CA425
Artificial Intelligence

Expressing Naughts & Crosses as a Reinforcement Learning problem

Name:	Ciaran McNally
Student ID:	58079978

Game Introduction

Naughts-and Crosses has Nine positions in total, lined up in a 3x3 format. There are eight win states to this game. A win occurs when either Player lines up 3 of their own symbols in a row. The winning positions are 3 going across the state-space, 3 going down the state-space and then the 2 diagonals. Each of the two players marks their symbol in turn. The Symbols commonly used are ‘X’ and ‘O’.

Before I began diving in and implementing a Q-learning solution to Naughts-and-Crosses, I decided to investigate if there were any inherent advantages or disadvantages in the game itself. I also thought this would be a useful way of testing that my code works. I made two bots play against each other 10 Million times. These Naive Players simply picked a random free position in alternating goes. The data I generated can be seen below.

Runs	Player 1 wins	Player 2 Wins	Draw
1	576362	296251	127387
2	575465	297807	126728
3	575888	297817	126295
4	575864	296962	127174
5	576341	296781	126878
6	576302	296406	127292
7	576433	296532	127035
8	575421	297397	127182
9	576189	297245	126566
10	575956	296560	127484
Total	5760221	2969758	1270021
% of 10 million	57.60221	29.69758	12.70021

This data would suggest that there is quite a significant advantage to the Player who moves first. I figured this out by setting it so Player 1 always moved first. This means I must set my game to choose the player who moves first at random to ensure my results in further subsequent stages aren’t off. I may also look into how this advantage could affect the learning rate of my Q-learning player at later stages.

Game Code - Game.java

```java
import java.util.Random;
import java.util.Arrays;

//This class represents the Noughts and Crosses Game Environment.
public class Game{
    //Stores game board
    private int[] state = new int[9];
    public int turn = 1;

    public void newBoard(){
        //reset state of Game Board
        for(int i=0;i<state.length;i++){
            state[i] = 0;
        }
        //randomize who takes first turn
        Random r = new Random();
        boolean whoFirst = r.nextBoolean();
        if(whoFirst)    turn = 1;
        else    turn = 2;
    }

    public String getState(){
        String s = Arrays.toString(state);
        String theState = s.replaceAll("[^\\d]","");
        return theState;
    }

    //Print out board - Debugging purposes...
    public void printState(String state){
        char[] sp = state.toCharArray();
        System.out.println(sp[0]+" "+sp[1]+" "+sp[2]);
        System.out.println(sp[3]+" "+sp[4]+" "+sp[5]);
        System.out.println(sp[6]+" "+sp[7]+" "+sp[8]+"\n");
    }

    //return next state string
    public String getNextState(int who, int action){
        int[] nextState = state;
        nextState[action] = who;
        String s = Arrays.toString(nextState);
        String theState = s.replaceAll("[^\\d]","");
        return theState;
    }

    public void makeMove(int who, int move){
        state[move] = who;
        if(who == 1)    turn=2;
        else    turn=1;
    }

    public int whoWins(){
        if((state[0]==state[1] & state[1]==state[2]) & (state[0]!=0))
            return state[0]; //top row
        else if((state[3]==state[4] & state[4]==state[5]) & (state[3]!=0))
            return state[3]; //middle row
        else if((state[6]==state[7] & state[7]==state[8]) & (state[6]!=0))
            return state[6]; //bottom row
        else if((state[0]==state[3] & state[3]==state[6]) & (state[0]!=0))
            return state[0]; //left side down
        else if((state[1]==state[4] & state[4]==state[7]) & (state[1]!=0))
            return state[1]; //middle down
        else if((state[2]==state[5] & state[5]==state[8]) & (state[2]!=0))
            return state[2]; //right side down
        else if((state[0]==state[4] & state[4]==state[8]) & (state[0]!=0))
            return state[0]; //left to right diag down
        else if((state[2]==state[4] & state[4]==state[6]) & (state[2]!=0))
            return state[2]; //right to left diag down
        else if(isDraw())
            return 3;
        else return 0;
    }

    public boolean isDraw(){
        int count = 0;
        for(int i=0; i < state.length; i++){
            if(state[i]==0)
                count++;
        }
        if(count > 0)   return false;
        else    return true;
    }

    public boolean isValidAction(int who, int move){
        if((state[move]==0) & (who == turn))
            return true;
        else return false;
    }
}
```

Random Player code - RPlayer.java

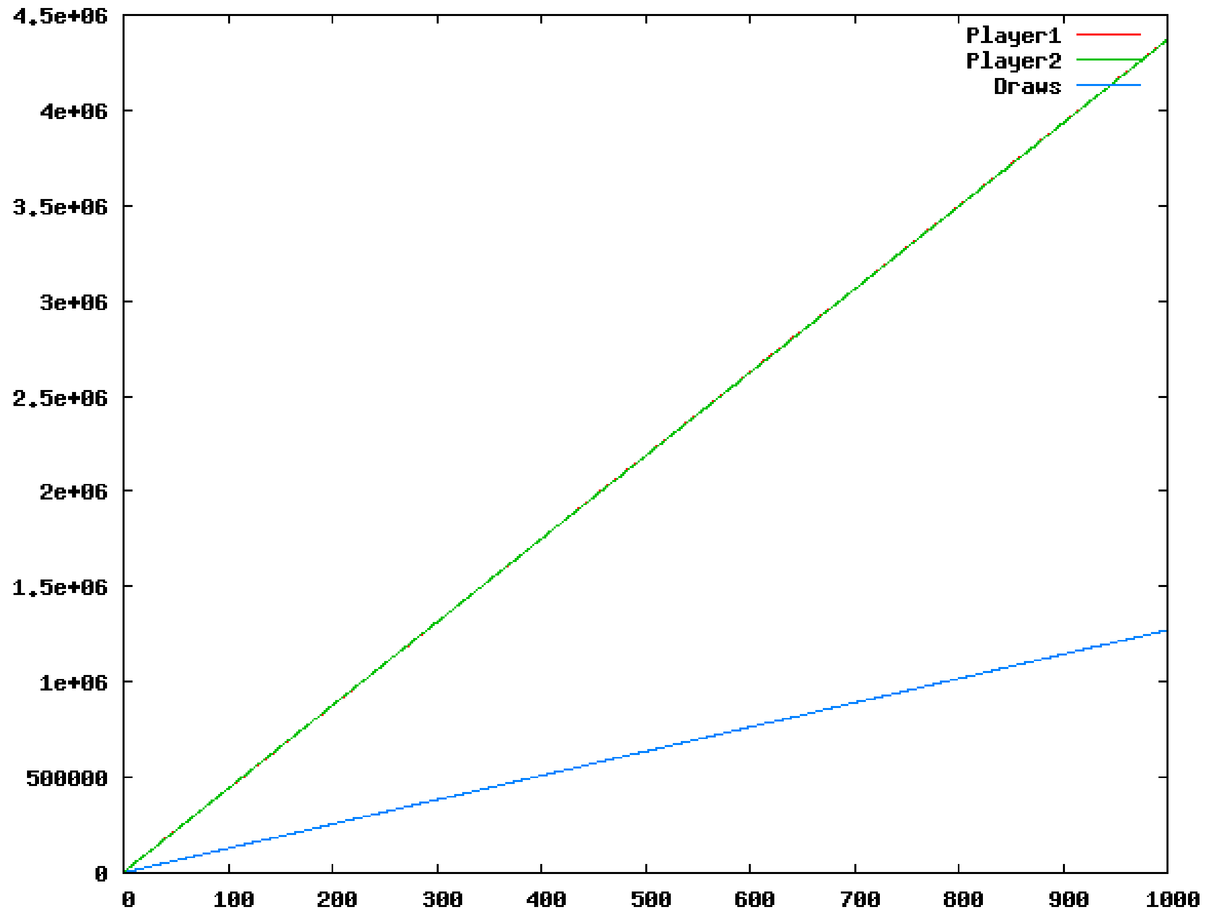
```java
import java.util.Random;

public class RPlayer{
    int currentMove;
    String currentState;
    public void nextAction(String state){
        Random r = new Random();
        int random = r.nextInt(9);
        currentMove = random;
        currentState = state;
    }
}
```

Random Player vs Random Player

In the first round of tests I tried a random Player vs a random Player and received very expected results. Below you will see that both players win approximately an equal number of games. I ran this test run 10 million times taking a snapshot of the win ratio every 10,000 runs and then plotted the result. Below (Image 1.0) you will notice that both players line up as expected.

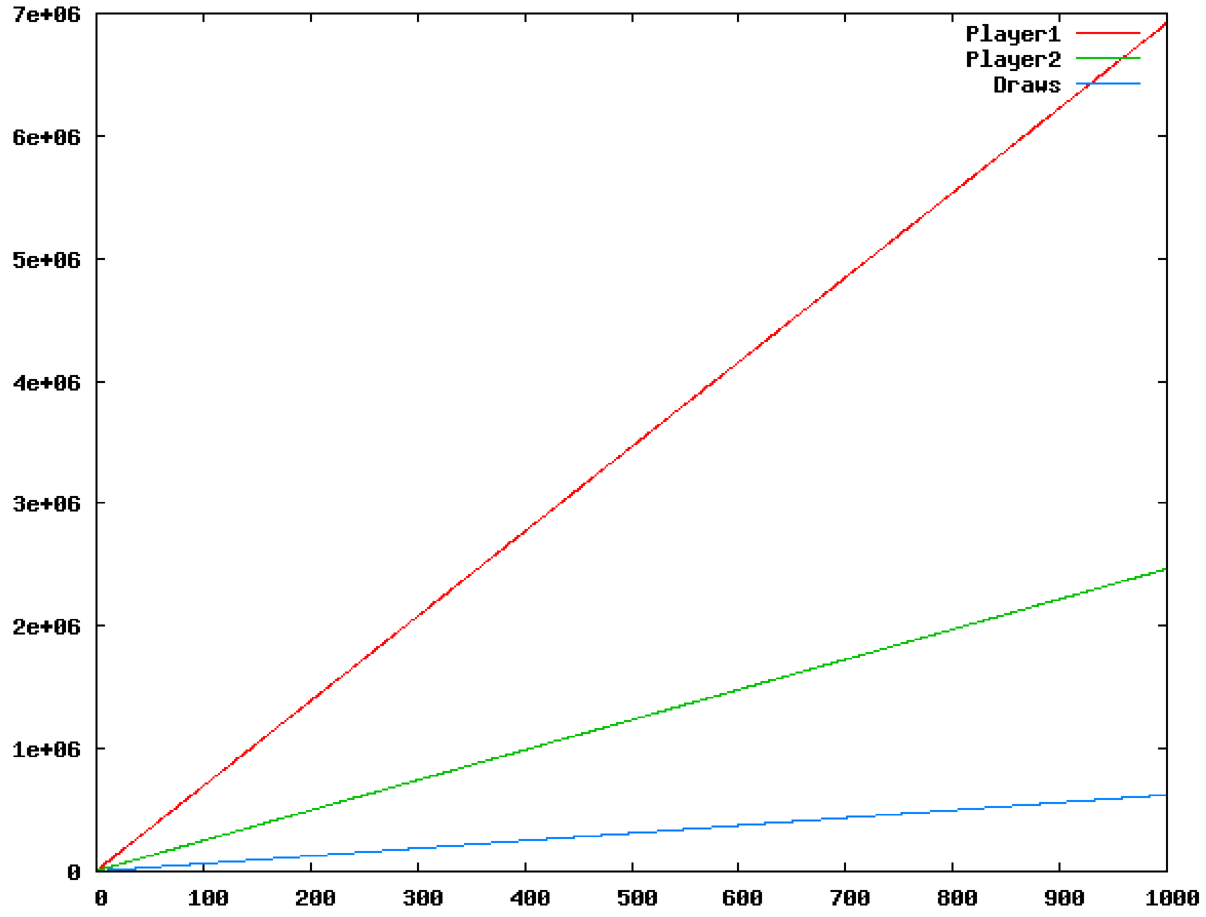
Image 1.0



Q-Learning Player vs Random Player

For the first test run of my regular Q-Learning Player I used a value of 0.8 for gamma. In this implementation I didn’t run the reward function after player 2’s moves. I ran this 10 million times taking the win ratio on 10,000 run intervals. I then plotted this data using Gnuplot. The results can be seen below in Image 1.1.

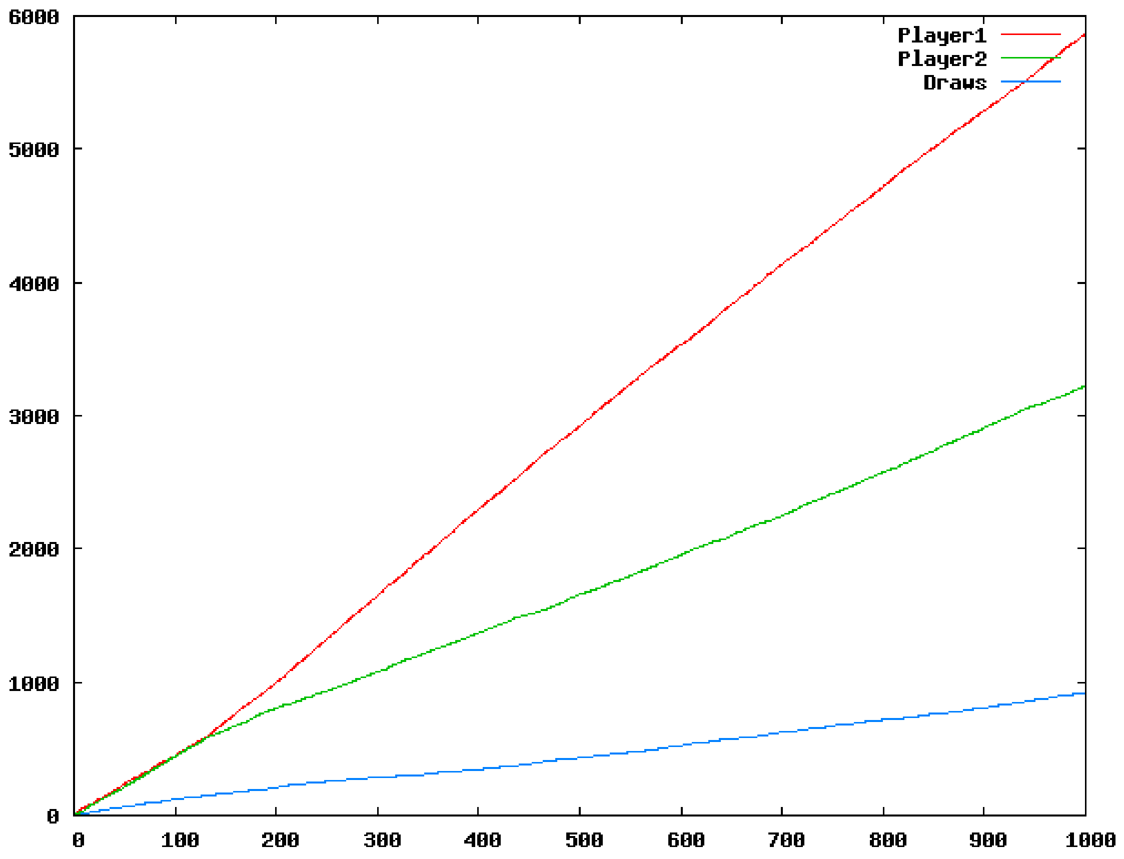
Image 1.1



We can see that there is a consistent rate of improvement to our Q-learning bot. The Q-learning Player is player 1, our random Player is player 2. If we change our gamma value I noticed there isn’t a huge effect on the outcome of the results. It just takes a little bit longer for the bot to start taking better moves.

Below (Image 1.2) you will find a microscopic snapshot of the first 10000 runs. I took a snapshot every 100 runs and plotted this data. I would have expected to find more of a curve but there is a point where the Q-player has enough data to start winning more and more frequently and we can see the results of Player 1 and Player 2 begin to separate.

Image 1.2



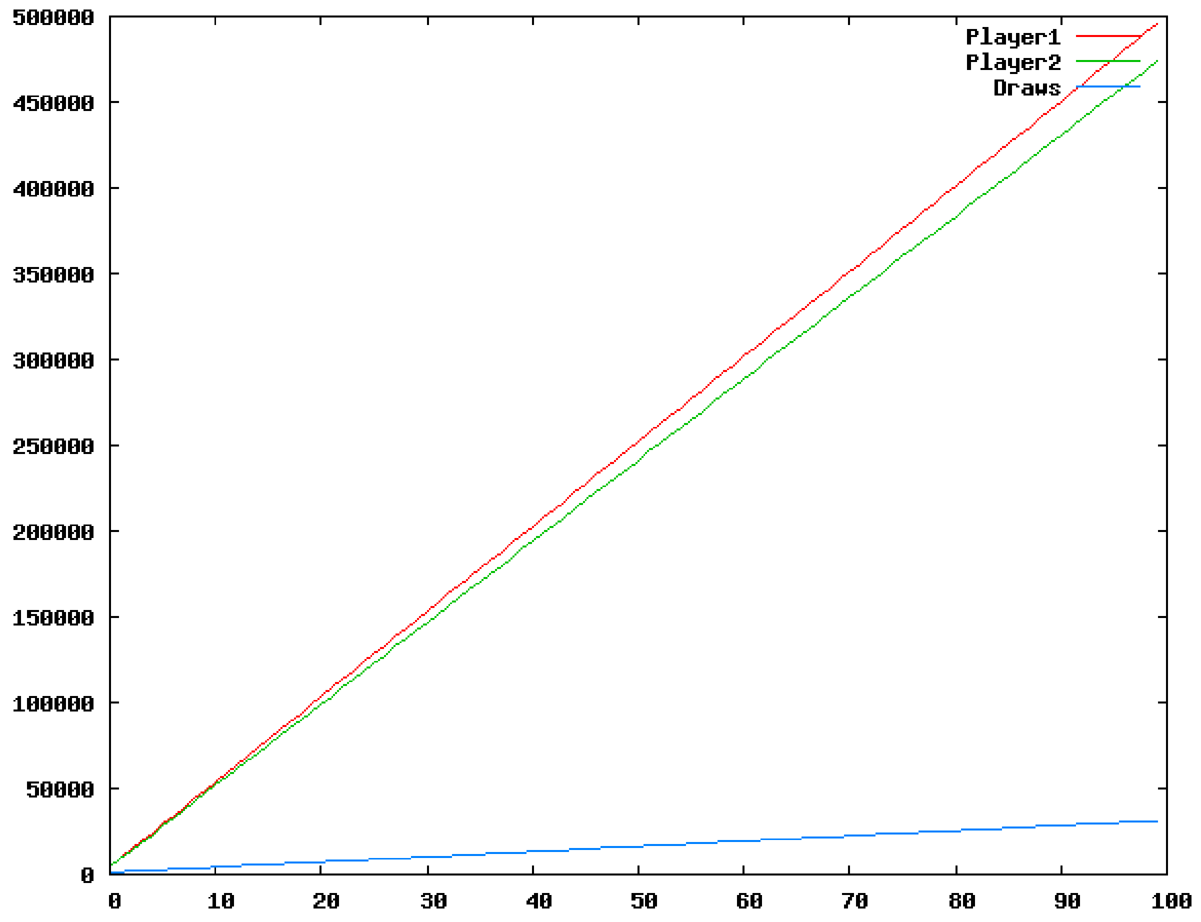
You can notice how at almost 2000 runs our bots are taking the equivalent of mostly random actions, then when our bot has enough data it starts to greatly improve.

Q-Learning Player vs Q-Learning Player

The data I generated for this was unexpected, I assumed both bots would learn to play against each other and this would eventually result in a lot more draws but this wasn't the case. This could possibly be an inconsistency in my implementation. Below (Image 1.3) you will find the outcome of my testing. I ran this test multiple times and what I found interesting is that either Player1 or 2 could take a slight lead.

You can see that after a point Player 1 has a slight lead that is then maintained. The same thing happened a few times with Player 2 on different test runs. It seems that once the Q-learning player takes a lead it maintains it, possibly it may have learned something that gives it an advantage over the other Q-Player.

Image 1.3



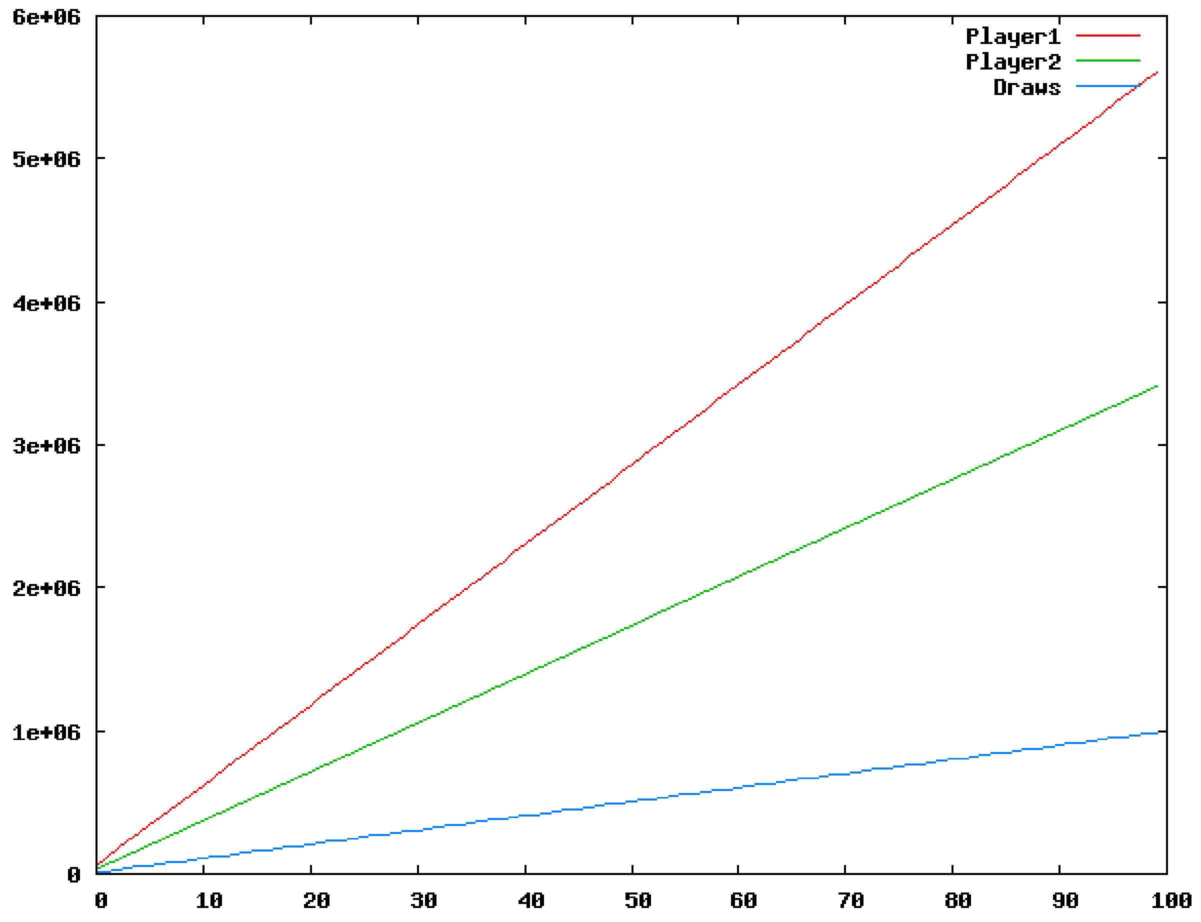
As I was getting some unexpected results I decided to modify my code and now my Q-learning algorithm updates the reward for every state, as opposed to only observing states based on the Players own moves. This resulted in me getting some different results so I reinvestigated.

Round2: Q-Learning Player vs Random Player

This time around my Q-learning algorithm updates itself on every state. This has resulted in different results. I modified my code so that I could look at different elements a bit clearer. I ran the same tests as before and did indeed get different data. I also added the ability to dump the Q-values in a json format. I would also like to implement a way of importing this data.

Below in the image 2.0 you will notice the Q-learning agent doesn't perform as well as in my recent testing. It does however demonstrate that the agent does beat the Random bot by a significant amount. This data below was 10 million runs, having a snapshot of the win ratio taken every 100,000 runs. We can see similarities in our old data when comparing the beginning stages. This Agent used a gamma value of 8.0 again.

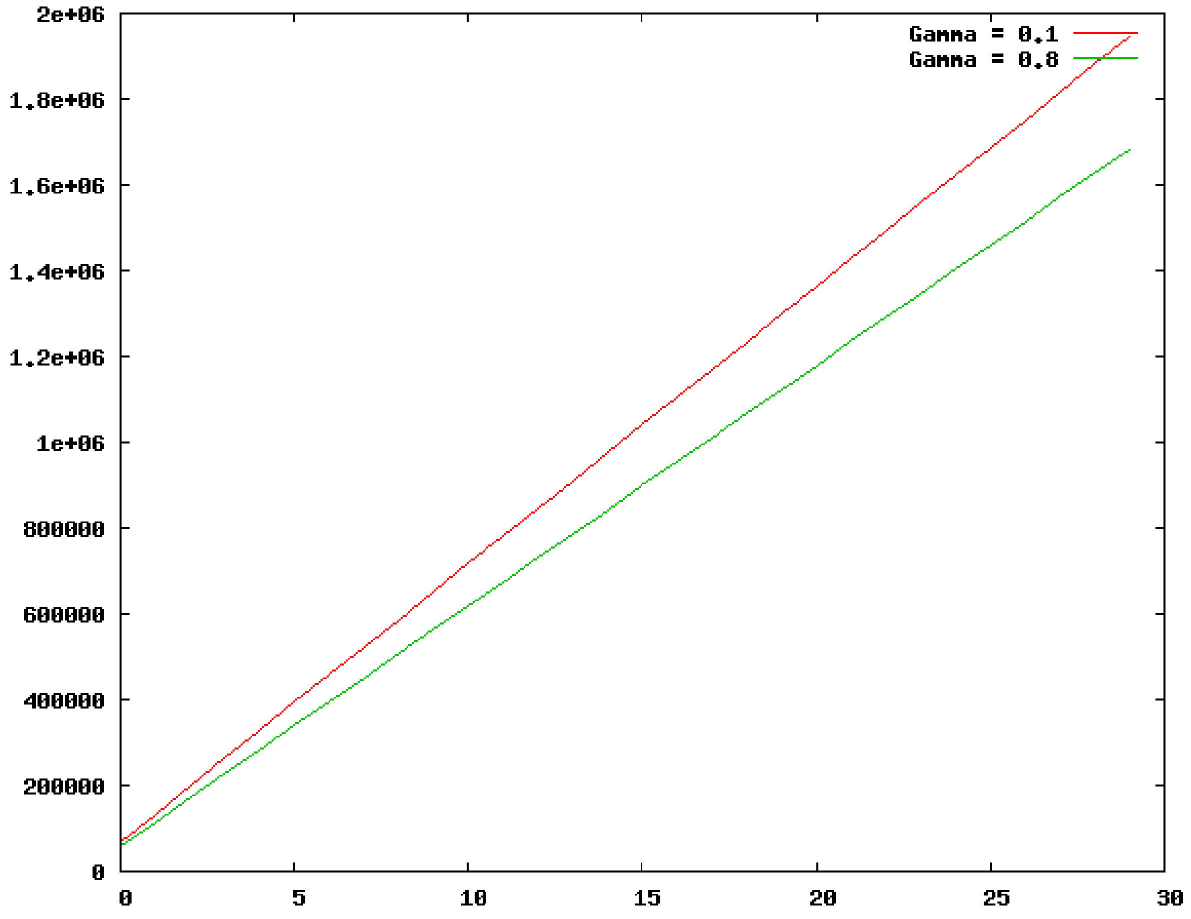
Image 2.0



Next I want to investigate what happens if I modify the discount factor, gamma. I ran the same tests as above but lowered gamma to 0.1. This had a very noticeable impact even in the early stages and our Q-learning agent performed much better. A lower discount award should make our agent aim for more short term targets, whereas a higher value will make it aim for a higher long term award. The results can be seen quite clearly below.

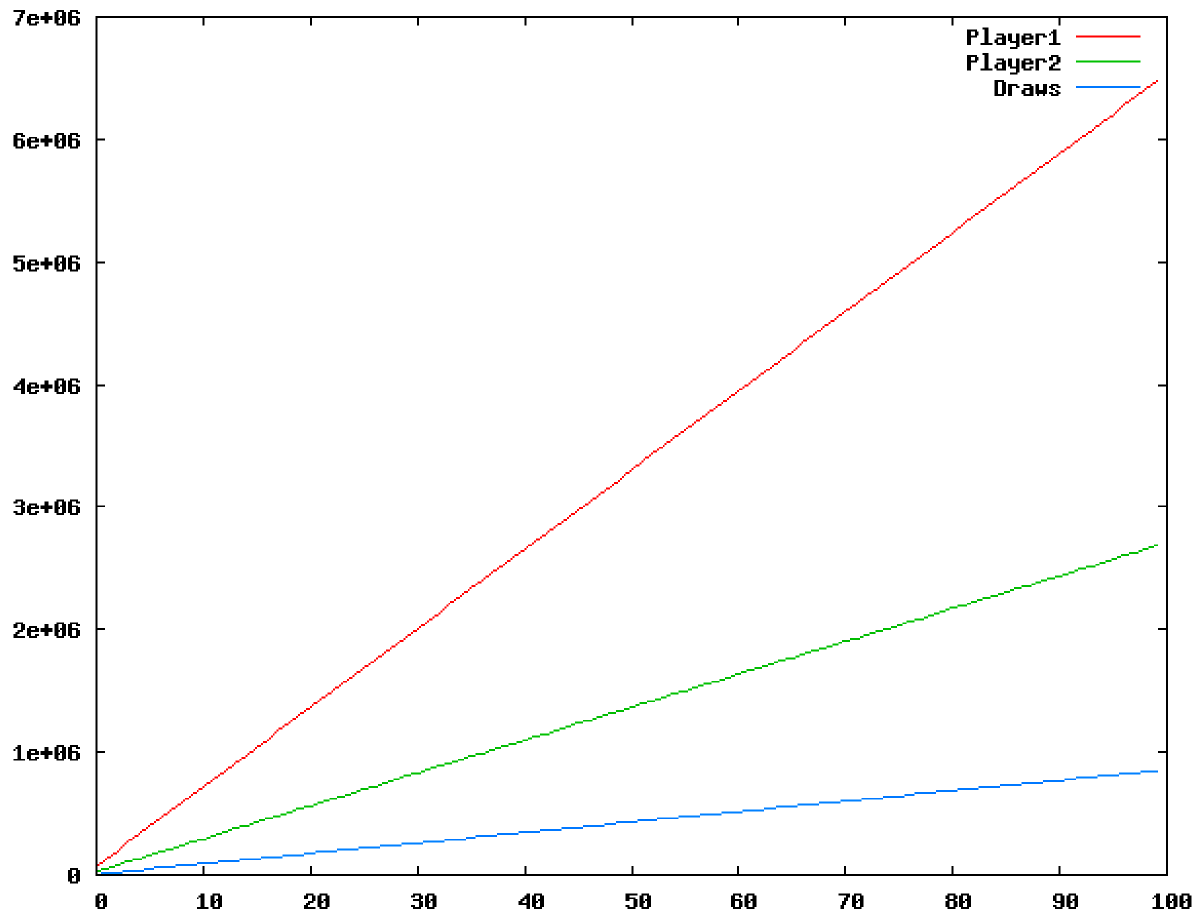
In Image 2.1 we can see that our short term reward Q-learning agent performs better from the offset. This image is a smaller snapshot of an overall trend. This was generated using 3 million runs with each different gamma setting, then I took a snapshot of the win ratio every 100,000 runs on interval.

Image 2.1



On the next page in image 2.2 you can see the overall results for our testing with a discount factor of 0.1. We can clearly see that this outperforms the results demonstrated in Image 2.0.

Image 2.2



I have successfully allowed the learning agent to store all of the Q-values in a json format. I can then continue from anywhere with these values and load them in on request. My only reason for not implementing the Running average element to this assignment was time constraints. It also might have been a little tricky with the way I have my code designed.

Q-Learning Data / Mind Data code - Q.java

```java
import java.util.*;
import org.json.simple.*;
import java.io.*;
public class Q{
    /*
     * This Stores state as a key, in String format
     * then double Q values for each action in state.
     */
    HashMap<String, double[]> qMap = new HashMap<String, double[]>();
    String mindImport = "";

    public void importMind(String mind){
        mindImport = mind;
        boolean isData = true;
        String mindData = "";
        try{
            mindData = strFromFile(mindImport);
        }
        catch(NoSuchElementException e){
            System.out.println("Nothing in file... no data imported!");
            isData = false;
        }
        if(isData){
            reuseMind(mindData);
        }
    }
    /*
     * Lookup a state and find the best action to take
     * if more than single action is the same
     * pick random from bunch
     */
    public int getBestAction(String state){
        double[] actions = new double[9];
        if(qMap.containsKey(state)){
            actions = qMap.get(state); //return all action values
        }
        double highestQ = -5000;
        ArrayList<Integer> next = new ArrayList<Integer>(); //possible choices
        int choice = 0;
        for(int i=0; i < actions.length; i++){
            if(actions[i] >= highestQ){
                highestQ = actions[i];
                choice = i;
            }
        }
        for(int i=0; i < actions.length; i++){
            if(actions[i] == actions[choice]){
                next.add(i);
            }
        }

        //return a random choice if more than one
        //or single choice if only one
        if(next.size()==1){
            return next.get(0);
        }
        else if(next.size() > 1){
            Random r = new Random();
            int random = r.nextInt(next.size());
            return next.get(random);
        }
        else{
            //System.out.println(next.size());
            Random r = new Random();
            int random = r.nextInt(9);
            return random;
        }
```

```
    }

    /*
     * Get Q value for particular state/action
     */
    public double getQ(String state, int action){
        double[] actions = new double[9];
        if(qMap.containsKey(state)){
            actions = qMap.get(state);
        }
        return actions[action];
    }


    /*
     * Update Q value for particular state/action with reward
     */
    public void update_Q(String state, int action, double reward){
        double[] actions = new double[9];
        if(qMap.containsKey(state)){
            actions = qMap.get(state);
        }
        actions[action] = reward;
        qMap.put(state, actions);
    }

    public void update_Q_reuse(String state, double[] actionsData){
        qMap.put(state, actionsData);
    }

    /*
        This populates our Hashmap with the Q values from a file
       */
       public void reuseMind(String data){
        Object obj2 = JSONValue.parse(data);
        JSONObject arr = (JSONObject) obj2;
        parseJson(arr);
    }

    /*
     * Parse our Json object to pull array and object info
     */
    public void parseJson(JSONObject arr){
        Set<Object> set = arr.keySet();
        Iterator<Object> iterator = set.iterator();
        double[] thisk = new double[9];
        String state = "";
        while (iterator.hasNext()){
            Object obj = iterator.next();
            if (arr.get(obj) instanceof JSONArray){
                //State and actions into Q
                state = obj.toString();
                thisk = getArray(arr.get(obj));
                update_Q_reuse(state, thisk);
            }
            else{
                if (arr.get(obj) instanceof JSONObject) {
                    parseJson((JSONObject) arr.get(obj));
                }
                else{
                    System.out.println(obj.toString());
                }
            }
        }
    }

   /*
    * Parse a JSONArray into a double array
```

```java
     */
    public double[] getArray(Object obj){
     JSONArray jsonArr = (JSONArray) obj;
     double[] vals = new double[9];
     for (int i=0; i < jsonArr.size(); i++) {
         vals[i] = (double)jsonArr.get(i);
     }
     return vals;

     //Print out board - Debugging purposes...
     public void printState(String state){
         char[] sp = state.toCharArray();
         System.out.println(sp[0]+" "+sp[1]+" "+sp[2]);
         System.out.println(sp[3]+" "+sp[4]+" "+sp[5]);
         System.out.println(sp[6]+" "+sp[7]+" "+sp[8]+"\n");
     }

     public void dump(){
         Set<String> enumk = qMap.keySet();
         Iterator<String> iter = enumk.iterator();
         while(iter.hasNext()) {
             String key = iter.next();
             double[] val = qMap.get(key);
             System.out.print(key+":  ");
             for(int i=0; i<val.length; i++){
                 System.out.print("a"+i+":"+val[i]+",");
             }
             System.out.println();
         }
     }

     public void dumpJSON(){
         JSONObject obj = new JSONObject();
         JSONArray arr = new JSONArray();
```

```java
        double[] val = new double[9];
        Set<Map.Entry<String, double[]>> set = qMap.entrySet();
        String key = "";
        for (Map.Entry<String, double[]> me : set) {
            key = me.getKey();
            val = me.getValue();
            for(int i=0; i<val.length; i++){
                arr.add(val[i]);
            }
            obj.put(key, arr);
            arr = new JSONArray();
        }

        String jsonString = JSONValue.toJSONString(obj);
        strToFile(mindImport, jsonString);
    }

   //write string data to a file
    public void strToFile(String filename, String data){
        File file = new File(filename);
        try{
            FileWriter dataf = new FileWriter(file);
            dataf.write(data);
            dataf.close();
        }
        catch(FileNotFoundException e){
            System.out.println("\nFile not found!");
        }
        catch(IOException e){
            System.out.println("\nIO Error!");
        }
    }

    //Reads string from a file
    public String strFromFile(String filename){
        String mydata = "";
        File file = new File(filename);
        try{
            Scanner data = new Scanner(file);
            mydata = data.nextLine();
            data.close();
        }
        catch(FileNotFoundException e){
            System.out.println("\nFile not found!");
        }
        return mydata;
    }
}
```

Q-Learning Player code - Player.java

```
import java.util.Random;

public class Player{
    int currentMove;
    String currentState;
    int lastMove;
    String lastState;
    Q q = new Q();

    Player(String mind){
        q.importMind(mind);
    }
    public void nextAction(String state){
        currentMove = q.getBestAction(state);
        currentState = state;
    }

    public void storeInLast(){
```

```java
        lastMove = currentMove;
        lastState = currentState;
    }

    //Q(y,b)
    public double retQNextAction(String state){
        return q.getQ(state, q.getBestAction(state));
    }

    public double retQ(String state, int action){
        return q.getQ(state, action);
    }

    public void rewards(String state, int action, double reward){
        double oldQ = retQ(state, action);
        double newQ = oldQ + reward;
        //System.out.println("State: "+state+", Action: "+action+", Value: "+newQ);
        q.update_Q(state, action, newQ);
    }

    public void directReward(String state, int action, double reward){
        double oldQ = retQ(state, action);
        q.update_Q(state, action, oldQ+reward);
    }

    public void dump(){
       q.dump();
    }

    public void dumpJSON(){
        q.dumpJSON();
    }

}
```

Running the game/tests.

I have wrote a simple bash script that can be used for compiling and running my code called Run.sh. I am using a JSON parsing library from the following location.
https://code.google.com/p/json-simple/
Simply include this in the directory to run/compile the code.
https://code.google.com/p/json-simple/downloads/detail?name=json-simple-1.1.1.jar&can=2&q=

Run.sh Script

```
#!/bin/bash
rm *.class
javac -cp "./json-simple-1.1.1.jar:." RunStore2.java
java -cp "./json-simple-1.1.1.jar:." RunStore2 $*
```

The RunStore2.java file contains the most recent version of my code. It simply accepts an int as a command line parameter. This is the amount of times you'd like to run the code.
For example (./Run.sh 1000) would run the code 1000 times and print the win/lose rations for the players. It would also by defaults store the Q values to the QData.json file. This file must be located in the run directory (it can be empty.)

Game running code - RunStore2.java

```
//Running our game
public class RunStore2{
    public static int countRuns = 0;
    public static Player p1 = new Player("QData.json");
    public static RPlayer p2 = new RPlayer();
    public static void main(String[] args){
        Game game = new Game();
        int[] results = new int[4];

        for(int i=0; i<Integer.parseInt(args[0]); i++){
            results[run(game)]++;
            countRuns += 1;
            if(countRuns % 100000==0){
                System.out.println("1:"+results[1]+", 2:"+results[2]+",
```

```java
draw:"+results[3]);
            }
        }
        for(int i=1; i<results.length; i++){
            if(i==3){
                System.out.println("Draws: "+results[i]);
            }
            else{
                System.out.println("Player "+i+": "+results[i]);
            }
        }
        p1.dumpJSON();
        //System.out.println("-------");
        //p1.dump();
    }

    public static int run(Game game){
        double reward = 0.0;
        double yb = 0.0;
        double gamma = 0.5;
        String nextState = "";
        String x = "";
        int a = -1;
        String y = "";
        int b = -1;
        game.newBoard();
        while(game.whoWins() <= 0){
            if(game.turn == 1){
                //Fill in our current move & state
                x = game.getState();
                p1.nextAction(game.getState());
                while(!game.isValidAction(1,p1.currentMove)){
                    p1.nextAction(game.getState());
                }
                a = p1.currentMove;
                y = game.getNextState(1, p1.currentMove);
                yb = p1.retQNextAction(y);

                game.makeMove(1, p1.currentMove);
            }

            //2nd player takes move
            else{
                x = game.getState();
                p2.nextAction(game.getState());
                while(!game.isValidAction(2, p2.currentMove)){
                    p2.nextAction(game.getState());
                }
                a = p2.currentMove;
                y = game.getNextState(2, p2.currentMove);
                yb = p1.retQNextAction(y);
                game.makeMove(2, p2.currentMove);
            }

            reward = gamma * yb;
            p1.rewards(x, a, reward);
        }
        if(game.whoWins() == 1){
            reward = 100.0;
            p1.directReward(game.getState(),a,reward);
            reward = 0.0;
        }
        else if(game.whoWins() == 2){
            reward = -100.0;
            p1.directReward(game.getState(),a,reward);
            reward = 0.0;
        }
        return game.whoWins();
```

```
    }

}
```