Markov Decision Processes

Salma Abd Elaziz (28)
Salma Mohamed Mohamed Attia (29)
SHadwa Abd Elmobdy Ali (31)

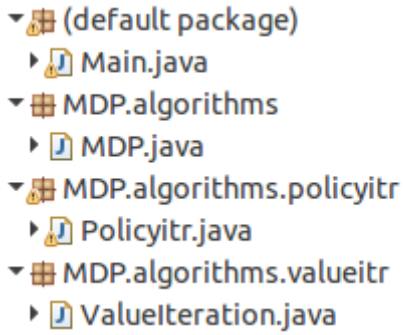
Problem Statement:

The 3x3 world shown in the following figure:

r	-1	10
-1	-1	-1
-1	-1	-1

The agent has four actions Up, Down, Right and Left.
The transition model is: 80% of the time the agent goes in the direction it selects; the rest of
the time it moves at right angles to the intended direction. A collision with a wall results in no
movement.

Project Structure:



- The ValueIteration class : class with the function that performs the value iteration algorithm, it takes a MDP object and the € (allowed error) as parameters.
- The Policyitr class : class with the function that perform the policy iteration algorithm, it it takes a MDP object as parameter.
- The Main class : the engine of the project, which calls the functions to run the algorithms.

Requirements:

1) Value Iteration:

Code :

```java
package MDP.algorithms.valueitr;

import java.awt.Point;
import java.util.Arrays;

import MDP.algorithms.MDP;

public class ValueIteration {

    static final Point termialState = new Point(0, 2);
    static final int TRUE_DIRECTION = 0;
    static final int RIGHT_DIRECTION = 1;
    static final int LEFT_DIRECTION = 2;
    static final int UP = 0;
    static final int DOWN = 1;
    static final int RIGHT = 2;
    static final int LEFT = 3;

    public static String[][] policy;

    public ValueIteration() {
        policy = new String[3][3];
    }

    public double[][] valueIteration(MDP mdp, double eps) {
        double delta = 0.0;
        double[][] u = new double[3][3];
        double[][] uDash = new double[3][3];
        uDash[termialState.x][termialState.y] = 10.0;

        do {
            copyArray(u, uDash);
            delta = 0.0;
            for (int i = 2; i >= 0; i--) {
                for (int j = 0; j < 3; j++) {
                    if (termialState.equals(new Point(i, j)))
                        break;
                    uDash[i][j] = getMaxUtility(i, j, u, mdp);
                    if (Math.abs(uDash[i][j] - u[i][j]) > delta)
                        delta = Math.abs(uDash[i][j] - u[i][j]);
                }
            }
        } while (delta >= (eps * (1 - mdp.getDiscountFactor())) / mdp.getDiscountFactor());

        return u;
    }

    private void copyArray(double[][] x, double[][] y) {
        for (int i = 0; i < y.length; i++) {
            x[i] = Arrays.copyOf(y[i], y[i].length);
        }
    }
```

```
    private double getMaxUtility(int row, int col, double[][] uDash, MDP mdp) {
        double maximum = Double.NEGATIVE_INFINITY;
        double[] transition = mdp.getTransition();

        // up
        Point[] neighbors = mdp.getDirection(new Point(row, col), UP);
        double val = transition[TRUE_DIRECTION]
                * (mdp.getReward()[neighbors[TRUE_DIRECTION].x][neighbors[TRUE_DIRECTION].y]
                        + mdp.getDiscountFactor() * uDash[neighbors[TRUE_DIRECTION].x][neighbors[TRUE_DIRECTION].y])
                + transition[RIGHT_DIRECTION]
                        * (mdp.getReward()[neighbors[RIGHT_DIRECTION].x][neighbors[RIGHT_DIRECTION].y]
                                + mdp.getDiscountFactor()
                                        * uDash[neighbors[RIGHT_DIRECTION].x][neighbors[RIGHT_DIRECTION].y])
                + transition[LEFT_DIRECTION]
                        * (mdp.getReward()[neighbors[LEFT_DIRECTION].x][neighbors[LEFT_DIRECTION].y]
                                + mdp.getDiscountFactor()
                                        * uDash[neighbors[LEFT_DIRECTION].x][neighbors[LEFT_DIRECTION].y]);

        if (maximum < val) {
            maximum = val;
            policy[row][col] = "up";
        }
        // left
        neighbors = mdp.getDirection(new Point(row, col), LEFT);
        val = transition[TRUE_DIRECTION]
                * (mdp.getReward()[neighbors[TRUE_DIRECTION].x][neighbors[TRUE_DIRECTION].y]
                        + mdp.getDiscountFactor() * uDash[neighbors[TRUE_DIRECTION].x][neighbors[TRUE_DIRECTION].y])
                + transition[RIGHT_DIRECTION]
                        * (mdp.getReward()[neighbors[RIGHT_DIRECTION].x][neighbors[RIGHT_DIRECTION].y]
                                + mdp.getDiscountFactor()
                                        * uDash[neighbors[RIGHT_DIRECTION].x][neighbors[RIGHT_DIRECTION].y])
                + transition[LEFT_DIRECTION]
                        * (mdp.getReward()[neighbors[LEFT_DIRECTION].x][neighbors[LEFT_DIRECTION].y]
                                + mdp.getDiscountFactor()
                                        * uDash[neighbors[LEFT_DIRECTION].x][neighbors[LEFT_DIRECTION].y]);
        if (maximum <= val) {
            maximum = val;
            policy[row][col] = "left";
        }
        // right
        neighbors = mdp.getDirection(new Point(row, col), RIGHT);
        val = transition[TRUE_DIRECTION]
                * (mdp.getReward()[neighbors[TRUE_DIRECTION].x][neighbors[TRUE_DIRECTION].y]
                        + mdp.getDiscountFactor() * uDash[neighbors[TRUE_DIRECTION].x][neighbors[TRUE_DIRECTION].y])
                + transition[RIGHT_DIRECTION]
                        * (mdp.getReward()[neighbors[RIGHT_DIRECTION].x][neighbors[RIGHT_DIRECTION].y]
                                + mdp.getDiscountFactor()
                                        * uDash[neighbors[RIGHT_DIRECTION].x][neighbors[RIGHT_DIRECTION].y])
                + transition[LEFT_DIRECTION]
                        * (mdp.getReward()[neighbors[LEFT_DIRECTION].x][neighbors[LEFT_DIRECTION].y]
                                + mdp.getDiscountFactor()
                                        * uDash[neighbors[LEFT_DIRECTION].x][neighbors[LEFT_DIRECTION].y]);

        if (maximum <= val) {
            maximum = val;
            policy[row][col] = "right";
        }
        // down
        neighbors = mdp.getDirection(new Point(row, col), DOWN);
        val = transition[TRUE_DIRECTION]
                * (mdp.getReward()[neighbors[TRUE_DIRECTION].x][neighbors[TRUE_DIRECTION].y]
                        + mdp.getDiscountFactor() * uDash[neighbors[TRUE_DIRECTION].x][neighbors[TRUE_DIRECTION].y])
                + transition[RIGHT_DIRECTION]
                        * (mdp.getReward()[neighbors[RIGHT_DIRECTION].x][neighbors[RIGHT_DIRECTION].y]
                                + mdp.getDiscountFactor()
                                        * uDash[neighbors[RIGHT_DIRECTION].x][neighbors[RIGHT_DIRECTION].y])
                + transition[LEFT_DIRECTION]
                        * (mdp.getReward()[neighbors[LEFT_DIRECTION].x][neighbors[LEFT_DIRECTION].y]
                                + mdp.getDiscountFactor()
                                        * uDash[neighbors[LEFT_DIRECTION].x][neighbors[LEFT_DIRECTION].y]);
        if (maximum < val) {
            maximum = val;
            policy[row][col] = "down";
        }
        return maximum;
    }

}
```

The Result:

```
-------------------- r = 100 ---------------------------------
The Utilities!
[879.6972044644679, 858.2607755480997, 10.0]
[858.2607755480997, 753.7368831714876, 596.0096355401553]
[743.3993553540205, 661.6295103178556, 581.3331043110234]
The Policy!
[left, left, null]
[up, left, left]
[up, left, left]
-------------------- r = 3 ----------------------------------
The Utilities!
[25.23982312639927, 24.407341564694242, 10.0]
[24.407341564694242, 20.434353281575348, 18.504375808713025]
[19.879682259120017, 16.86990254633998, 15.210156117213486]
The Policy!
[left, left, null]
[up, up, up]
[up, up, up]
-------------------- r = 0 ----------------------------------
The Utilities!
[14.452944494210634, 17.93398187737637, 10.0]
[12.665558496541063, 14.666268381216149, 17.933981877376375]
[10.083154198803165, 11.749411227240765, 14.252426467431407]
The Policy!
[right, right, null]
[up, up, up]
[up, up, up]
-------------------- r = -3 ---------------------------------
The Utilities!
[13.987084378330794, 17.92366693640366, 10.0]
[11.371153782585965, 14.561957153186547, 17.92366693640366]
[9.202158937568488, 11.599305511685085, 14.229447914819493]
The Policy!
[right, right, null]
[right, right, up]
[right, right, up]
```

2) Policy Iteration:

Code:

```
package MDP.algorithms.policyitr;

import java.awt.Point;
import java.lang.reflect.Field;
import java.util.Arrays;
import MDP.algorithms.MDP;

public class Policyitr {
    private MDP mdp;
    private double[][] utilities;
    private String[][] policies;

    public Policyitr(MDP mdp) {
        this.mdp = mdp;
        utilities = new double [mdp.getNoRow()][mdp.getNoCol()];
        utilities[0][2] = 10;
        policies = new String [mdp.getNoRow()][mdp.getNoCol()];
        IntializePolicies();
        System.out.println("Intial Utility Values:");
        System.out.println("======================");
        System.out.println(Arrays.deepToString(utilities));
        System.out.println("Intial Policies:");
        System.out.println("================");
        System.out.println(Arrays.deepToString(policies));
        mainAlgorithm();
        System.out.println("Final Utility Values:");
        System.out.println("======================");
        System.out.println(Arrays.deepToString(utilities));
        System.out.println("Final Policies:");
        System.out.println("======================");
        System.out.println(Arrays.deepToString(policies));
    }

    private void IntializePolicies() {
        for (int row = 0; row < policies.length; row++) {
            for (int col = 0; col < policies[row].length; col++) {
                policies[row][col] = mdp.getActions()[(int)(Math.random() * 4)];
            }
        }
    }

private double[][] policyEvaluation() {
    double[][] utilitesTemp = new double [mdp.getNoRow()][mdp.getNoCol()];
    utilitesTemp [0][2] = 10;
    Point[] neighbours = new Point [3];
    for (int row = 0; row < utilities.length; row++) {
        for (int col = 0; col < utilities[row].length; col++) {
            if(col == 2 && row == 0) break;
            try {
                Field f = MDP.class.getField(policies[row][col]);
                neighbours = mdp.getDirection(new Point(row, col), f.getInt(null));
            } catch (NoSuchFieldException el) {
                el.printStackTrace();
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
            utilitesTemp[row][col] = getStateUtility ("TRUE_DIRECTION", neighbours) +
                                     getStateUtility ("RIGHT_DIRECTION", neighbours) +
                                     getStateUtility ("LEFT_DIRECTION", neighbours);
        }
    }
    return utilitesTemp;
}
```

```
    private double getStateUtility (String action, Point[] neighbours) {
        int direction = 0;
        try {
            Field f = MDP.class.getField(action);
            direction = f.getInt(null);
        } catch (NoSuchFieldException e1) {
            e1.printStackTrace();
        } catch (IllegalArgumentException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
        return (mdp.getTransition()[direction]) *
                    ((mdp.getReward()[neighbours[direction].x][neighbours[direction].y]) +
                            (mdp.getDiscountFactor() * utilities[neighbours[direction].x][neighbours[direction].y]));
    }
    private Object[] getMaxUtility(Point state) {
        double maximum = Double.POSITIVE_INFINITY;
        double val = 0;
        String maxDirection = "UP";
        Object[] returnArr = new Object[2];
        //up
        maximum = getStateUtility("TRUE_DIRECTION", mdp.getDirection(state, mdp.UP)) +
                  getStateUtility("RIGHT_DIRECTION", mdp.getDirection(state, mdp.UP)) +
                  getStateUtility("LEFT_DIRECTION", mdp.getDirection(state, mdp.UP));
        // left
        val = getStateUtility("TRUE_DIRECTION", mdp.getDirection(state, mdp.LEFT)) +
              getStateUtility("RIGHT_DIRECTION", mdp.getDirection(state, mdp.LEFT)) +
              getStateUtility("LEFT_DIRECTION", mdp.getDirection(state, mdp.LEFT));
        if (maximum < val) {
            maximum = val;
            maxDirection = "LEFT";
        }
        // right
        val = getStateUtility("TRUE_DIRECTION", mdp.getDirection(state, mdp.RIGHT)) +
              getStateUtility("RIGHT_DIRECTION", mdp.getDirection(state, mdp.RIGHT)) +
              getStateUtility("LEFT_DIRECTION", mdp.getDirection(state, mdp.RIGHT));
        if (maximum < val) {
            maximum = val;
            maxDirection = "RIGHT";
        }
        // down
        val = getStateUtility("TRUE_DIRECTION", mdp.getDirection(state, mdp.DOWN)) +
              getStateUtility("RIGHT_DIRECTION", mdp.getDirection(state, mdp.DOWN)) +
              getStateUtility("LEFT_DIRECTION", mdp.getDirection(state, mdp.DOWN));
        if (maximum < val) {
            maximum = val;
            maxDirection = "DOWN";
        }
        returnArr[0] = maximum;
        returnArr[1] = maxDirection;
        return returnArr;
    }


    private void mainAlgorithm() {
        boolean unchanged = true;
        do {
            utilities = policyEvaluation();
            unchanged = true;
            for (int row = 0; row < utilities.length; row++) {
                for (int col = 0; col < utilities[row].length; col++) {
                    if(col == 2 && row == 0) break;
                    Object[] maximum = getMaxUtility(new Point(row, col));
                    if((double)maximum[0] > utilities[row][col]) {
                        policies [row][col] = (String)maximum[1];
                        unchanged = false;
                    }
                }
            }
        } while (!unchanged);
    }
}
```

The Result:

```
------------------ r = 100 ------------------
  Intial Utility Values:
  ======================
  [[0.0, 0.0, 10.0],
  [0.0, 0.0, 0.0],
  [0.0, 0.0, 0.0]]
  Intial Policies:
  ================
  [[UP, DOWN, DOWN],
  [UP, LEFT, RIGHT],
  [DOWN, UP, UP]]
  Final Utility Values:
  =====================
  [[879.7072139752679, 858.2707850588997, 10.0],
  [858.2707850588997, 753.7468926822877, 596.0186328532341],
  [743.4093648648206, 661.6395198286557, 581.34300135541]]
  Final Policies:
  =====================
  [[UP, LEFT, DOWN],
  [UP, LEFT, LEFT],
  [UP, LEFT, LEFT]]
  ------------------ r = 3 ------------------
  Intial Utility Values:
  ======================
  [[0.0, 0.0, 10.0],
  [0.0, 0.0, 0.0],
  [0.0, 0.0, 0.0]]
  Intial Policies:
  ================
  [[RIGHT, UP, DOWN],
  [DOWN, DOWN, RIGHT],
  [LEFT, RIGHT, LEFT]]
  Final Utility Values:
  =====================
  [[25.250670919336756, 24.418083051933177, 10.0],
  [24.418083051933177, 20.44413905929698, 18.505464302567834],
  [19.890252035547626, 16.87899505507166, 15.212136101983834]]
  Final Policies:
  =====================
  [[UP, LEFT, DOWN],
  [UP, UP, UP],
  [UP, UP, UP]]
  ------------------ r = 0 ------------------
  Intial Utility Values:
  ======================
  [[0.0, 0.0, 10.0],
  [0.0, 0.0, 0.0],
  [0.0, 0.0, 0.0]]
  Intial Policies:
  ================
  [[UP, RIGHT, LEFT],
  [DOWN, RIGHT, UP],
  [UP, LEFT, LEFT]]
  Final Utility Values:
  =====================
  [[14.453284090684372, 17.934054736697032, 10.0],
  [12.66632346778796, 14.666553448825512, 17.93405473669703],
  [10.084924312080648, 11.750313635400655, 14.252799601766947]]
  Final Policies:
  =====================
  [[RIGHT, RIGHT, LEFT],
  [UP, UP, UP],
  [UP, UP, UP]]
```

```
------------------ r = -3 ------------------
Intial Utility Values:
======================
[[0.0, 0.0, 10.0],
[0.0, 0.0, 0.0],
[0.0, 0.0, 0.0]]
Intial Policies:
================
[[LEFT, LEFT, LEFT],
[DOWN, LEFT, DOWN],
[RIGHT, UP, LEFT]]
Final Utility Values:
======================
[[13.98744267158874, 17.92373711607027, 10.0],
[11.372023417501756, 14.56223084026607, 17.92373711607027],
[9.203859689635554, 11.600041958323882, 14.229774175626092]]
Final Policies:
======================
[[RIGHT, RIGHT, LEFT],
[RIGHT, RIGHT, UP],
[RIGHT, RIGHT, UP]]
```

The MDP class:

```java
package MDP.algorithms;

import java.awt.Point;
import java.util.Arrays;

public class MDP {
    public static final int TRUE_DIRECTION = 0;
    public static final int RIGHT_DIRECTION = 1;
    public static final int LEFT_DIRECTION = 2;
    public static final int UP = 0;
    public static final int DOWN = 1;
    public static final int RIGHT = 2;
    public static final int LEFT = 3;
    private int noCol, noRow;
    private double discountFactor;


    private double[] transition;
    private String[] actions;
    private int[][] reward;
    public MDP(int noRow, int noCol, double discountFactor, int r,
                    double trueDirection, double leftDirection, double rightDirection) {
        this.noRow = noRow;
        this.noCol = noCol;
        this.discountFactor = discountFactor;
        actions = new String [4];
        fillActions(actions);
        reward = new int [noRow][noCol];
        fillReward(reward);
        reward[0][0] = r;
        reward [0][2] = 10;
        transition = new double[3];
        transition[TRUE_DIRECTION] = trueDirection;
        transition[RIGHT_DIRECTION] = rightDirection;
        transition[LEFT_DIRECTION] = leftDirection;
    }

    public String[] getActions() {
        return actions;
    }

    private void fillActions(String[] actions) {
        actions[0] = "UP";
        actions[1] = "DOWN";
        actions[2] = "RIGHT";
        actions[3] = "LEFT";
    }
```

```java
    public int[][] getReward() {
        return reward;
    }

    private void fillReward(int[][] reward) {
        for (int[] row: reward) Arrays.fill(row, -1);
    }

    public double[] getTransition() {
        return transition;
    }

    public double getDiscountFactor() {
        return discountFactor;
    }

    public int getNoRow() {
        return noRow;
    }

    public int getNoCol() {
        return noRow;
    }

    private boolean checkRowBorder(int rowIndex){
        if(0 <= rowIndex && rowIndex < noRow) {
            return true;
        }
        return false;
    }
    private boolean checkColBorder(int colIndex){
        if(0 <= colIndex && colIndex < noCol) {
            return true;
        }
        return false;
    }
```

```java
    public Point[] getDirection(Point stateIndex, int direction) {
        Point[] upStates = new Point[3];
        int indicator = 0;
        if (UP == direction || LEFT == direction){
            indicator = -1;
        }else if(DOWN == direction || RIGHT == direction){
            indicator = 1;
        }
        if (UP == direction || DOWN == direction){
            if(checkRowBorder((stateIndex.x) + indicator))
                upStates[TRUE_DIRECTION] = new Point((stateIndex.x) + indicator, stateIndex.y);
            else
                upStates[TRUE_DIRECTION] = new Point(stateIndex.x, stateIndex.y);
            if(checkColBorder((stateIndex.y) + indicator))
                upStates[LEFT_DIRECTION] = new Point(stateIndex.x, (stateIndex.y) + indicator);
            else
                upStates[LEFT_DIRECTION] = new Point(stateIndex.x, stateIndex.y);
            if(checkColBorder((stateIndex.y) + (indicator * -1)))
                upStates[RIGHT_DIRECTION] = new Point(stateIndex.x, (stateIndex.y) + (indicator * -1));
            else
                upStates[RIGHT_DIRECTION] = new Point(stateIndex.x, stateIndex.y);
        }else if (LEFT == direction || RIGHT == direction){
            if(checkColBorder((stateIndex.y) + indicator))
                upStates[TRUE_DIRECTION] = new Point(stateIndex.x, (stateIndex.y) + indicator);
            else
                upStates[TRUE_DIRECTION] = new Point(stateIndex.x, stateIndex.y);
            if(checkRowBorder((stateIndex.x) + (indicator * -1)))
                upStates[LEFT_DIRECTION] = new Point(stateIndex.x + (indicator * -1), stateIndex.y);
            else
                upStates[LEFT_DIRECTION] = new Point(stateIndex.x, stateIndex.y);
            if(checkRowBorder((stateIndex.x) + indicator))
                upStates[RIGHT_DIRECTION] = new Point(stateIndex.x + indicator , stateIndex.y);
            else
                upStates[RIGHT_DIRECTION] = new Point(stateIndex.x, stateIndex.y);
        }
        return upStates;
    }
}
```