```java
package hawk.dove.game;
import java.util.List;
import sim.engine.SimState;
import sim.engine.Steppable;
import ec.util.MersenneTwisterFast;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;


public class PlayerAgent implements Steppable {
    private static final MersenneTwisterFast randomNumberGenerator = new MersenneTwisterFast();
    private float payOff;
    private float prevPayOff;
    private final String name;
    private Strategy strategy;
    public Boolean isPlaying;
    public PlayerAgent(String name) {
        this.name = name;
        this.payOff = 0;
        this.prevPayOff = 0;
        this.isPlaying = false;
        if(randomNumberGenerator.nextInt(5000) % 2 == 0)
            this.strategy = Strategy.Hawk;
        else
            this.strategy = Strategy.Dove;
    }

    public float getPayOff() {
        return this.payOff;
    }

    public String getName() {
        return name;
    }

    public Strategy getStrategy() {
        return strategy;
    }

    public float getPrevPayOff() {
        return prevPayOff;
    }

    public boolean updatePayOff(Strategy OpponentStrategy,int value, int cost){
        this.prevPayOff = this.payOff;
        boolean isWinning = false;
        if(this.strategy == Strategy.Hawk && OpponentStrategy == Strategy.Hawk)
            this.payOff = this.prevPayOff + value / 2 - cost;
        else if(this.strategy == Strategy.Hawk && OpponentStrategy == Strategy.Dove)
        {
            isWinning = true;
            this.payOff = this.prevPayOff + value;
        }
        else if(this.strategy == Strategy.Dove && OpponentStrategy == Strategy.Dove)
            this.payOff = this.prevPayOff + value / 2;
        else if(this.strategy == Strategy.Dove && OpponentStrategy == Strategy.Hawk)
            this.payOff = this.prevPayOff;
        return isWinning;
    }

    public boolean changeStrategy(Strategy OpponentStrategy){
        boolean isNegativeUtility = this.payOff < this.prevPayOff;
        boolean OpposedStrategy = OpponentStrategy !=
this.strategy;
        boolean res = isNegativeUtility || OpposedStrategy;
        if(res)
        {
            if(this.strategy == Strategy.Hawk)
                this.strategy = Strategy.Dove;
            else
                this.strategy = Strategy.Hawk;
        }
        return res;
    }

    @Override
    public String toString() {
        return this.getName() + " { " + ((this.getStrategy() == Strategy.Hawk)? "Hawk" : "Dove") + " }";
    }

    public boolean requestToEnterBattle(Battle battle){
        if(battle.isBattleRoomFull())
            return false;
        battle.Players.push(this);
        return true;
    }
    @Override
    public void step(SimState state) {
        HawkDoveGame game = (HawkDoveGame) state;
        List<Battle> battleRooms = game.BattleRooms;
        for(int i = 0; i < battleRooms.size(); i++)
        {
            Battle battleRoom = battleRooms.get(i);
            if(battleRoom.isBattleRoomFull() == false)
            {
                if(this.requestToEnterBattle(battleRoom) == false)
                    continue;
                game.writer.write("\n" + this.getName() + " has entered " + battleRoom.BattleRoomName);
                game.writer.write("\n-----------------------------------------------------");
                battleRoom.Battle(game.writer);
                BattleReport battleReport = new BattleReport(battleRoom);
                boolean add = game.BattleReports.add(battleReport);
                try {
                    BattleReport.logBattle(battleReport);
                } catch (IOException ex) {
                    Logger.getLogger(PlayerAgent.class.getName()).log(Level.SEVERE, null, ex);
                }
                if(battleRooms.remove(battleRoom))
                {
                    game.writer.write("\n" + battleRoom.BattleRoomName + " has finished");
                    game.writer.write("\n-----------------------------------------------------");
                }
                return;
            }
        }
        int cost = 0;
        int value = 0;
        switch(game.Cost)
            {
                case Constant:
                    cost = 10;
                    break;
                case UniformDistribution:
                    cost = game.costUniformDistributer.
```

```java
nextInt();
                    break;
                case NormalDistribution:
                    cost = game.costNormalDistributer.nextInt();
                    break;
            }

            switch(game.Value)
            {
                case Constant:
                    value = 100;
                    break;
                case NormalDistribution:
                    value = game.costNormalDistributer.nextInt();
                    break;
            }
        Battle newBattle = new Battle(value, cost, this);
        if(battleRooms.add(newBattle))
        {
            game.writer.write("\n" + newBattle.BattleRoomName + "Created by " + this.getName());
            game.writer.write("\n------------------------------------------------------");
            game.writer.write("\n" + this.getName()  + " has entered " + newBattle.BattleRoomName);
            game.writer.write("\n------------------------------------------------------");
        }

    }
}

```