sequence of moves made by the agents during the game.

```
public void testDefensive() {
    System.out.println("Against Defensive Agent");
    int[] results = valueIterationAgent.playAgainstDefensiveAgent();
    assertEquals(0, results[0]);
    assertEquals(0, results[1]);
    assertEquals(0, results[2]);
}

public void testAggressive() {
    System.out.println("Against Aggressive Agent");
    int[] results = valueIterationAgent.playAgainstAggressiveAgent();
    assertEquals(1, results[0]);
    assertEquals(0, results[1]);
    assertEquals(0, results[2]);
}

public void testRandom() {
    System.out.println("Against Random Agent");
    int[] results = valueIterationAgent.playAgainstRandomAgent();
    assertEquals(0, results[0]);
    assertEquals(0, results[1]);
    assertEquals(0, results[2]);
}

public void testHuman() {
    System.out.println("Against Human");
    int[] results = valueIterationAgent.playAgainstHuman();
    assertEquals(0, results[0]);
    assertEquals(0, results[1]);
    assertEquals(0, results[2]);
}

public void testOther() {
    System.out.println("Against Other");
    int[] results = valueIterationAgent.playAgainstOther();
    assertEquals(0, results[0]);
    assertEquals(0, results[1]);
    assertEquals(0, results[2]);
}

public void testQuida() {
    System.out.println("Against Quida");
    int[] results = valueIterationAgent.playAgainstQuida();
    assertEquals(0, results[0]);
    assertEquals(0, results[1]);
    assertEquals(0, results[2]);
}
```

The output window shows the execution trace for each method, displaying the moves made by the agents:

- testDefensive()**:
 - Playing move: X(1,2)
 - Playing move: O(1,1)
 - Playing move: X(1,1)
 - Playing move: O(2,1)
 - Playing move: X(2,1)
 - Playing move: O(2,2)
 - Playing move: X(2,2)
 - Playing move: O(3,2)
 - Playing move: X(3,2)
 - Playing move: O(3,1)
 - Playing move: X(3,1)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
 - Playing move: O(2,0)
 - Playing move: X(2,0)
 - Playing move: O(2,1)
 - Playing move: X(2,1)
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(2,2)
 - Playing move: X(2,2)
 - Playing move: O(3,2)
 - Playing move: X(3,2)
 - Playing move: O(3,1)
 - Playing move: X(3,1)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
- testAggressive()**:
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(2,2)
 - Playing move: X(2,2)
 - Playing move: O(3,2)
 - Playing move: X(3,2)
 - Playing move: O(3,1)
 - Playing move: X(3,1)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
- testRandom()**:
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(2,2)
 - Playing move: X(2,2)
 - Playing move: O(3,2)
 - Playing move: X(3,2)
 - Playing move: O(3,1)
 - Playing move: X(3,1)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
- testHuman()**:
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(2,2)
 - Playing move: X(2,2)
 - Playing move: O(3,2)
 - Playing move: X(3,2)
 - Playing move: O(3,1)
 - Playing move: X(3,1)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
- testOther()**:
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(2,2)
 - Playing move: X(2,2)
 - Playing move: O(3,2)
 - Playing move: X(3,2)
 - Playing move: O(3,1)
 - Playing move: X(3,1)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
- testQuida()**:
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(1,2)
 - Playing move: X(1,2)
 - Playing move: O(2,2)
 - Playing move: X(2,2)
 - Playing move: O(3,2)
 - Playing move: X(3,2)
 - Playing move: O(3,1)
 - Playing move: X(3,1)
 - Playing move: O(1,0)
 - Playing move: X(1,0)
 - Playing move: O(1,0)
 - Playing move: X(1,0)