

Hosam Farouk

Value Iteration Agent VS Random Agent:

```
import static org.junit.Assert.assertEquals;

public class TestValueIteration {
    @Test
    public void testDefensive() {
        System.out.println("Against Defensive Agent:");
        int[] results=TestPolicyIterationAgent.playAgainstEachOther(new
        System.out.println("Wins: " + results[0] + " Losses: " + results
        assertEquals(0, results[1]);
    }

    @Test
    public void testAggressive() {
        System.out.println("Against Aggressive Agent:");
        int[] results=TestPolicyIterationAgent.playAgainstEachOther(new
```

```
<terminated> TestValueIteration [JUnit] /Users/hosambassem/p2/pool/plugins/org.eclipse.justi.openjdk.hotspot.j
Playing move: X(1,0)

|X| | |
| |0| |
|X|0| |

X won!
Wins: 50 Losses: 0 Draws: 0
Against Defensive Agent:
Playing move: X(2,0)
```

Wins: 50 | Losses: 0 | Draws: 0

Value Iteration Agent VS Defensive Agent:

```
import static org.junit.Assert.assertEquals;

public class TestValueIteration {
    @Test
    public void testDefensive() {
        System.out.println("Against Defensive Agent:");
        int[] results=TestPolicyIterationAgent.playAgainstEachOther(new
        System.out.println("Wins: " + results[0] + " Losses: " + results
        assertEquals(0, results[1]);
    }

    @Test
    public void testAggressive() {
        System.out.println("Against Aggressive Agent:");
        int[] results=TestPolicyIterationAgent.playAgainstEachOther(new
```

```
<terminated> TestValueIteration [JUnit] /Us
Playing move: X(1,2)

|0|X|0|
|X|0|X|
|X|0|X|

It's a draw.
Wins: 40 Losses: 0 Draws: 10
Against Aggressive Agent:
Playing move: X(2,0)
```

Wins: 40 | Losses: 0 | Draws: 10

Value Iteration Agent VS Aggressive Agent:

```
import static org.junit.Assert.assertEquals;

public class TestValueIteration {
    @Test
    public void testDefensive() {
        System.out.println("Against Defensive Agent:");
        int[] results=TestPolicyIterationAgent.playAgainstEachOther(new
        System.out.println("Wins: " + results[0] + " Losses: " + results
        assertEquals(0, results[1]);
    }

    @Test
    public void testAggressive() {
        System.out.println("Against Aggressive Agent:");
        int[] results=TestPolicyIterationAgent.playAgainstEachOther(new
        System.out.println("Wins: " + results[0] + " Losses: " + results
        assertEquals(0, results[1]);
    }

    @Test
    public void testAggressive() {
        System.out.println("Against Aggressive Agent:");
        int[] results=TestPolicyIterationAgent.playAgainstEachOther(new
```

```
<terminated> TestValueIteration [JUnit] /Users
Playing move: X(1,1)

|X| | |
|0|X| |
|X|0| |

Playing move: 0(2,2)

|X| | |
|0|X| |
|X|0| |

Playing move: X(0,2)

|X| |X|
|0|X| |
|X|0|0|

X won!
Wins: 50 Losses: 0 Draws: 0
```

Wins: 50 | Losses: 0 | Draws: 0

Function Implementations in 2-3 sentences:

iterate():

- This function performs k iterations of value iteration over all the states in the MDP.
- It iterates through all the states, skipping the terminal states and calculates the Q value using a helper method I created calculateQValue.
- And then it updates the states V value by taking the maximum Q value.

extractPolicy():

- This function derives a policy based on the computed value. It performs a single step of expectimax for each state to find the optimal move.
- As the iterate method it skips the terminal states and for non terminal states it calculates the Q value for all possible moves.
- And then it selects the move with the highest Q value for each state and stores it in the policy hashmap.

calculateQValue():

- This is a helper method that I implemented to compute the Q value for a given state and move.
- It first generates all possible transitions.
- Then it calculates the Q value using Bellmans Equation with a loop to account for all the transitions.