Assignment 2

Yuan Sun
48500904

Question (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.

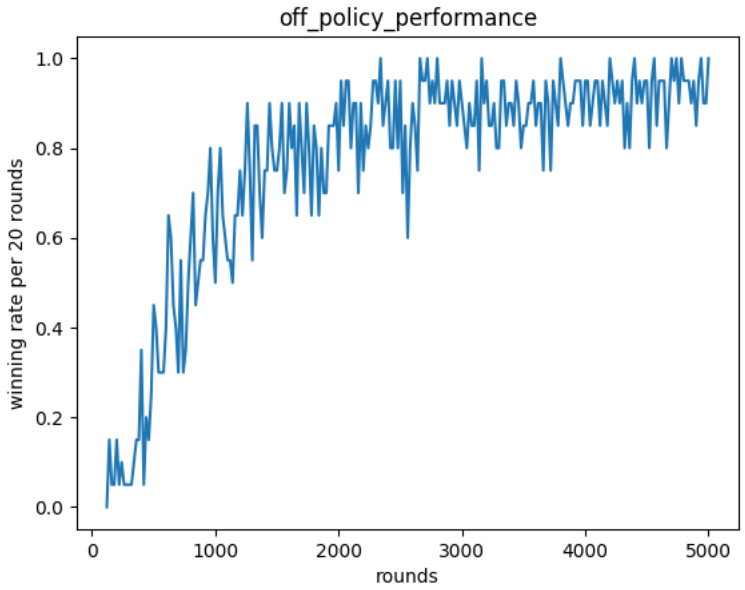

Fig. 1 learning convergence
(learning rate=0.3, discount factor is 0.9, expolre rate is 0.5 for the first 200 rounds)

Comment:

As shown in the graph, the winning rate per 20 rounds increases from 0% to nearly 95%. When reaches 2700 rounds, the performance begins to converge, which means that it becomes stable and cannot be further improved and the learning is completed.

The convergence speed in based on the number of actions and states and learning rate and the robot's opponent. Here I used five actions and 7860 states. I found out that although setting more actions can make the robot perform more actions but took longer to train the look up table. The look up table consists of mostly zeros in the beginning and the convergence is too slow. So I reduced the number of actions and combined the radar and gun actions into one action (in which the robot can first scan the enemy and then turn the gun to it and then fire). So the learning can converge in 2700 rounds.

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

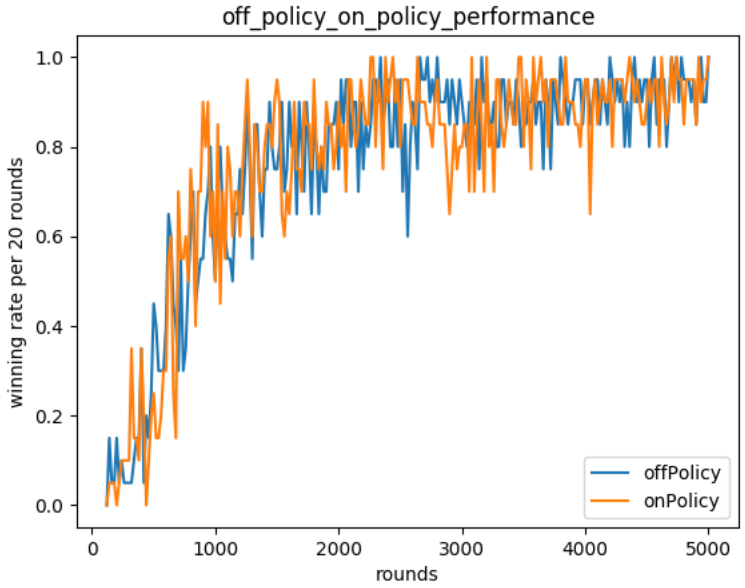

Fig. 2 on and off policy performance
(learning rate=0.3, discount factor is 0.9, expolre rate is 0.5 for the first 200 rounds)

Comment:

For off-policy learning, the winning rate per 20 rounds increases from 0% to 95%.

For on-policy learning, the winning rate per 20 rounds also increases from 0% to 95%.

This shows that off-policy has similar convergence trend as on-policy, The difference between off-policy and on-policy is that off-policy updates the LUT using the greedy move actions, whereas on policy uses the actions actually taken to update the LUT. In my case, both of them can converge and reach relatively similar results.

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

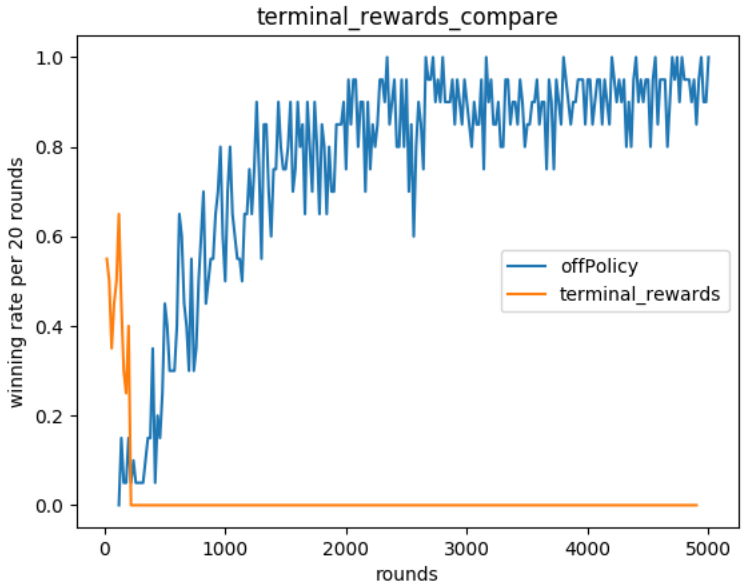

Fig. 3 terminal rewards compared with off-policy
(learning rate=0.3, discount factor is 0.9, expolre rate is 0.5 for the first 200 rounds)

Comment:
For intermediate reward, the winning rate per 20 rounds increases from 0% to 95%.
For terminal reward, the winning rate in the beginning 200 rounds are about 50%, however becomes zero after 200 rounds. The reason is that in the first 200 rounds, the explore rate is 0.5, which means that the robot can still explore new actions and sometimes happen to do some right actions. While after the exploration phase, since there is only reward at the terminal, it cannot update the LUT for each action it takes. Therefore, it doesn't know what actions trigger rewards, and cannot make the correct action.

Question (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.

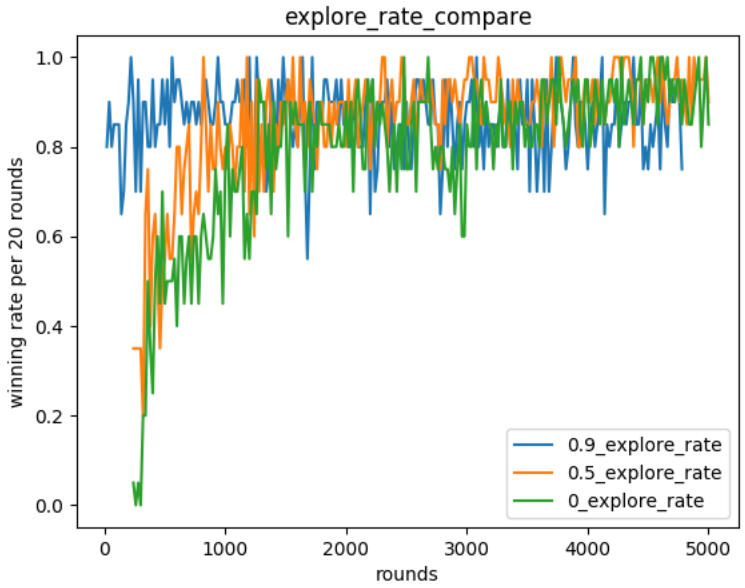

Fig. 4 explore rate compare
(learning rate=0.3, discount factor is 0.9)

Comment:

For exploration rate ε equals 0, which is no exploration at all, the winning rate per 20 rounds increases from converges the slowest. Notice that for zero explore rate, the winning rate is very low at the beginning and around 1000 rounds, it is still below the other two. Since it has no exploratory moves, it can only update the LUT based on the intermediate rewards for each state and action, which is slow.

For different exploration rate ε, the higher the ε, the faster the converge. 0.9 exploratory rate converges faster than 0.5 exploratory rate, since the higher the rate, the faster the robot can learn about different actions and states, it can quickly update its LUT and quickly converge. However, keeping a high explore rate will result in some stupid moves in the end. The blue line has constant explore rate of 0.9, while orange line the has exploratory rate of 0.5 in the first 200 rounds, so the blue line is slightly lower than the orange line from 2000 to 5000 rounds.

APPENDIX:

newRobot.java:

```
package a2;
import java.awt.*;
import java.awt.geom.*;
import java.io.File;
import java.io.IOException;
import java.io.PrintStream;

import robocode.*;
```

```
public class newRobot extends AdvancedRobot
{
	public static final double PI = Math.PI;
	private Target target;
	private LUT table;
	private Learner learner;
	private boolean found=false;
	private double reward = 0.0;
	private double firePower;
	private int direction = 1;
	private int isHitWall = 0;
	private int isHitByBullet = 0;
	private int winFlag = 0;
	private static int countForWin=0;
	double rewardForWin=100;//100
	double rewardForDeath=-20;//-20
	double accumuReward=0.0;
	private static int count=0;

	public void run()
	{

		table = new LUT();
		loadData();
		learner = new Learner(table);
		target = new Target();
		target.distance = 1000;


		setColors(Color.green, Color.white, Color.green);
		setAdjustGunForRobotTurn(true);
		setAdjustRadarForGunTurn(true);
		setAdjustRadarForRobotTurn(true);
		turnRadarRightRadians(2 * PI);
		while (true)
		{
			if(getRoundNum()<200)//
			learner.explorationRate=0.9;//0.5
//			else if(getRoundNum()<getNumRounds()/5)///3
//				learner.explorationRate=0.0;//0.2
			else
				learner.explorationRate=0.9;
			robotMovement();
			execute();

		}
	}


	private void robotMovement()
	{
		int state = getState();
		int action = learner.selectAction(state);

		learner.learn(state, action, reward);
		//accumuReward+=reward;
		reward = 0.0;
```

```
                switch (action)
                {
                case Action.ahead:
                        setAhead(Action.aheadDistance);
                        break;
                case Action.back:
                        setBack(Action.aheadDistance);
                        break;
                case Action.aheadLeft:
                        setAhead(Action.aheadDistance);
                        setTurnLeft(Action.turnDegree);
                        break;
                case Action.aheadRight:
                        setAhead(Action.aheadDistance);
                        setTurnRight(Action.turnDegree);
                        break;
                case Action.fireOne:
                        findTargetFire();
                        if (getGunHeat() == 0) {
                                setFire(1);
                        }
                        break;
                case Action.fireTwo:
                        findTargetFire();
                        if (getGunHeat() == 0) {
                                setFire(2);
                        }
                        break;
                case Action.fireThree:
                        findTargetFire();
                        if (getGunHeat() == 0) {
                                setFire(3);
                        }
                        break;
                }
}
        private void findTargetFire() {
                found=false;
                while(!found) {
                        setTurnRadarLeft(360);
                        execute();
                }
                double gunOffset=(getGunHeading()-getHeading())/360*2*PI-target.bearing;
                setTurnGunLeftRadians(NormaliseBearing(gunOffset));
                execute();
        }


        private int getState()
        {
                int heading = State.getHeading(getHeading());
                int XPosition = State.getXPosition(getX());
                int YPosition = State.getYPosition(getY());
                int targetDistance = State.getTargetDistance(target.distance);
                int targetBearing = State.getTargetBearing(target.bearing);
                int energy = State.getEnergy(getEnergy());
```

```
		int state = State.Mapping[heading][targetDistance][targetBearing][XPosition][YPosition];   //[isHitWall]
		return state;
}


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 change = e.getBullet().getPower() * 9;
		out.println("Bullet Hit: " + change);
		accumuReward += change;
		int state = getState();
		int action = learner.selectAction(state);
		learner.learn(state, action, change);
	}
}


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

	accumuReward += change;
	int state = getState();
	int action = learner.selectAction(state);
	learner.learn(state, action, 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);
		accumuReward += change;
		int state = getState();
		int action = learner.selectAction(state);
		learner.learn(state, action, change);
	}
	isHitByBullet = 1;
}

public void onHitRobot(HitRobotEvent e)
{
	if (target.name == e.getName())
	{
		double change = -6.0;
		out.println("Hit Robot: " + change);
		accumuReward += change;
		int state = getState();
		int action = learner.selectAction(state);
		learner.learn(state, action, change);
	}
}

public void onHitWall(HitWallEvent e)
{
```

```
		double change = -(Math.abs(getVelocity()) * 0.5 );
		out.println("Hit Wall: " + change);
		accumuReward += change;
		isHitWall = 1;
		int state = getState();
		int action = learner.selectAction(state);
		learner.learn(state, action, change);
	}

	public void onScannedRobot(ScannedRobotEvent e)
	{
		if ((e.getDistance() < target.distance)||(target.name == e.getName()))
		{
			found=true;
			//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 = 1000;
		}

	}

	public void onWin(WinEvent event)
	{
		winFlag=1;
		File file = getDataFile("accumReward1.dat");
		accumuReward+=rewardForWin;
		int state = getState();
		int action = learner.selectAction(state);
		learner.learn(state, action, rewardForWin);
		saveData();
		countForWin++;
		count++;
		PrintStream w = null;
		try
		{
```

```
                        w = new PrintStream(new RobocodeFileOutputStream(file.getAbsolutePath(), true));
                        if(count==20){
                                count=0;
                                w.println(Math.abs(accumuReward)+" "+countForWin*5+" "+"
"+learner.explorationRate);
                                accumuReward=0;
                                countForWin=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);
                        }

                }
                saveResult2(winFlag);
        }
        public void saveResult(BattleEndedEvent e) {
                if(getRoundNum()%50==0) {
                        File file = getDataFile("saveResult1.dat");
                        int a=e.getResults().getFirsts();
                        PrintStream w = null;
                        try
                        {
                                w = new PrintStream(new RobocodeFileOutputStream(file.getAbsolutePath(), true));
                                w.println(a);
                                if (w.checkError())
                                        System.out.println("Could not save the data!");  //setTurnLeft(180 -
(target.bearing + 90 - 30));
                                w.close();
                        }

                        catch (IOException e1)
                        {
                                System.out.println("IOException trying to write: " + e1);
                        }
                        finally
                        {
                                try
                                {
                                        if (w != null)
                                                w.close();
```

```
                                        }
                                        catch (Exception e2)
                                        {
                                                System.out.println("Exception trying to close witer: " + e2);
                                        }
                                }
                        }
        }
        public void saveResult2(int winFlag) {
                if(true) {
                        File file = getDataFile("saveResult1.dat");
                        PrintStream w = null;
                        try
                        {
                                w = new PrintStream(new RobocodeFileOutputStream(file.getAbsolutePath(), true));
                                w.println(winFlag);
                                if (w.checkError())
                                        System.out.println("Could not save the data!");  //setTurnLeft(180 -
(target.bearing + 90 - 30));
                                w.close();
                        }

                        catch (IOException e1)
                        {
                                System.out.println("IOException trying to write: " + e1);
                        }
                        finally
                        {
                                try
                                {
                                        if (w != null)
                                                w.close();
                                }
                                catch (Exception e2)
                                {
                                        System.out.println("Exception trying to close witer: " + e2);
                                }
                        }
                }
        }
        public void onDeath(DeathEvent event)
        {
                winFlag=0;
                accumuReward+=rewardForDeath;
                count++;
                int state = getState();
                int action = learner.selectAction(state);
                learner.learn(state, action, rewardForDeath);

                saveData();
                File file = getDataFile("accumReward1.dat");
                PrintStream w = null;
                try
                {

                        w = new PrintStream(new RobocodeFileOutputStream(file.getAbsolutePath(), true));
                        if(count==20){
                                count=0;
```

```
                                        w.println(Math.abs(accumuReward)+" "+countForWin*5+" "+"
"+learner.explorationRate);
                                        accumuReward=0;
                                        countForWin=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);
                                }

                        }
                        saveResult2(winFlag);
                }

                public void loadData()
                {
                        try
                        {
                                table.load(getDataFile("movement1.dat"));
                        }
                        catch (Exception e)
                        {
                        }
                }

                public void saveData()
                {
                        try
                        {
                                table.save(getDataFile("movement1.dat"));
                        }
                        catch (Exception e)
                        {
                                out.println("Exception trying to write: " + e);
                        }
                }
}
```

state.java:

```
package a2;

public class State
{
```

```
public static final int numHeading = 4;
public static final int numTargetDistance = 10;
public static final int numTargetBearing = 4;
public static final int XPosition = 8;
public static final int YPosition = 6;
//public static final int numEnergy =3 ;
public static final int numStates;
public static final int Mapping[][][][][];

static
{
 Mapping = new int[numHeading][numTargetDistance][numTargetBearing][XPosition][YPosition];//[numHitWall]
 int count = 0;
 for (int a = 0; a < numHeading; a++)
  for (int b = 0; b < numTargetDistance; b++)
   for (int c = 0; c < numTargetBearing; c++)
    for (int d = 0; d < XPosition; d++)
     for (int e = 0; e < YPosition; e++)
         //for (int f = 0; f < numEnergy; f++)
     Mapping[a][b][c][d][e] = count++;

 numStates = count;
}

public static int getHeading(double heading)
{
          double unit=(360/numHeading);

          return (int)((heading-1)/unit);
          /*
 double angle = 360 / numHeading;
 double newHeading = heading + angle / 2;
 if (newHeading > 360.0)
  newHeading -= 360.0;
 return (int)(newHeading / angle);  */
}

public static int getTargetDistance(double value)
{
 int distance = (int)(value / 100.0); //30
 if (distance > numTargetDistance - 1)
  distance = numTargetDistance - 1;
 return distance;
}
public static int getDistanceToWall(double x,double y)
{
          double dis=Math.min(Math.min(x, y),Math.min(800-x, 600-y));
 if (dis > 10.0)
  return 1;
 else
          return 0;
}

public static int getTargetBearing(double bearing)
{  double unit=(360/numTargetBearing);

          return (int)((bearing+180-1)/unit);
           /*
```

```
   double PIx2 = Math.PI * 2;
   if (bearing < 0)
     bearing = PIx2 + bearing;
   double angle = PIx2 / numTargetBearing;
   double newBearing = bearing + angle / 2;
   if (newBearing > PIx2)
     newBearing -= PIx2;
   return (int)(newBearing / angle);  */
  }
  public static int getEnergy(double energy)
  {
   if(energy>80)
            return 2;
   else if(energy>30)
            return 1;
   else
   return 0;
  }
  public static int getXPosition(double XPosition)
  {
   return (int)((XPosition-1)/100);
  }
  public static int getYPosition(double YPosition)
  {
   return (int)((YPosition-1)/100);
  }
}
```

Action.java:

```
package a2;

public class Action {
            public static final int ahead=0;
            public static final int back = 1;
            public static final int aheadLeft = 2;
            public static final int aheadRight = 3;
            public static final int fireOne = 4;//radarRight
            public static final int fireTwo = 5;
            public static final int fireThree = 6;

            public static final int numActions = 7;

            public static final double aheadDistance = 150.0;
            public static final double backDistance = 150.0;//100.0

            public static final double turnDegree =  15.0;//
            public static final double turnGunDegree =  15.0;//
            public static final double turnRadarDegree =  15.0;
}
```

LUT.java:

```
package a2;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;
import robocode.*;
import robocode.Robot;
```

```
public class LUT implements LUTInterface{
	public double[][] table;

	public LUT(){
		table = new double[State.numStates][Action.numActions];
		initialiseLUT(); // set all function values to be zero
	}
	public double max_Q(int state) //return the maximum value in the LUT
	 {
	   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;
	 }
	@Override
	public double outputFor(double[] X) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public double train(double[] X, double argValue, String type) {
		// TODO Auto-generated method stub
		return 0;
	}
	public int getBestAction(int state){
	   double maxinum = Double.NEGATIVE_INFINITY;
	   int bestAction = 0;
	   for (int i = 0; i < table[state].length; i++)
	   {
	     if (table[state][i] > maxinum){
	       maxinum = table[state][i];
	       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 load(File file){
	   BufferedReader read = null;
	   try{
	     read = new BufferedReader(new FileReader(file));
	     for (int i = 0; i < State.numStates; i++)
	       for (int j = 0; j < Action.numActions; j++) {
	         table[i][j] = Double.parseDouble(read.readLine());
	       }
	   }
	   catch (IOException e){
	     System.out.println("initialiseLUT. IOException trying to open reader: " + e);
	     initialiseLUT();
```

```
    }
    catch (NumberFormatException e){
      initialiseLUT();
    }
    finally{
            try{
                    if (read != null)
                            read.close();
            }
            catch (IOException e)
            {
                    //   System.out.println("IOException trying to close reader: " + e);
            }
    }
  }

  @Override
  public void save(File file) {
            int i,j=0,count=0;
            PrintStream write = null;
            try{
            write = new PrintStream(new RobocodeFileOutputStream(file));
            count++;
            System.out.println("count is: "+count);
            for ( i = 0; i < State.numStates; i++)
                    for (j = 0; j < Action.numActions; j++) {
                            //System.out.println("i:"+i+"j"+j);
                            write.println(new Double(table[i][j])); //write.println(new Double(table[i][j]));

                    }
            System.out.println("i is"+i+"j is "+j);
            if (write.checkError())
                    System.out.println("Could not save the data!");
            write.close();
            }
            catch (IOException e){
                    //   System.out.println("IOException trying to write: " + e);
            }
            finally{
                    try{
                            if (write != null)
                                    write.close();
                    }
                    catch (Exception e){
                            //     System.out.println("Exception trying to close witer: " + e);
                    }
            }
  }

  @Override
  public void initialiseLUT(){
            for (int i = 0; i < State.numStates; i++)
                    for (int j = 0; j < Action.numActions; j++)
                            table[i][j] = 0.0;

  }

  @Override
```

```
	public int indexFor(double[] X) {
		// TODO Auto-generated method stub
		return 2;
	}

}
```

Target.java:

```
package a2;


import java.awt.geom.*;

class Target
{
 String name;
 public double bearing;
 public double head;
 public long ctime;
 public double speed;
 public double x, y;
 public double distance;
 public double changehead;
 public double energy;

 public Point2D.Double guessPosition(long when)
 {
  double diff = when - ctime;
  double newY, newX;

  /**if the change in heading is significant, use circular targeting**/
  if (Math.abs(changehead) > 0.00001)
  {
   double radius = speed/changehead;
   double tothead = diff * changehead;
   newY = y + (Math.sin(head + tothead) * radius) - (Math.sin(head) * radius);
   newX = x + (Math.cos(head) * radius) - (Math.cos(head + tothead) * radius);
  }
  /**If the change in heading is insignificant, use linear**/
  else {
   newY = y + Math.cos(head) * speed * diff;
   newX = x + Math.sin(head) * speed * diff;
  }
  return new Point2D.Double(newX, newY);
 }

 public double guessX(long when)
 {
  long diff = when - ctime;
  System.out.println(diff);
  return x+Math.sin(head)*speed*diff;
 }
 public double guessY(long when)
 {
  long diff = when - ctime;
  return y+Math.cos(head)*speed*diff;
 }
```

```
}
```

Learner.java:

```
package a2;


import java.util.Random;


public class Learner
{
 public static final double LearningRate =0.3;  // 0.1
 public static final double DiscountRate =0.9 ;   //0.9
 public static double explorationRate = 0.0;  //0.5
 private int lastState;
 private int lastAction;
 private boolean first = true;
 private LUT table;

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

 public void learn(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.max_Q(state));
    table.setQValue(lastState, lastAction, newQValue);
    System.out.println(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 thres = Math.random();
```

```
		int actionIndex = 0;

		if (thres<explorationRate)
		{//randomly select one action //from action(0,1,2)
			//Random ran = new Random();
			actionIndex = (int)(Math.random()*(Action.numActions-1 - 0 + 1));
		}
		else
		{//e-greedy
			actionIndex=table.getBestAction(state);
		}
		return actionIndex;
	}

}
```