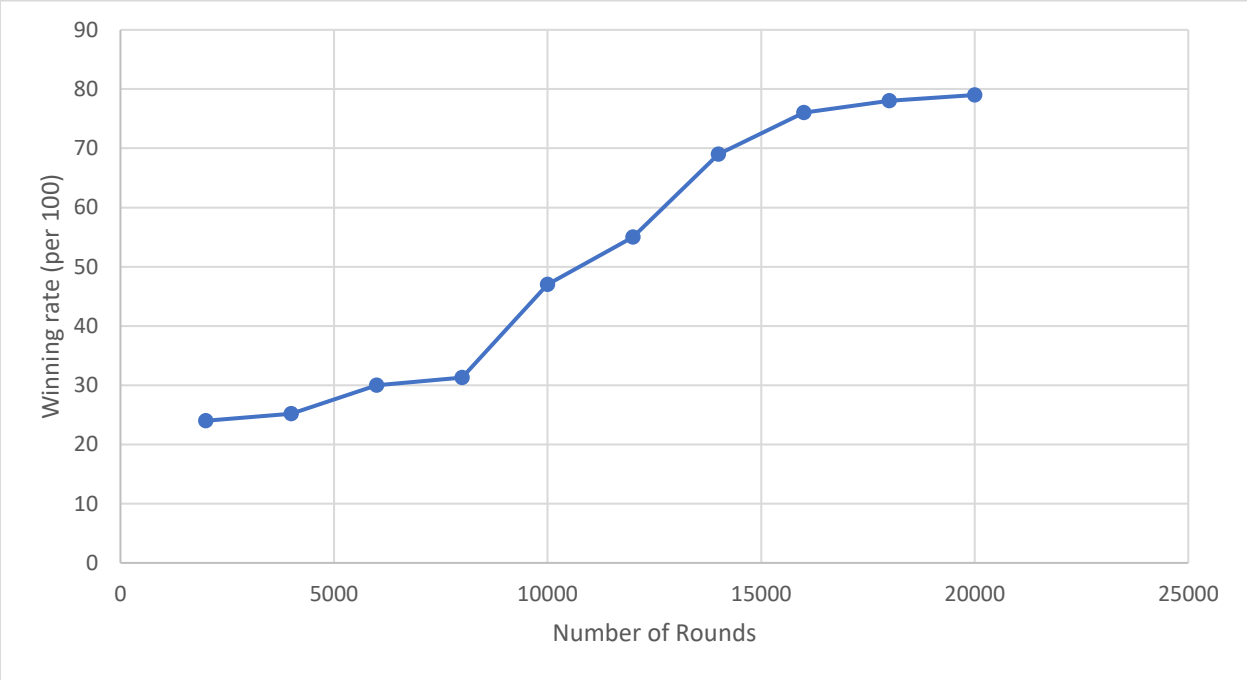
CPEN-502 101

In the Reinforcement learning, the learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. A learning agent must be able to sense the state of its environment to some extent and must be able to take actions that affect the state [1].

The Robocode is a programming game. where the goal is to code a robot battle tank to compete against other robots in a battle arena [2]. As a part of the coursework for assignment for implementation of Reinforcement Learning, I have implemented FnlBot.

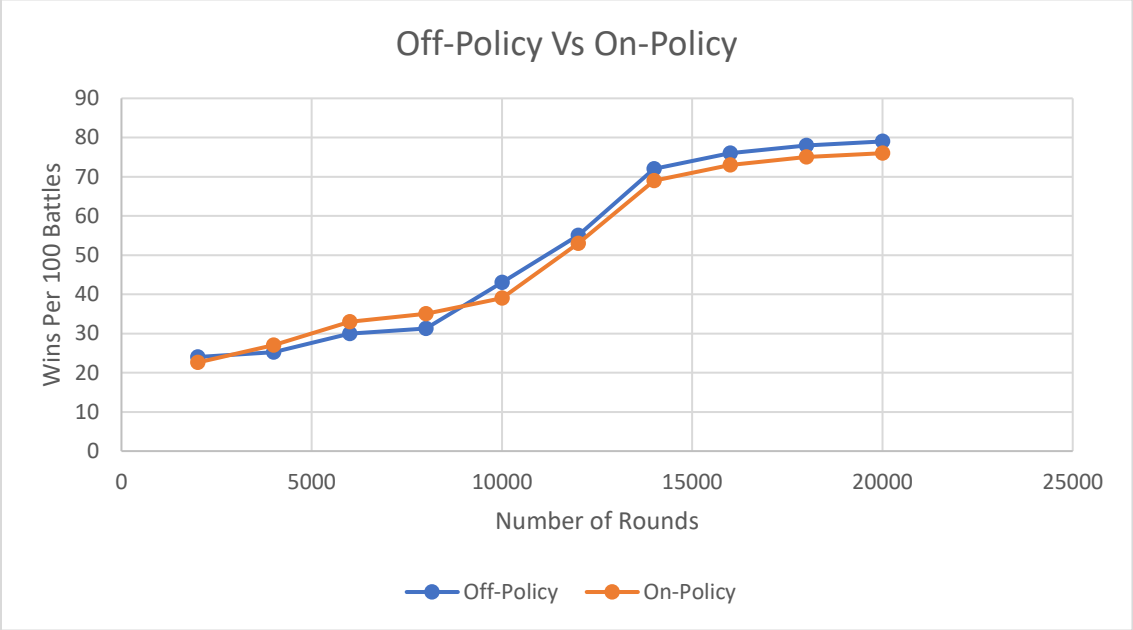
1a) Draw a graph of a parameter that reflects a measure of progress of learning and comment on the convergence of learning of your robot.



Graph(a) Number of Battles Vs Winning rate per 100

The robot was trained for about 20000 battles against the sample robot TrackFire and after 20,000 rounds, the convergence reached approximately 80%. After approximately 9000 rounds, the winning rate started increasing and kept on getting better as the number of battles increased.

2b) Using your robot, show a graph comparing the performance of your robot using on-policy learning vs off-policy learning.

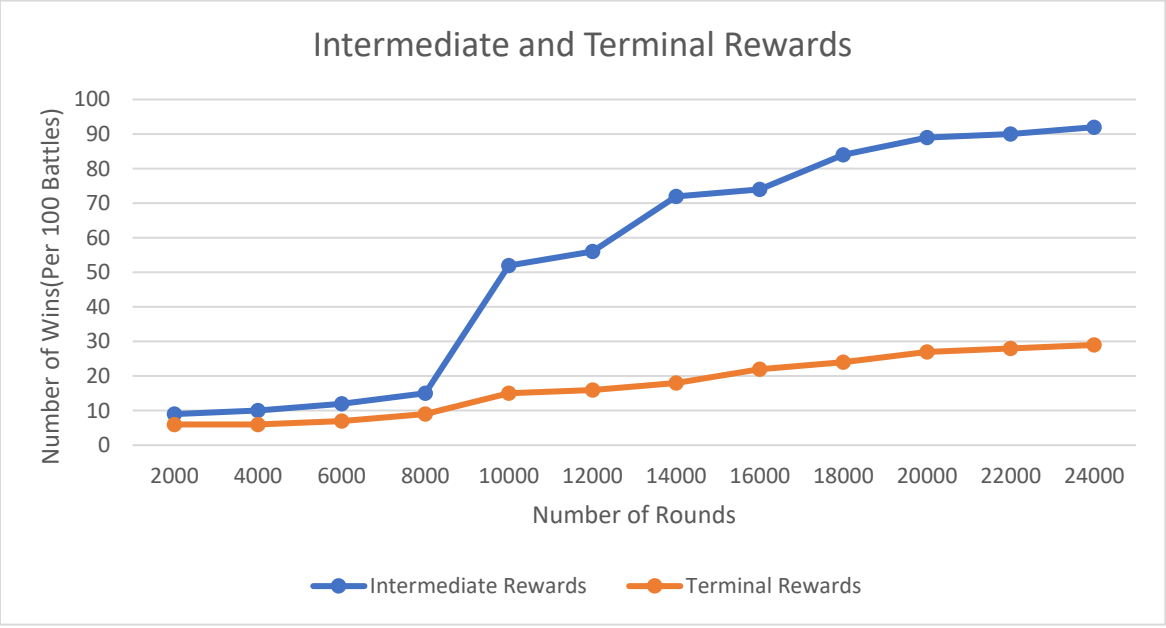


Graph(b) The performance of Robot using Off-policy and On-policy

To make the comparison between the On-Policy and Off-policy, the robot was trained against the same enemy TrackFire for 20000 number of rounds.

Initially off-Policy worked better up to 8000 rounds but after 10000 rounds On-Policy shows better convergence. So, Q-Learning(Off-Policy) works better for Robocode.

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

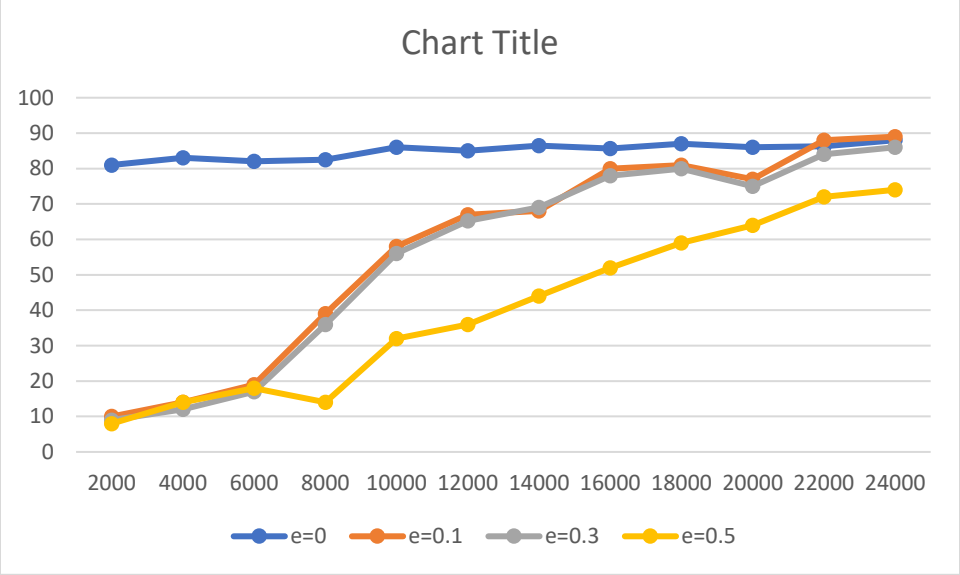


Graph(c) The performance of Robot comparing Intermediate and Terminal Rewards

Using only Terminal Rewards gives poor results while using the Intermediate Rewards along with Terminal rewards gives better performance.

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.



Graph(d) The performance of Robot comparing e=0, e=0.1, e=0.3and e=0.5

The above graph shows the performance of the robot using the different epsilon values while being played for 24000 rounds. As the value of epsilon decreases, the robot performs better.

Appendix

Source Code

FnlBot.java

```
package fnl;
import static robocode.util.Utils.normalRelativeAngleDegrees;
import java.util.Random;
import robocode.AdvancedRobot;
import robocode.BattleEndedEvent;
import robocode.BulletHitEvent;
import robocode.DeathEvent;
import robocode.HitByBulletEvent;
import robocode.HitWallEvent;
import robocode.RoundEndedEvent;
import robocode.ScannedRobotEvent;
import robocode.WinEvent;


public class FnlBot extends AdvancedRobot {

static int batnum = 1;
double alpha = 0.5;
double gamma = 0.9;
int reward;
int numdistance = 5;
int nummenergy = 5;
int numsinebearing = 4;  //**
int numcosbearing = 4;    //**
static int total_states = 5 * 5 * 4 * 4;
static int total_actions = 5;
```

```
int state[][][][] = new int[numdistance][nummenergy][numsinebearing][numcosbearing];
static int action[] = new int[total_actions];

int qdist = 0; //**
int qmener = 0; //**
int qsinbearing = 0;  //**
int qcosbearing = 0;  //**
boolean explore;    //**
boolean greedy;     //**
double epsilon = 0.5;
static int countb = 1;
double gunpower;
static int wincount = 0;
static int gon = 1;              //**
int Results[] = new int[200];

static myLUT LUT = new myLUT(total_states, total_actions);

static int noOfRounds = 0;
static int noOfBattlesWon = 0;
static int noOfBattlesLost = 0;

public void stateindex() {
int m = 0;
for (int i = 0; i < numdistance; i++) {
for (int j = 0; j < nummenergy; j++) {
for (int k = 0; k < numsinebearing; k++) {
for (int l = 0; l < numcosbearing; l++) {
state[i][j][k][l] = m;
m++;
}
```

```
}
}
}
}
public void run() {
stateindex();
while (true) {
turnGunRight(180);
}
}


/* Getting the state and deciding the actions based on policy */
public void onScannedRobot(ScannedRobotEvent e) {
int rwd;
int d_present;
int em_present;
int bsin_present;
int bcos_present;
int a_present = 0;
int d_next;
int em_next;
int bsin_next;
int bcos_next;
int a_next;
double q_present;
double q_next;
int present_state = 0;
int next_state;
int randomaction;
double randomexplore;
```

```
Random dice = new Random();
double distancetobot = e.getDistance();
qdist = quantdistance(distancetobot);

double myenergy = getEnergy();
qmener = quantenergy(myenergy);

double bearingtobot = e.getBearingRadians();
qsinbearing = quantsinbearing(bearingtobot);
qcosbearing = quantcosbearing(bearingtobot);

double absoluteBearing = getHeading() + e.getBearing(); // To point the gun at the enemy
double bearingFromGun = normalRelativeAngleDegrees(absoluteBearing- getGunHeading());

turnGunRight(bearingFromGun);
if (getGunHeat() == 0)
        {
        gunpower = 2;
        }
else
{
        gunpower = 0;
}

// Current state action//
d_present = qdist;
em_present = qmener;
bsin_present = qsinbearing;
bcos_present = qcosbearing;

randomexplore = Math.random();// explore or greedy
```

```
if(randomexplore<epsilon && gon==0)
        {
        explore= true;
        greedy=false;
        }
else
{
        explore= false;
        greedy=true;
}

present_state = state[d_present][em_present][bsin_present][bcos_present];

if (explore)
{
        randomaction = dice.nextInt(5);
        System.out.println(randomaction);
        a_present = randomaction;
}
else if (greedy)
{
        a_present = LUT.getMaxQValueact(present_state);
}

q_present = LUT.getQValue(present_state, a_present);
reward = 0;
rl_action(a_present);
rwd = reward;
scan();
```

```
distancetobot = e.getDistance();
qdist = quantdistance(distancetobot);
myenergy = getEnergy();
qmener = quantenergy(myenergy);
bearingtobot = e.getBearingRadians();
qsinbearing = quantsinbearing(bearingtobot);
qcosbearing = quantcosbearing(bearingtobot);
d_next = qdist;
em_next = qmener;
bsin_next = qsinbearing;
bcos_next = qcosbearing;

next_state = state[d_next][em_next][bsin_next][bcos_next];
a_next = LUT.getMaxQValueact(next_state);
q_next = LUT.getQValue(next_state, a_next);

q_present = q_present + alpha * (rwd + gamma * q_next - q_present);
LUT.setQValue(present_state, a_present, q_present);
}



/* Quantizing the distance */
public int quantdistance(double dist) {
int qdisttoenemy = 0;
if (dist > 0 && dist <= 200) {
qdisttoenemy = 0;
} else if (dist > 200 && dist <= 400) {
qdisttoenemy = 1;
} else if (dist > 400 && dist <= 600) {
qdisttoenemy = 2;
} else if (dist > 600 && dist <= 800) {
```

```
qdisttoenemy = 3;
} else if (dist > 800 && dist <= 1000) {
qdisttoenemy = 4;
}
return qdisttoenemy;
}

/* Quantizing the energy */
public int quantenergy(double ener) {
int qenergy = 0;
if (ener >= 0 && ener <= 20) {
qenergy = 0;
} else if (ener > 20 && ener <= 40) {
qenergy = 1;
} else if (ener > 40 && ener <= 60) {
qenergy = 2;
} else if (ener > 60 && ener <= 80) {
qenergy = 3;
} else if (ener > 80 && ener <= 100) {
qenergy = 4;
}
return qenergy;
}

/* Quantizing the sine bearing */
public int quantsinbearing(double bear) {
double sinbear = Math.sin(bear);
int qbear = 0;
if (sinbear >= -1 && sinbear <= -0.5) {
qbear = 0;
} else if (sinbear > -0.5 && sinbear <= 0) {
```

```
qbear = 1;
} else if (sinbear > 0 && sinbear <= 0.5) {
qbear = 2;
} else if (sinbear > 0.5 && sinbear <= 1) {
qbear = 3;
}
return qbear;
}

//Quantizing the cosine bearing
public int quantcosbearing(double bear) {
double cosbear = Math.cos(bear);
int qbear = 0;
if (cosbear >= -1 && cosbear <= -0.5) {
qbear = 0;
} else if (cosbear > -0.5 && cosbear <= 0) {
qbear = 1;
} else if (cosbear > 0 && cosbear <= 0.5) {
qbear = 2;
} else if (cosbear > 0.5 && cosbear <= 1) {
qbear = 3;
}
return qbear;
}

/* Defining what action to take */
public void rl_action(int act) {
switch (act) {
case 0: {
ahead(100);
break;
```

```
}
case 1: {
back(100);
break;
}
case 2: {
turnLeft(90);
ahead(100);
break;
}
case 3: {
turnRight(90);
ahead(100);
break;
}
case 4: {
fire(gunpower);
break;
}
default: {
doNothing();
break;
}
}
}


/* All rewards */
public void onWin(WinEvent e) {
reward = reward + 10;
wincount++;
```

```
noOfBattlesWon++;
System.out.println("In on win, no of battles won is " + noOfBattlesWon);
System.out.println("reward onwin " + reward);
LUT.saveData();
}

public void onDeath(DeathEvent e) {
reward = reward - 10;
noOfBattlesLost++;
LUT.saveData();

System.out.println("reward ondeath " + reward);
System.out.println("Inside onDeath , no of battles lost is " + noOfBattlesLost);
}

public void onHitByBullet(HitByBulletEvent e) {
reward = reward - 5;
}

public void onBulletHit(BulletHitEvent e) {
reward = reward + 5;
}

public void onBattleEnded(BattleEndedEvent e) {
// LUT.saveData();
// LUT.saveresults();
System.out.println("Inside onBattleEnded");
LUT.displayWins();
}

public void onRoundEnded(RoundEndedEvent e) {
```

```
//System.out.println("I am calling save");
LUT.saveData();
batnum++;
if (batnum == 10000) {
gon = 1;
}
if (countb == 100) {
LUT.battlewin(wincount);
countb = 0;
wincount = 0;
} else {
countb++;
}
//noOfRounds++;
System.out.println("Inside onRoundEnded, no of rounds is " + batnum);
//LUT.displayWins();
//LUT.saveresults();
}
public void onHitWall(HitWallEvent e) {
reward -= 3.5;
double xPos = this.getX();
double yPos = this.getY();
double width = this.getBattleFieldWidth();
double height = this.getBattleFieldHeight();
if (yPos < 80)// too close to the bottom
{
turnLeft(getHeading() % 90);
// System.out.println("Get heading");
// System.out.println(getHeading());
if (getHeading() == 0) {
turnLeft(0);
```

```
}
if (getHeading() == 90) {
turnLeft(90);
}
if (getHeading() == 180) {
turnLeft(180);
}
if (getHeading() == 270) {
turnRight(90);
}
ahead(150);
//System.out.println("Too close to the bottom");
if ((this.getHeading() < 180) && (this.getHeading() > 90)) {
this.setTurnLeft(90);
} else if ((this.getHeading() < 270) && (this.getHeading() > 180)) {
this.setTurnRight(90);
}
} else if (yPos > height - 80) { // to close to the top
//System.out.println("Too close to the Top");
if ((this.getHeading() < 90) && (this.getHeading() > 0)) {
this.setTurnRight(90);
} else if ((this.getHeading() < 360) && (this.getHeading() > 270)) {
this.setTurnLeft(90);
}
turnLeft(getHeading() % 90);
//System.out.println("Get heading");
//System.out.println(getHeading());
if (getHeading() == 0) {
turnRight(180);
}
if (getHeading() == 90) {
```

```
turnRight(90);
}
if (getHeading() == 180) {
turnLeft(0);
}
if (getHeading() == 270) {
turnLeft(90);
}
ahead(150);
} else if (xPos < 80) {
turnLeft(getHeading() % 90);
// System.out.println("Get heading");
// System.out.println(getHeading());
if (getHeading() == 0) {
turnRight(90);
}
if (getHeading() == 90) {
turnLeft(0);
}
if (getHeading() == 180) {
turnLeft(90);
}
if (getHeading() == 270) {
turnRight(180);
}
ahead(150);
} else if (xPos > width - 80) {
turnLeft(getHeading() % 90);
// System.out.println("Get heading");
// System.out.println(getHeading());
if (getHeading() == 0) {
```

```
turnLeft(90);
}
if (getHeading() == 90) {
turnLeft(180);
}
if (getHeading() == 180) {
turnRight(90);
}
if (getHeading() == 270) {
turnRight(0);
}
ahead(150);
}
}
}
```

myLUT.java

```
package fnl;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.Writer;

public class myLUT {
        //static File save_test =new File ("C:/New Folder/new.txt");
//private static final Writer File = null;
int index=0;
int wincount=0;
```

```
static int battle[] =new int[200];
int numstates;
int numactions;
static private double[][] LUT;
myLUT(int nstates,int nactions)
{
numstates=nstates;
numactions=nactions;
LUT = new double[nstates][nactions];

loadData();
}
public void initialize()
{
for (int i = 0; i < numstates; i++)
for (int j = 0; j <numactions; j++)
LUT[i][j] = 0.0;
for(int i=0;i< 200;i++)
{
battle[i]=0;
}
}
public int getMaxQValueact(int state)
{
double Qmaxa=0;
int act=0;
for(int a=0;a<numactions;a++)//predicted next action
{
if(LUT[state][a]>Qmaxa)
{
Qmaxa=LUT[state][a];
```

```
act=a;
}
else
{ continue;
}
}
return(act);
}
public double getQValue(int state, int action)
{
return LUT[state][action];
}
public void setQValue(int state, int action, double value)
{
LUT[state][action] = value;
}
public void loadData()
{
BufferedReader r = null;
try
{
r = new BufferedReader(new FileReader("C:/New Folder/LUT.txt"));
for (int i = 0; i < numstates; i++)
for (int j = 0; j < numactions; j++)
LUT[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()
{

try
{
PrintWriter writer= new PrintWriter("C:/New Folder/LUT.txt");
for (int i = 0; i < numstates; i++)
for (int j = 0; j < numactions; j++)
writer.println(new Double(LUT[i][j]));
if (writer.checkError())
System.out.println("Could not save the data!");
writer.close();
}
catch (IOException e)
```

```
{
System.out.println("IOException trying to write: " + e);
}
}
/*public void display(int bat,int round)
{
	BufferedWriter writer = null;
	try
	{
		writer = new BufferedWriter(new FileWriter(save_test));

	writer.write(new Double(bat)+"\t" +new Double(round));



	}
	catch (IOException e)
	{

		} finally {
			try {
				if (writer != null)
					writer.close();
			} catch (IOException e) {
			}
		}
	}
*/
public void battlewin(int winnum)
{
battle[index]=winnum;
index=index+1;
```

```
System.out.println("Battlewin is called"+":"+winnum);
}
public void displayWins(){
for(int i=0;i<battle.length;i++){
System.out.println(battle[i]);
}
}
}
```

References

[1] Reinforcement Learning, 2nd edition, Richard S. Sutton and Andrew G. Barto

[2] Robocode.io, https://robocode.sourceforge.io/docs/ReadMe.html