Project 2a Report

98552169 Zhen Wang

Introduction

This project is to use reinforcement learning to simulate the battles of the designed robot against the sample robot. It is implemented using Q-learning with lookup table. To simplify the problem, states and actions should be represented with much smaller state and action spaces. The chosen states include the distance between the robot and the target, the heading, the bearing and so on, and actions include going ahead, going back, turn to other directions and so on. The look up table is represented as a two-dimension matrix with state and action representing each dimension and the value is the Q-value. The result for the assignment is shown below.

Assignment

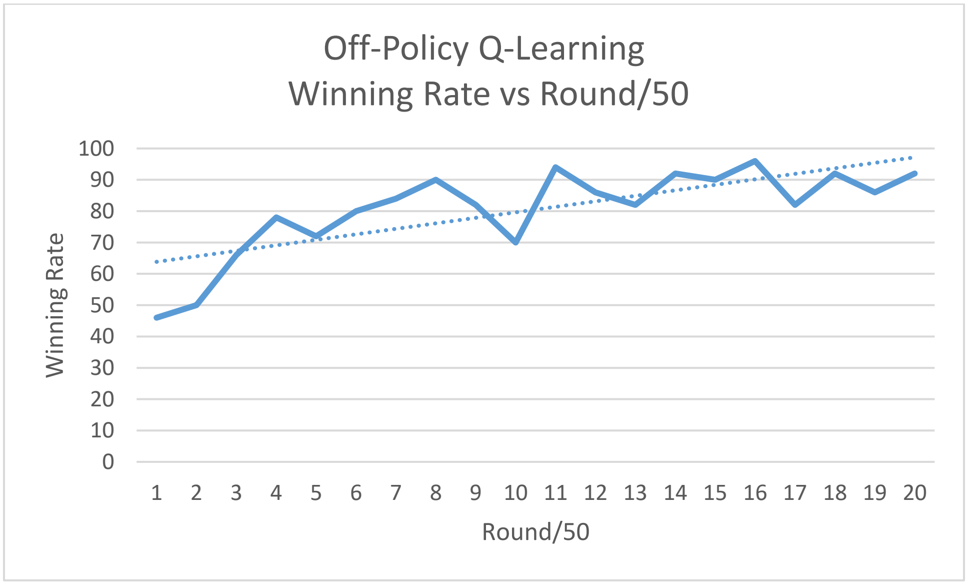
2) Once you have your robot working, measure its learning performance as follows:

a) Draw a graph of a parameter that reflects a measure of progress of learning and comment on the convergence of learning of your robot.

To show the convergence of learning, I chose to use winning rate vs round of battle. To make the graph shows more directly, I sampled the data every 50 rounds of battle and did 1000 rounds to see the convergence, which meant that there were 20 sample points shown in the graph. The target robot was set to be MyFirstRobot, as the result was much clearer for this target.

This part is about using off-policy Q-learning to train the robot against another. The exploration rate was set to be 0, the learning rate was set to be 0.1 and the discount rate was set to be 0.9.

In this case, the graph below showed the trend of the winning rate in 1000 rounds of battle. At the very beginning, the winning rate was less than 50%. With a quick increase, it was almost stable at around 90%.

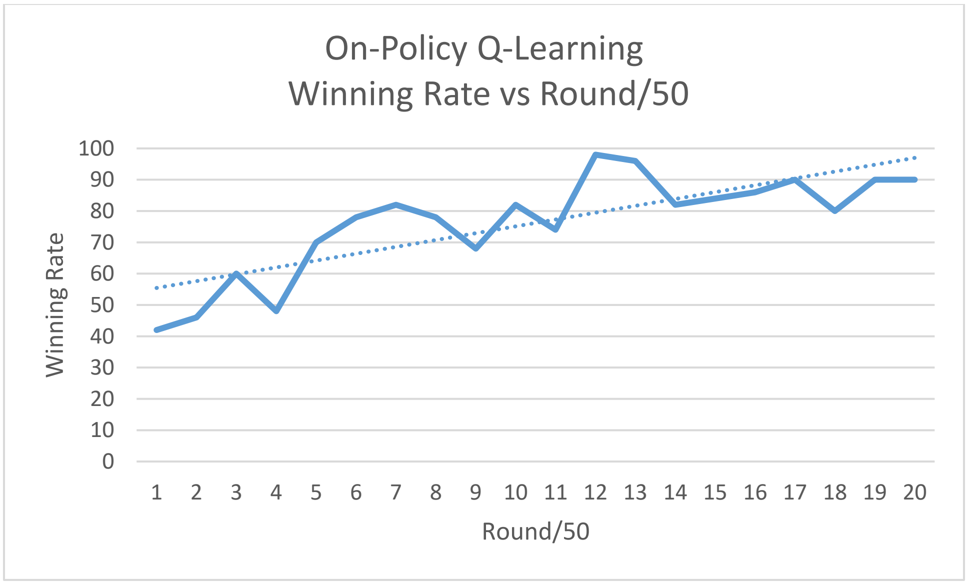


Graph 2a. off-policy Q-learning

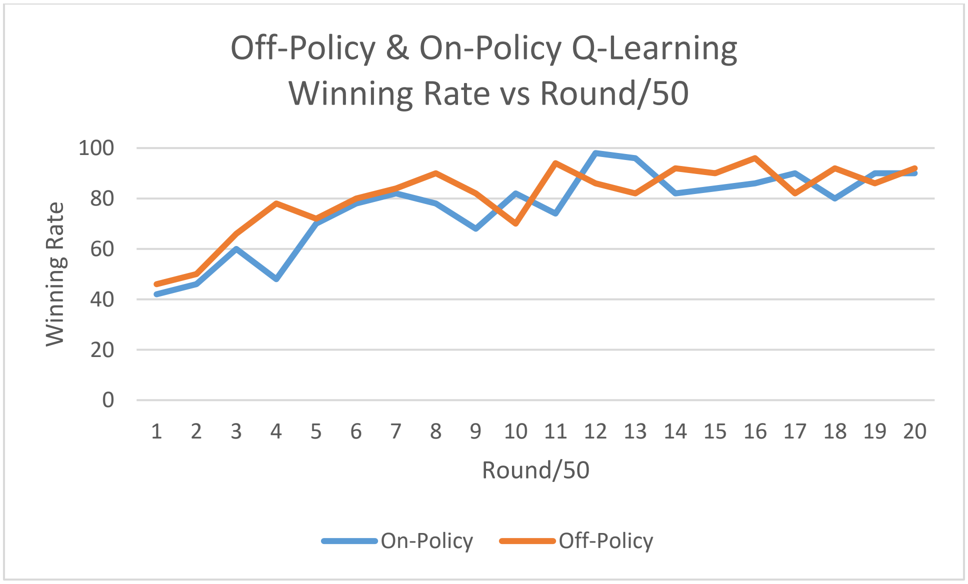
b) Using your robot, show a graph comparing the performance of your robot using on-policy learning vs off-policy learning.

This part is about using on-policy Q-learning to train the robot against another. The exploration rate was set to be 0, the learning rate was set to be 0.1 and the discount rate was set to be 0.9.

In this case, the graph below showed the trend of the winning rate in 1000 rounds of battle. At the very beginning, the winning rate was less than 50%. With a quick increase, it was almost stable at around 90%. So actually from the second graph in this part, the on-policy and the off-policy showed similar trend and even similar data line.



Graph 2b-1. on-policy Q-learning

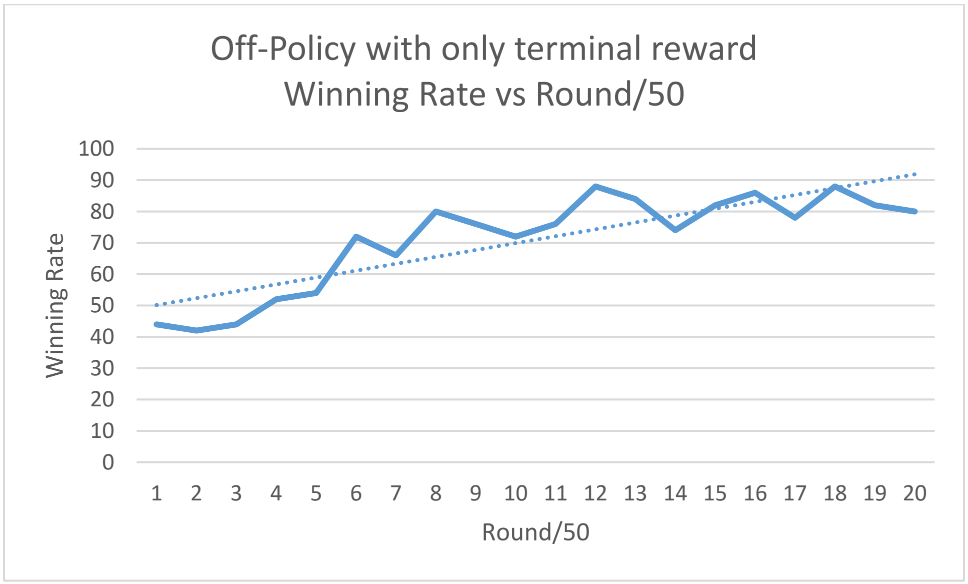


Graph 2b-2. off- and on- policy Q-learning

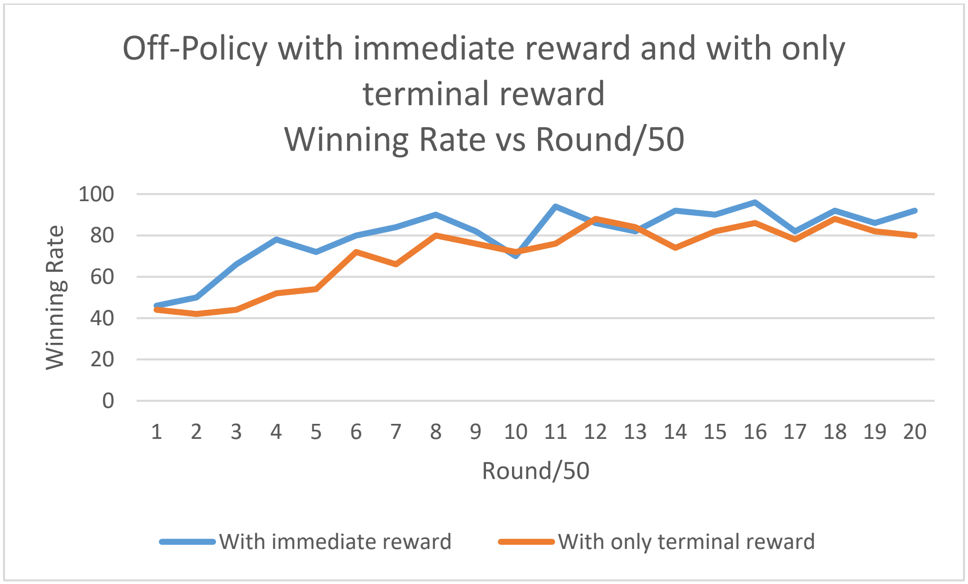
c) Implement a version of your robot that assumes only terminal rewards and show & compare its behaviour with one having intermediate rewards.

This part is about the effect of the terminal reward and the immediate reward. The exploration rate was set to be 0, the learning rate was set to be 0.1 and the discount rate was set to be 0.9. The terminal reward for winning was set to be 100, and that for losing was set to be -10. For the case with only terminal reward, other conditions like when hit by bullet, hit by the wall and so on would not generate rewards.

In this case, the graph below showed the trend of the winning rate in 1000 rounds of battle. At the very beginning, the winning rate was less than 50%. With an increase, it was almost stable at around 80%. From the second graph of this part, both of them converge but the winning rate for the case with only terminal rewards is obviously lower than the case with not only terminal rewards but also immediate rewards.



Graph 2c-1. off-policy Q-learning with only terminal reward



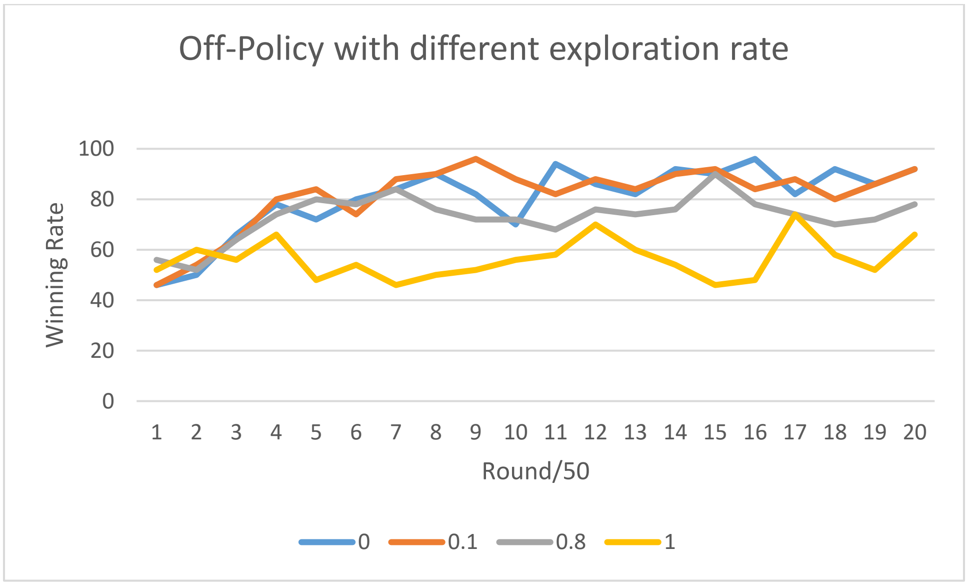
Graph 2c-2. effect of immediate rewards

3) This part is about exploration. While training via RL, the next move is selected randomly with probability ε and greedily with probability 1 − ε

a) Compare training performance using different values of ε including no exploration at all. Provide graphs of the measured performance of your tank vs ε.

This part is about the effect of exploration rate. The exploration rate means the probability to take random actions. When it is set to be 0, it means that all actions are chosen greedily. And when it is set to be 1, it means that all actions are taken randomly.

The graph shows the trends with exploration rates of 0, 0.1, 0.8 and 1. The learning rate was set to be 0.1 and the discount rate was set to be 0.9. It is clear that they converge in different winning rate although vibrate happens frequently. With exploration rate of 0 and 0.1, the two lines are similar to each other. The line with exploration rate of 1 is around 5 and seems not to increase but vibrate around 50%.



Graph 3a. effect of exploration rate

Source code

```
package myrobot;

import java.awt.*;
import java.awt.geom.*;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintStream;
import java.util.ArrayList;

import robocode.*;
```

```
public class QLearningBot extends AdvancedRobot
{
  public static final double PI = Math.PI;
  private Target target;
  private QTable table=new QTable();
  private Learner learner;
  private double reinforcement = 0.0;
  private double firePower;
  private int direction = 1;
  private int isHitWall = 0;
  private int isHitByBullet = 0;
  private ArrayList<Double> accumDiff=new
ArrayList<Double>();
  private QTable preTable=new QTable();
  double rewardForWin=100;
  double rewardForDeath=-10;
  double accumuReward=0.0;
  private static int round=0;
  private static int winTimes=0;

  public void run()
  {


    loadData();
    learner = new Learner(table);
    target = new Target();
    target.distance = 500;

    setColors(Color.green, Color.white, Color.green);
    setAdjustGunForRobotTurn(true);
    setAdjustRadarForGunTurn(true);
    turnRadarRightRadians(2 * PI);
    if(getRoundNum()>500)
    {
            learner.ExploitationRate=0.3;
     }

    while (true)
    {
      robotMovement();
      firePower = 3000/target.distance;
      if (firePower > 3)
        firePower = 3;
      radarMovement();
      gunMovement();
      if (getGunHeat() == 0) {
        setFire(firePower);
      }
      execute();
    }
  }

  void doMovement()
  {
    if (getTime()%20 == 0)
    {
      direction *= -1;                    //reverse direction
      setAhead(direction*300);      //move in that
direction
    }
    setTurnRightRadians(target.bearing + (PI/2));
//every turn move to circle strafe the enemy
  }
```

```
private void robotMovement()
{
  int state = getState();
  int action = learner.selectAction(state);
  out.println("Action selected: " + action);
  learner.learn(state, action, reinforcement);
  accumuReward+=reinforcement;
  reinforcement = 0.0;
  isHitWall = 0;
  isHitByBullet = 0;

  switch (action)
  {
    case Action.RobotAhead:
      setAhead(Action.RobotMoveDistance);
      break;
    case Action.RobotBack:
      setBack(Action.RobotMoveDistance);
      break;
    case Action.RobotAheadTurnLeft:
      setAhead(Action.RobotMoveDistance);
      setTurnLeft(Action.RobotTurnDegree);
      //setTurnLeft(180 - (target.bearing + 90 - 30));
      break;
    case Action.RobotAheadTurnRight:
      setAhead(Action.RobotMoveDistance);
      setTurnRight(Action.RobotTurnDegree);
      //setTurnRight(target.bearing + 90 - 30);
      break;
    case Action.RobotBackTurnLeft:
      setAhead(Action.RobotMoveDistance);
      setTurnRight(Action.RobotTurnDegree);
      //setTurnRight(target.bearing + 90 - 30);
      break;
    case Action.RobotBackTurnRight:
      setAhead(target.bearing);
      setTurnLeft(Action.RobotTurnDegree);
      //setTurnLeft(180 - (target.bearing + 90 - 30));
      break;
  }
}

  private int getState()
  {
    int heading = State.getHeading(getHeading());
    int targetDistance =
State.getTargetDistance(target.distance);
    int targetBearing =
State.getTargetBearing(target.bearing);
    out.println("State(" + heading + ", " +
targetDistance + ", " + targetBearing + ", " +
isHitWall + ", " + isHitByBullet + ")");
    int state =
State.Mapping[heading][targetDistance][targetBea
ring][isHitWall][isHitByBullet];
    return state;
  }

  private void radarMovement()
  {
    double radarOffset;
    if (getTime() - target.ctime > 4) { //if we haven't
seen anybody for a bit....
```

```
     radarOffset = 4*PI;
            //rotate the radar to find a target

   } else {


     //next is the amount we need to rotate the
radar by to scan where the target is now

     radarOffset = getRadarHeadingRadians() -
(Math.PI/2 - Math.atan2(target.y - getY(),target.x -
getX()));

     //this adds or subtracts small amounts from the
bearing for the radar to produce the wobbling

     //and make sure we don't lose the target

     radarOffset = NormaliseBearing(radarOffset);

     if (radarOffset < 0)

       radarOffset -= PI/10;

     else

       radarOffset += PI/10;

   }

   //turn the radar

   setTurnRadarLeftRadians(radarOffset);

 }


 private void gunMovement()

 {

   long time;

   long nextTime;

   Point2D.Double p;

   p = new Point2D.Double(target.x, target.y);

   for (int i = 0; i < 20; i++)

   {

     nextTime =
(int)Math.round((getrange(getX(),getY(),p.x,p.y)/(2
0-(3*firePower))));

     time = getTime() + nextTime - 10;

     p = target.guessPosition(time);

   }

   //offsets the gun by the angle to the next shot
based on linear targeting provided by the enemy
class

   double gunOffset = getGunHeadingRadians() -
(Math.PI/2 - Math.atan2(p.y - getY(),p.x -  getX()));


setTurnGunLeftRadians(NormaliseBearing(gunOffs
et));

  }


  //bearing is within the -pi to pi range

  double NormaliseBearing(double ang) {

    if (ang > PI)

      ang -= 2*PI;

    if (ang < -PI)

      ang += 2*PI;

    return ang;

  }


  //heading within the 0 to 2pi range

  double NormaliseHeading(double ang) {

    if (ang > 2*PI)

      ang -= 2*PI;

    if (ang < 0)

      ang += 2*PI;

    return ang;

  }


  //returns the distance between two x,y
coordinates

  public double getrange( double x1,double y1,
double x2,double y2 )
```

```
  {
    double xo = x2-x1;
    double yo = y2-y1;
    double h = Math.sqrt( xo*xo + yo*yo );
    return h;
  }

  //gets the absolute bearing between to x,y
coordinates
  public double absbearing( double x1,double y1,
double x2,double y2 )
  {
    double xo = x2-x1;
    double yo = y2-y1;
    double h = getrange( x1,y1, x2,y2 );
    if( xo > 0 && yo > 0 )
    {
      return Math.asin( xo / h );
    }
    if( xo > 0 && yo < 0 )
    {
      return Math.PI - Math.asin( xo / h );
    }
    if( xo < 0 && yo < 0 )
    {
      return Math.PI + Math.asin( -xo / h );
    }
    if( xo < 0 && yo > 0 )
    {
      return 2.0*Math.PI - Math.asin( -xo / h );
    }
    return 0;
  }

  public void onBulletHit(BulletHitEvent e)
  {
    if (target.name == e.getName())
    {
      //double power = e.getBullet().getPower();
      //double change = 4 * power + 2 * (power - 1);
      double change = e.getBullet().getPower() * 9;
      out.println("Bullet Hit: " + change);
      reinforcement += change;
    }
  }

  public void onBulletHitBullet(BulletHitBulletEvent
e)
  {
    //
  }

  public void onBulletMissed(BulletMissedEvent e)
  {
    double change = -e.getBullet().getPower();
    out.println("Bullet Missed: " + change);
    reinforcement += change;
  }

 /* public void onHitByBullet(HitByBulletEvent e)
  {
    if (target.name == e.getName())
    {
      double power = e.getBullet().getPower();
      double change = -(4 * power + 2 * (power - 1));
      out.println("Hit By Bullet: " + change);
      reinforcement += change;
    }
    isHitByBullet = 1;
   }

   public void onHitRobot(HitRobotEvent e)
   {
    if (target.name == e.getName())
    {
      double change = -6.0;
      out.println("Hit Robot: " + change);
      reinforcement += change;
    }
   }

  /* public void onHitWall(HitWallEvent e)
   {

    double change = -(Math.abs(getVelocity()) * 0.5
  - 1);
    out.println("Hit Wall: " + change);
    reinforcement += change;
    isHitWall = 1;
   }

   /**
    * onScannedRobot: What to do when you see
  another robot
    */
```

```
 public void onScannedRobot(ScannedRobotEvent
e)
 {
   if ((e.getDistance() <
target.distance)||(target.name == e.getName()))
   {
     //the next line gets the absolute bearing to the
point where the bot is
     double absbearing_rad =
(getHeadingRadians()+e.getBearingRadians())%(2*
PI);
     //this section sets all the information about our
target
     target.name = e.getName();
     double h =
NormaliseBearing(e.getHeadingRadians() -
target.head);
     h = h/(getTime() - target.ctime);
     target.changehead = h;
     target.x =
getX()+Math.sin(absbearing_rad)*e.getDistance();
//works out the x coordinate of where the target is
     target.y =
getY()+Math.cos(absbearing_rad)*e.getDistance();
//works out the y coordinate of where the target is
     target.bearing = e.getBearingRadians();
     target.head = e.getHeadingRadians();
     target.ctime = getTime();
             //game time at which this scan was
produced
     target.speed = e.getVelocity();
     target.distance = e.getDistance();
     target.energy = e.getEnergy();
   }
 }

 public void onRobotDeath(RobotDeathEvent e)
 {

   if (e.getName() == target.name)
     target.distance = 500;
 }

 public void onWin(WinEvent event)
 {

             reinforcement+=rewardForWin;
             accumuReward+=rewardForWin;
     robotMovement();
     int sum=0;
     for(int i=0;i<State.NumStates;i++)
             {
                        for(int
j=0;j<Action.NumRobotActions;j++){

            sum+=Math.pow(preTable.getQValue(i, j)-
table.getQValue(i, j), 2);
                        }
             }
     round++;
     winTimes++;
     saveData();
                        int winningFlag=7;
                        File file =
getDataFile("accumReward.dat");
                        //PrintStream stream = new
PrintStream(new
RobocodeFileOutputStream(file.getAbsolutePath(),
true);

                        PrintStream w = null;
```

```
                              try
                              {
                                w = new PrintStream(new
RobocodeFileOutputStream(file.getAbsolutePath(),
true));
                                //w.println(accumuReward+"
"+getRoundNum()+"\t"+winningFlag+" "+"
"+learner.ExploitationRate+" "+sum);
                                if(round==10){
                                          w.println(winTimes+"
"+Math.abs(accumuReward)+" "+sum);
                                          winTimes=0;
                                          round=0;
                                }
                                if (w.checkError())
                                  System.out.println("Could not
save the data!");  //setTurnLeft(180 -
(target.bearing + 90 - 30));
                                w.close();
                              }
                              catch (IOException e)
                              {

System.out.println("IOException trying to write: " +
e);
                              }
                              finally
                              {
                                try
                                {
                                  if (w != null)
                                    w.close();
                                }
                                catch (Exception e)
                                {
```

```
                            System.out.println("Exception
trying to close witer: " + e);

                            }

                          }

  }


  public void onDeath(DeathEvent event)

  {

              saveData();

              int sum=0;

              for(int i=0;i<State.NumStates;i++)

                {

                        for(int
j=0;j<Action.NumRobotActions;j++){


             sum+=Math.pow(preTable.getQValue(i, j)-
table.getQValue(i, j), 2);

                        }

                }

              round++;

              reinforcement+=rewardForDeath;

              accumuReward+=rewardForDeath;

             robotMovement();

             File file = getDataFile("accumReward.dat");


       int losingFlag=5;

                        PrintStream w = null;

                          try

                          {

                            w = new PrintStream(new
RobocodeFileOutputStream(file.getAbsolutePath(),
true));
                    //w.println(accumuReward+"
"+getRoundNum()+"\t"+losingFlag+" "+"
"+learner.ExploitationRate+" "+sum);

                    if(round==10){

                          w.println(winTimes+"
"+Math.abs(accumuReward)+" "+sum);

                          winTimes=0;

                          round=0;

                    }

                    if (w.checkError())

                      System.out.println("Could not
save the data!");

                    w.close();

                  }

                  catch (IOException e)

                  {


System.out.println("IOException trying to write: " +
e);

                  }

                  finally

                  {

                    try

                    {

                      if (w != null)

                        w.close();

                    }

                    catch (Exception e)

                    {

                      System.out.println("Exception
trying to close witer: " + e);

                    }


                  }
```

```
 }

 public static void writeDiff(String file, String
conent) {
            BufferedWriter out = null;
            try {
                out = new BufferedWriter(new
OutputStreamWriter(
                     new FileOutputStream(file,
true)));
                    out.write(conent);
               } catch (Exception e) {
                  e.printStackTrace();
               } finally {
                  try {
                     out.close();
                  } catch (IOException e) {
                     e.printStackTrace();
                  }
               }
            }

 public void saveDiff(ArrayList<Double>
accumDiff,File file){
           PrintStream w = null;
            try
            {
             w = new PrintStream(new
RobocodeFileOutputStream(file));
             for(int i=0;i<accumDiff.size();i++){
                    w.println(new
Double(accumDiff.get(i)));
             }
              if (w.checkError())
                System.out.println("Could not save
the data!");
              w.close();
            }
            catch (IOException e)
            {
              System.out.println("IOException trying
to write: " + e);
            }
            finally
            {
              try
              {
                if (w != null)
                  w.close();
              }
              catch (Exception e)
              {
                System.out.println("Exception trying
to close witer: " + e);
              }
            }
 }
 public void loadDiff(File file,ArrayList<Double>
accumDiff)
 {
   BufferedReader r = null;
   try
   {
     r = new BufferedReader(new FileReader(file));
```

```
      String a=null;
      while((a=r.readLine())!=null){

accumDiff.add(Double.parseDouble(r.readLine()));
      }
    }
    catch (IOException e)
    {
      System.out.println("IOException trying to open
reader: " + e);
    }
    catch (NumberFormatException e)
    {
    }
    finally
    {
      try
      {
        if (r != null)
          r.close();
      }
      catch (IOException e)
      {
        System.out.println("IOException trying to
close reader: " + e);
      }
    }
  }
  public void loadData()
  {
    try
    {
            preTable.loadData(getDataFile("movemen
t.dat"));

            table.loadData(getDataFile("movement.da
t"));
    }
    catch (Exception e)
    {
    }
  }

  public void saveData()
  {
              /*double sum=0;
              for(int i=0;i<State.NumStates;i++)
               {
                         for(int
j=0;j<Action.NumRobotActions;j++){

            sum+=Math.pow(preTable.getQValue(i, j)-
table.getQValue(i, j), 2);
                         }
               }

//loadDiff(getDataFile("accumDiff.dat"),accumDiff)
;
              accumDiff.add(sum);

saveDiff(accumDiff,getDataFile("accumDiff.dat"));




                //for(int k=0;k<accumDiff.size();k++){
                    //System.out.println("accumDiff
is : ");

            //System.out.print(accumDiff.get(1));
                    //System.out.println("\t");
              //}
                     */
    try
    {
      table.saveData(getDataFile("movement.dat"));
    }
    catch (Exception e)
    {
      out.println("Exception trying to write: " + e);
    }
  }
}

package myrobot;

import java.io.*;
import robocode.*;

public class QTable
{
  private double[][] table;

  public QTable()
  {
    table = new
double[State.NumStates][Action.NumRobotAction
s];
    initialize();
  }

  private void initialize()
  {
    for (int i = 0; i < State.NumStates; i++)
      for (int j = 0; j < Action.NumRobotActions; j++)
        table[i][j] = 0.0;
  }

  public double getMaxQValue(int state)
  {
    double maxinum = Double.NEGATIVE_INFINITY;
    for (int i = 0; i < table[state].length; i++)
    {
      if (table[state][i] > maxinum)
        maxinum = table[state][i];
    }
    return maxinum;
  }

  public int getBestAction(int state)
  {
    double maxinum = Double.NEGATIVE_INFINITY;
    int bestAction = 0;
    for (int i = 0; i < table[state].length; i++)
    {
      double qValue = table[state][i];
      //System.out.println("Action " + i + ": " +
qValue);
      if (qValue > maxinum)
      {
        maxinum = qValue;
      bestAction = i;
     }
    }
   return bestAction;
  }

  public double getQValue(int state, int action)
  {
   return table[state][action];
  }

  public void setQValue(int state, int action, double
value)
  {
   table[state][action] = value;
  }

  public void loadData(File file)
  {
   BufferedReader r = null;
   try
   {
    r = new BufferedReader(new FileReader(file));
    for (int i = 0; i < State.NumStates; i++)
     for (int j = 0; j < Action.NumRobotActions; j++)
      table[i][j] =
Double.parseDouble(r.readLine());
   }
   catch (IOException e)
   {
    System.out.println("IOException trying to open
reader: " + e);
    initialize();
   }
   catch (NumberFormatException e)
   {
    initialize();
   }
   finally
   {
    try
    {
     if (r != null)
      r.close();
    }
    catch (IOException e)
    {
     System.out.println("IOException trying to
close reader: " + e);
    }
   }
  }

  public void saveData(File file)
  {
   PrintStream w = null;
   try
   {
    w = new PrintStream(new
RobocodeFileOutputStream(file));
    for (int i = 0; i < State.NumStates; i++)
     for (int j = 0; j < Action.NumRobotActions; j++)
      w.println(new Double(table[i][j]));
      if (w.checkError())
        System.out.println("Could not save the
data!");
      w.close();
    }
    catch (IOException e)
    {
      System.out.println("IOException trying to
write: " + e);
    }
    finally
    {
      try
      {
        if (w != null)
          w.close();
      }
      catch (Exception e)
      {
        System.out.println("Exception trying to close
witer: " + e);
      }
    }
  }
}

package myrobot;

import java.util.Random;

public class Learner
{
  public static final double LearningRate = 0.1;
  public static final double DiscountRate = 0.9;
  public static double ExploitationRate = 1;
  private int lastState;
  private int lastAction;
  private boolean first = true;
  private QTable table;

  public Learner(QTable table)
  {
    this.table = table;
  }

  public void learn(int state, int action, double
reward)
  {
    System.out.println("Reinforcement: " + reward);
    if (first)
      first = false;
    else
    {
      double oldQValue = table.getQValue(lastState,
lastAction);
      double newQValue = (1 - LearningRate) *
oldQValue + LearningRate * (reward +
DiscountRate * table.getMaxQValue(state));
      System.out.println("Old Q-Value: " + oldQValue
+ ", New Q-Value: " + newQValue + ", Different: " +
(newQValue - oldQValue));
      table.setQValue(lastState, lastAction,
newQValue);
    }
    lastState = state;

    lastAction = action;

  }

  public void learnSARSA(int state, int action, 
double reward) {

                        if (first)

                                    first = false;

                        else {

                                    double oldQValue = 
table.getQValue(lastState, lastAction);


                                    double newQValue = (1 - 
LearningRate) * oldQValue

                                                                  + 
LearningRate * (reward + DiscountRate * 
table.getQValue(state, action));


                                    table.setQValue(lastState, 
lastAction, newQValue);

                        }

                        lastState = state;

                        lastAction = action;

            }


  /*public int selectAction(int state)

  {

    double qValue;

    double sum = 0.0;

    double[] value = new 
double[Action.NumRobotActions];

    for (int i = 0; i < value.length; i++)

    {

      qValue = table.getQValue(state, i);

      value[i] = Math.exp(ExploitationRate * qValue);

      sum += value[i];

      System.out.println("Q-value: " + qValue);

    }


    if (sum != 0)

      for (int i = 0; i < value.length; i++)

      {

        value[i] /= sum;

        System.out.println("P(a|s): " + value[i]);

      }

    else

      return table.getBestAction(state);


    int action = 0;

    double cumProb = 0.0;

    double randomNum = Math.random();

    System.out.println("Random Number: " + 
randomNum);

    while (randomNum > cumProb && action < 
value.length)

    {

      cumProb += value[action];

      action++;

    }

    return action - 1;

  } */

  public int selectAction(int state)

  {


                        double thres = Math.random();


                        int actionIndex = 0;
                        if (thres<ExploitationRate)
                        {//randomly select one action
from action(0,1,2)
                                    Random ran = new
Random();
                                    actionIndex =
ran.nextInt(((Action.NumRobotActions-1 - 0) + 1));
                        }
                        else
                        {//e-greedy

            actionIndex=table.getBestAction(state);
                        }
                        return actionIndex;
            }



}
```