The problem is the n-bodies problem, which is a problem that n bodies are created. Each body can have a radius, mass, velocity, and force. The radius tells how big the body is; this and the radius are used with combination of universal gravity to get the force that is applied to other bodies. In this java program, several different threads were created from 1, 2, 3, 4, 8, and 16. The statistics showed that with 2000 iterations that on average 2 threads were the fastest, and in 4000 iterations that 1 thread was the fastest. The data was provided by running on Cambridge, which can mean that several people were using it at the time it was tested giving the impression that 1 thread is faster in 4000 iteration of the problem.

The timing was set at 2000 and 400 iterations or loops on each body that could have a collision with another body. This depends on two main factors: the position of the body because it could spawn next to another body thus caused a collision, and the direction of the velocity because it could be going towards another body. So, in each timing test there is a sense of randomness on where each body is at each time. This could mean that in *figure 1* and *figure 2* that the one type of thread number could be given a less collision than another. To minimize this, the tests were done 3 times with each thread combination with the corresponding iteration number. For *figure 1*, gave more range than *figure 2,* which is 3388 milliseconds and 2462 milliseconds. To make sure that the information is correct a GUI was made to verify that the when a collision happened, it can be justified why it happened at that time instead of analyzing points from the console.

Figure 1

Figure 2

The GUI was made to give a visual of what is happening, and uses the same calls as the console version. This, GUI, was used to tell that the collision was correct. Another way is that the console prints when a collision happens by printing the positions of the two bodies that collided with each other. This data can be used to solve if the all radiuses are the same to make sure that a collision did happens manually. Then, the velocity and the force could be solved by using the position points for each body. The force depends on how close a body is to another. Bodies that are getting closer change from their current velocity, thus showing a force by another body. The GUI shows a body with a curve path, and this gets worse the lighter the body is compared to the other body. Then the velocity can be proven correct by using the points that are printed to the console, because a slope can be determined by two points from the same body. To emphasize this using one body and the velocity is consistent throughout the iterations. The GUI also had a timing too because how close it is to the console version of the program.

The console version’s timings were done to test how effective the parallel threading is, so the GUI was also done because it uses the same simulation as the console version. The GUI’s timing was slower than the console version, but in the GUI there is a thread.sleep(50) which for 2000 iterations means that 100000 milliseconds minimum is added to the GUI. When minus that section out the GUI ran about 2000 milliseconds faster than the console version. This can because it does not have to print the position of all the bodies while the console version does. Meaning that it can be possible that the sleep is less than the system printing all the bodies locations. The versions of the tests ran at different times.

The times of the version differed; this could have changed by the position of the bodies and their velocities. This is because each body could have been further apart from each other or on top of each other. Then, the velocity of each body could have made a collision too. The reason why collisions would slow down the timing is because it has more to calculate and print that it had a collision in both versions. That is each test was ran at least three times, so that it can limit those factors from being effective. The n-bodies problem is an unique problem because it brings several different types of factors to each body in the problem.