11-30-2009

Chaos by design: using pseudo-randomness to make artificial intelligence more "human"

Eric Moore

2009 - 5.0

University Honors Program Research Project

APPROVAL PAGE

Student Name: Eric Moore

Department: Computer Science

College: Science

Degree: Bachelor's of Science

Project Advisor: Dr. Harry Delugach

Full title of project as it should appear in Graduation Program and transcript:

Chaos by Design: Using Pseudo-Randomness to Make Artificial Intelligence More "Human"

Abstract:
This paper examines techniques to make artificial intelligence seem more "human" by adding random elements to the sequential execution of instructions. Statistical data is analyzed from experimental runs of an example implementation of these techniques. The question of what makes intelligence "human" is briefly explored, and the advantages and disadvantages of randomness in artificial intelligence are considered.

Approved by:

Project Advisor: _____ (signature) Date: 11-30-2009

Department Chair: H S Rangwala (signature) Date: 11-30-09

University Honors Program Director: _____ (signature) Date: 11-30-2009

Chaos by Design: Using Pseudo-Randomness to Make Artificial Intelligence More "Human"

Eric Moore
The University of Alabama in Huntsville
mooree@uah.edu

30 November 2009

Abstract

This paper examines techniques to make artificial intelligence seem more "human" by adding random elements to a deterministic algorithm. Statistical data is analyzed from experimental runs of an example implementation of these techniques. These results are compared with the performance of a human on the same task. The question of what makes intelligence "human" is briefly explored, and the advantages and disadvantages of randomness in artificial intelligence are considered.

1 Introduction

Could a machine ever truly think like a human? Could it at least give the illusion of thinking like a human? These questions have been explored time and again in the contexts of both academia and entertainment. Successful films such as *I, Robot* have artistically examined the implications of such a form of artificial intelligence. These are questions which bring into focus the unique nature of humanity and therefore command the attention of many. Concepts such as "free will" and "sentience" are frequently considered in respect to artificial intelligence. What are the motives behind the thought processes of humans and could a machine ever make decisions as autonomously, and sometimes unpredictably, as a human? As

hardware and software technology progresses ever onwards, the possibilities continually increase for experimentation with artificial intelligence. Countless powerful and efficient algorithms have been developed to make specific decisions with mathematical precision based on sets of data. This is useful and impressive technology, but is a mathematically flawless method of decision-making "human"? Consequently, there has been an increase in the research of artificial intelligence algorithms which have a degree of unpredictability. One example is the Conference on Uncertainty in Artificial Intelligence [1].

The intelligent task examined in this paper is a number guessing game. The experimental program has an algorithmically efficient method of guessing numbers, shrinking the range of possible outcomes each time, but occasionally a random element is introduced. A certain percentage of the time, which is a customizable number, the program will take a "blind guess" within the current range of possible outcomes. The question this experiment attempts to address is whether or not unpredictability through occasionally random outcomes makes artificial intelligence more like human intelligence. This question is brought into the spotlight directly with a comparison between the way the program plays the number guessing game and the way a child plays the game. How do the results compare? Keep reading to find out!

2 The Development of the Project

Winston Churchill said "There is nothing wrong with change, if it is in the right direction." [2] Since it was first undertaken, this project has gone through many changes in focus and scope. In its original state, the goal of the project was to study the plausibility of the artificial intelligence depicted in popular movies. This was obviously a

very broad topic, destined to be brought to a smaller focus as the project developed. The project's next evolutionary leap led to an investigation of learning algorithms to study the feasibility of the learning displayed by the A.I. in many movies. The intent at this point was to develop an implementation of a simple game which would learn the rules through repetitive playing of the game, based on wins and losses. An example of such a program is listed in the references at the end of the paper [3]. While this topic proved interesting and informative, it soon became obvious that it was still too broad for a one-semester project, and also that it did not address one of the most fascinating questions at the root of most films about Artificial Intelligence. This question is whether or not a machine could ever "decide" to behave unexpectedly.

In its final form, the focus of the project moved to the exploration of occasionally unpredictable programs. In order to keep the scope narrow and allow for more experimentation, the idea of a number guessing game was devised. By adding a variable random element to the sequential algorithm for guessing numbers, the concept could be investigated in detail. In order to truly analyze the results, the idea emerged of comparing the program's method of playing the game with that of a child. The statistics and strategies of both are compared and contrasted to determine the similarities. This added a human element to the experiment, making it truly interesting and unique.

While the experimental code may seem like a very simple and almost elementary example, it demonstrates a concept which could be applied to much larger problems. It also keeps complications to a minimum, allowing for a greater focus on the area of interest, which is the addition of random outcomes into the flow of execution.

3 The Theory

Every experiment has a purpose, which is usually the testing of a working theory or conjecture. So what is the driving theory behind this research project? It could perhaps be stated in this way: *There is an element of "pseudo-randomness" to human thought that, if simulated, can make machine intelligence more similar in effect to human intelligence*. What is meant by "pseudo-randomness"? In this context, the term is used to describe behavior that does not emerge from an obvious logical path, but from a combination of conditions which are external to the problem being considered. This behavior is "pseudo-random" because, while it may appear to be truly random, it is actually based on external factors. Harvard researcher Salil Vadhan said "Pseudorandomness is the theory of efficiently generating objects that 'look random' despite being constructed with little or no randomness." [4] Some would argue that this is true of all apparent randomness in the universe, but this question will be left to philosophers lest this paper become truly chaotic!

Translating this element of human thinking into the realm of machine intelligence is perhaps the most formidable challenge to creating programs which behave like humans. Also, the introduction of pseudo-randomness into programmatic algorithms causes a loss in predictability and sometimes in efficiency. Keep in mind, however, that true human intelligence *frequently* makes mistakes. In the words of the English poet Alexander Pope, "To err is human." Therefore, we can expect an occasional loss in mathematical precision as we attempt to introduce this less predictable, illogical element into a machine. So why explore this theory at all if it will probably cause a loss in efficiency? It should be explored because of what could possibly be gained, not only in the study of Computer Science, but also in

understanding ourselves! It is truly an interdisciplinary study, one which provides evidence that Computer Science is in fact a "science" in the true sense of the word; a field of study which helps us to understand our world.

Perhaps the most exciting part, however, is that we *really can't predict* with any consistency how pseudo-randomness will affect the performance of a program. By definition, randomness is unpredictable. Pseudo-randomness may not be true randomness, but it is certainly less predictable than a standard logical path. If we can learn to at least roughly predict the behavior of pseudo-random algorithms, it could help us predict the pseudo-random behavior of humans.

4 The Experiment

It is time to put all of this theorizing to the test! As outlined in section 2, the experimental code examined in this paper plays a number guessing game. It may seem childish, but that is precisely why it is a good choice for this research topic. It is likely that only in youth can human intelligence be observed largely free of the effects of higher academic studies, which can train the mind to think more mathematically and logically. Also, by studying the effects of pseudo-randomness on something as simple as a guessing game, we can examine the theory at its most basic level.

4.1 The Game

The number guessing game is very simple and involves two players. One of the players chooses numbers and the other player attempts to guess these numbers. It is played in repetitive "rounds", in which guesses are taken until the number is found. The rules of the game are as follows:

-Player 1 chooses a number within a defined range.
-Player 2 is asked to guess the number.
-Upon each guess, Player 1 will tell Player 2 if their guess is equal to (i.e., a "win"), greater than, or less than the number being guessed.
-Player 2 can guess as many times as necessary.

4.2 The Code

In the experimental code, the computer plays the role of both Player 1 and Player 2. There are two different strategies for taking guesses which are employed by the program. The first is an efficient binary search strategy, which is guaranteed to complete in at least O(log n) time [5]. Studies show that binary search is the most mathematically effective way to solve this problem, so this strategy represents the best that machine intelligence has to offer in regards to playing a game like this. The other strategy used by the program consists of taking a pseudo-random guess, within the range of possible choices, based on a random number generator. The frequency of use for this strategy is determined by a customizable percentage which can be set by the experimenter. If the percentage is set to 20, for example, then the program will use the pseudo-random strategy twenty percent of the time. The game is played for every number within the defined range and then statistics are generated based on the number of guesses required for each number. All of the guesses and the final statistics of each round are printed to the screen for analysis by the user. The code is included at the end of this paper.

4.3 The Results from the Program

The program outputs the average number of guesses, the min and max guesses, and the standard deviation after all numbers in the

range have been guessed. The most interesting of these are the averages, but the other statistics have been included in case they reveal anything interesting about the performance of the program at different percentages of pseudo-randomness. Here are the results from the program after running it for different percentages of randomness, separated by intervals of ten, with a range of 0 to 100. At zero percent randomness, the strategy is purely binary search.

Randomness	**Average # of Guesses**	**Standard Deviation**	**Min # of Guesses**	**Max # of Guesses**
0%	5.812	1.318	1	7
10%	5.871	1.318	1	8
20%	5.891	1.547	1	9
30%	6.624	1.541	1	10
40%	6.089	1.666	1	10
50%	6.416	1.753	2	11
60%	6.614	2.139	1	12
70%	6.970	2.056	2	12
80%	6.683	2.384	1	13
90%	7.277	2.239	1	13
100%	7.446	2.543	1	13

4.3 Analyzing the Results from the Program

Given the range of 0 to 100, which one hundred and one choices, we know that the binary search strategy will be guaranteed to find the number in log(101) time, which is 6.658211, or 7 guesses in this case. With zero percent randomness, we see that the max number of guesses was 7, which is consistent with this projection. The average is somewhat lower, though. This is to be expected, since binary search takes *at most* log(n) time and will often find the number more quickly. As the percentage of randomness increases, there is generally an increase in the average number of guesses required to find the number. This increase is not linear, however, which is interesting. Figure 1 is a graph of the average number of guesses as the percentage of randomness grows.

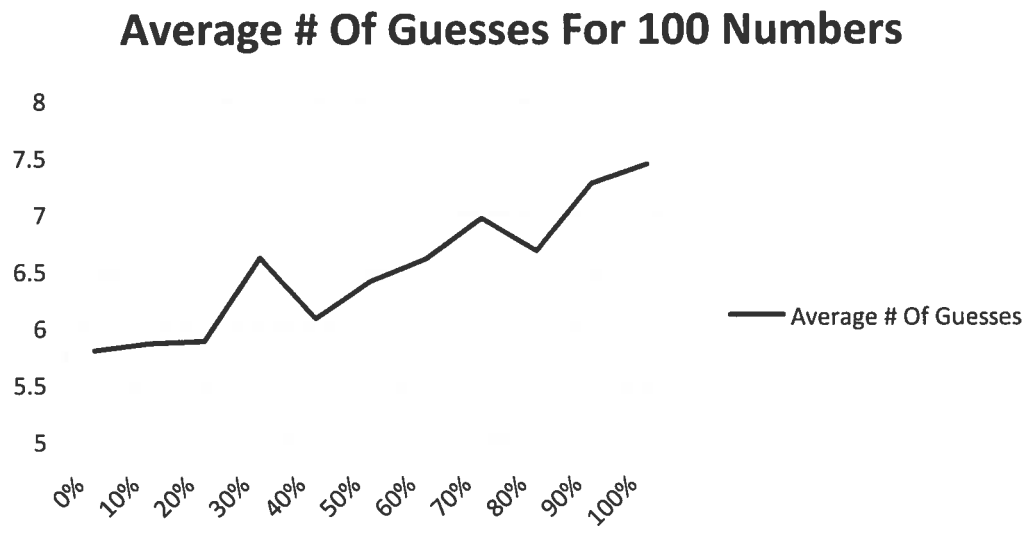

Figure 1

As can be seen in Figure 1, there are two places where the program actually became more efficient as the percentage of randomness increased. This happens between thirty and forty percent, as well as between seventy and eighty percent. Both of these cases

demonstrate the unpredictability that pseudo-randomness introduces into the program. Most notable, however is the improvement between thirty and forty percent, as it is the most significant increase in efficiency. It is interesting to discover that, in any case, using less of the binary search strategy could improve the performance of the program. Since we know that binary search is the most efficient way to find a number, using less of this strategy should theoretically cause the program to be less efficient in all cases. In order to investigate this interesting anomaly, the range between thirty and forty percent was investigated in more detail. Figure 2 shows a graph of the average number of guesses at each individual percentage between thirty and forty. This time there was a smaller increase in efficiency between these two percentages, but in between there was considerable fluctuation. It would appear that the smaller the change in randomness, the less predictable the outcome will be.

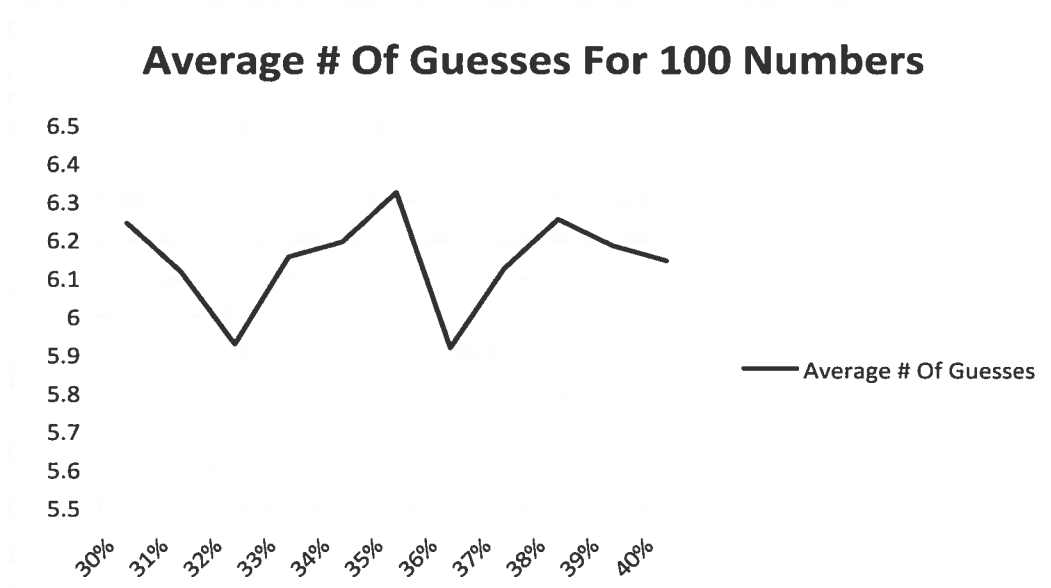

Figure 2

Since the results seem to be more consistent at a macro level,

the program was run with a range of 0 to 1000 to see if the curve is more uniform. Figure 3 shows a graph of the average number of guesses as randomness increases for 1000 numbers.

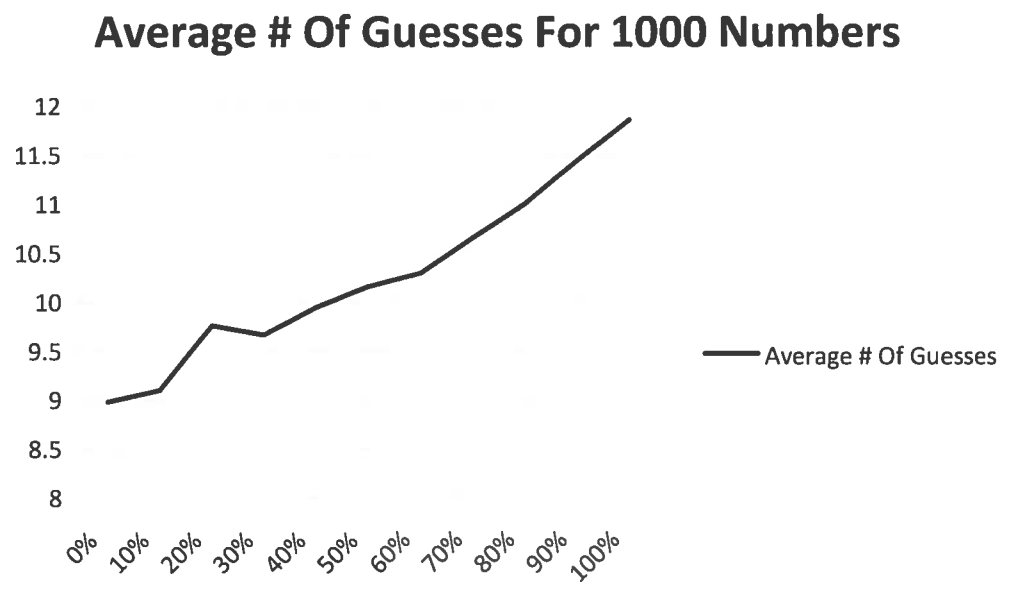

Figure 3

It is clear from Figure 3 that the increase is much closer to linear when the problem domain is larger. There is one case of improved efficiency, between twenty and thirty percent, but it is singular and much less dramatic than the improvements seen with 100 numbers. This does demonstrate, however, that randomness will consistently introduce some degree of unpredictability into the equation.

All of these results show that the pseudo-random strategy definitely changes the performance of the program to varying degrees, but whether or not it makes the program more "human" is the question which this paper is most concerned with. Testing the theory stated in Section 3 will require the addition of another element into the experiment.

4.4 Playing the Game with a Human

In order to determine whether or not the addition of a certain percentage of pseudo-randomness into the program makes it perform more similarly to a human, the guessing game was played with a human, an anonymous ten-year-old female. A child of this age was chosen primarily for two reasons: a ten-year-old wants to win, and a ten-year-old has probably not yet been trained to use more mathematically efficient methods to solve this type of problem. Because of these reasons, the results should show the performance of a purely human intelligence. The number guessing game was played for twenty different numbers between 0 and 100, selected by a random number generator to ensure that the conditions are the same as they are in the program. Each of the ten-year-old's guesses were recorded for analysis. These results can be compared to the results from the program at each percentage of randomness to find the closest match, which should theoretically reveal the approximate percentage of randomness employed by the human subject.

4.5 The Results from the Human

After playing the game for 20 numbers between 0 and 100, the average number of guesses required by the ten-year-old to determine a number was calculated.

Average # of guesses for human = **6.05**

This is closest to the algorithm's behavior at forty percent random, which had an average of *6.08911*. It is very interesting that the average from the human is closest to the most surprisingly efficient percentage of randomness demonstrated by the program. If you recall, when the program was run for numbers between 0 and 100, the average improved unexpectedly and significantly between thirty and

forty percent.

Before being told about the program and the methods it uses to take guesses, the ten-year-old was asked if she had a strategy. She responded that she sometimes took random guesses and sometimes picked "things in the middle". Sound familiar? It would appear that the strategies used by a human are quite similar to the methods employed by the program, particularly at forty percent random. Following is a list of the guesses taken by the ten-year-old for one number.

The number is **29**
Guesses:
50, 30, 20, 25, 27, **29**

We see that she began by guessing 50, which is following a binary search strategy. The second and third guesses (30 and 20) depart from this strategy, however, picking numbers close to the middle of the range but not right in the middle. The fourth and fifth guesses (25 and 27) return to a binary search strategy, and the sixth (29) is simply one of two possible choices. This round of the game seems consistent with the conjecture that the human strategy was approximately forty percent pseudo-random. It is obvious that these guesses were not completely random, though. When given the choice, the ten-year-old tended to pick numbers ending with zero or five. This trend can be seen throughout all twenty numbers guessed by the human.

Also worth noting is that the ten-year-old occasionally couldn't always remember previous guesses she had made and attempted to guess the same number more than once, or numbers which had already been determined to be outside the range of possible results. These are all factors which must be taken into account when

considering the "humanizing" of machine intelligence. In the case of this game, these imperfections appear to be simulated fairly accurately using a random number generator.

5 Conclusions

Let us consider the results of the experiment as they relate to the theory put forward in Section 3. The efficient and mathematically precise binary search strategy performed as expected, but the results of the human experiment confirmed that human intelligence is not always this precise or efficient. In fact, the human's performance matched most closely with that of the program at forty percent pseudo-random. It would seem, therefore, that the addition of pseudo-randomness did cause the machine intelligence to more closely resemble human intelligence. These results suggest that, in cases where there are no serious consequences, we can expect the motivations behind a human's decisions to be approximately forty percent pseudo-random. This is a truly interesting suggestion, one which is worth investigating in more detail.

It is easy to dismiss a number guessing game as trivial, but these results could also be applied to a larger and more complex problem. The reasons why someone picks a certain number as a guess within a defined range could translate to the reasons why someone chooses to eat at a certain restaurant for lunch. Vague and illogical motivations such as nondescript feelings or the power of suggestion can play a large role in this decision, just as in a number guessing game. A certain restaurant "sounding better" than another, or a certain advertisement being particularly persuasive, are conditions which fall into the pseudo-random category of motivations. Again, choosing a restaurant to eat at seems like a trivial decision, but it

demonstrates how a small problem can translate to a larger one. Often people make decisions based on a "gut feel" or "intuition", concepts which are hard to define. These motivations usually only make sense to the person experiencing them, and may be meaningless to an unbiased observer. In fact, these reasons can seem as arbitrary to someone else as a pseudo-random number generator. The results of the experiment detailed in the previous sections support this claim, demonstrating at a basic level that a pseudo-random number generator can cause an apparently similar outcome to that of human pseudo-randomness.

The case is not being made that the concept of a "will" amounts to a glorified pseudo-random number generator. There is a clear and profound difference between the two, since a computer has no real "understanding" of what is taking place and has not actually made a decision of its own accord. Looking at what some of the most influential thinkers have said about the will, it quickly becomes obvious that it would be extremely difficult to argue that a machine possesses this faculty. Speaking about the concept of a will, Aristotle said "the mind in question is that which makes its calculations with an end in view" [6]. Similarly, the theologian Thomas Aquinas said "to 'will' implies the simple appetite for something: wherefore the will is said to regard the end, which is desired for itself" [7]. Does the computer have an end in view when it makes a decision? No, it only executes its instructions one at a time, regardless of what comes next. It only gives the appearance of making a decision, which is what this paper is actually concerned with. The case being made is not that a program's pseudo-randomness can be comparable in nature to a human's motivations, but that they can be similar in effect. The experiment with the number guessing game demonstrates that a certain

percentage of pseudo-randomness can help a program effectively *simulate* human intelligence.

6 Acknowledgements

This project has been informative and enjoyable, and I hope that others will find the results as fascinating as I have. I would like to thank Dr. Harry Delugach for his advisement and help with the development and implementation of the project. The UAH Honors Program has been an important part of my undergraduate college experience and I am glad that I was able to develop my Honors Senior Research Project with the program's new director as my advisor. I want to acknowledge the National Consortium for MASINT Research (NCMR) for their Scholars Grant which, funding my final semester at UAH, made it possible for me to take the classes necessary to finish the Honors Program. Thanks to Dr. Tim Newman for his nomination, Dr. Sara Graves for her support of NCMR at UAH, and Dr. Peter Bythrow for making the scholarship possible. Lastly, I would like to thank the anonymous ten-year-old "human subject" for her time and enthusiasm about participating in the experiment.

References

[1] Meek, C. 2004. "Proceedings of the 20th conference on Uncertainty in artificial intelligence." In *ACM International Conference Proceeding Series,* Vol. 70. (AUAI Press, Arlington, VA, USA, 2004), http://portal.acm.org.elib.uah.edu/citation.cfm?id=1036843&coll=ACM&dl=ACM&CFID=65377318&CFTOKEN=31971700# Accessed 11/29/09

[2] "Winston Churchill Quotes" British Orator, Author and Prime Minister during World War II. 1874-1965, http://thinkexist.com/quotation/there_is_nothing_wrong_with_change-if_it_is_in/179038.html Accessed 11/29/09

[3] Gajic, Zarko. 2001. "A Simple example of Artificial Intelligence using Delphi" About.com Guide, http://delphi.about.com/od/gameprogramming/a/aigamesample.htm Accessed 11/29/09

[4] Vadhan, S. 2007. "The unified theory of pseudorandomness" guest column. SIGACT News 38, 3 (Sep. 2007), 39-54, http://doi.acm.org/10.1145/1324215.1324225 Accessed 11/29/09

[5] Pfaff, B. 2004. "Performance analysis of BSTs in system software." In *Proceedings of the Joint international Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, June 10 - 14, 2004). SIGMETRICS '04/Performance '04. ACM, New York, NY, 410-411, http://doi.acm.org/10.1145/1005686.1005742 Accessed 11/29/09

[6] Aristotle. "On the soul, with an English translation by W.S. Hett, London" WILLIAM HEINEMANN LTD / Cambridge, Massachusetts, Harvard University Press, 1964, p. 187- 189 Accessed 11/29/09

[7] Aquinas, Thomas. "Summa Theologica" Translated by The Fathers of the English Dominican Province, 1920, http://www.newadvent.org/summa/ Accessed 11/29/09

Appendix: Experimental Code

```
//GuessANumberVarying.cpp
//Author: Eric Moore
//Date: 10/28/09
//Description: This program tests the effects of adding a random
//element into artificial intelligence. A number-guessing game is
//played using a binary search algorithm. The unique part of this
//program is that a customizable percentage of the guesses are taken
//at random within the current range of possible answers instead of
//choosing the midpoint. The purpose of this program is to observe how
//adding a random element to the mathematical approach affects the
//performance of the algorith and, more importantly, if it makes the
//intelligence more "human". This will be examined by comparing the
//results of this program with the way a child plays this same game.
//*********************************************************************
//Revision 1: Created basic structure of code with interactive
//functionality and the binary search method of guessing.
//*********************************************************************
//Revision 2: Added random logic.
//*********************************************************************
//Revision 3: Made random percentage customizable and automated the
//choosing of numbers to be guessed. Program loops through each number
//from 0 to 100 and prints out how many guesses it takes to determine
//each number.
//*********************************************************************
//Revision 4: Added statistical calculations (average, min/max, std
//dev).
//*********************************************************************
//Revision 5: Added comments.

//include files
#include <iostream>
#include <time.h>
#include <math.h>

using namespace std;

//define variables
int numberchosen;
int guessarray[101];
int currentmin, currentmax, currentguess, choice, method, percentage;

//function prototypes
int guess(int num);
int randguess(int min, int max);
int strategicguess(int min, int max);
int run(int percentage);
int calcaverage();
int calcstddev();
int calculateminandmax();

//Function: main
//Inputs: None.
//Returns: int
```

```
//Description: Initializes program, retrieves customizable "random"
//percentage, and calls the functions to perform the main program
//functionality on every number from 0 to 100.
int main()
{
   srand(time(NULL)); //seed the random number generator
   //prompt for percentage of randomness and read in value
   cout << "Welcome to the number determination game!\n" <<
           "\nWhat percentage of the time should the guess be random? "
           << "Enter an integer from 0 to 100 and then press enter:\n";
   cin >> percentage;
   //check validity of input
   if (percentage < 0 || percentage > 100)
   {
      cout << "Invalid number.\n\n";
      return main();
   }
   //play game for each number from 0 to 100
   for(int i=0; i<=100; i++)
   {
      numberchosen = i;
      run(percentage);
   }
   //calculate statistics
   calcaverage();
   calcstddev();
   calculateminandmax();
   return 1;
}

//Function: guess
//Inputs: int num (a guess)
//Returns: int (number which means greater than, less than, or equal)
//Description: Compares a guess (num) with the current number
//(numberchosen)
int guess(int num)
{
      if (numberchosen > num)
            return 1; // number is greater than the guess
      else if (numberchosen < num)
            return 2; // number is less than the guess
      else if (numberchosen == num)
            return 3; // number is equal to the guess
}

//Function: randguess
//Inputs: int min, int max
//Returns: int (a guess)
//Description: generates a random guess
int randguess(int min, int max)
{
      int currentguess = (int) ((max - min)*rand()/(RAND_MAX));
      currentguess = currentguess + min;
      return currentguess;
}

//Function: strategicguess
```

```
//Inputs: int min, int max
//Returns: int (a guess)
//Description: generates a guess in the middle of the current range
int strategicguess(int min, int max)
{
      int currentguess = min + ((max - min) / 2);
      return currentguess;
}

//Function: run
//Inputs: int percentage (of randomness)
//Returns: int
//Description: Plays the guessing game at a certain percentage of
//randomness
int run(int percentage)
{
   int numguesses = 0; //no guesses yet
   //set the max to 100 and the min to 0
   currentmax = 100;
   currentmin = 0;
   bool found = false; //the number has not yet been found
   while (found == false) //until the number is guessed
   {
            numguesses++; //keeeps track of number of guesses
            //select method for guess
            method = (int) (100*rand()/(RAND_MAX + 1.0));
            if (method < percentage) //take a random guess
                  currentguess = randguess(currentmin, currentmax);
            else //take a strategic guess
                  currentguess = strategicguess(currentmin,
                  currentmax);

            choice = guess(currentguess); //check guess against number
            if (choice == 3) // equal
                  found = true;
            else if (choice == 2) // less than (guess was too high)
                  currentmax = currentguess - 1;
            else // greater than (guess was too low)
                  currentmin = currentguess + 1;
   }
   //print out the number of guesses it took
   cout << "It took " << numguesses << " guesses to determine the
            number " << numberchosen << endl;
   //add this number of guesses to the array for statistical
   //generation
   guessarray[numberchosen] = numguesses;
   return 1;
}

//Function: calcaverage
//Inputs: None
//Returns: int
//Description: calculates the average number of guesses required to
//find a number
int calcaverage()
{
   double average;
```

```
    //calculate sum
    int sum=0;
    for (int i=0; i<=100; i++)
    {
        sum+=guessarray[i];
    }

    //calculate average and print it out
    average = sum / 101.0;
    cout << "The average number of guesses was " << average << endl;
    return 1;
}

//Function: calcstddev
//Inputs: None
//Returns: int
//Description: calculates the standard deviation of the number of
//guesses
int calcstddev()
{
    double sum = 0;
    double STD_DEV = 0; // returning zeros
    double result;

    for (int i = 0; i <= 100; i++)
    {
        sum = sum + guessarray [i];
        STD_DEV = STD_DEV + pow((double)guessarray[i], 2);
    }
    result =  sqrt ((STD_DEV/101) - (pow(sum/101,2)));
    //print out the standard deviation
    cout << "The standard deviation of guesses was " << result << endl;
    return 1;
}

//Function: calculateminandmax
//Inputs: None
//Returns: int
//Description: calculates the min and max number of guesses
int calculateminandmax()
{
    int min=100, max=0;
    for (int i=0; i<=100; i++)
    {
        if (guessarray[i] < min)
            min = guessarray[i];
    }
    for (int j=0; j<=100; j++)
    {
        if (guessarray[j] > max)
            max = guessarray[j];
    }
    //print out the min and max
    cout << "The min number of guesses was " << min << endl;
    cout << "The max number of guesses was " << max << endl;
    return 1;
}
```