My implementation of the Pinball Project is written in C#. I chose to use the language for programming because C is more optimized than many other languages, which is useful for this project because it requires a very large number of tests. My algorithms are based on the project description, and are very simple, involving few variables and lines of code in each method. The project is written with 2 major classes, a main class, where data is recorded and program is executed, and the background pinball class, which has all the algorithms which check for intersection, reflection, and hits.

In the pinball class, the method public run(double angle) executes a while loop which checks for a condition when there are no more circles for the particle to hit. While the loop runs, position and velocity are updated with intersection and reflection methods, respectively. In the main method, for ease of use, the program allows the user to select how he wants to proceed with the program, whether to run a random number of tests, or using a random angle. In the first case, a nested for loop runs 100\*x executions of the run(..) method in the pinball class, where x is the number of tests/100 the user wants to run. I used the 100 multiplier because in my data collection, I record the angle with maximum hits in each of the 100 runs. That way, I get 100 angles, all of which stand for the top 100 hits. It also makes it easier to see the progress of the program when a large number of tests are conducted.

Although I have tried to optimize my code as much as possible, it is not as fast as I had tried to make it. On average, 1 billion runs take about 30 seconds to execute on my computer. I believe that my code could have been written to execute the same number of trials at much faster speed, although it might have required some compromises in the way my data is collected. For data collection, I primarily use a text file, called `out.txt`, which is located in the home folder of the program. A text file is important to my program because the data that I collect from the trials may be easily lost in a console output. Also, a file allows faster writes, and is not limited to size as the console might be. The data recorded depends on what the user chooses to try. For random angles, a list of 100 angles with top hits is recorded. For specific angles, the number of hits is displayed in console, followed by a list of circles in the order hit. Also, my complete code runs in a surrounding loop that executes the entire program while the user doesn’t wish to quit.

The greatest number of hits that I was able to achieve was 18. To get this number, however, I had to use brute force. For the first few times that I ran the program, I used a random number generator that returned double values with a 15-digit precision. I got an 18-hit in one such trial, but since my data was not parsed correctly to display all 15-digits of precision, I was only able to retrieve the first 9 bits of the double value. Using that value, I ran my program for each value starting from that number (x-0.25), with a 1.0 X 10^-14 precision. It took my code 4 hours to find the right angle (I had programmed it to stop executing once it reached 18). Although I had initially found that number in a random trial, it was necessary for me to use tougher methods to get the same value again. For the final report, I used two methods for reporting frequency and relative frequency I used a random number generator and a sequential number incremental. For the sequential trials, I found an initial random angle between 0 and 1.0, and then incremented the angle by 0.5 in each iteration. That way, I got a sequence of angles all equally spaced from 0 to 360 degrees. Not all of top 10 maximum angles that I have reported in this project are not are from the trials that I conducted for the random and sequential runs in this report.

In the random trials, on average, the highest number of hits I got was 13-14. In the particular data used in this report, I got a maximum of 13 hits. The angles in sequential order did not fare as well, however. The most I got for this report form that method was 4. I think that if my initial random angle was different, I would have gotten better hits. However, since it is random, it cannot be controlled. What was surprising about the number of hits in sequential order was that the number of hits for 2 hits and 3 hits were exactly the same. Although it unclear to me why this happened, it is a trend that I saw in different trials of the same sort.

I think that my final maximum angle is one of the few angles in the range that produces very high number of hits. Although I conduced several tests, I never got anything higher than 18. I do not think that it is the highest number attainable, because angles with more precession would yield better patterns. I used a double-precision data type for all my calculations. Had I used a 128-bit decimal data type, I would have gotten a 30-digit precision, compared to the 16 digits I had used for this project. However, I think that even 18 hits is a very high number because the average for all the trials was about 13, which is much lower.

Max Angles:

|  |  |
| --- | --- |
| 1) 94.5372630745615 | 18 |
| circle1, circle2, circle3, circle2, circle1, circle2, circle3, circle1, circle2, circle1, circle2, circle3, circle2, circle1, circle2, circle1, circle2, circle3, escape. | |

|  |  |
| --- | --- |
| 2) 95.53703633252331 | 16 |
| circle1, circle2, circle1, circle2, circle1, circle2, circle1, circle2, circle3, circle2, circle1, circle3, circle2, circle3, circle1, circle3, escape. | |

|  |  |
| --- | --- |
| 3) 94.5290664995993 | 15 |
| circle1, circle2, circle3, circle2, circle3, circle2, circle3, circle1, circle3, circle1, circle2, circle3, circle1, circle2, circle1, escape. | |

|  |  |
| --- | --- |
| 4) 84.555319184230 | 14 |
| circle1, circle3, circle1, circle2, circle3, circle1, circle3, circle2, circle1, circle2, circle1, circle2, circle1, circle2, escape. | |

|  |  |
| --- | --- |
| 5) 85.4708580470043 | 14 |
| circle1, circle3, circle2, circle3, circle2, circle3, circle1, circle3, circle1, circle3, circle2, circle1, circle3, circle2, escape. | |

|  |  |
| --- | --- |
| 6) 94.5372630745615 | 13 |
| circle1, circle2, circle3, circle1, circle3, circle2, circle1, circle3, circle1, circle2, circle1, circle2, circle1, escape. | |

|  |  |
| --- | --- |
| 7) 325.462675361178 | 13 |
| circle3, circle2, circle1, circle2, circle3, circle2, circle3, circle2, circle3, circle2, circle3, circle2, circle3, escape. | |

|  |  |
| --- | --- |
| 8) 214.537318446924 | 12 |
| circle2, circle3, circle1, circle3, circle2, circle3, circle2, circle1, circle3, circle2, circle3, circle2, escape. | |

|  |  |
| --- | --- |
| 9) 324.56277899796 | 12 |
| circle3, circle2, circle3, circle1, circle3, circle1, circle2, circle3, circle1, circle2, circle1, circle2, escape. | |

|  |  |
| --- | --- |
| 10) 215.536021061957 | 12 |
| circle2, circle3, circle2, circle3, circle2, circle1, circle2, circle1, circle2, circle1, circle2, circle3, escape. | |

Random:

100,000,000 program executions, with random angles 0< **θ** < 360.

|  |  |  |
| --- | --- | --- |
| Number of hits before escaping. | Frequency | Relative Frequency |
| 0 | 71,962,775 | 71.962775 % |
| 1 | 22,438,900 | 22.438900 % |
| 2 | 4,568,821 | 4.568821 % |
| 3 | 841,322 | 0.841322 % |
| 4 | 153,836 | 0.153836 % |
| 5 | 28,030 | 0.028030 % |
| 6 | 5,128 | 0.005128 % |
| 7 | 984 | 0.000984 % |
| 8 | 166 | 0.000166 % |
| 9 | 31 | 0.000031 % |
| 10 | 6 | 0.000006 % |
| 11 | 0 | 0.0 % |
| 12 | 0 | 0.0 % |
| 13 | 1 | 0.000001 % |
| 14 | 0 | 0.0 % |
| 15 | 0 | 0.0 % |
| 16 | 0 | 0.0 % |
| 17 | 0 | 0.0 % |
| 18 | 0 | 0.0 % |

Sequential:

100,000,000 program executions, with angles spaced by 0.5 degrees, where 0< **θ** < 360

|  |  |  |
| --- | --- | --- |
| Number of hits before escaping | Frequency | Relative Frequency |
| 0 | 71,666,678 | 71.666678 % |
| 1 | 23,333,324 | 23.333324 % |
| 2 | 1,666,666 | 1.666666 % |
| 3 | 1,666,666 | 1.666666 % |
| 4 | 166,666 | 0.166666 % |
| 5 | 0 | 0.0 % |
| 6 | 0 | 0.0 % |
| 7 | 0 | 0.0 % |