ECE 592 COURSEWORK PART 2

NAME : ZheZhong Jiang (Harvey)

Student id : 46311130

*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.*

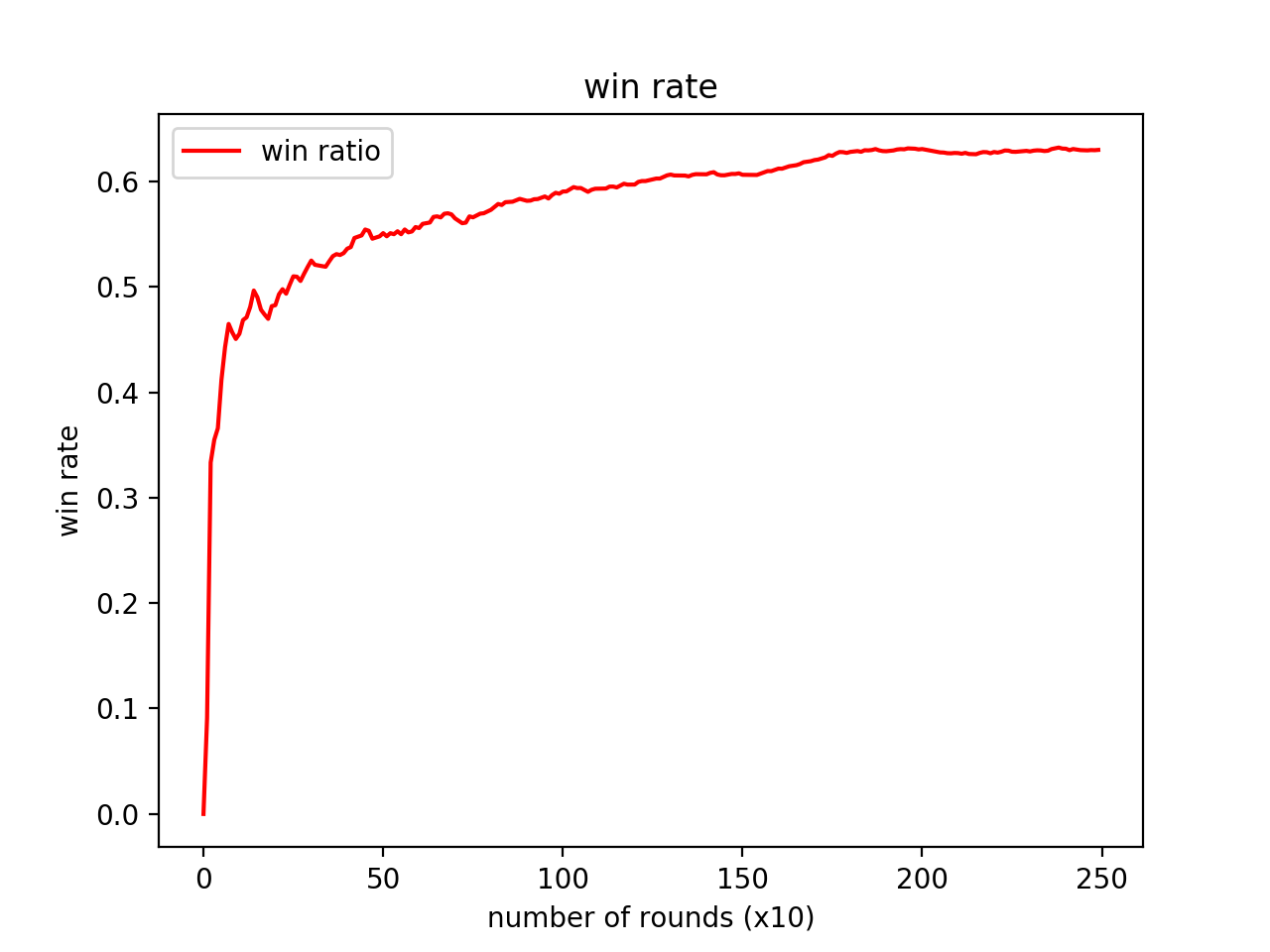


Figure 1

Basic Set up :

Enemy robot : tracker

Number of Rounds : 2500 runs

Off policy : True

Epsilon : 0.1

From the graph , it is observed that the robot is able to learn fairly fast within the first 500 rounds , as shown by a jump of winning rate to nearly 50 percent. After 1500 rounds of training , the winning rate slowly converges to 60 %.

To confirm that my robot is indeed learning, I train the robot with epsilon = 1.0 for 3500 runs , and then using the updated Q table ,turn epsilon to 0 and observed a increase of winning rate right away

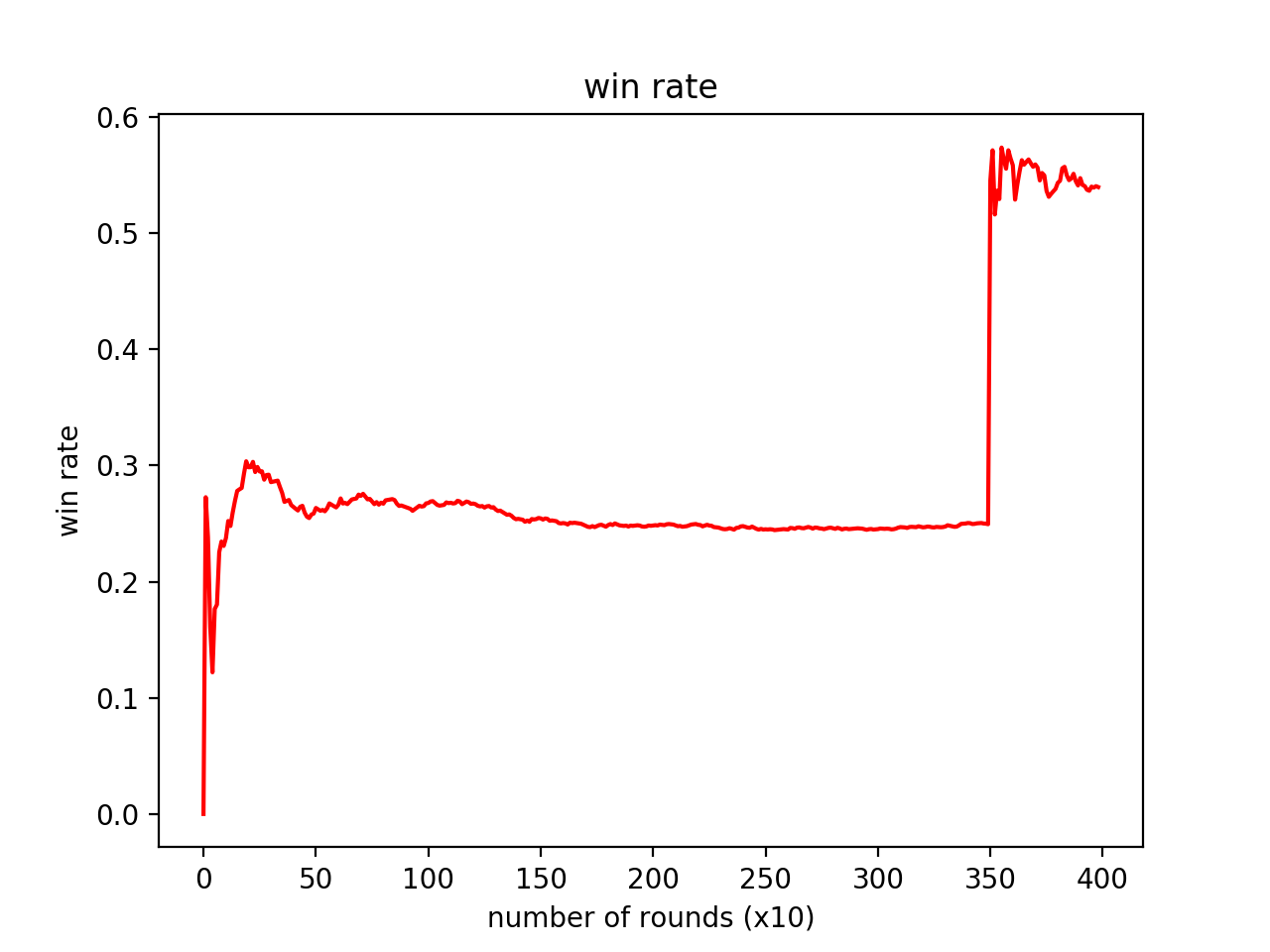


Figure 2

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

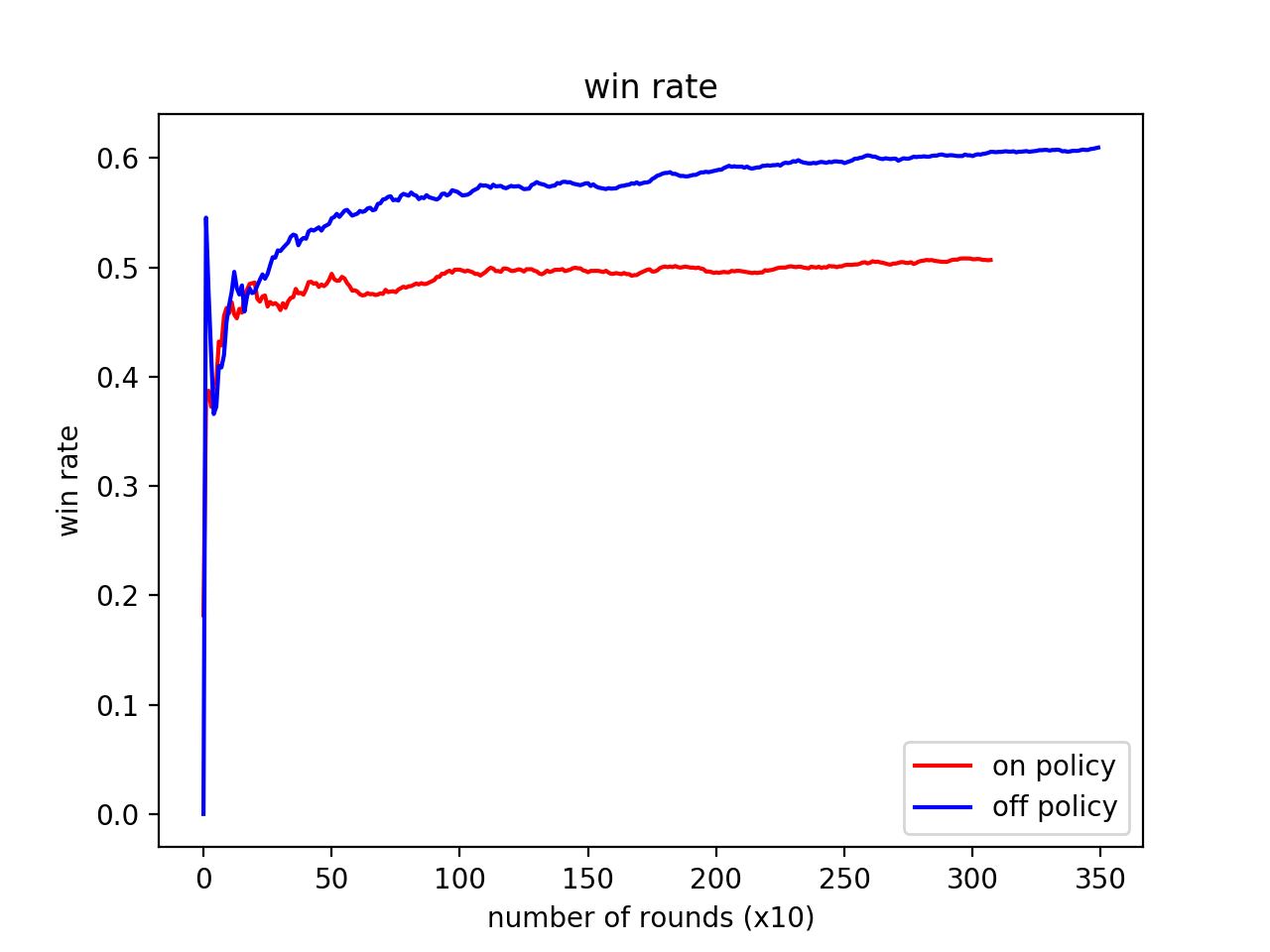


Figure 3.

This is plot of my robot versus tracker in 3500 rounds . Red line is the winning rate of on policy learning and the blue line is the off policy learning. In both cases , the epsilon is set to 0.1 so that both training sessions are in the exactly same set up except for on/off policy. As shown on the graph , off policy learning performs better than on policy learning with a 10 percent higher winning rate on average.

Off policy seems to perform better because every time an update is made to the Q table , it evaluates the current policy but following another with larger Q value. This means that the program can learn the optimal policy regardless of the current behaviour policy(It learns no matter what the robot does). As a result off policy is more “flexible” when facing a learning problem. On the other hand , on policy learns the value from the actions robot take, It is safer in a way, but may get stuck in local minima.

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

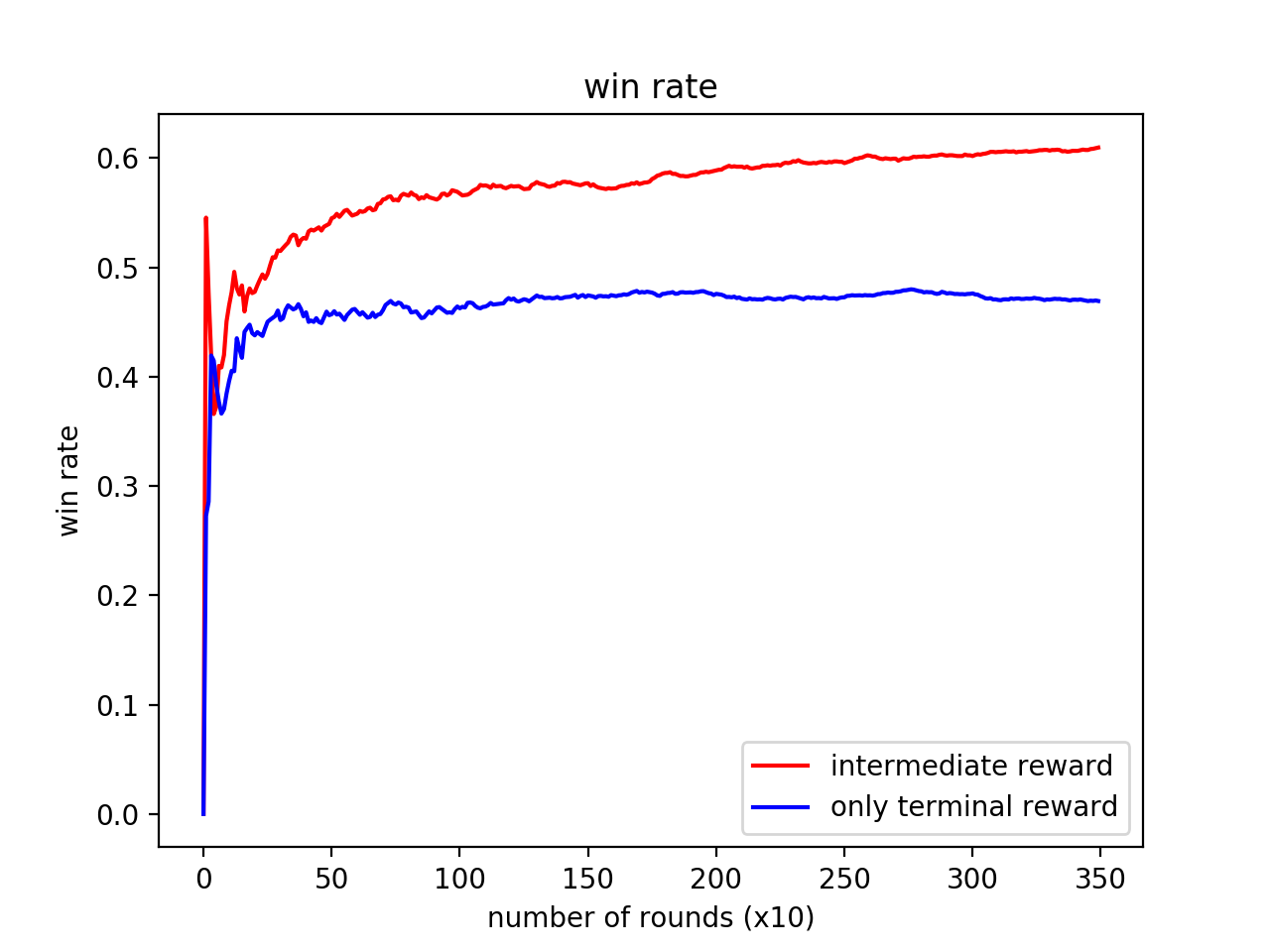


Figure 4

This is the plot of my robot versus tracker , with only terminal reward applied to learning , the winning rate is stuck between 40 and 50 %. One of the possible reason why including intermediate reward is better is that the Q table will be updated as soon as an reward is observed in a round (such as hitting a wall , getting hit by bullet) and utilize that information to evaluate the performance of current action. With only terminal reward , the robot is missing the information happened in the battle and not able to figure out a “link” between actions taken and final result.

*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*

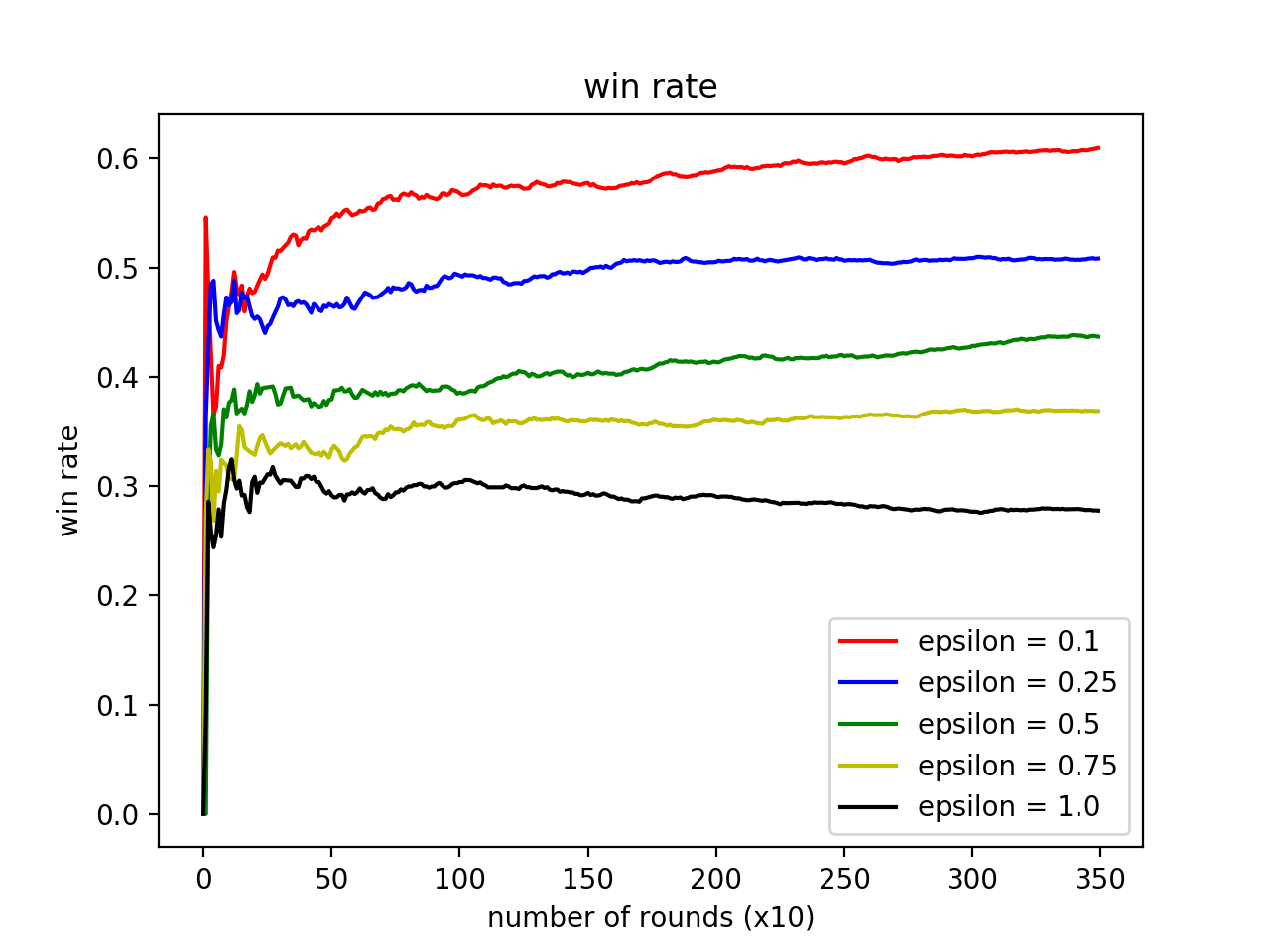


Figure 5

As shown above , smaller epsilon always perform better than larger epsilon, with an epsilon = 1.0, the robot is taking purely random actions all the time. With smaller epsilon , the robot is still taking random actions to explore other possibilities but most of the time it sticks to greedy move with largest Q value.

Below my code :

|  |
| --- |
| package HarveyJ; |
|  |  |
|  | import java.awt.geom.Point2D; |
|  | import java.io.BufferedReader; |
|  | import java.io.File; |
|  | import java.io.FileReader; |
|  | import java.io.IOException; |
|  | import java.io.PrintStream; |
|  | import java.util.Arrays; |
|  | import java.util.Random; |
|  |  |
|  | import robocode.AdvancedRobot; |
|  | import robocode.BulletHitEvent; |
|  | import robocode.DeathEvent; |
|  | import robocode.HitByBulletEvent; |
|  | import robocode.HitRobotEvent; |
|  | import robocode.RobocodeFileOutputStream; |
|  | import robocode.RoundEndedEvent; |
|  | import robocode.ScannedRobotEvent; |
|  | import robocode.WinEvent; |
|  |  |
|  | public class Q\_learning\_LUT extends AdvancedRobot { |
|  | public double PI = Math.PI; |
|  | // state representations will be a 4 tuple{x\_a,y\_a,distance\_to\_enemy,Bearing} |
|  | // below is the quantized states |
|  | public static int X\_coor = 8; |
|  | public static int Y\_coor = 6; |
|  | public static int Distance = 10; |
|  | public static int Bearing = 4; |
|  | double getBearing ; |
|  |  |
|  | public int current\_state ; |
|  | public int next\_state ; |
|  | double current\_q = 0.0; |
|  | double next\_q = 0.0; |
|  | double reward = 0.0; |
|  | double total\_reward\_per\_action = 0.0; |
|  | double cum\_reward; |
|  | double [] reward\_array = new double [4500]; // record rewards for multiple battles |
|  | public static double [] win\_rate = new double [1000]; |
|  | public static int win\_count = 0; |
|  | public static int index\_win = 0; |
|  | // hyper paramaters |
|  | double alpha = 0.15; // learning rate |
|  | double gamma = 0.9; // discount factor |
|  | double epsilon = 0.0; |
|  |  |
|  | static int row\_num = 8\*6\*10\*4; |
|  | static int col\_num = 5; |
|  | boolean initialize = false; |
|  | boolean learning = true ; |
|  | boolean offpolicy = true; |
|  | public static double [][] Q\_table = new double [row\_num][col\_num]; // This Q\_table is a String matrix, use this to save Q\_table on disk |
|  | double [][] Q\_table\_double = new double[row\_num][col\_num]; // This Q\_table is a double matrix , use this to perform numeric operations |
|  | public static int index1 = 0; |
|  | private int previousState; |
|  | private int previousAction; |
|  | public static Enemy enemy ; |
|  |  |
|  | public class Enemy{ |
|  | public int y; |
|  | public int x; |
|  | public int distance; |
|  | public int bearing; |
|  | } |
|  | public void initialize\_Q\_table(){ |
|  | for (int i=0; i<row\_num; i++) |
|  | for (int j=0; j<col\_num; j++) |
|  | Q\_table[i][j]=0.0; |
|  | } |
|  |  |
|  |  |
|  | public void saveTable(File file){ |
|  | PrintStream w = null; |
|  | try { |
|  | w = new PrintStream(new RobocodeFileOutputStream(file)); |
|  | for (int i = 0; i < Q\_table.length; i++) |
|  | for (int j = 0; j < col\_num; j++) |
|  | w.println(new Double(Q\_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); |
|  | } |
|  | } |
|  | }// working |
|  | public void saveData(File file){ |
|  | PrintStream w = null; |
|  | try { |
|  | w = new PrintStream(new RobocodeFileOutputStream(file)); |
|  | for (int i = 0; i < win\_rate.length; i++) |
|  | w.println(new Double(win\_rate[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 onRoundEnded(RoundEndedEvent e\_1) { |
|  | System.out.println("win rate is !!!" + win\_count/((double)getRoundNum()+1.0)); |
|  | if(getRoundNum() % 10 ==0) { |
|  | win\_rate[index\_win] = win\_count/((double)getRoundNum()+1); |
|  | System.out.println("win rate array is" + Arrays.toString(win\_rate)); |
|  | index\_win +=1; |
|  | saveData(getDataFile("win\_rate.txt")); |
|  | } |
|  | index1=index1+1; |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  | public void loadData(File file){ |
|  | BufferedReader r = null; |
|  | try{ |
|  | r = new BufferedReader(new FileReader(file)); |
|  | for (int i = 0; i < row\_num; i++) |
|  | for (int j = 0; j < col\_num; j++) |
|  | Q\_table[i][j] = Double.parseDouble(r.readLine()); |
|  | } |
|  | catch (IOException e) { |
|  | System.out.println("IOException trying to open reader: " + e); |
|  | initialize\_Q\_table(); |
|  | saveTable(getDataFile("LUT.txt")); |
|  | } |
|  | catch (NumberFormatException e) { |
|  | initialize\_Q\_table(); |
|  | saveTable(getDataFile("LUT.txt")); |
|  | } |
|  | catch (NullPointerException e) { |
|  | initialize\_Q\_table(); |
|  | saveTable(getDataFile("LUT.txt")); |
|  | } |
|  | finally { |
|  | try { |
|  | if (r != null) |
|  | r.close(); |
|  | } |
|  | catch (IOException e) { |
|  | System.out.println("IOException trying to close reader: " + e); |
|  | } |
|  | } |
|  | } // working oad function works |
|  |  |
|  | public void run() { |
|  | enemy= new Enemy(); |
|  | loadData(getDataFile("LUT.txt")); |
|  |  |
|  | turnGunRight(360); // initial scan |
|  | while(true) { |
|  | if (learning == false) { |
|  | int action = randInt(0,4); |
|  | take\_action(action); |
|  | turnGunRight(360); |
|  | } |
|  | else { |
|  |  |
|  | //Q\_learning starts |
|  |  |
|  | //step 1 get initial state ----works |
|  | current\_state = getState(); |
|  | System.out.println("current\_state "+ current\_state); |
|  | //step 2 find the action that would result in maximum Q\_value ---- works |
|  | int action = choose\_action(current\_state); |
|  |  |
|  | if (offpolicy == true) { |
|  | next\_q = max((Q\_table[current\_state])); |
|  | } |
|  | else { |
|  | System.out.println("on policy"); |
|  | next\_q = Q\_table[current\_state][action]; |
|  | } |
|  | Q\_table[previousState][previousAction] += alpha\*(total\_reward\_per\_action+gamma\*next\_q-Q\_table[previousState][previousAction]); |
|  | previousState = current\_state; |
|  | previousAction = action; |
|  | total\_reward\_per\_action = 0.0; |
|  | take\_action(action); |
|  | // step 5 , after taking action , register the new state ---- works |
|  | turnGunRight(360); |
|  | execute(); |
|  | } |
|  |  |
|  | } |
|  | } |
|  |  |
|  | public void onScannedRobot(ScannedRobotEvent e) { |
|  |  |
|  | enemy.x = quantize\_position(getX()); |
|  | enemy.y = quantize\_position(getY()); |
|  | enemy.distance = quantize\_distance(e.getDistance()); |
|  | enemy.bearing = quantize\_bearing(e.getBearing()); |
|  | fire(3); |
|  |  |
|  | System.out.println("x coor is " + enemy.x); |
|  |  |
|  | } |
|  | public void onWin(WinEvent event) { |
|  | System.out.println("winning !!!"); |
|  | win\_count += 1; |
|  | System.out.println("win count is " + win\_count); |
|  | saveTable(getDataFile("LUT.txt")); |
|  |  |
|  |  |
|  |  |
|  | } |
|  | public void onDeath(DeathEvent event){ |
|  | saveTable(getDataFile("LUT.txt")); |
|  | } |
|  |  |
|  | // get the current state |
|  | public int getState(){ |
|  |  |
|  | return index(enemy.x,enemy.y,enemy.distance,enemy.bearing); |
|  | } |
|  |  |
|  | // get the index of state |
|  | static public int index( int x, int y, int distance , int bearing ){ |
|  | int index = 0; |
|  | index = x\*Bearing\*Distance\*Y\_coor+ |
|  | y\*Distance\*Bearing+distance\*Bearing+bearing; |
|  | return index; |
|  | } // working |
|  |  |
|  |  |
|  | /\* used in exlpore mode to randomly pick actions \*/ |
|  | public int randInt(int min, int max) { |
|  |  |
|  |  |
|  | Random rand = new Random(); |
|  |  |
|  | // nextInt is normally exclusive of the top value, |
|  | // so add 1 to make it inclusive |
|  | int randomNum = rand.nextInt((max - min) + 1) + min; |
|  |  |
|  | return randomNum; |
|  | } |
|  |  |
|  | public static int state\_index(String state,String[][] Q\_table){ |
|  | int index = 0; |
|  | for (int i=0;i<Q\_table.length;i++) { |
|  |  |
|  | if(Q\_table[i][0].equals(state)) { |
|  | index = i; |
|  |  |
|  | } |
|  | } |
|  | return index; |
|  | } |
|  | public int choose\_action(int state){ |
|  | if(Math.random()>epsilon) { |
|  | return argmax(state); |
|  | } |
|  | else { |
|  | System.out.println("I am taking random action"); |
|  | return randInt(0,4); |
|  | } |
|  |  |
|  | } |
|  | public double max(double[] array) { |
|  | double largest = -99999999999.0; |
|  | // starts with index 1 because the first element is the state number |
|  | for ( int i = 0; i < array.length; i++ ) |
|  | { |
|  | if ( array[i] > largest ) |
|  | { |
|  | largest = array[i]; |
|  |  |
|  | } |
|  | } |
|  | return largest; |
|  | } |
|  | public int argmax(int state) { |
|  | int index = 0; // if the row is all zeros then just take the first action |
|  | double largest = -99999999; |
|  | // starts with index 1 because the first element is the state number ,but we are only |
|  | // interested in the actions |
|  | if (all\_zero(Q\_table[state])){ |
|  | System.out.println("I am taking random action"); |
|  | index = randInt(0,4); |
|  | } |
|  | else { |
|  | System.out.println("I am taking greedy action"); |
|  | for ( int i = 1; i < Q\_table[state].length; i++ ) |
|  | { |
|  | if ( Q\_table[state][i] > largest ) |
|  | { |
|  | largest = Q\_table[state][i]; |
|  | index = i; |
|  | } |
|  | } |
|  |  |
|  | } |
|  | return index ; |
|  |  |
|  | } |
|  | public boolean all\_zero(double array[]) { |
|  | boolean all0 = true; |
|  | for (int i = 0 ; i< array.length;i++) { |
|  | if(array[i]!=0) { |
|  | all0 = false; |
|  | } |
|  | } |
|  | return all0; |
|  | } |
|  | public void take\_action(int action\_index) { |
|  | if(action\_index ==0) { |
|  | ahead(100); |
|  | //setAhead(200); |
|  |  |
|  | } |
|  | else if (action\_index==1) { |
|  | back(100); |
|  | //setBack(200); |
|  |  |
|  | } |
|  | else if(action\_index==2) { |
|  |  |
|  | //setTurnLeft(90); |
|  | //setAhead(200); |
|  | turnLeft(90); |
|  | ahead(100); |
|  |  |
|  |  |
|  | } |
|  | else if(action\_index==3) { |
|  | //setTurnRight(90); |
|  | //setAhead(100); |
|  | turnRight(90); |
|  | ahead(100); |
|  |  |
|  | } |
|  | else if(action\_index == 4) { |
|  | //setTurnLeft(180); |
|  | //setAhead(100); |
|  | turnLeft(180); |
|  | ahead(100); |
|  | } |
|  | } // take action works |
|  | public int quantize\_position(double x\_coor) { |
|  | int quantized = 0; |
|  | //System.out.println("The argument I receive is "+ x\_coor); |
|  | if((x\_coor>=0)&&(x\_coor<100)) { |
|  | quantized = 0; |
|  | } |
|  | else if((x\_coor>=100)&&(x\_coor<200)) { |
|  | quantized = 1 ; |
|  | } |
|  | else if((x\_coor>=200)&&(x\_coor<300)) { |
|  | quantized = 2 ; |
|  | } |
|  | else if((x\_coor>=300)&&(x\_coor<400)) { |
|  | quantized = 3 ; |
|  | } |
|  | else if((x\_coor>=400)&&(x\_coor<500)) { |
|  | quantized = 4 ; |
|  | } |
|  | else if((x\_coor>=500)&&(x\_coor<600)) { |
|  | quantized = 5 ; |
|  | } |
|  | else if((x\_coor>=600)&&(x\_coor<700)) { |
|  | quantized = 6 ; |
|  | } |
|  | else if((x\_coor>=700)&&(x\_coor<800)) { |
|  | quantized = 7 ; |
|  | } |
|  |  |
|  | return quantized; |
|  | } // quantized coordinates works |
|  |  |
|  | public int quantize\_bearing(double bearing\_angle) { |
|  | int bearing = 0; |
|  | if ((bearing\_angle>=0)&&(bearing\_angle<90)){ |
|  | bearing= 0; |
|  |  |
|  | } |
|  | else if ((bearing\_angle>=90)&&(bearing\_angle<=180)){ |
|  | bearing= 1; |
|  |  |
|  | } |
|  | else if ((bearing\_angle<0)&&(bearing\_angle>=-90)){ |
|  | bearing= 2; |
|  |  |
|  | } |
|  | else if ((bearing\_angle<-90)&&(bearing\_angle>=-180)){ |
|  | bearing= 3; |
|  |  |
|  | } |
|  | return bearing; |
|  |  |
|  | }// quantized bearing works |
|  | public int quantize\_distance(double distance) { |
|  | int dist = 0; |
|  | if((distance>=0)&&(distance<100)) { |
|  | dist = 0; |
|  | } |
|  | else if((distance>=100)&&(distance<200)) { |
|  | dist = 1 ; |
|  | } |
|  | else if((distance>=200)&&(distance<300)) { |
|  | dist = 2 ; |
|  | } |
|  | else if((distance>=300)&&(distance<400)) { |
|  | dist = 3 ; |
|  | } |
|  | else if((distance>=400)&&(distance<500)) { |
|  | dist = 4 ; |
|  | } |
|  | else if((distance>=500)&&(distance<600)) { |
|  | dist = 5 ; |
|  | } |
|  | else if((distance>=600)&&(distance<700)) { |
|  | dist = 6 ; |
|  | } |
|  | else if((distance>=700)&&(distance<800)) { |
|  | dist = 7 ; |
|  | } |
|  | else if((distance>=800)&&(distance<900)) { |
|  | dist = 8 ; |
|  | } |
|  | else if((distance>=900)&&(distance<1000)) { |
|  | dist = 9 ; |
|  | } |
|  | return dist; |
|  | } |
|  | // reward functions |
|  | public void onHitRobot(HitRobotEvent event){ |
|  | double reward\_1 =-2; |
|  | total\_reward\_per\_action += reward\_1; |
|  | //System.out.println("HITT robot"); |
|  | } |
|  | public void onBulletHit(BulletHitEvent event){ |
|  | double reward\_2=3; |
|  | total\_reward\_per\_action += reward\_2; |
|  | //System.out.println("HITTING BULLET"); |
|  | } |
|  | public void onHitByBullet(HitByBulletEvent event){ |
|  | double reward\_3=-3; |
|  | total\_reward\_per\_action += reward\_3; |
|  | //System.out.println(" GETTING HIT BY BULLET "); |
|  | } |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | } |