Gravitas

Analysis, Design, Testing and Evaluation

Contents

Analysis 3
Background 3
Problem 3
Users 3
Calculation procedure 4
System Requirements 6
Is there a need for a bespoke system? 7
Objectives 8
Design 10
Potential Solutions 10
Navigation Plan 11
Form Design 12
Input, Process, Storage, Output Tables 17
Validation 20
Flow chart of running simulation 21
Graphic drawing process 22
Pseudo code 22
Check Planet Merging 22
Pseudo code 22
Sort by Radius 23
Example code 24
Gravitational attraction Calculation Process 25
Worked Example 25
Step wise refinement 28
Pseudo code 29
Saving and Loading 31
Outline 31
Pseudo code 31
Event Driven Aspects 32
Mouse Down 32
Mouse Up 33
Mouse Move 33
Clear all click 34
Play clicked 34

Add planet clicked 35
Delete planet clicked 35
Information tool click 35
Save Button click 35
Load Button click 35
Show Advanced checkbox changed 36
Volatile Storage 36
Class Planet 36
Global Variables 37
Technical Solution 38
Testing 64
Unit Testing 64
Unit testing screenshots 71
System Testing 118
White Box Testing 118
Black Box Testing 132
Evaluation 134
Self-Evaluation 134
User feedback 137
Evaluation of feedback 138

Analysis

Background

The Long Eaton School is a specialist science academy and, in particular, has a large focus on astronomy and astrophysics. The school has its own observatory and offers GCSE astronomy as well as doing astrophysics as the optional module in the A level physics course. To complement the excellent facilities there are astrophysics specialists in the science department.

Of course, within the curriculum, students study gravitational forces. In GCSE physics it is mostly about gravitational attraction however in A level and also the GCSE astronomy it is covered in more detail looking also at gravitational fields etc.

As well as official teaching in lessons the school hosts several extracurricular astronomy events each year and has a dedicated astronomy club for students and a society for the wider community. These make use of the observatory and often have guest speakers and demonstrations on astronomy.
The school observatory also hosts evening when local scout groups etc. visit the observatory and use the telescope.

To go along with this observation through the telescope the school does a lot of explanation about gravitational attraction, planet formation and the composition of astronomical bodies. Despite the need to explain these systems the school lacks a system that they can use to easily explain the concepts to a wider audience.

Problem

Having spoken with the specialist astrophysics teachers they have asked me to create a way of them demonstrating the motion of celestial bodies as a result of gravity. What I aim to do is to make teaching and explaining the effect of gravity simpler by creating a tool that can simulate how planets, stars etc. move. The purpose of the system is to be educational so it must be intuitive and easy to use. It must also accurately map the movement of bodies based on the distances between them and their masses.

Currently teachers can explain motion as a result of gravity by drawing diagrams, sometimes there are animations but these take time to make and only show one scenario. Another option is, of course, mathematics, but this takes time and it isn't very easy to understand if it's someone's first time on the subject.
I aim to eliminate these problems by creating a system that does all of the maths for the user and then shows them a graphical simulation of what would happen.

Users

As I and the physics teachers see it there are 8 different types of user that this system needs to be used by:

GCSE Physics students – these students really just need to understand the basic concepts, the tool would be used as more of a gimmick to get the students interested. For this reason, the program needs to be easy to use and mostly graphical as if students just want to try to create a little orbit simulation they don't want to be messing around working out positions and velocities.

GCSE Astronomy students – the Astronomy students need to understand slightly more about gravitational fields and in particular how a radial field atracts. In a GCSE astronomy lesson the tool would still be slightly gimmicky but it is likely that the students would take a deeper interest and

attempt to create more complex simulations. For this reason, the program needs to have advanced functionality that is still reasonably clear but is in some way hidden upon initial use.

A level Physics student – these students are likely to be the most advanced users and will be using the program to its full potential of simulating complex systems with multiple objects and perhaps trying to demonstrate advanced examples such as binary star systems with Lagrange points. Because of these advanced needs there needs to be advanced options and a way to see details about a simulation so as to allow the user to do calculations on planet trajectories etc.

Teachers – A teacher will be using the program primarily as a demonstration tool. They may be demonstrating a complex system involving multiple objects for A level students or just a simple orbit for a GCSE student. The teacher has all of the needs of the above three users above, the main difference is that the teacher will likely be trying to explain the simulation as it goes on so it may be an idea to have a way of controlling the simulation speed.

Astronomy club – the astronomy club is a group of students who do extracurricular activities surrounding astronomy and astrophysics. The group could make use of the tool but it would be more as a fun gimmick than a legitimate tool as the ages of the students tends to be slightly lower, so the more complex functionality may currently be beyond them.

Astronomy society – the society is more of a wider group of the general public, as well as older students, who use the school's facilities and do more advanced activities related to astronomy. The tool may be something that they wish to play with but the teachers and I decided that this was more as a gimmick that a proper tool. Nevertheless, it should still have the functionality to satisfy advanced needs.

Group visits – Often local scout, beaver and other such groups come into the school to use the observatory. It was suggested that the tool could be used whilst students were waiting to use the telescope, or as a secondary activity to fill time. Again, the tool would be playing a role more as an interesting gimmick than a legitimate tool.

Open evenings – these evenings, for example stargazing live, involve a wide community outreach where students are invited to bring their friends and family to see demonstrations and talks about astronomy. The evenings usually involve guest speakers and exhibits around the school and observatory and the teachers have suggested that have a small tool to exhibit the motion of objects as a result of gravity may be something interesting for people to play with whilst they wait to use the telescope or whilst they are looking around the school.

The A level physicists need an accurate but also customizable model with lots of data input and output. However, at the other end of the spectrum the GCSE physicists need a much simpler and more graphical solution. Because of these varied needs I will have to consider how I exhibit data and how the user inputs and receives outputs from the system.
Although there is a variety in the needs of the users I think that it is possible to deliver a solution that is acceptable to all.

Calculation procedure

Given the inputs of the masses and locations in space of several objects it should be possible to calculate their movements. It is important to note that this will be calculated in a Newtonian manner, meaning that all laws of general relativity are ignored and the speed of Light and the speed

at which gravitational forces take effect is infinite. Doing it this way is not completely accurate, but on the scale of our solar system the differences are negligible to the extent that when calculating orbital paths for satellites and craft in space companies almost always use Newtonian gravity.

Gravitational Force

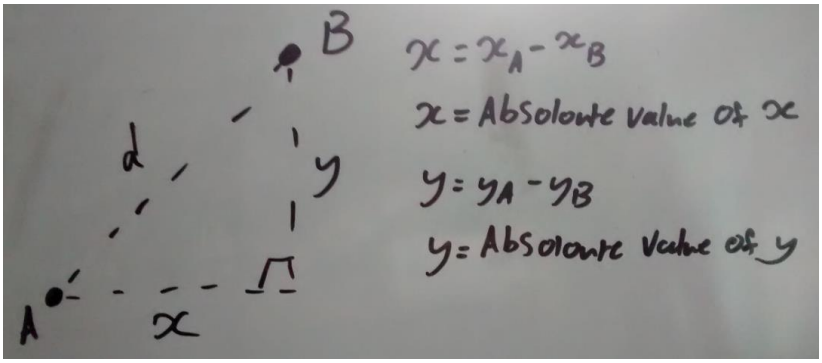
With the mass of the planets known it is now possible to use newton's equation to calculate the gravitational force of attraction:

$$F = \frac{m_1 m_1 G}{r^2}$$

G = Gravitational constant = 6.6740831313131x10-11N m2/kg2
m = mass of planets 1 and 2
r = Distance which will be calculated in the following way:

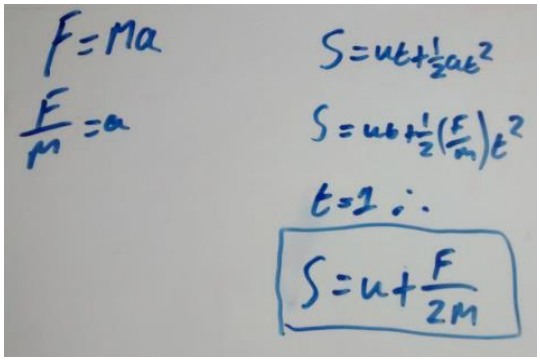
Distance calculations

At multiple points in the program I will need to find the distance between two points, for this I will simply use Pythagoras's theorem as I will have the X and Y positions of both objects and will therefore be able to find the difference in the X and Y.

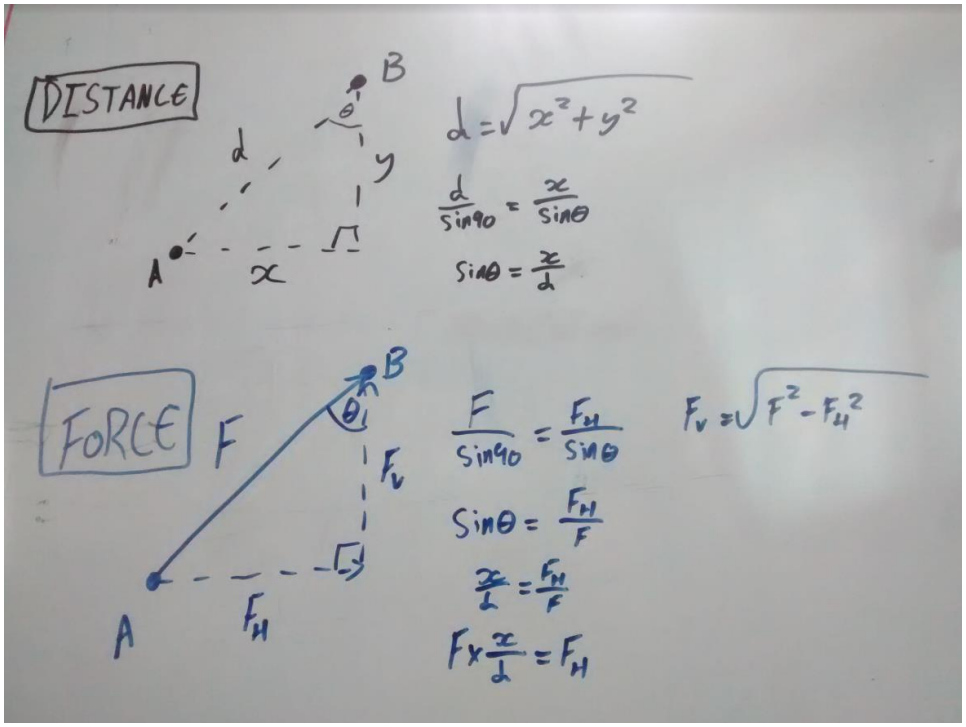


Displacements

With the force known I now need to know the displacement of the planet. This will be found using SUVAT. By substituting F = ma into one of the SUVAT equations (as shown in the following image) the displacement can be found.



However, it's not as simple as that, as well as the scalar quantity of distance I also need to know what direction the planet should go. For this I need to break the planet's motion down into the X and Y. The following image shows how the force will be split into an X and a Y component:



Firstly, the distance between the planets (d) is found. Then, the angle theta can be calculated by using the sine rule.
Now that theta is known we can draw the same right angled triangle containing planets A and B but instead of distance the lengths of this triangle represent force. The total force (hypotenuse) is found using the formula as discussed earlier (see Gravitational Force).

Again, by using the sine rule a value can be found for the horizontal force. As the sine and inverse sine cancel we see that force multiplied by the distance of that side, divided by the length of the hypotenuse gives the force of the horizontal.
The vertical force could also be found in this way or by using Pythagoras' theorem as is done in the image

The final thing to do is to substitute my values into the equation that was found earlier:

$$S = u + F(2m)^{-1}$$

Now, I should have a displacement in X and Y which can be added to the planets current coordinates to update its location.

System Requirements

These are the things that the final solution should include:

-Have an animated GUI showing the moving bodies
-Change Mass of the bodies by changing the density and radius
-Have an initial velocity system, this will allow the users to create orbits easily by providing a perpendicular velocity to the pull of gravity.
-Change colour of bodies to easily identify the different bodies that the user creates.

-Clear and intuitive user interface for first time users, the program is designed to educate so it is important that the solution is intuitive and first time users are not put off by a long-winded manual and confusing controls.
-Have the majority of inputs performed graphically so that it is easy for the user to understand the forces acting on the planet.
-Have an option to save simulations and also to load them later
-Have advanced functionality that allows advanced users to create more complex models, however this should not interfere with the simple user interface for the majority of users.

Is there a need for a bespoke system?

In this section I'm going to talk about some off the shelf programs and go through the advantages and disadvantages of using them. Then I will summarise what the best solution is and if I think that none of the below are appropriate, whether it is necessary to create my own solution.

Grav-Sim

This is the overarching name for three different simulators: FastSim, GravSim and FineSim. FastSim is the lightweight version and FineSim the highly accurate and detailed version, GravSim is something of a middle ground. All three programs are free and available via the internet. The advantage of these programs is that there is a variety in the detail and complexity that the user may wish to peruse, however the issue is that these programs are all console based. That is fine for advanced users who are more looking for the data retrieval aspect but it's not very useful for a first time user. It would be intimidating and too complicated, users would likely not know what the different values meant and it would make them not want to use the program.
One advantage of it, however, is that it does model in three dimensions which most of these solutions do not. However that is not a must have and would likely confuse less adept users.

Gravity Simulator – Test Tube Games

www.testtubegames.com/gravity.html

The main issue with this is that it is on a games website and so is blocked on the school system. The school could unblock the site but this would give access to other games. As well as this it requires flash which means that the system admins have to make sure it is constantly up to date. This would be a waste of time.

Newtonian Gravity Simulator - Sympatico

http://www3.sympatico.ca/michael.enns/

This is an excellent website which provides a very good simulator. It runs well on the school system and contains no adverts that might make it inappropriate for use by students. The way that it creates models however is not exactly what the physics faculty are looking for. This program works by having specific models. The user can alter the mass, x position, y position and velocities by editing text but I don't think it would be a very easy to use system for someone that is very new to the idea of velocity and vectors. As well as this, the user cannot create an entirely original model. They can only create a random simulation that does have a rather impressive number of objects but having spoken to the physics department they are looking for something more programmable by the user.

To summarise the options above; these three examples represent a wide range of solutions but they all have similar flaws.

The ‘game’ style programs such as the first example listed above often have inappropriate adverts and are often blocked on the school system. As well as this they usual require flash or java which I cannot be sure will be up to date on the school system. These solutions also fall down on the ability of the user to save their simulations.

The second style of program seems to be a tool for a professional they clearly require more processing power as they often run locally on the computer as a standalone program rather than the flash based websites such as in the first set I talked about. These programs lack the accessible graphical user interface that is a primary need for younger students but they are highly customisable and usually model in 3D or handle more advanced simulations more easily as this is what they are designed to do; GravSim, for example will easily handle 10000 objects at a time. Simulators that have the functionality of these more advanced programs but also a graphical user interface usually require reasonably high spec computers to run all of the graphic calculations or cost a reasonable fee themselves. Anything that costs money is pretty much ruled out from the list of options as the physics department are looking for a free solution.

While there are middle grounds between these two, such as the third program that I have discussed above there are usually issues with it such as a lack of customizability or cost. For this reason, I think that there needs to be a bespoke system created that is going to fit the needs of the physics department. Otherwise they will end up with a solution that falls short of some of their primary needs which will lead to either a lack of use or a poor quality of teaching from the staff using it and a low understanding of the students.

Objectives

System Objectives

Have a smooth animation that does not flash and is not stop start

Allow the user to create and delete planets on the screen

Have a pause and play mode so that the user can edit their model and then play it to see what happens

Have most of the functionality in one window so that the user is not confused by many different forms opening and closing.

Processing Objectives

Map the movement of, at least, of five bodies

Allow the user to define an initial velocity for a body that will be stored as a vector

Realistically mimic movement due to gravity of the bodies

Resolve a minimum of six forces into one vector

Calculate mass from density and radius for each planet

User Objectives

Allow the user to alter the radius, density and colour of a planet

All text boxes, buttons etc. should be clearly labelled to make it intuitive

Allow the user to delete one planet at a time

Allow the user to delete the entire simulation at once

Have an additional advanced menu for more knowledgeable users, this should not intrude on the base UI.

Design

Potential Solutions

C++

C++ is reasonably good for graphics and it is definitely possible to create a program with the 2D graphics I intend to have. However, I have no experience with C++ and it is not very similar to the language that I am most familiar with, VB.net. It would take a long time to learn C++ because of its dissimilarity. I get the impression from various tutorials that it is quite a difficult language to learn, one video tutorial had almost 40 episodes before it reached the sort of level of graphic complexity that I will be going for.

That said C++ is a popular language and there is a good deal of high quality free support available. It also uses the visual studio IDE like VB.Net, which means that I do not have to go out and find a new piece of software.

Java

Java is not a very intuitive language and it would definitely take a lot of time for me to get used to it, as I've no experience with it.

Java is however portable because of the Java virtual machine but this is not a necessity for the school as they only use windows. As I understand it the graphic abilities are there but difficult to access as there is no graphical user interface with the IDE like there is in visual studio for VB.net and C#. Another key point to consider is that Java is frequently updated and so it would be up to the system administrators to keep the virtual machine up to date.

Flash

Flash is likely not in depth enough for what I want to do, the system needs to be highly customizable for each user which may be doable with flash but it needs to be an intuitive design which flash probably would not allow.

It is also reasonably expensive, however it is already on the school system therefore that problem is somewhat overcome. In terms of compatibility it would be able to work on mac and PC as there is a version of flash for both, however, again this is not a huge issue as the school system is windows only.

VB.Net

The main selling point of VB.net is that I have a lot of experience with it compared to any other language. I have used it frequently for the past year and I am very familiar with it. I don't have that much experience with using the graphics library however, I have written a few programs and I think that I know enough to make the program work. It is also very easy to create windows form applications in visual studio. The other advantage of VB.Net is that it is very widely used and so there is a lot of free support. The only slight disadvantage that I can see with using VB.Net is that it isn't portable to other operating systems but seeing as the school use windows exclusively that isn't particularly an issue.

Another issue that I may have is that vb.net is not designed to be write frequently updating graphical programs so I may find that I have issues with jumpy animation or lag. Another large advantage of VB.net is that it is an object oriented language which works well with event driven programs. It is likely that my program will work in these ways and so it may be a good idea to use VB.net which works well with these sorts of systems.

Python/Pygame
Pygame is a set of libraries for python that can be used to create video games; it includes functionality for graphical user interfaces. I have experience using python but, as I understand it, Pygame is a reasonably extensive add on to python meaning there could be a good deal of new syntax to learn. Although it would not be like learning a language from scratch it wouldn't be as easy as if I used VB.net which I already know and am confident with.
The main advantage of Pygame is that it's running on a very lightweight base which means that it is quite responsive unlike something like VB.net that isn't inherently designed to run a fast updating graphics program.

Processing
Processing is not commonly used; it was built for the electronic arts new media art and visual design communities. It is designed to be an easy to learn first time language with an emphasis on graphical programs. It builds on Java but has simplified syntax as well as a graphics user interface. Because of all of these things it would probably not take me as long to learn as some other languages. In addition, the fact that it is intended to be used to create graphics is exactly the sort of thing I would need.
The main disadvantage is that it is not a very widely used language and I will struggle therefore to find the sort of support that I would get from a more widely used language such as VB.net. Processing is free and open source but I would have to find and download the IDE etc.

C#
Although I do not know C# it is very similar to VB.net which I have a lot of experience with, it would therefore probably be the easiest program, of those that I don't already know, to use. The graphics on C# are very good and easily accessible due to the drag and drop style IDE in visual studio. Disadvantages to C# are that it is interpreted and so is less efficient than C++ for example. It also has very poor compatibility with anything other than windows due to its integration with .Net. This however is not an enormous issue, as I have said, because the program need only work on the windows based school system.

My choice:

For me the best tool to use seems to be VB.Net, the graphics properties that it does have are more than capable of doing what I need and I am already very familiar with it and its graphics library so once I have designed the program I can immediately get to work on it.

Navigation Plan

Most of my program will operate out of one form to keep it clear to the user where they need to be looking and to be clear about how they affect things on the screen. I think that if users are searching through different forms looking for an option they are going to be less inclined to play around with the model in a light-hearted way as I and the physics teachers want. This program needs to be lightweight and playful but also packed full of useful features for more advanced users.

The program will open into the Canvass window and the only deviations from this will be in using the load and save buttons, which will open file explorer windows so that users can decide where they want to save their simulations or where they wish to load their simulation from.

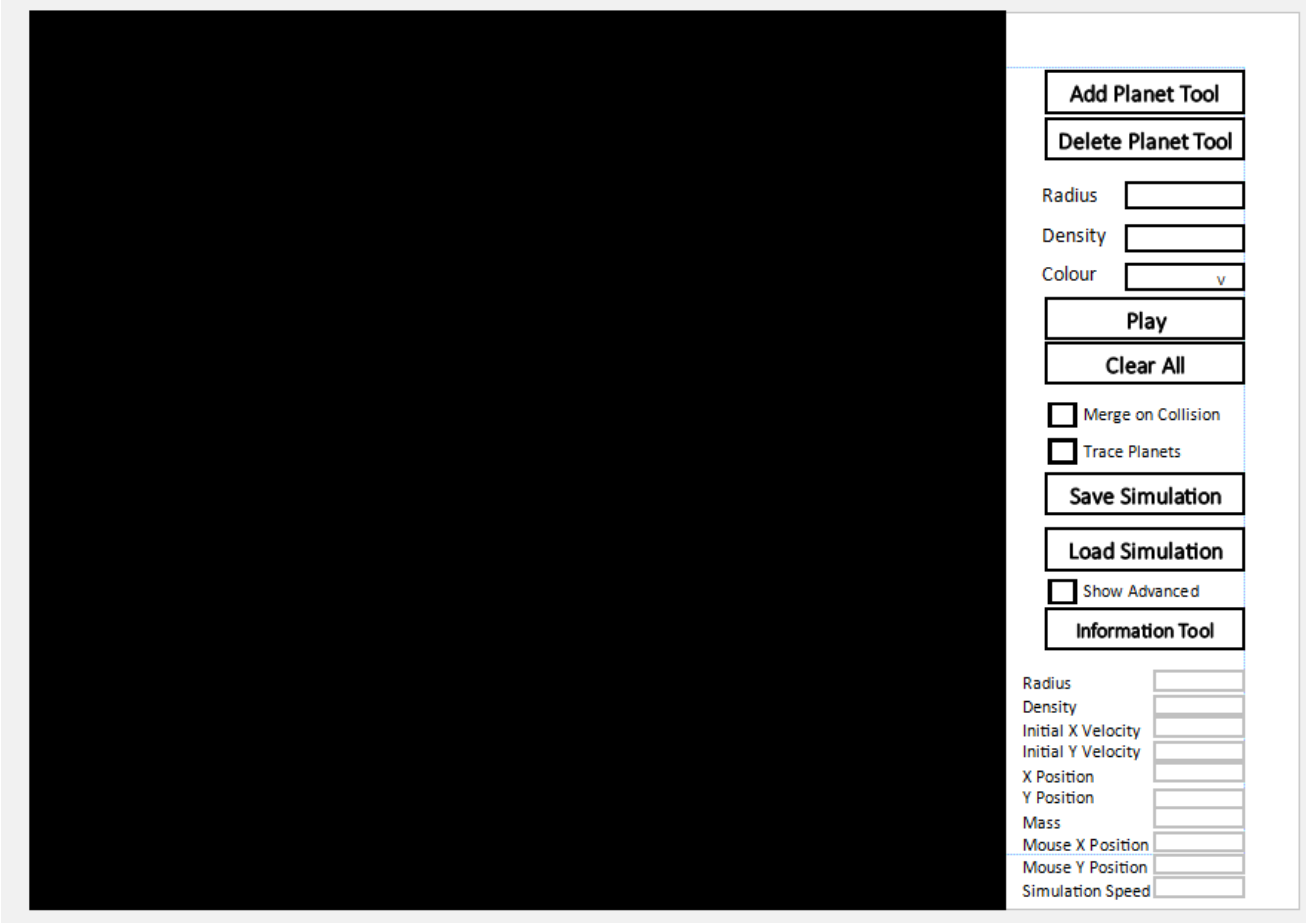
Form Design

My initial plan for the form is to have a black canvas to the left which the user can interact and draw on by clicking on it. On the right hand side of the screen I want to have a tool bar with various options and controls for the simulation and also what happens when they click on the canvas e.g. create a planet, delete a planet.
The program is going to work by having two different modes, the first will be the paused mode which will be as shown below, all buttons will be usable and the user can add and delete objects to the canvas. The second mode will be the running mode, in this mode all of the buttons will be disabled apart from the pause/ play button and the clear all button. The objects on the canvas will be animated but the user will not be allowed to add more objects or delete them without pausing the simulation.

The Advanced tab will be shown or hidden by the use of the show advanced checkbox. If it is ticked, then the advanced tab will show if it is not ticked then it will be hidden. This is to reduce visual complexity when the user first starts the program as, I feel, that if a user launches the program for the first time and is confronted by a wall of different options they may feel overwhelmed. This way advanced users can access the features that they want but they are hidden from sight for the users that do not require them.

Without advanced tab:

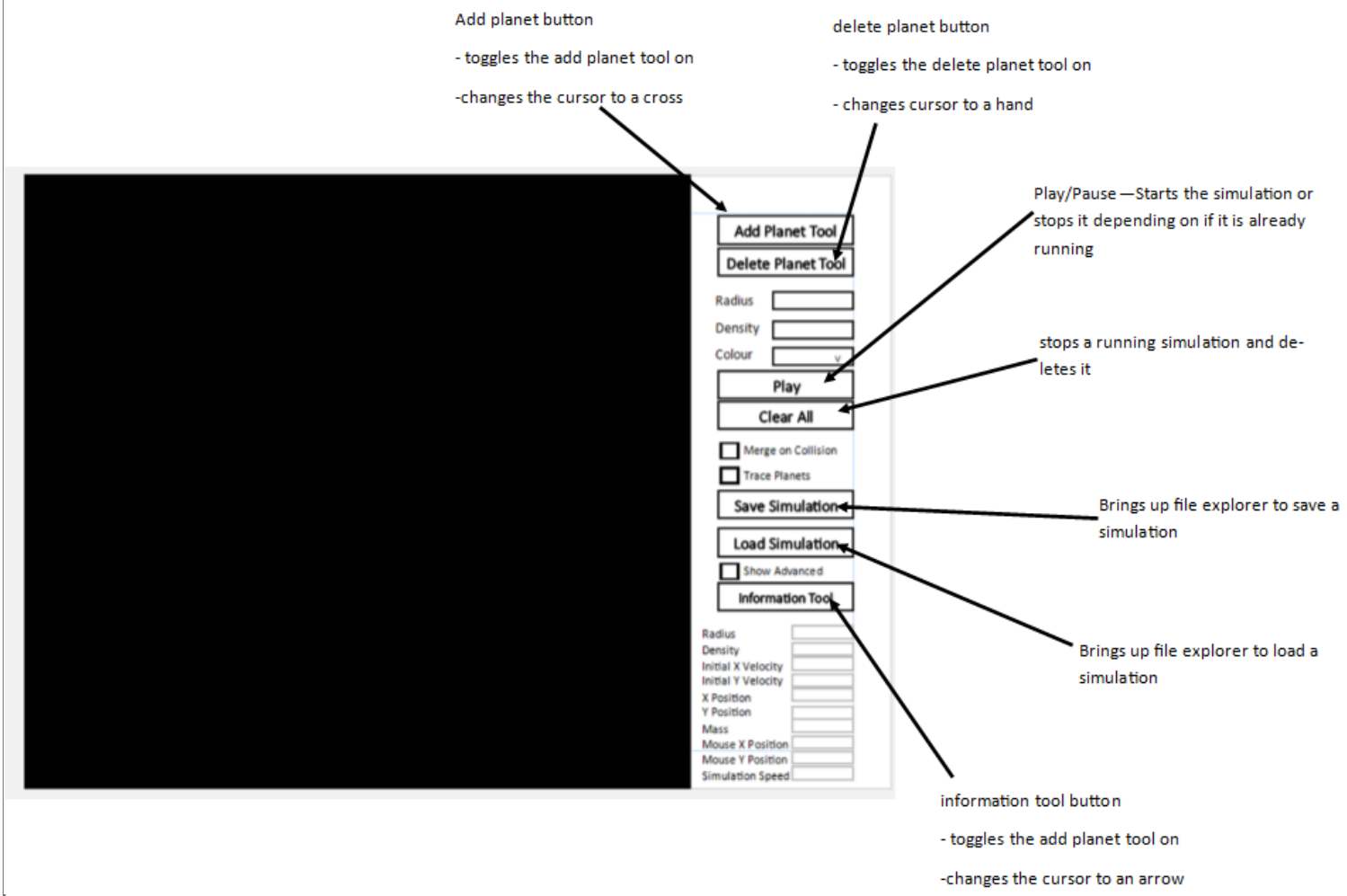
With Advanced Tab:



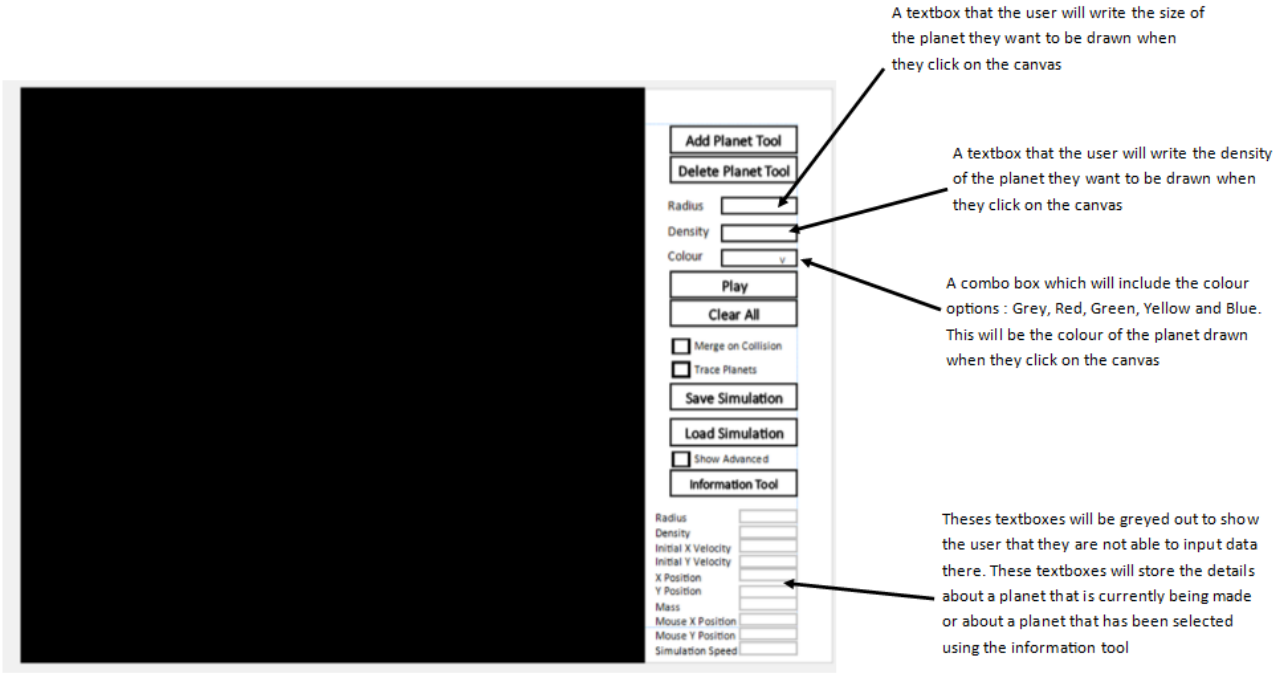
Within the paused mode of the program there will be a few more options that affect the way the program acts. For example, having different tools selected will change what happens when the user clicks on the canvas.
The add planet tool will allow the user to create a planet when they click on the canvas, the details for the planet that is created will come from the textboxes and combo box on the right hand side of the screen. The delete planet tool will delete a planet on the canvas if it is clicked on, finally, the information tool will give details of a clicked on planet in the advanced tab (if it is showing).

As well as the show advanced check box there are two other check boxes, trace planets and merge. Ticking trace planets will mean that when the simulation is running a solid line will be drawn behind the planet to show its motions. The merge checkbox is also for the running simulation only and if it is ticked then when two planets touch each other they will become one planet. Although this is not the most realistic version of an astronomical collision it does allow the user to see how objects may start to clump together in a large gas cloud for example.

Buttons with brief outline of purposes



This is just a brief outline of what the buttons will do, a more in depth description will follow in the IPSO table



Tools in the paused mode

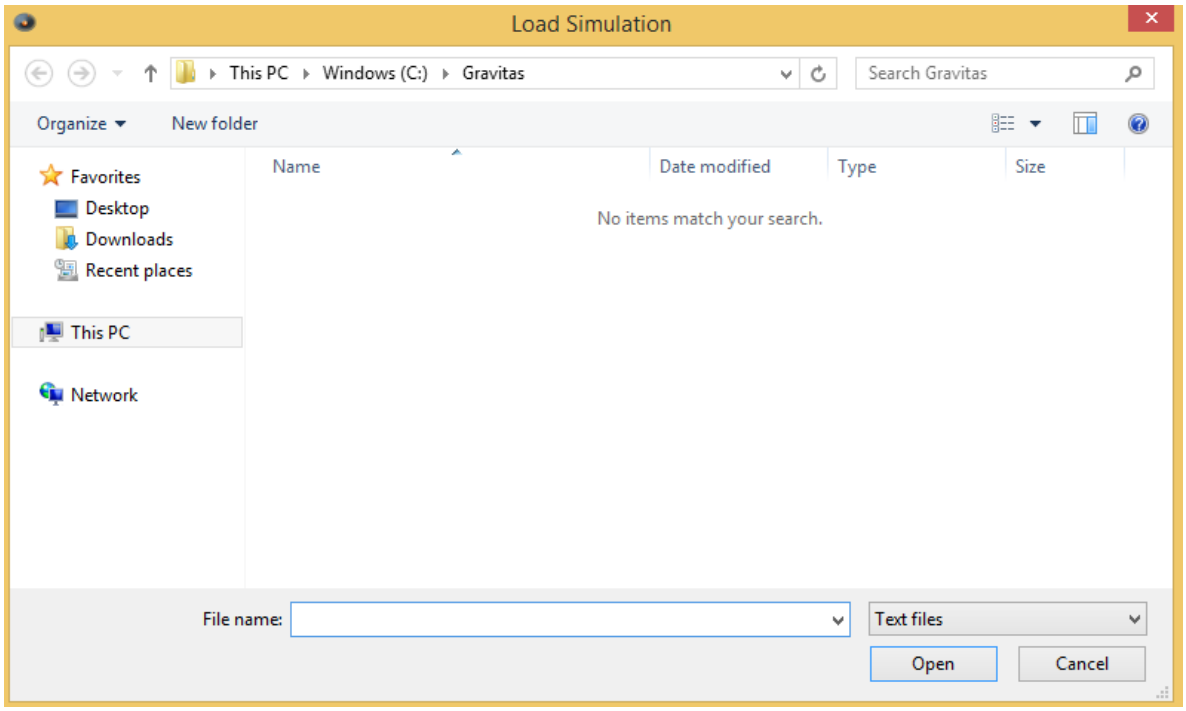
I always want it to be very clear which tool the user has selected at any given time, I intend to do this by disabling or 'greying out' the button. This should clearly demonstrate to the user that they have that button selected and so cannot re-select it, it also makes their other two options stand out more clearly.
In a similar vein, I want the three tool buttons as well as the save and load buttons to be disabled when the simulation is playing. This is because the user cannot use them whilst the simulation is running, if it were to work that the user could click on the button and nothing would happen I think it would lead to confusion and frustration as to why the program did not appear to be working properly. However, by having the buttons disabled the user can see that it is an obvious design choice for these buttons to be useless in the programs current state.

Form Initialisation

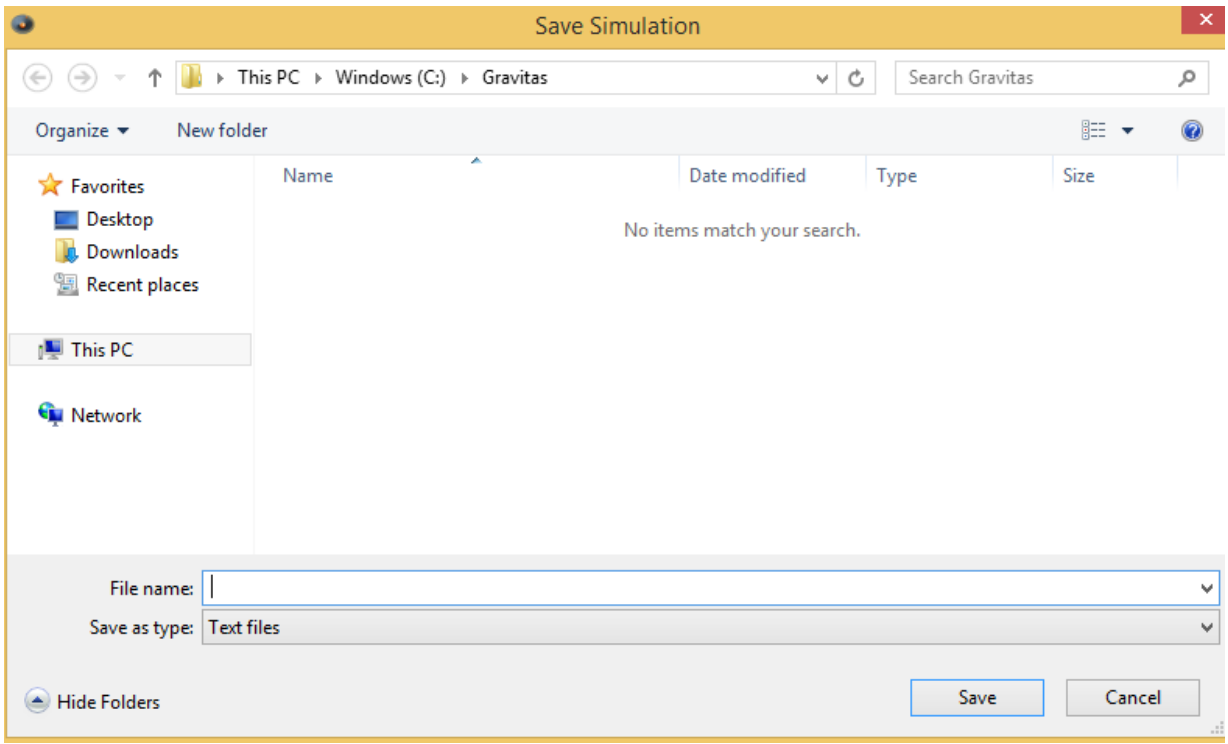
When the above form fist loads there will be pre-given values in the radius, density and colour boxes, the program will also start with the add planet tool selected. This means that the user will immediately be able to start clicking on the canvas to create a simulation without having to look at the menu bar on the right.
The initial value for radius and density will be 20 and 5 respectively and the initial colour will be Grey. There is little reason for these values other than they are reasonably low which will probably make it easier to create a complex simulation than if the user started off with very high values.

Save and Load Dialogue

Load:



Save:



Input, Process, Storage, Output Tables

For Canvas

Input	Process	Storage	Output
Add Planet Button Clicked			-Disable Add planet button -Enable delete planet button -Change cursor to cross
Delete planet button clicked			-Disable delete planet button -Enable add planet button -Change cursor to hand
Play button clicked (where play button text = pause)	Stop simulation		-enable delete planet button -enable add planet button -enable save simulation button -enable load simulation button - select information tool
Play button clicked (where play button text = play)	Start simulation (see running simulation flowchart)		-Disable delete planet button -disable add planet button -disable load button -disable save button -disable information tool
Clear all clicked	**Are you sure message** **If yes:** -Stop simulation -delete all planets -redraw graphics **If no:** Continue running simulation		**Are you sure message** **If yes:** enable delete planet button -enable add planet button -enable save simulation button -enable load simulation button -disable information tool
Save simulation clicked			Show save simulation dialogue

Load simulation clicked			Show load simulation dialogue
Trace planets checked	**If simulation is running:** Continue with simulation but do not clear canvas on draw graphics		
Trace planets unchecked	**If simulation running:** Clear the canvas when redrawing planets		
Merge checked	**If simulation is running:** Check if planets merge after calculating new positions		
Merge unchecked	**If simulation is running:** Stop checking if planets should merge		
Show advanced checked			Show advanced panel **If simulation is not running:** Select the information tool
Show advanced unchecked			hide advanced panel **If simulation is not running:** Select the add planet tool
Numeric up down changed	**If simulation is running:** Change speed to speed in numeric up down		

The following are all different variations on the state of the program when the user clicks on the canvas. If a variation is not listed (such as a click with add planet whilst the simulation is running) then it is because it is not possible (in this case because the add planet tool is disabled on simulation start) or because it falls into another event.

Click on canvas events where:

Condition	**Process**	**Storage**	**Output**
Simulation paused and Add planet selected	Create a planet with radius and density specified in textboxes on the right and with		Update graphics with this new planet

	a centre around the mouse click		
Simulation paused and information selected and clicked on planet	Put details of that planet in the advanced tab		Draw ring around selected planet
Simulation paused and delete planet selected and clicked on planet	Delete clicked on planet		Update graphics
Simulation paused and information selected and not clicked on planet			Update graphics to remove a selection ring if one has been drawn Clear advanced tab

Other mouse events

Many of the main inputs when creating a simulation will come via different mouse actions using the add planet tool. Clicking on a planet will draw it on the canvas with the initially specified radius and density (and of course location as the place the user has clicked) however the initial velocity will be indicated by the user dragging the mouse in the opposite direction of travel.
Doing so will draw a line from the centre of the created planet to the new position of the cursor, the length of this line will specify the magnitude of the vector and the direction will be the opposite of the direction the planet will travel, this is because I want to have a slingshot effect. The user should think of it like pulling back a piece of elastic so that when it is released the planet will fly off in the opposite direction to which the planet is pulled. The velocity is finalised when the user lifts up the mouse button.

By adding planets in this graphical manner, the user can very quickly build up a complex model without having to understand much about mechanics. All they have to understand is the reasonably intuitive concept that if you pull a piece of elastic one way then it will fly the other when released.

(The reason that these are not listed in the IPSO tables is that I felt it was too complex a set of conditions to easily be put into a table)

For Save Simulation Dialogue

Input	Process	Storage	Output
Save clicked	Save simulation with file name given as text file	Write to text file with name given: Simulation data	

For Load Simulation Dialogue

Input	Process	Storage	Output
Load clicked	Load simulation with file name given as text file	Read from text file	Draw simulation on canvas

Validation

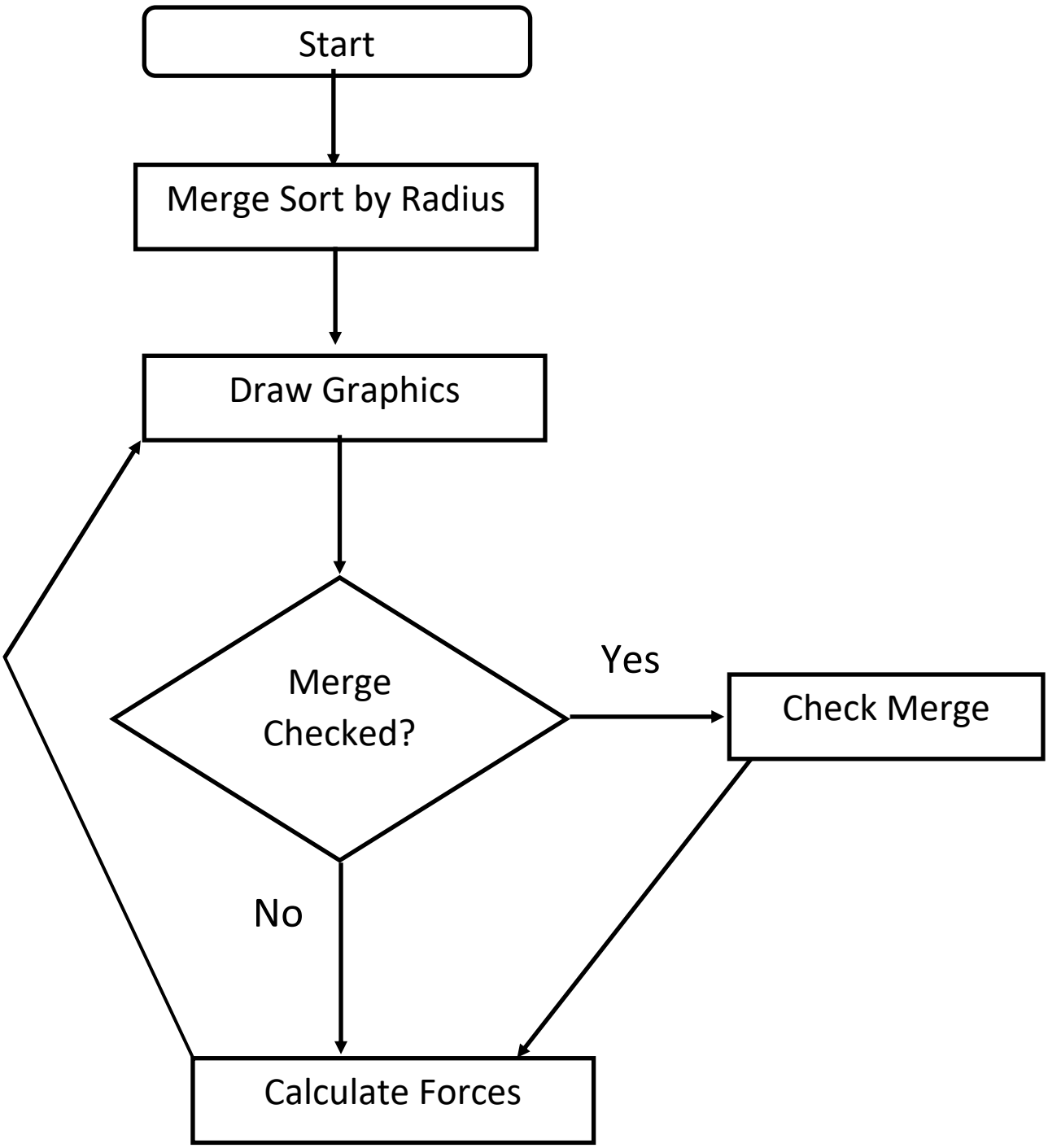
The main text inputs are the radius and density, for that reason these must have validation to prevent the user entering text etc. The Radius will be checked to ensure that it is an integer in the range 1 to 120 and the density will be checked to make sure it is an integer in the range 1 to 1000.

Other inputs are the simulation speed which will be a numeric up down counter, this will be limited to a range of 0 to 10 and the counter itself should have inbuilt validation to prevent the entry of a non-integer.

Another input is the colour combo box but that too has inbuilt validation in that the user can only select a preapproved input.

Flow chart of running simulation

As the main procedural part of the program, this is how the actual running of the simulation will work. In the following sections, I break down the different sub routines that are referred to here (e.g. Draw Graphics).



Graphic drawing process

This section will discuss what will happen during the 'Draw Graphics' stage of the flowchart above.

The graphics will work by having the canvas as a graphics object, drawing graphics will cause the canvas to be cleared black and then all of the current planets will be drawn on the canvas.

Pseudo code

```
CLEAR Canvas
FOR ALL Planets:
        IF Planet.InUse = TRUE:
                DRAW ELIPSE ON CANVAS(CENTRE = (planets(n).X, planets(n).Y), RADIUS =
planets(n).radius, FILLCOLOUR = planets(n).colour)
        END IF
END FOR
```

Check Planet Merging

In reality when two planets get close together and collide some of them will fuse and some of them will be ejected, however as the user is not looking for an incredibly true to reality version they simply want an option to have the two planets stick together when they touch.
To do this I will be checking for every planet, how close they are together and if they are close enough then the new planets density and radius will be calculated from the two parent planets.
In this, I will also be making sure that momentum and kinetic energy are conserved as this is an elastic collision.

Pseudo code

```
For every planet
        if planet.inuse = true then
                For every other Pln
                    if pln.X > (planet.x – planet.radius) AND pln.Y > (planet.Y – planet.radius)
                        Planet.InitialXVel = ((planet.InitialVelocity.Xcomp * planet.mass) +
(pln.InitialVelocity.Xcomp * pln.mass)) / (planet.mass + pln.mass)
                        Planet.InitialYvel = ((planet.InitialVelocity.Ycomp * planet.mass) +
(pln.InitialVelocity.Ycomp * pln.mass)) / (planet.mass + pln.mass)
                        Planet.Radius = planet.radius + pln.radius
                        Planet.density =(planet.mass + pln.mass) / ((4 / 3) * Math.PI *
Math.Pow(planet.Radius, 3))
                        pln.InUse = False
                        pln.Radius = Nothing
                        pln.Density = Nothing
                        pln.Xposition = Nothing
                        pln.Yposition = Nothing
                    End if
                End For
        End if
End For
```

The above code will find out if two planets, pln and planet, are close enough that they must merge by seeing whether the centre of one planet is within the radius of the other. If so, the new radius for

this combined planet is the two radii added together and the velocity is calculated by taking the momentum of the two planets, adding them together, and then dividing by their masses to get the resultant velocity:

$$\rho = mv$$

$$v = \frac{(m_1 v_1) + (m_2 v_2)}{(m_1 + m_2)}$$

Sort by Radius

As my screen will have to cleared black and redrawn every cycle there is a possibility that the screen will flicker slightly if the simulation is being asked to run too quickly. The reason that a picture flickers is that it is redrawing the image a very short time after it has been originally drawn. To reduce this potential flicker, I will be sorting the planets in ascending order by radius. This means that the largest objects on the screen will be drawn first and so the flickering will occur mostly on the small objects. Whilst this is not ideal it will be less noticeable than having a flicker on the largest objects.

To sort the planets I will be using a merge sort, the planets will be sorted by radius so that the planets are drawn in decreasing order of size.

To begin with, the merge sort algorithm has two key steps:

- Divide – this is where an array with length greater than one is split in two, this process is repeated until all of the sub-arrays are of size 1
- Conquer – The sub-arrays are compared and the values sorted until all of the values are back in one array and are now all in the correct order

If we have an array with values (56, 3, 19, 37, 3, 24, 55) then it will be sorted in the following way:

Array A = (56, 3, 19, 37, 3, 24, 55, 11)

Array B = (56, 3, 19, 37) Array C = (3, 24, 55, 11)

Array D = (56, 3) Array E = (19, 37) Array F = (3, 24) Array G = (55, 11)

H=(56) I=(3) J=(19) K=(37) L=(3) M=(24) N=(55) O=(11)

D = (3, 56) E = (19, 37) F = (3, 24) G = (11, 55)

B = (3, 19, 37, 56) C = (3, 11, 24, 55)

A = (3, 3, 11, 19, 24, 37, 55, 56)

The original array is broken down into arrays of size one (H, I, J, K etc.) and then these are put back together in increasing order.

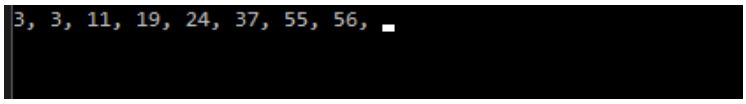
Example code

This VB code is some that I developed for a different project but it follows the same logic as the above explanation. The array is broken down into sub arrays of size 1 and then put back together in descending order.

```
Public Sub MergeSort(ByVal ar() As Integer)
        DoMergeSort(ar, 0, ar.Length - 1)
    End Sub

    Private Sub DoMergeSort(ByVal array() As Integer, ByVal Min As Integer, ByVal Max
As Integer)
        If Min >= Max Then
            Return
        End If
        Dim length As Integer = Max - Min + 1
        Dim middle As Integer = Math.Floor((Min + Max) / 2)
        DoMergeSort(array, Min, middle)
        DoMergeSort(array, middle + 1, Max)
        Dim temp(array.Length - 1) As Integer
        For i As Integer = 0 To length - 1
            temp(i) = array(Min + i)
        Next
        Dim m1 As Integer = 0
        Dim m2 As Integer = middle - Min + 1
        For i As Integer = 0 To length - 1
            If m2 <= Max - Min Then
                If m1 <= middle - Min Then
                    If temp(m1) > temp(m2) Then
                        array(i + Min) = temp(m2)
                        m2 += 1
                    Else
                        array(i + Min) = temp(m1)
                        m1 += 1
                    End If
                Else
                    array(i + Min) = temp(m2)
                    m2 += 1
                End If
            Else
                array(i + Min) = temp(m1)
                m1 += 1
            End If
        Next
End Sub
```

I put this code into a simple console application and setup the array with the numbers from the example above, this was the given output:



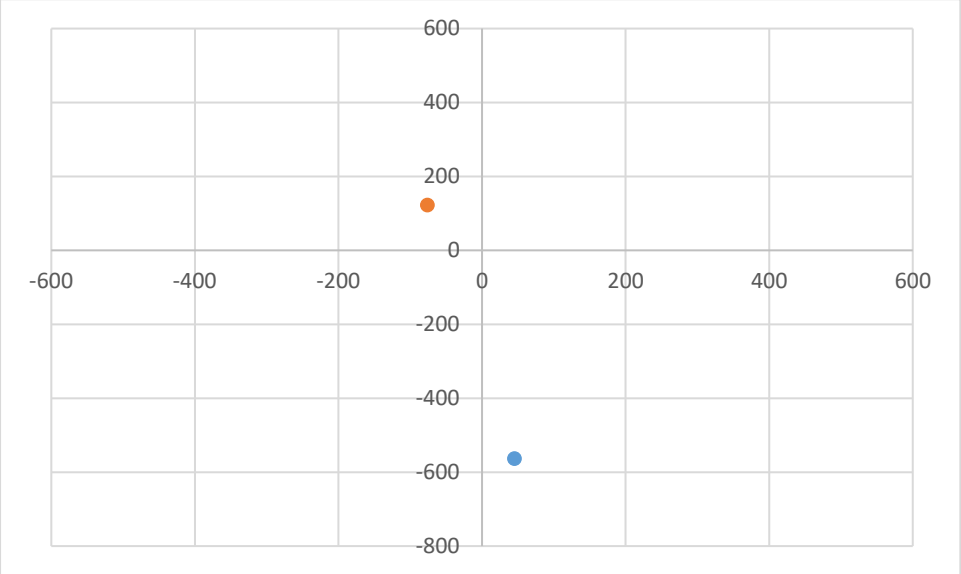
The only extra thing that needs to be done in the final program is to rematch each planet with it's radius in the now sorted array.

Gravitational attraction Calculation Process

Worked Example

This is a worked example of the calculation process that I explained in my analysis section. It demonstrates how a human would work through calculating a planet's new position.

Planet1	Planet2
Radius = 29	Radius = 50
Density = 10	Density = 12
X Position = 45	X Position = -76
Y Position = -563	Y Position = 126
Initial X Velocity = 23	Initial X Velocity = 6
Initial Y Velocity = 1	Initial Y Velocity = -15



Volume:

Planet1= 4/3*π* radius^3
Planet1= 4π/3 * 29^3
Planet1= 102160.4m^3

Planet2= = 4/3*π* radius^3
Planet2= 4π/3 * 50^3
Planet2= 523598.8m^3

Mass:

Planet1= Volume * density
Planet1= 102160.4 * 10
Planet1= 1021604

Planet2= Volume * density
Planet2= 523598.8* 12
Planet2= 6283185

Distance:
distance = sqrt((Planet1 X position – Planet2 X position)^2 + (Planet1 Y position – Planet2Yposition)^2)

Distance = sqrt((45 - -76)^2) + (-536 – 126)^2)
Distance = sqrt(121^2 + -662^2)
Distance = sqrt(14641+438244)
Distance = 672.967

Force:
Force = (Planet1Mass * Planet2 Mass * Gravitational contant) / distance^2
Force = (1021604 * 6283185 * 6.67x10^-11) / 672.97^2
Force = 9.45x10^-4

Horizontal Force:
Planet1 Horizontal force = Force * ((Planet1 X position – Planet2 X position) / Total Distance)
Planet1 Horizontal force = 9.45x10^-4 * (45 - -76) / 672.967)
Planet1 Horizontal force = 1.699x10^-4

Planet2 Horizontal force = Force * ((Planet1 X position – Planet2 X position) / Total Distance)
Planet2 Horizontal force = 9.45x10^-4 * (45 - -76) / 672.967)
Planet2 Horizontal force = 1.699x10^-4

Vertical Force:
Planet1 Vertical force = sqrt(force^2 – Planet1 horizontal force ^ 2)
Planet1 Vertical force = sqrt((9.45x10^-4)^2 – (1.699x10^-4)^2)
Planet1 Vertical force = 9.296x10^-4

Planet2 Vertical force = sqrt(force^2 – Planet2 horizontal force ^ 2)
Planet2 Vertical force = sqrt((9.45x10^-4)^2 – (1.699x10^-4)^2)
Planet2 Vertical force = 9.296x10^-4

Displacements:

Horizontal displacement
Planet1 Horizontal displacement = Planet1 initial X velocity + (Planet1 horizontal force / (2 * Planet1 mass)
Planet1 Horizontal displacement = 23 + (1.699x10^-4 / 2)
Planet1 Horizontal displacement = 23.00008495

Planet2 Horizontal displacement = Planet2 initial X velocity + (Planet2 horizontal force / (2 * Planet2 mass)
Planet2 Horizontal displacement = 6 + (1.699x10^-4 / (2* 6283185)
Planet2 Horizontal displacement = 6

Vertical displacement
Planet1 vertical displacement = Planet1 initial Y velocity + (Planet1 vertical force / (2 * Planet1 mass)
Planet1 vertical displacement = 1 + (9.296x10^-4/ (2 * 1021604)
Planet1 vertical displacement = 1

Planet2 vertical displacement = Planet2 initial Y velocity + (Planet2 vertical force / (2 * Planet2 mass)
Planet2 vertical displacement = -15 + (9.296x10^-4/ (2 * 6283185)

Planet2 vertical displacement = -15

Update positions + initial velocity update

Planet1 X position + Planet1 horizontal displacement
45 + 23.00008495 = 68.00008495

Planet1 Y position + Planet1 vertical displacement
-563 + 1 = -562

Planet2 X position + Planet2 horizontal displacement
-76 + 6 = -70

Planet2 Y position + Planet2 vertical displacement
126 – 15 = 111

Planet1 Initial X velocity = Planet1 horizontal displacement
= 23.00008495

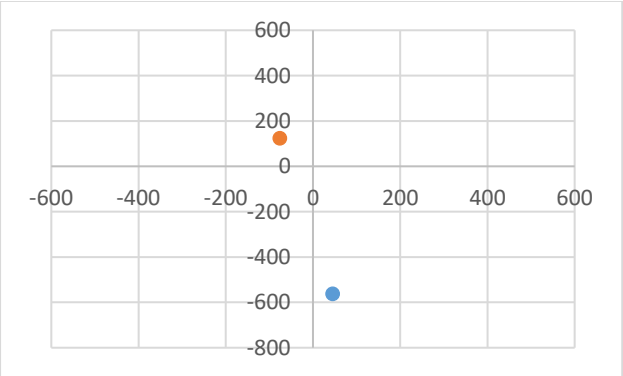
Planet1 Initial Y velocity = Planet1 vertical displacement
= 1

Planet2 Initial X velocity = Planet2 horizontal displacement
= 6

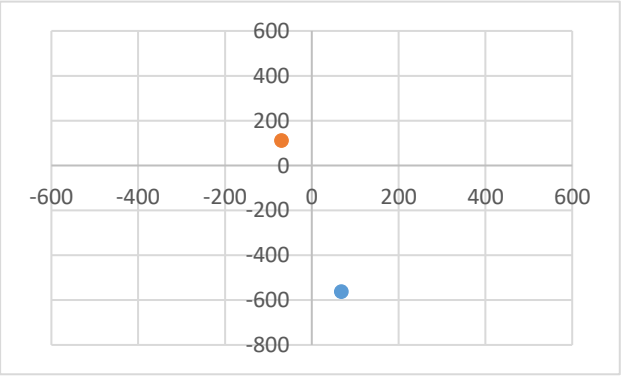
Planet2 Initial Y velocity = Planet2 vertical displacement
= -15

Planet1	Planet2
Radius = 29	Radius = 50
Density = 10	Density = 12
X Position =68.00008495	X Position = -70
Y Position = -562	Y Position = 111
Initial X Velocity = 23.00008495	Initial X Velocity = 6
Initial Y Velocity = 1	Initial Y Velocity = -15

Before:



After:



Step wise refinement

This process is how the program will find the new position of a planet as a result of movement from gravitational attraction. In the program it will run the following algorithm for every planet in the simulation.

Level 0	Calculate new position of planet
Level 1	1: Find force 2: Find Horizontal Force Component 3: Find Vertical Force Component 4: Calculate Horizontal displacement 5: Calculate Vertical displacement 6: change planets location
Level 2	1.1: Find Distance 1.2: Calculate total force value 2.1: Horizontal force = Force * (X-distance / Total Distance) 3.1: Vertical Force = $\sqrt{F^2 - F_V^2}$ 4.1: Horizontal Displacement = initial Velocity + (Horizontal Force/(2 * mass)) 5.1: Horizontal Displacement = initial Velocity + (Vertical Force/(2 * mass)) 6.1: X position = X Position + Horizontal Displacement 6.2: Y position = Y Position + Vertical Displacement
Level 3	1.1.1: Distance = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 1.2.1: Force = $\frac{m_1 m_1 G}{r^2}$

Pseudo code

Mass calculations

This will be used to calculate a mass for a planet. This will be a function in the class planet.

```
Function GetVolume(Planet)
Vol =  (4/3) * Pi * (Planet.radius * Planet.radius * Planet.radius)
Return vol
End Function
Function GetMass(Planet)
        Mass = Planet.density * GetVolume(Planet)
        Return mass
End function
```

Gravitational Force

The gravitational force is calculated in the following way; this is necessary for calculating the new position of a planet.

```
GravConst = 0.0000000000667
Force = GravConst * GetMass(Planet1) * GetMass(Planet2)
Force = GravConst / (GetDistance(planet1,planet2) * GetDistance(planet1,planet2))
Distance calculations
Function GetDistance(Planet1, Planet2)
        Distance = (Planet1.X – Planet2.X) * (Planet1.X – Planet2.X)
        Distance = Distance + ((Planet1.Y – Planet2.Y) * (Planet1.Y – Planet2.Y))
        Distance = Sqrt(Distance)
        Return Distance
End function
```

Full Calculation Pseudo Code

In the program the calculation process will take place in one sub routine. This will calculate an overall vector for each planet and then move the planet appropriately.

```
n = 0
WHILE n != NumberOfPlanets:
        k = 0
        WHILE planet(n) != planet(k):
                distance = SQRT(POW((planet(k).x – planet(n).x),2) + POW((planet(k).y –
planet(n).y),2))
                force = planet(n).mass * planet(k).mass * GravitationalConstant
                force = force / (distance * distance)
                forceHorizontal = force * ((ABS(planets(k).x - planets(n).x) / Total Distance)
                forceVertical = SQRT((force*force) – (forceHorizontal*forceHorizontal))
                planets(n).displacements(k).xchange = planets(n).InitialVelocity + (forceHorizontal
/(2*planets(n).mass))
                planets(n).verticalDisplacement = planets(n).InitialVelocity +
(forceVertical/(2*planets(n).mass))
                IF planets(n).X > planets(k).x THEN
                        planets(n).displacements(k).xdirection = “+”
                ELSE
                        planets(n).displacements(k).xdirection = “-“

                IF planets(n).y > planets(k).y THEN
                        planets(n).displacements(k).ydirection = “+”
                ELSE
                        planets(n).displacements(k).ydirection = “-“
                END IF
        END WHILE
END WHILE
FOR ALL Planets:
        FOR ALL VALUES OF k:
                IF planets(n).displacement(k).xdirection = “+” THEN
                        totalX = totalX + planets(n).displacemtns(k)
                ELSE
                        totalX =totalX – planets(n).displacements(k).Xchange
                IF planets(n).displacement(k).ydirection = “+” THEN
                        totalY = totalY + planets(n).displacemtns(k)
                ELSE
                        totalY =totalY – planets(n).displacements(k).Ychange
        END FOR
END FOR
```

Saving and Loading

Outline

The saving and loading will work by using an open file dialog and a save file dialog when the user clicks the load of save buttons, respectively. What these objects do is open the file explorer window. I can then access the windows properties to find out what the user is doing and therefore which file they wish to open. The reading/ writing to text files will be done using stream reader and stream writer.

Pseudo code

Pseudo code for saving Process:

In the subroutine below path is passed from a previous procedure. This would be the file path that has come from the open file dialog.

```
Sub SaveSim(Path)
    Path
    writer = New IO.StreamWriter(Path)
    For Each planet In planets
        If planet.InUse = True Then
            writer.WriteLine(planet.Radius)
            writer.WriteLine(planet.Density)
            writer.WriteLine(planet.Colour)
            writer.WriteLine(planet.Xposition)
            writer.WriteLine(planet.Yposition)
            writer.WriteLine(planet.InitialVelocity.Xcomp)
            writer.WriteLine(planet.InitialVelocity.Ycomp)
        End If
    Next

End Sub
```

Pseudo code for loading Process:

The subroutine simply opens a new file and writes out the data for each planet that is currently in use.

```
Sub LoadSim(path)

    ClearPlanets()
    reader = New IO.StreamReader(path)
    textLine = reader.ReadLine()
    Do Until textLine = Nothing
        For Each planet In planets
            If planet.InUse = False Then
               planet.Radius = textLine
                planet.Density = reader.ReadLine()
                planet.Colour = reader.ReadLine()
                planet.Xposition = reader.ReadLine()
                planet.Yposition = reader.ReadLine()
```

```
            planet.InitialVelocity.Xcomp = reader.ReadLine()
            planet.InitialVelocity.Ycomp = reader.ReadLine()
            planet.InUse = True
            Exit For
          End If
        Next
        textLine = reader.ReadLine()
      Loop
      DrawGraphics()

End sub
```

In this pseudo code the procedure ClearPlanets() simply clears out every planet from the array. This is used so that a model is not loaded on top of a model that is currently in use. The DrawGraphics() subroutine simply updates the simulation graphics.

Event Driven Aspects

Much of the program works in an event driven manner with the program remaining in a waiting state until the user performs an action such as a button click, the program then executes the code that it has for this action before again waiting for another input.

In this section, I will explain how I intend to code some of the more complex areas of this side of the program and will supply pseudo code where necessary.

Mouse Down

```
IF addCursor = TRUE THEN
        FOR n = 0 TO 20
                IF Planets(n).InUse = FALSE THEN
                        planets(n).Radius = TextBoxRadius.text
                        planets(n).Density = TextBoxDensity.text
                        planets(n).Colour = ComboBoxColour.text
                        planets(n).XPosistion = ClickPosistion.X
                        planets(n).YPosistion = ClickPosistion.Y
                        planets(n).InUse = TRUE
                        planets(n).InitialVelocity = TextBoxVelocity
                        planets(n).OriginalX = ClickPosistion.X
                        planets(n).OriginalY = ClickPosistion.Y
                        EXIT FOR
                END FOR
        END FOR
ELSEIF deletecursor = TRUE THEN
        Y = ClickPosistion.Y
        X = ClickPosistion.X

        FOR n = 0 TO 200
                l = SQRT((2* (planets(n).radius)^2))
                d = SQRT(planets(n).radius – (0.5*l^2))
                IF (planets(n).X + d < X) AND (planets(n).X – d > X) AND (planets(n).Y + d < Y) AND
```

```
(planets(n).Y – d > Y) THEN
                        planets(n).InUse = FALSE
                END IF

        END FOR

ELSEIF InfoCursor = TRUE THEN
        Y = ClickPosistion.Y
        X = ClickPosistion.X

        FOR n = 0 TO 200
                l = SQRT((2* (planets(n).radius)^2))
                d = SQRT(planets(n).radius – (0.5*l^2))
                IF (planets(n).X + d < X) AND (planets(n).X – d > X) AND (planets(n).Y + d < Y) AND
(planets(n).Y – d > Y) THEN
                        DRAW ELIPSE ON CANVAS(CENTRE = (planets(n).X, planets(n).Y), RADIUS =
planets(n).radius + 5, FILLCOLOUR = White)

                        DRAW ELIPSE ON CANVAS(CENTRE = (planets(n).X, planets(n).Y), RADIUS =
planets(n).radius, FILLCOLOUR = planets(n).colour)

                        AdvancedTextBoxRad = planet(n).Radius

                        AdvancedTextBoxDen = planet(n).Density

                        AdvancedTextBoxIntialX = planet(n).InitialX

                        AdvancedTextBoxInitialY = planet(n).intialY

                        AdvancedTextBoxPosX = planet(n).Position

                        AdvancedTextBoxPosY = planet(n).Position


                END IF

        END FOR

END IF
```

Mouse Up

When this event is triggered, it will create a planet with the velocity created from the length and direction of the line drawn in the mouse move event. However a planet is only created if the add planet tool is selected.

Mouse Move

If the mouse is being held down and the add planet tool is in use then graphically draw a line between original click and current mouse position as well as measuring and storing the length and direction of this line ready for the mouse up event.

Clear all click

```
ShowDialogue

If dialogue.Answer = Yes

        DeleteAllPlanets()

        DrawGraphics()

End if
```

Play clicked

```
IF BtnPlayPause = “Play” then

BtnAdd = disabled
BtnDel = disabled
BtnInfo = disabled

BtnSave = disabled
BtnLoad = disabled

BtnPlayPause = “Paused”

LOOP UNTIL BtnPlayPause PRESSED OR ClearAll PRESSED:

DrawCanvas()
CalcForces()

END LOOP
ELSE

        DrawCanvas()

BtnAdd = enabled
BtnDel = enabled
BtnInfo = enabled

BtnSave = enabled
BtnLoad = enabled

BtnPlayPause = “Play”


End IF
```

Add planet clicked

cursorType = cross
addCursor = true
InfoCursor = false
DelCursor = false

BtnAdd = disabled
BtnDel = enabled
BtnInfo = enabled

Delete planet clicked

cursorType = hand
addCursor = false
InfoCursor = false
DelCursor = true

BtnAdd = enabled
BtnDel = disabled
BtnInfo = enabled

Information tool click

cursorType = arrow
addCursor = false
InfoCursor = true
DelCursor = false

BtnAdd = enabled
BtnDel = enabled
BtnInfo = disabled

Save Button click

This will show the user the save dialogue and then will retrieve the path and file name when the user clicks save. This is then passed to the save code that was shown earlier and the data for the simulation can be written to the textfile

Load Button click

This will show the user the load file dialogue and will get the path as well as the file name when the user clicks the load button on the dialogue. The path is then passed to the load subroutine which will read the data from the textfile at that location.

Show Advanced checkbox changed

If AdvCheckBox.checked = true
    Then show AdvancedTab
ELSE
    Hide AdvancedTab

Volatile Storage

Class Planet

The class planet will act as a template for all of the celestial objects in a simulation. It will have the following methods and attributes.

Attributes: Structure Displacement -XComp as decimal -YComp as decimal -InUse as Boolean Private: _radius as decimal _density as decimal _colour as string _xposision as decimal _yposition as decimal _inuse as Boolean Public: -InitialVelocity as displacement
Methods Functions: - Volume() - Mass() Properties: - Radius as decimal - Density as decimal - Colour as string - Xposition as decimal - Yposition as decimal - InUse as boolean

So that each object in a simulation is easily identifiable, they will all be stored in an array. This way, each planet has a unique identifier and an identifying value can be passed to different subroutines more easily than passing the entire instance of that class.
Unfortunately, that means that the array will be a global set of variables but it is not an easy or necessary step to pass all of that data around the procedures.

Global Variables

A large issue in designing this program is the issue of global variables. Because of the fact that the program is event driven and yet has many complex states means that when an event occurs the program may need to know what state it is in to know what needs to be done.

For example, when the mouse click on canvas event is triggered the action of the program depends on which of the three main tools is currently selected. It is not possible to pass a variable from the button select event to the canvas click event because one does not necessarily follow the other.

One solution could be to have hidden textboxes that store values representing these states, however, these are not much better than a global variable as they can be incorrectly altered from anywhere in the program.

Technical Solution

```
Imports System.Threading
Imports System.IO



Public Class Canvas

    'global variables:
    Dim g As Graphics
    Dim planets(20) As planet ' stores all instances of the class planet
    Dim RunThread As Boolean = False ' represents whether the simulation is running or paused, true = running
    Dim MouseDown As Boolean = False ' used to show in the program if the mouse button is held down or not, True = down
    Dim ClickX As Integer ' used to show the X location of a click
    Dim ClickY As Integer ' used to show the Y location of a click
    Dim WhichCursor As String


    Sub sortByRadius()
        Dim Rad(20) As Integer
        Dim conversion(20) As planet
        For n = 0 To 20
            conversion(n) = New planet
        Next

        For index = 0 To 20
            Rad(index) = planets(index).Radius ' puts the masses into an array
        Next

        MergeSort(Rad) 'sorts the massess of the planets into ascending order

        'now that the radii have been ordered in an array they are matched to their planets again, when a planet is matched it is
moved to a temporary array (conversion)
        'this means that the planets are re-ordered into the conversion array with ascending radii

        Dim index2 As Integer ' this stores the next available slot in the temporary conversion array
        For plnIndex = 0 To 20 ' for every value in the Mass array
            For index = 0 To 20  ' for every planet
```

```
                If Not ((planets(plnIndex).InUse = False) Or (Rad(index) = 0)) Then 'if the planet is in use in the current
simulation
                    If Math.Round(planets(plnIndex).Radius) = Rad(index) Then ' if the mass of that planet matches the mass in the
array
                        conversion(index2).Radius = planets(plnIndex).Radius ' puts all of the values into the conversion array so
that the main array can be re-ordered
                        conversion(index2).Density = planets(plnIndex).Density
                        conversion(index2).Colour = planets(plnIndex).Colour
                        conversion(index2).Xposition = planets(plnIndex).Xposition
                        conversion(index2).Yposition = planets(plnIndex).Yposition
                        conversion(index2).InUse = planets(plnIndex).InUse
                        conversion(index2).InitialVelocity = planets(plnIndex).InitialVelocity
                        planets(plnIndex).InUse = False
                        index2 += 1
                    End If
                End If

            Next
        Next
        ClearPlanets()  ' now delete all of the planets from the planets array as they have been put into the temporary conversion
array

        For index = 0 To 20 ' moves all values from conversion to planets so that it can be used in the rest of the program.
            If conversion(index).InUse = True Then
                planets(index).Radius = conversion(index).Radius
                planets(index).Density = conversion(index).Density
                planets(index).Colour = conversion(index).Colour
                planets(index).Xposition = conversion(index).Xposition
                planets(index).Yposition = conversion(index).Yposition
                planets(index).InUse = conversion(index).InUse
                planets(index).InitialVelocity = conversion(index).InitialVelocity
            End If
        Next
    End Sub ' orders the planets into descending order by radii in the array planets

    Public Sub MergeSort(ByVal ar() As Integer)
        DoMergeSort(ar, 0, ar.Length - 1)
    End Sub ' passes the radii to the merge sorting sub with minimum and maximum values
```

```
    Private Sub DoMergeSort(ByVal array() As Integer, ByVal Min As Integer, ByVal Max As Integer)
        'keeps down the array into halves and then quaters the sub arrays etc. - Divide stage
        ' then recombines the array in descending order  - Conquer Stage
        If Min >= Max Then
            Return
        End If
        Dim length As Integer = Max - Min + 1
        Dim middle As Integer = Math.Floor((Min + Max) / 2)
        DoMergeSort(array, Min, middle)
        DoMergeSort(array, middle + 1, Max)
        Dim temp(array.Length - 1) As Integer
        For i As Integer = 0 To length - 1
            temp(i) = array(Min + i)
        Next
        Dim m1 As Integer = 0
        Dim m2 As Integer = middle - Min + 1
        For i As Integer = 0 To length - 1
            If m2 <= Max - Min Then
                If m1 <= middle - Min Then
                    If temp(m1) > temp(m2) Then
                        array(i + Min) = temp(m2)
                        m2 += 1
                    Else
                        array(i + Min) = temp(m1)
                        m1 += 1
                    End If
                Else
                    array(i + Min) = temp(m2)
                    m2 += 1
                End If
            Else
                array(i + Min) = temp(m1)
                m1 += 1
            End If
        Next

    End Sub ' performs the recursive merge sort

    Sub loadSim(ByVal path As String)
        ClearPlanets() ' resets entire array
```

```
        Dim reader As StreamReader = New StreamReader(path) 'opens the text file in the location path
        Dim textLine As String = reader.ReadLine() ' textline stores a line of text read from the file that was opened in the above
line
        Do Until textLine = Nothing ' = keep reading lines until you reach a blank line
            For Each planet In planets ' = go through every instance in the array
                If planet.InUse = False Then ' = if the inuse marker is false then the array location is empty so new data can be
stored in it
                    'store the various details about the planet in the memory locations:
                    planet.Radius = textLine
                    planet.Density = reader.ReadLine()
                    planet.Colour = reader.ReadLine()
                    planet.Xposition = reader.ReadLine()
                    planet.Yposition = reader.ReadLine()
                    planet.InitialVelocity.Xcomp = reader.ReadLine()
                    planet.InitialVelocity.Ycomp = reader.ReadLine()

                    planet.InUse = True ' shows that this position in the array is now in use
                    Exit For ' can now leave for as a position has beeen found
                End If
            Next
            textLine = reader.ReadLine()
        Loop
        g.Clear(Color.Black)
        DrawGraphics() ' refresh graphics on screen to show the new simulation that has been loaded above
        reader.Close() ' close the text document
    End Sub ' loads simulations

    Sub SaveSim(ByVal Path As String)
        Dim writer As IO.StreamWriter = New IO.StreamWriter(Path) ' open file in specified location
        For Each planet In planets ' = go through every instance in the array
            If planet.InUse = True Then ' = if the inuse marker is true then there is a planet whos data has to be saved
                'output to the textfile all nessecary data on the planet:
                writer.WriteLine(planet.Radius)
                writer.WriteLine(planet.Density)
                writer.WriteLine(planet.Colour)
                writer.WriteLine(planet.Xposition)
                writer.WriteLine(planet.Yposition)
                writer.WriteLine(planet.InitialVelocity.Xcomp)
                writer.WriteLine(planet.InitialVelocity.Ycomp)
```

```
            End If
        Next
        writer.Close() ' close the text file

    End Sub ' saves simulations

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.KeyPreview = True ' needed for keyboard shortcuts

        NumericUpDownSpeed.Text = 5

        Me.DoubleBuffered = True ' supposed to help with flickering graphics - I am still unsure whether it does or not
        SetStyle(ControlStyles.OptimizedDoubleBuffer, True)
        Me.SetStyle(ControlStyles.UserPaint, True)
        Me.SetStyle(ControlStyles.AllPaintingInWmPaint, True)


        ' hides textboxes that stores initial velocity - this is a feature that was going to be visible to users but it made more
sense from a design point of view to take it away, however it's nessecary to the code.
        TextBoxInitialVelXComp.Hide()
        TextBoxInitialVelYComp.Hide()
        ' hides textboxes that stores X and Y locations in graphics - this is a feature that was going to be visible to users but it
made more sense from a design point of view to take it away, however it's nessecary to the code.
        TextBoxXcor.Hide()
        TextBoxYcor.Hide()

        ButtonInfoTool.Enabled = False
        'starting data for the 3 textboxes - this gives user an idea of what sort of val they need to enter without having to look at
tool tip
        TextBoxDensity.Text = 5
        TextBoxRadius.Text = 20
        ComboBoxColour.Text = "Grey"



        'window formating etc. :
        Panel1.Hide() ' hidden so that it can be shown again - this refreshes the panel so that it apears properly
        Me.WindowState = FormWindowState.Maximized ' maximises window
        Panel2.Dock = DockStyle.Right 'docks controls to right hand side of screen
        ' makes canvas the same size as screen
```

```
        PictureBoxCanvas.Width = (Me.Width)
        PictureBoxCanvas.Height = (Me.Height)

        g = Me.PictureBoxCanvas.CreateGraphics 'enables graphics on the canvas

        ClearPlanets() 'sets up the array of planets
        DrawGraphics()
        'makes sure there is a location for the program to open when the load or save buttons are pressed
        If (Not System.IO.Directory.Exists("C:\Gravitas")) Then
            System.IO.Directory.CreateDirectory("C:\Gravitas")
        End If

        TextBoxAdvRad.Enabled = False
        TextBoxAdvDen.Enabled = False
        TextBoxAdvInitialX.Enabled = False
        TextBoxAdvInitialY.Enabled = False
        TextBoxAdvMass.Enabled = False
        TextBoxAdvXpos.Enabled = False
        TextBoxAdvYpos.Enabled = False
    End Sub ' intial formating for the form and setting up graphics etc.

    Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown
        If e.Shift And e.KeyCode.ToString = "A" Then ' shift-A = add planet tool
            ButtonAddPlanet.PerformClick()
        End If

        If e.Shift And e.KeyCode.ToString = "D" Then ' shift-D = delete planet tool
            ButtonDeletePlanet.PerformClick()
        End If

        If e.Shift And e.KeyCode.ToString = "R" Then ' shift-R = Run simultion
            ButtonPlay.PerformClick()
        End If

        If e.Control And e.KeyCode.ToString = "S" Then ' ctrl-S = save simulation
            ButtonSave.PerformClick()
        End If

        If e.Control And e.KeyCode.ToString = "L" Then  ' ctrl-L = load simulation
            ButtonSave.PerformClick()
```

```
        End If
    End Sub   'handles keyboard Shortcuts

    Sub CreatePlanet()
        Dim strVel As String '  temporary storage for the output of the initial velocity textboxes, this is so that validation can
take place

        For Each planet In planets ' = go through array
            If planet.InUse = False Then '  = until you find a position that is not currently in use

                'establishes values for radius etc. from textboxes
                planet.Radius = TextBoxRadius.Text
                planet.Density = TextBoxDensity.Text
                planet.Colour = ComboBoxColour.Text
                planet.Xposition = TextBoxXcor.Text
                planet.Yposition = TextBoxYcor.Text
                planet.InUse = True 'marks that planet as in use, therefore it won't be written over

                strVel = TextBoxInitialVelYComp.Text
                If TextBoxInitialVelYComp.Text = Nothing Then 'if the textbox is blank then taking it and directly putting it into an
integer would cause it to crash
                    planet.InitialVelocity.Ycomp = 0
                Else
                    planet.InitialVelocity.Ycomp = TextBoxInitialVelYComp.Text
                End If

                strVel = TextBoxInitialVelXComp.Text
                If TextBoxInitialVelXComp.Text = Nothing Then 'if the textbox is blank then taking it and directly putting it into an
integer would cause it to crash
                    planet.InitialVelocity.Xcomp = 0
                Else
                    planet.InitialVelocity.Xcomp = TextBoxInitialVelXComp.Text
                End If

                Exit For
            End If
        Next
    End Sub ' Gathers data for new entry into the planet array

    Sub DrawGraphics()
```

```
        If Not CheckBoxTrace.Checked Then
            g.Clear(Color.Black)
        End If
        For Each planet In planets ' = every instance of planet in the array
            If planet.InUse = True Then 'if its in use flag is true then it needs to be drawn
                Dim x As Decimal = (planet.Xposition - 0.5 * (planet.Radius * 2))
                Dim y As Decimal = (planet.Yposition - 0.5 * (planet.Radius * 2))


                'there isn't a way to have one fill statement so this select case draws the planet with the appropriate colour
                Select Case planet.Colour
                    Case "Grey"
                        g.FillEllipse(Brushes.Gray, x, y, (planet.Radius * 2), (planet.Radius * 2))
                    Case "Blue"
                        g.FillEllipse(Brushes.Blue, x, y, (planet.Radius * 2), (planet.Radius * 2))
                    Case "Yellow"
                        g.FillEllipse(Brushes.Yellow, x, y, (planet.Radius * 2), (planet.Radius * 2))
                    Case "Red"
                        g.FillEllipse(Brushes.Red, x, y, (planet.Radius * 2), (planet.Radius * 2))
                    Case "Green"
                        g.FillEllipse(Brushes.Green, x, y, (planet.Radius * 2), ((planet.Radius * 2)))
                End Select


            End If
        Next


    End Sub 'updates the graphic output

    Private Sub DoWork()
        sortByRadius()
        While RunThread = True ' until the pause button is pressed runThread will be true, therefore this will keep looping until
pause is pressed.
            DrawGraphics() ' update grapics
            'if the merge on collision checkbox is ticked then this boolean will be true and so the program will check to see whether
any planets need to merge and merge them if they do
            If CheckBoxCollision.Checked = True Then
                checkMerge()
            End If
```

```
            calcForces() ' find new planet positions
        End While
    End Sub  ' checks pause button has not been pressed

    Sub checkMerge()
        Dim merge As Boolean = False ' will be true when two planets should merge
        For Each planet In planets ' = for every instance of planet in the array
            If planet.InUse <> False Then ' only checks the planets that are currently in use in the simulation
                ' will go through every planet in the array that is in use, OTHER than the current planet. This means that each
planet gets compared to every other planet
                For Each pln In planets
                    If pln.InUse <> False And Not ((planet.Xposition = pln.Xposition) And (planet.Yposition = pln.Yposition) And
(planet.Radius = pln.Radius)) Then

                        'if the planet is within the other planet in both the x and the y then merge will become true to show that
they should merge together
                        If pln.Xposition > (planet.Xposition - planet.Radius) And pln.Xposition < (planet.Xposition + planet.Radius)
And pln.Yposition > (planet.Yposition - planet.Radius) And pln.Yposition < (planet.Yposition + planet.Radius) Then
                            merge = True
                        End If


                        'merges the planets together if they should be
                        If merge = True Then
                            Dim M As Decimal
                            If planet.mass >= pln.mass Then ' makes sure that the smaller planet is added to the larger
                                planet.InitialVelocity.Xcomp = ((planet.InitialVelocity.Xcomp * planet.mass) +
(pln.InitialVelocity.Xcomp * pln.mass)) / (planet.mass + pln.mass)
                                planet.InitialVelocity.Ycomp = ((planet.InitialVelocity.Ycomp * planet.mass) +
(pln.InitialVelocity.Ycomp * pln.mass)) / (planet.mass + pln.mass)
                                planet.Radius += pln.Radius  'adds the radii together
                                M = (planet.mass + pln.mass)
                                planet.Density = M / ((4 / 3) * Math.PI * Math.Pow(planet.Radius, 3))


                                'planet.Density = (planet.mass + pln.mass) / ((4 / 3) * Math.PI * (Math.Pow((planet.Radius +
pln.Radius), 3))) ' finds density by mass / volume
                                ' finds intial velocity by finding the momentum's and adding them
```

```
                            'removes other planet from simulation
                            pln.InUse = False
                            pln.Radius = Nothing
                            pln.Density = Nothing
                            pln.Xposition = Nothing
                            pln.Yposition = Nothing

                        ElseIf planet.mass < pln.mass Then ' makes sure that the smaller planet is added to the larger
                            pln.InitialVelocity.Xcomp = ((planet.InitialVelocity.Xcomp * planet.mass) +
(pln.InitialVelocity.Xcomp * pln.mass)) / (planet.mass + pln.mass)
                            pln.InitialVelocity.Ycomp = ((planet.InitialVelocity.Ycomp * planet.mass) +
(pln.InitialVelocity.Ycomp * pln.mass)) / (planet.mass + pln.mass)
                            pln.Radius += planet.Radius  'adds the radii together
                            M = (planet.mass + pln.mass)
                            pln.Density = M / ((4 / 3) * Math.PI * Math.Pow(pln.Radius, 3))

                            'removes other planet from simulation
                            planet.InUse = False
                            planet.Radius = Nothing
                            planet.Density = Nothing
                            planet.Yposition = Nothing
                            planet.Yposition = Nothing

                        End If
                        merge = False ' resets merge for later checks
                    End If
                End If
            Next
        End If
    Next
End Sub ' checks to see whether there are any planets that need to be merged into one and if so it does this

Sub calcForces()


    Dim distance As Decimal ' stores a distance between two planets
    Dim force As Decimal  ' stores the force between two planets
    Dim ForceHor As Decimal ' stores the horizontal component of the force between two planets
    Dim ForceVer As Decimal  ' stores the vertical component of the force between two planets
```

```
        Dim GravConst As Decimal = 0.00667 ' gravitational constant - changed for my model so that it works in an interesting way in
the scale that I have


        Dim index1 As Integer
        Dim index2 As Integer

        For index1 = 0 To 20

            For index2 = 0 To 20
                If (planets(index1).InUse = True) And (planets(index2).InUse = True) And index1 < index2 Then

                    distance = Math.Sqrt(Math.Pow((planets(index1).Xposition - planets(index2).Xposition), 2) +
Math.Pow((planets(index1).Yposition - planets(index2).Yposition), 2))
                    force = planets(index1).mass * planets(index2).mass
                    force = force * GravConst
                    force = force / Math.Pow(distance, 2)
                    ForceHor = force * ((planets(index1).Xposition - planets(index2).Xposition) / distance)
                    ForceVer = Math.Sqrt(Math.Abs(Math.Pow(force, 2) - Math.Pow(ForceHor, 2)))



                    'for planets(index1)
                    Dim tempXval As Decimal = ForceHor / (2 * planets(index1).mass)
                    Dim tempYval As Decimal = ForceVer / (2 * planets(index1).mass)

                    If planets(index1).Xposition > planets(index2).Xposition Then
                        tempXval = Math.Abs(tempXval) * -1
                    Else
                        tempXval = Math.Abs(tempXval)
                    End If
                    If planets(index1).Yposition > planets(index2).Yposition Then
                        tempYval = Math.Abs(tempYval) * -1
                    Else
                        tempYval = Math.Abs(tempYval)
                    End If

                    planets(index1).InitialVelocity.Xcomp += tempXval
                    planets(index1).InitialVelocity.Ycomp += tempYval
```

```
                'for planets(index2)
                tempXval = ForceHor / (2 * planets(index2).mass)
                tempYval = ForceVer / (2 * planets(index2).mass)

                If planets(index2).Xposition > planets(index1).Xposition Then
                    tempXval = Math.Abs(tempXval) * -1
                Else
                    tempXval = Math.Abs(tempXval)
                End If
                If planets(index2).Yposition > planets(index1).Yposition Then
                    tempYval = Math.Abs(tempYval) * -1
                Else
                    tempYval = Math.Abs(tempYval)
                End If

                planets(index2).InitialVelocity.Xcomp += tempXval
                planets(index2).InitialVelocity.Ycomp += tempYval

            End If
        Next

        planets(index1).Xposition += (planets(index1).InitialVelocity.Xcomp / 50)
        planets(index1).Yposition += (planets(index1).InitialVelocity.Ycomp / 50)
    Next


    CalcSimSpeed() ' goes to sub to calculate the waiting period between calculations and therefore the frame rate/ speed of
simulation

End Sub ' calculates the movement of all of the planets and updates the positions of the planets accordingly

Sub CalcSimSpeed()

    Dim speed As String = NumericUpDownSpeed.Text ' gets speed value from the nummeric up down which has a range 0 to 10

    'the variable speed represents the number of milliseconds that the program will wait before calculating the next frame of the
animation

    If Empty(speed) = True Then ' checks to see that the speed is not nothing
```

```
            speed = 50 ' if so the speed is set to 50
        Else
            If IsInt(speed) = True Then ' makes sure the speed is castable as an integer
                speed = CInt(speed)
                If speed = 0 Then ' is the speed is 0 then the program cannot wait 0 milliseconds because the flicker would be so bad
that the program would be unusable
                    speed = 1 ' instead the wait time is set to 1 millisecond which is still bad but managable
                Else
                    speed = speed * 10 ' if the speed is not 0 then it is multiplied by ten as there is no noticable difference
between 1,2,3 etc. but there is between 20, 30, 50 etc.
                End If
            Else ' if it is not it may be a negative number etc.
                speed = 50 ' sets speed to 50
                speed = CInt(speed) ' casts speed to an integer
            End If

        End If

        Threading.Thread.Sleep(speed) ' waits the amount of time calculated above.
    End Sub ' used to workout the appropriate speed for the model to run at based on input from numericUpDownSpeed. Also has
validation for this input

    Function Empty(ByVal Value As String)
        If Value = Nothing Then
            Return True
        Else
            Return False
        End If
    End Function 'function used to determine whether the given string is null or has a value, True if Null, False if Not Null.

    Function IsInt(ByVal val As String)
        Dim val1 As Integer
        Dim length As Integer = val.Length
        Dim Invalid As Boolean = False


        For n = 0 To (length - 1) ' for every character in the string
            val1 = Asc(val(n)) ' get the ascii value of the character
            If Not (val1 >= 48 And val1 <= 57) Then ' if the ascii value is between 48 and 57 then the character is a number
```

```
                Return False 'if not then the whole value is not a number and will cause an error when it is cast as an integer so
false is returned
                MsgBox("False")
            End If
        Next

        If val = Nothing Then ' incase the value is nothing and length = 0
            Return True ' the integer value will end up being 0
        End If

        Return True ' if it reaches this stage and it has not returned false then it must be a number so true is returned

    End Function 'checks to see if the givin value is an integer - used for validation

    Sub ClearPlanets()
        Dim index As Integer
        ' goes through positions 0 to 20 and puts in a new instance of planet
        For index = 0 To 20
            planets(index) = New planet '
        Next
    End Sub ' sets up the planet array as a series of 20 instances of planet all without any values, this is used both at the start
of the program to create the array and at any point that all planets must be deleted

    Sub PutInAdv(ByVal position As Integer)
        'Puts in :

        TextBoxAdvRad.Text = planets(position).Radius ' Radius
        TextBoxAdvDen.Text = planets(position).Density 'Density
        TextBoxAdvInitialX.Text = planets(position).InitialVelocity.Xcomp ' Initial Velocity X component
        TextBoxAdvInitialY.Text = planets(position).InitialVelocity.Ycomp ' Initial Velocity Y component
        TextBoxAdvXpos.Text = planets(position).Xposition ' Planet Y position
        TextBoxAdvYpos.Text = planets(position).Yposition ' Planet X position
        TextBoxAdvMass.Text = planets(position).mass 'Planet Mass

    End Sub ' puts the appropriate values into the advanced tab for the required planet

    Sub DrawOneObj(ByVal col As String, ByVal diameter As Integer, ByVal x As Integer, ByVal y As Integer)
        x -= 0.5 * diameter ' re-adjusts the X and Y locations as the regular draw ellipse will not put the centre right on the click
location
        y -= 0.5 * diameter
```

```
        Select Case col ' draws the circle of the chosen colour and size in the click location
            Case "Grey"
                g.FillEllipse(Brushes.Gray, x, y, diameter, diameter)
            Case "Blue"
                g.FillEllipse(Brushes.Blue, x, y, diameter, diameter)
            Case "Yellow"
                g.FillEllipse(Brushes.Yellow, x, y, diameter, diameter)
            Case "Red"
                g.FillEllipse(Brushes.Red, x, y, diameter, diameter)
            Case "Green"
                g.FillEllipse(Brushes.Green, x, y, diameter, diameter)
        End Select
    End Sub ' draws just one planet when given: location (x,y) diameter and colour

    Function ValidateInput(ByVal Str As String, ByVal field As String, ByVal min As Integer, ByVal max As Integer) ' min and max are
the upper and lower bounds of excepted values, field is the name of the input (e.g. Radius), Str is the input
        If Str = Nothing Then ' checks that it is not null
            MsgBox("You Must Enter A {0} Before You Create A Planet!", field)
        Else
            If IsInt(Str) = True Then ' goes to function that ensures it is an integer
                If CInt(Str) < min Or CInt(Str) > max Then ' checks range
                    MsgBox(field & " must be integer between " & min & " and " & max)
                Else
                    Return True ' if it meets all requirements then it can return true
                End If
            Else
                MsgBox(field & " must be integer between " & min & " and " & max)
            End If
        End If
    End Function ' makes sure that a input is a valid integer - true if valid, false if invalid

    Private Sub PictureBoxCanvas_mousedown(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles
PictureBoxCanvas.MouseDown
        TextBoxXcor.Text = e.X ' gets the X and Y location of the click and stores it in the textboxes so the user can see it
numericaly
        TextBoxYcor.Text = e.Y
        ' A big list of nested ifs that decide what needs to happen when the user clicks down on the canvas:

        If RunThread = False Then ' if the simulation is NOT running then
```

```
            TextBoxAdvRad.Text = "" ' clear all textboxes
            TextBoxAdvDen.Text = ""
            TextBoxAdvInitialX.Text = ""
            TextBoxAdvInitialY.Text = ""
            TextBoxAdvXpos.Text = ""
            TextBoxAdvYpos.Text = ""
            TextBoxAdvMass.Text = ""
            If CheckBoxTrace.Checked Then ' clear the canvas if the trace planets tool is on
                g.Clear(Color.Black)
            End If
            DrawGraphics()
            If (WhichCursor = "A") Then ' If the user wants to create a planet
                Dim inputStr As String
                Dim field As String
                Dim rangeMin As String
                Dim rangeMax As String
                inputStr = TextBoxRadius.Text
                field = "Radius"
                rangeMin = 1
                rangeMax = 120
                If ValidateInput(inputStr, field, rangeMin, rangeMax) Then ' validates Radius
                    inputStr = TextBoxDensity.Text
                    field = "Density"
                    rangeMin = 1
                    rangeMax = 1000
                    If ValidateInput(inputStr, field, rangeMin, rangeMax) Then ' validates Density
                        MouseDown = True ' sets this variable to true so that the program knows the mouse button is currently being
held down
                        ClickX = e.X ' gets the X and Y location of the click
                        ClickY = e.Y
                        'works out details for the advanced tab and puts them in for the user to see
                        TextBoxAdvDen.Text = TextBoxDensity.Text ' outputs the new planets density
                        TextBoxAdvRad.Text = TextBoxRadius.Text ' outputs the new planets radius
                        TextBoxAdvMass.Text = ((4 / 3) * (Math.PI) * Math.Pow((TextBoxRadius.Text), 3)) * (TextBoxDensity.Text) '
outputs the new planets mass
                        TextBoxAdvXpos.Text = e.X ' outputs the new planets X and Y position
                        TextBoxAdvYpos.Text = e.Y
                        Dim col As String = ComboBoxColour.Text ' gets colour from combo box
                        Dim diameter As Integer = (TextBoxRadius.Text * 2) ' finds the diameter as double the radius
                        Dim x As Integer = TextBoxXcor.Text ' gets the X and Y location of the click
```

```
                    Dim y As Integer = TextBoxYcor.Text
                    DrawOneObj(col, diameter, x, y)
                End If
            End If
        ElseIf WhichCursor = "D" Then 'If user is trying to delete a planet
            Dim index As Integer = CheckClick(e.X, e.Y) ' function that checks to see if a click is within a planet - returns
that planets number OR 23 if no planet is selected
            If index <> 23 Then
                planets(index).InUse = False
                planets(index).Radius = Nothing
                planets(index).Density = Nothing
                planets(index).Colour = Nothing
                planets(index).InitialVelocity.Xcomp = Nothing
                planets(index).InitialVelocity.Ycomp = Nothing
                planets(index).TotalVelocity.Xcomp = Nothing
                planets(index).TotalVelocity.Ycomp = Nothing
                g.Clear(Color.Black)
                DrawGraphics() ' update graphics so that user no longer sees the deleted planet on screen
                MouseDown = False ' mousedown is now false
            End If
        ElseIf WhichCursor = "I" And (CheckBoxToggleAdvanced.Checked = True) Then 'if neither tool is in use then it selects the
planet and displays its data in the advanced tab as the information tool must be in use
            Dim index As Integer = CheckClick(e.X, e.Y) ' function that checks to see if click is within a planet - returns that
planets number OR 23 if no planet is selected
            If index = 23 Then
                DrawGraphics()
                TextBoxAdvRad.Text = ""
                TextBoxAdvDen.Text = ""
                TextBoxAdvInitialX.Text = ""
                TextBoxAdvInitialY.Text = ""
                TextBoxAdvXpos.Text = ""
                TextBoxAdvYpos.Text = ""
                TextBoxAdvMass.Text = ""
            Else
                planets(index).SelectForAdvanced = True
                PutInAdv(index)
                g.Clear(Color.Black)
                DrawGraphics()
                'draws a white circle larger than the planet then redraws the planet on top
                'this gives the impression that there is a white circle around the planet showing it as selected
```

```
                    Dim d As Integer = planets(index).Radius * 2 ' finds diameter
                    g.FillEllipse(Brushes.White, (planets(index).Xposition - planets(index).Radius - 5), (planets(index).Yposition -
planets(index).Radius - 5), (d + 10), (d + 10)) ' draws the white circle - canvas does not have to be cleared because the only
changes made are hidden by this larger white circle
                    ' then draws the regular planet on top
                    DrawOneObj(planets(index).Colour, (planets(index).Radius * 2), planets(index).Xposition,
planets(index).Yposition)
                End If
            End If
        End If

    End Sub ' essentialy a big list of nested ifs that decide what needs to happen when the user clicks down on the canvas

    Function CheckClick(ByVal x As Integer, ByVal y As Integer)
        Dim index As Integer = 0
        For index = 0 To 20 ' for every planet in array
            If planets(index).InUse = True Then ' where it is currently being used in the model
                If (x > (planets(index).Xposition - planets(index).Radius)) And (x < (planets(index).Xposition +
planets(index).Radius)) Then ' if the x coordinate is between the planets upper and lower radius bound
                    If (y > (planets(index).Yposition - planets(index).Radius)) And (y < (planets(index).Yposition +
planets(index).Radius)) Then ' and if the y is between the upper and lower bound for that planet
                        Return index ' return the number planet in the array
                    End If
                End If
            End If
        Next
        Return 23 ' if not any of the planets in the array it returns a value greater than 20 to show that it could not be found
    End Function ' looks through every planet trying to find a planet that contains the passed coordinates

    Private Sub PictureBoxCanvas_mousemove(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles
PictureBoxCanvas.MouseMove
        TextBoxMouseXcor.Text = e.X ' updates mouse location
        TextBoxMouseYcor.Text = e.Y
        If WhichCursor = "A" And (MouseDown = True) Then ' if the user just created a planet and is still holding the mouse down this
means they are drawing a line to give it an intial velocity
            ' DrawGraphics() ' updates graphics
            Dim yCord, xCord As String ' gets the X and Y location for centre of planet
            xCord = TextBoxXcor.Text
            yCord = TextBoxYcor.Text
            g.Clear(Color.Black)
```

```
            DrawGraphics()
            g.DrawLine(Pens.Green, CInt(xCord), CInt(yCord), CInt(e.X), CInt(e.Y)) 'draws green line from centre of planet to cursor
location
            ' draws the main planet again
            Dim col As String = ComboBoxColour.Text ' gets colour
            Dim diameter As Integer = (TextBoxRadius.Text * 2) ' gets diameter
            Dim x As Integer = TextBoxXcor.Text ' gets centre
            Dim y As Integer = TextBoxYcor.Text
            DrawOneObj(col, diameter, x, y)
            Me.TextBoxInitialVelXComp.Text = (ClickX - e.X) ' stores the intial velocity in the textbox based on the length and
direction of the line that has been drawn
            Me.TextBoxInitialVelYComp.Text = (ClickY - e.Y)
            Me.TextBoxAdvInitialX.Text = (ClickX - e.X)
            Me.TextBoxAdvInitialY.Text = (ClickY - e.Y)
        End If
    End Sub ' sorts out what to do when the mouse is move on the canvas

    Private Sub PictureBoxCanvas_mouseUp(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles
PictureBoxCanvas.MouseUp

        If WhichCursor = "A" And (MouseDown = True) Then ' if the user has created a planet and is now releasing there mouse click
            MouseDown = False 'shows that the mouse click is now over

            CreatePlanet()  ' creates a planet where the user clicked
            If CheckBoxTrace.Checked Then
                g.Clear(Color.Black)
            End If
            DrawGraphics()  ' updates graphics which will now include the new planet

            TextBoxInitialVelXComp.Clear() ' clears out all of the textboxes so that the next planet's data is not corrupted by
leftover data from this planet
            TextBoxInitialVelYComp.Clear()
            TextBoxXcor.Clear()
            TextBoxYcor.Clear()
            TextBoxAdvDen.Clear()
            TextBoxAdvRad.Clear()
            TextBoxAdvXpos.Clear()
            TextBoxAdvYpos.Clear()
            TextBoxAdvInitialX.Clear()
            TextBoxAdvInitialY.Clear()
```

```
            TextBoxAdvMass.Clear()
        End If

    End Sub ' sorts out what happens when the user releases the click on the canvas

    Private Sub ButtonClearAll_Click(sender As Object, e As EventArgs) Handles ButtonClearAll.Click
        If RunThread = True Then
            RunThread = False ' stops the model while the user reads the dialog below
            Dim result As Integer = MessageBox.Show("Do you want to delete your entire model? Any unsaved changes will be lost", "Are
you sure?", MessageBoxButtons.YesNo) ' brings up an 'are you sure' box
            If Not result = DialogResult.No Then ' if the user clicks continue

                ButtonAddPlanet.Enabled = True ' enables all of the buttons that were disabled whilst the simulation was running
                ButtonLoad.Enabled = True
                ButtonSave.Enabled = True
                ButtonPlay.Text = "Play" ' and resets the pause button to a play button as the simulation has now ended
                ButtonDeletePlanet.Enabled = True

                ClearPlanets() ' deletes every planet in the array
                g.Clear(Color.Black)
                ButtonInfoTool.PerformClick()
            Else
                RunThread = True ' restarts the model if the user does selects no on the message box
                Dim MainThread As Thread = New Thread(AddressOf DoWork) ' puts the work of running the simulation in a new parralel
thread as the program needs to be animating the screen as well as waiting for the pause button click event at the same time - this
requires two parralel processing threads
                MainThread.IsBackground = True
                MainThread.Start()
            End If
        Else
            Dim result As Integer = MessageBox.Show("Do you want to delete your entire model? Any unsaved changes will be lost", "Are
you sure?", MessageBoxButtons.YesNo) ' brings up an 'are you sure' box
            If Not result = DialogResult.No Then ' if the user clicks continue

                ClearPlanets() ' deletes every planet in the array
                g.Clear(Color.Black)
                ButtonInfoTool.PerformClick()
```

```
            End If
        End If

    End Sub ' stops the array and deletes all of the planets that were in it

    Private Sub ButtonPlay_Click(sender As Object, e As EventArgs) Handles ButtonPlay.Click

        If ButtonPlay.Text = "Play" Then ' if the simulation was previously paused and the user now wants it to run:

            ButtonPlay.Text = "Pause" ' changes the interface to make more sense for a running simulation by disabling certain
buttons and therefore tools
            ButtonAddPlanet.Enabled = False
            ButtonDeletePlanet.Enabled = False
            ButtonInfoTool.Enabled = False
            ButtonSave.Enabled = False
            ButtonLoad.Enabled = False

            TextBoxAdvDen.Clear() ' clears out all of the textboxes in the advanced tab
            TextBoxAdvInitialX.Clear()
            TextBoxAdvInitialY.Clear()
            TextBoxAdvRad.Clear()
            TextBoxAdvXpos.Clear()
            TextBoxAdvYpos.Clear()
            TextBoxAdvMass.Clear()

            g.Clear(Color.Black)

            RunThread = True ' sets the boolean that represents if the simulation is running to true
            WhichCursor = "NONE" ' disables the tools that effect clicking on the canvas

            Cursor = Cursors.Arrow ' resets cursor to and arrow to demonstrate to the user that they no longer are using the tool
that they had previously selected

            Dim MainThread As Thread = New Thread(AddressOf DoWork) ' puts the work of running the simulation in a new parralel
thread as the program needs to be animating the screen as well as waiting for the pause button click event at the same time - this
requires two parralel processing threads
            MainThread.IsBackground = True
            MainThread.Start()

        Else ' pauses the simulation
```

```
            'resets UI to paused mode by re-enabling tools etc. :
            ButtonPlay.Text = "Play"
            ButtonAddPlanet.Enabled = True
            ButtonDeletePlanet.Enabled = True
            ButtonLoad.Enabled = True
            ButtonSave.Enabled = True
            ButtonInfoTool.Enabled = True

            RunThread = False ' marks the simulation as paused
            DrawGraphics() ' updates the graphics
            ButtonInfoTool.PerformClick()
        End If

    End Sub ' sorts out what to do when the play/pause button is pressed

    Private Sub ButtonAddPlanet_Click(sender As Object, e As EventArgs) Handles ButtonAddPlanet.Click
        Cursor = Cursors.Cross ' changes the cursor type to a cross so that the user can visualy see what tool they have selected
        WhichCursor = "A" ' sets the delete tool and info tool booleans to false as they are not in use and the add tool to true as
this is in use


        ButtonDeletePlanet.Enabled = True ' changes the buttons to disable the tool that is currently selected - this way the user
can easily see which tool they are using and what their other options are
        ButtonAddPlanet.Enabled = False
        ButtonInfoTool.Enabled = True
    End Sub ' sets up the add planet tool

    Private Sub ButtonDeletePlanet_Click(sender As Object, e As EventArgs) Handles ButtonDeletePlanet.Click
        Cursor = Cursors.Hand ' changes the cursor type to a cross so that the user can visualy see what tool they have selected
        WhichCursor = "D" ' sets the add tool and info tool booleans to false as they are not in use and the delete tool to true as
this is in use


        ButtonDeletePlanet.Enabled = False ' changes the buttons to disable the tool that is currently selected - this way the user
can easily see which tool they are using and what their other options are
        ButtonAddPlanet.Enabled = True
        ButtonInfoTool.Enabled = True
    End Sub ' sets up the delete planet tool

    Private Sub ButtonInfoTool_Click(sender As Object, e As EventArgs) Handles ButtonInfoTool.Click
```

```
        Cursor = Cursors.Arrow ' changes the cursor type to a cross so that the user can visualy see what tool they have selected
        WhichCursor = "I" ' sets the add tool and delete tool booleans to false as they are not in use and the info tool to true as
this is in use


        ButtonDeletePlanet.Enabled = True ' changes the buttons to disable the tool that is currently selected - this way the user
can easily see which tool they are using and what their other options are
        ButtonAddPlanet.Enabled = True
        ButtonInfoTool.Enabled = False
    End Sub ' sets up the information tool

    Private Sub ButtonSave_Click(sender As Object, e As EventArgs) Handles ButtonSave.Click
        'sets up the save file dialogue
        With SaveFileDialog1
            .FileName = "" ' file name is inputed by user into dialogue later
            .Title = "Save Simulation" ' title of window
            .InitialDirectory = "C:\Gravitas" ' start location of the file explorer navigation
            .Filter = "Text files|*.txt" ' text files only
        End With
        SaveFileDialog1.ShowDialog() ' shows dialogue
        If SaveFileDialog1.FileName <> "" Then ' makes sure the user puts in a name for the save
            SaveSim(SaveFileDialog1.FileName) ' goes to sub which saves the file
        End If

    End Sub ' sorts out the save file dialogue comming up and what it says to the user

    Private Sub ButtonLoad_Click(sender As Object, e As EventArgs) Handles ButtonLoad.Click
        'sets up the load file dialogue
        With OpenFileDialog1
            .FileName = "" ' file name is inputed by user into dialogue so for now it is blank
            .Title = "Load Simulation" ' title of window
            .InitialDirectory = "C:\Gravitas" ' start location of the file explorer navigation
            .Filter = "Text files|*.txt" ' text files only
            .ShowDialog() ' shows dialogue to user
        End With
        Dim path As String = OpenFileDialog1.FileName ' Gets file name
        If OpenFileDialog1.FileName <> "" Then ' if it's not Null
            loadSim(path) ' it is passed to the load sub which sorts out loading the simulation into the program
        End If
    End Sub ' sorts out the load file dialogue
```

```
    Private Sub CheckBox1_CheckedChanged(sender As Object, e As EventArgs) Handles CheckBoxToggleAdvanced.CheckedChanged
        If Not CheckBoxToggleAdvanced.Checked Then ' user wants to hide AdvTab
            Panel1.Hide() ' hide advanced tab
            If RunThread = False Then
                DrawGraphics()
            End If
            ButtonAddPlanet.PerformClick() ' selects the add planet tool so that the user is not still using the info tool which they
cannot now see
        Else ' user wants advTab to show
            Panel1.Show()
            ButtonInfoTool.PerformClick() ' selects the info tool as this is how the
        End If
    End Sub 'Sorts out whether the advanced tab needs to be showing when the user changes the show advanced checkbox

End Class



Class planet
    Structure displacement
        Public Xcomp As Decimal
        Public Ycomp As Decimal
        Public InUse As Boolean
    End Structure

    Private _radius As Decimal
    Private _density As Decimal
    Private _colour As String
    Private _xposistion As Decimal
    Private _yposistion As Decimal
    Private _inuse As Boolean

    Public SelectForAdvanced As Boolean
    Public InitialVelocity As displacement
    Public TotalVelocity As displacement

    Public Property Radius As Decimal
        Set(value As Decimal)
            _radius = value
```

```
        End Set
        Get
            Return _radius
        End Get
    End Property

    Public Property Density As Decimal
        Set(value As Decimal)
            _density = value
        End Set
        Get
            Return _density
        End Get
    End Property

    Public Property Colour As String
        Set(value As String)
            _colour = value
        End Set
        Get
            Return _colour
        End Get
    End Property

    Public Property Xposition As Decimal
        Set(value As Decimal)
            _xposistion = value
        End Set
        Get
            Return _xposistion
        End Get
    End Property

    Public Property Yposition As Decimal
        Set(value As Decimal)
            _yposistion = value
        End Set
        Get
            Return _yposistion
        End Get
```

```
    End Property

    Public Property InUse As Boolean
        Set(value As Boolean)
            _inuse = value
        End Set
        Get
            Return _inuse
        End Get
    End Property


    Function volume()
        Dim vol As Decimal
        vol = Math.Pow(_radius, 3)
        vol = vol * Math.PI
        vol = vol * (4 / 3)
        Return vol
    End Function

    Function mass()
        Dim mas As Decimal
        mas = volume() * _density
        Return mas
    End Function

End Class
```

Testing

Unit Testing

In this section of testing I will be aiming to test the basic functions of the program, that it has accurate validation and that the buttons perform as expected.

Test Number	Test	Test Description	Expected outcome	Passed/ Failed	Comments
1	Add button click	Add Button is clicked	-Disable Add planet button -Enable delete planet button -Change cursor to cross	Passed	
2	Delete button click	Delete planet button is clicked	-Disable delete planet button -Enable add planet button -Change cursor to hand	Passed	
3	Info button click	Information button is clicked	-enables delete planet button -enables add planet button -Change cursor to arrow -disables information button	Passed	
4	Play button click – playing	Play button clicked where the text on the button says ‘pause’	-stop simulation -enable delete planet button -enable add planet button -enable save simulation button -enable load simulation button - select information tool	Passed	
5	Play button click - paused	Play button clicked where the text on the button says ‘play’	- Start simulation -Disable delete planet button -disable add planet button	Passed	

			-disable load button -disable save button -disable information tool		
6	Clear all	Clear all clicked	Are You sure message appears	Passed	
7	Clear all Y	Click yes on clear all 'are you sure' message	-Stop simulation -delete all planets -redraw graphics enable delete planet button -enable add planet button -enable save simulation button -enable load simulation button -disable information tool	Passed	
8	Clear all N	Click no on clear all 'are you sure' message	Continue running simulation	Passed	
9	Save opened	Save simulation clicked	Show save simulation dialogue	Passed	
10	Load opened	Load simulation clicked	Show load simulation dialogue	Passed	
11	Trace planets checked	Trace planets checked whilst simulation running	Lines appear behind moving objects	Passed	
12	Trace planets unchecked	Trace planets unchecked whilst simulation running	Lines behind moving objects disappear	Passed	
13	Merge checked	Merge checked whilst simulation running and	Planets merge	Passed	

		two planets touching			
14	Merge unchecked	Merge unchecked whilst simulation running and two planets touching	Planets do not merge	Passed	
15	Show advanced checked	Show advanced checked	Advanced tab shows	Passed	
16	Show advanced unchecked	Show advanced unchecked	Advanced tab hidden	Passed	
17	Numeric up down - down	The down arrow on the numeric up down in the advanced tab is clicked	Number decreases by one	Passed	
18	Numeric up down - up	The up arrow on the numeric up down in the advanced tab is clicked	Number increases by one		
19	Numeric up down – non integer	The non-integer value 'G' is entered to the numeric up down	Number does not change		
20	Numeric up down – greater than limit	A value of 11 is entered which is higher than the maximum limit for this input	It should revert to 10		
21	Numeric up down – less than limit	A value of -1 is entered which is lower than the minimum value for this input	It should revert to 0		
22	Numeric up down – max acceptable value	A value of 10 is entered which is the highest value this input will take	It should be accepted and simulation should run at speed 10		
23	Numeric up down –	A value of 0 is entered which	It should be accepted and		

	min acceptable value	is the lowest value this input will take	simulation should run at speed 0		
24	Canvas click – radius and density in range	Click on canvas with add planet tool selected where radius and density are 30 and 400 respectively (these are within the accepted range 1 to 120 for radius and 1 to 1000 for density)	Planet will be drawn on click location	Passed	
25	Validation on radius with a non integer	Enter a value of 20.7 to radius	Message box with “Radius must be integer between 1 and 120” should appear	Passed	
26	Validation on radius- greater than range	Enter a value of 121 to radius	Message box with “Radius must be integer between 1 and 120” should appear	Passed	
27	Validation on radius – less that range	Enter a value of 0 to radius	Message box with “Radius must be integer between 1 and 120” should appear	Passed	
28	Validation on radius maximum edge	Enter a value to the radius of 120	Planet should be drawn	Passed	
29	Validation on radius minimum edge	Enter a value to the radius of 1	Planet should be drawn	Passed	
30	Validation on radius against strings	Enter a value of ‘string’ to radius	Message box with “Radius must be integer between 1 and 120” should appear		

31	Validation on density with a non integer	Enter a value of 500.6 to the density	Message box with “Density must be integer between 1 and 1000” should appear	Passed	
32	Validation on density- greater than range	Enter a value of 1001	Message box with “Density must be integer between 1 and 1000” should appear	Passed	
33	Validation on density – less that range	0	Message box with “Density must be integer between 1 and 1000” should appear	Passed	
34	Validation on density maximum edge	1000	Message box with “Density must be integer between 1 and 1000” should appear	Passed	
35	Validation on density minimum edge	1	Message box with “Density must be integer between 1 and 1000” should appear	Passed	
36	Validation on density against strings	Enter a value of ‘string’ to density	Message box with “Density must be integer between 1 and 1000” should appear		
37	Draw when add planet selected	After selecting add planet I will click on the screen to see if a planet is drawn.	Planet with specified radius, density and colour is created on my click location		
38	Planet size	I will draw a three planets, one with radius 5, one with radius 40 and one with radius 120	The planets with the larger radii should appear proportionally larger on the screen		
39	Planet colours	I will draw five planets all the	The blue planet should appear		

		same apart from that they will each have a different colour	as such when draw, as should the red and the blue etc.		
40	Delete planet correct	I will create a planet and then click within its radius using the delete tool	It should vanish as it will have been deleted		
41	Delete planet incorrect	I will create a planet and click outside of its radius using the delete tool	The planet should remain unchanged		

loading dialogue

Test Number	Test	Test Description	Expected outcome	Passed/ Failed	Comments
42	Load valid text file	Loading in a valid text file that has been created by the program using the save function	Window closes and simulation is loaded		
43	Access non-text file	Try to view and load a non – text file	Cannot view a non-text file as text file is the only file option viewable		
44	Click cancel	Open the load dialogue and click the cancel button	Window closes		
45	Click close in dialogue window	Open the load dialogue and click the close window button in the top right corner	Window closes		

Saving dialogue

Test Number	Test	Test Description	Expected outcome	Passed/ Failed	Comments
46	Save file	I will created a simulation and then save it	The window closes and the		

			saved file has been created		
47	Click cancel	I will open the save dialogue and then attempt to close it using the cancel button next to the save button	The window closes		
48	Click close window	I will open the save dialogue and then attempt to close it using the close window button in the top right corner of the dialogue	The window closes		
49	Click save when null	Click the save button when the file name textbox is empty	Clicking save has no effect		
50	Save over existing	I will click on an existing file instead of typing a new name into the file name textbox	It should have an ‘are you sure message’		

Unit testing screenshots

Test 1: Add button click

Before:

After:

Test 2: Delete button click

Before:

After:

Test 3: information button click

Before:

After:

Test 4: play button click (playing)

Before:

After:

Test 5: Play button click (paused)

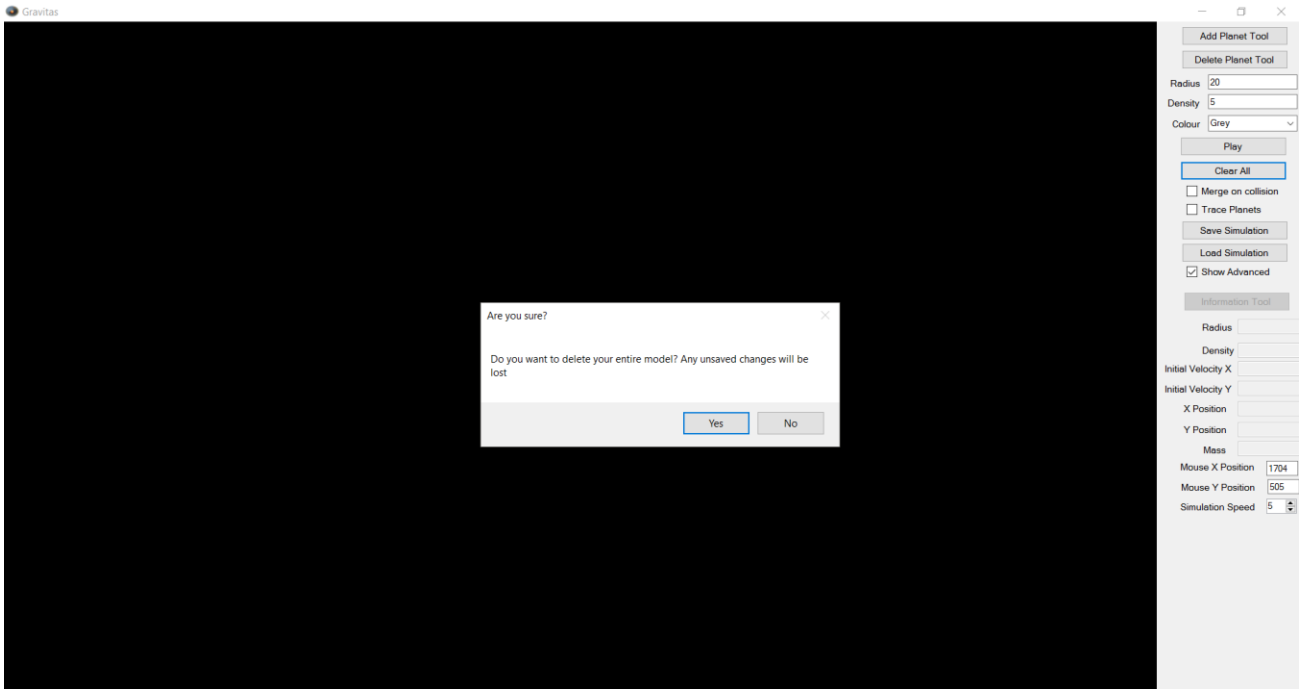
Before:

After:

Test 6: clear all clicked

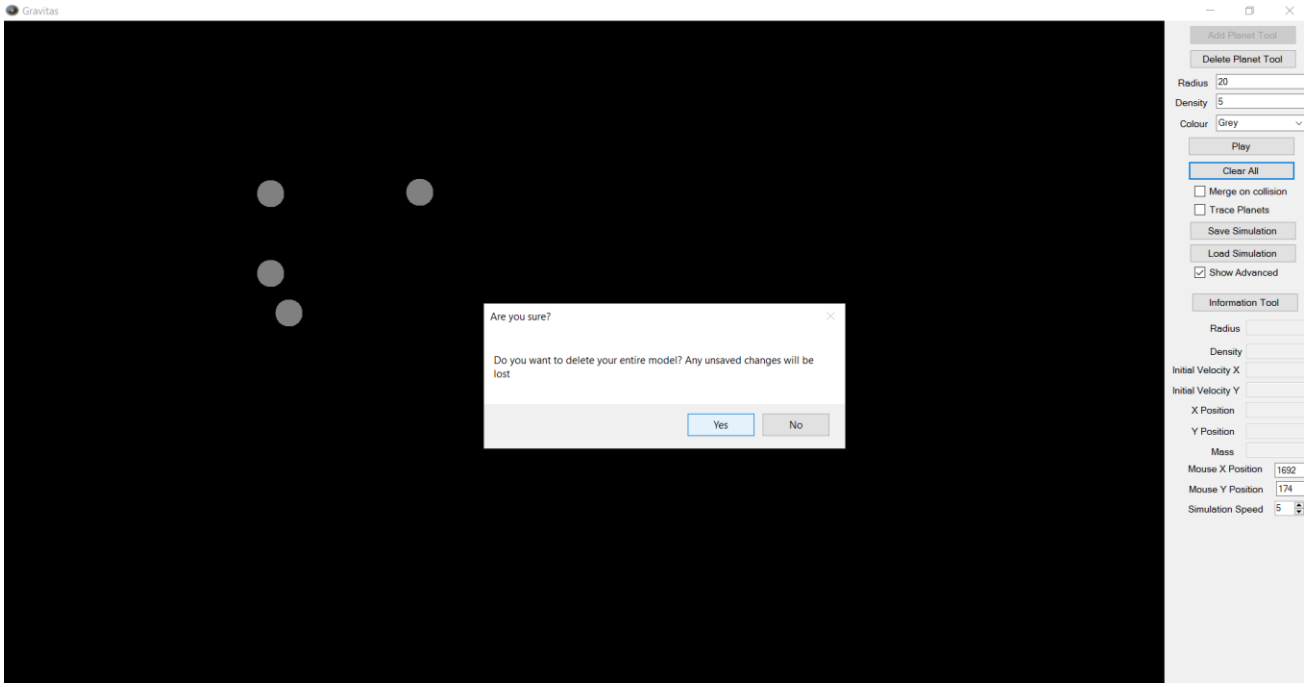
Before:

After:

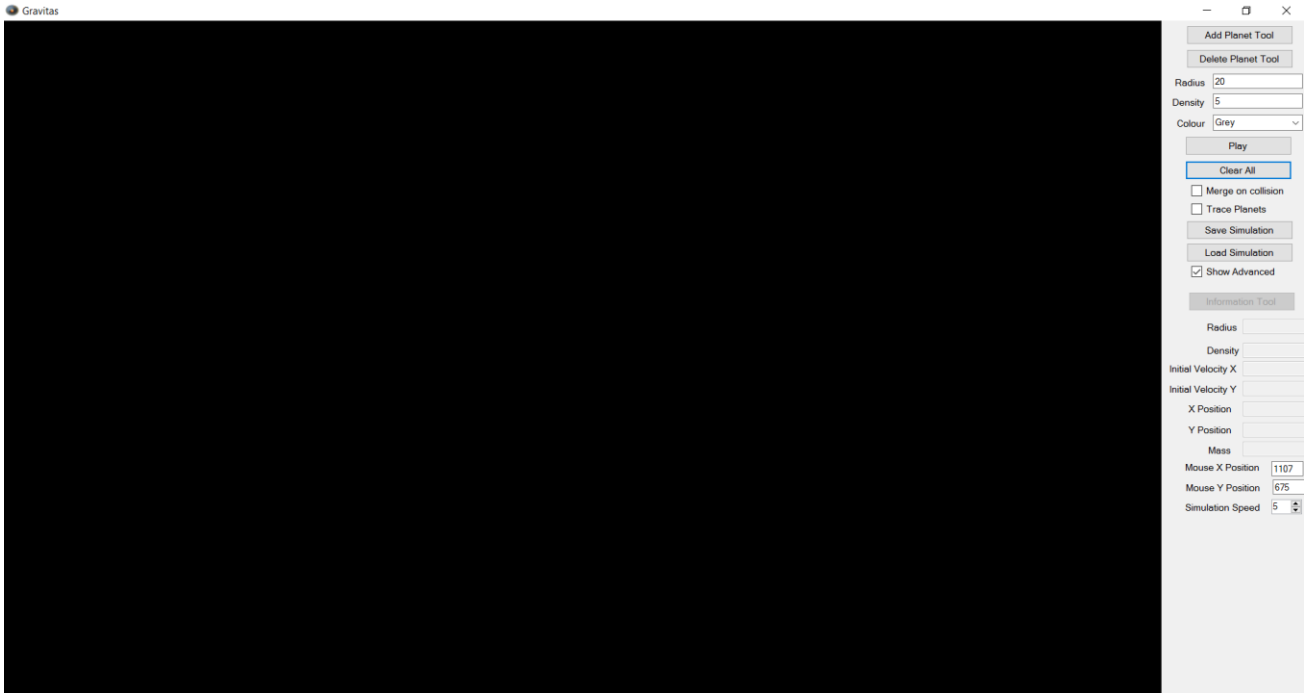


Test 7: click yes on clear all

Before:

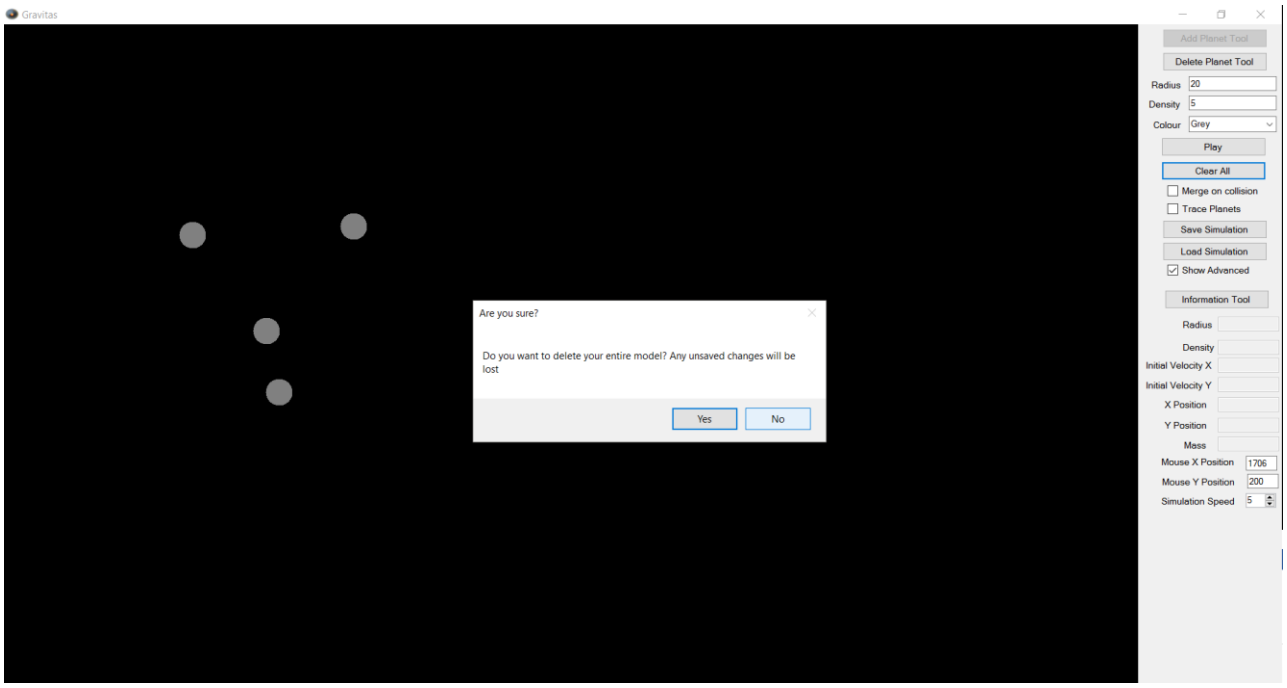


After:

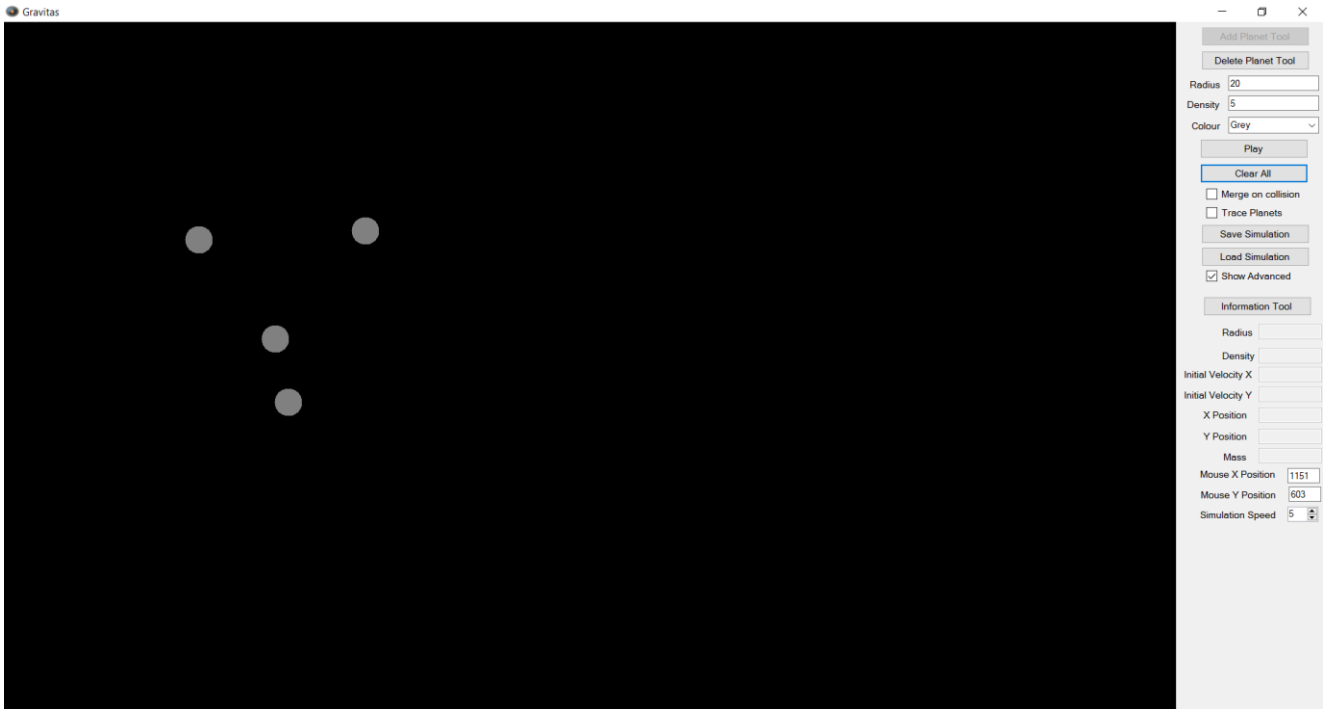


Test 8: Click no on clear all

Before:

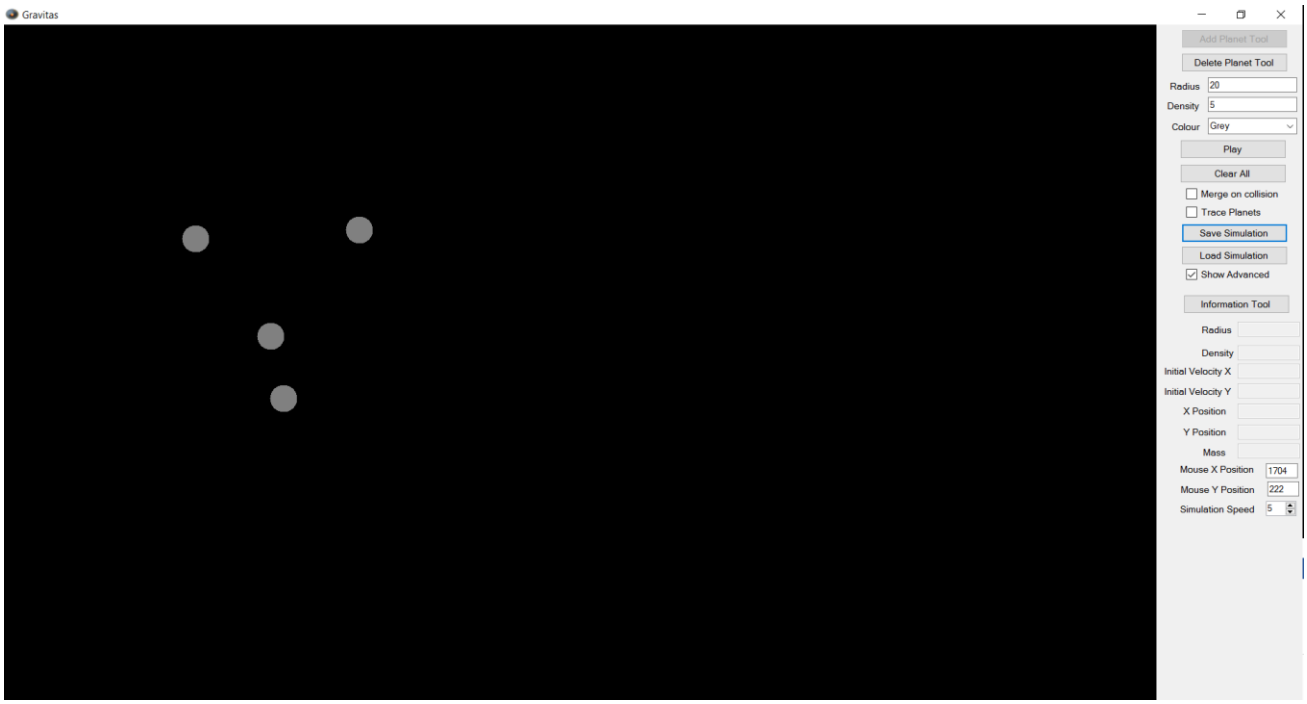


After:

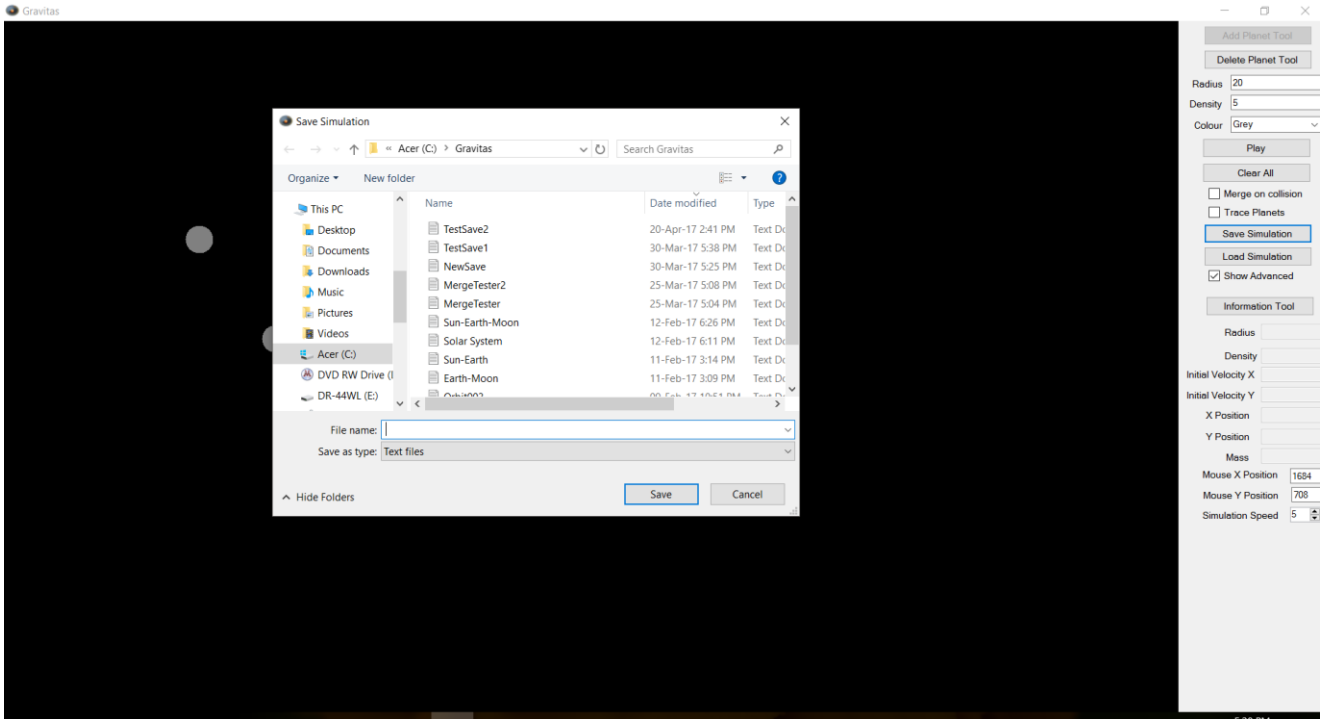


Test 9: click save

Before:

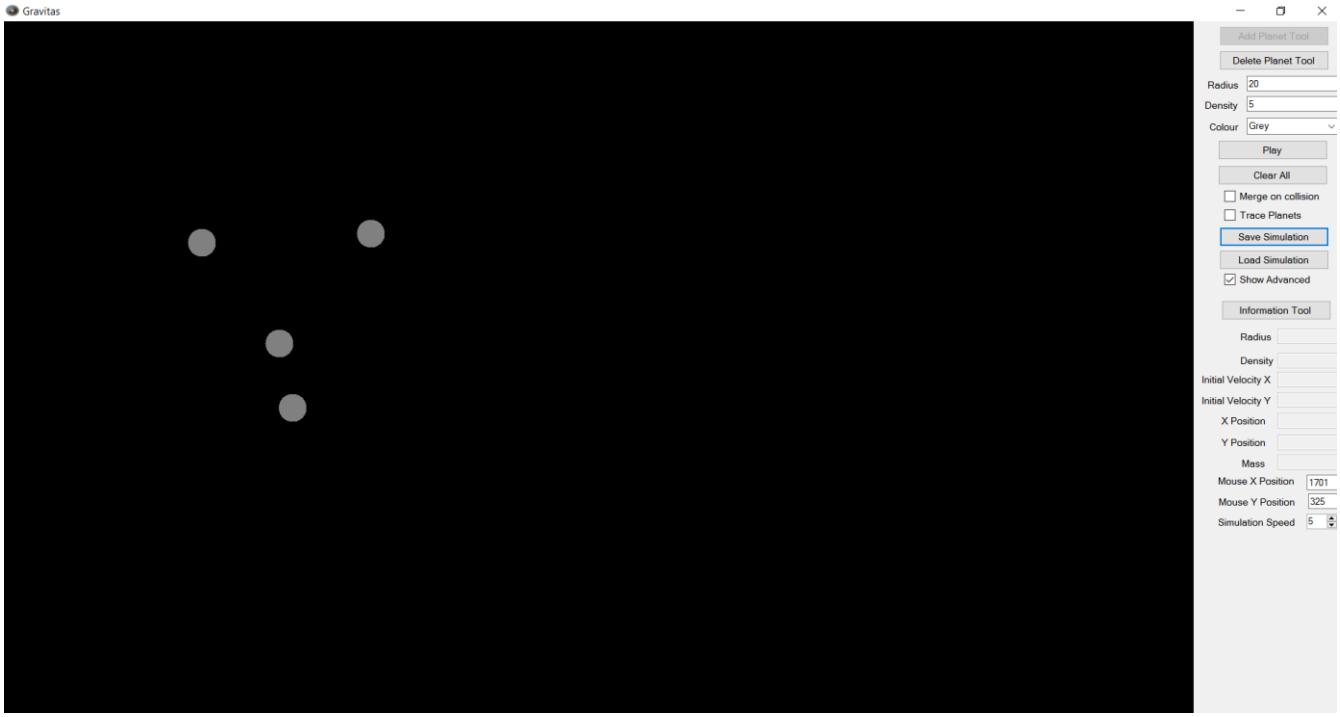


After:

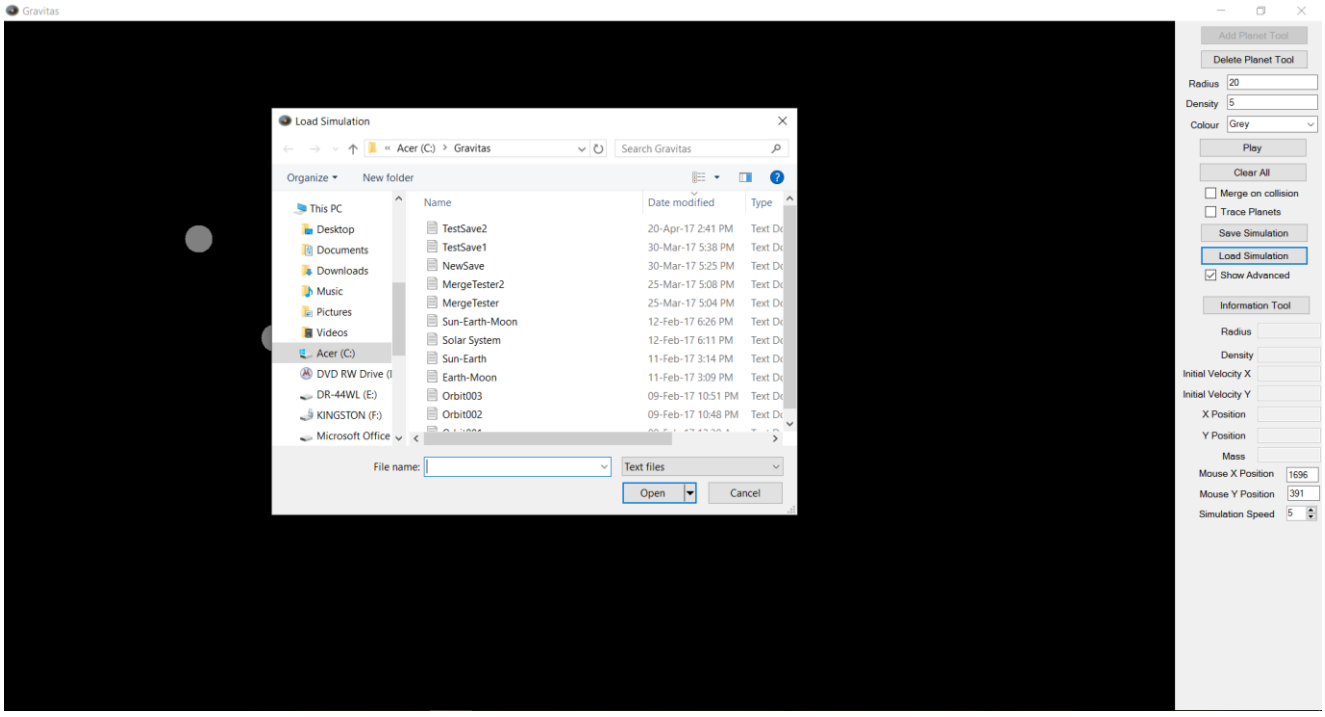


Test 10: Click load

Before:

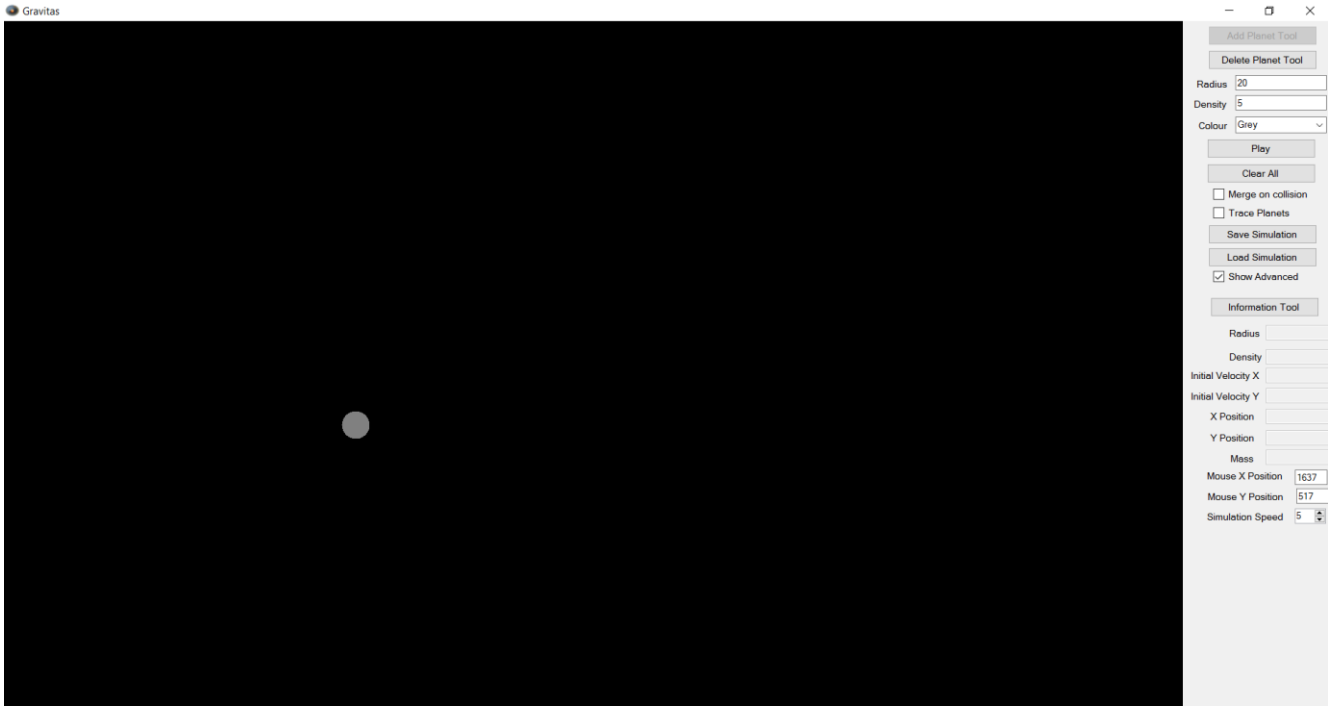


After:

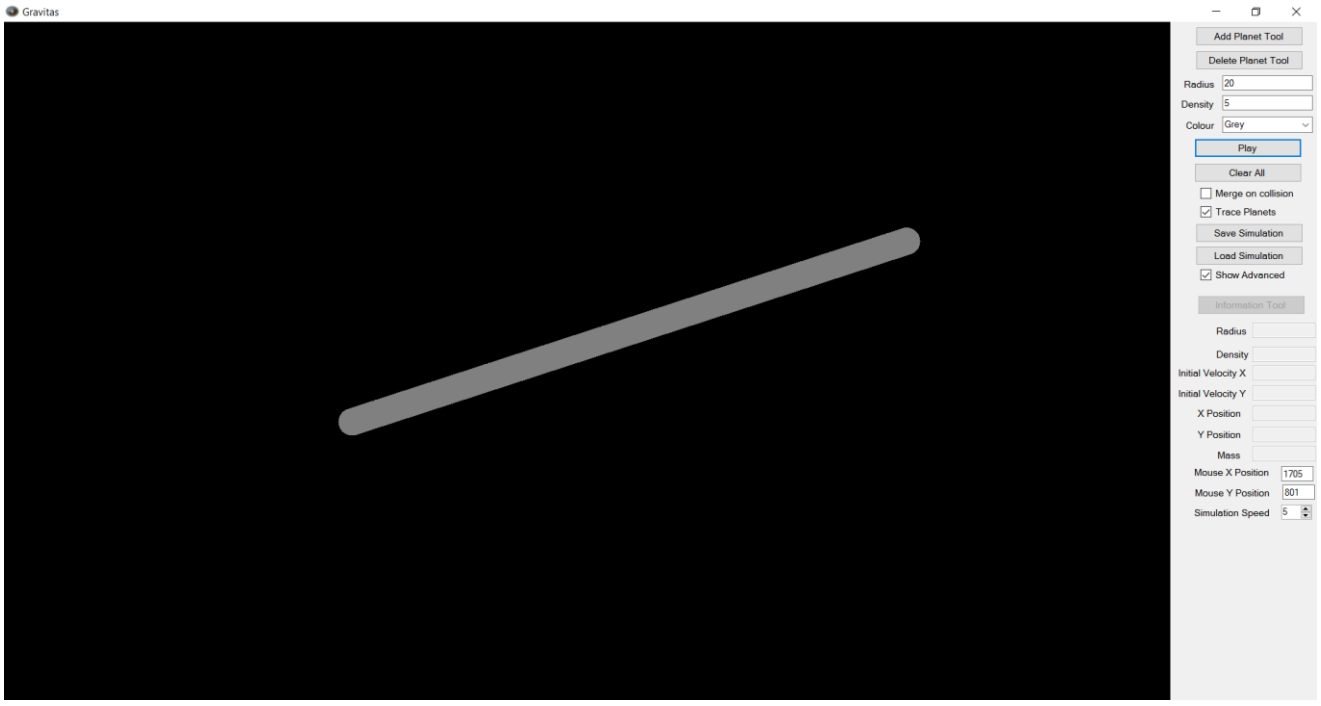


Test 11: Trace planets checked

Before:

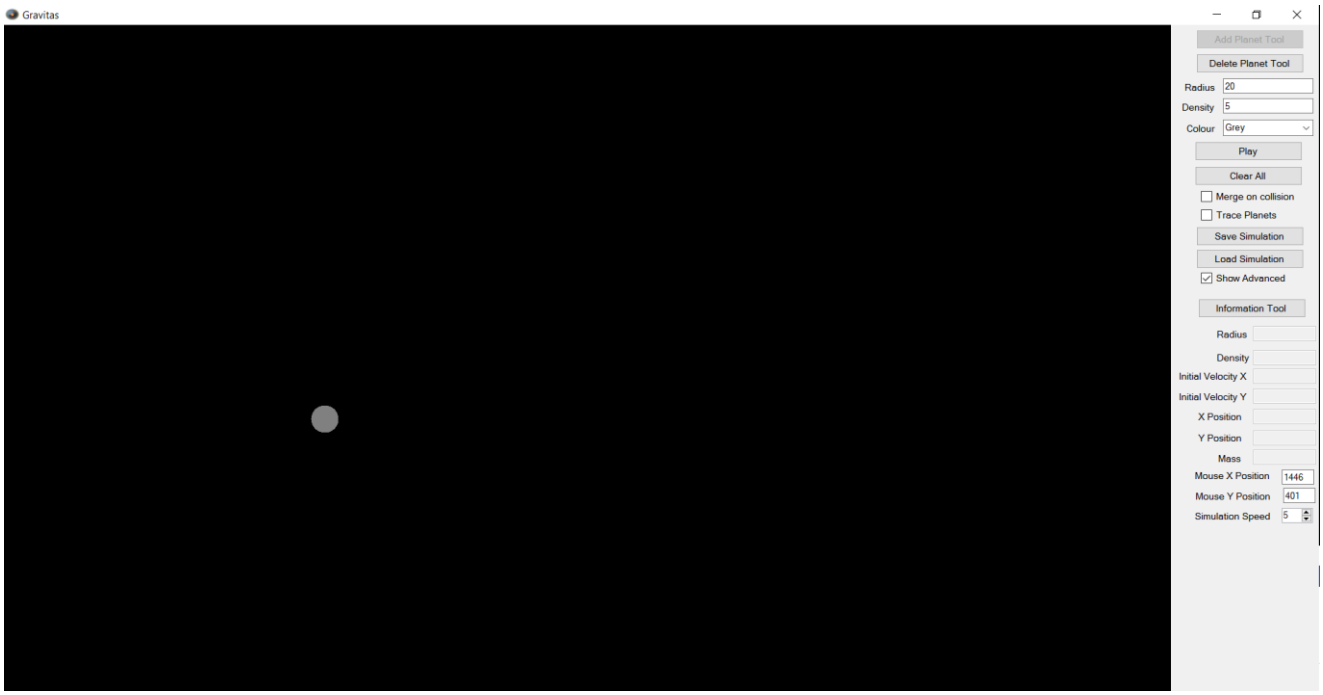


After:

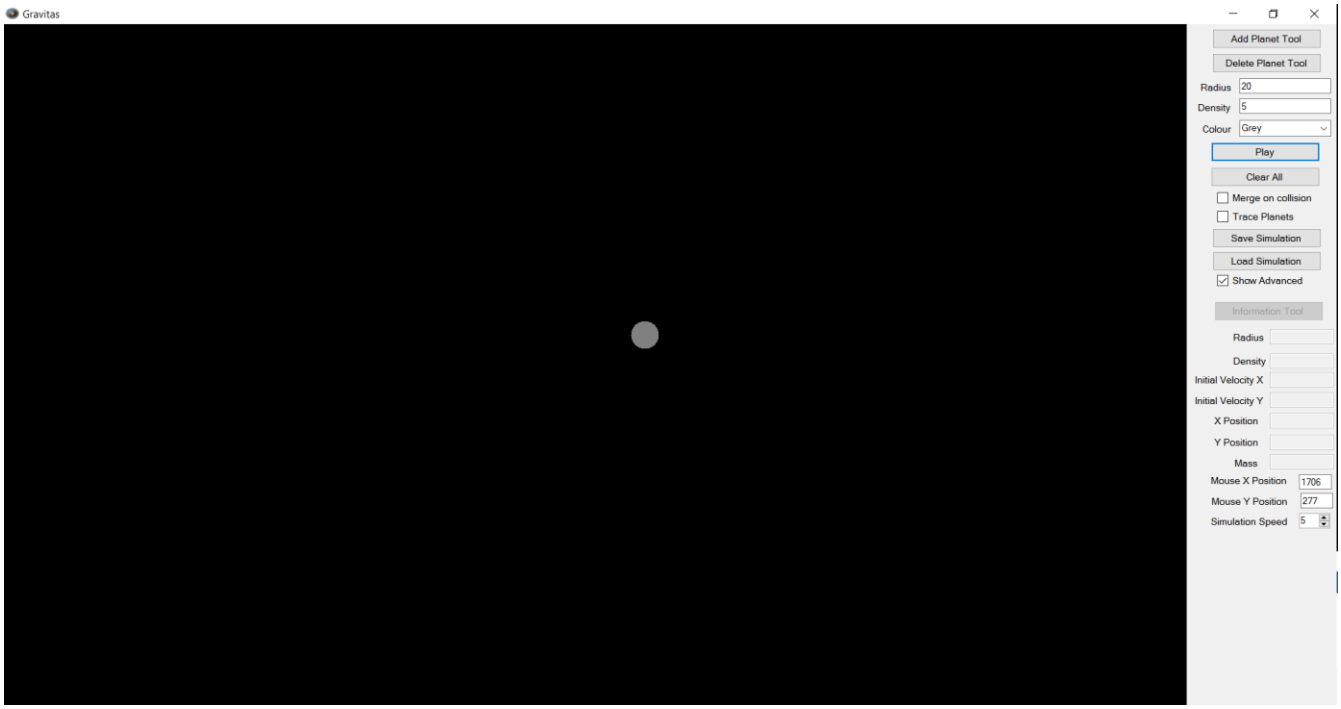


Test 12: Trace planets unchecked

Before:

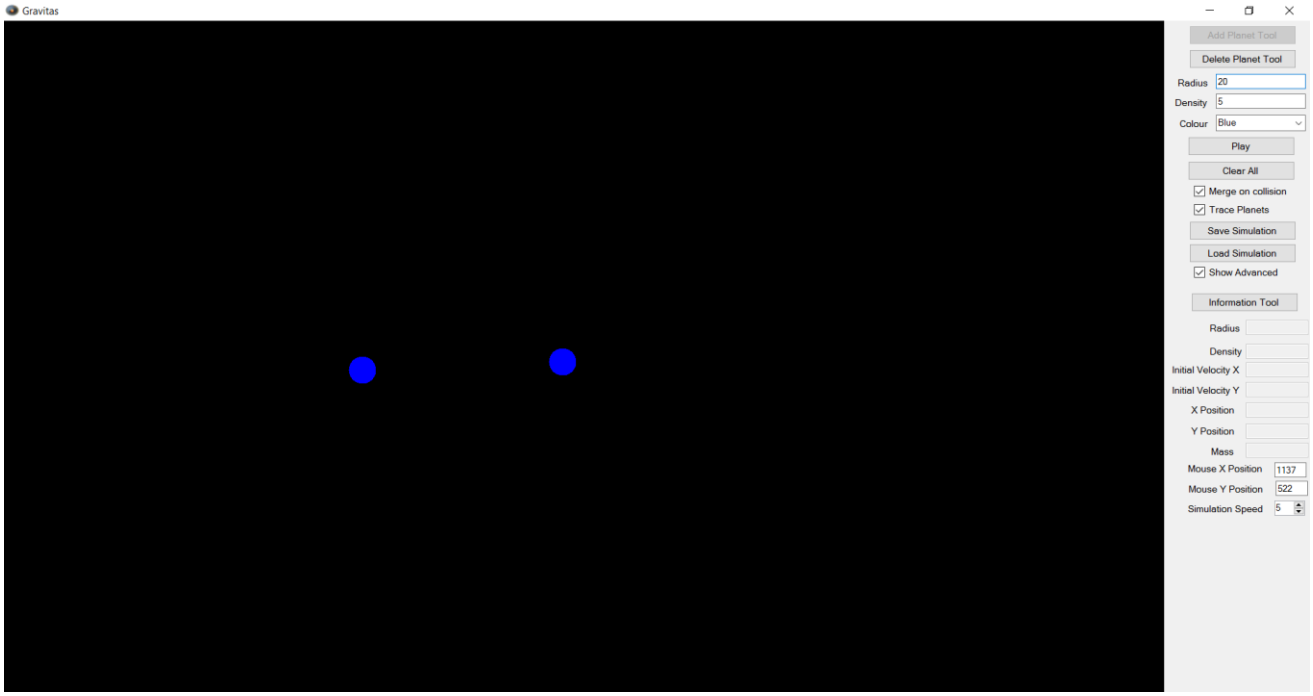


After:

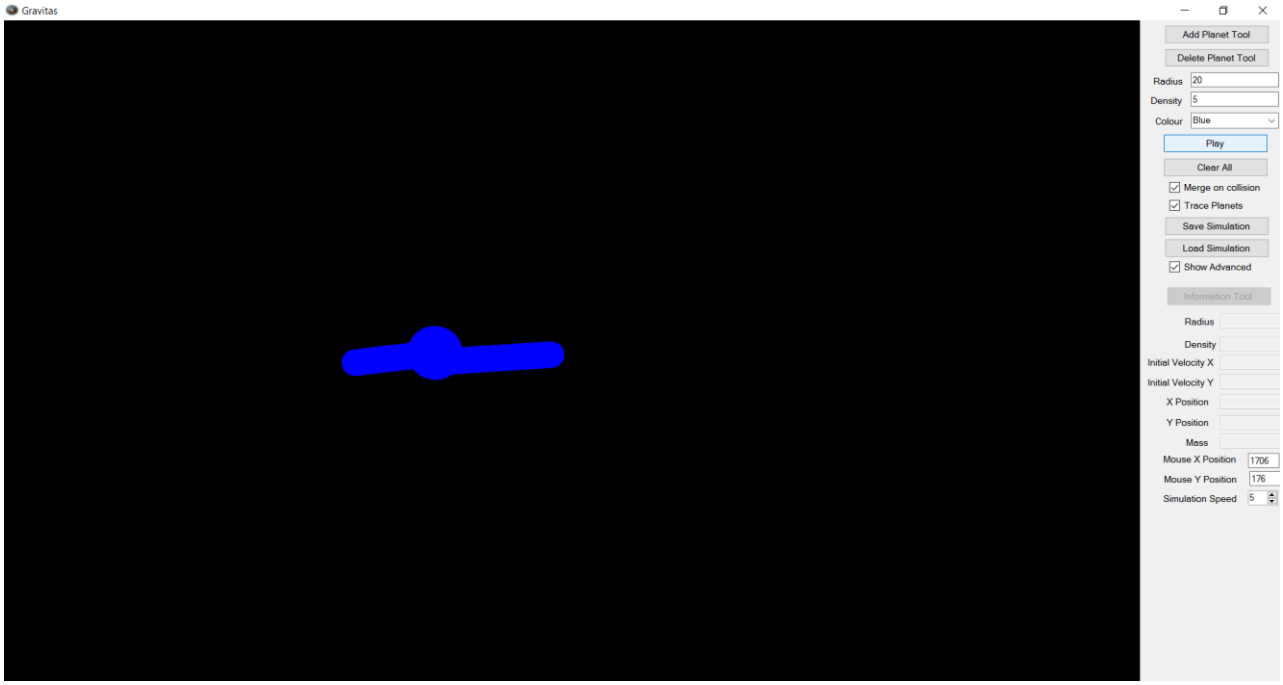


Test 13: merge checked

Before:

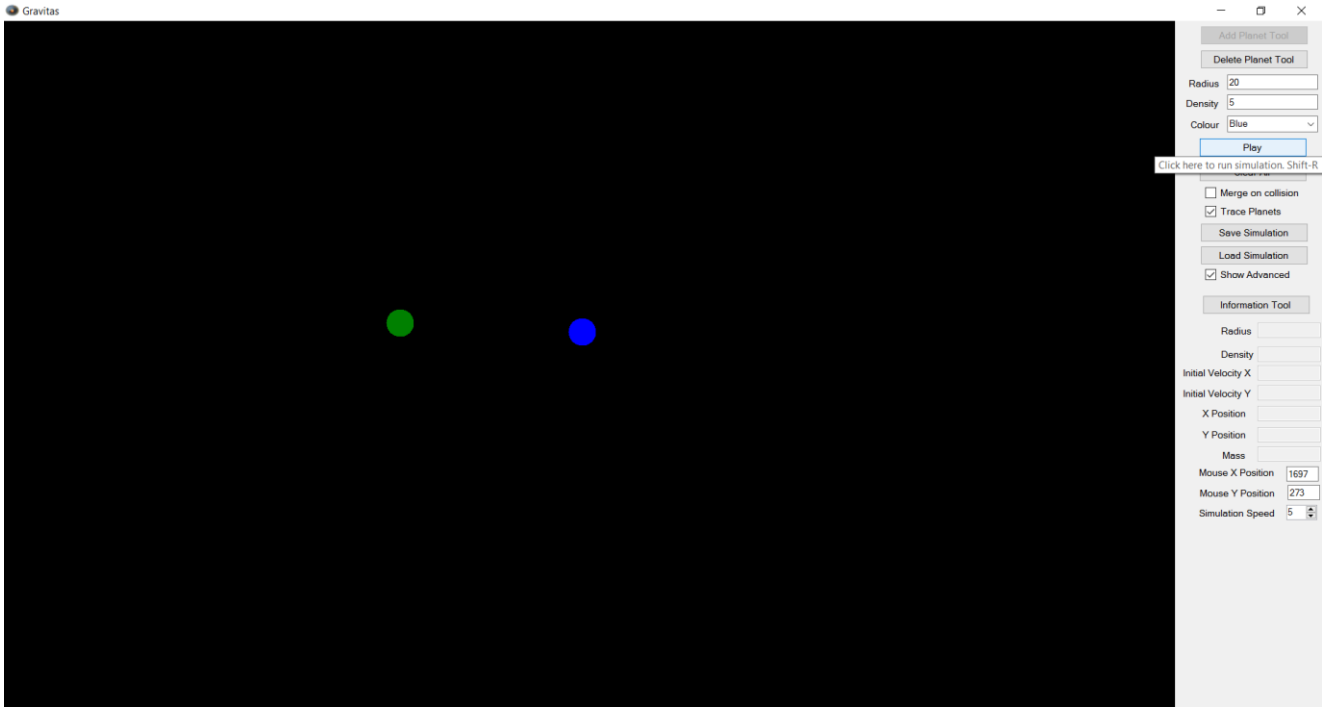


After:

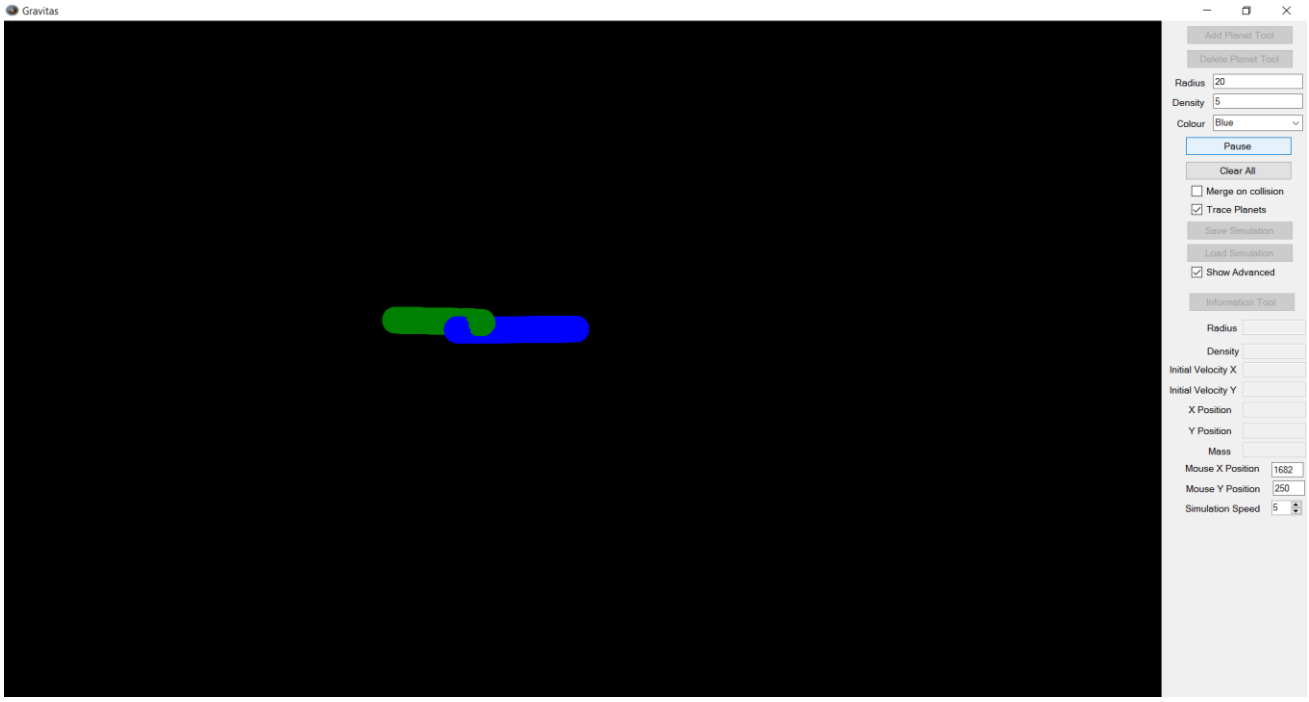


Test 14: merge unchecked

Before:



After:



Test 15: show advanced checked

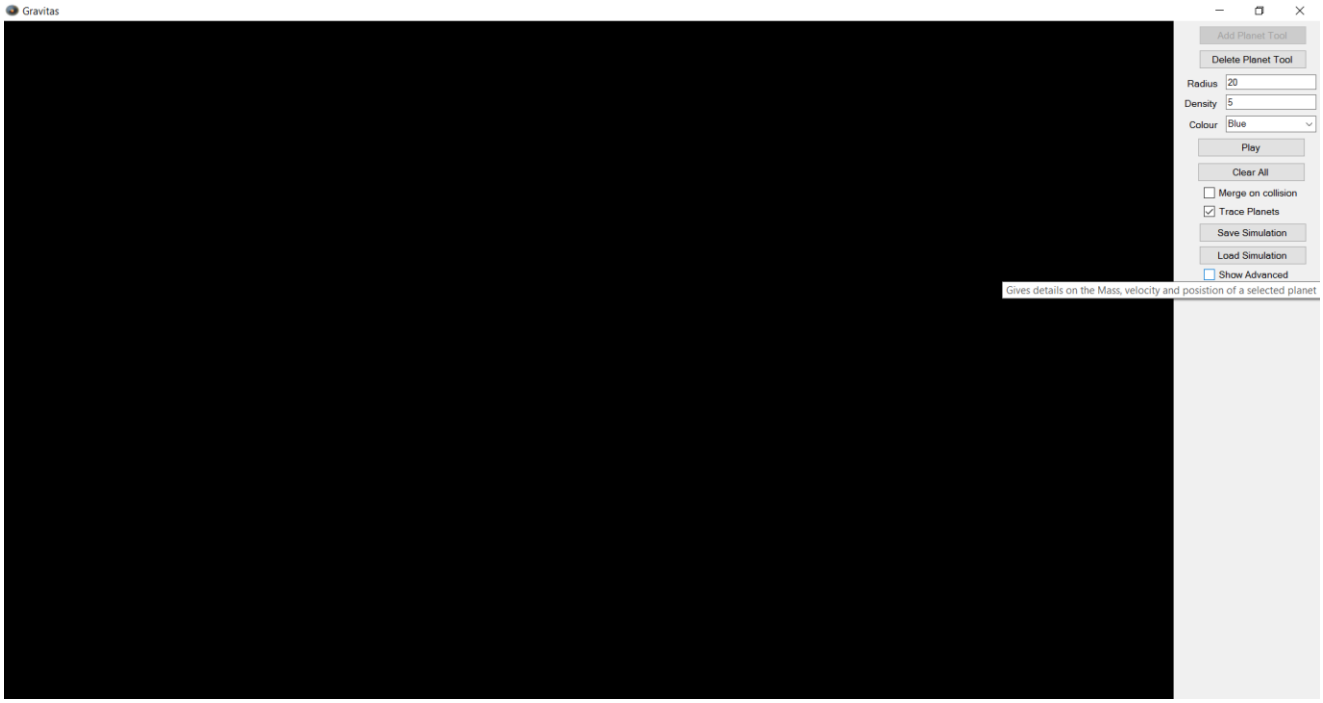
Before:

After:

Test 16: show advanced unchecked

Before:

After:



Test 17: numeric up down - down

Before:

After:

Test 18: numeric up down - up

Before:

After:

Test 19: numeric up down – non-integer

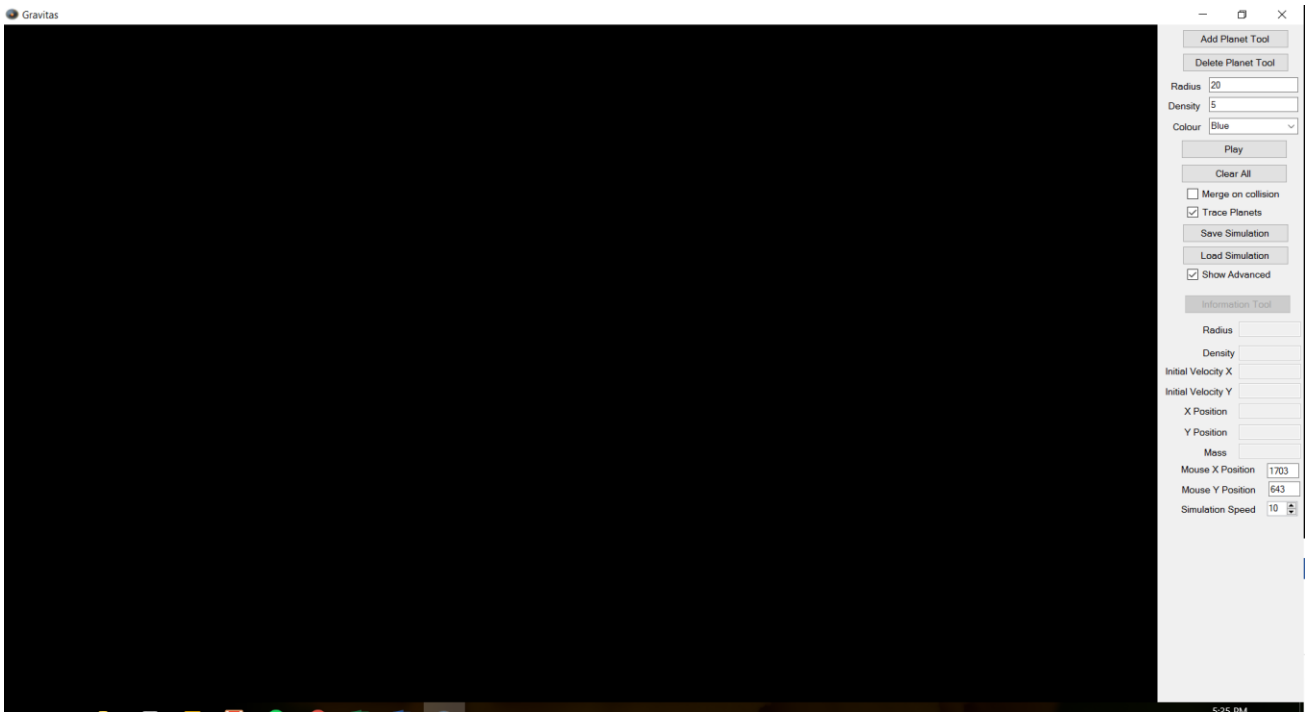
Before:

After:

Test 20: numeric up down – greater than acceptable range

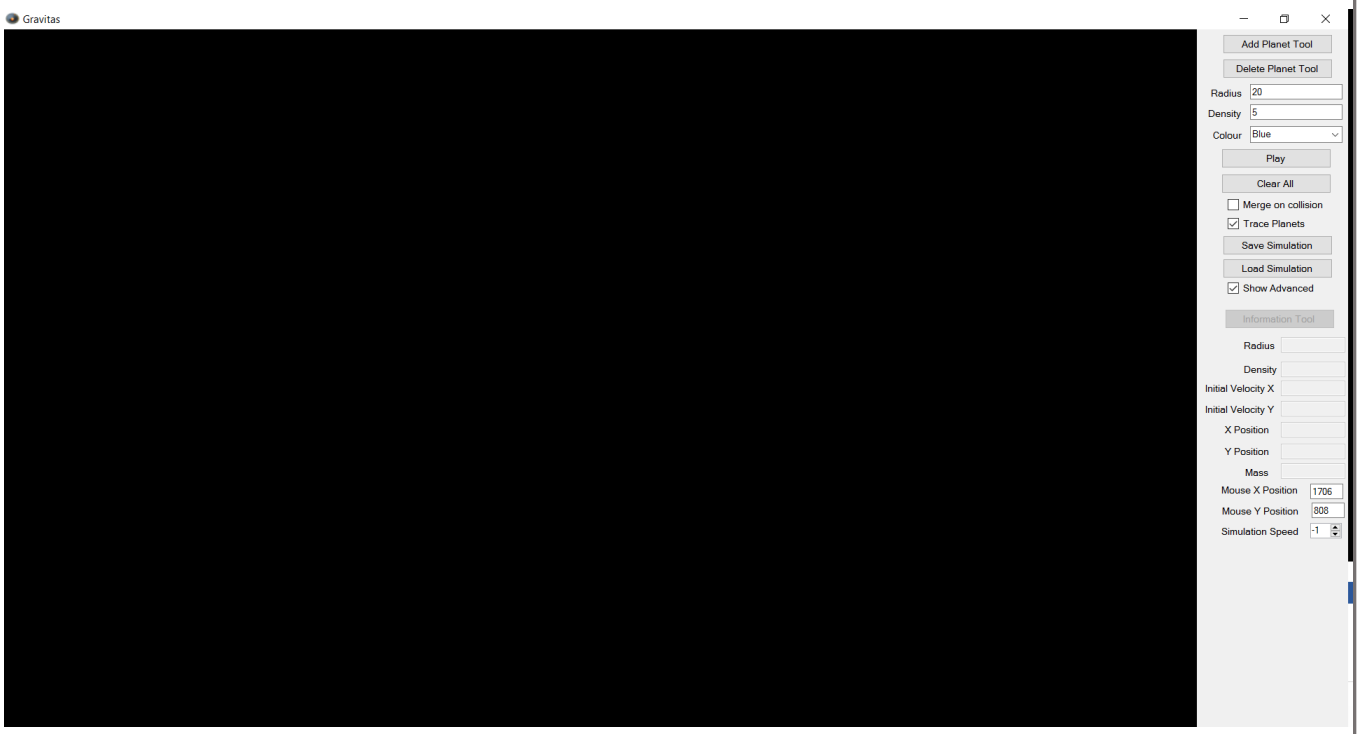
Before:

After:



Test 21: numeric up down – less than acceptable range

Before:



After:

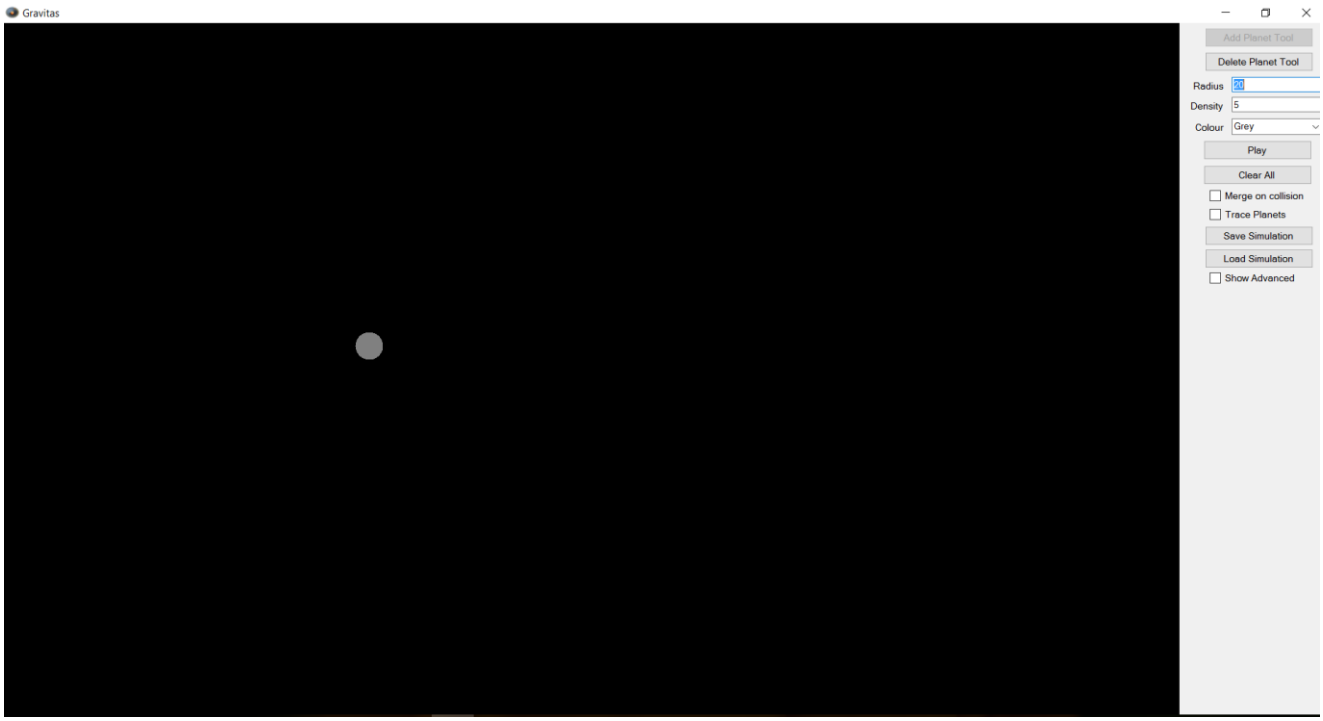
Test 22: numeric up down – max acceptable value

Test 23: numeric up down – min acceptable value

Test 24: Radius and density – acceptable value

Before:

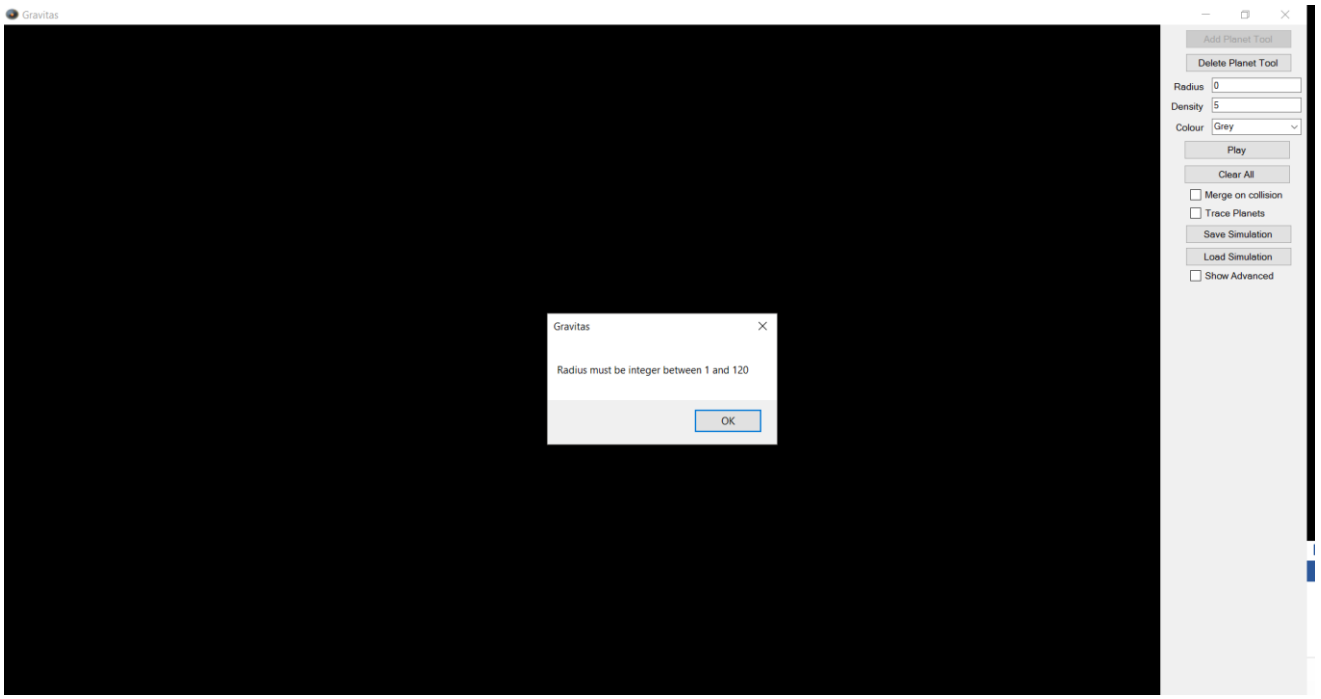
After:



Test 25: Radius – less than acceptable range

Before:

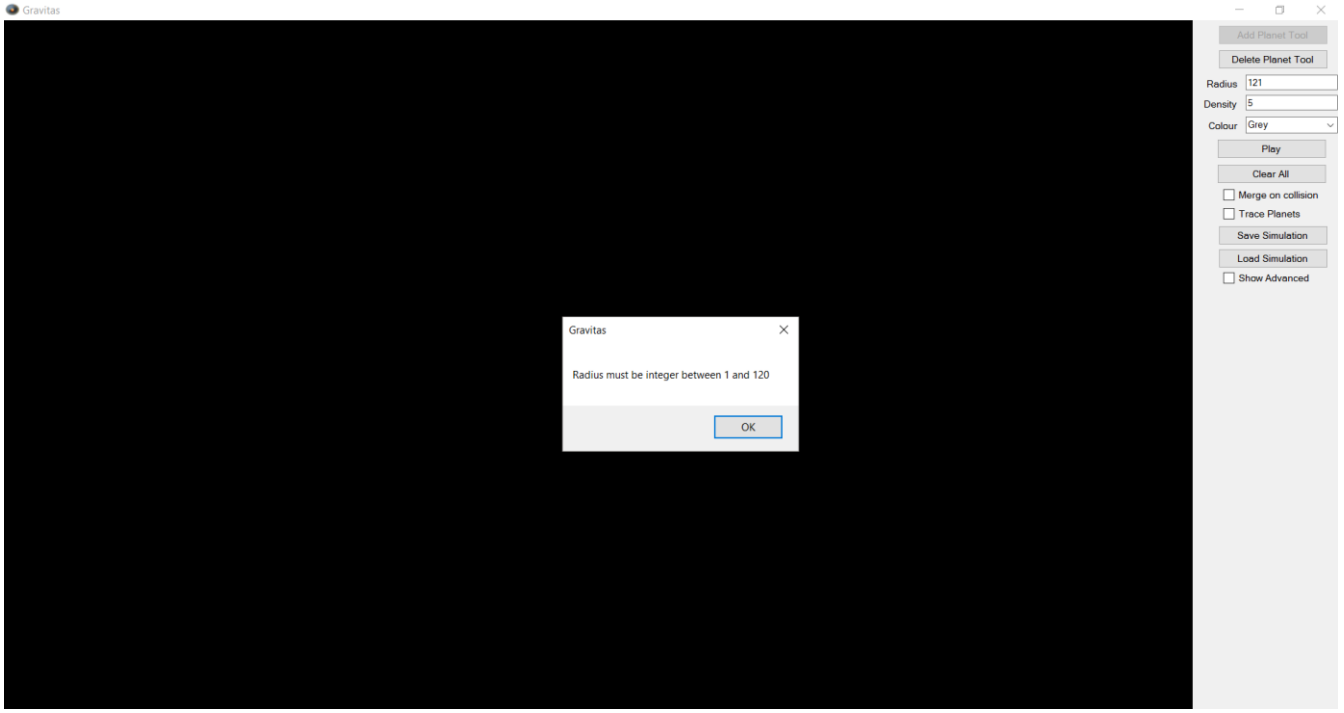
After:



Test 26: Radius – greater than acceptable range

Before:

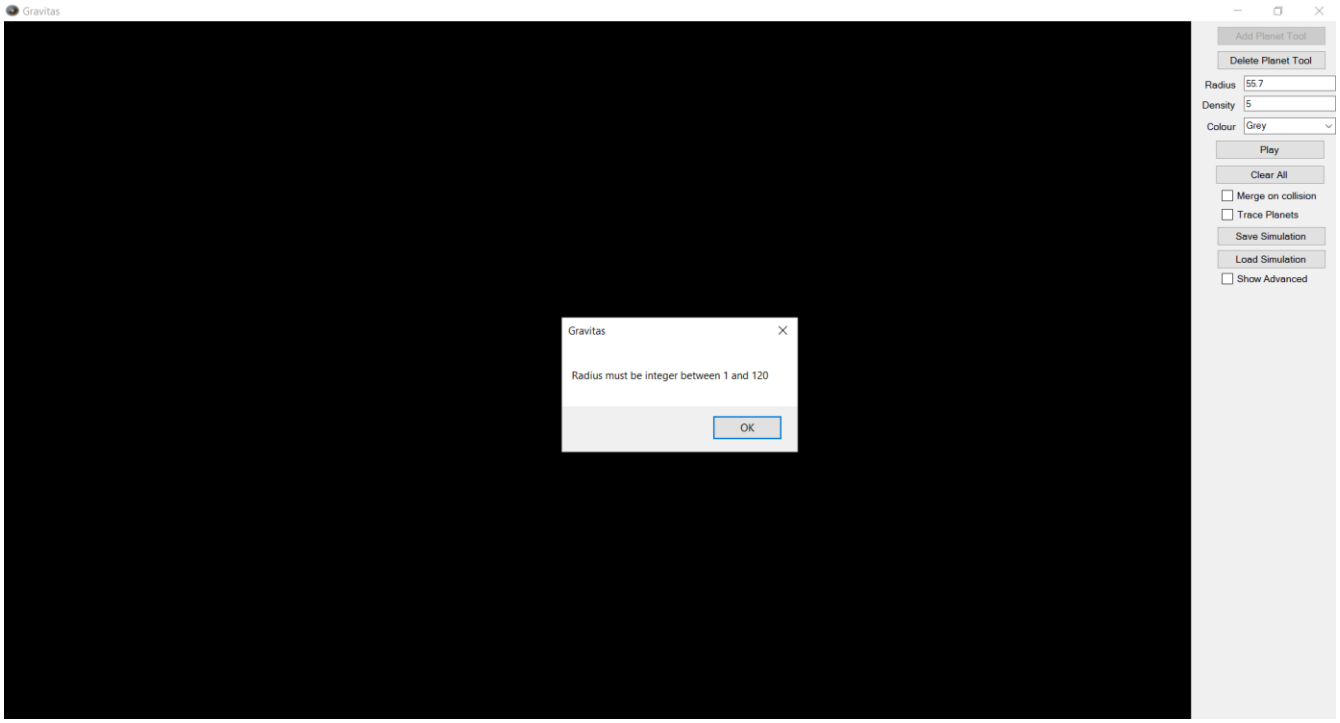
After:



Test 27: Radius – non-integer

Before:

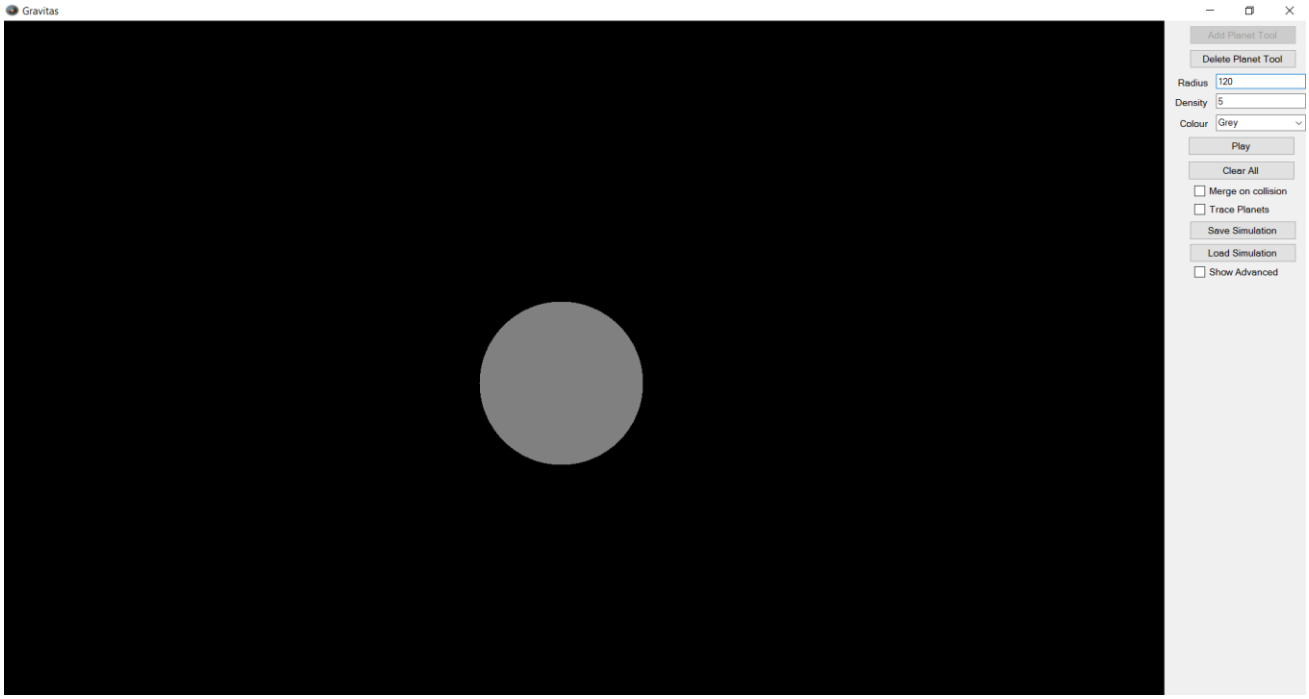
After:



Test 28: Radius – max acceptable value

Before:

After:



Test 29: Radius – min acceptable value

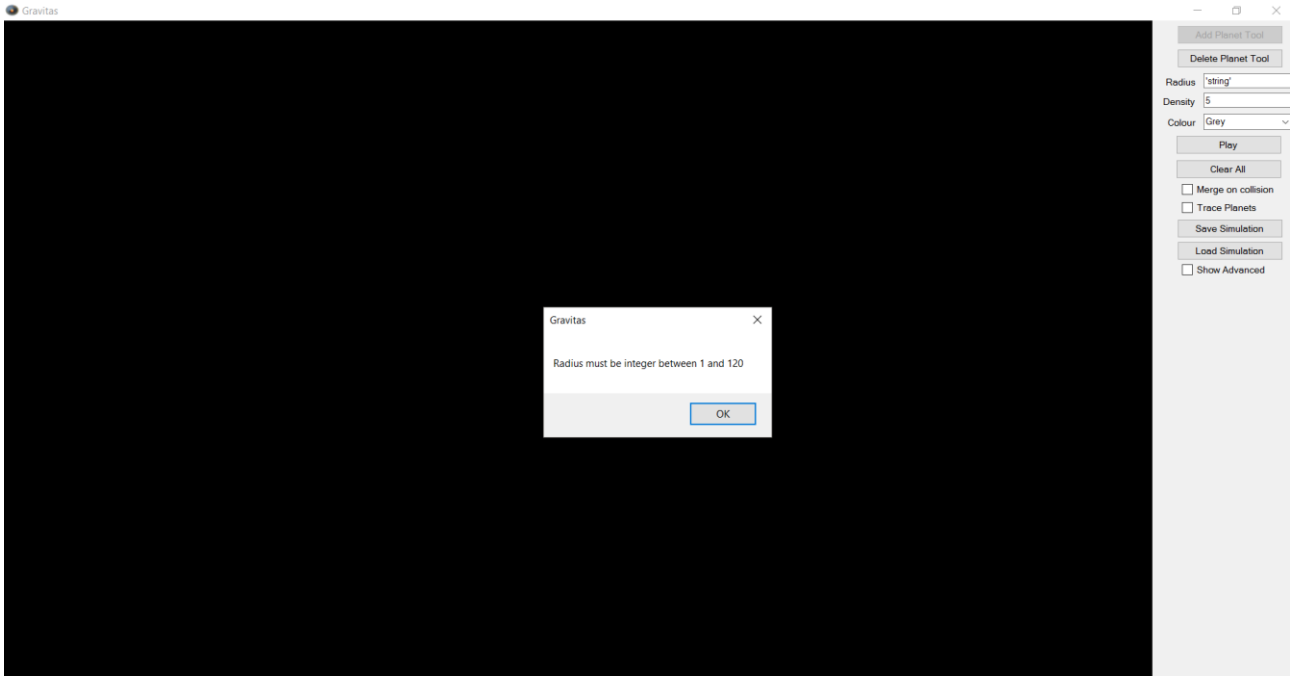
Before:

After:

Test 30: Radius – String

Before:

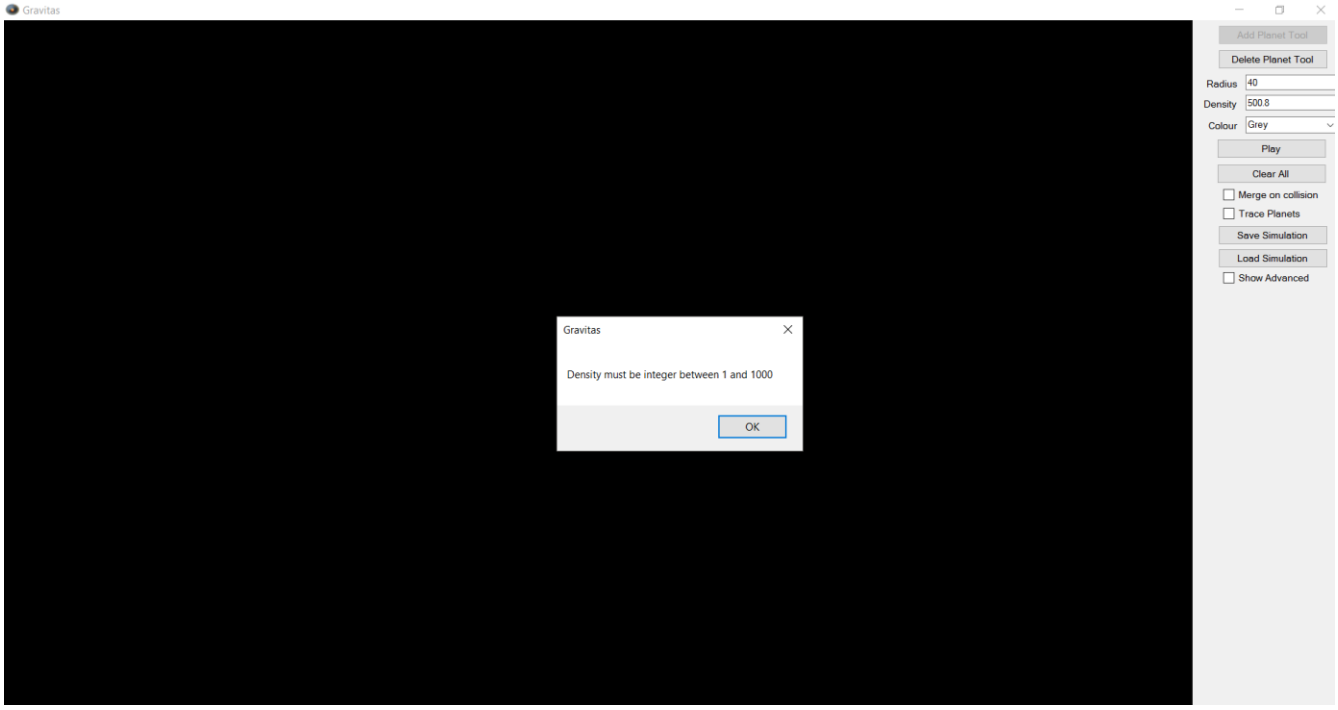
After:



Test 31: Density – non-integer

Before:

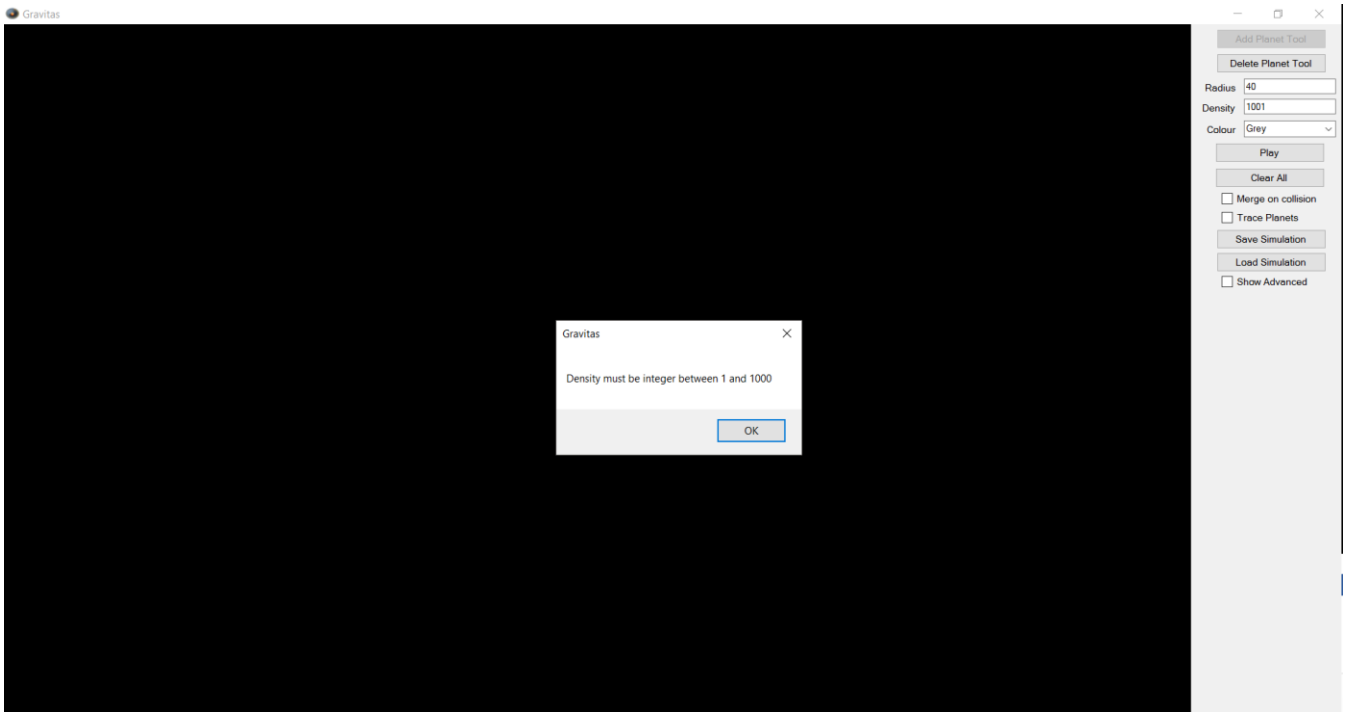
After:



Test 32: Density – greater than acceptable range

Before:

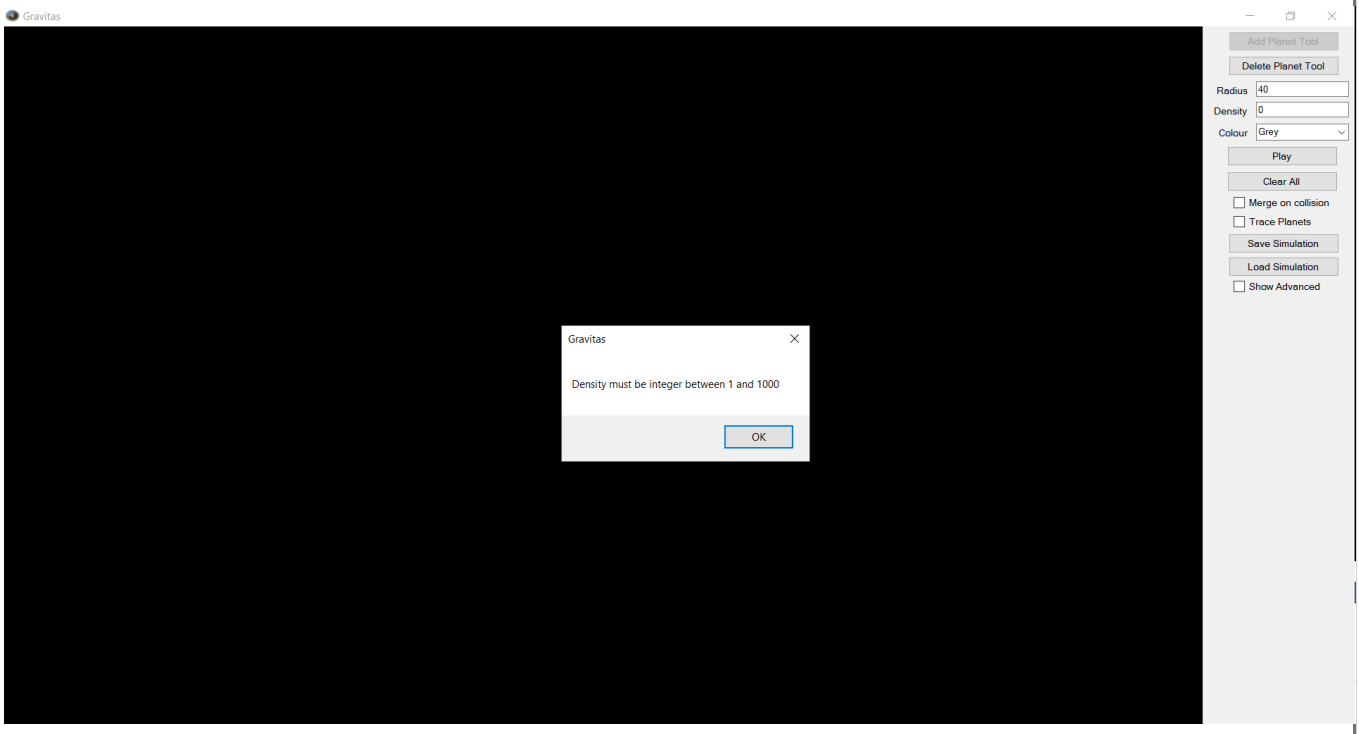
After:



Test 33: Density – less than acceptable range

Before:

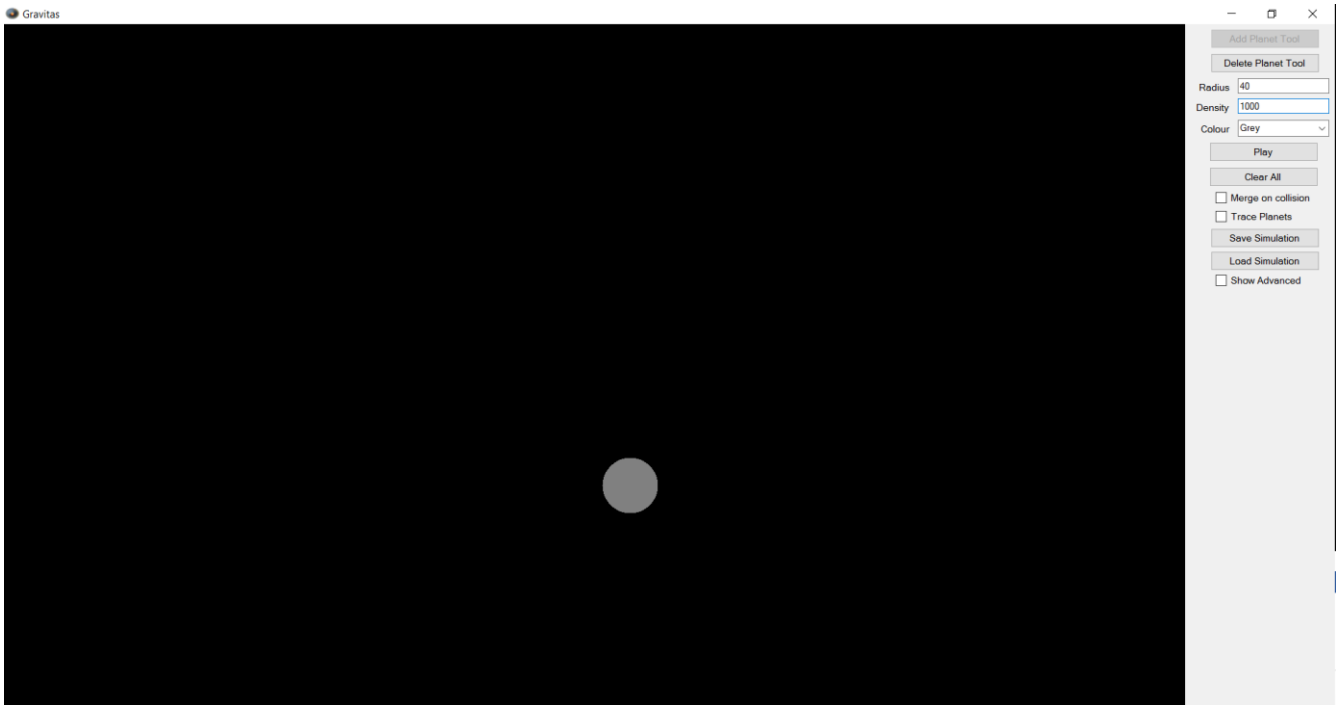
After:



Test 34: Density – max acceptable value

Before:

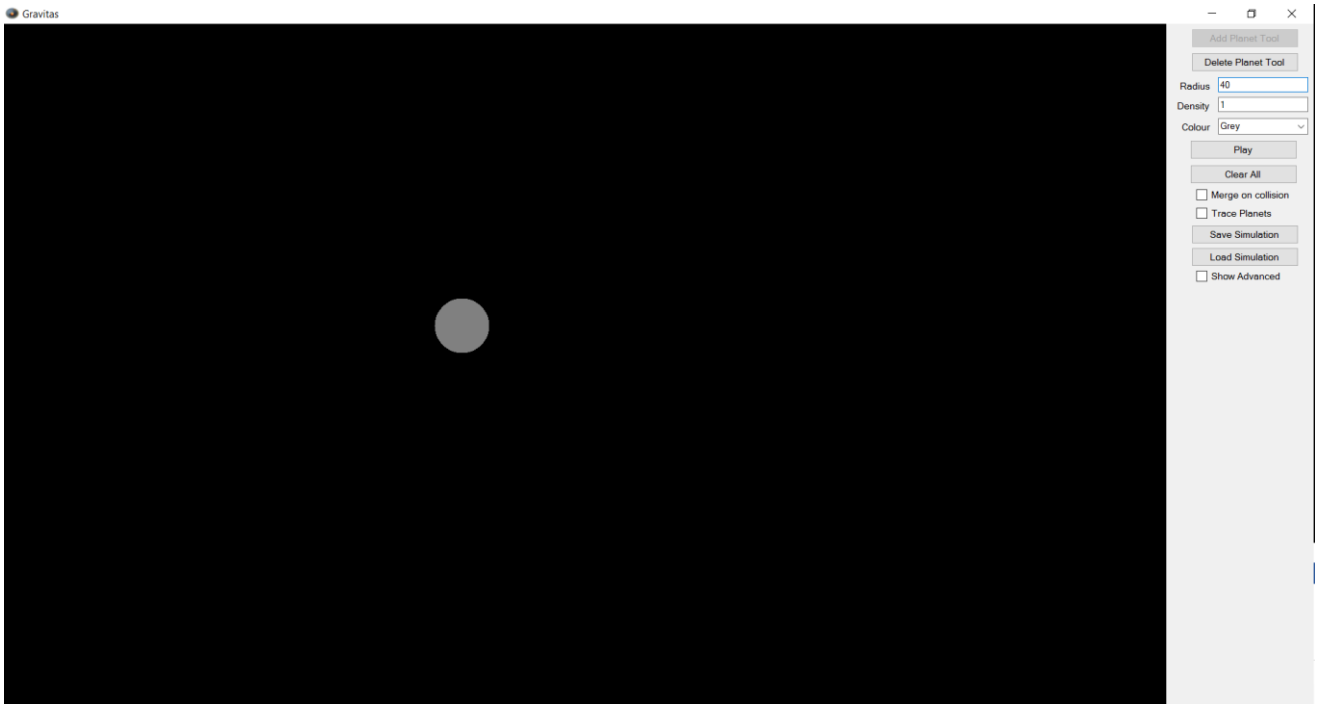
After:



Test 35: Density – min acceptable value

Before:

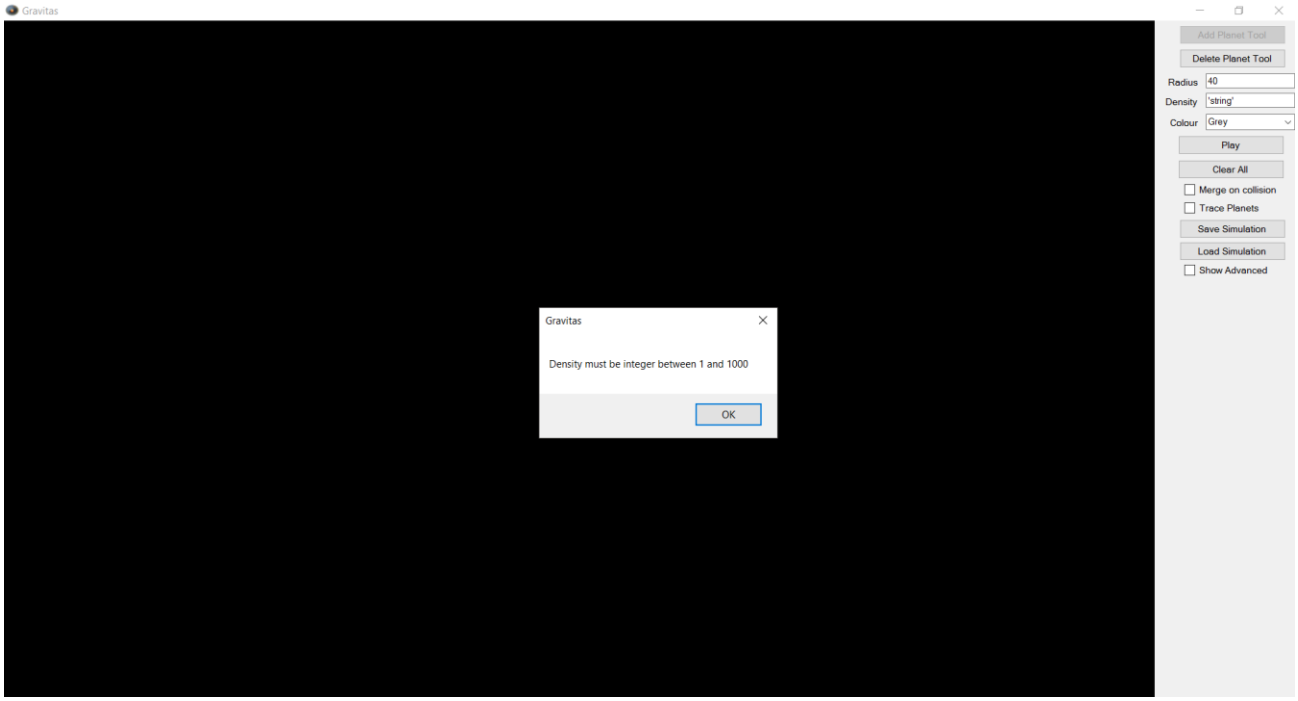
After:



Test 36: Density – String

Before:

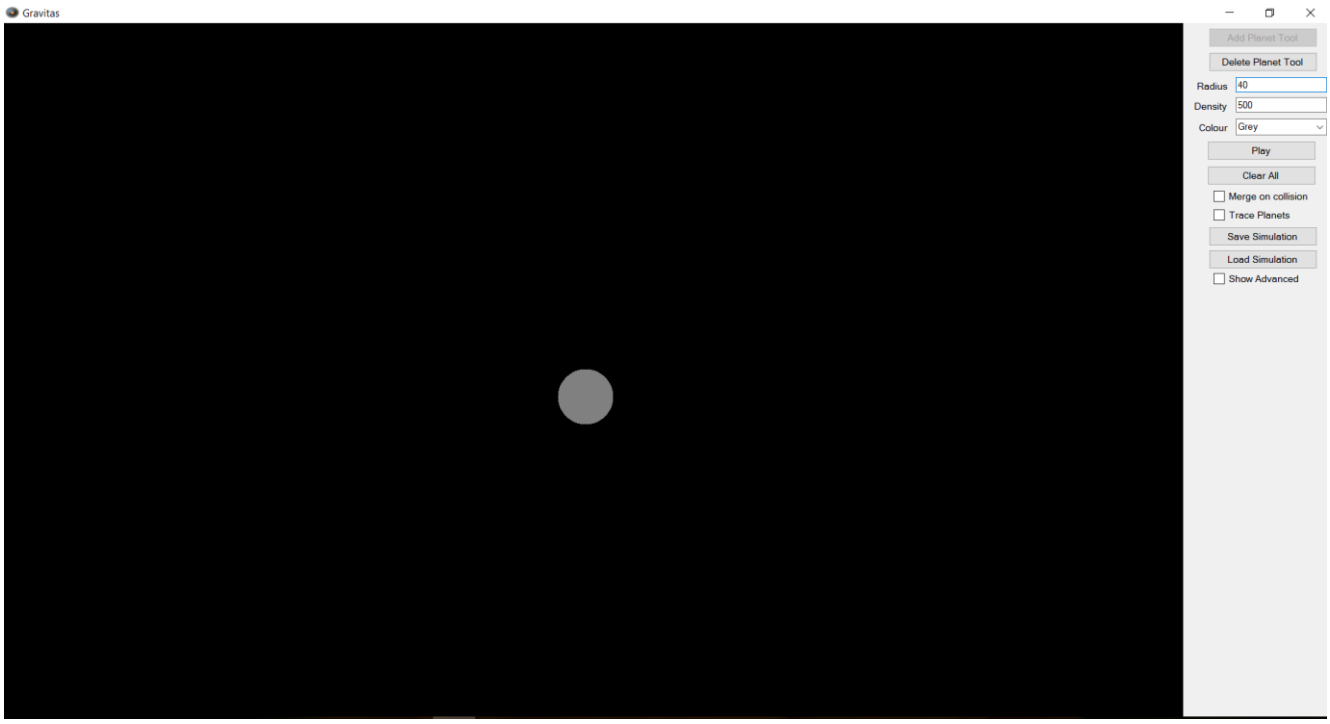
After:



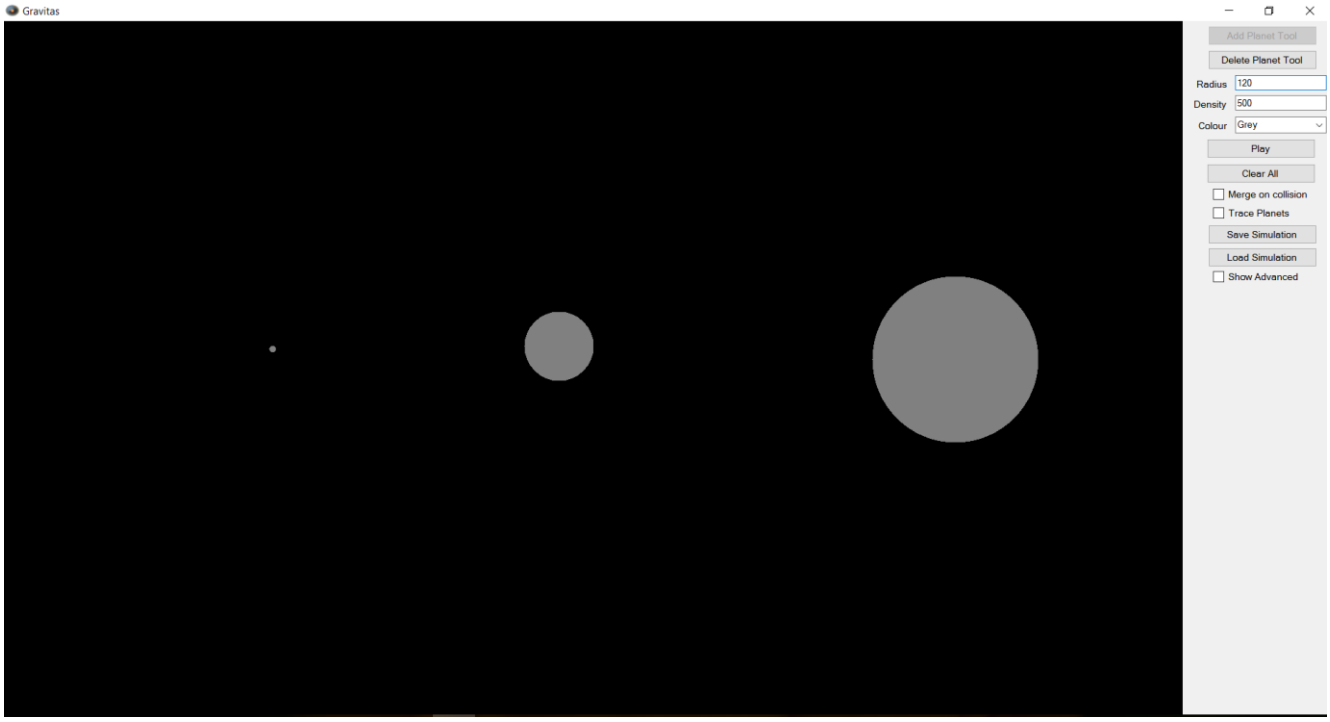
Test 37: Draw planet

Before:

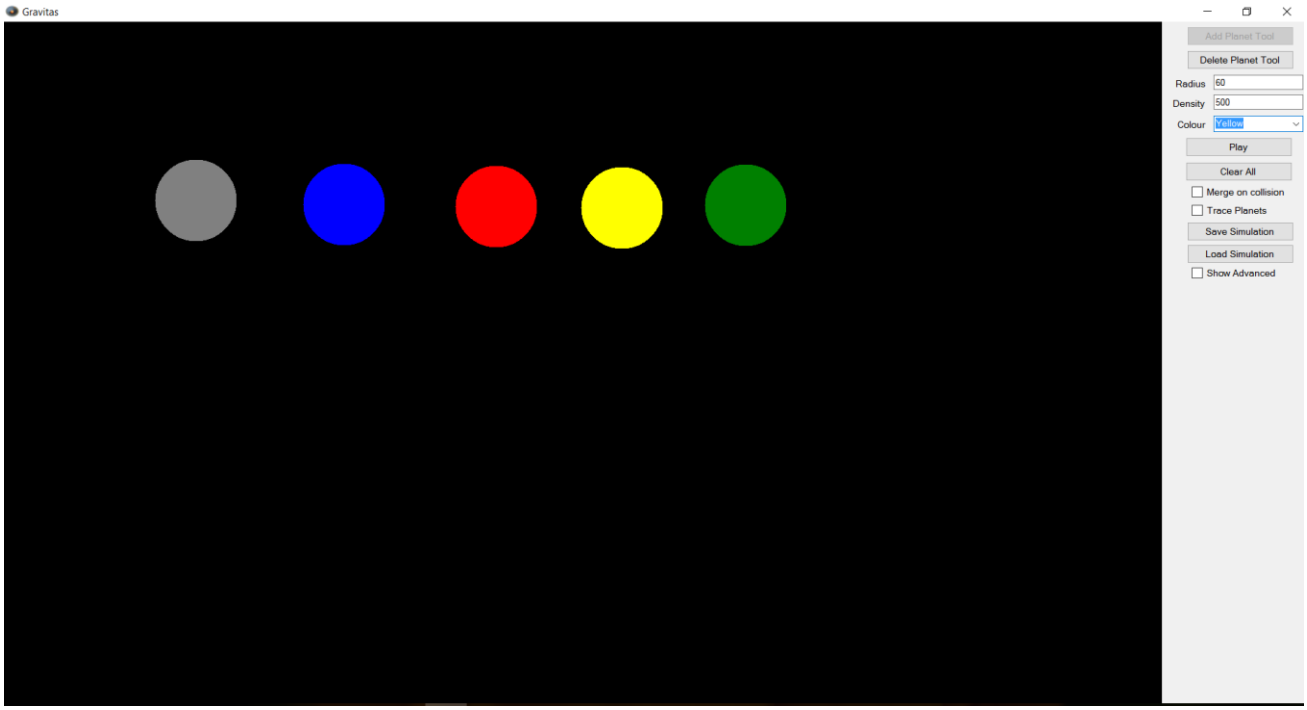
After:



Test 38: change planet size

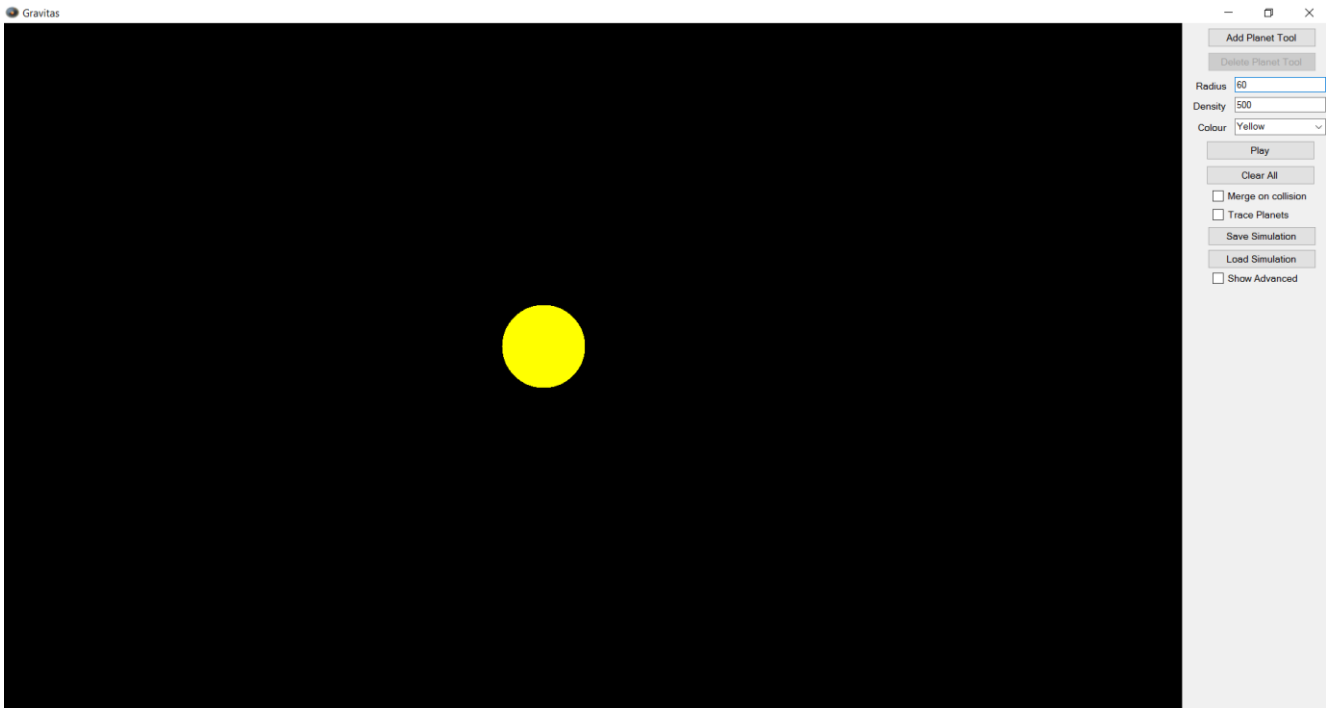


Test 39: change planet colour



Test 40: delete planet (inside radius)

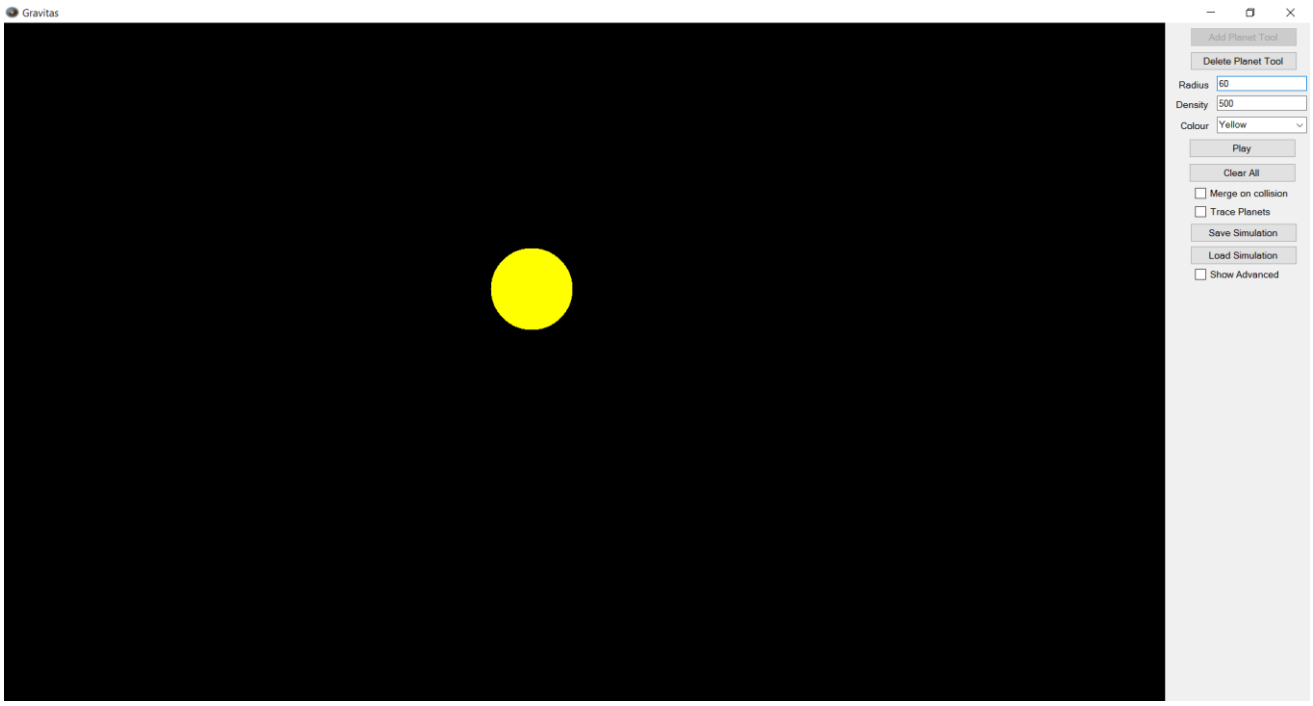
Before:



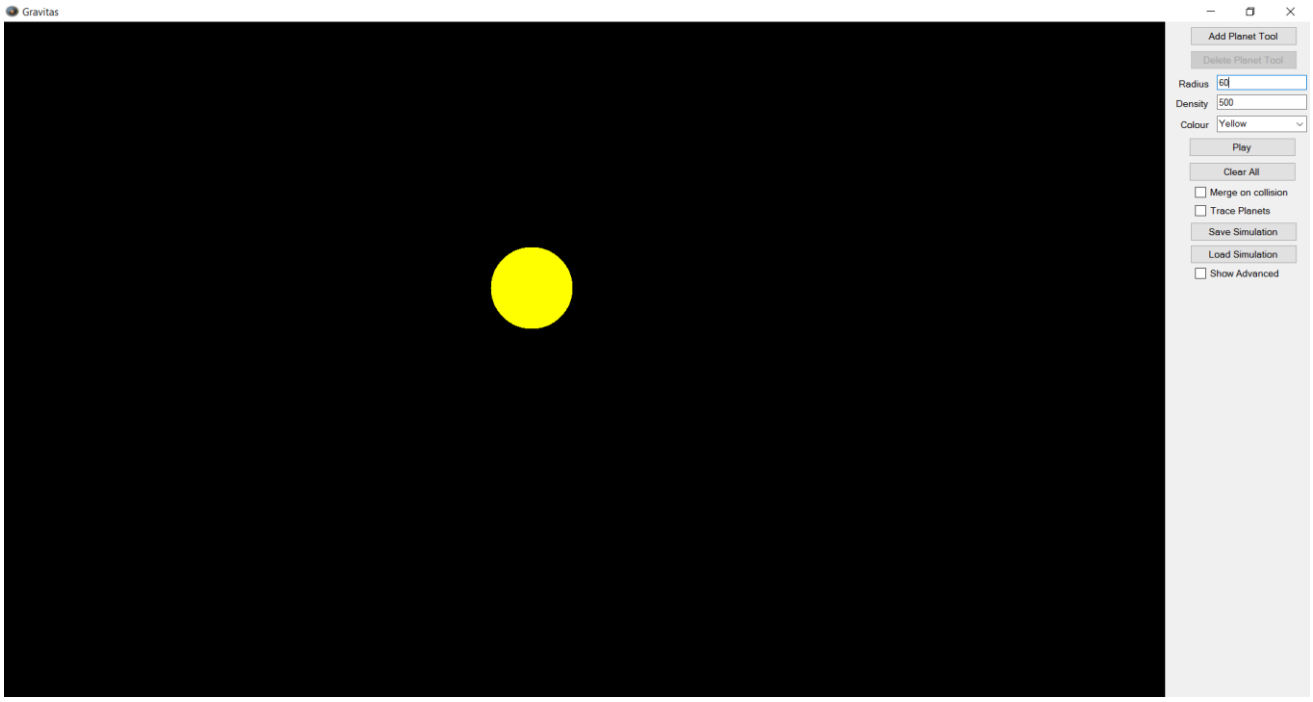
After:

Test 41: delete planet (outside radius)

Before:

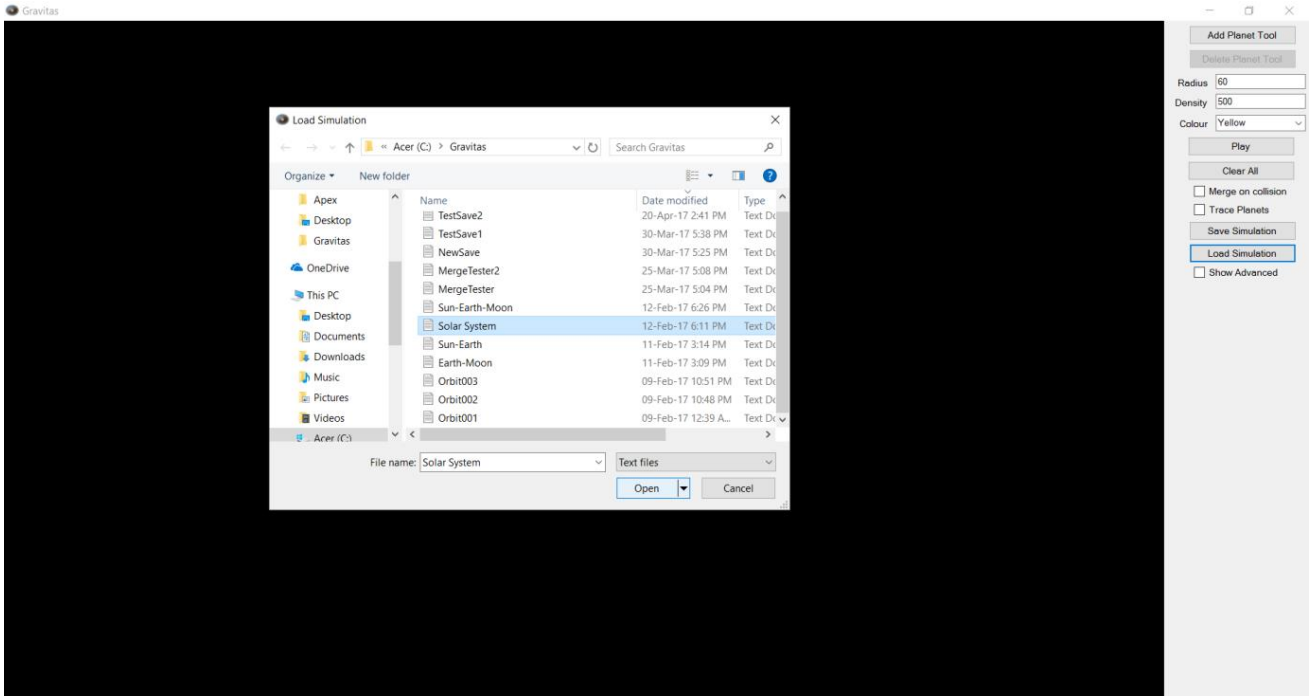


After:

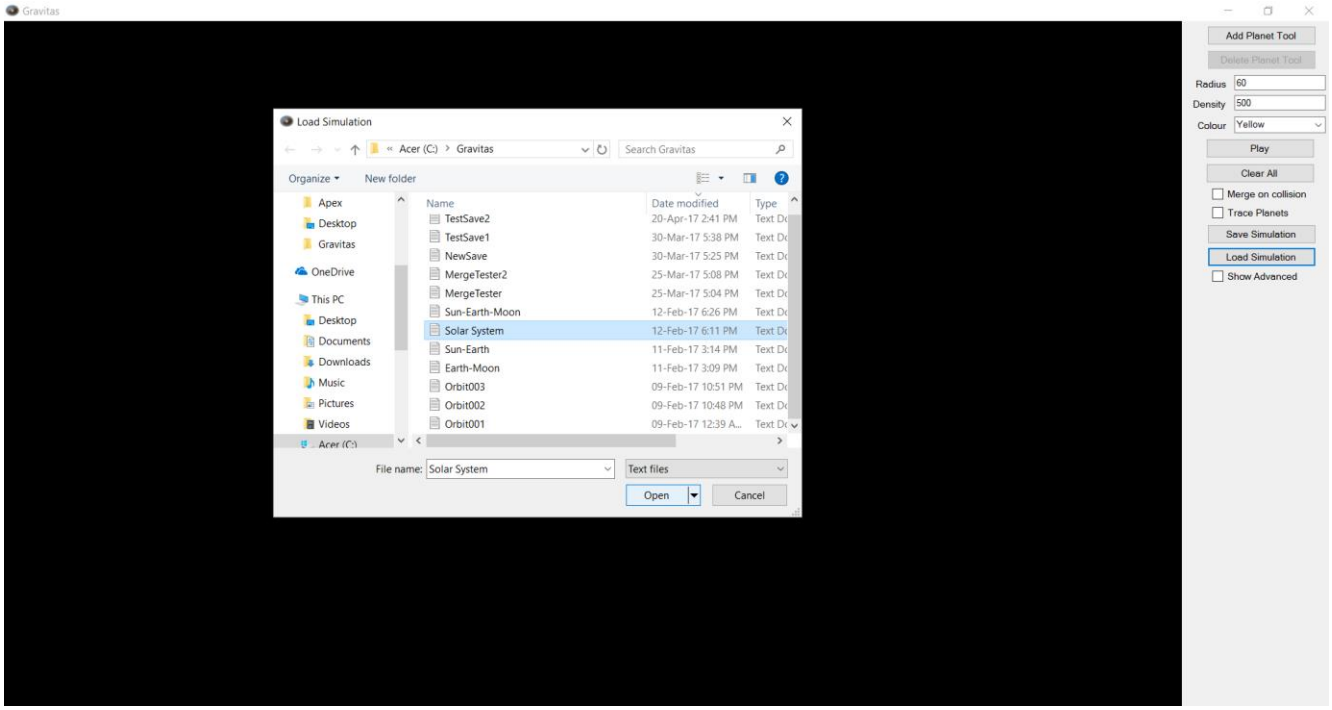


Test 42: load valid file

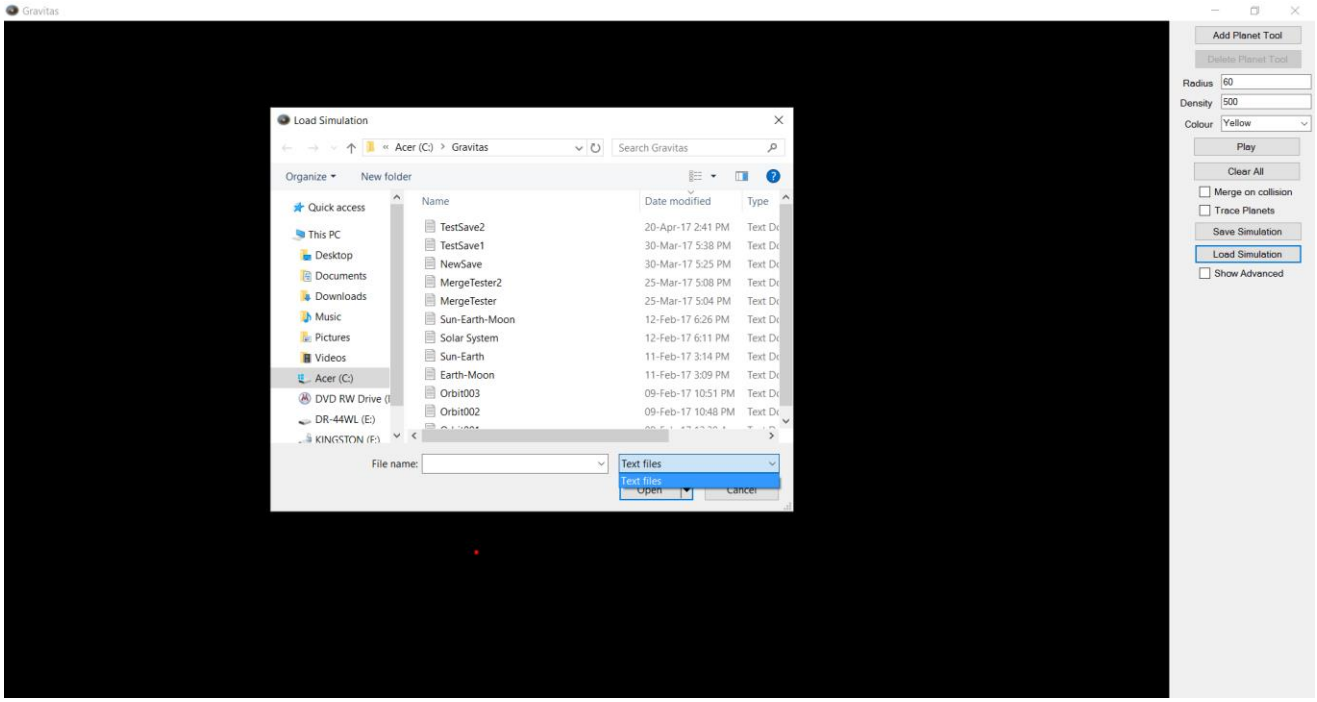
Before:



After:

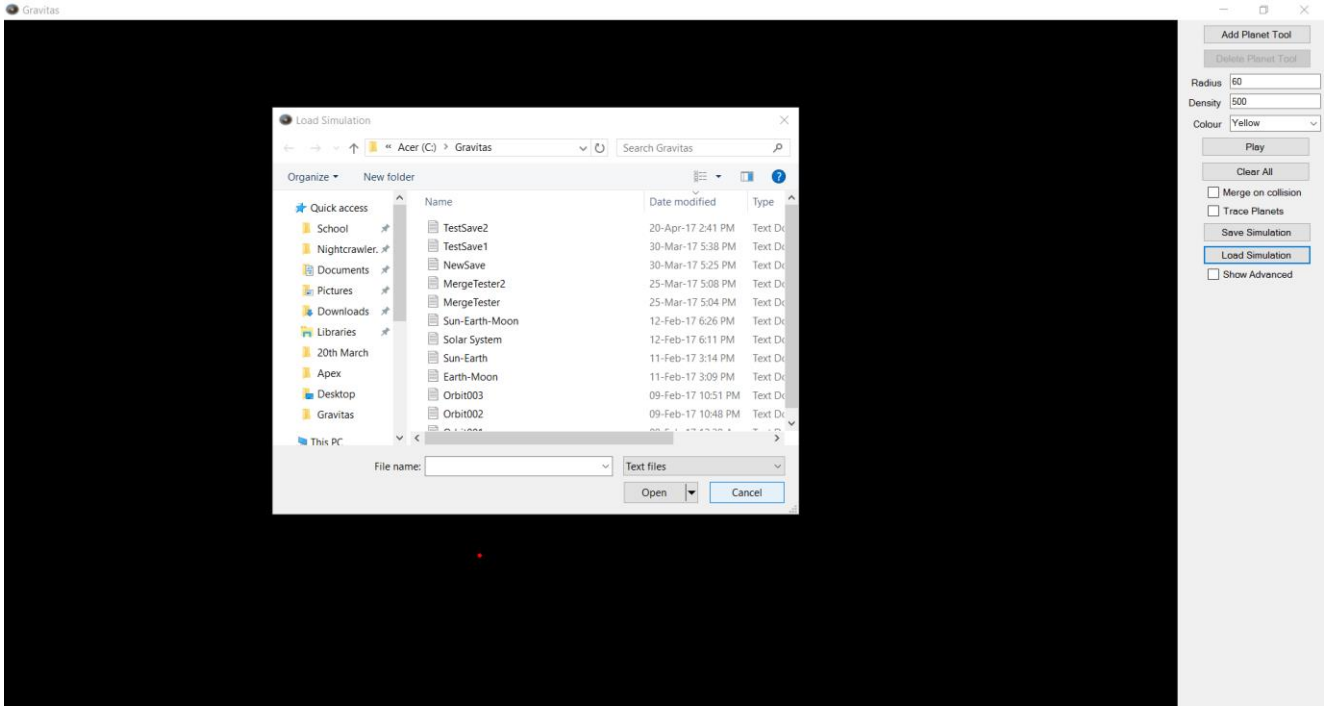


Test 43: Access non-text file

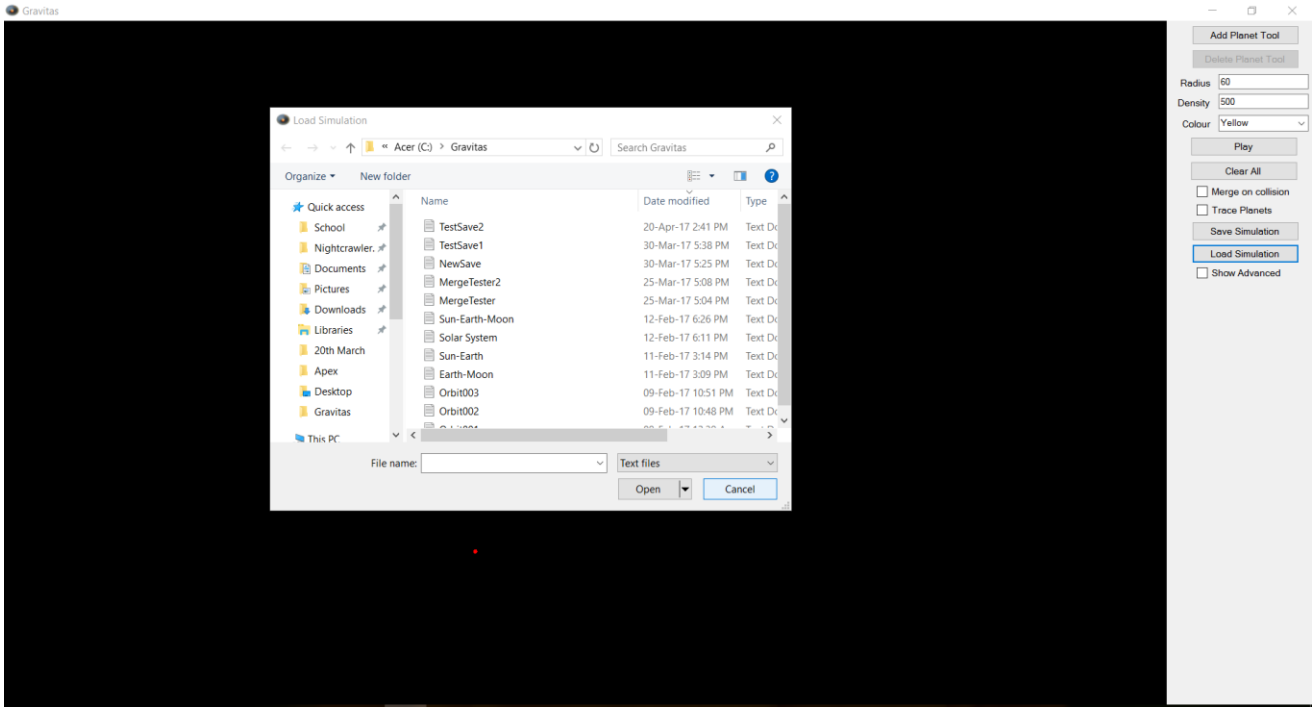


Test 44: click cancel (load)

Before:

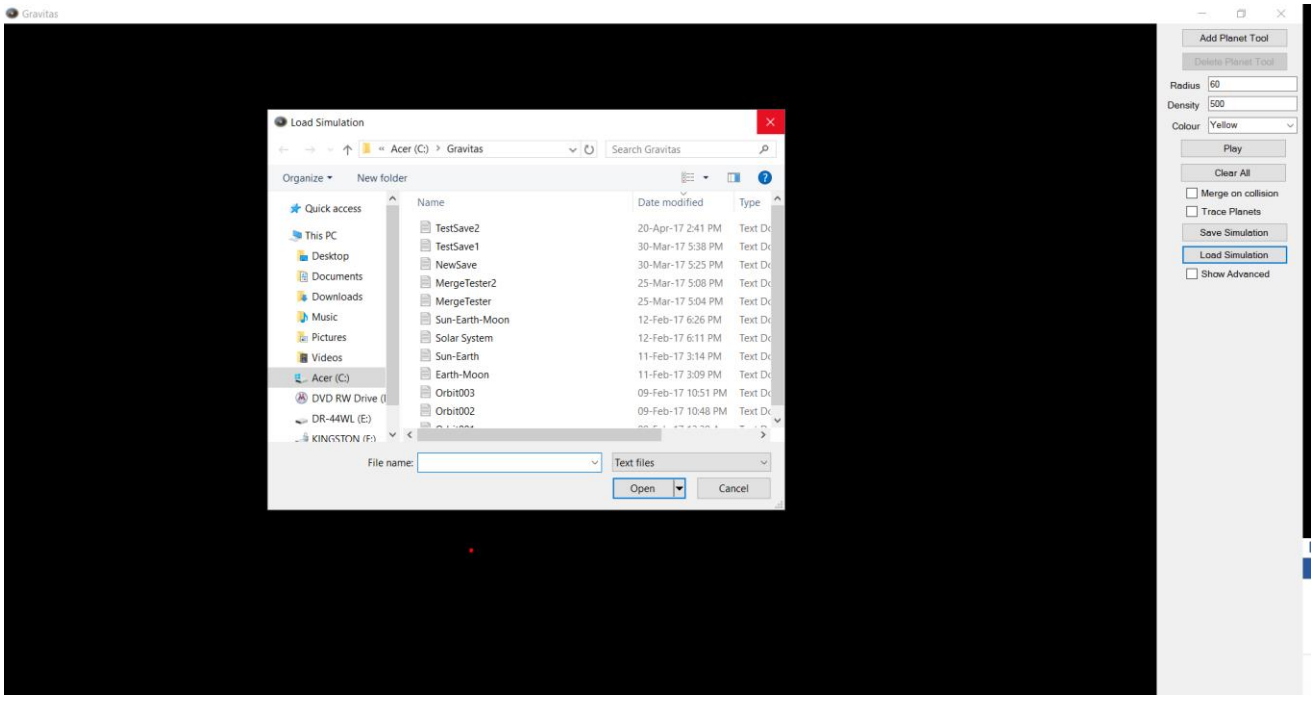


After:

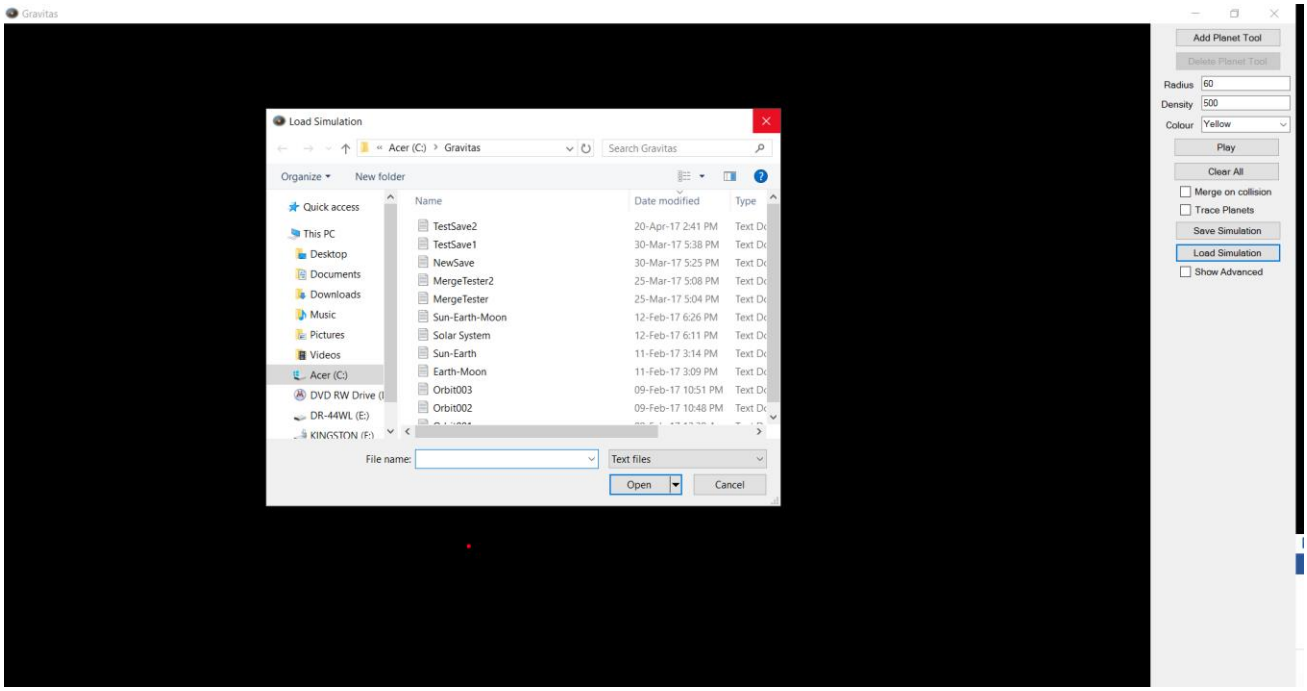


Test 45: click close window (load)

Before:

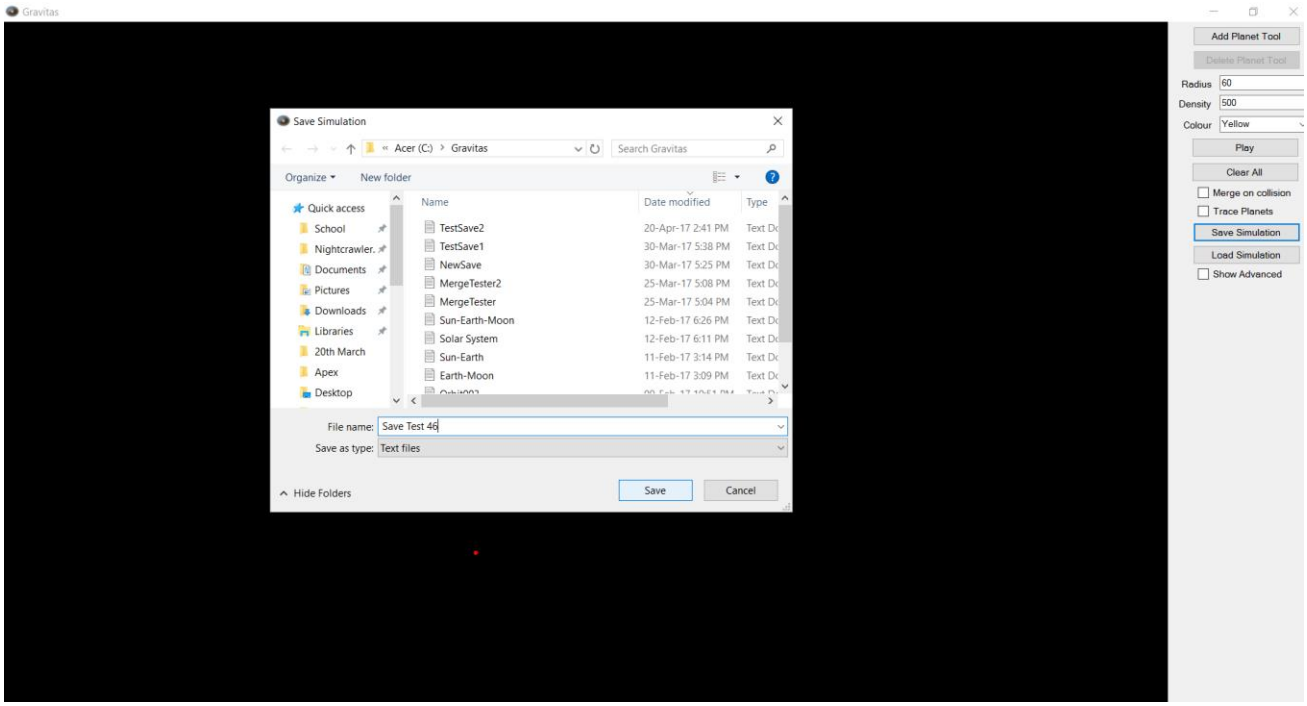


After:

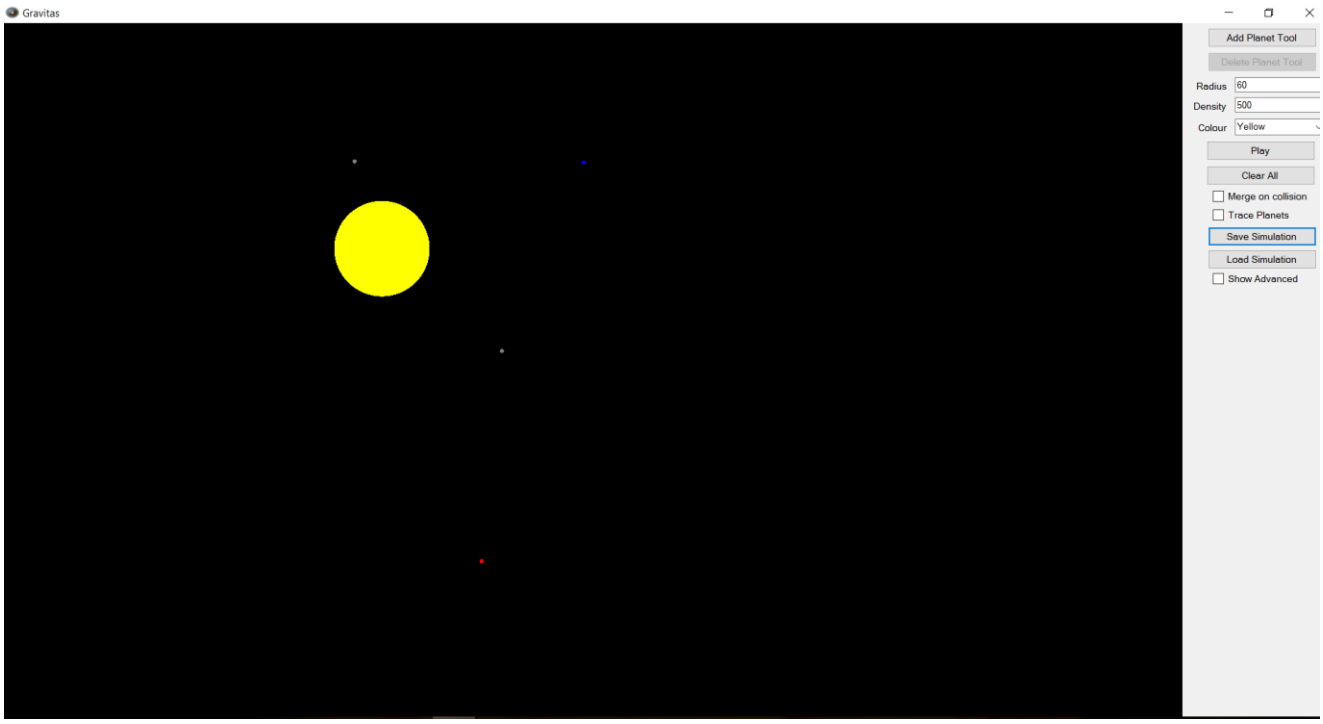


Test 46: save file

Before:

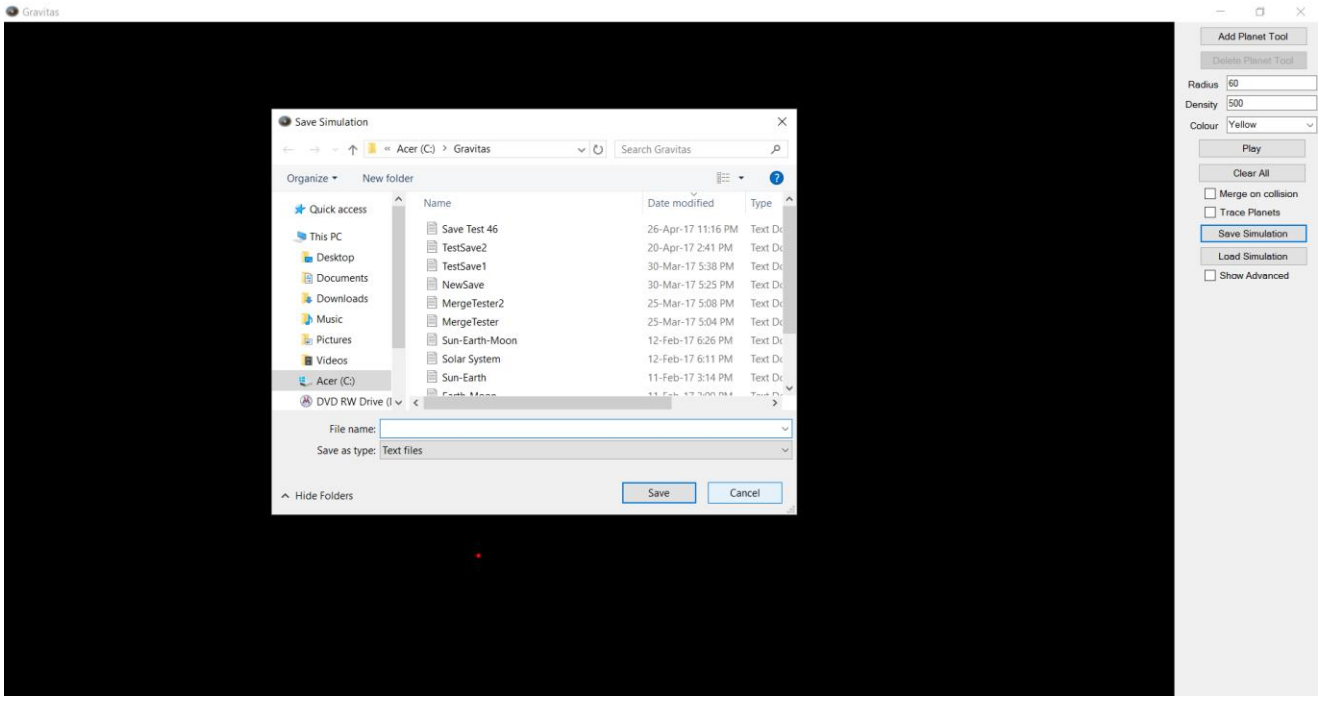


After:

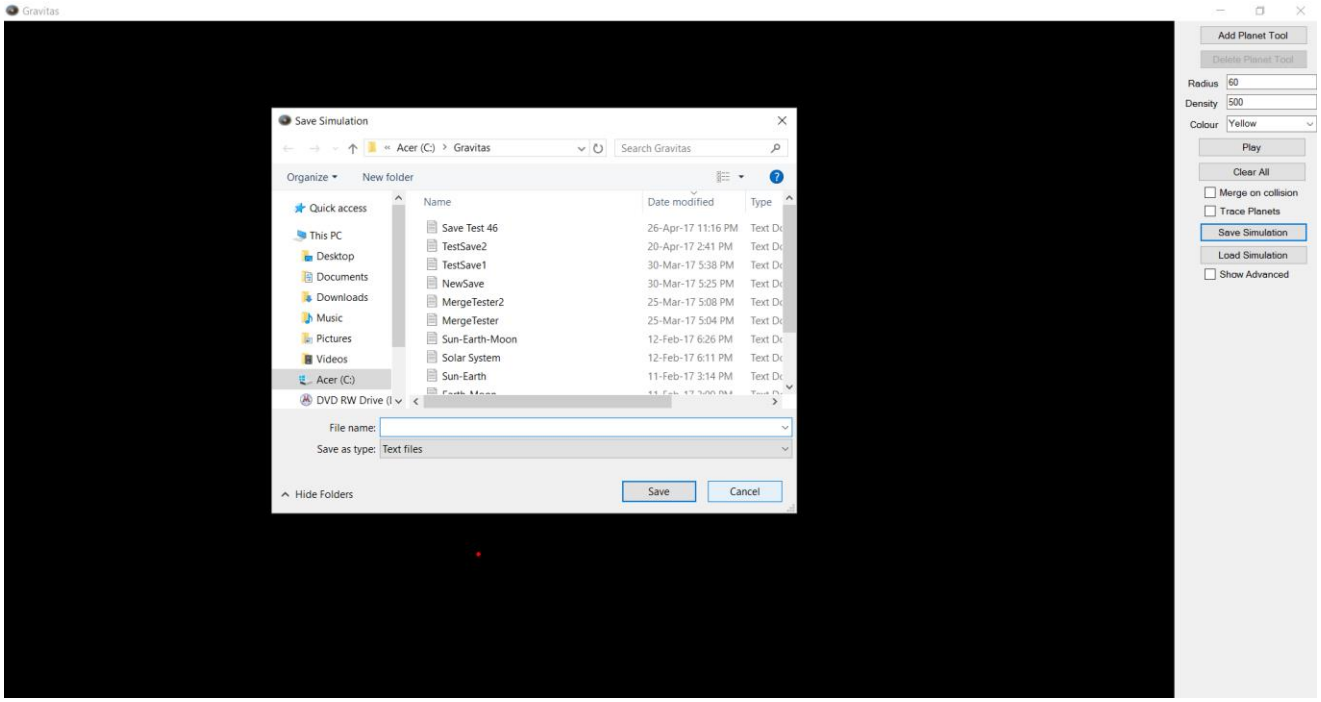


Test 47: click cancel (save)

Before:

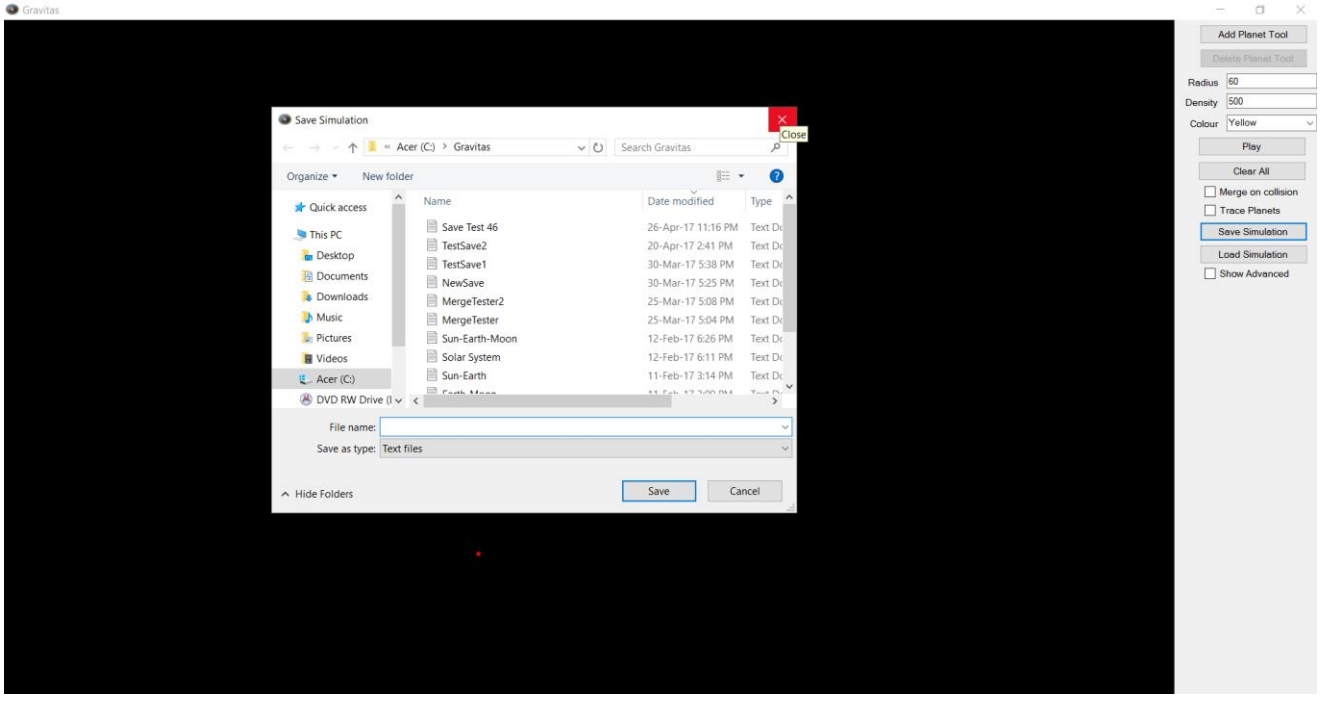


After:

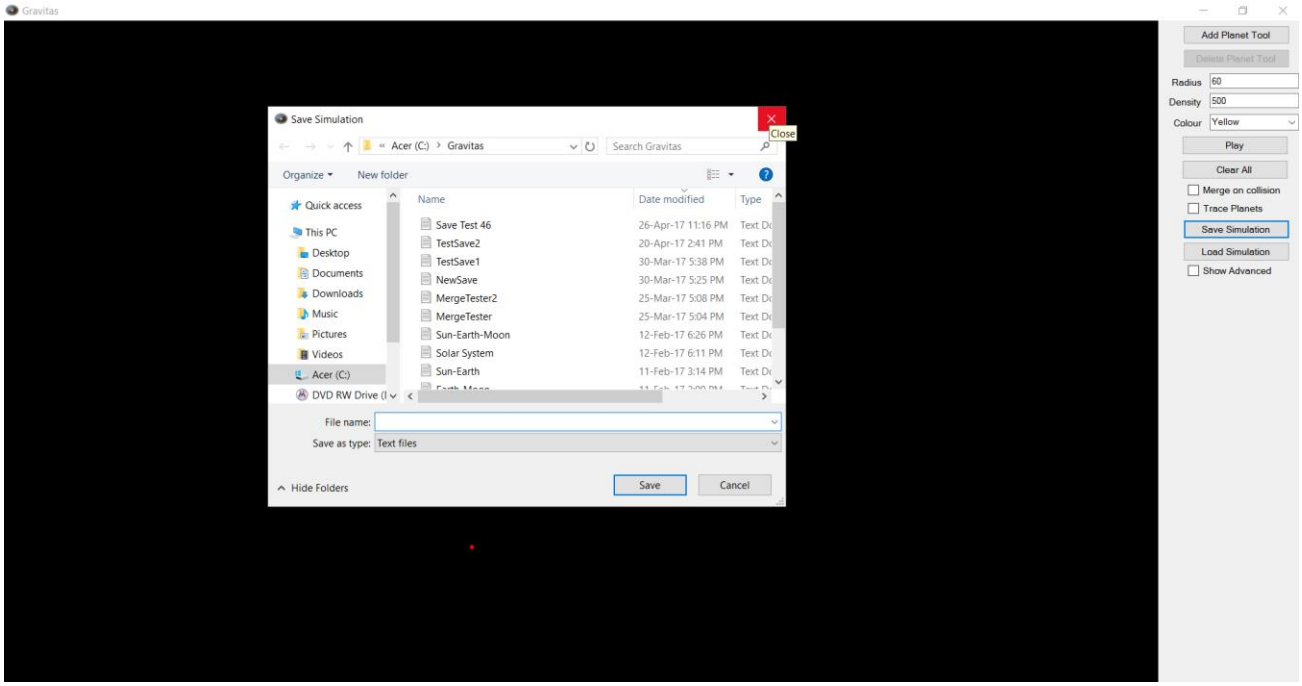


Test 48: click close window (save)

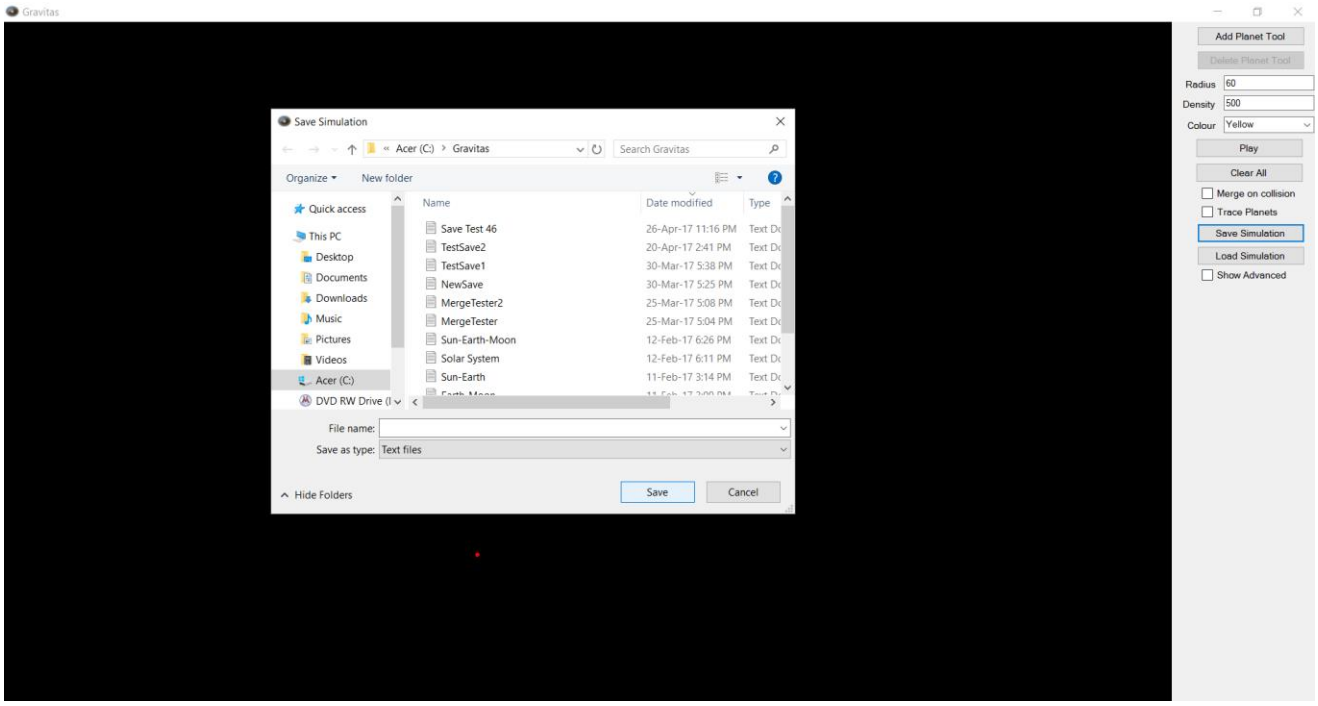
Before:



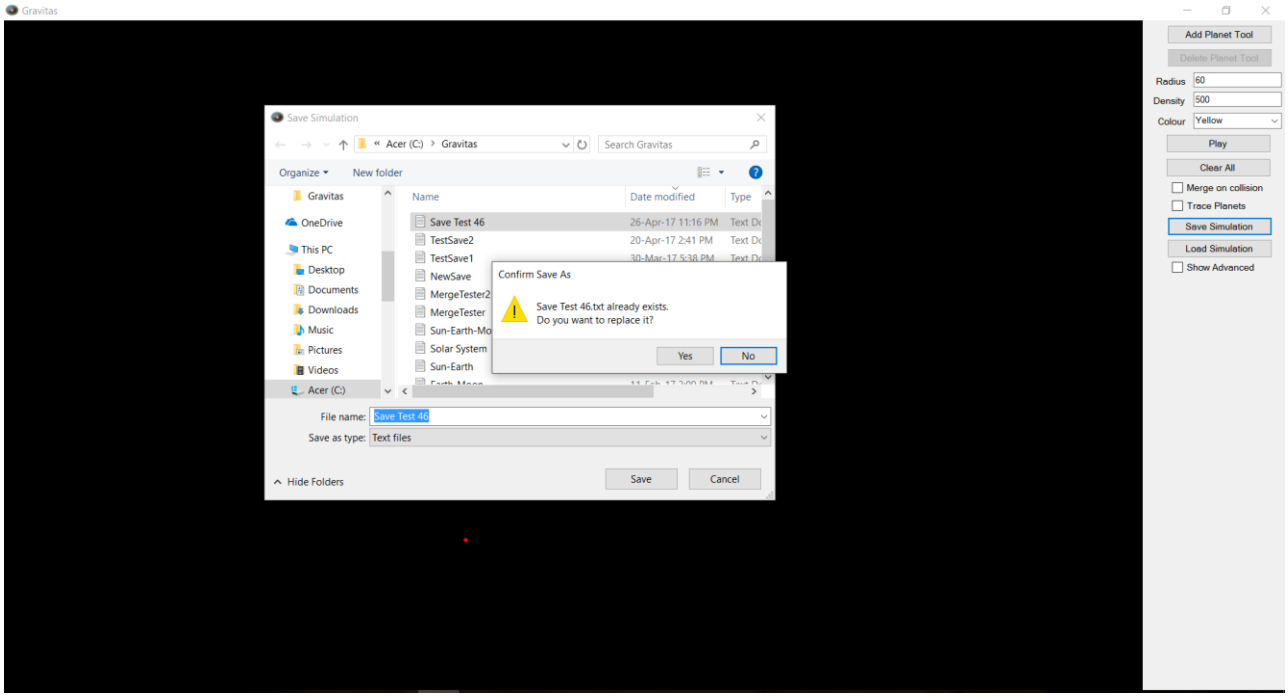
After:



Test 49: click save where filename is null



Test 50: save over existing file



System Testing

In the system testing, I will be identifying whether the simulations work as intended and accurately demonstrates the movement of celestial bodies.

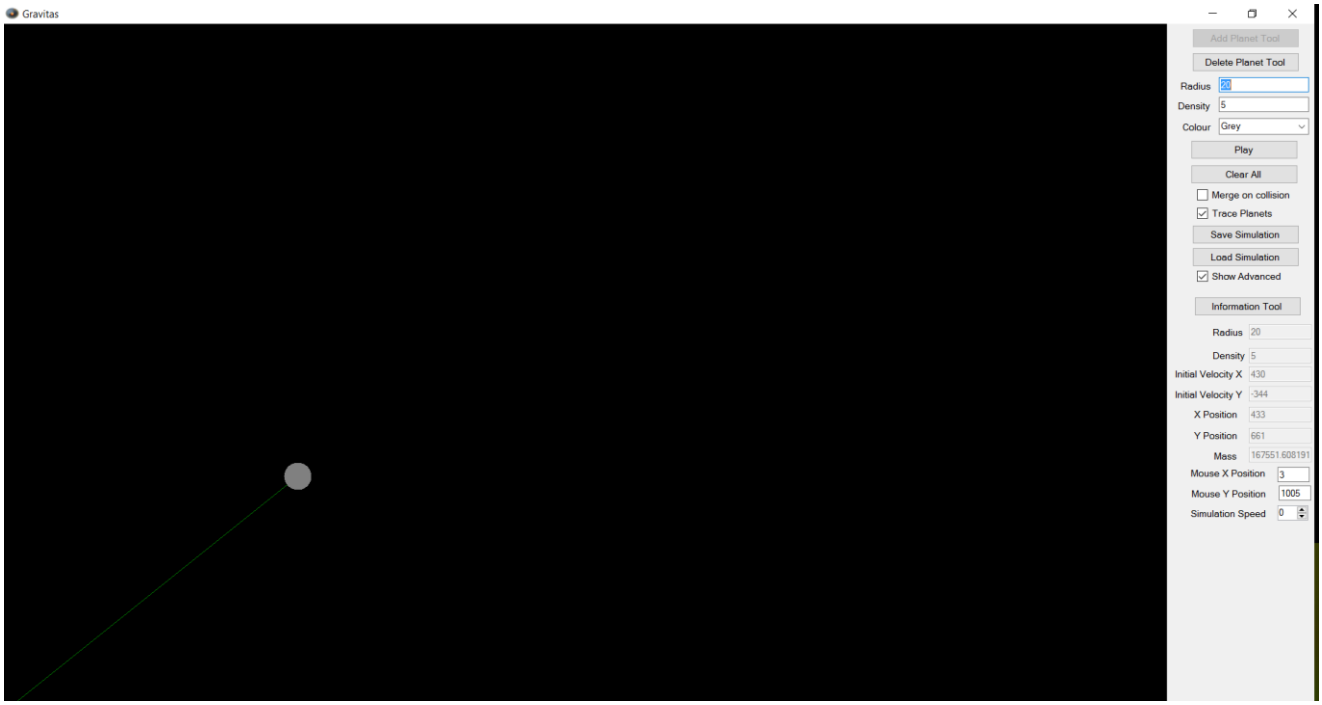
White Box Testing

Test 1: Initial velocity

Test Description: I will place a planet on the screen with an initial velocity that should send it to the top right of the screen. To help demonstrate the movement of the planet in a still image I will select the trace planet option

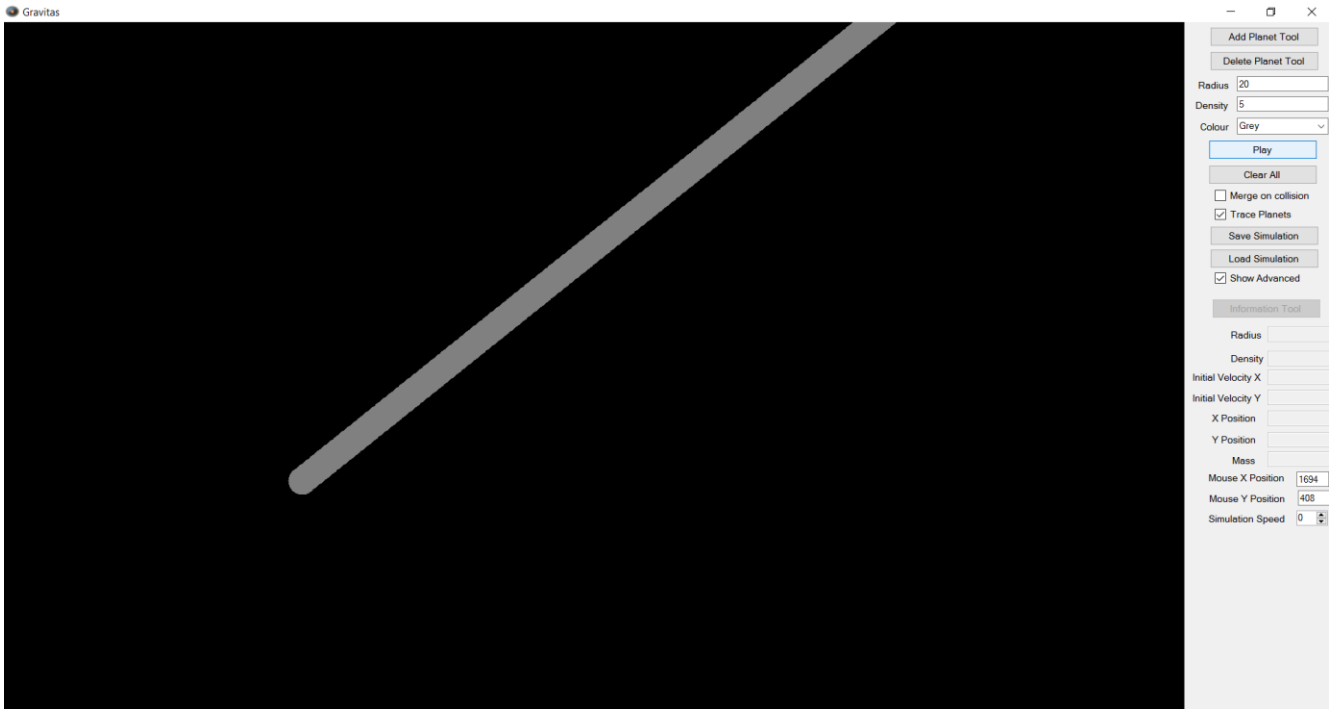
Expected Outcome: I expect the object to move in the opposite direction as my line

Screenshot Before[1]*:*



Screenshot After:

[1] For some reason the screenshot failed to pick up the green initial velocity line, in this screen shot there should be a green line from the centre of the grey planet to the bottom left corner of the screen



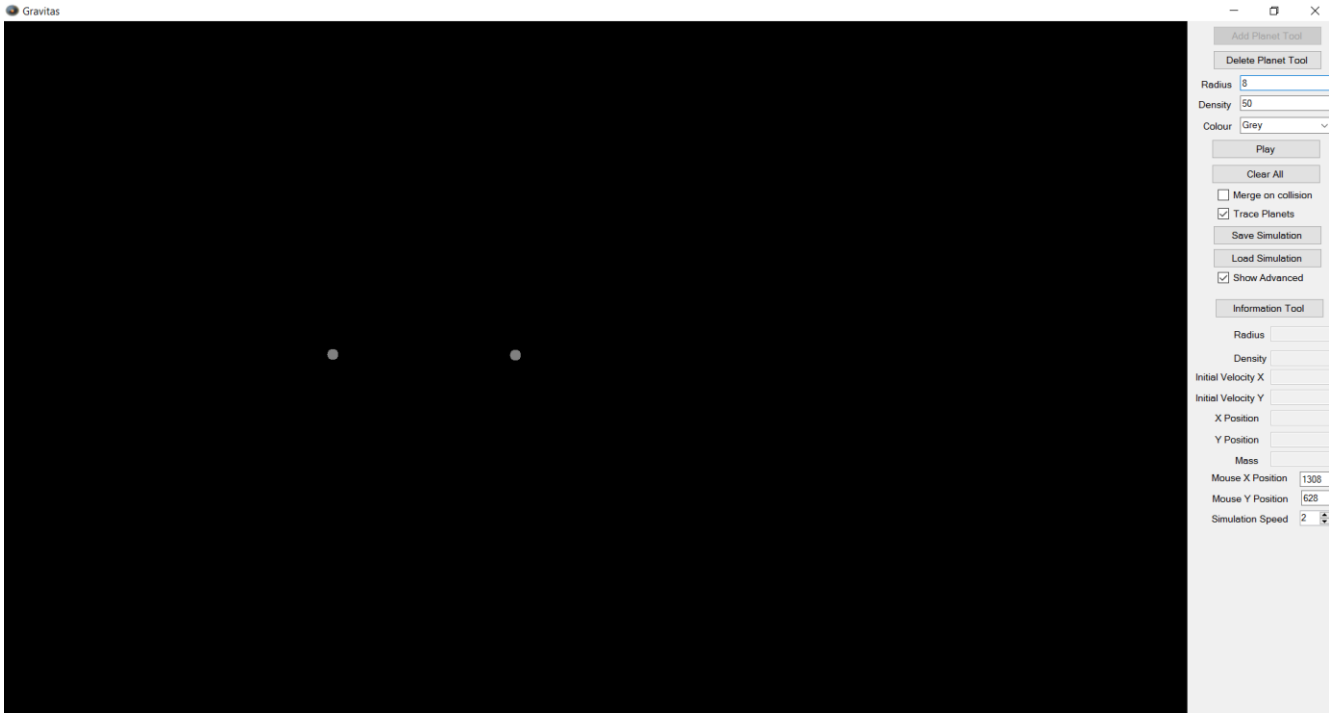
Test 2: Gravitational Attraction

Part a: 2 Planets

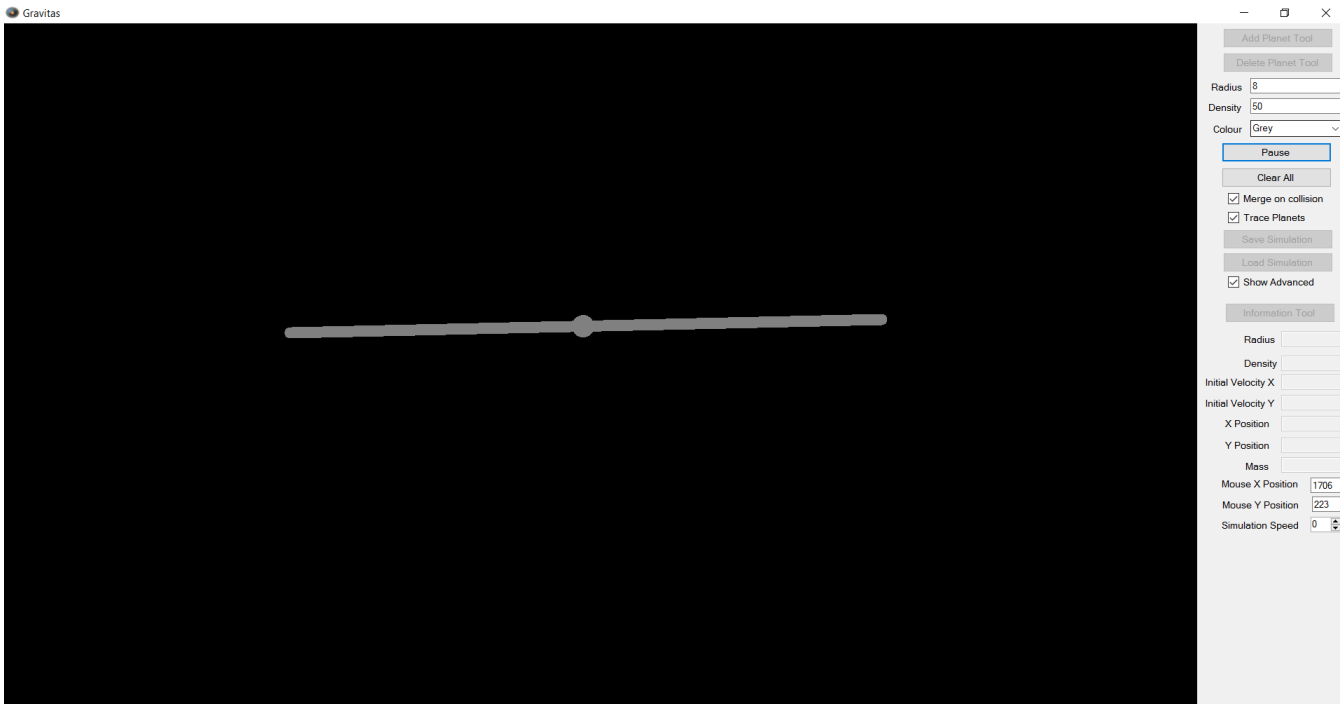
Test Description: I will place two objects of high mass but with no initial velocity and see if they are attracted as they would be in space by gravity. The radius for each will be 30 and the density 500. Again, I will put trace on to clearly show their movements in a still image

Expected Outcome: The planets should accelerate toward each other

Screenshot Before:



Screenshot After:

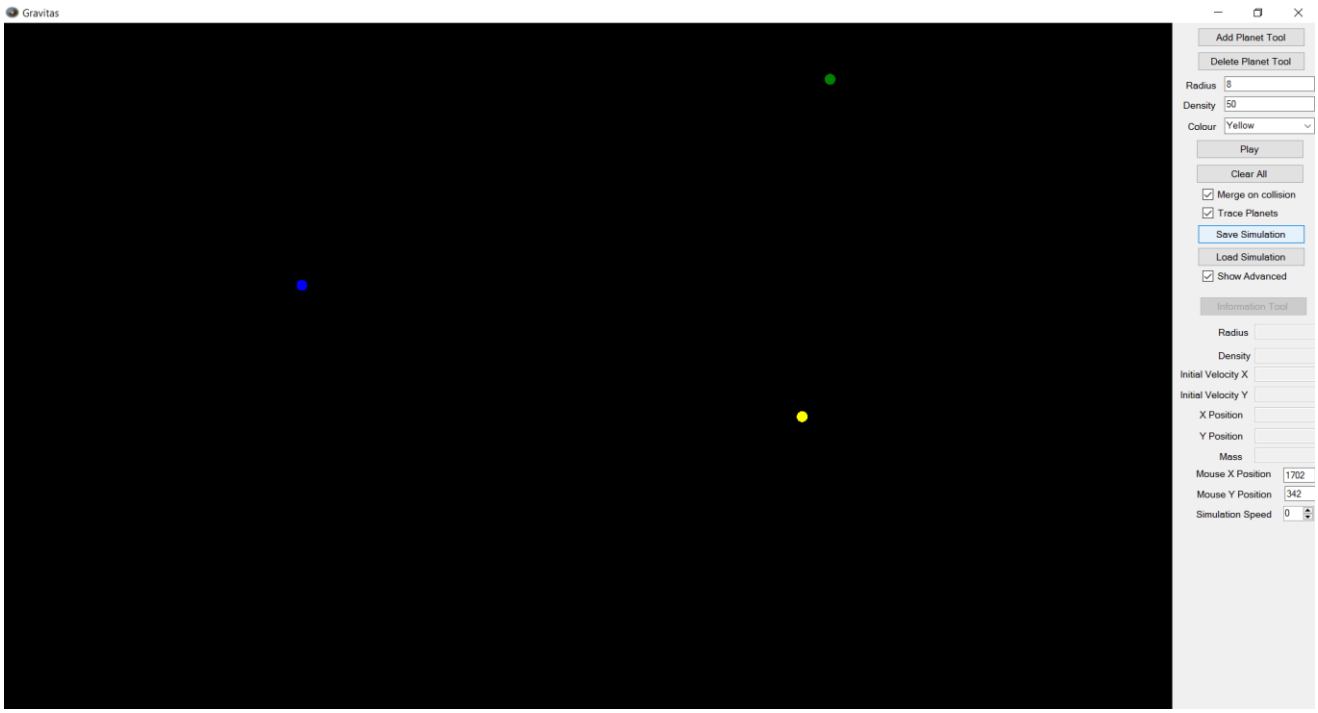


Part b: 3 Planets

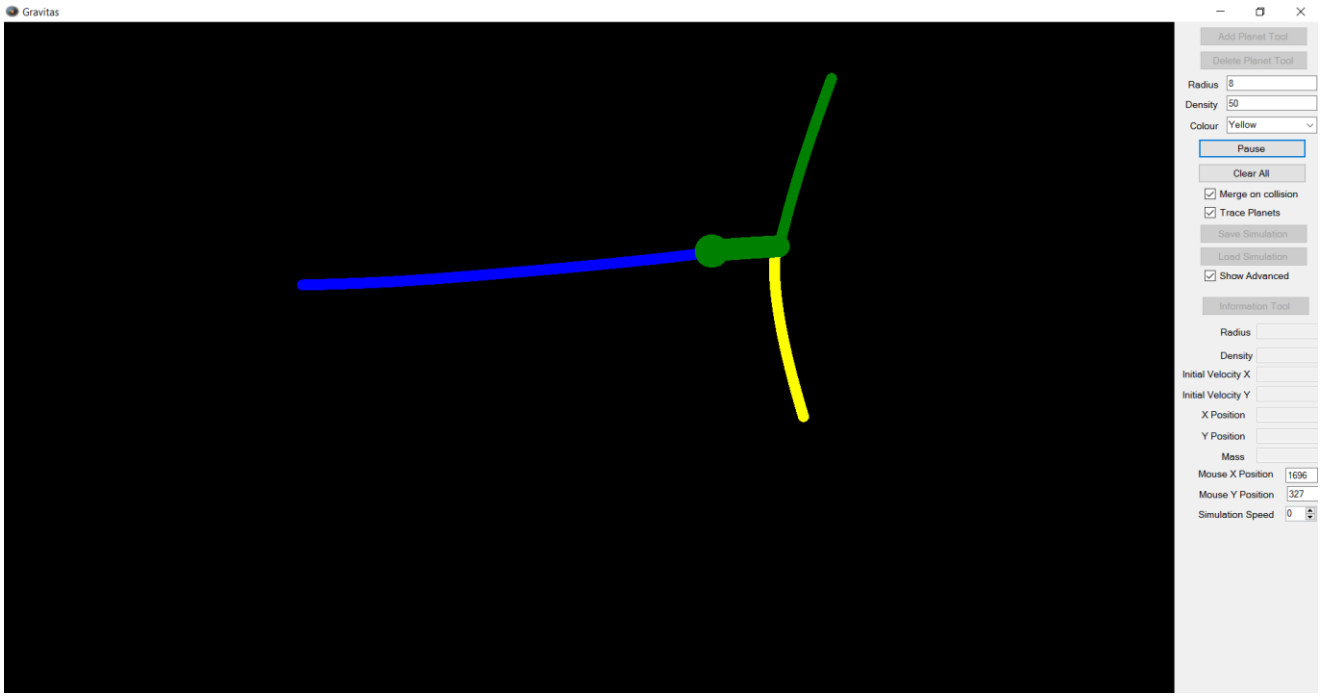
Test Description: This will be the same as part a, however I will now use 3 planets instead of 2

Expected Outcome: The planets should accelerate toward each other

Screenshot Before:



Screenshot After:

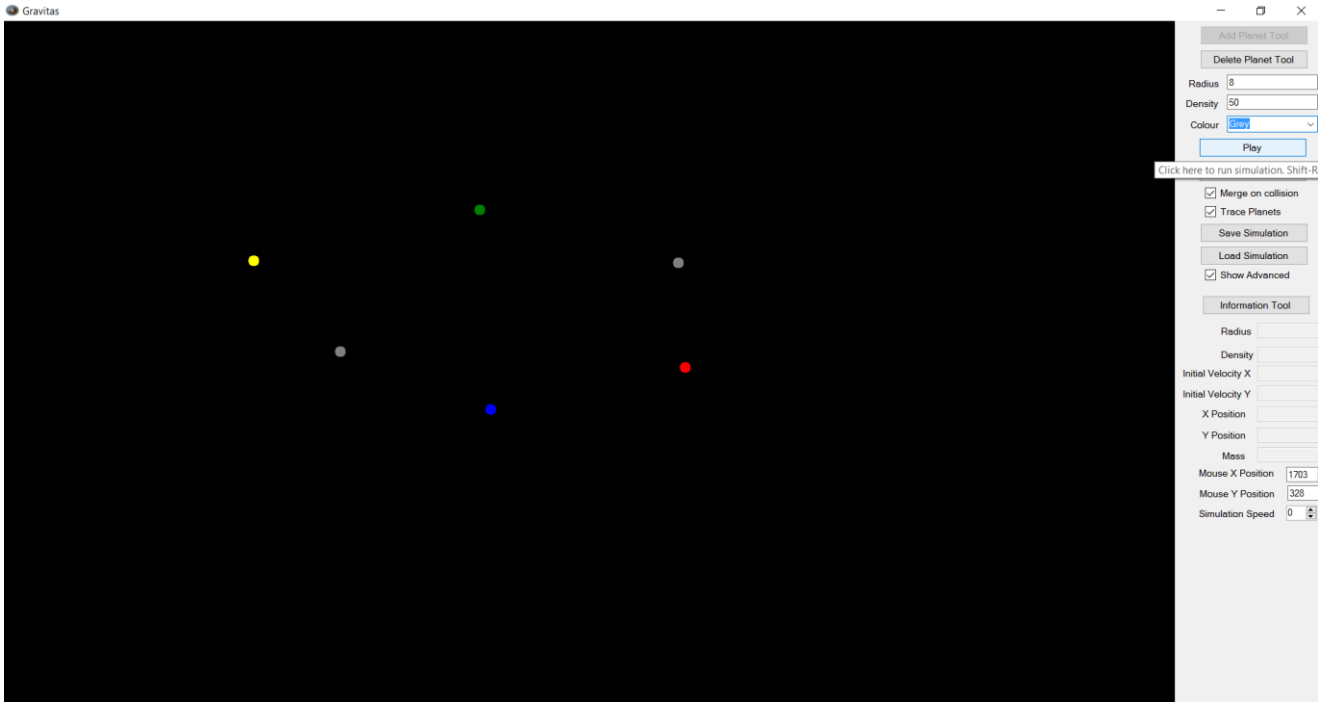


Part c: 6 Planets

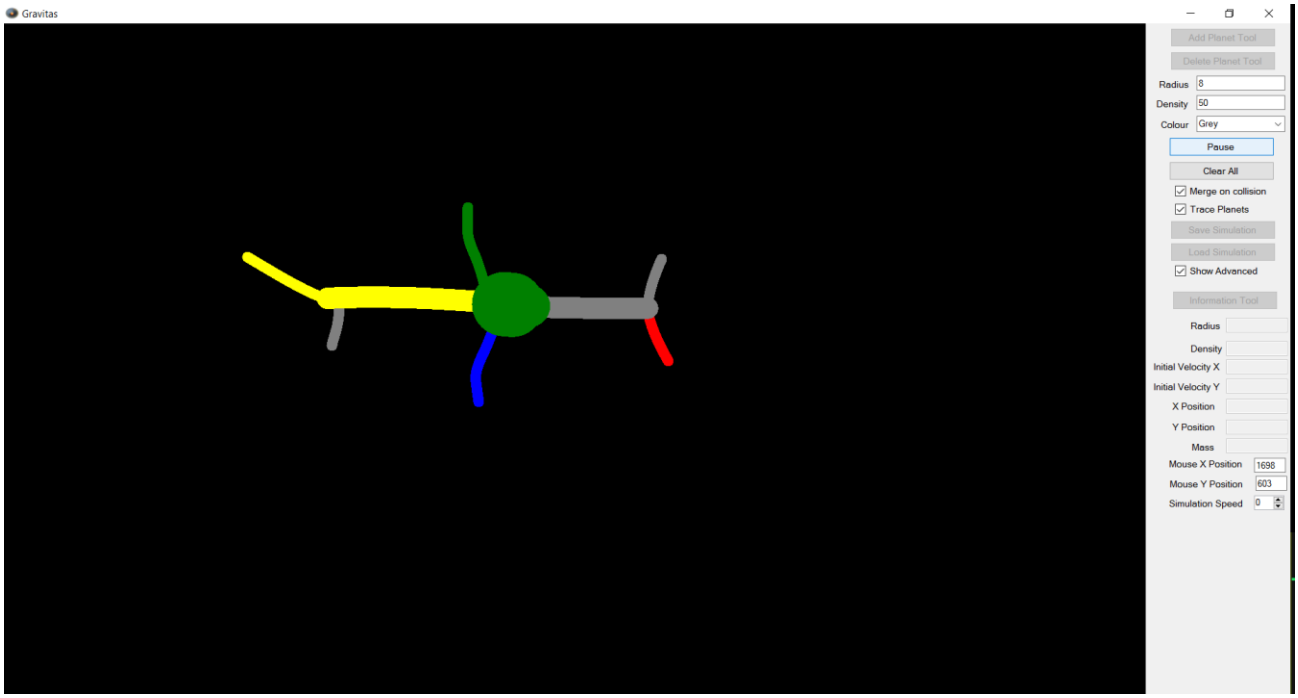
Test Description: This will also be the same as parts a and b but with 6 planets

Expected Outcome: The planets should accelerate toward each other

Screenshot Before:



Screenshot After:



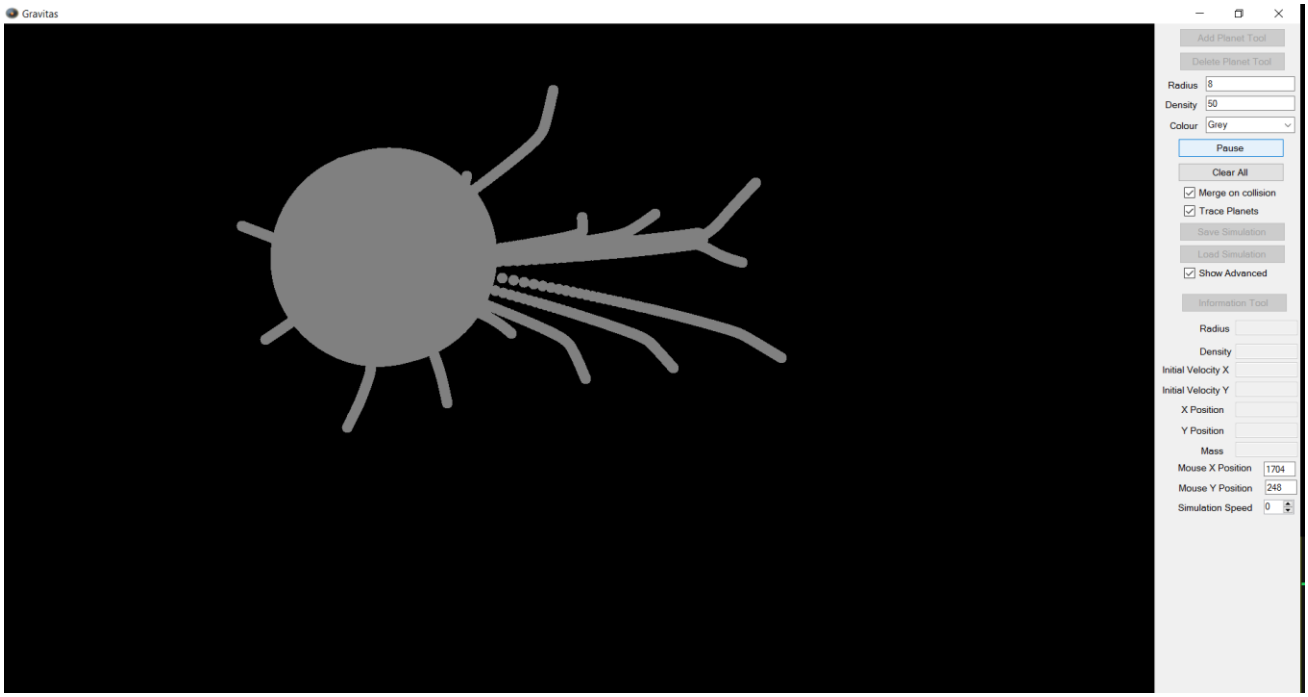
Part d: 20 planets

Test Description: This final test will examine the programs ability to take the maximum load of 20 unique planets at a time

Expected Outcome: The planets should accelerate toward each other

Screenshot Before:

Screenshot After:

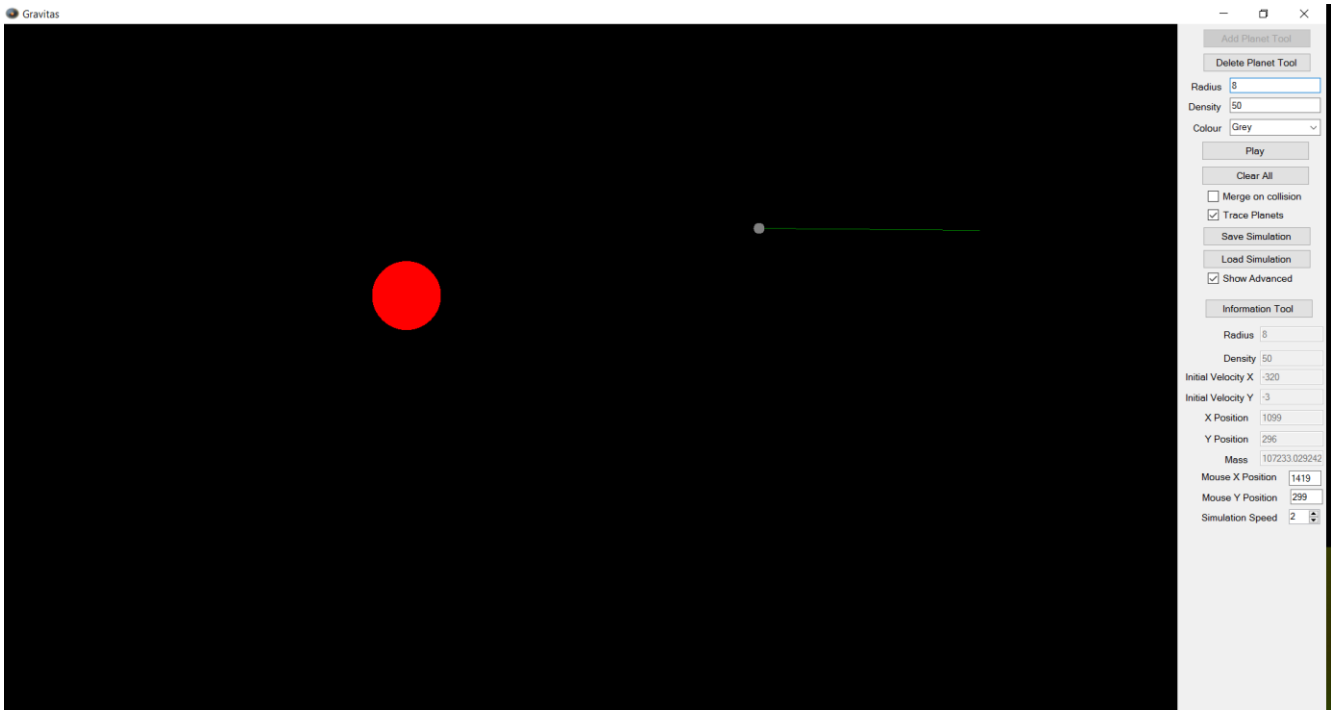


Test 3: Gravitational deflection

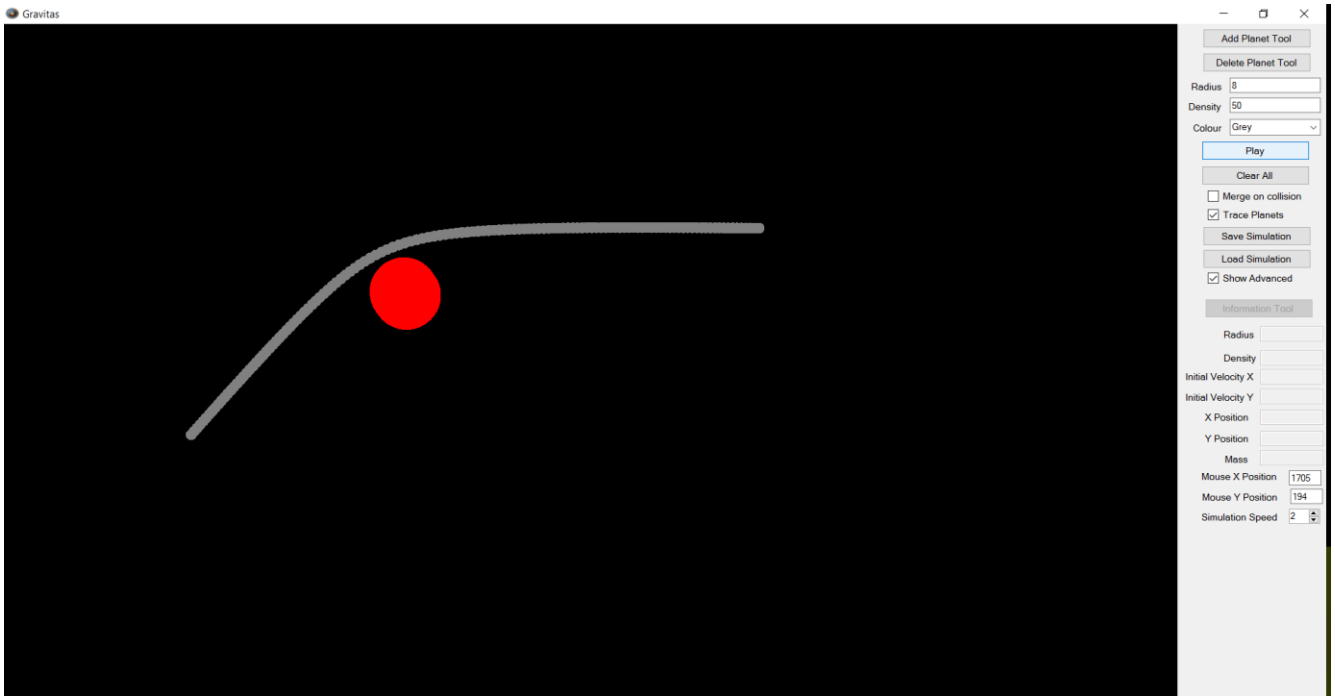
Test Description: I will place an object of high mass but with no initial velocity on the screen and pass another object with an initial velocity close by it

Expected Outcome: The moving object should be deflected from its course and pulled around the back side of the stationary planet

Screenshot Before:



Screenshot After:



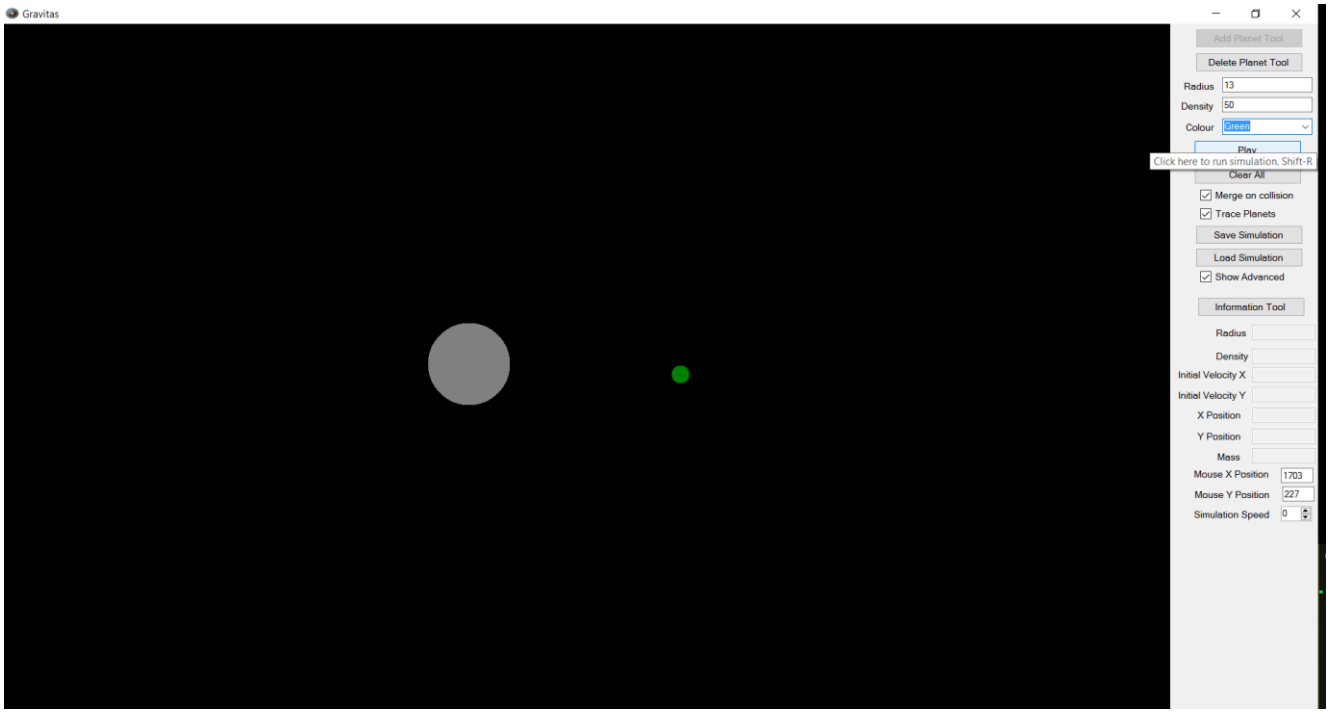
Test 4: Orbit

Part a: 2 Planets

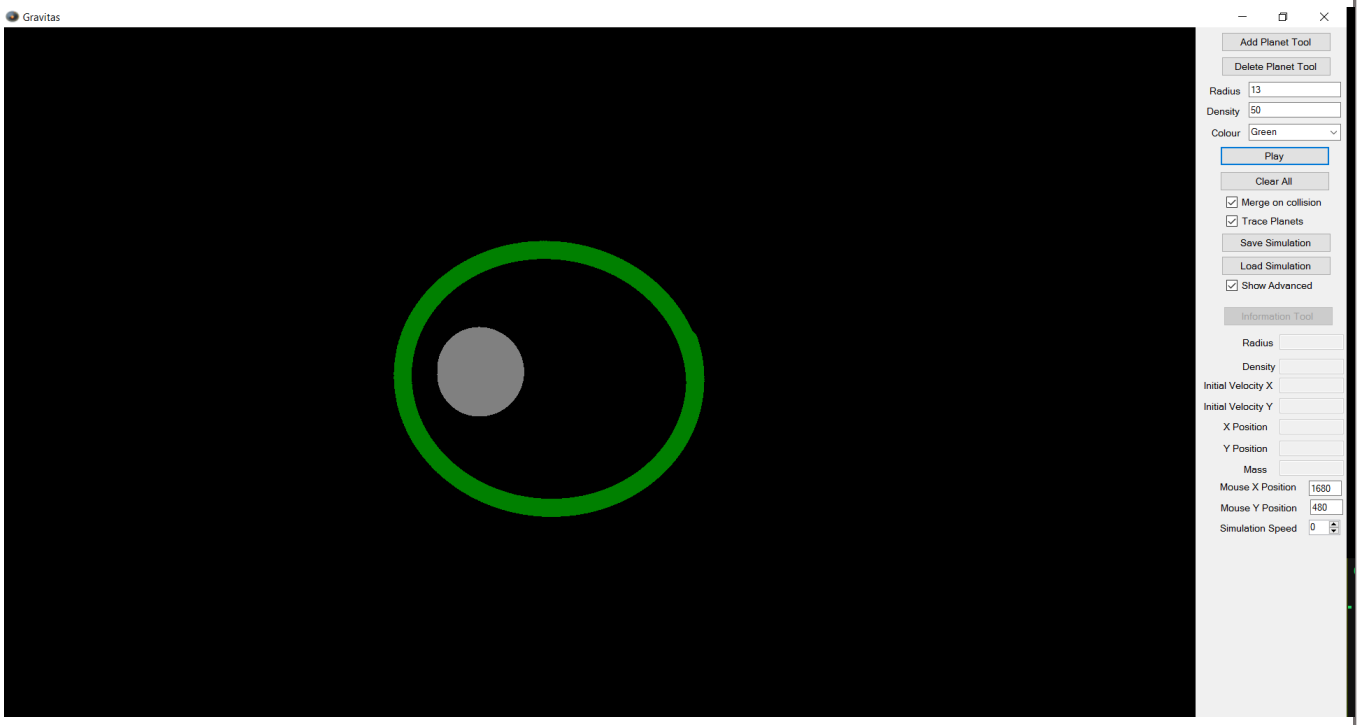
Test Description: I will add a stationary planet in roughly the centre of the screen and then a smaller planet to its right, this smaller planet will have an initial velocity that will send the planet towards the top of the screen

Expected Outcome: The resultant force between the smaller planets initial velocity and the pull of the larger planet should cause it to orbit the planet and return to its starting position.

Screenshot Before:



Screenshot After:

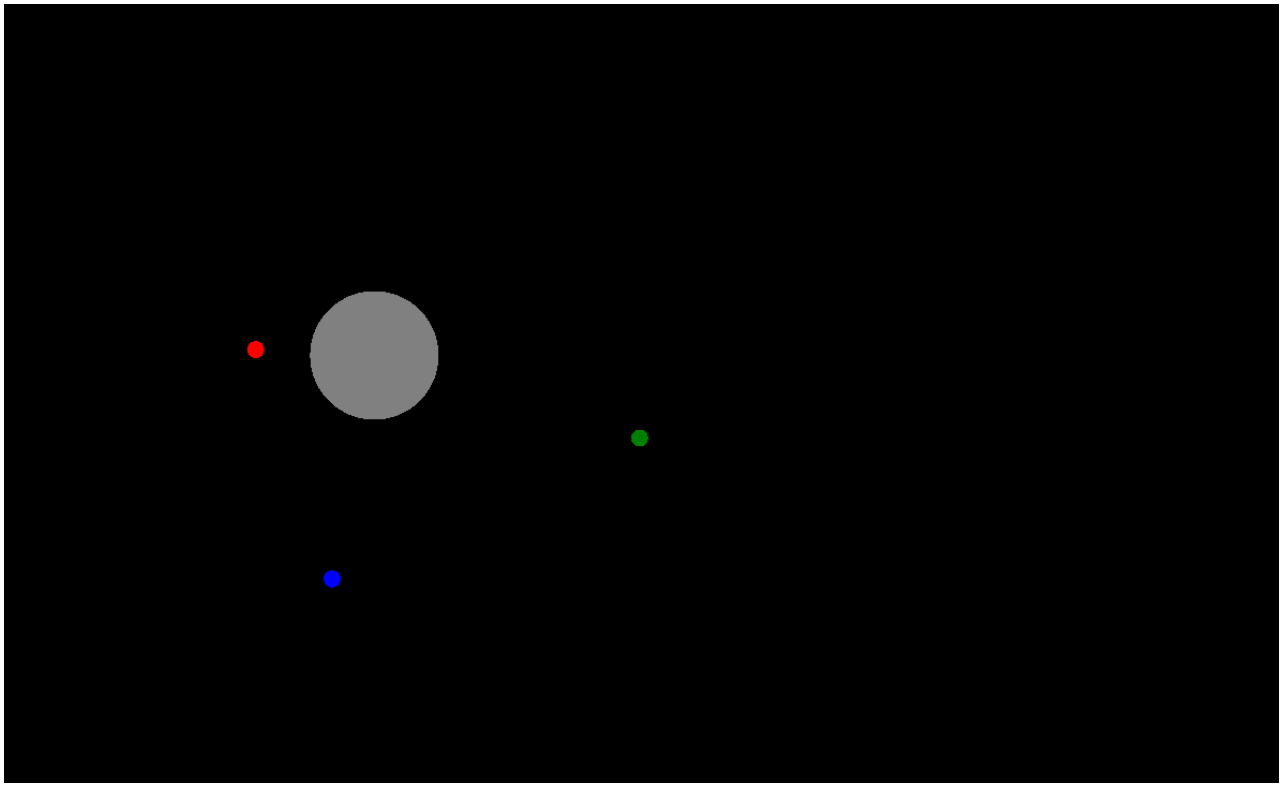


Part b: 3 Planets

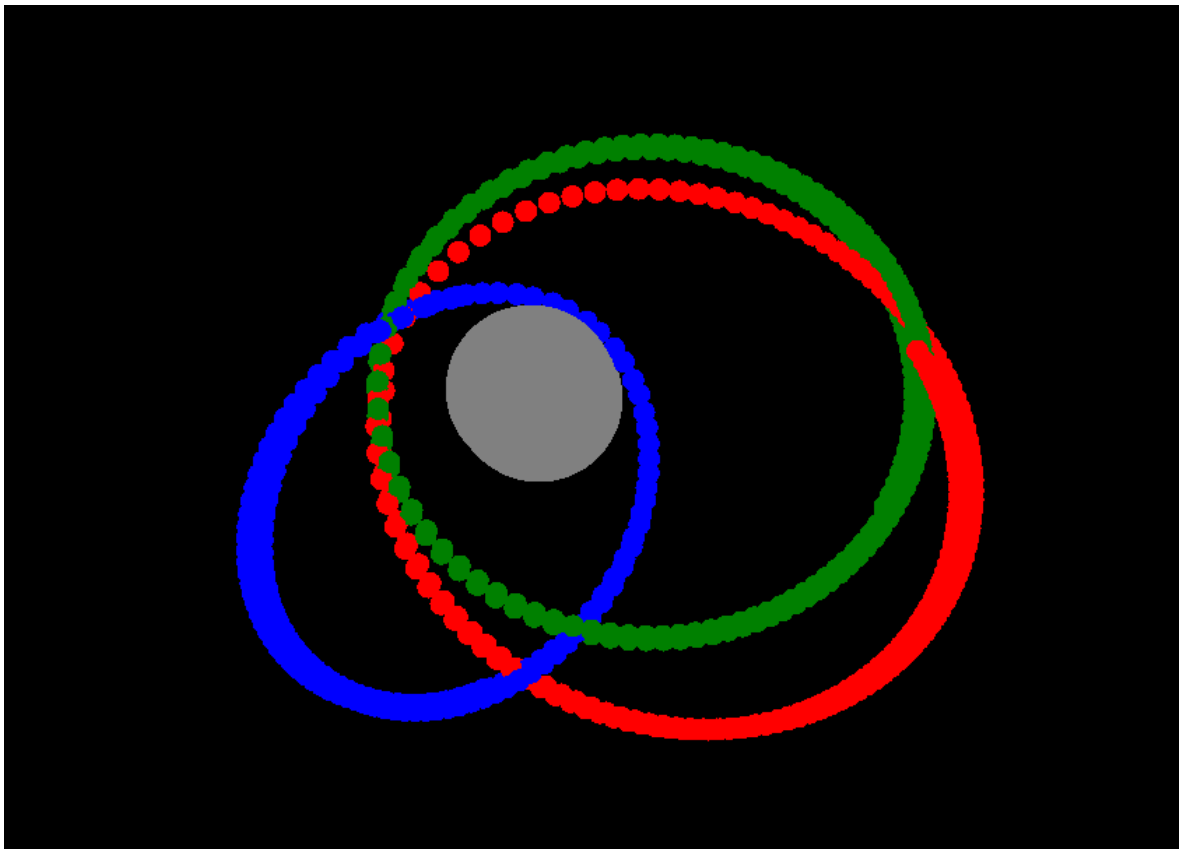
Test Description: This will be the same as part a, however I will now use 3 planets instead of 2

Expected Outcome: The smaller planets should orbit the largest object

Screenshot Before:



Screenshot After:

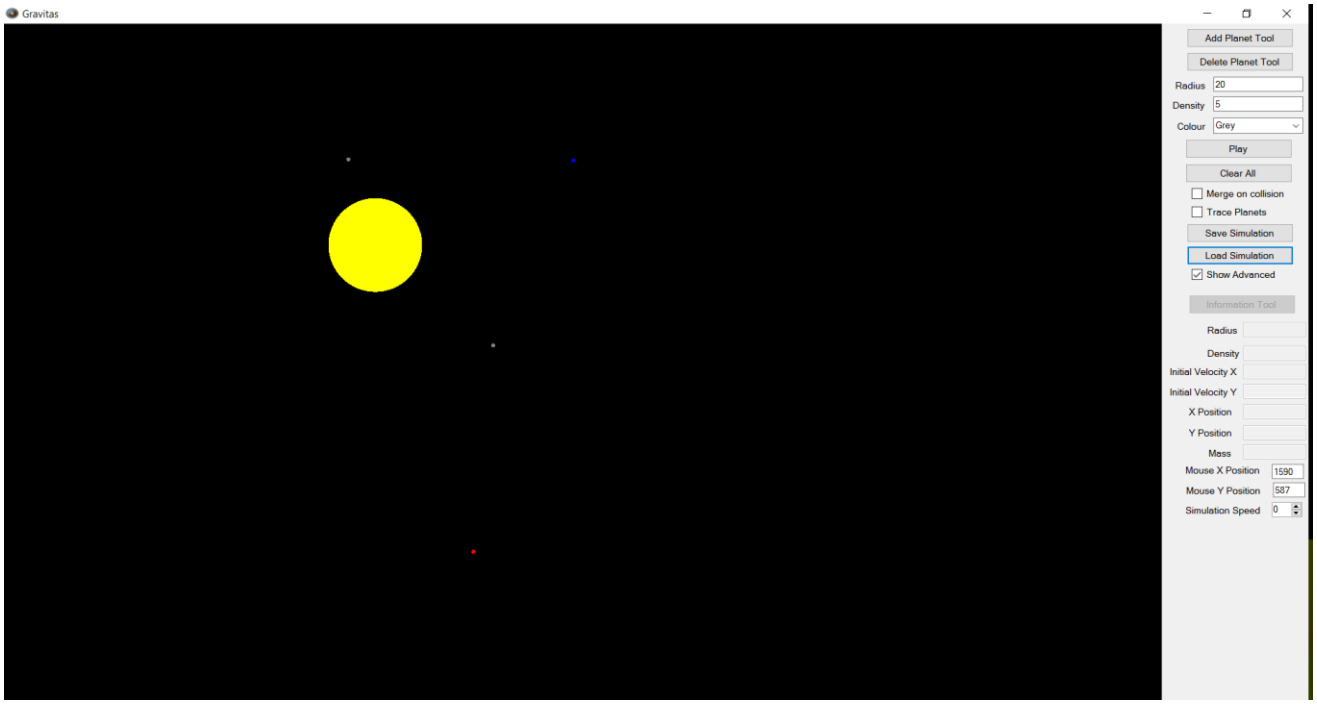


Part c: 4 Planets

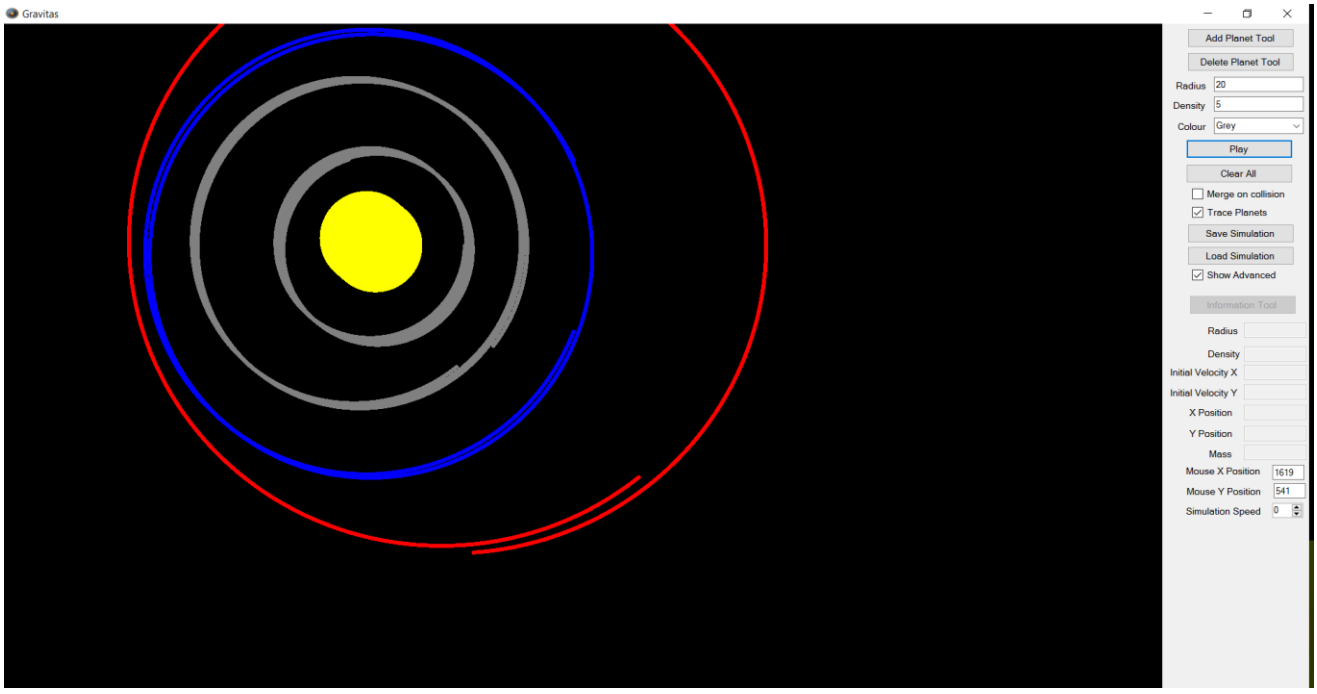
Test Description: This will also be the same as parts a and b but with 6 planets

Expected Outcome: The smaller planets should orbit the largest object

Screenshot Before:



Screenshot After:

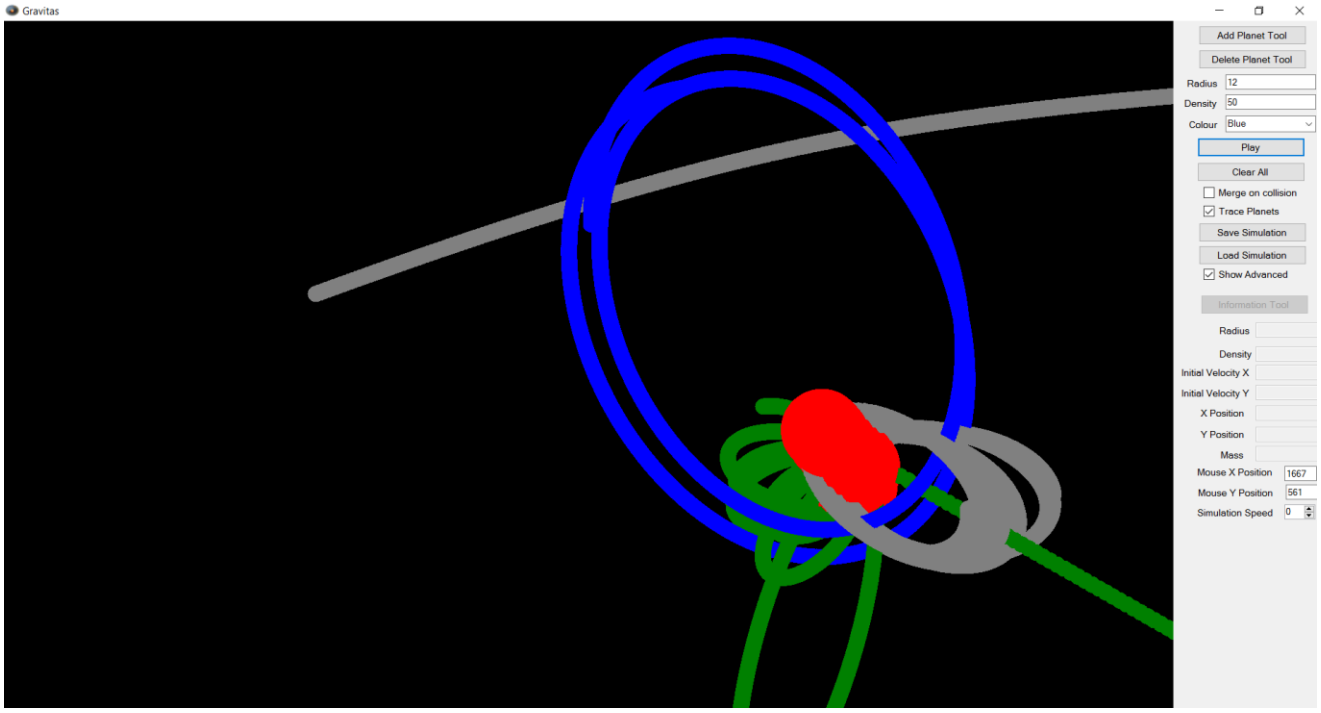


Test 5: Save and Load

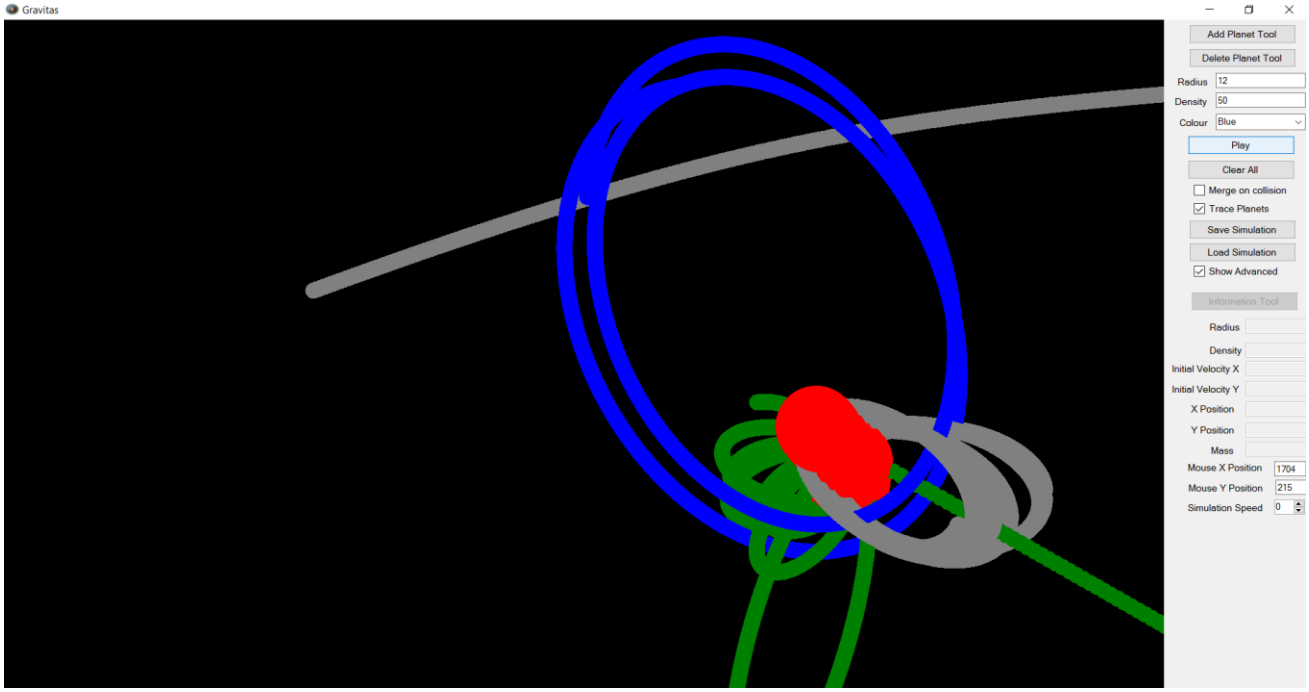
Test Description: To ensure that the save and load functions work correctly I will create a simulation, save it and then run it. After that, I will clear the canvas and reload the simulation, if it runs the same as the first time then I can be sure that the program is properly recording all of the data and loading it in properly. In each case, I have run the simulations for five seconds at speed 0.

Expected Outcome: Unsaved run screenshot should look identical to the saved run screenshot.

Screenshot of unsaved running:



Screenshot of saved running:

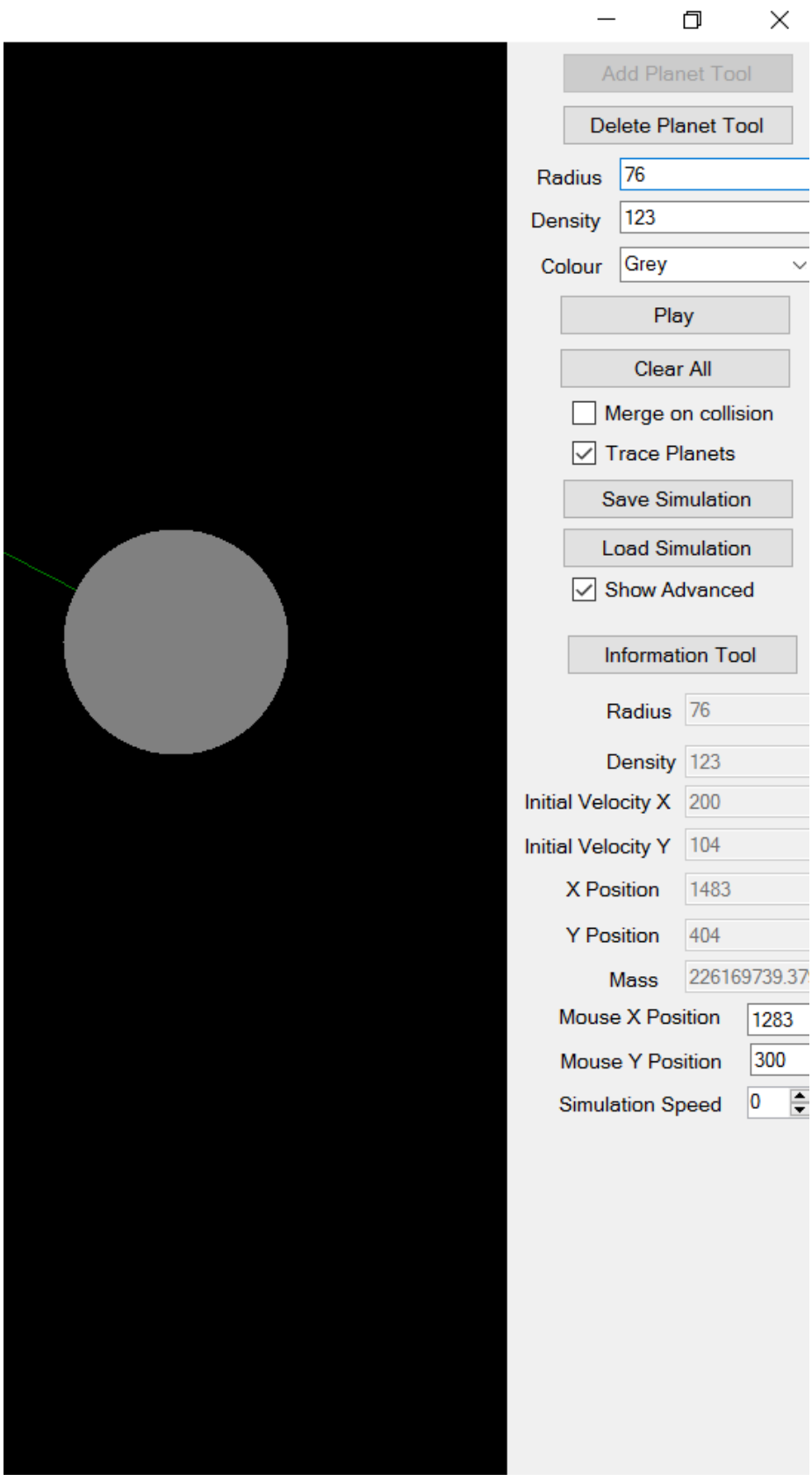


Test 6: Information tool

Test Description: I will create a planet with certain details and then I will click on it again with the information tool.

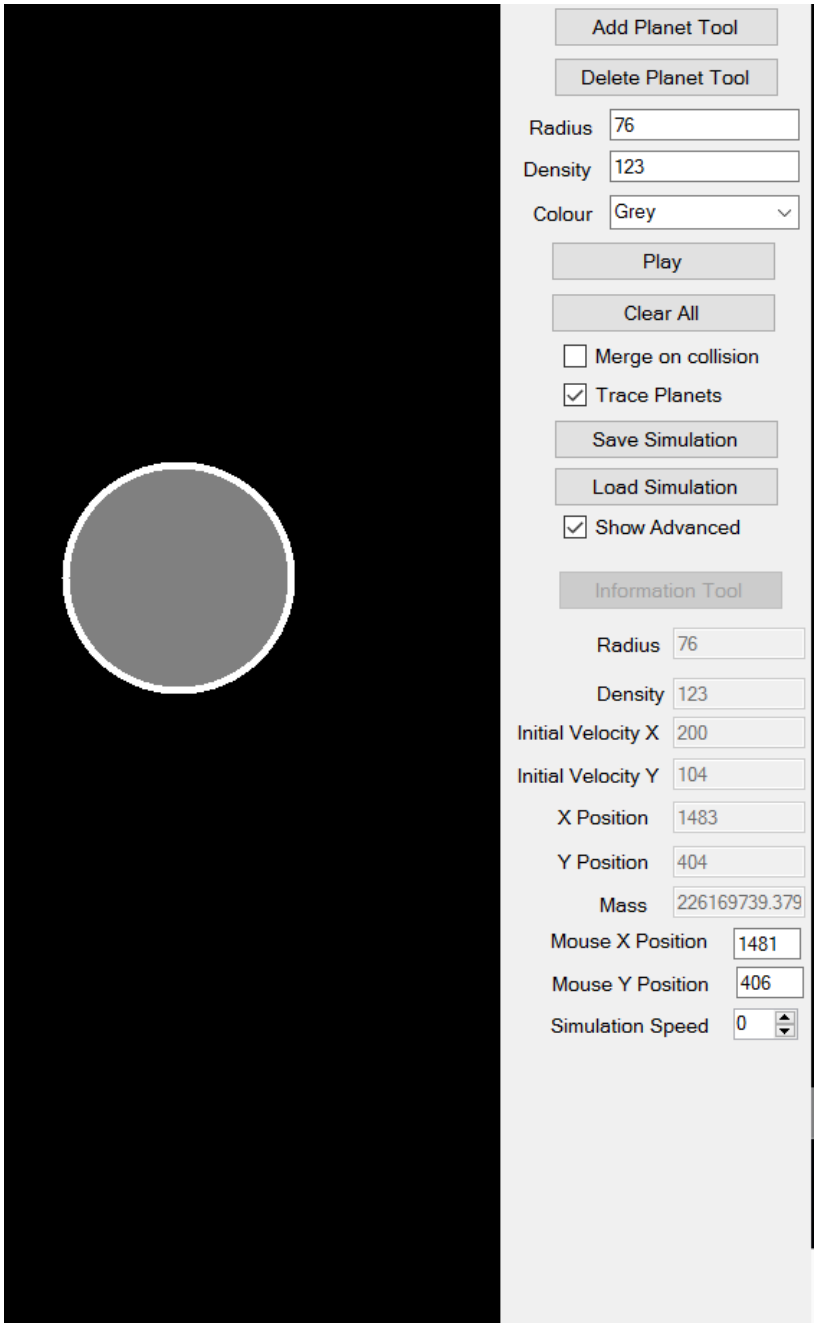
Expected Outcome: The planet should be highlighted in white and its details correctly put into the advanced tab.

Screenshot Before:



The details of this planet can be seen in the panel on the right

Screenshot After:

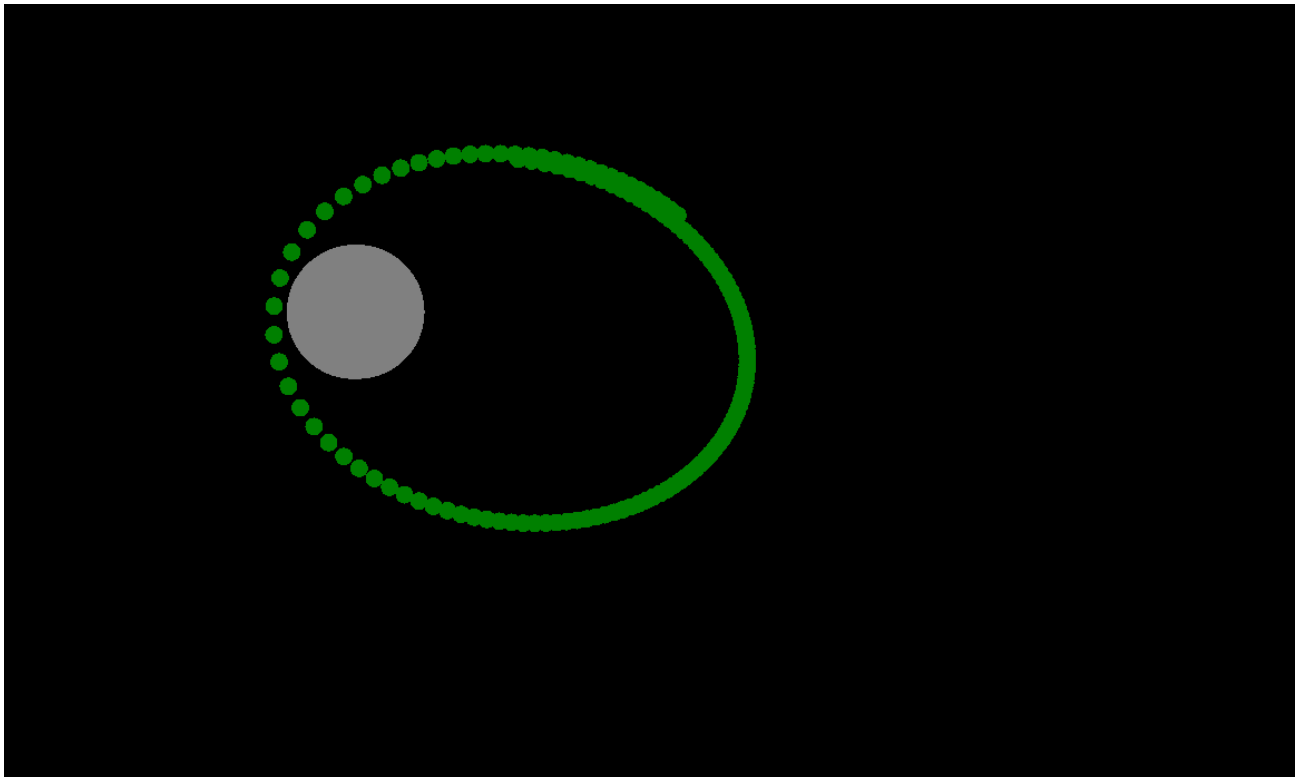


Test 7: Simulation speed

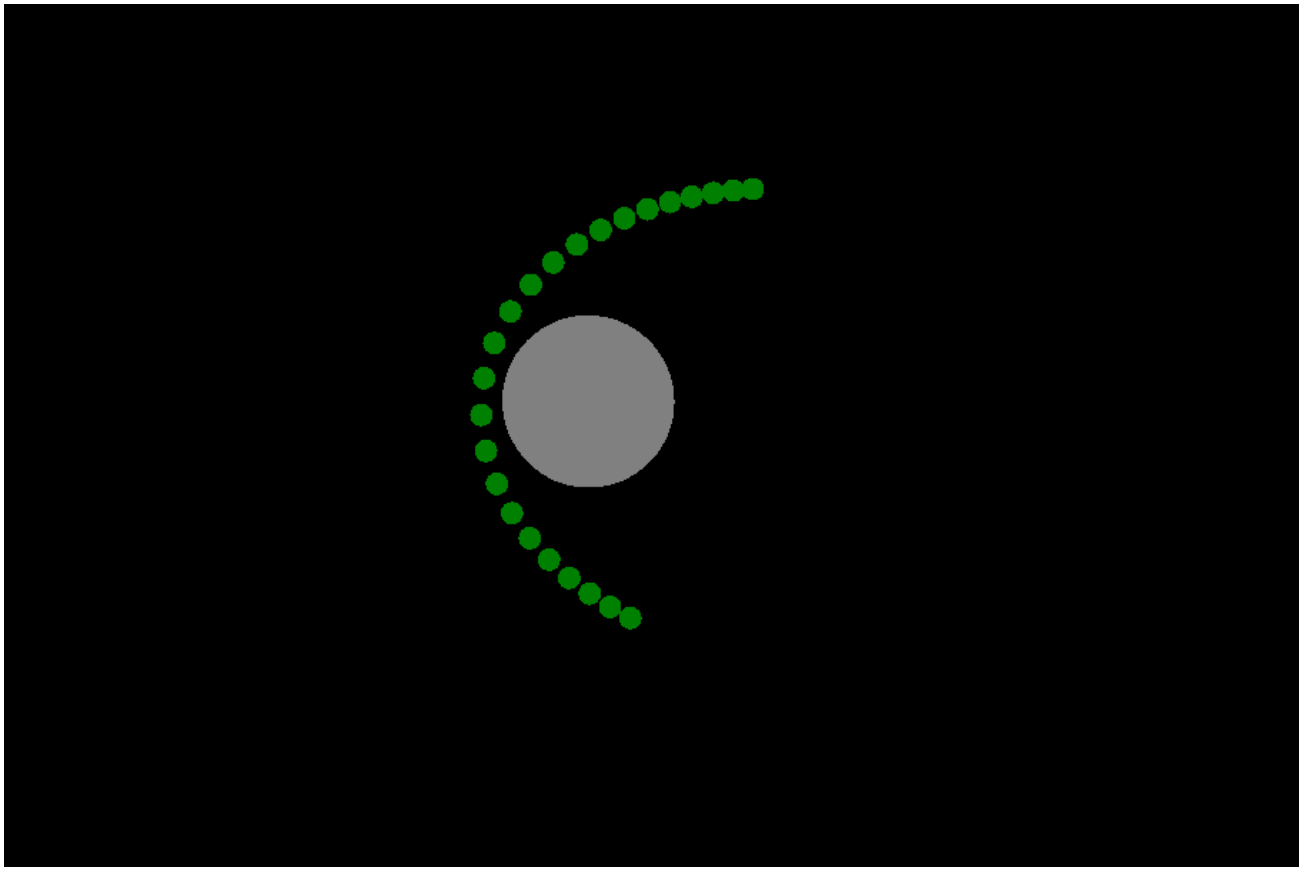
Test Description: To test the simulations speed feature I will run a simulation at speed 10 for 3 seconds and then run the same simulation at speed 0 for 3 seconds

Expected Outcome: The objects in the simulation running at speed 0 should move faster and so travel further in the same time

Screenshot of speed 0:



Screenshot of speed 10:



Black Box Testing

Because this is quite a complex program with many different options to tick or untick etc., it means that there are many different scenarios to test. Although I have outlined most of the common uses and functions of the program in the above testing there are other scenarios possible that I have not considered. I think that the ideal thing to do in this case is black box testing, this will allow me to have the program tested in ways that I was not expecting it to have to perform and to see if any bugs arise.

The two testers that I used were a Year 11 physics student and a year 13 physics student. I felt these were the main target audience for my program and the most likely students that the teacher would want to use the program with.

I firstly gave the program to each tester on a laptop and asked them to create an orbit, neither of them had any experience with the program and so this was a test of how intuitive the program was to use. I then allowed the testers to continue to play with the program for around fifteen minutes noting anything interesting that they were doing below.

Tester 1: GCSE Physics Student (Year 11)

Outline of what they did	**Comments**
Creates 20 planets – engaged trace and merge – altered speed to 0 – all planets merged – cleared all	
Create 5 planets and plays with varying speed – then they put trace on to watch them move – they paused the program and deleted a planet cleared all	
Create a large planet with five satellites in attempt of orbit – unsuccessful	
Placed multiple planets on screen then repeatedly pressed the play/pause – program crashed	This is likely due to the graphics object being called from the two-simultaneously running threads at once. I had encountered this problem before when I added the advance tab show/hide tick box, however, despite fixing it there I did not realise it was an issue with the play pause button. It is likely to be more complex of an issue with the play pause button as there is a lot more code running in relation to the graphics than there was with the show/ hide advance tab

Main points:

- He commented on flashing at speed 2 with 20 bodies on screen
- He suggested adding an introductory home screen with instructions on how to use the program
- He was also annoyed that when trace was on and he deleted a single planet all of the traces where deleted

- The crashing when repeatedly pressing play/pause doesn't seem like a large issue to me and it was likely enhanced by the fact that there were 20 bodies on the screen at that time

Tester 2: A Level Physics student (Year 13)

Outline of what they are doing	Comments
Attempted to create an orbit and was successful after a few attempts varying the initial velocity	They seemed to find the initial velocity entry system initially slightly confusing as they were expecting the planet to follow the line rather than move away from it. However, after playing with it for a few moments they understood and commented that it was 'like a piece of elastic'
They next saved the simulation that they had created	
They then attempted to load in a text file from another file location which had not been formatted in the way gravitas required – the program crashed when it tried to load this file	Whilst it would be safer to use a unique file format so that other programs cannot create incompatible files which gravitas may try to load it is not something which is achievable in the current timescale. It is definitely something I would add, however, given more time. Also, I think that it is important to note that the tester was deliberately trying to crash the program and most users would not stray from the premade 'Gravitas Saves' folder than has been created for them in their C drive.

Main points:

- She had to spend less time getting to grips with the way the program worked, however she still commented that she thought it would be difficult for a younger child to immediately understand
- Another alternative to having a unique file format would be to put some sort of validation before a file is loaded into the program. Perhaps, when every file is saved it has a number or string on the first line that the program can recognize so that it knows it was made by Gravitas and therefore is safe to try and load. The issue is that if I did it in that way it would mean that all previous saved files would be incompatible with the new version.

Evaluation

In this final section I will be going back to the physics teachers to see what they think of the solution that I've made and to see if they are happy that it meets the goals we set out in the analysis section.

In this first section, however, I will be evaluating my own solution based on the requirements and objectives in my analysis section.

Self-Evaluation

System Requirements:

- *Have an animated GUI showing the moving bodies*
  I think that this has definitely been achieved as my solution simulates the movement of bodies and the trace tool is helpful in seeing where the planets have moved from
- *Change Mass of the bodies by changing the density and radius*
  This has also definitely been achieved as the way that a planets mass is controlled in the program is by changing its radius and density
- *Have an initial velocity system, this will allow the users to create orbits easily by providing a perpendicular velocity to the pull of gravity*
  Again, as shown in test 4 of my system testing, providing a perpendicular velocity in this way will produce an orbit
- *Change colour of bodies to easily identify the different bodies that the user creates*
  Within the program there are five different colours that planets can be, I feel that this is an appropriate amount especially considering that the user initially asked to simulate five different planets at once. I think that, with the colours provided it is easy to create an easy to understand, yet complex simulation.
- *Clear and intuitive user interface for first time users, the program is designed to educate so it is important that the solution is intuitive and first time users are not put off by a long-winded manual and confusing controls.*
  I think that as shown in the black box system testing it does not take long for a user to pick up the program and start making some reasonably complex simulations.
- *Have the majority of inputs performed graphically so that it is easy for the user to understand the forces acting on the planet.*
  The main inputs for creating a planet are to change the radius and density and also to click and drag on the screen to create a planet and give it an initial velocity. It would be preferable to have the radius and density done graphically in some way but I could not think of a particularly easy and non-complex solution. I think that having a clearly labelled textbox is a good enough alternative.
- *Have an option to save simulations and also to load them later*
  I am happy that this works well, it may have been preferable to have the saves in a format other than text files as it is possible that the text file could be edited to a form that the program is not expecting to have to interpret. However, leaving the saved files as text files means that if someone wanted to do something like, make a spreadsheet to create saved files with perfect orbits, it would be much easier for them.
- *Have advanced functionality that allows advanced users to create more complex models, however this should not interfere with the simple user interface for the majority of users.*
  This requirement, I feel, has been somewhat fulfilled with the addition of the advanced tab, it allows he user to investigate the masses of the planets and calculate their pulls if they wish

to. A possible future feature may be to add an option to see the force pulling a planet from the other planets.

I will indicate the success of these objectives by using a traffic light system. Green for an achieved objective, amber for a partially achieved or unachieved objective (as long as there as is a reason why I couldn't achieve it or why I shouldn't do it in that way) and finally red if there is an objective I have not achieved for no reason.

System Objectives

Objective	Comments
Have a smooth animation that does not flash and is not stop start	Compared to my initial efforts to create this program the flickering has been massively reduced. There is no flicker when running at a speed of 3 or above on most computers (or when trace is on at any speed), however below 3 the flicker slowly gets worse. This flickering has been a large issue for me and I think that the main reason is that VB isn't particularly good for graphics. I did, in my design section, anticipate that it might not be as good as C# in this respect but I did not realise that it would be to this extent. If I were to start this process again I would probably be a lot more tempted to user C# or C++ even though I have gotten the flickering in my program down to a manageable level.
Allow the user to create and delete planets on the screen	Using the add and delete planet tools the user can add and remove planets by clicking on them
Have a pause and play mode so that the user can edit their model and then play it to see what happens	The user adds in their planets with velocities etc. and then clicks play to see what effect gravity will have
Have most of the functionality in one window so that the user is not confused by many different forms opening and closing.	There is only one form in the program, however the saving and loading opens a windows explorer dialogue but I feel that is reasonable as it is likely to be a format that the user is familiar with

Processing Objectives

Objective	Comments
Map the movement of, at least, of five bodies	My program can handle up to 20 objects and could do more, the limiting factor is the size of the array planets which has size 20. If the user wanted more than 20 planets then that is a possible change
Allow the user to define an initial velocity for a body that will be stored as a vector	By dragging the mouse after creating a planet the user can give it a velocity. This line's length and direction are then recorded and it is stored as a vector

Realistically mimic movement due to gravity of the bodies	I feel that it can mimic spherical and elliptical orbits as well as diverting a moving planet. The motion, to me, seems to fit with the idea I have of gravitational motion however, it is up to the physics teachers whether it is totally accurate or has some inaccuracies that I have failed to spot.
Resolve a minimum of six forces into one vector	As the program can handle and accurately map 20 different objects I feel that it is more than capable of this. To calculate the motion of one planet it must resolve the forces acting on it by every other planet
Calculate mass from density and radius for each planet	The only two text inputs for a planet are density and radius (aside from colour but this has no effect other than visually). From the density and radius the planets mass is calculated

User Objectives

Objective	Comments
Allow the user to alter the radius, density and colour of a planet	All three of these things work and changing them will change the type of planet the user will create when clicking on the canvas
All text boxes, buttons etc. should be clearly labelled to make it intuitive	All of the tools and textboxes that the user can interact with have tool tips and clear labels
Allow the user to delete one planet at a time	Using the delete tool the user can delete a single planet
Allow the user to delete the entire simulation at once	Using the clear all tool the user can delete the entire simulation
Have an additional advanced menu for more knowledgeable users, this should not intrude on the base UI.	The advanced tab is hidden from the user until it is requested by ticking the 'show advanced' tick box. The tab then shows and will give the user additional details on a selected planets mass, velocity, position etc. These details are crucial when trying to design a complex system where doing calculations with your values is crucial.

Possible improvements:

For all of these improvements I will list the improvement and then a brief outline on how I might add this to my current solution.

- Add a zoom function so the user can see off screen planets – For the zoom I would need to rescale all of the planets' radii to a certain factor depending on how much the user has zoomed in or out, the input for the zooming would probably be scrolling the mouse wheel up and down. Add a camera tracking function which will make a particular object the centre of the screen

- Add a way to copy and paste planets – I could do this perhaps by adding a right click menu when the user clicks on a planet, or perhaps by having the user use ctrl-C and ctrl-V. the main problems with these are how the user positions the planet they have pasted, it would probably be necessary to have a move planet tool which would be drag and drop.
- Add a predictive line to the planets movements when drawing the initial velocity line – this would make it easier to create an orbit graphically as it would require less guesswork – this could possibly be done by having the program calculate a few steps ahead when adding a new planet to the canvas. The program would then have to figure out where the planet will be after every calculation cycle and plot that point, finally it needs to draw a line through the point.
- Add an option to switch off gravitational attraction or to reverse it – this should be possible simply by putting a minus sign into the motion to invert it. As for switching of the gravity this would be possible by simply by cutting out the piece of code that calculates the forces and modifies the initial velocity with the motion from gravitational attraction.
- Add a tool that has the camera track one planet as the centre of the screen - I would do this not by moving the camera and the grid on which all the planets sit, but by moving the tracked planet to the centre of the grid and then moving all of the other planets by not only their own forces but also the relative motion. This would be reasonably complex and I think would take some time to work out exactly how the motion would be calculated but I think it would be possible.

Personal Evaluation

I've tried to create a fully working solution that fulfils the basic needs of the users. There are definitely improvements that I would like to make such as zooming and moving the camera position, however, because of a limited time scale and because I've been slightly limited by my choice of programming language I decided not to try to implement them. I felt it was more important that the user got a bug free and finished program with limited features than an advanced program which had major bugs and design shortfalls.

User feedback

Comments from black box tester 1 – GCSE Physics Student

'I think it needs a menu which has an option for a tutorial mode or a set of instructions'

'the flashing is a little bit distracting but it is usable at slower speeds'

Comments from black box tester 2 – A level Physics Student

'there are a few bugs, but on the whole it's not bad, it would definitely benefit on being able to move the camera around or zooming out though'

Comments from physics teacher

'It is an acceptable solution, in terms of being able to change the simulation it is very easy however it isn't especially intuitive to use and students would need explanation before they were able to use it properly.'

'It has all of the desired properties and tools but it would be helpful to have the zoom function'

Evaluation of feedback

Looking at the feedback that I got from my testers and also from the physics teacher it seems that the program isn't as intuitive as I had thought. This is obviously a possibility when designing a system as you have designed it in a way you think is sensible but it might not make sense to someone else. If I were to make changes to my system, I would like to add a tutorial mode where it shows you how to create an orbit etc.

Given more time I would also like to try and add functionality to zoom and pan the camera, however, I'm not sure that this would be possible to do (smoothly if at all) in VB.Net. I think it may be more sensible to re-do the system in a language such as C# or C++ which can handle graphics better. As I have said previously in this document, despite looking at a variety of languages at the very start of my design phase I underestimated just how complex it might be to do smooth and easy to manipulate graphics in VB.net. I think that if I had done the program in a C language I could have spent a lot less time trying to get acceptable graphics and more time fixing these few bugs that have crept into the final version as well as adding much need features such as the moving camera and tutorial mode.