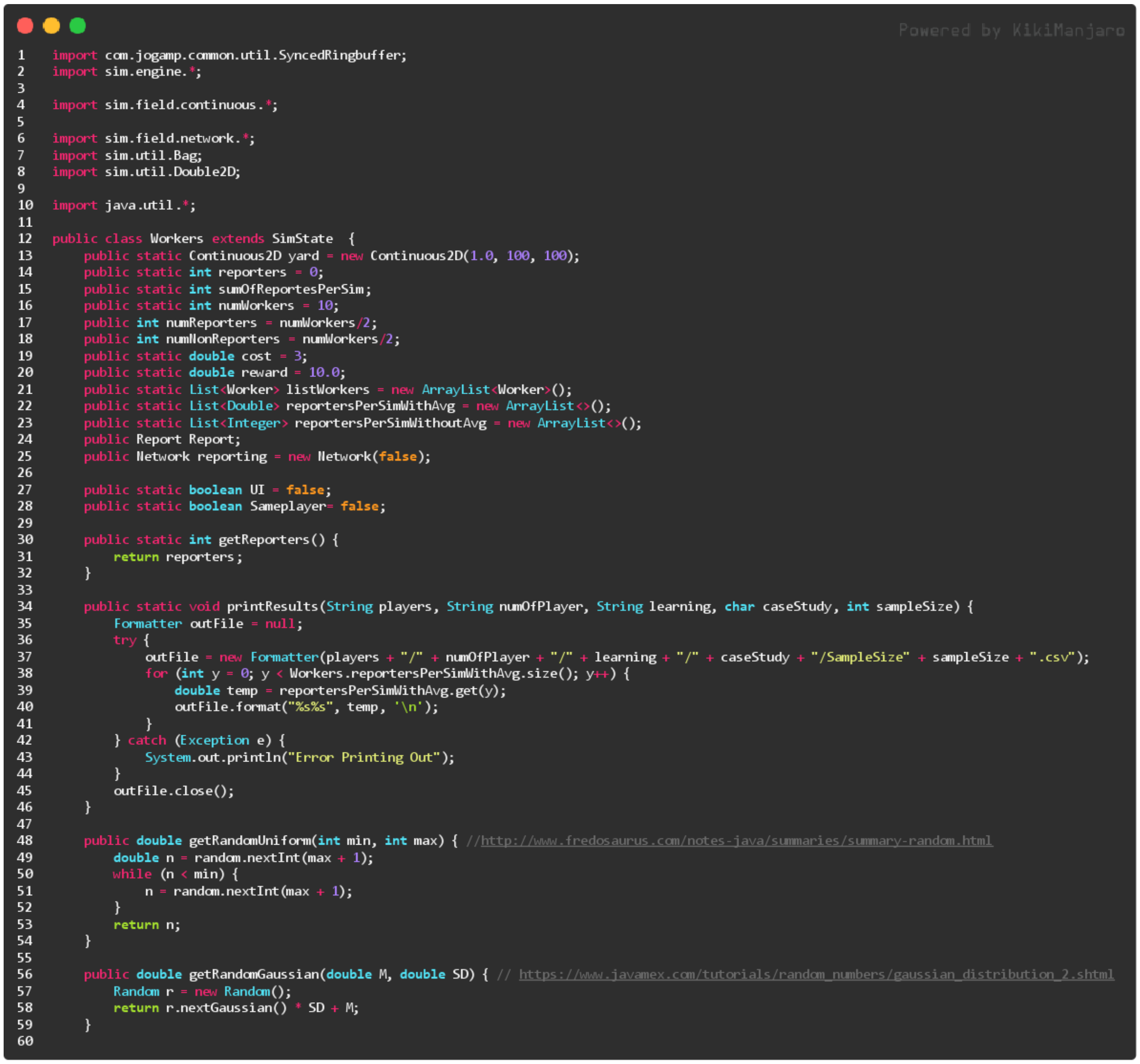
Name: Aly Khaled Anas Dawood

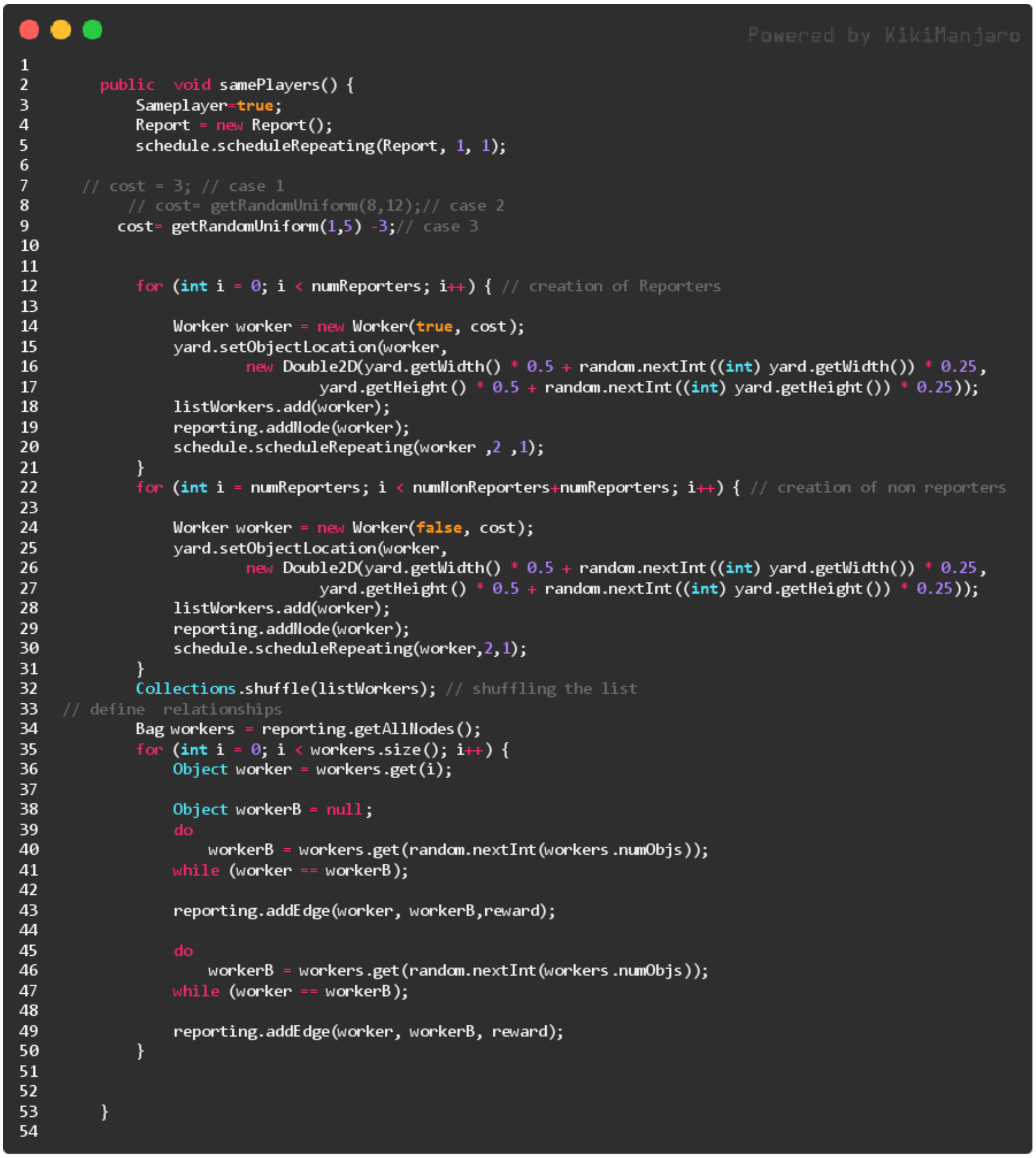
ID:18104862

Workers Class

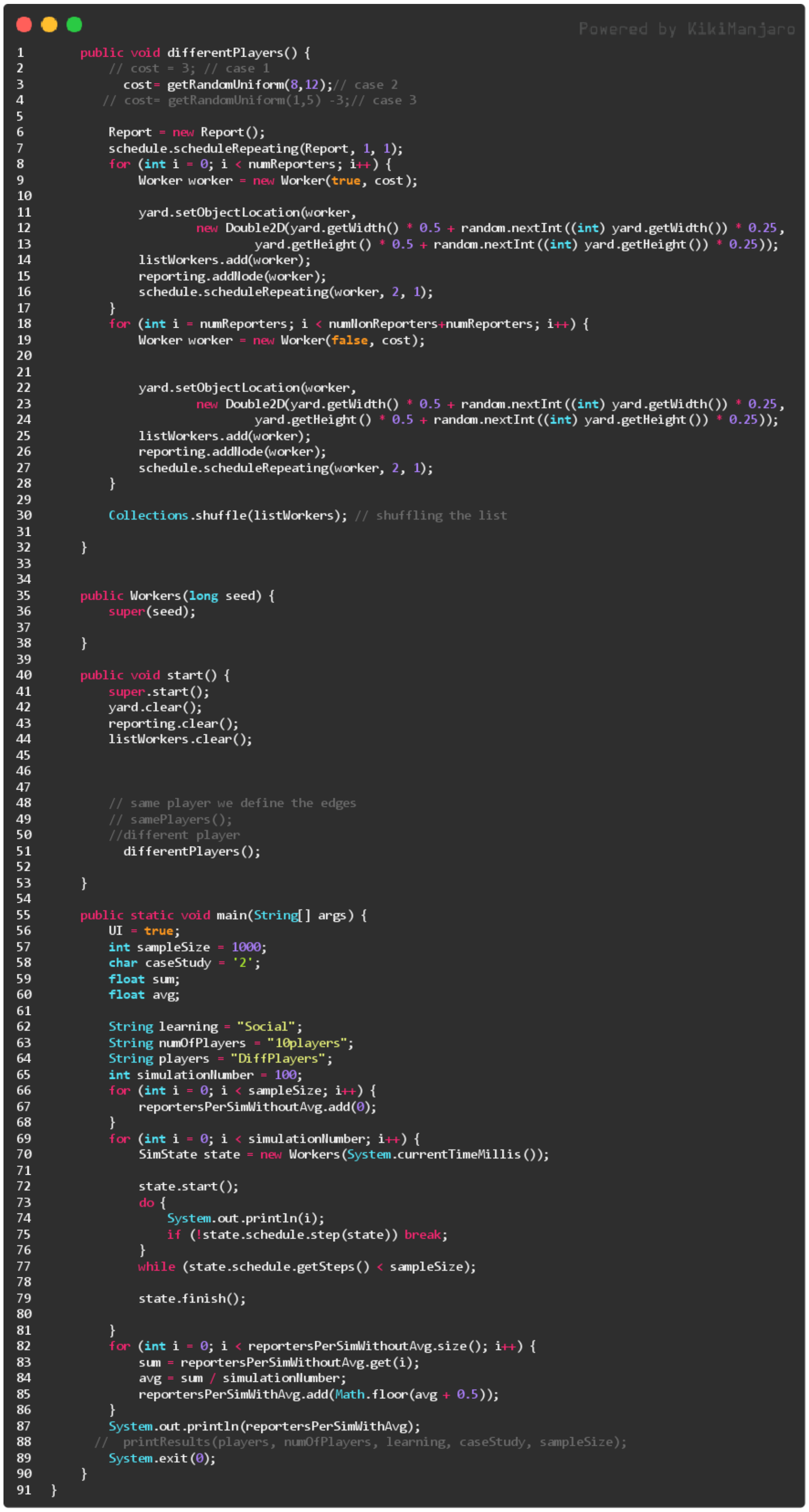


Powered by KikiManjaro

```java
import com.jogamp.common.util.SyncedRingbuffer;
import sim.engine.*;

import sim.field.continuous.*;

import sim.field.network.*;
import sim.util.Bag;
import sim.util.Double2D;

import java.util.*;

public class Workers extends SimState  {
    public static Continuous2D yard = new Continuous2D(1.0, 100, 100);
    public static int reporters = 0;
    public static int sumOfReportesPerSim;
    public static int numWorkers = 10;
    public int numReporters = numWorkers/2;
    public int numNonReporters = numWorkers/2;
    public static double cost = 3;
    public static double reward = 10.0;
    public static List<Worker> listWorkers = new ArrayList<Worker>();
    public static List<Double> reportersPerSimWithAvg = new ArrayList<>();
    public static List<Integer> reportersPerSimWithoutAvg = new ArrayList<>();
    public Report Report;
    public Network reporting = new Network(false);

    public static boolean UI = false;
    public static boolean Sameplayer= false;

    public static int getReporters() {
        return reporters;
    }

    public static void printResults(String players, String numOfPlayer, String learning, char caseStudy, int sampleSize) {
        Formatter outFile = null;
        try {
            outFile = new Formatter(players + "/" + numOfPlayer + "/" + learning + "/" + caseStudy + "/SampleSize" + sampleSize + ".csv");
            for (int y = 0; y < Workers.reportersPerSimWithAvg.size(); y++) {
                double temp = reportersPerSimWithAvg.get(y);
                outFile.format("%s%s", temp, '\n');
            }
        } catch (Exception e) {
            System.out.println("Error Printing Out");
        }
        outFile.close();
    }

    public double getRandomUniform(int min, int max) { //http://www.fredosaurus.com/notes-java/summaries/summary-random.html
        double n = random.nextInt(max + 1);
        while (n < min) {
            n = random.nextInt(max + 1);
        }
        return n;
    }

    public double getRandomGaussian(double M, double SD) { // https://www.javamex.com/tutorials/random_numbers/gaussian_distribution_2.shtml
        Random r = new Random();
        return r.nextGaussian() * SD + M;
    }

```

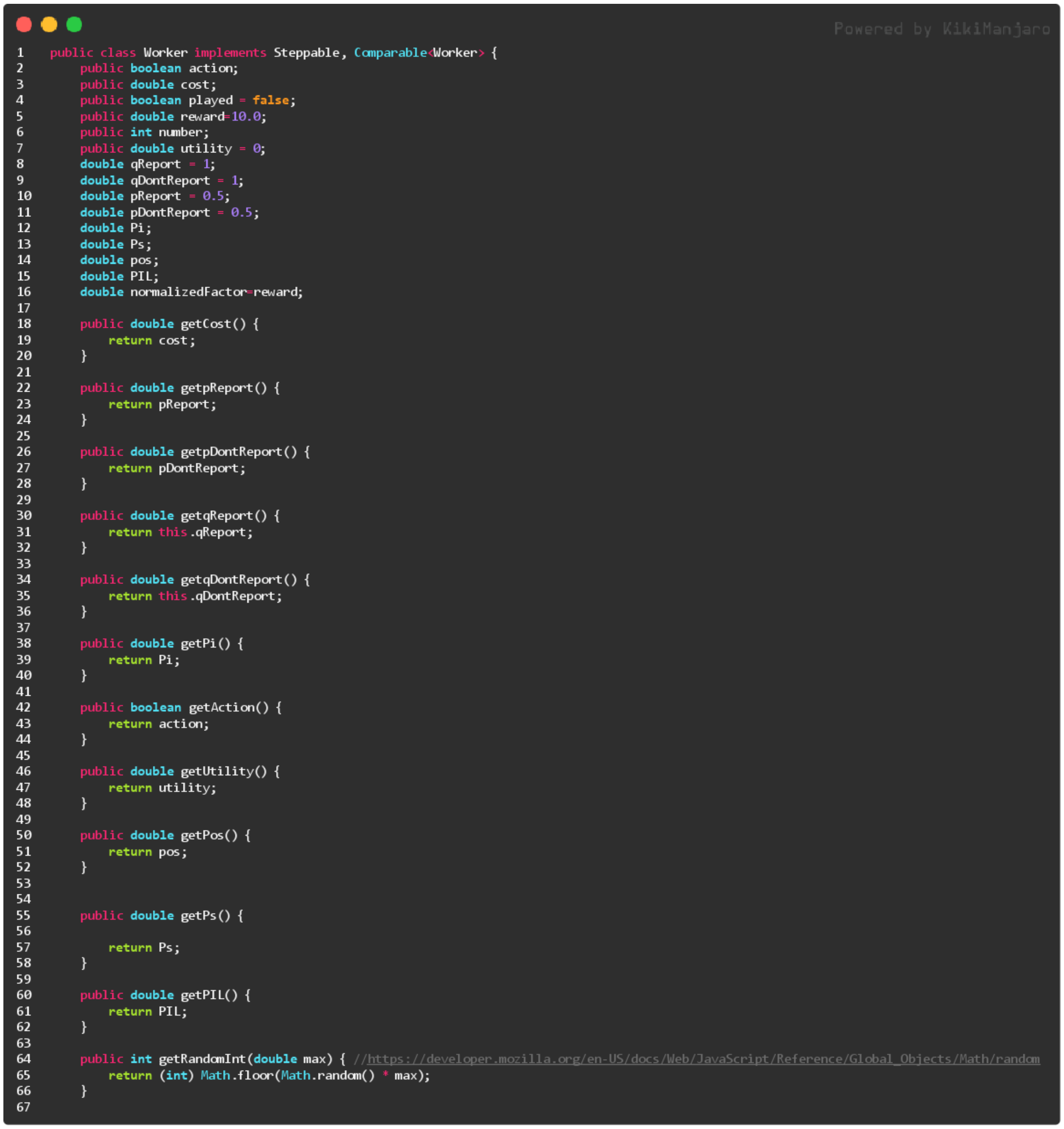


```java

    public  void samePlayers() {
        Sameplayer=true;
        Report = new Report();
        schedule.scheduleRepeating(Report, 1, 1);

  // cost = 3; // case 1
       // cost= getRandomUniform(8,12);// case 2
      cost= getRandomUniform(1,5) -3;// case 3


        for (int i = 0; i < numReporters; i++) { // creation of Reporters

            Worker worker = new Worker(true, cost);
            yard.setObjectLocation(worker,
                    new Double2D(yard.getWidth() * 0.5 + random.nextInt((int) yard.getWidth()) * 0.25,
                            yard.getHeight() * 0.5 + random.nextInt((int) yard.getHeight()) * 0.25));
            listWorkers.add(worker);
            reporting.addNode(worker);
            schedule.scheduleRepeating(worker ,2 ,1);
        }
        for (int i = numReporters; i < numNonReporters+numReporters; i++) { // creation of non reporters

            Worker worker = new Worker(false, cost);
            yard.setObjectLocation(worker,
                    new Double2D(yard.getWidth() * 0.5 + random.nextInt((int) yard.getWidth()) * 0.25,
                            yard.getHeight() * 0.5 + random.nextInt((int) yard.getHeight()) * 0.25));
            listWorkers.add(worker);
            reporting.addNode(worker);
            schedule.scheduleRepeating(worker,2,1);
        }
        Collections.shuffle(listWorkers); // shuffling the list
// define  relationships
        Bag workers = reporting.getAllNodes();
        for (int i = 0; i < workers.size(); i++) {
            Object worker = workers.get(i);

            Object workerB = null;
            do
                workerB = workers.get(random.nextInt(workers.numObjs));
            while (worker == workerB);

            reporting.addEdge(worker, workerB,reward);

            do
                workerB = workers.get(random.nextInt(workers.numObjs));
            while (worker == workerB);

            reporting.addEdge(worker, workerB, reward);
        }


    }

```

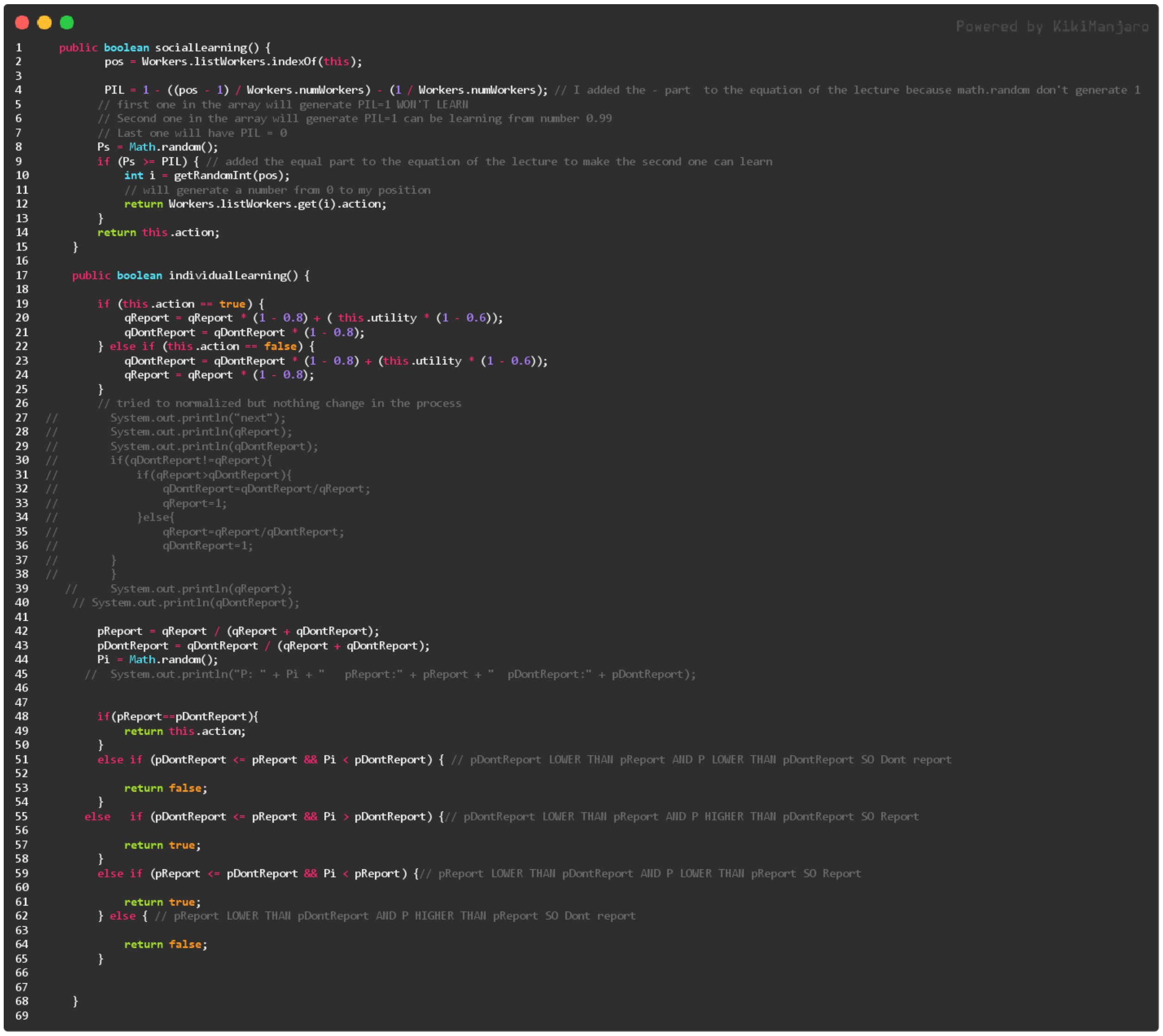


```java
    public void differentPlayers() {
        // cost = 3; // case 1
          cost= getRandomUniform(8,12);// case 2
       // cost= getRandomUniform(1,5) -3;// case 3

        Report = new Report();
        schedule.scheduleRepeating(Report, 1, 1);
        for (int i = 0; i < numReporters; i++) {
            Worker worker = new Worker(true, cost);

            yard.setObjectLocation(worker,
                    new Double2D(yard.getWidth() * 0.5 + random.nextInt((int) yard.getWidth()) * 0.25,
                            yard.getHeight() * 0.5 + random.nextInt((int) yard.getHeight()) * 0.25));
            listWorkers.add(worker);
            reporting.addNode(worker);
            schedule.scheduleRepeating(worker, 2, 1);
        }
        for (int i = numReporters; i < numNonReporters+numReporters; i++) {
            Worker worker = new Worker(false, cost);


            yard.setObjectLocation(worker,
                    new Double2D(yard.getWidth() * 0.5 + random.nextInt((int) yard.getWidth()) * 0.25,
                            yard.getHeight() * 0.5 + random.nextInt((int) yard.getHeight()) * 0.25));
            listWorkers.add(worker);
            reporting.addNode(worker);
            schedule.scheduleRepeating(worker, 2, 1);
        }

        Collections.shuffle(listWorkers); // shuffling the list

    }


    public Workers(long seed) {
        super(seed);

    }

    public void start() {
        super.start();
        yard.clear();
        reporting.clear();
        listWorkers.clear();



        // same player we define the edges
        // samePlayers();
        //different player
          differentPlayers();

    }

    public static void main(String[] args) {
        UI = true;
        int sampleSize = 1000;
        char caseStudy = '2';
        float sum;
        float avg;

        String learning = "Social";
        String numOfPlayers = "10players";
        String players = "DiffPlayers";
        int simulationNumber = 100;
        for (int i = 0; i < sampleSize; i++) {
            reportersPerSimWithoutAvg.add(0);
        }
        for (int i = 0; i < simulationNumber; i++) {
            SimState state = new Workers(System.currentTimeMillis());

            state.start();
            do {
                System.out.println(i);
                if (!state.schedule.step(state)) break;
            }
            while (state.schedule.getSteps() < sampleSize);

            state.finish();

        }
        for (int i = 0; i < reportersPerSimWithoutAvg.size(); i++) {
            sum = reportersPerSimWithoutAvg.get(i);
            avg = sum / simulationNumber;
            reportersPerSimWithAvg.add(Math.floor(avg + 0.5));
        }
        System.out.println(reportersPerSimWithAvg);
      //  printResults(players, numOfPlayers, learning, caseStudy, sampleSize);
        System.exit(0);
    }
}
```

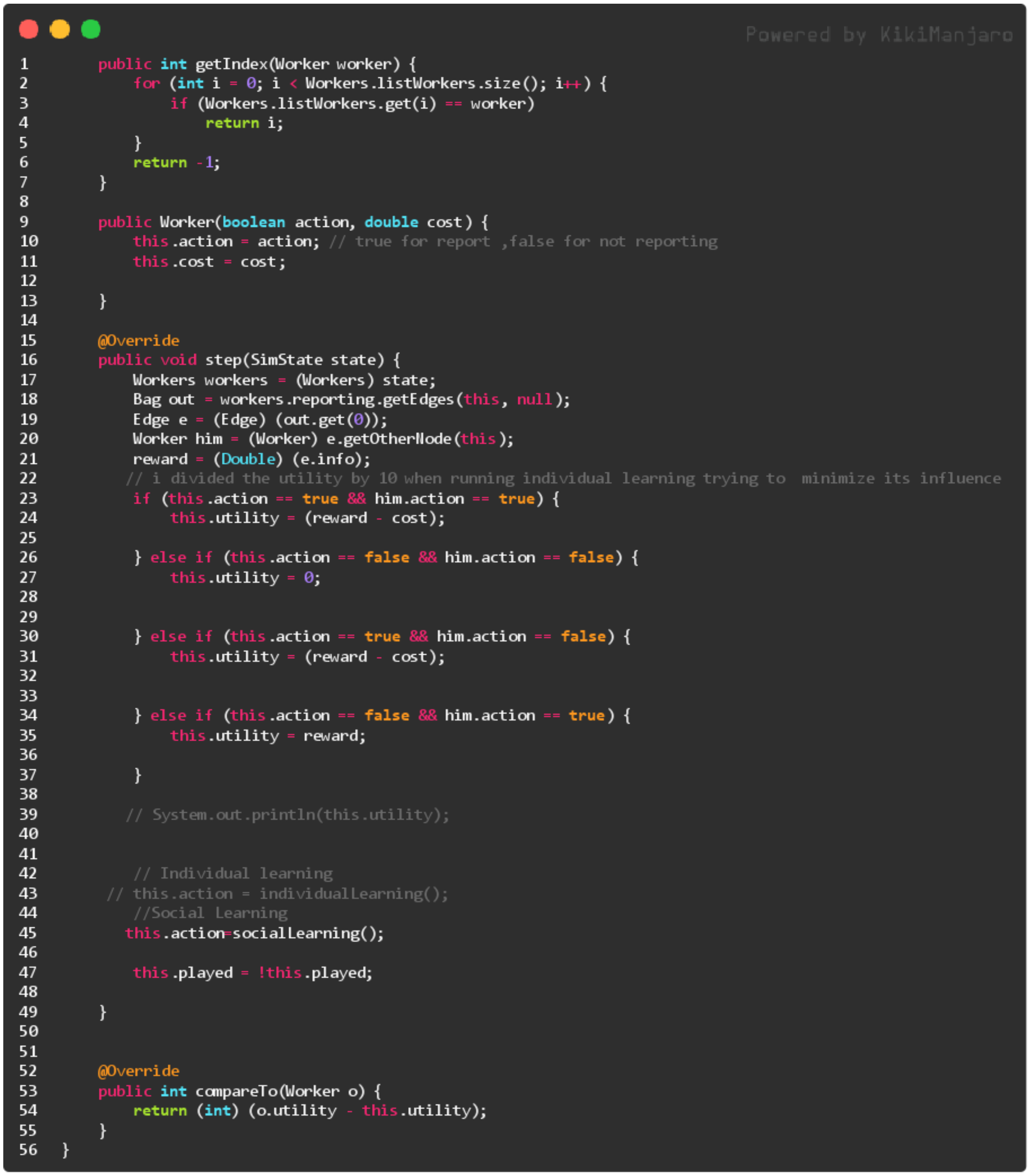
Worker Class



```java
public class Worker implements Steppable, Comparable<Worker> {
    public boolean action;
    public double cost;
    public boolean played = false;
    public double reward=10.0;
    public int number;
    public double utility = 0;
    double qReport = 1;
    double qDontReport = 1;
    double pReport = 0.5;
    double pDontReport = 0.5;
    double Pi;
    double Ps;
    double pos;
    double PIL;
    double normalizedFactor=reward;

    public double getCost() {
        return cost;
    }

    public double getpReport() {
        return pReport;
    }

    public double getpDontReport() {
        return pDontReport;
    }

    public double getqReport() {
        return this.qReport;
    }

    public double getqDontReport() {
        return this.qDontReport;
    }

    public double getPi() {
        return Pi;
    }

    public boolean getAction() {
        return action;
    }

    public double getUtility() {
        return utility;
    }

    public double getPos() {
        return pos;
    }


    public double getPs() {

        return Ps;
    }

    public double getPIL() {
        return PIL;
    }

    public int getRandomInt(double max) { //https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random
        return (int) Math.floor(Math.random() * max);
    }

```



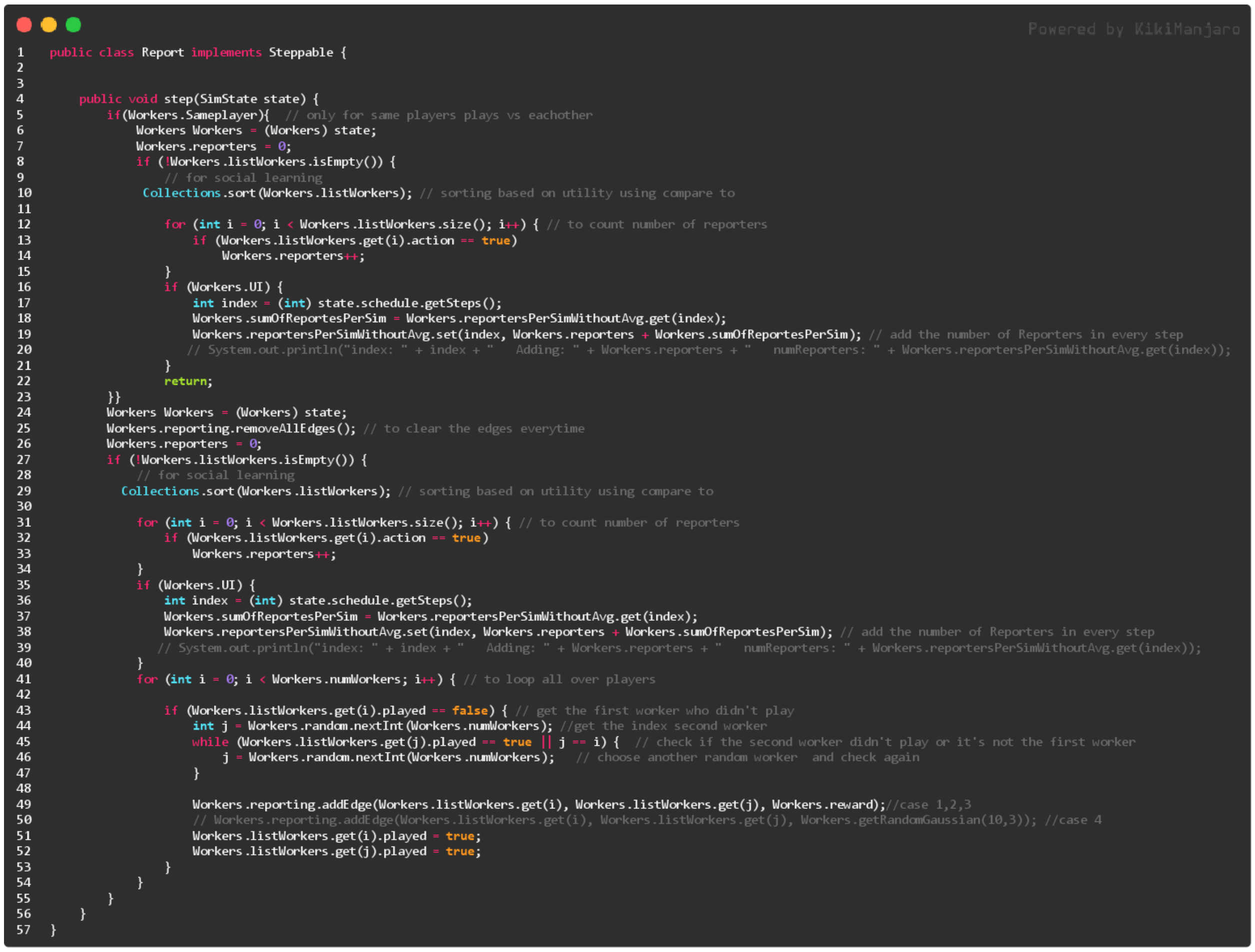
```java
    public boolean socialLearning() {
           pos = Workers.listWorkers.indexOf(this);

           PIL = 1 - ((pos - 1) / Workers.numWorkers) - (1 / Workers.numWorkers); // I added the - part  to the equation of the lecture because math.random don't generate 1
          // first one in the array will generate PIL=1 WON'T LEARN
          // Second one in the array will generate PIL=1 can be learning from number 0.99
          // Last one will have PIL = 0
          Ps = Math.random();
          if (Ps >= PIL) { // added the equal part to the equation of the lecture to make the second one can learn
              int i = getRandomInt(pos);
              // will generate a number from 0 to my position
              return Workers.listWorkers.get(i).action;
          }
          return this.action;
      }

      public boolean individualLearning() {

          if (this.action == true) {
              qReport = qReport * (1 - 0.8) + ( this.utility * (1 - 0.6));
              qDontReport = qDontReport * (1 - 0.8);
          } else if (this.action == false) {
              qDontReport = qDontReport * (1 - 0.8) + (this.utility * (1 - 0.6));
              qReport = qReport * (1 - 0.8);
          }
          // tried to normalized but nothing change in the process
//          System.out.println("next");
//          System.out.println(qReport);
//          System.out.println(qDontReport);
//          if(qDontReport!=qReport){
//              if(qReport>qDontReport){
//                  qDontReport=qDontReport/qReport;
//                  qReport=1;
//              }else{
//                  qReport=qReport/qDontReport;
//                  qDontReport=1;
//          }
//          }
   //       System.out.println(qReport);
    // System.out.println(qDontReport);

          pReport = qReport / (qReport + qDontReport);
          pDontReport = qDontReport / (qReport + qDontReport);
          Pi = Math.random();
        //  System.out.println("P: " + Pi + "   pReport:" + pReport + "  pDontReport:" + pDontReport);


          if(pReport==pDontReport){
              return this.action;
          }
          else if (pDontReport <= pReport && Pi < pDontReport) { // pDontReport LOWER THAN pReport AND P LOWER THAN pDontReport SO Dont report

              return false;
          }
        else   if (pDontReport <= pReport && Pi > pDontReport) {// pDontReport LOWER THAN pReport AND P HIGHER THAN pDontReport SO Report

              return true;
          }
          else if (pReport <= pDontReport && Pi < pReport) {// pReport LOWER THAN pDontReport AND P LOWER THAN pReport SO Report

              return true;
          } else { // pReport LOWER THAN pDontReport AND P HIGHER THAN pReport SO Dont report

              return false;
          }


      }

```



Powered by KikiManjaro

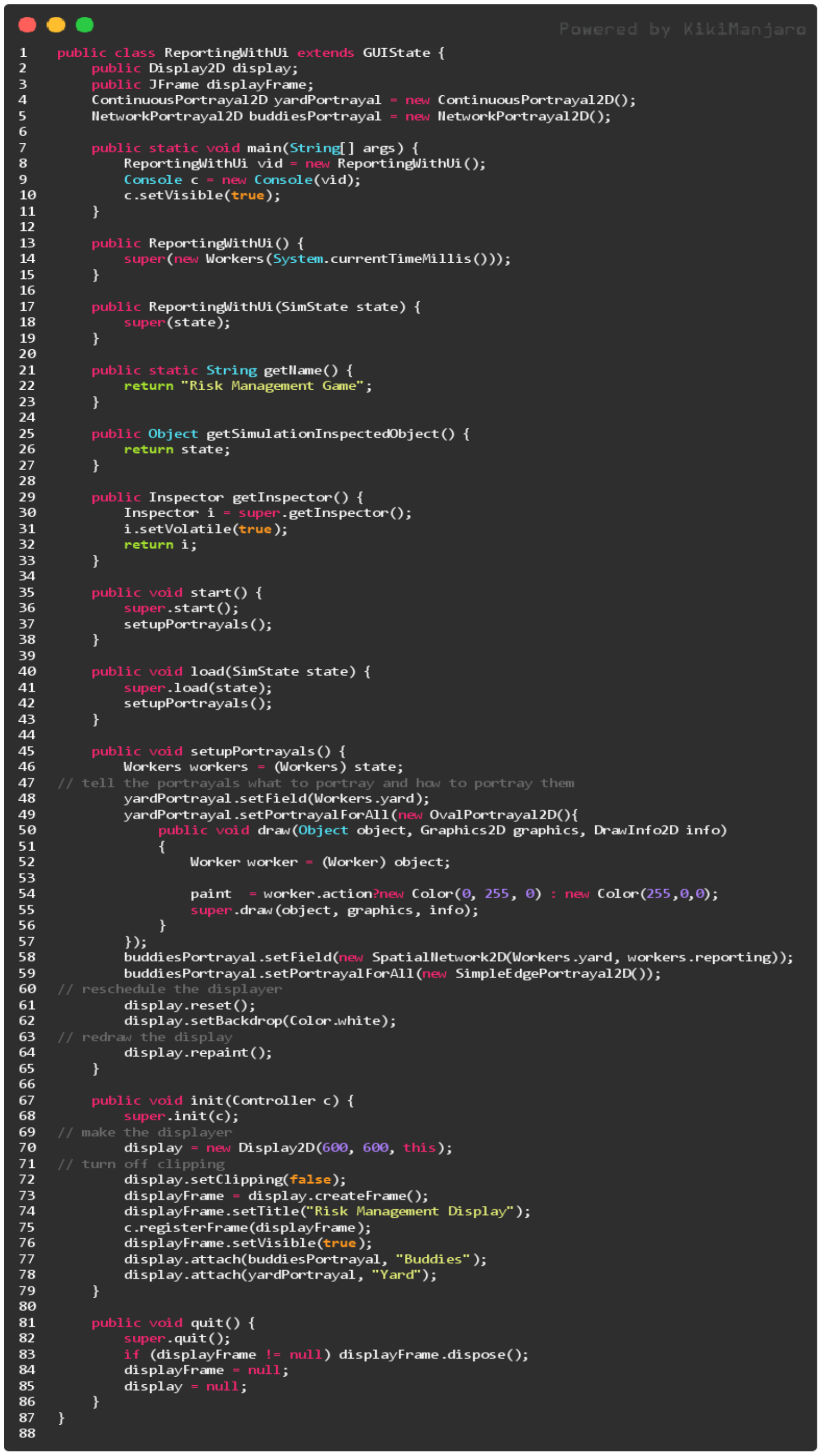
```java
    public int getIndex(Worker worker) {
        for (int i = 0; i < Workers.listWorkers.size(); i++) {
            if (Workers.listWorkers.get(i) == worker)
                return i;
        }
        return -1;
    }

    public Worker(boolean action, double cost) {
        this.action = action; // true for report ,false for not reporting
        this.cost = cost;

    }

    @Override
    public void step(SimState state) {
        Workers workers = (Workers) state;
        Bag out = workers.reporting.getEdges(this, null);
        Edge e = (Edge) (out.get(0));
        Worker him = (Worker) e.getOtherNode(this);
        reward = (Double) (e.info);
       // i divided the utility by 10 when running individual learning trying to  minimize its influence
        if (this.action == true && him.action == true) {
            this.utility = (reward - cost);

        } else if (this.action == false && him.action == false) {
            this.utility = 0;


        } else if (this.action == true && him.action == false) {
            this.utility = (reward - cost);


        } else if (this.action == false && him.action == true) {
            this.utility = reward;

        }

       // System.out.println(this.utility);


        // Individual learning
     // this.action = individualLearning();
        //Social Learning
       this.action=socialLearning();

        this.played = !this.played;

    }


    @Override
    public int compareTo(Worker o) {
        return (int) (o.utility - this.utility);
    }
}
```

Report Class



```java
public class Report implements Steppable {


    public void step(SimState state) {
        if(Workers.Sameplayer){  // only for same players plays vs eachother
            Workers Workers = (Workers) state;
            Workers.reporters = 0;
            if (!Workers.listWorkers.isEmpty()) {
                // for social learning
             Collections.sort(Workers.listWorkers); // sorting based on utility using compare to

                for (int i = 0; i < Workers.listWorkers.size(); i++) { // to count number of reporters
                    if (Workers.listWorkers.get(i).action == true)
                        Workers.reporters++;
                }
                if (Workers.UI) {
                    int index = (int) state.schedule.getSteps();
                    Workers.sumOfReportesPerSim = Workers.reportersPerSimWithoutAvg.get(index);
                    Workers.reportersPerSimWithoutAvg.set(index, Workers.reporters + Workers.sumOfReportesPerSim); // add the number of Reporters in every step
                   // System.out.println("index: " + index + "   Adding: " + Workers.reporters + "   numReporters: " + Workers.reportersPerSimWithoutAvg.get(index));
                }
                return;
        }}
        Workers Workers = (Workers) state;
        Workers.reporting.removeAllEdges(); // to clear the edges everytime
        Workers.reporters = 0;
        if (!Workers.listWorkers.isEmpty()) {
            // for social learning
          Collections.sort(Workers.listWorkers); // sorting based on utility using compare to

            for (int i = 0; i < Workers.listWorkers.size(); i++) { // to count number of reporters
                if (Workers.listWorkers.get(i).action == true)
                    Workers.reporters++;
            }
            if (Workers.UI) {
                int index = (int) state.schedule.getSteps();
                Workers.sumOfReportesPerSim = Workers.reportersPerSimWithoutAvg.get(index);
                Workers.reportersPerSimWithoutAvg.set(index, Workers.reporters + Workers.sumOfReportesPerSim); // add the number of Reporters in every step
               // System.out.println("index: " + index + "   Adding: " + Workers.reporters + "   numReporters: " + Workers.reportersPerSimWithoutAvg.get(index));
            }
            for (int i = 0; i < Workers.numWorkers; i++) { // to loop all over players

                if (Workers.listWorkers.get(i).played == false) { // get the first worker who didn't play
                    int j = Workers.random.nextInt(Workers.numWorkers); //get the index second worker
                    while (Workers.listWorkers.get(j).played == true || j == i) {  // check if the second worker didn't play or it's not the first worker
                        j = Workers.random.nextInt(Workers.numWorkers);   // choose another random worker  and check again
                    }

                    Workers.reporting.addEdge(Workers.listWorkers.get(i), Workers.listWorkers.get(j), Workers.reward);//case 1,2,3
                    // Workers.reporting.addEdge(Workers.listWorkers.get(i), Workers.listWorkers.get(j), Workers.getRandomGaussian(10,3)); //case 4
                    Workers.listWorkers.get(i).played = true;
                    Workers.listWorkers.get(j).played = true;
                }
            }
        }
    }
}
```

ReportingWithUi Class



```java
public class ReportingWithUi extends GUIState {
    public Display2D display;
    public JFrame displayFrame;
    ContinuousPortrayal2D yardPortrayal = new ContinuousPortrayal2D();
    NetworkPortrayal2D buddiesPortrayal = new NetworkPortrayal2D();

    public static void main(String[] args) {
        ReportingWithUi vid = new ReportingWithUi();
        Console c = new Console(vid);
        c.setVisible(true);
    }

    public ReportingWithUi() {
        super(new Workers(System.currentTimeMillis()));
    }

    public ReportingWithUi(SimState state) {
        super(state);
    }

    public static String getName() {
        return "Risk Management Game";
    }

    public Object getSimulationInspectedObject() {
        return state;
    }

    public Inspector getInspector() {
        Inspector i = super.getInspector();
        i.setVolatile(true);
        return i;
    }

    public void start() {
        super.start();
        setupPortrayals();
    }

    public void load(SimState state) {
        super.load(state);
        setupPortrayals();
    }

    public void setupPortrayals() {
        Workers workers = (Workers) state;
// tell the portrayals what to portray and how to portray them
        yardPortrayal.setField(Workers.yard);
        yardPortrayal.setPortrayalForAll(new OvalPortrayal2D(){
            public void draw(Object object, Graphics2D graphics, DrawInfo2D info)
            {
                Worker worker = (Worker) object;

                paint  = worker.action?new Color(0, 255, 0) : new Color(255,0,0);
                super.draw(object, graphics, info);
            }
        });
        buddiesPortrayal.setField(new SpatialNetwork2D(Workers.yard, workers.reporting));
        buddiesPortrayal.setPortrayalForAll(new SimpleEdgePortrayal2D());
// reschedule the displayer
        display.reset();
        display.setBackdrop(Color.white);
// redraw the display
        display.repaint();
    }

    public void init(Controller c) {
        super.init(c);
// make the displayer
        display = new Display2D(600, 600, this);
// turn off clipping
        display.setClipping(false);
        displayFrame = display.createFrame();
        displayFrame.setTitle("Risk Management Display");
        c.registerFrame(displayFrame);
        displayFrame.setVisible(true);
        display.attach(buddiesPortrayal, "Buddies");
        display.attach(yardPortrayal, "Yard");
    }

    public void quit() {
        super.quit();
        if (displayFrame != null) displayFrame.dispose();
        displayFrame = null;
        display = null;
    }
}
```