UBC-CPEN-502 Assignment 2

Reinforcement Learning (Look Up Table)

Name: Yue Shen Student #: 76006386

(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 my learning progress obviously, I choose to battle with robot *TrackFire*,

The total number of rounds is **2000**. With **every 50** rounds, I calculate the winning rates, which results in forty winning rates.

Table-(2)a Settings for (2)a

Learning Rate	Discount Rate	Exploration Rate	Policy	Rewards
0.1	0.9	0.0	Off	Intermediate and Terminal

With other basic parameters set according to Table-(2)a, the trend graph of these forty wining rates is shown as Figure-(2)a.

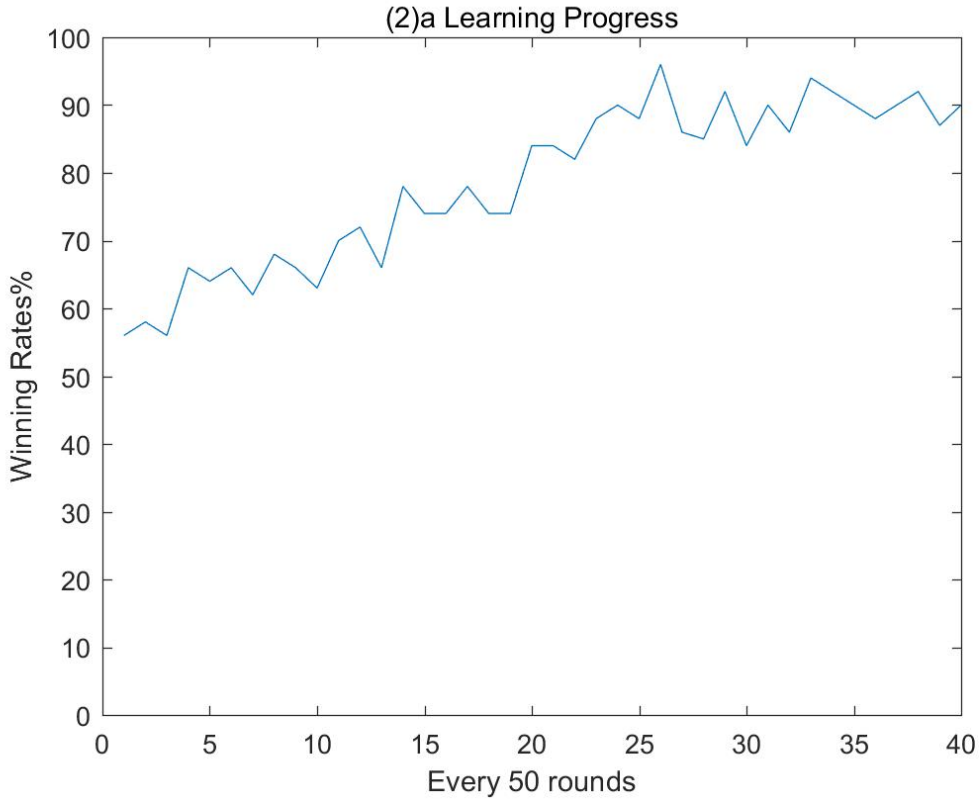

Figure-(2)a Learning progress(off policy)

As we can see, during the first 20 x 50 rounds, the winning rate increases from around 60% to 80%. After that, my robot's winning rate can reach 90% and slightly fluctuates between 85% and 95%.

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

Table-(2)b Settings for (2)b

Learning Rate	Discount Rate	Exploration Rate	**Policy**	Rewards
0.1	0.9	0.0	Off / On	Intermediate and Terminal

With other basic parameters set according to Table-(2)b, the blue curve in Figure-(2)b shows the learning progress of off-policy, the red curve in Figure-(2)b shows the learning progress of off-policy (same as the last figure.)

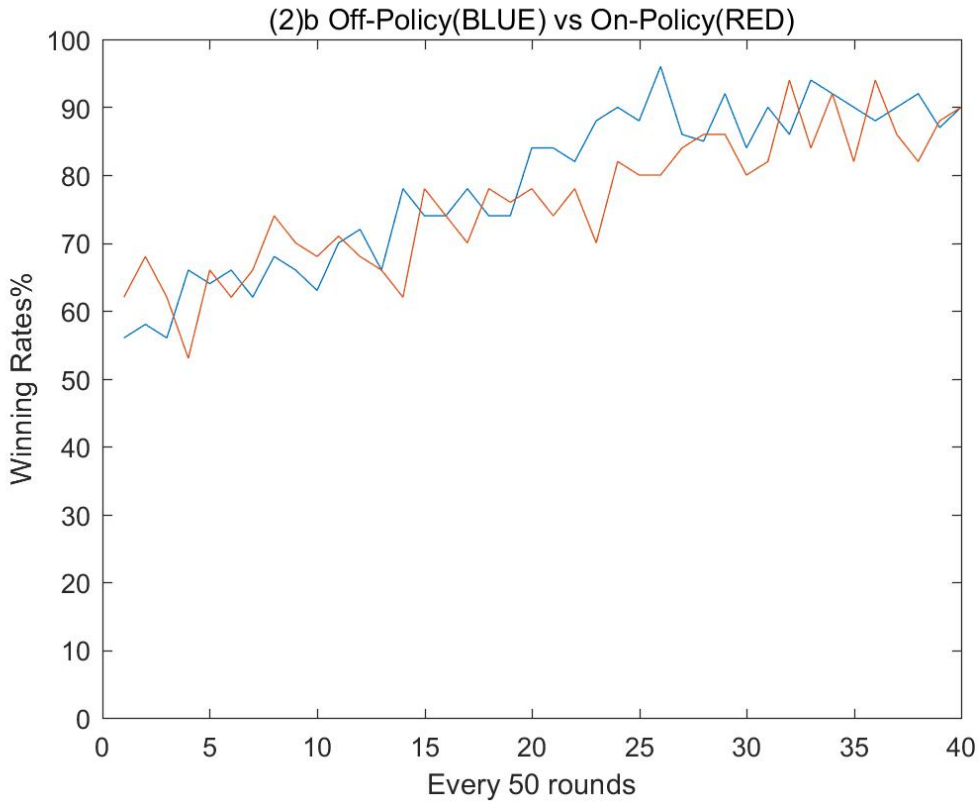

Figure-(2)b Off-policy vs On-policy

With On-policy, the performance could get better by training. Nevertheless, the curve of Off-policy is more stable than of On-policy. Also, with Off-policy, the winning rate gets faster to reach 90%.

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

Table-(2)c Settings for (2)c

Learning Rate	Discount Rate	Exploration Rate	Policy	**Rewards**
0.1	0.9	0.0	Off	Intermediate and Terminal / Only Terminal

With other basic parameters set according to Table-(2)c, the blue curve in Figure-(2)c shows the learning progress with intermediate and terminal rewards, the red curve in Figure-(2)c shows the learning progress with only terminal rewards.

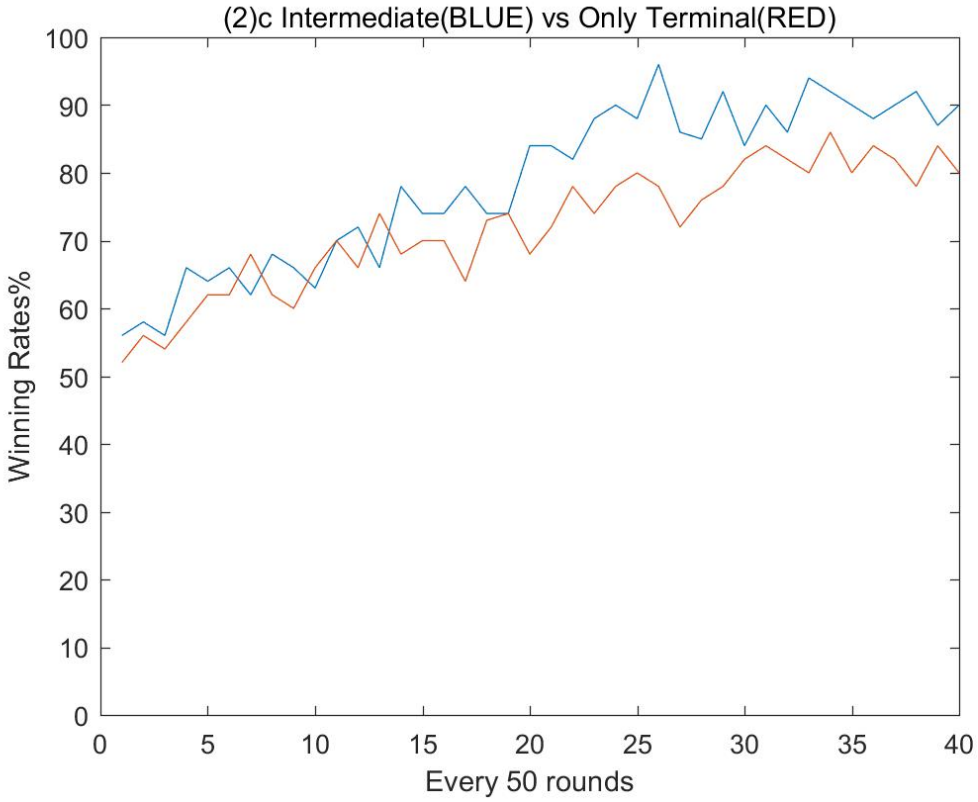

Figure-(2)c Intermediate vs Only Terminal

Both two curves increase during first 1000 rounds, however the red curve (only terminal rewards) is hard to reach 90% after that, which means having no intermediate rewards will result in worse performance.

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

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

Table-(3)a Settings for (3)a

Learning Rate	Discount Rate	**Exploration Rate**	Policy	Rewards
0.1	0.9	0.0 / 0.3 / 0.6 / 1.0	Off	Intermediate and Terminal

Since it's not clear to put all four curves in one single graph, I choose to split them into two graphs.

The figure-(3)a.1 shows the performance with exploration rate 0.0(Blue) and 1.0(Red), and the figure-(3)a.2 shows the performance with exploration rate 0.3(Yellow) and 0.6(Purple).

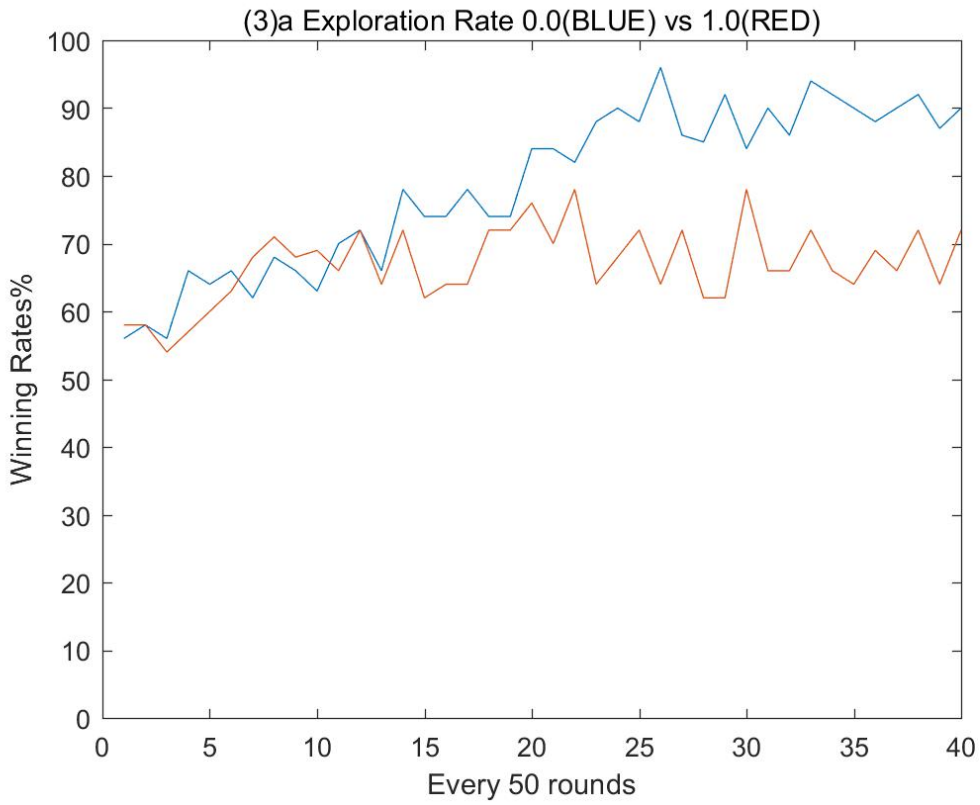

Figure-(3)a.1 Exploration rate 0.0 vs 1.0

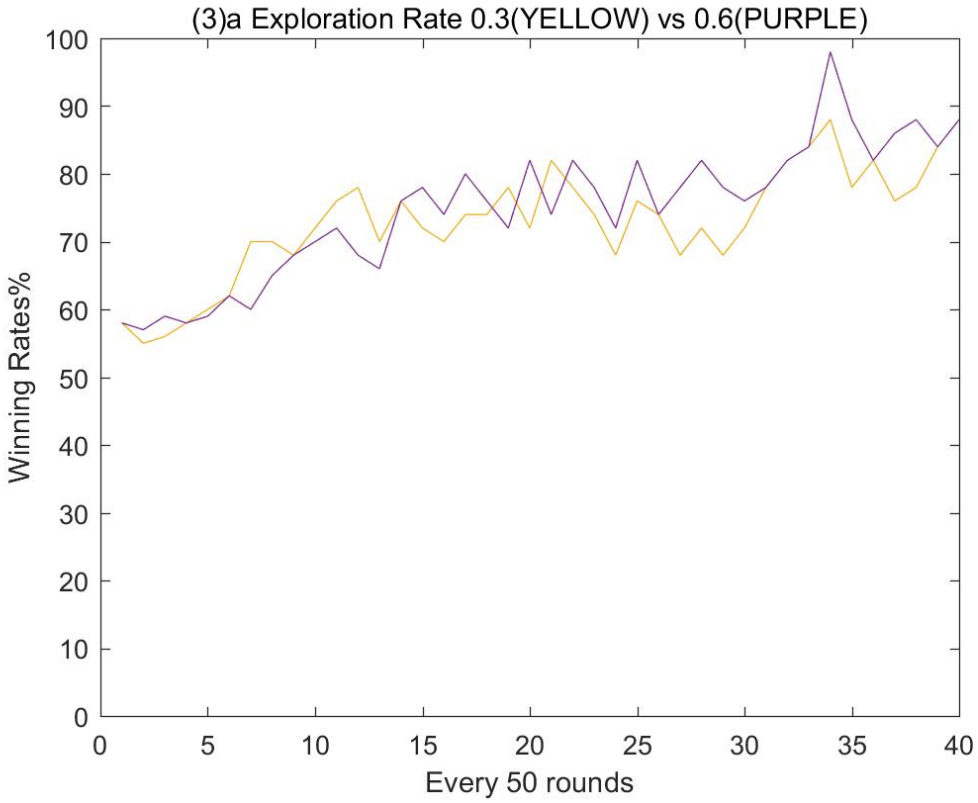

Figure-(3)a.2 Exploration rate 0.3 vs 0.6

There is no huge difference between learning trends with exploration rate 0.3 and learning trends with exploration rate 0.6.

However, with exploration rate 1.0, the winning rate always fluctuates around 60% to 70%, failing to increase. All in all, learning with exploration rate 0.0 outputs the best performance.

Appendix

1. LUT.java

```
package com.shenyue;

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

public class LUT implements LUTInterface {
    // States Arguments
    public static final int headingNum = 4;
    public static final int OpponentDistanceNum = 4;
    public static final int OpponentBearingNum = 4;
    public static final int XPositionNum = 8;
    public static final int YPositionNum = 6;
    public static final int statesNum = headingNum * OpponentDistanceNum *
OpponentBearingNum * XPositionNum * YPositionNum;
    public static final int actionsNum = 5;
    public static int States[][][][][][] = new
int[headingNum][OpponentDistanceNum][OpponentBearingNum][XPositionNum][YPos
itionNum];

    public static int getHeading(double argValue) {
        // Four kinds of heading
        int heading = 0;
        if (argValue >= 0 && argValue < Math.PI / 2)
            heading = 0;
        if (argValue >= Math.PI / 2 && argValue < Math.PI)
            heading = 1;
        if (argValue >= Math.PI && argValue < Math.PI * 3 / 2)
            heading = 2;
        if (argValue >= Math.PI * 3 / 2)
            heading = 3;
        return heading;
    }

    public static int getOpponentDistance(double argValue) {
        // Four kinds of distance: close, near, far, very far
        int distance = (int) (argValue / 100);
        if (distance > OpponentDistanceNum - 1)
            distance = OpponentDistanceNum - 1;
        return distance;
    }
```

```
public static int getOpponentBearing(double argValue) {
    // Four kinds of bearing
    int bearing = 0;
    argValue = argValue + Math.PI;
    if (argValue >= 0 && argValue < Math.PI / 2)
        bearing = 0;
    if (argValue >= Math.PI / 2 && argValue < Math.PI)
        bearing = 1;
    if (argValue >= Math.PI && argValue < Math.PI * 3 / 2)
        bearing = 2;
    if (argValue >= Math.PI * 3 / 2)
        bearing = 3;
    return bearing;
}

public static int getXPosition(double argValue) {
    int x = (int) (argValue / 100);
    if (x >= 0 && x < 1)
        x = 0;
    if (x >= 1 && x < 2)
        x = 1;
    if (x >= 2 && x < 3)
        x = 2;
    if (x >= 3 && x < 4)
        x = 3;
    if (x >= 4 && x < 5)
        x = 4;
    if (x >= 5 && x < 6)
        x = 5;
    if (x >= 6 && x < 7)
        x = 6;
    if (x >= 7 && x <= 8)
        x = 7;
    return x;
}

public static int getYPosition(double argValue) {
    int y = (int) (argValue / 100);
    if (y >= 0 && y < 1)
        y = 0;
    if (y >= 1 && y < 2)
        y = 1;
    if (y >= 2 && y < 3)
        y = 2;
```

```
    if (y >= 3 && y < 4)
        y = 3;
    if (y >= 4 && y < 5)
        y = 4;
    if (y >= 5 && y < 6)
        y = 5;
    return y;
}

// Look Up Table
public double[][] LUTable;

public LUT() {
    LUTable = new double[statesNum][actionsNum];
    initialiseLUT(); // set all function values to be zero
}

@Override
public void initialiseLUT() {
    // Initialize table to zero
    for (int i = 0; i < statesNum; i++)
        for (int j = 0; j < actionsNum; j++)
            LUTable[i][j] = 0.0;

    // Initialize States
    int count = 0;
    for (int a = 0; a < headingNum; a++)
        for (int b = 0; b < OpponentDistanceNum; b++)
            for (int c = 0; c < OpponentBearingNum; c++)
                for (int d = 0; d < XPositionNum; d++)
                    for (int e = 0; e < YPositionNum; e++)
                        States[a][b][c][d][e] = count++;
}

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

public int getBestAction(int state) {
```

```
    double maxQ = Double.NEGATIVE_INFINITY;
    int bestAction = 0;
    for (int i = 0; i < LUTable[state].length; i++) {
        if (LUTable[state][i] > maxQ) {
            maxQ = LUTable[state][i];
            bestAction = i;
        }
    }
    return bestAction;
}

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

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

@Override
public int indexFor(double[] X) {
    return 2;
}

@Override
public double outputFor(double[] X) {
    return 2;
}

@Override
public double train(double[] X, double argValue) {
    return 0;
}

@Override
public void load(File file) {
    BufferedReader read = null;
    try {
        read = new BufferedReader(new FileReader(file));
        for (int i = 0; i < statesNum; i++)
            for (int j = 0; j < actionsNum; j++)
                LUTable[i][j] = Double.parseDouble(read.readLine());
    } catch (IOException e) {
        initialiseLUT();
```

```
            } catch (NumberFormatException e) {
                initialiseLUT();
            } finally {
                try {
                    if (read != null)
                        read.close();
                } catch (IOException e) {

                }
            }
        }

        @Override
        public void save(File file) {
            PrintStream write = null;
            try {
                write = new PrintStream(new RobocodeFileOutputStream(file));
                for (int i = 0; i < statesNum; i++)
                    for (int j = 0; j < actionsNum; j++)
                        write.println(LUTable[i][j]);

                if (write.checkError())
                    System.out.println("Could not save the data!");
                write.close();
            } catch (IOException e) {

            } finally {
                try {
                    if (write != null)
                        write.close();
                } catch (Exception e) {

                }
            }
        }

    }
```

2. Learning.java

```
package com.shenyue;

import java.util.Random;

public class Learning {
    public static final double learningRate = 0.1;
    public static final double discountRate = 0.9;
    public static double explorationRate = 0.0;
    private int previousState;
    private int previousAction;
    public LUT LUTable;
    public Learning(LUT LUTable) {
        this.LUTable = LUTable;
    }

    public void LUTlearning(int currentState, int currentAction, double reward,
boolean offPolicy) {
        double previousQ = LUTable.getQValue(previousState, previousAction);
        if (offPolicy) {
            double currentQ = (1 - learningRate) * previousQ
                    + learningRate * (reward + discountRate *
LUTable.getMaxQ(currentState));
            LUTable.setQValue(previousState, previousAction, currentQ);
        } else { // onPolicy
            double currentQ = (1 - learningRate) * previousQ
                    + learningRate * (reward + discountRate *
LUTable.getQValue(currentState, currentAction));
            LUTable.setQValue(previousState, previousAction, currentQ);
        }
        previousState = currentState;
        previousAction = currentAction;
    }

    public int selectAction(int state) {
        double random = Math.random();
        if (explorationRate > random) {
            Random ran = new Random();
            return ran.nextInt(((LUT.actionsNum - 1 - 0) + 1));
        } else { // Pure greedy
            return LUTable.getBestAction(state);
        }
    }
}
```

3. syRobot.java

```
package com.shenyue;

import java.awt.*;
import java.awt.geom.*;
import java.io.IOException;
import java.io.PrintStream;

import robocode.*;

public class syRobot extends AdvancedRobot {
  // Action
  public static final int moveAhead = 0;
  public static final int moveBack = 1;
  public static final int turnLeft = 2;
  public static final int turnRight = 3;
  public static final int robotfire = 4;
  public static final double aheadDistance = 150.0;
  public static final double backDistance = 100.0;
  public static final double turnDegree = 20.0;
  public static final int actionsNum = 5;

  // Opponent
  public String opponentName;
  public double opponentSpeed;
  public double opponentBearing;
  public long opponentTime;
  public double opponentX;
  public double opponentY;
  public double opponentDistance;
  public double opponentHead;
  public double opponentChangehead;
  public double opponentEnergy;

  public Learning learning;
  private double firePower;
  public static int count;
  public static int winCount;
  public static double reward;
  public static double winningRates;

  public static boolean intermediate = true; // false for only terminal reward
  public static boolean offPolicy = true; // false for on-Policy
```

```
public double goodReward = 5;
public double badReward = -5;


public void run() {
  learning = new Learning(new LUT());
  loadTable();
  opponentDistance = 1000; // Set Opponent distance to 'far'
  // Set my syRobot
  setColors(Color.white, Color.pink, Color.pink);
  setAdjustGunForRobotTurn(true);
  setAdjustRadarForGunTurn(true);
  turnRadarRightRadians(2 * Math.PI);

  while (true) {
    int state = getState();
    int action = learning.selectAction(state);
    learning.LUTlearning(state, action, reward, offPolicy);
    reward = 0.0;
    switch (action) {
    case moveAhead:
      setAhead(aheadDistance);
      break;
    case moveBack:
      setBack(backDistance);
      break;
    case turnLeft:
      setTurnLeft(turnDegree);
      break;
    case turnRight:
      setTurnRight(turnDegree);
      break;
    case robotfire:
      fire(1);
      break;
    }
    radarAction();
    gunAction(2);
    execute();
  }
}

private int getState() {
  int heading = LUT.getHeading(getHeading());
```

```
    int oppoDistance = LUT.getOpponentDistance(opponentDistance);
    int oppoBearing = LUT.getOpponentBearing(opponentBearing);
    int x = LUT.getXPosition(getX());
    int y = LUT.getYPosition(getY());
    int state = LUT.States[heading][oppoDistance][oppoBearing][x][y];
    return state;
  }

  private void radarAction() {
    double radarRotate;
    if (getTime() - opponentTime > 4) {
      radarRotate = 4 * Math.PI; // Rotate radar to find an opponent
    } else {
      radarRotate = getRadarHeadingRadians() - (Math.PI / 2 -
Math.atan2(opponentY - getY(), opponentX - getX()));
      radarRotate = nomslizeBearing(radarRotate);
      if (radarRotate < 0)
        radarRotate -= Math.PI / 10;
      else
        radarRotate += Math.PI / 10;
    }
    setTurnRadarLeftRadians(radarRotate);
  }

  private void gunAction(double power) {
    long currentTime;
    long nextTime;
    Point2D.Double opponentPosition = new Point2D.Double(opponentX,
opponentY);
    // Distance between my robot and opponent
    double distance = Math.sqrt((opponentPosition.x - getX()) *
(opponentPosition.x - getX())
        + (opponentPosition.y - getY()) * (opponentPosition.y - getY()));
    for (int i = 0; i < 20; i++) {
      // Calculate time to reach the opponent
      nextTime = (int) Math.round(distance / (20 - 3 * firePower));
      currentTime = getTime() + nextTime;
      opponentPosition = guessPosition(currentTime);
    }
    // Set off the gun
    double gunOffSet = getGunHeadingRadians()
        - (Math.PI / 2 - Math.atan2(opponentPosition.y - getY(),
opponentPosition.x - getX()));
    setTurnGunLeftRadians(nomslizeBearing(gunOffSet));
```

```
    if (getGunHeat() == 0) {
      setFire(power);
    }
  }

  public void onScannedRobot(ScannedRobotEvent e) {
    if ((e.getDistance() < opponentDistance) || (opponentName == e.getName()))
{
      double absBearing = (getHeadingRadians() + e.getBearingRadians()) % (2 *
Math.PI);
      opponentName = e.getName();
      double head = nomslizeBearing(e.getHeadingRadians() - opponentHead);
      head = head / (getTime() - opponentTime);
      opponentChangehead = head;
      opponentX = getX() + Math.sin(absBearing) * e.getDistance();
      opponentY = getY() + Math.cos(absBearing) * e.getDistance();
      opponentBearing = e.getBearingRadians();
      opponentHead = e.getHeadingRadians();
      opponentTime = getTime();
      opponentSpeed = e.getVelocity();
      opponentDistance = e.getDistance();
      opponentEnergy = e.getEnergy();
    }
  }

  public Point2D.Double guessPosition(long time) {
    double newX, newY;
    if (Math.abs(opponentChangehead) > 0.00001) {
      double radius = opponentSpeed / opponentChangehead;
      double totalHead = (time - opponentTime) * opponentChangehead;
      newX = opponentX + (Math.cos(opponentHead) * radius) -
(Math.cos(opponentHead + totalHead) * radius);
      newY = opponentY + (Math.sin(opponentHead + totalHead) * radius) -
(Math.sin(opponentHead) * radius);
    } else {
      newX = opponentX + Math.sin(opponentHead) * opponentSpeed * (time -
opponentTime);
      newY = opponentY + Math.cos(opponentHead) * opponentSpeed * (time -
opponentTime);
    }
    return new Point2D.Double(newX, newY);
  }

  double nomslizeBearing(double argValue) {
```

```
  if (argValue > Math.PI)
    argValue -= 2 * Math.PI;
  if (argValue < -Math.PI)
    argValue += 2 * Math.PI;
  return argValue;
}

double nomalizeHeading(double argValue) {
  if (argValue > 2 * Math.PI)
    argValue -= 2 * Math.PI;
  if (argValue < 0)
    argValue += 2 * Math.PI;
  return argValue;
}

public void onBulletHit(BulletHitEvent e) {
  if (intermediate) {
    if (opponentName == e.getName()) {
      reward += 0.2;
    }
  }
}

public void onBulletMissed(BulletMissedEvent e) {
  if (intermediate) {
    reward -= 0.2;
  }

}

public void onHitByBullet(HitByBulletEvent e) {
  if (intermediate) {
    if (opponentName == e.getName()) {
      reward -= 0.2;
    }
  }
}

public void onHitWall(HitWallEvent e) {
  if (intermediate) {
    reward -= 0.1;
  }
}
```

```
  public void onRobotDeath(RobotDeathEvent e) {
    if (e.getName() == opponentName)
      opponentDistance = 1000;
  }

  // Terminal rewards
  public void onWin(WinEvent event) {
    saveTable();
    reward += goodReward;
    count += 1;
    winCount += 1;
    PrintStream file = null;
    try {
      file = new PrintStream(new
RobocodeFileOutputStream(getDataFile("winning-rates.dat").getAbsolutePath(),
true));
      if (count == 19) {
        winningRates = (double) (winCount) / 20;
        file.println(winningRates);
        reward = 0;
        winCount = 0;
        count = 0;
        if (file.checkError())
          System.out.println("Save Error!");
        file.close();
      }
    } catch (IOException e) {
      System.out.println(e);
    } finally {
      try {
        if (file != null)
          file.close();
      } catch (Exception e) {
        System.out.println(e);
      }
    }
  }

  public void onDeath(DeathEvent event) {
    saveTable();
    reward += badReward;
    count += 1;
    PrintStream file = null;
    try {
```

```
      file = new PrintStream(new
RobocodeFileOutputStream(getDataFile("winning-rates.dat").getAbsolutePath(),
true));
      if (count == 19) {
        winningRates = (double) (winCount) / 20;
        file.println(winningRates);
        reward = 0;
        winCount = 0;
        count = 0;
        if (file.checkError())
          System.out.println("Save Error!");
        file.close();
      }
    } catch (IOException e) {
      System.out.println(e);
    } finally {
      try {
        if (file != null)
          file.close();
      } catch (Exception e) {
        System.out.println(e);
      }
    }
  }

  public void loadTable() {
    try {
      learning.LUTable.load(getDataFile("LUT.dat"));
    } catch (Exception e) {
      out.println("Load Error!" + e);
    }
  }

  public void saveTable() {
    try {
      learning.LUTable.save(getDataFile("LUT.dat"));
    } catch (Exception e) {
      out.println("Save Error!" + e);
    }
  }
}
```