Randomness on Misere Nim

Ramsey Alsheikh

Abstract

Misere Nim is a game where 2 players take turns taking at least one unit in exactly one group from groups of 1, 3, 5, and 7 units, where the player who takes the last unit loses. The purpose of this experiment was to test the effect of random moves on a Misere Nim bot's chances of winning in a computerized game of Misere Nim. The hypothesis was if 10,000 matches of computerized misere Nim are played, a no strategy bot going second would win at least 2 times against a strategy bot going first. A program ran a bot that played strategically against a bot that played randomly in misere Nim 10,000 times. This program ran three times, and outputted a text file with the results, which was recorded. It also played Misere Nim 10,000 times against two randomly-playing bots. This round also ran three times and outputted a text file with the results, which was recorded. The game with a strategy bot resulted in the strategy bot winning on average 9999 2/3 times, and the no strategy bot winning an average of 1/3 times. The game with two no strategy bots resulted in bot 1 winning on average 5010 times. The hypothesis wasn't supported, and the hypothesis was a severe underestimation of how many times the strategy bot would win. These results can be applied to game theory and strategic analysis, by illustrating the extent to which a strategy helps to win games.

Introduction

Statement of Purpose

The purpose of this experiment is to test the effect of random moves on a Misere Nim bot's chances of winning in a computerized game of Misere Nim. 10,000 matches of a Misere Nim will be played by two bots - one which makes perfect moves (going first) and one who makes random moves (going second). In Misere Nim, whoever goes second can always win if they make perfect moves, and the first player can only win if their opponent makes a mistake. The dependent variable is how many times the random bot wins. The independent variable is how many games has been played beforehand. If the random bot wins more matches than the best move bot, that would suggest randomness can produce desirable solutions under the right circumstances. Constant variables include the computer the program is running on (Macbook Air 13-inch 2017), the version of Java being used (11.0.2), and the operating system being used (MacOS Mojave 10.14.3). The test subject is the Macbook Air.

Background Research

Misere Nim is a version of the game Nim, whose rules are as follows. One pile each of 1 unit, 3 units, 5 units, and 7 units are set up at the start of the game. The first player may take as many units as he/she would like from exactly one row. The player who is forced to take the last unit loses. Artificial Intelligence is a form of computing in where a computer tries to simulate natural intelligence and interact with the real world. (Haaxma, 2014). In this experiment, an AI is needed for the strategy bot to follow an algorithm and react to its environment. An AI is not needed for the randomly playing bot, as it will be playing 'randomly'. However, it's random moves are really pseudo random (almost everything in the non-quantum word is). In order to make the bot play truly randomly, quantum mechanics would have to be used, which is beyond the scope of this experiment (AcÍn, 2016). Quantum mechanics has been used in the past to make truly random numbers, at the National Institute of Standards and Technology in the past (NIST's new quantum, 2018). In order to make the bots, they will be coded with a programming language, which is a semi-natural and intuitive set of keywords used by programmes to make logical instructions for the computer (World of Computer Science, 2007). The specific programming language being used is Java, a general purpose programming language that allows programmers to write code that can be run almost anywhere, and is widely used in programming. (The Hutchinson Encyclopedia, 2018). This way, multiple scientists can test the Java program written in the experiment on their own computers, fostering replication. The version of Nim being played was invented by Charles Leonard Boutan in 1901 (Rougetet, 2014). That same year, he also release an algorithmic solution to his game. (Rougetet, 2014) A computer that can play Nim perfectly has been made before in the 1939 New York World Fair (Tesla, 2017). The law of truly large numbers states that when interactions are repeated over and over, unlikely events are bound to happen. (Skeptic, 2014). The program will be run on a Macbook, a laptop version of the original Macintosh from 1984. (The Gazette, 2011). It uses the operating system Mac OS X, which was released in 2000, the year Steve Jobs became the CEO of Apple. (The Gazette, 2011).

Hypothesis

If 10,000 matches of computerized misere Nim are played, a no strategy bot going second will two times against a bot that plays perfect strategy. This was number was calculated as follows. Each outcome in Misere Nim has a likelihood of happening. In Misere Nim using our rules, there are 162,032 theoretically possible outcomes, 5,412 of which result in the random player winning. The sum of the probabilities of the random player winning outcomes is 3,391. The sum of the probabilities of the strategic player winning outcomes is 9,459,440. 3,391 divided by 9,459, 440 is around 0.0002, or 0.02 percent. 0.0002 multiplied by 10,000 is 2, therefore, in theory, the random player should win twice. This experiment will test whether that actually happens.

Method

Materials

- Macbook Air 2017 (or any computer with Linux, macOS X, or Windows installed that is capable of running Java 11)
- Java Development Kit 11
- Charging Cable
- Source code, located at https://github.com/DatOneRam/ScienceFair

Procedure

1. Ensure the computer has at least a ten percent charge. If it doesn't, charge it using the charging cable
2. Download the source code
3. Run Test.class using the Terminal/Command Prompt command "java Test" when in the program's folder
4. Wait for program to finish running
5. Open the text file using any text editor
6. Record results

Results

	Number Of Wins in Trial 1	Number Of Wins in Trial 2	Number Of Wins in Trial 3	Averages
Strategy Bot	10,000	9,999	10,000	9,999 2/3
Random Bot	0	1	0	1/3

Table 1. Experimental Group Trial Results

	Number of Wins in Trial 1	Number of Wins in Trial 2	Number of Wins in Trial 3	Averages
Bot One	5,011	5,000	5,019	5,010
Bot Two	4,989	5,000	4,981	4,990

Table 2. Control Group Trial Results

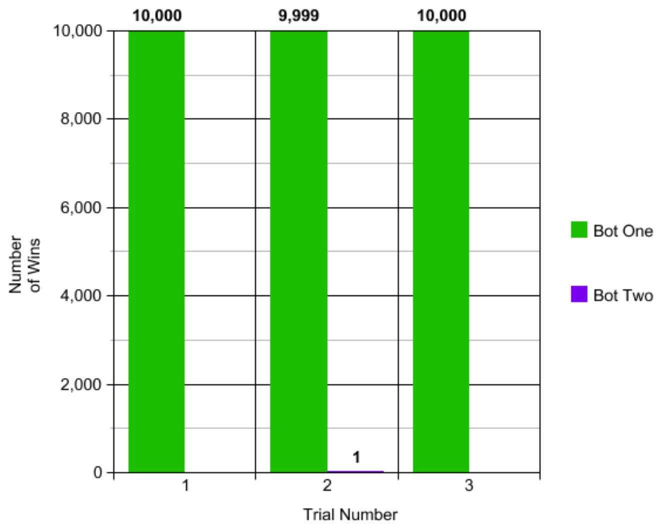


Figure 1. Win Distribution Among the Experimental Group

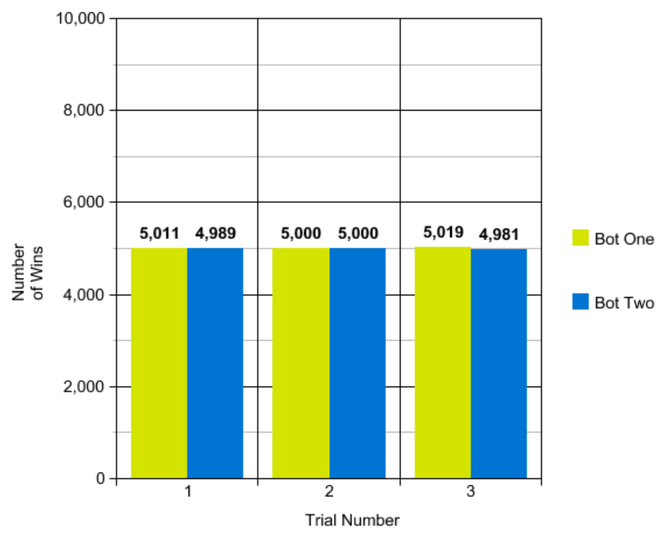


Figure 2. Win Distribution Among the Control Group 3

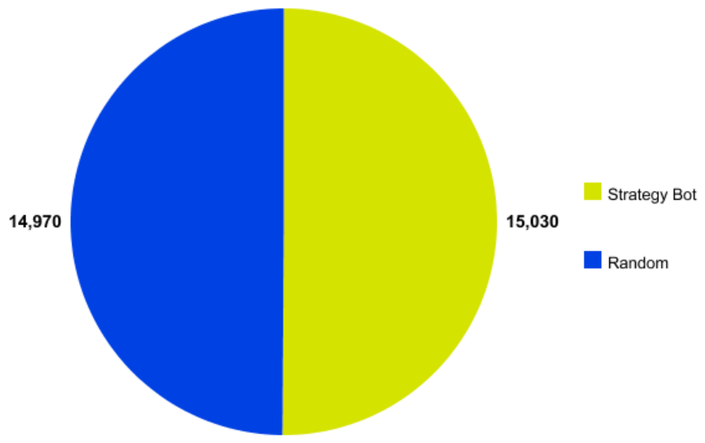
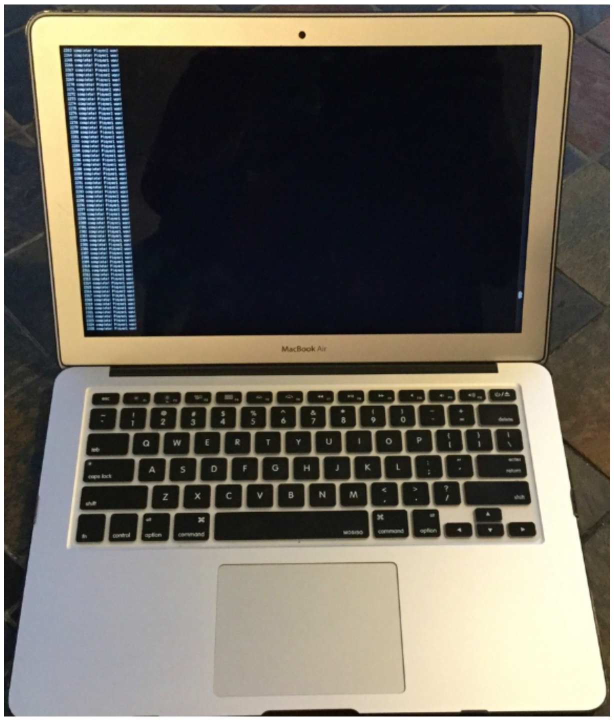
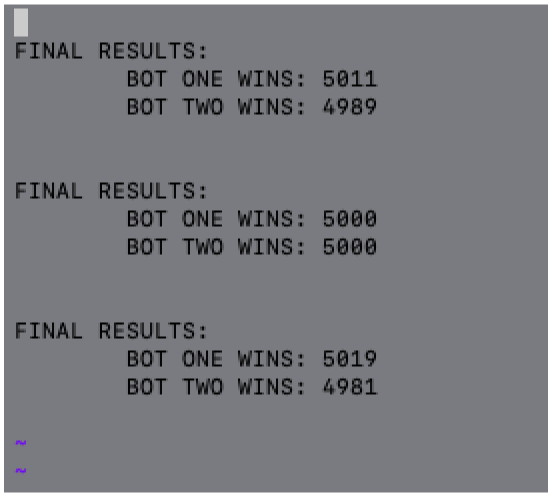


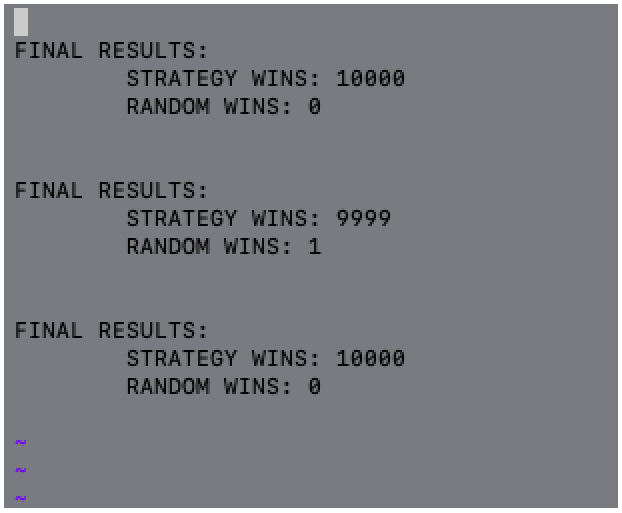
Figure 4. Control All Trials Results


Photograph 1. Client Computer 3/14/19



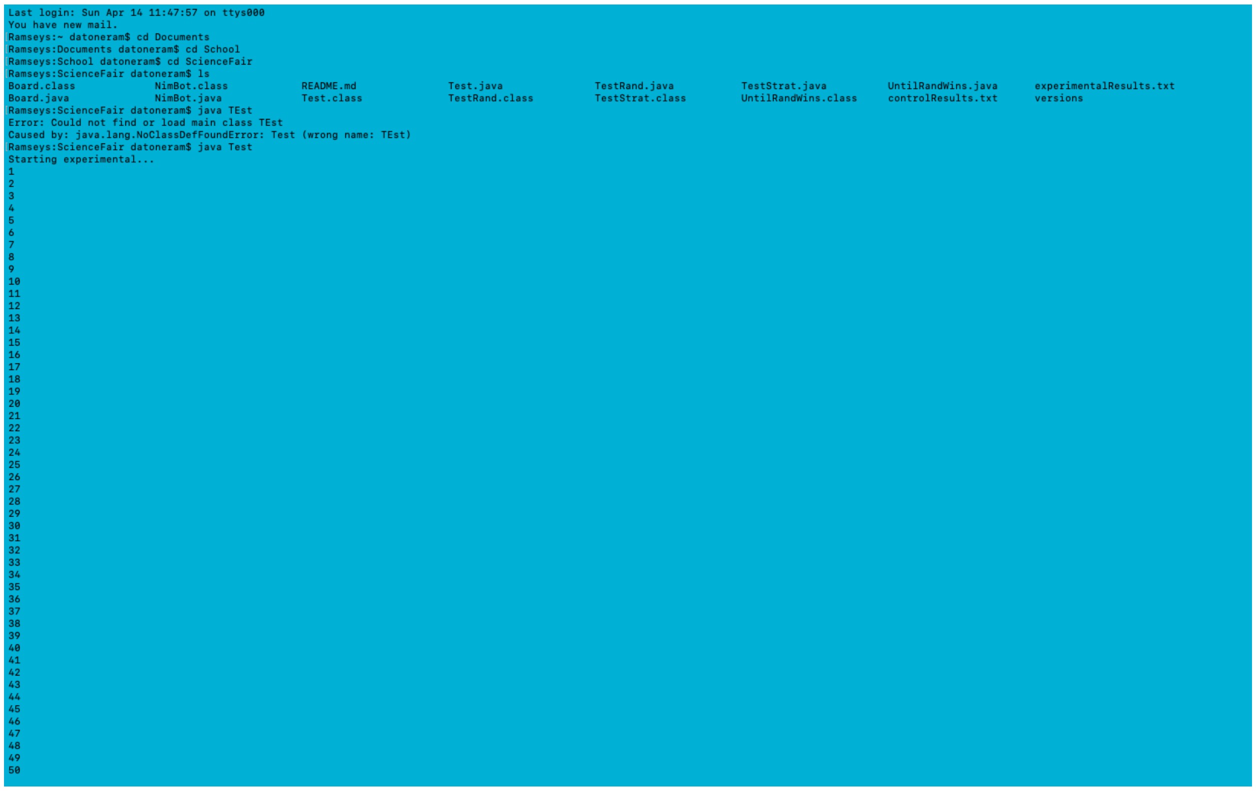
```
FINAL RESULTS:
        BOT ONE WINS: 5011
        BOT TWO WINS: 4989


FINAL RESULTS:
        BOT ONE WINS: 5000
        BOT TWO WINS: 5000


FINAL RESULTS:
        BOT ONE WINS: 5019
        BOT TWO WINS: 4981

~
~
```

Photograph 2. Control Results Text File 4/14/19



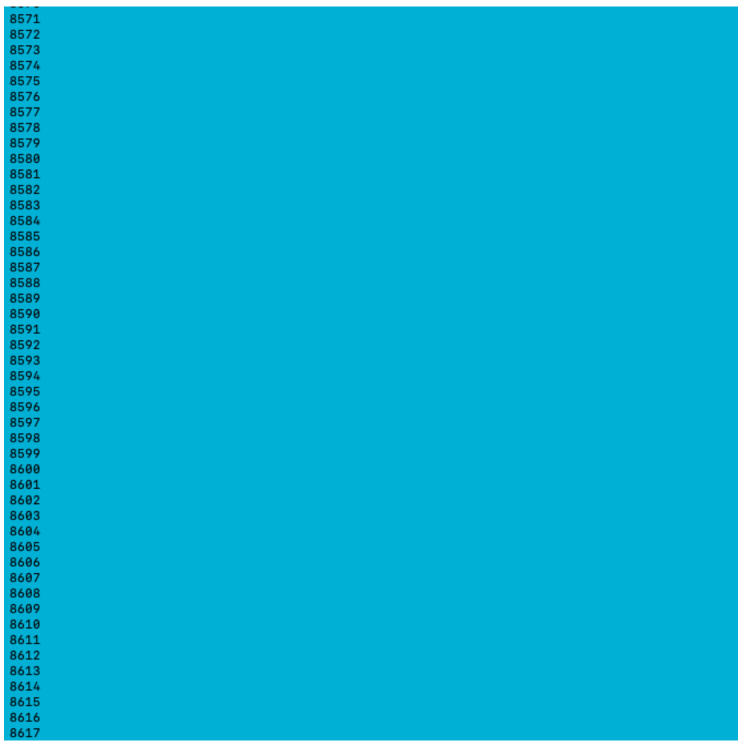
```
FINAL RESULTS:
	STRATEGY WINS: 10000
	RANDOM WINS: 0


FINAL RESULTS:
	STRATEGY WINS: 9999
	RANDOM WINS: 1


FINAL RESULTS:
	STRATEGY WINS: 10000
	RANDOM WINS: 0

~
~
~
```

Photograph 3. Experimental Results Text File 4/4/19



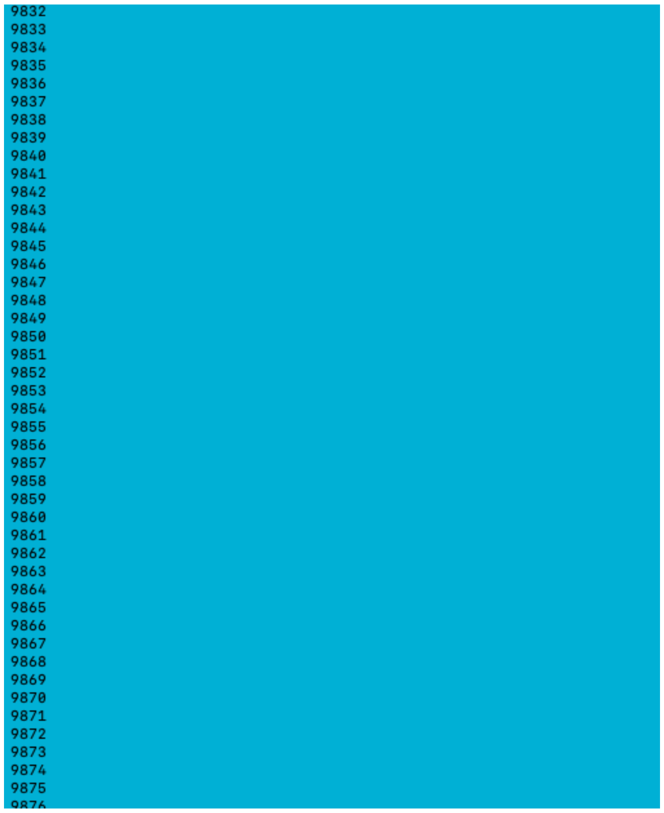
```
Last login: Sun Apr 14 11:47:57 on ttys000
You have new mail.
Ramseys:~ datoneram$ cd Documents
Ramseys:Documents datoneram$ cd School
Ramseys:School datoneram$ cd ScienceFair
Ramseys:ScienceFair datoneram$ ls
Board.class          NimBot.class         README.md            Test.java            TestRand.java        TestStrat.java       UntilRandWins.java   experimentalResults.txt
Board.java           NimBot.java          Test.class           TestRand.class       TestStrat.class      UntilRandWins.class  controlResults.txt   versions
Ramseys:ScienceFair datoneram$ java TEst
Error: Could not find or load main class TEst
Caused by: java.lang.NoClassDefFoundError: Test (wrong name: TEst)
Ramseys:ScienceFair datoneram$ java Test
Starting experimental...
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
```

Photograph 4. Commands to Run the Experimental and Control Groups 4/4/19



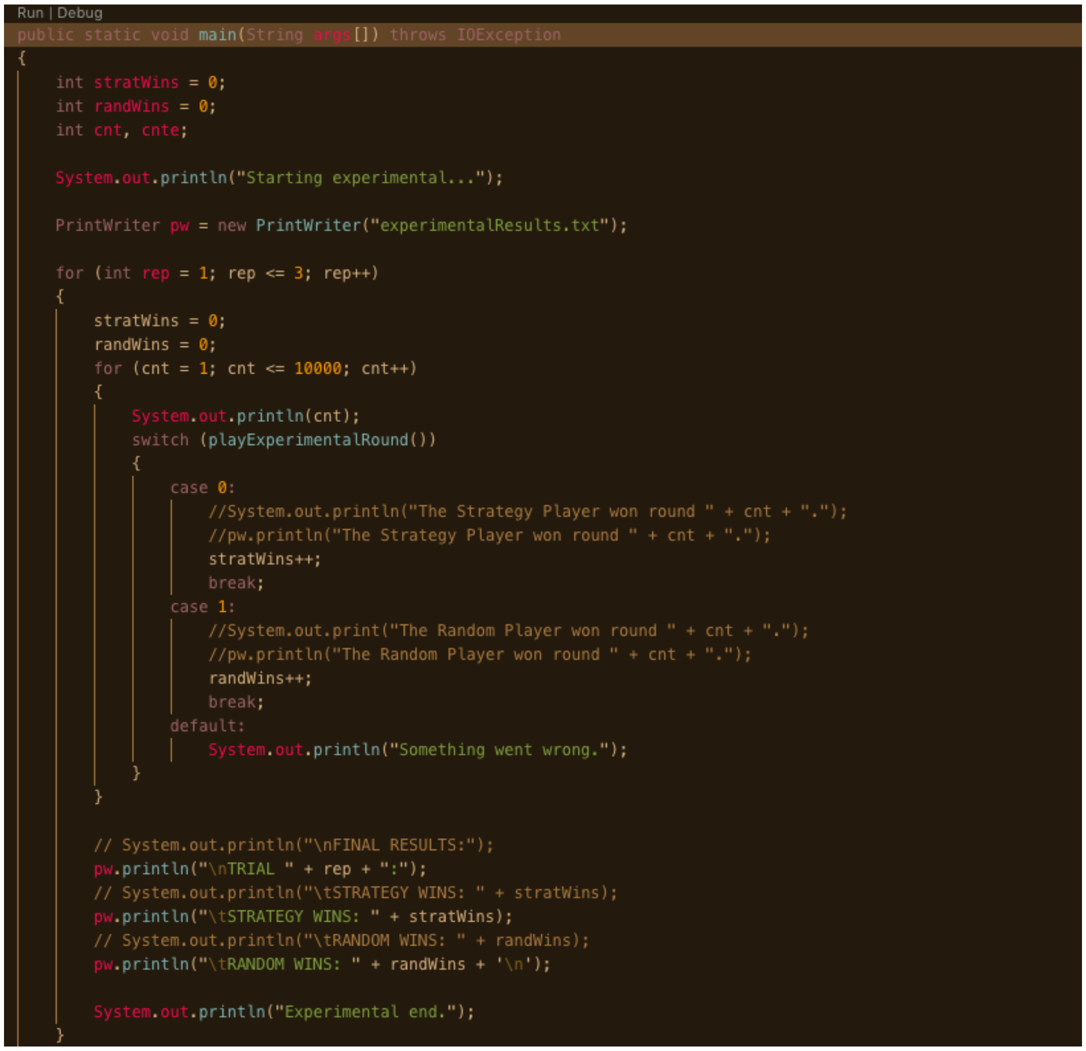
```
8571
8572
8573
8574
8575
8576
8577
8578
8579
8580
8581
8582
8583
8584
8585
8586
8587
8588
8589
8590
8591
8592
8593
8594
8595
8596
8597
8598
8599
8600
8601
8602
8603
8604
8605
8606
8607
8608
8609
8610
8611
8612
8613
8614
8615
8616
8617
```

Photograph 5. Control Group Running 4/4/19



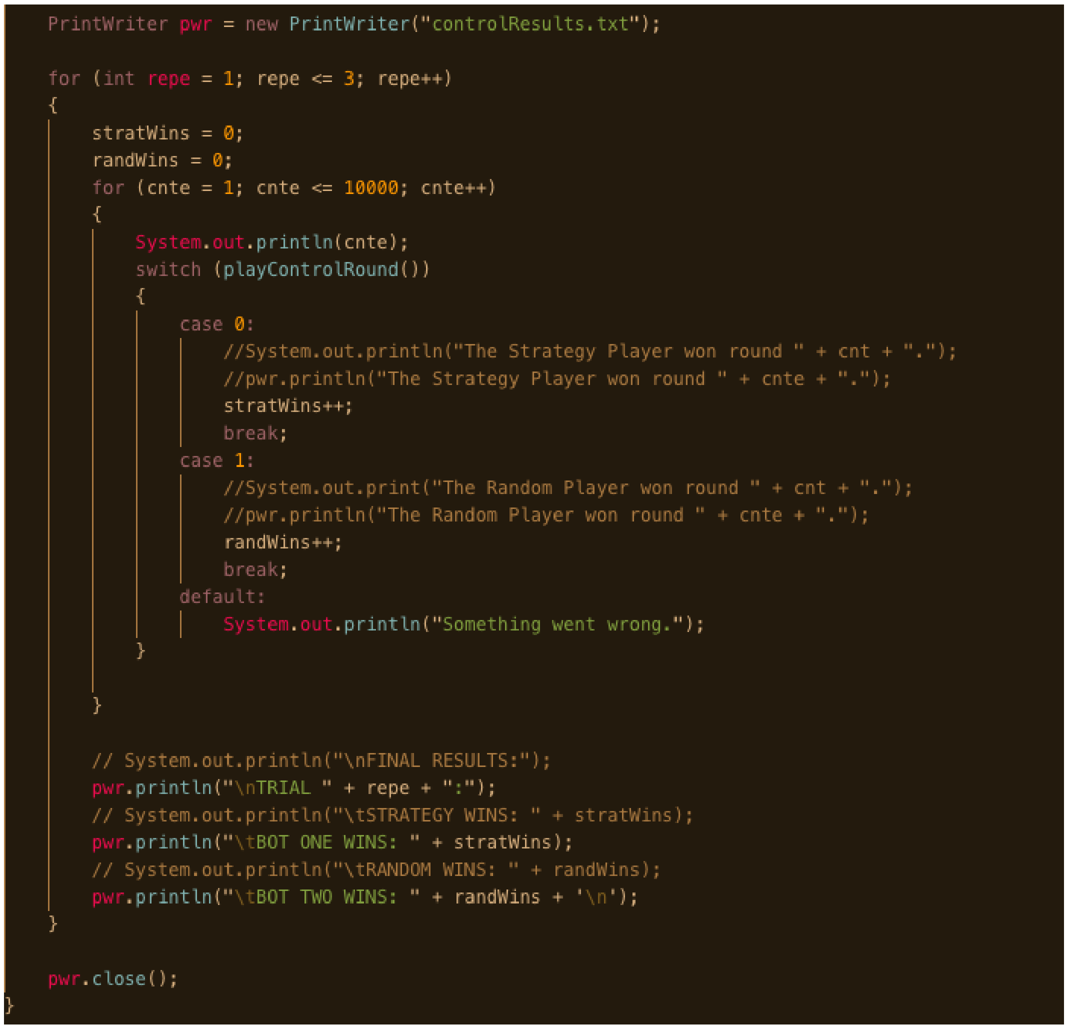
```
9832
9833
9834
9835
9836
9837
9838
9839
9840
9841
9842
9843
9844
9845
9846
9847
9848
9849
9850
9851
9852
9853
9854
9855
9856
9857
9858
9859
9860
9861
9862
9863
9864
9865
9866
9867
9868
9869
9870
9871
9872
9873
9874
9875
9876
```

Photograph 6. Experimental Group Running 4/4/19



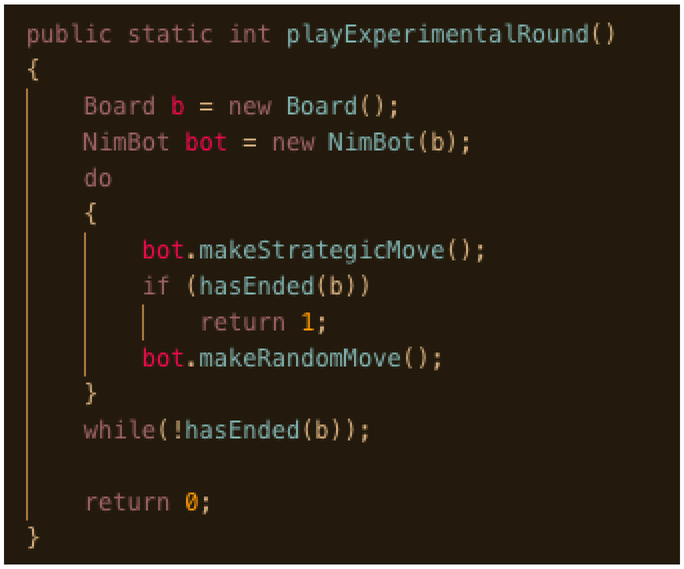
```
Run | Debug
public static void main(String args[]) throws IOException
{
    int stratWins = 0;
    int randWins = 0;
    int cnt, cnte;

    System.out.println("Starting experimental...");

    PrintWriter pw = new PrintWriter("experimentalResults.txt");

    for (int rep = 1; rep <= 3; rep++)
    {
        stratWins = 0;
        randWins = 0;
        for (cnt = 1; cnt <= 10000; cnt++)
        {
            System.out.println(cnt);
            switch (playExperimentalRound())
            {
                case 0:
                    //System.out.println("The Strategy Player won round " + cnt + ".");
                    //pw.println("The Strategy Player won round " + cnt + ".");
                    stratWins++;
                    break;
                case 1:
                    //System.out.print("The Random Player won round " + cnt + ".");
                    //pw.println("The Random Player won round " + cnt + ".");
                    randWins++;
                    break;
                default:
                    System.out.println("Something went wrong.");
            }
        }

        // System.out.println("\nFINAL RESULTS:");
        pw.println("\nTRIAL " + rep + ":");
        // System.out.println("\tSTRATEGY WINS: " + stratWins);
        pw.println("\tSTRATEGY WINS: " + stratWins);
        // System.out.println("\tRANDOM WINS: " + randWins);
        pw.println("\tRANDOM WINS: " + randWins + '\n');

        System.out.println("Experimental end.");
    }
```

Photograph 7. Experimental Group Code 4/4/19



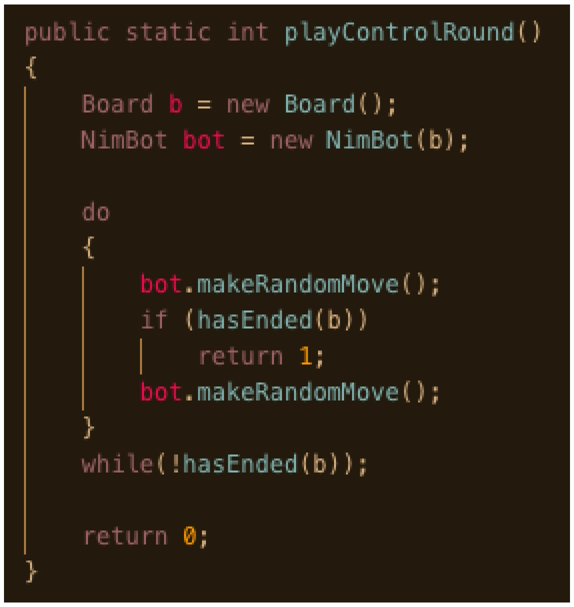
```java
    PrintWriter pwr = new PrintWriter("controlResults.txt");

    for (int repe = 1; repe <= 3; repe++)
    {
        stratWins = 0;
        randWins = 0;
        for (cnte = 1; cnte <= 10000; cnte++)
        {
            System.out.println(cnte);
            switch (playControlRound())
            {
                case 0:
                    //System.out.println("The Strategy Player won round " + cnt + ".");
                    //pwr.println("The Strategy Player won round " + cnte + ".");
                    stratWins++;
                    break;
                case 1:
                    //System.out.print("The Random Player won round " + cnt + ".");
                    //pwr.println("The Random Player won round " + cnte + ".");
                    randWins++;
                    break;
                default:
                    System.out.println("Something went wrong.");
            }

        }

        // System.out.println("\nFINAL RESULTS:");
        pwr.println("\nTRIAL " + repe + ":");
        // System.out.println("\tSTRATEGY WINS: " + stratWins);
        pwr.println("\tBOT ONE WINS: " + stratWins);
        // System.out.println("\tRANDOM WINS: " + randWins);
        pwr.println("\tBOT TWO WINS: " + randWins + '\n');
    }

    pwr.close();
}
```

Photograph 8. Control Group Code 4/4/19



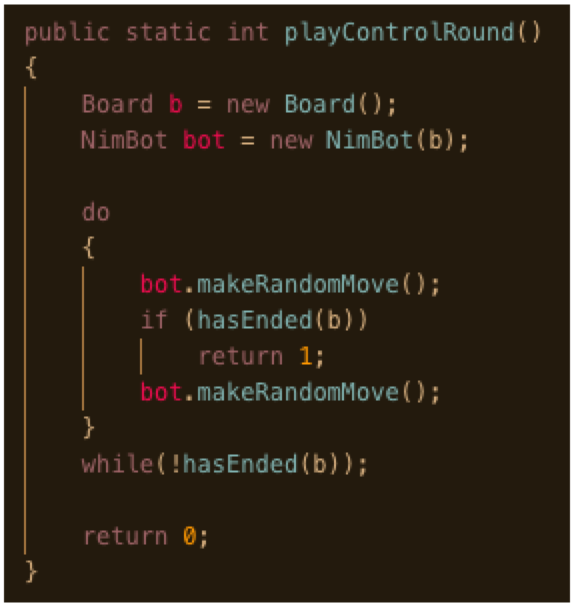
```java
public static int playExperimentalRound()
{
    Board b = new Board();
    NimBot bot = new NimBot(b);
    do
    {
        bot.makeStrategicMove();
        if (hasEnded(b))
            return 1;
        bot.makeRandomMove();
    }
    while(!hasEnded(b));

    return 0;
}
```

Photograph 9. Experimental Round Code 4/4/19



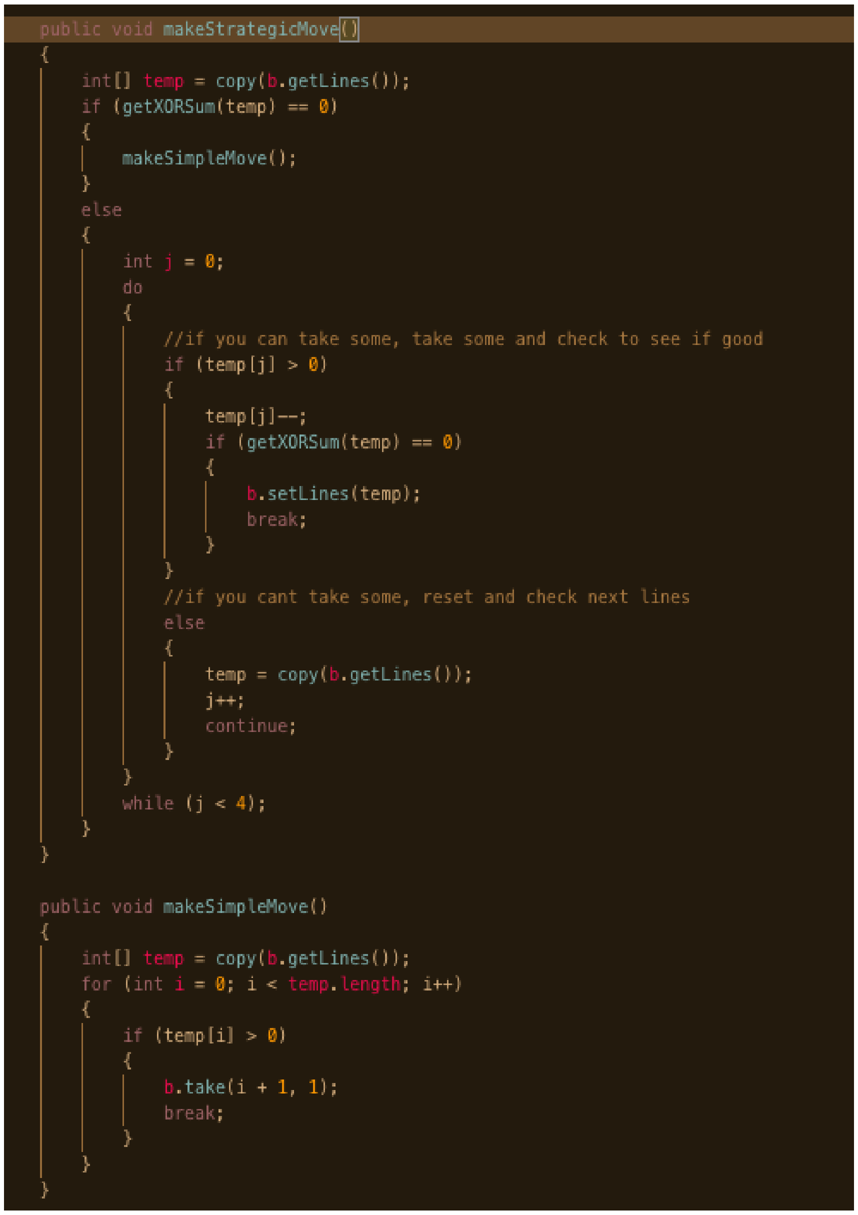
```java
public static int playControlRound()
{
    Board b = new Board();
    NimBot bot = new NimBot(b);

    do
    {
        bot.makeRandomMove();
        if (hasEnded(b))
            return 1;
        bot.makeRandomMove();
    }
    while(!hasEnded(b));

    return 0;
}
```

Photograph 10. Control Group Code 4/4/19



```java
public static int playControlRound()
{
    Board b = new Board();
    NimBot bot = new NimBot(b);

    do
    {
        bot.makeRandomMove();
        if (hasEnded(b))
            return 1;
        bot.makeRandomMove();
    }
    while(!hasEnded(b));

    return 0;
}
```

Photograph 11. Code to Make a Random Move 4/4/19



```
public void makeStrategicMove()
{
    int[] temp = copy(b.getLines());
    if (getXORSum(temp) == 0)
    {
        makeSimpleMove();
    }
    else
    {
        int j = 0;
        do
        {
            //if you can take some, take some and check to see if good
            if (temp[j] > 0)
            {
                temp[j]--;
                if (getXORSum(temp) == 0)
                {
                    b.setLines(temp);
                    break;
                }
            }
            //if you cant take some, reset and check next lines
            else
            {
                temp = copy(b.getLines());
                j++;
                continue;
            }
        }
        while (j < 4);
    }
}

public void makeSimpleMove()
{
    int[] temp = copy(b.getLines());
    for (int i = 0; i < temp.length; i++)
    {
        if (temp[i] > 0)
        {
            b.take(i + 1, 1);
            break;
        }
    }
}
```

Photograph 12. Code to Make a Strategic Move 4/4/19

Calculations

Average = sum of the data / amount of the data

Unconditional Probability = how many times something happened / total number of chances it had to happen

The average number wins of no strategy bot in the experimental group was ⅓. This was calculated by added up all the wins of the no strategy bot from all the trials (onee) and then dividing by the number of trials (three). The average number of wins for the strategy bot was 9,999 ⅔. This was found by dividing the total number of wins of the strategy bot (29,999) by the number of trials (three). The average amount the no strategy bot won was about 6 times less than the predicted amount in the hypothesis. The average number of wins for bot 1 of the control group was 5010. This was found by adding up all the wins of bot 1 in the control group (15,030) and dividing by the number of trials (three). The unconditional probability of the no strategy bot winning in the control group was about 50.3% (5,030/10,000), and its counterpart in the control group (bot 1) probability of winning was about 99.99% (29,999 ⅔/10,000).

Discussion

Conclusion

In this experiment, a computer bot that could play misere Nim perfectly played against a computer bot that played misere Nim randomly 10,000 times. The perfect strategy bot went first, and the no strategy bot went after the strategy bot's first turn. The experiment was set up this way because the bot without a strategy could, theoretically, win every match it played if it made no mistakes because the randomly playing bot went second. In misere Nim, going second guarantees you a victory if you make perfect moves. If the no strategy bot made at least one mistake, the strategy bot would then proceed to win that match. Furthermore, the hypothesis of this experiment was if 10,000 matches of computerized misere Nim are played, a no strategy bot going second will win at least five times against a strategy bot going first. The hypothesis was not supported. The no strategy bot won once out of 30,000 matches, and the strategy bot won an average of 9999 2/3 times. Since it did not win an average of 2 times every 10,000 matches, the hypothesis was not supported. A revised hypothesis would be that if 10,000 matches of Misere Nim are played, a strategy bot will have a ⅓ chance of losing once.

Applications

This experiment is of use to society through its demonstration of how a strategy improves your chances of winning. When both bots played randomly, the two bots garnered around the same number of wins. However, when one bot played strategically, its chances increased significantly. Bot 1, playing without a strategy, originally had an approximately fifty percent chance of winning. When it adopted a strategy, that chance increased by around forty nine percent. This illustrates that adopting a strategy in real life - whether it be in war, politics, or business - is extremely helpful, and more far more beneficial than making random decisions. The percent increase in the chances of winning will vary depending on the type of competition being played - playing with strategy might improve your chances differently in other games.

Limitations

This experiment could be improved if the code regarding the perfect strategy was peer-reviewed by professional software developers. Making an algorithm to play perfectly has many areas in which a mistake could be made that influences the outcome of the experiment, and having others review the code would help in making sure the perfect strategy was, in fact, perfect. It would also be beneficial to run the experiment on different types of computers. If the code was run on different types of computers, it would ensure the computer the code was run on in this experiment did not influence the percent increase in the chance of winning for the bots. Many challenges in the development process were encountered that were not expected, and the time spent debugging took away from testing time. Testing the perfect strategy algorithm against make perfect moves against human players who know how to win every time they go second in misere Nim (which is possible and has been taught to humans) would help verify that the code behind testing the perfect strategy accomplished its goal.

Error Analysis

The nature of what we are measuring being affected (the chances of the bots to win) is statistical. In theory it is possible all three trials were outliers, and not close to the mean. The code was run on the same computer every time, however, other factors relating to the computer, such as battery power left, location of the computer, temperature of environment, etc. were variable, and could possibly have affected the results. In the recording of data, numbers may have been recorded differently than what was on the screen due to human error. For example, the value of 6,435 could be recorded by mistake instead of the value of 6,453.

Future Analysis

One topic that could be pursued is the math behind why the results were achieved. Since Nim is purely a game of numbers, it is feasible that there is a formula, pattern, or algorithm behind the chances of each bot winning. In addition, running an alternate version of this experiment with non-misere Nim as the game of choice would reveal if the act of toggling the win conditions affects the dynamics of the game, since in misere Nim, the last player to take a unit loses. This is in contrast to regular Nim, in which the last player to take a unit wins. Lastly, a topic that could be looked upon in the future is the chances of bots winning when they both play with strategy. The results of that experiment would verify or dismiss the notion tha a bot going second can always win.

References

Acín, A. (2016). Certified randomness in quantum physics. *Nature*. Retrieved from SIRS Researcher database.

Citation Tools Choose Citation Format: Haaxma-Jurek, J. (2014). Artificial intelligence. In *The Gale Encyclopedia of Science* (5th ed.). Retrieved February 7, 2019, from https://go.galegroup.com/ps/retrieve.do?tabID=Reference&resultListType=RESULT_LIST&searchResultsType=SingleTab&searchType=TopicSearchForm&currentPosition=2&docId=GALE%7CCV2644030176&docType=Topic+overview&sort=Relevance&contentSegment=&prodId=SCIC&contentSet=GALE%7CCV2644030176&topicId=QTJWNR378116673&searchId=R6&userGroupName=delray3411&inPS=true#

In Search of One App for Many Targets. (n.d.). *Electronic Design*. Retrieved from SIRS Researcher database.

Java (programming language). (2018). In *The Hutchinson Encyclopedia*. Retrieved from SIRS Researcher database.

NIST's new quantum method generates really random numbers. (2018). *UPI Space Daily*.
Retrieved from SIRS Researcher database.

Programming. (n.d.). In *World of Computer Science*. Retrieved from Gale Virtual Reference Library database.

Rougetet, L. (2014). A prehistory of Nim. *The College Mathematics*, *45*(5). Retrieved from JSTOR database.

Steve Jobs: A Silicon Valley legend's legacy. (2011, October 7). *The Gazette*. Retrieved from SIRS Researcher database.

Tesla, K. (2017, September). Computer Games On The Go. *Mechanical Engineering*. Retrieved from SIRS Researcher database.