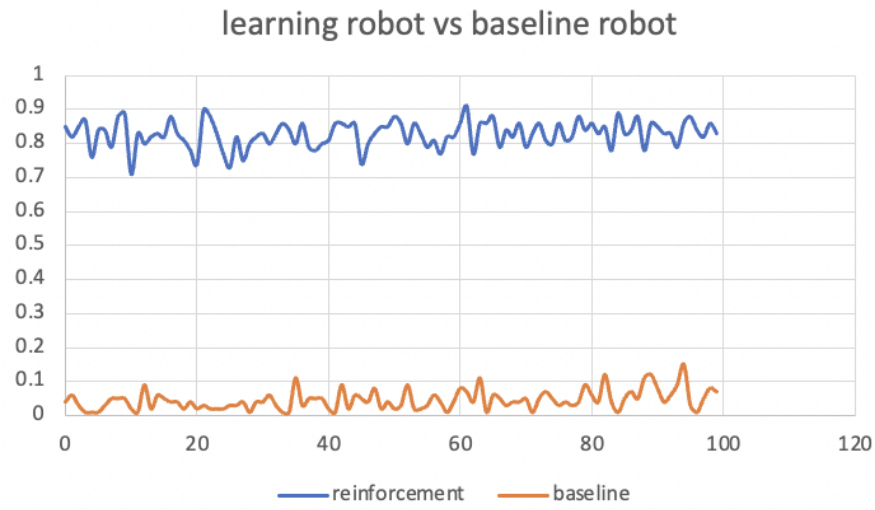
CPEN502 Assignment 2

Jiaqi Zhang (63174551)

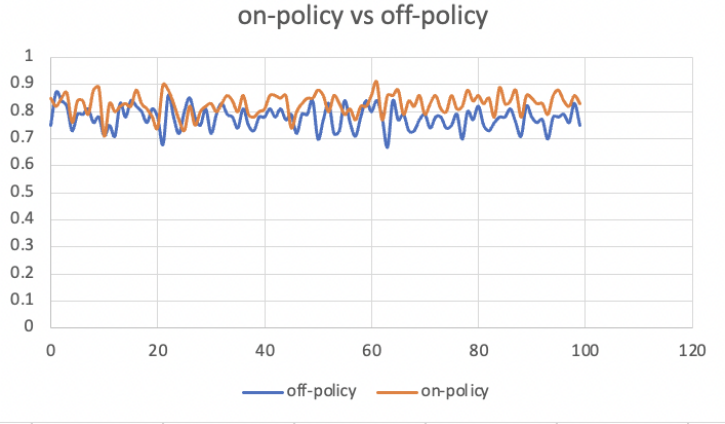
(2) Once you have your robot working, measure its learning performance as follows:

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.



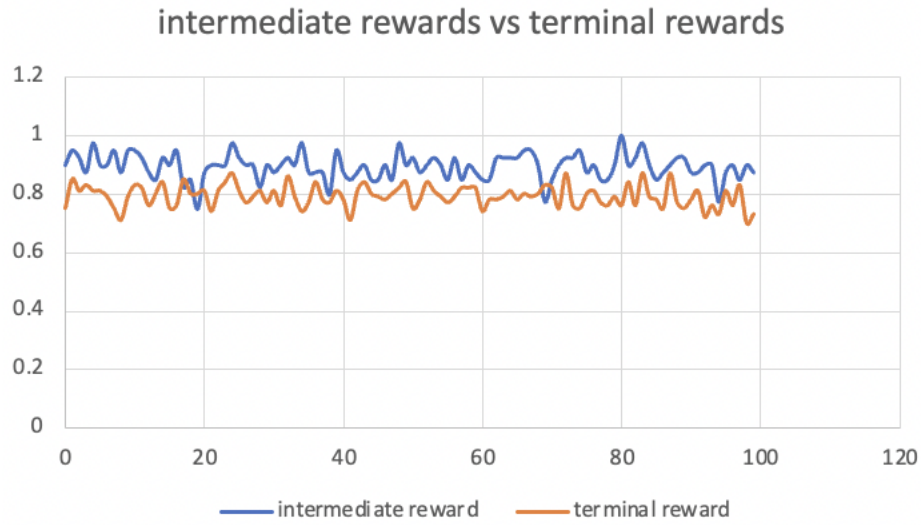
The convergence win rate of my learning robot is around 0.84. I performed 10000 rounds in total. Each win rate is calculated by the number of winning games per 100 rounds. The learning robot performed much better than the baseline robot. The reason why my robot converges quickly is that the size of the look-up table is not very big, and the robot does not need to explore much.

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



The performance of my robot using on-policy performs slightly better than off-policy, but no huge difference.

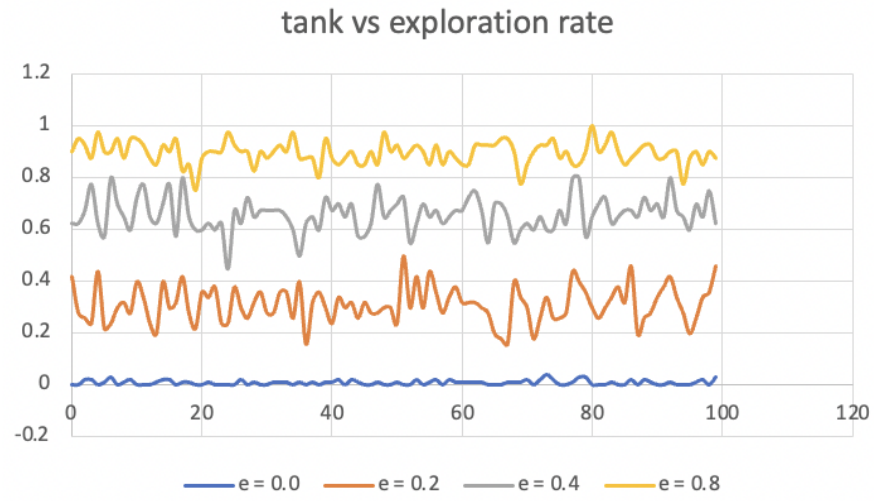
c) Implement a version of your robot that assumes only terminal rewards and show & compare its behaviour with one having intermediate rewards.



The learning robot with intermediate rewards performed better than the robot with terminal rewards.

(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.



As we can see from the chart, the bigger the exploration rate, the better tank performed. When there was no exploration at all, the tank performed worst and did not learn at all.

Action.java

```java
package ece.cpen502;

public class Action {
    public static final int ROBOT_UP = 0;
    public static final int ROBOT_UP_LONG = 1;
    public static final int ROBOT_DOWN = 2;
    public static final int ROBOT_DOWN_LONG = 3;
    public static final int ROBOT_LEFT = 4;
    public static final int ROBOT_RIGHT = 5;
    public static final int ROBOT_FIRE = 6;
    public static final int ROBOT_NUM_ACTIONS = 7;
    public static final double ROBOT_MOVE_SHORT_DISTANCE = 100.0;
    public static final double ROBOT_MOVE_LONG_DISTANCE = 300.0;
    public static final double ROBOT_TURN_DEGREE =  30.0;
}
```

Statistic.java

```java
package ece.cpen502;

import java.util.ArrayList;

public class Statistics {
    private static final int PERIOD = 100;
    private static ArrayList<Integer> scoreList = new ArrayList<>();
    private int numRounds;

    public Statistics(int numRounds) {
        this.numRounds = numRounds;
    }

    public static void saveScore(int win) {
        scoreList.add(win);
    }

    public static void printScoreTable() {
        int size = scoreList.size();
        for(int i=0; i<size; i++) {
            System.out.println("score list: " + i + " " + scoreList.get(i));
        }
    }
```

```
    public static void printWinRates() {
        int size = scoreList.size();
        int groups = size / PERIOD;
        double cnt;
        for(int i=0; i<groups; i++) {
            cnt = 0.0;
            for(int j=0; j<PERIOD; j++) {
                cnt += scoreList.get(i*PERIOD + j);
            }
            System.out.println("win list: " + i + " " + (cnt/PERIOD));
        }
    }
}
```

Enemy.java

```
package ece.cpen502;

import java.awt.geom.Point2D;

public class Enemy {
    public String name;
    public double bearing;
    public double heading;
    public double changeHeading;
    public double x, y;
    public double distance, speed;
    public long ctime;

    public Enemy(String name) {
        this.name = name;
    }

    public Point2D.Double getNextPosition(long gaussTime) {
        double diff = gaussTime - ctime;
        double nextX = x + Math.sin(heading) * speed * diff;
        double nextY = y + Math.cos(heading) * speed * diff;
        return new Point2D.Double(nextX, nextY);
    }
}
```

State.java

```
package ece.cpen502;

public class State {
    public static final double TOTAL_ANGLE = 360.0;
    public static final double CIRCLE = Math.PI * 2;
    public static int NumStates;
    public static final int NUM_DISTANCE = 10;
    public static final int NUM_BREARING = 4;
    public static final int NUM_HEADING = 4;
    public static final int NUM_HIT_BY_BULLETS = 2;
    public static final int NUM_HIT_WALL = 2;
    public static final int NUM_ENERGY = 5;
    public static int states[][][][][][];

    static {
        states = new
int[NUM_DISTANCE][NUM_BREARING][NUM_HEADING][NUM_HIT_BY_BULLETS][NUM_HIT_WALL
][NUM_ENERGY];
        int cnt = 0;
        for(int a=0; a<NUM_DISTANCE; a++) {
            for(int b=0; b<NUM_BREARING; b++) {
                for(int c=0; c<NUM_HEADING; c++) {
                    for(int d=0; d<NUM_HIT_BY_BULLETS; d++) {
                        for(int e=0; e<NUM_HIT_BY_BULLETS; e++) {
                            for(int f=0; f<NUM_ENERGY; f++) {
                                states[a][b][c][d][e][f] = cnt++;
                            }
                        }
                    }
                }
            }
        }
        NumStates = cnt;
    }

    public static int getDistance(double distance) {
        int res = (int)(distance / 100.0);
        return Math.min(NUM_DISTANCE-1, res);
    }

    public static int getHeading(double heading) {
        double angle = TOTAL_ANGLE / NUM_HEADING;
        double newHeading = heading+angle/2;
        while (newHeading > TOTAL_ANGLE) {
            newHeading-=TOTAL_ANGLE;
        }
        return (int)(newHeading/angle);
    }

    public static int getBearing(double bearing) {
        double angle=CIRCLE / NUM_BREARING;
        double newBearing = bearing;
        if(bearing < 0) {
            newBearing += CIRCLE;
        }
```

```
        newBearing += angle / 2;
        if(newBearing > CIRCLE) {
            newBearing = newBearing - CIRCLE;
        }
        return (int) (newBearing / angle);
    }

    public static int getEnergyLevel(double energy) {
        double levels = 100 / NUM_ENERGY;
        return Math.min((int)(energy/levels), NUM_ENERGY-1);
    }
}
```

LookUpTable.java

```
package ece.cpen502;

public class LookUpTable {
    private double table[][];

    public LookUpTable() {
        this.table = new double[State.NumStates][Action.ROBOT_NUM_ACTIONS];
        initializeLUT();
    }

    public void initializeLUT() {
        for(int i=0; i<State.NumStates; i++) {
            for(int j=0; j<Action.ROBOT_NUM_ACTIONS; j++) {
                table[i][j] = 0;
            }
        }
    }

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

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

    public double getMaxValue(int state) {
        double maxValue = -10;
        for(int i=0; i<Action.ROBOT_NUM_ACTIONS; i++) {
            maxValue = Math.max(table[state][i], maxValue);
        }
        return maxValue;
    }
```

```
    public int getBestAction(int state) {
        double maxValue = -10;
        int action = 0;
        for(int i=0; i<Action.ROBOT_NUM_ACTIONS; i++) {
            if(table[state][i] > maxValue) {
                maxValue = table[state][i];
                action = i;
            }
        }
        return action;
    }
}
```

LearningAgent.java

```
package ece.cpen502;

import java.util.ArrayList;

public class LearningAgent {
    public static final double LEARNING_RATE = 0.2;
    public static final double DISCOUNT_RATE = 0.8;
    public static double EXPLORE_RATE = 0.8;
    private int prevState = -1;
    private int prevAction = -1;
    private boolean firstRound = true;
    private LookUpTable table;
    public ArrayList<String> finalStates = new ArrayList<>();

    public LearningAgent(LookUpTable table) {
        this.table = table;
    }

    public void Learn(int currState, int currAction, double reward, boolean
isOnPolicy, boolean isIntermidiateRewards) {
        if(!isIntermidiateRewards) {
            finalStates.add(currState+"-"+currAction);
            return;
        }
        double newValue;
        if(firstRound) {
            firstRound = false;
        } else {
            double oldValue = table.getQValue(prevState, prevAction);
```

```
            if(isOnPolicy) {
                newValue = oldValue + LEARNING_RATE * (reward + DISCOUNT_RATE
* table.getQValue(currState, currAction)
                        -oldValue);
            } else {
                newValue = oldValue + LEARNING_RATE * (reward + DISCOUNT_RATE
* table.getMaxValue(currState) - oldValue);
            }
            table.setQValue(prevState, prevAction, newValue);
        }
        prevState = currState;
        prevAction = currAction;
    }

    public int getNextAction(int state) {
        double random = Math.random();
        if(random < EXPLORE_RATE) {
            return (int)(Math.random() * Action.ROBOT_NUM_ACTIONS);
        }
        return table.getBestAction(state);
    }

    public void feedReward(double value) {
        int n = finalStates.size();
        double currValue, nextValue;
        String[] strs = finalStates.get(n-1).split("-");
        int state = Integer.valueOf(strs[0]);
        int action = Integer.valueOf(strs[1]);
        table.setQValue(state, action, value);
        nextValue = value;
        for(int i=n-2; i>=0; i--) {
            strs = finalStates.get(i).split("-");
            state = Integer.valueOf(strs[0]);
            action = Integer.valueOf(strs[1]);
            currValue = table.getQValue(state, action);
            currValue += LEARNING_RATE * (DISCOUNT_RATE * nextValue -
currValue);
            table.setQValue(state, action, currValue);
            nextValue = currValue;
        }
    }
}
```

MyRobot.java

```java
package ece.cpen502;

import java.awt.Color;
import java.awt.geom.Point2D;
import java.util.ArrayList;

import robocode.*;

public class MyRobot extends AdvancedRobot {
    private static final boolean ON_POLICY = true;
    private static final boolean INTERMEDIATE_REWARD = true;
    private static final boolean BASELINE_ROBOT= false;
    private static final double BASE_DISTANCE = 400.0;
    private Enemy enemy;
    private static LookUpTable table;
    private LearningAgent agent;
    private double reward;
    private double firePower = 1;
    private int isHitByBullet = 0;
    private int isHitWall = 0;
    private ArrayList<Integer> scores = new ArrayList<>();

    public void run() {
        //state = new State();
        table = new LookUpTable();
        agent = new LearningAgent(table);
        enemy = new Enemy("enemy");
        enemy.distance = 10000;

        setAllColors(Color.red);
        setAdjustGunForRobotTurn(true);
        setAdjustRadarForGunTurn(true);
        turnRadarRightRadians(2 * Math.PI);

        while(true) {
            if(!BASELINE_ROBOT) {
                firePower = BASE_DISTANCE / enemy.distance;
                firePower = Math.min(3, firePower);
            }
            radarMovement();
            gunMovement();
            robotMovement();
            execute();
        }
    }

    public void radarMovement() {
        setTurnRadarRightRadians(Double.POSITIVE_INFINITY);
    }

    public void robotMovement() {
        int action;
        if(BASELINE_ROBOT) {
            action = (int)(Math.random() * Action.ROBOT_NUM_ACTIONS);
        } else {
```

```
            int state = getState();
            action= agent.getNextAction(state);
            agent.Learn(state, action, reward, ON_POLICY,
INTERMEDIATE_REWARD);
            reward = 0.0;
            isHitByBullet = 0;
            isHitWall = 0;
        }

        switch (action) {
            case Action.ROBOT_UP:
                setAhead(Action.ROBOT_MOVE_SHORT_DISTANCE);
                break;
            case Action.ROBOT_UP_LONG:
                setAhead(Action.ROBOT_MOVE_LONG_DISTANCE);
                break;
            case Action.ROBOT_DOWN:
                setBack(Action.ROBOT_MOVE_SHORT_DISTANCE);
                break;
            case Action.ROBOT_DOWN_LONG:
                setBack(Action.ROBOT_MOVE_LONG_DISTANCE);
                break;
            case Action.ROBOT_LEFT:
                setTurnLeft(Action.ROBOT_TURN_DEGREE);
                break;
            case Action.ROBOT_RIGHT:
                setTurnRight(Action.ROBOT_TURN_DEGREE);
                break;
            case Action.ROBOT_FIRE:
                setFire(firePower);
                break;
        }
    }

    private int getState() {
        int heading = State.getHeading(getHeading());
        int bearing = State.getBearing(enemy.bearing);
        int distance = State.getDistance(enemy.distance);
        int energy = State.getEnergyLevel(getEnergy());
        return
State.states[distance][bearing][heading][isHitByBullet][isHitWall][energy];
    }

    public void onScannedRobot(ScannedRobotEvent e) {
        if ((enemy.name == e.getName()) || (e.getDistance() <
enemy.distance)) {
            enemy.name = e.getName();
            double bearingRadius = (getHeadingRadians() +
e.getBearingRadians()) % (2 * Math.PI);
            double heading = normaliseBearing(e.getHeadingRadians() -
enemy.heading);
            heading /= (getTime() - enemy.ctime);
            enemy.changeHeading = heading;
            enemy.distance = e.getDistance();
            enemy.x = Math.sin(bearingRadius) * enemy.distance + getX();
            enemy.y = Math.cos(bearingRadius) * enemy.distance + getY();
            enemy.ctime = getTime();
```

```
            enemy.speed = e.getVelocity();
            enemy.bearing = e.getBearingRadians();
            enemy.heading = e.getHeadingRadians();
        }
    }

    private void gunMovement() {
        long gaussTime, nextTime;
        double gunOffset;
        Point2D.Double p = new Point2D.Double(enemy.x, enemy.y);
        for (int i=0; i<20; i++) {
            nextTime = (int)Math.round((getEuDistance(getX(),getY(),p.x,p.y)
/ (20 - (3 * firePower))));
            gaussTime = getTime() + nextTime - 10;
            p = enemy.getNextPosition(gaussTime);
        }

        gunOffset = normaliseBearing(getGunHeadingRadians() -
                (Math.PI/2 - Math.atan2(p.y - getY(),p.x -  getX())));
        setTurnGunLeftRadians(gunOffset);
    }

    public double getEuDistance(double x1, double y1, double x2, double y2)
    {
        double x = x1 - x2;
        double y = y1 - y2;
        return Math.sqrt(x * x + y * y);
    }

    double normaliseBearing(double degree) {
        if (degree > Math.PI) {
            degree -= 2*Math.PI;
        }
        if (degree < -Math.PI) {
            degree += 2*Math.PI;
        }
        return degree;
    }

    public void onHitWall(HitWallEvent e){
        isHitWall = 1;
        if(INTERMEDIATE_REWARD) {
            reward -= 5;
        }
    }

    public void onBulletHit(BulletHitEvent e) {
        if(INTERMEDIATE_REWARD) {
            reward += 10;
        }
    }

    public void onHitByBullet(HitByBulletEvent e) {
        isHitByBullet = 1;
        if(INTERMEDIATE_REWARD) {
            reward -= 10;
        }
```

```
    }

    public void onBulletMissed(BulletMissedEvent e) {
        if(INTERMEDIATE_REWARD) {
            reward -= 3;
        }
    }

    public void onDeath(DeathEvent event) {
        scores.add(0);
        Statistics.saveScore(0);
        if(BASELINE_ROBOT) {
            return;
        }
        if(INTERMEDIATE_REWARD) {
            reward -= 60;
        } else {
            agent.feedReward(0);
        }
    }

    public void onWin(WinEvent event) {
        //System.out.println("win! ");
        //scores.saveScore(1);
        Statistics.saveScore(1);
        if(BASELINE_ROBOT) {
            return;
        }
        if(INTERMEDIATE_REWARD) {
            reward += 60;
        } else {
            agent.feedReward(1);
        }
    }

    public void onBattleEnded(BattleEndedEvent event) {
        Statistics.printWinRates();
    }
}
```