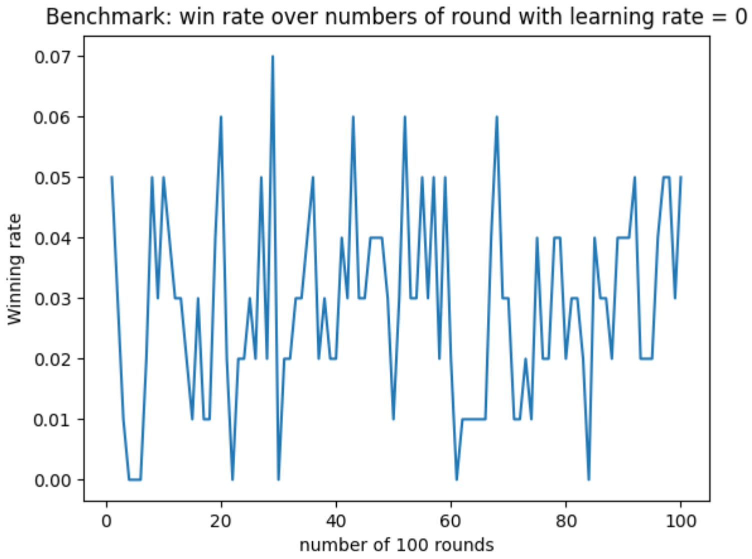
CPEN 502 Part1b Report

Shanny Lu - 57267783

In this file, I use green to highlight my answer to each question.

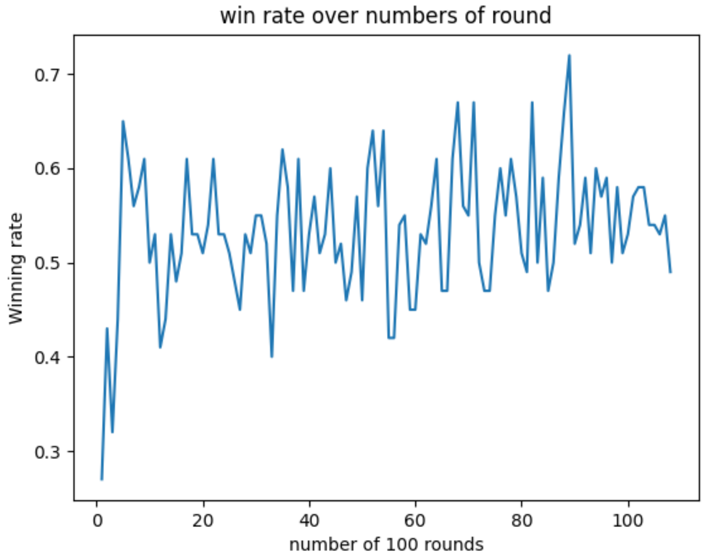


Comments: This is the benchmark for my robot's performance, by setting learning rate = 0. The other parameters are discount factor = 0.9, and epsilon = 0.0. As we can tell from the graph, the benchmark winning rate of my robot remains at around 0.04 when the learning rate = 0. Note that I picked robot **tracker** to battle with my implemented robot for this following report.

Part(2)

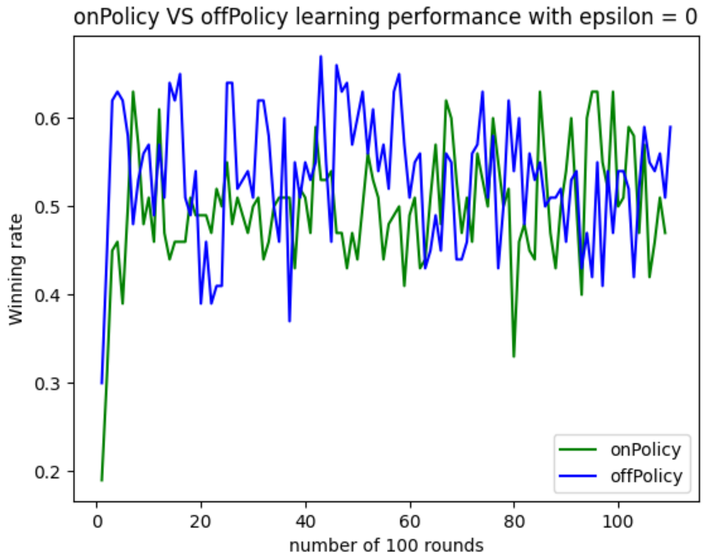
Once you have your robot working, measure its learning performance as follows:

(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.

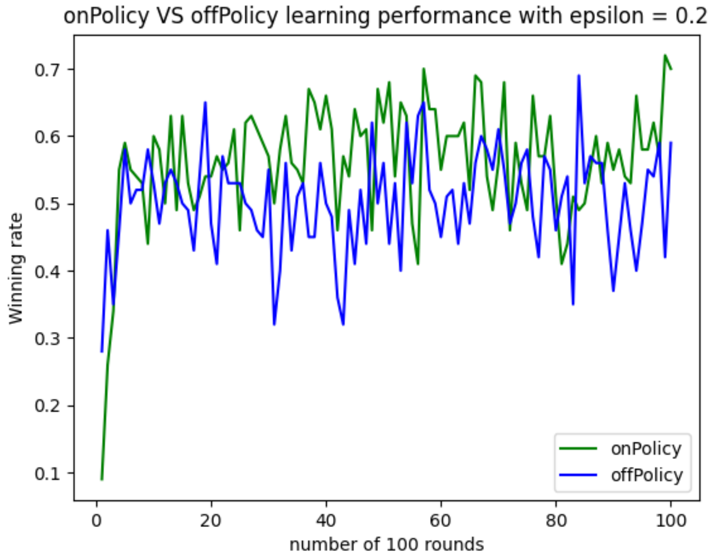


Comments: The parameters used for this graph are learning rate = 0.1, discount factor = 0.9, epsilon = 0.0. As we can tell from the (2a) figure, the winning rate of my robot increases dramatically compared to the benchmark and converges to around 0.55 after 1000 rounds.

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

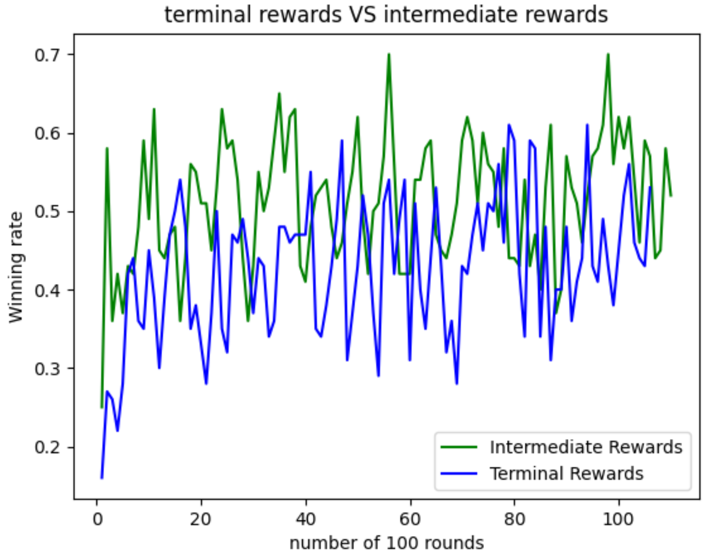


Comments: The parameters used for this graph are learning rate = 0.1, discount factor = 0.9, epsilon = 0.0. The performance between on-policy and off-policy is pretty similar when epsilon = 0.



Comments: The parameters used for this graph are learning rate = 0.1, discount factor = 0.9, epsilon = 0.2. With epsilon = 0.2, the on policy performs better than the off policy when epsilon = 0.2.

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

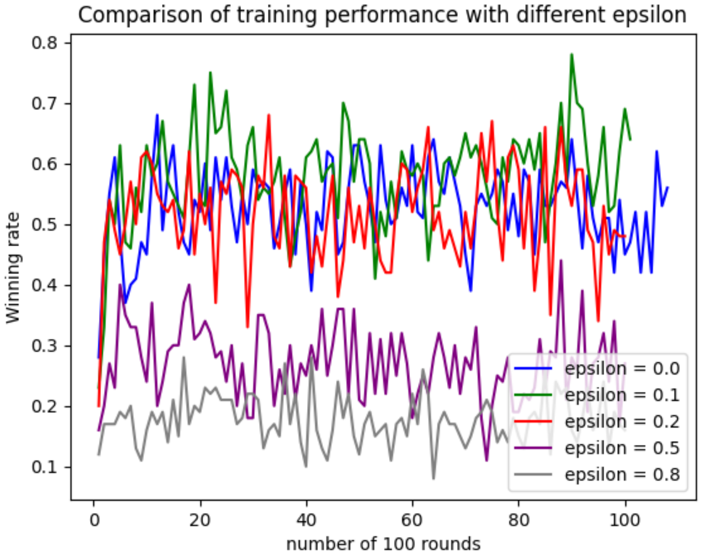


Comments: The parameters used for this graph are learning rate = 0.1, discount factor = 0.9, epsilon = 0.0, off policy. We can tell from the graph that the training including intermediate rewards performs better than the training with terminal rewards only.

Part(3)

This part is about exploration. While training via RL, the next move is selected randomly with probability 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.



Comment: The parameters used for this graph are learning rate = 0.1, discount factor = 0.9, off policy. The graph above consists of the train performance for different epsilon values, particularly when epsilon = 0.0, 0.1, 0.2, 0.5 or 0.8. As we can tell, the performance gets worse when the value of epsilon gets too large, particularly when epsilon > 0.5. And the performance is similar when epsilon values are smaller, particularly when epsilon = 0, 0.1 or 0.2.

Appendix: Source Code Implemented

A LookUpTable.java

```java
import java.io.File;
import java.io.IOException;
import java.util.Random;

public class LookUpTable implements LUTInterface{

    private int myEnergy;
    private int enemyEnergy;
    private int DistanceToEnemy;
    private int DistanceToCenter;
    private int ActionSize;
    private double[][][][][] LUT;
    // keep track of the used actions
    private int[][][][][] visits;


    public LookUpTable(int myEnergy, int enemyEnergy, int DistanceToEnemy, int
        DistanceToCenter, int Action){
        this.myEnergy = myEnergy;
        this.enemyEnergy = enemyEnergy;
        this.DistanceToEnemy = DistanceToEnemy;
        this.DistanceToCenter = DistanceToCenter;
        this.ActionSize = Action;
        this.LUT = new double[myEnergy][enemyEnergy][DistanceToEnemy][DistanceToCenter][Action];
        this.visits = new int[myEnergy][enemyEnergy][DistanceToEnemy][DistanceToCenter][Action];
        initialiseLUT();
    }


    public int visits(double[] X) throws ArrayIndexOutOfBoundsException {
        if (X.length != 5) {
            throw new ArrayIndexOutOfBoundsException();
        } else {
            int i = (int) X[0];
            int j = (int) X[1];
            int k = (int) X[2];
            int m = (int) X[3];
            int n = (int) X[4];
            return visits[i][j][k][m][n];
        }
    }

    public int getExploratoryMove() {
        Random ran = new Random();
        int res = ran.nextInt(ActionSize);
        return res;
    }


    public int getGreedyMove(int myEnergy, int enemyEnergy, int DistanceToEnemy, int
        DistanceToCenter){
```

```
    double bestQ = -Double.MAX_VALUE;
    int GreedyAction = -1;
    for(int i=0; i<ActionSize; i++){
        if(LUT[myEnergy][enemyEnergy][DistanceToEnemy][DistanceToCenter][i] > bestQ){
            bestQ = LUT[myEnergy][enemyEnergy][DistanceToEnemy][DistanceToCenter][i];
            GreedyAction = i;
        }
    }
    return GreedyAction;
}

@Override
public double outputFor(double[] X) {
    return 0;
}

@Override
public double train(double[] X, double argValue) throws ArrayIndexOutOfBoundsException {
    if (X.length != 5) {
        throw new ArrayIndexOutOfBoundsException();
    } else {
        int i = (int) X[0];
        int j = (int) X[1];
        int k = (int) X[2];
        int m = (int) X[3];
        int n = (int) X[4];
        LUT[i][j][k][m][n] = argValue;
        visits[i][j][k][m][n]++;
    }
    return 1;
}

public double getValueFromLUT(int myEnergy, int enemyEnergy, int DistanceToEnemy, int
    DistanceToCenter, int ActionSize){
    return LUT[myEnergy][enemyEnergy][DistanceToEnemy][DistanceToCenter][ActionSize];
}

@Override
public void save(File argFile) {

}

@Override
public void load(String argFileName) throws IOException {

}

@Override
public void initialiseLUT() {
    for(int i=0; i<myEnergy; i++){
        for(int j = 0; j < enemyEnergy; j++) {
            for(int k = 0; k < DistanceToEnemy; k++) {
                for(int m = 0; m < DistanceToCenter; m++) {
                    for(int n = 0; n < ActionSize; n++) {
                        LUT[i][j][k][m][n] = Math.random();
                        visits[i][j][k][m][n] = 0;
                    }
                    }
                }
            }
        }
    }

    @Override
    public int indexFor(double[] X) {
        return 0;
    }

    public int getMyEnergy(){
        return myEnergy;
    }

    public void setMyEnergy(int myEnergy){
        this.myEnergy = myEnergy;
    }

    public int getEnemyEnergy(){
        return enemyEnergy;
    }

    public void setEnemyEnergy(int enemyEnergy){
        this.enemyEnergy = enemyEnergy;
    }

    public int getDistanceToEnemy(){
        return DistanceToEnemy;
    }

    public void setDistanceToEnemy(int DistanceToEnemy){
        this.DistanceToEnemy = DistanceToEnemy;
    }

    public int getDistanceToCenter(){
        return DistanceToCenter;
    }

    public void setDistanceToCenter(int DistanceToCenter){
        this.DistanceToCenter = DistanceToCenter;
    }

    public int getActionSize(){
        return ActionSize;
    }

    public void setActionSize(int Action){
        this.ActionSize = Action;
    }
}
```

B BumbleBee.java

```java
import robocode.*;

import java.awt.*;
import java.io.File;

public class BumbleBee extends AdvancedRobot {

    public enum enumEnergy {zero, low, average, high, highest} // for myEnergy and enemyEnergy
    public enum enumDistance {closest, close, medium, far, farthest} // for DistanceToEnemy and
        DistanceToCenter
    public enum enumAction {attack, forward, backward, left, right}
    public enum enumOperationMode {performScan, performAction}

    static private LookUpTable LUT= new LookUpTable(
            enumEnergy.values().length,
            enumEnergy.values().length,
            enumDistance.values().length,
            enumDistance.values().length,
            enumAction.values().length
    );


    // my state
    public double myX = 0.0;
    public double myY = 0.0;
    public double myEnergy = 0.0;

    // Enemy state
    public double enemyBearing = 0.0;
    public double enemyEnergy = 0.0;
    public double DistanceToEnemy = 0.0; //enemyDistance


    public double centerX = 0.0;
    public double centerY = 0.0;

    // CurrentState Initialization
    private enumEnergy myCurrentEnergy = enumEnergy.highest;
    private enumEnergy enemyCurrentEnergy = enumEnergy.highest;
    private enumDistance currentDisToEnemy = enumDistance.farthest;
    private enumDistance currentDisToCenter = enumDistance.farthest;
    private enumAction currentAction = enumAction.forward;
    private enumOperationMode operationMode = enumOperationMode.performScan;


    // PreviousState Initialization
    private enumEnergy myPrevEnergy = enumEnergy.highest;
    private enumEnergy enemyPrevEnergy = enumEnergy.highest;
    private enumDistance prevDisToEnemy = enumDistance.farthest;
    private enumDistance prevDisToCenter = enumDistance.farthest;
    private enumAction prevAction = enumAction.forward;

    // RL learning parameters
    private double gamma = 0.9;
    private double alpha = 0.1; // 0.0
    private double epsilon = 0.8; // exploration rate: 0.0, 0.1, 0.2, 0.5, 0.8
    private boolean offPolicy = true; // true for Q-leaning, false for Sarsa
```

```java
// reward
private double currentReward = 0.0;
private double negativeReward = -0.1; // set to 0 when only consider terminal
private double positiveReward = 0.5; // set to 0 when only consider terminal
private double negativeTerminalRewards = -0.2;
private double positiveTerminalRewards = 1.0;

// number of round
static int TotalRound = 0;
static int TotalWins = 0;
static int round = 0;

// Logging
static String logFilename = "epsilon=08.log";
static LogFile log = new LogFile();



@Override
public void run() {
    super.run();

    setGunColor(Color.blue);
    setBodyColor(Color.cyan);
    setBulletColor(Color.black);
    setRadarColor(Color.gray);
    setScanColor(Color.green);

    centerX = getBattleFieldWidth()/2;
    centerY = getBattleFieldHeight()/2;





    while(true){
        switch (operationMode){
            case performScan:{
                currentReward = 0.0;
                turnRadarLeft(90);
                break;
            }
            case performAction:{
                if(Math.random() <= epsilon){
                    currentAction = enumAction.values()[LUT.getExploratoryMove()];
                } else {
                    double DistanceToCenter = getDistFromCenter(myX,myY,centerX,centerY);

                    currentAction = enumAction.values()[LUT.getGreedyMove(
                            getEnumEnergyOf(myEnergy).ordinal(),
                            getEnumEnergyOf(enemyEnergy).ordinal(),
                            getEnumDistOf(DistanceToEnemy).ordinal(),
                            getEnumDistOf(DistanceToCenter).ordinal()
                    )];
                }
                switch (currentAction){
                    case attack:
                        setRadarColor(Color.red);
                        double amountToTurn = getHeading() - getGunHeading() + enemyBearing;
                        if(amountToTurn == 360.0 || amountToTurn == -360.0){
                            amountToTurn = 0.0;
                        }
                        turnGunRight(amountToTurn);
                        fire(5);
                        break;

                    case forward:
                        setAhead(100);
                        execute();
                        break;

                    case backward:
                        setBack(100);
                        execute();
                        break;

                    case left:
                        setTurnLeft(30);
                        setAhead(100);
                        execute();
                        break;

                    case right:
                        setTurnRight(30);
                        setAhead(100);
                        execute();
                        break;
                }
            }
            // Update previous Q
            double[] X = new double[]{
                    myPrevEnergy.ordinal(),
                    enemyPrevEnergy.ordinal(),
                    prevDisToEnemy.ordinal(),
                    prevDisToCenter.ordinal(),
                    prevAction.ordinal()
            };

            double QValue = getQValue(currentReward,offPolicy);
            LUT.train(X, QValue);
            operationMode = enumOperationMode.performScan;
            execute();
        }
    }
}

public double getQValue(double currentReward, boolean offPolicy){

    // for sarsa on policy
    double currentQValue = LUT.getValueFromLUT(
            myCurrentEnergy.ordinal(),
            enemyCurrentEnergy.ordinal(),
            currentDisToEnemy.ordinal(),
            currentDisToCenter.ordinal(),
            currentAction.ordinal()
    );

    int GreedyMove = LUT.getGreedyMove(
            myCurrentEnergy.ordinal(),
            enemyCurrentEnergy.ordinal(),
            currentDisToEnemy.ordinal(),
            currentDisToCenter.ordinal()
    );

    // for q-learning off policy
    double maxQValue = LUT.getValueFromLUT(
            myCurrentEnergy.ordinal(),
            enemyCurrentEnergy.ordinal(),
            currentDisToEnemy.ordinal(),
            currentDisToCenter.ordinal(),
            GreedyMove
    );

    double prevQValue = LUT.getValueFromLUT(
            myPrevEnergy.ordinal(),
            enemyPrevEnergy.ordinal(),
            prevDisToEnemy.ordinal(),
            prevDisToCenter.ordinal(),
            prevAction.ordinal()
    );

    double newQValue;
    // Q-learning (off-policy)
    if(offPolicy){
        newQValue = prevQValue + alpha * (currentReward + gamma * maxQValue - prevQValue);
    }else {
        // Sarsa (on-policy)
        newQValue = prevQValue + alpha * (currentReward + gamma * currentQValue -
            prevQValue);
    }

    return newQValue;
}


public enumEnergy getEnumEnergyOf(double energy){
    enumEnergy res;
    if(energy < 0) {
        return null;
    } else if(energy == 0){
        res = enumEnergy.zero;
    } else if(energy < 20) {
        res = enumEnergy.low;
    } else if(energy < 40){
        res = enumEnergy.average;
    } else if (energy < 60){
        res = enumEnergy.high;
    } else {
        res = enumEnergy.highest;
    }
```

```
        return res;
    }

    public enumDistance getEnumDistOf(double dist){
        enumDistance res ;
        if(dist < 0) {
            return null;
        } else if(dist < 100){
            res = enumDistance.closest;
        } else if(dist < 300){
            res = enumDistance.close;
        } else if(dist < 500){
            res = enumDistance.medium;
        } else if(dist < 700){
            res = enumDistance.far;
        } else {
            res = enumDistance.farthest;
        }
        return res;
    }

    public double getDistFromCenter(double myX, double myY, double centerX, double centerY){
        double dist;
        dist = Math.sqrt(Math.pow(myX - centerX, 2) + Math.pow(myY - centerY, 2));
        return dist;
    }


    /**
     * Fire when we see a robot
     */
    @Override
    public void onScannedRobot(ScannedRobotEvent e){
        super.onScannedRobot(e);
        myX = getX();
        myY = getY();
//        myHeading = getHeading();
        enemyBearing = e.getBearing();
        DistanceToEnemy = e.getDistance();
        enemyEnergy = e.getEnergy();
        myEnergy = getEnergy();

        // Update previous state
        myPrevEnergy = myCurrentEnergy;
        enemyPrevEnergy = enemyCurrentEnergy;
        prevDisToEnemy = currentDisToEnemy;
        prevDisToCenter = currentDisToCenter;
        prevAction = currentAction;
        operationMode = enumOperationMode.performAction;

        // Update current state
        myCurrentEnergy = getEnumEnergyOf(getEnergy());
        enemyCurrentEnergy = getEnumEnergyOf(e.getEnergy());
        currentDisToEnemy = getEnumDistOf(e.getDistance());
        currentDisToCenter = getEnumDistOf(getDistFromCenter(myX,myY,centerX,centerY));
        operationMode = enumOperationMode.performAction;
    }
```

```java
/**
 * We were hit! Turn perpendicular to the bullet,
 * so our seesaw might avoid a future shot.
 */
@Override
public void onHitByBullet(HitByBulletEvent e) {
    currentReward += negativeReward;
}

@Override
public void onBulletHit(BulletHitEvent event) {
    currentReward += positiveReward;
}

@Override
public void onBulletMissed(BulletMissedEvent event) {
    currentReward += negativeReward;
}

@Override
public void onHitRobot(HitRobotEvent event) {
    currentReward += negativeReward;
    setBack(200);
    fire(3);
    setTurnRight(60);
    execute();
}

@Override
public void onHitWall(HitWallEvent event) {
    currentReward += negativeReward;
    setBack(200);
    setTurnRight(60);
    execute();
}


public void saveToLog() {
    if ((TotalRound % 100 == 0) && (TotalRound != 0)) {
        double winPercentage = (double) TotalWins / 100;
        System.out.println(String.format("%d, %.3f", ++round, winPercentage));
        File folderDst1 = getDataFile(logFilename);
        log.writeToFile(folderDst1, winPercentage, round);
        TotalWins = 0;
    }
}


@Override
public void onWin(WinEvent event) {
    currentReward = positiveTerminalRewards;

    // update previous Q
    double[] X = new double[]{
            myPrevEnergy.ordinal(),
                enemyPrevEnergy.ordinal(),
                prevDisToEnemy.ordinal(),
                prevDisToCenter.ordinal(),
                prevAction.ordinal()
        };

        double QValue = getQValue(currentReward,offPolicy);
        LUT.train(X, QValue);

        TotalWins++;
        TotalRound++;
        saveToLog();
    }

    @Override
    public void onDeath(DeathEvent event) {
        currentReward = negativeTerminalRewards;

        // update previous Q
        double[] X = new double[]{
                myPrevEnergy.ordinal(),
                enemyPrevEnergy.ordinal(),
                prevDisToEnemy.ordinal(),
                prevDisToCenter.ordinal(),
                prevAction.ordinal()
        };

        double QValue = getQValue(currentReward, offPolicy);
        LUT.train(X, QValue);

        TotalRound++;
        saveToLog();
        //saveTable();
    }


}
```