

A-Level COmputer Science NEA

Gravity simulator

“Gravity explains the motions of planets,  
 but it cannot explain who sets the planets in motion.”

* Sir Isaac Newton

Alexander Fitzsimons

Contents

[1 Analysis 3](#_Toc2534099)

[1.1 Introduction 3](#_Toc2534100)

[1.2 Background 3](#_Toc2534101)

[1.3 Current solutions 4](#_Toc2534102)

[1.4 User Requirements 5](#_Toc2534103)

[1.4.1 Analysing the feedback 6](#_Toc2534104)

[1.5 Programming Language 7](#_Toc2534105)

[1.6 Objectives 8](#_Toc2534106)

[1.6.1 Initial Objectives 8](#_Toc2534107)

[1.6.2 Final Objectives 8](#_Toc2534108)

[1.6.3 Extension Objectives 9](#_Toc2534109)

[1.7 Acceptable Limitations 10](#_Toc2534110)

[2 Documented Design 11](#_Toc2534111)

[2.1 Project Overview 11](#_Toc2534112)

[2.1.1 Hierarchy chart 11](#_Toc2534113)

[2.1.2 Basic flowchart 12](#_Toc2534114)

[2.1.3 Calculations of the motions of objects 13](#_Toc2534115)

[2.1.4 14](#_Toc2534116)

[2.1.5 Object Search Optimisation 14](#_Toc2534117)

[2.2 GUI 15](#_Toc2534118)

[2.2.1 GUI design 15](#_Toc2534119)

[2.2.2 Panels 17](#_Toc2534120)

[*2.2.2.1* *Simulation Display* 17](#_Toc2534121)

[*2.2.2.2* *Tool Bar* 17](#_Toc2534122)

[*2.2.2.3* *Information Panel* 17](#_Toc2534123)

[*2.2.2.4* *Time Panel* 17](#_Toc2534124)

[*2.2.2.5* *Graph plotter* 18](#_Toc2534125)

[*2.2.2.6* *Learning panel* 18](#_Toc2534126)

[2.2.3 GUI Flowcharts 20](#_Toc2534127)

[2.3 Physics 23](#_Toc2534128)

[2.4 Saving simulations 24](#_Toc2534129)

[2.5 Object orientated programming 24](#_Toc2534130)

[2.6 Other External Files 26](#_Toc2534131)

[2.7 Libraries and modules 27](#_Toc2534132)

[2.8 Final algorithms pseudo code 28](#_Toc2534133)

[2.8.1 Classes 28](#_Toc2534134)

[2.8.2 Simulating Gravity 30](#_Toc2534135)

[3 Technical Solution 31](#_Toc2534136)

[3.1 The Code 31](#_Toc2534137)

[3.2 Other files 67](#_Toc2534138)

[3.3 Running the Final Program 69](#_Toc2534139)

[3.3.1 Starting up 69](#_Toc2534140)

[3.3.2 Pressing the exit button 69](#_Toc2534141)

[3.3.3 Adding an object in: 70](#_Toc2534142)

[3.3.4 Added multiple objects of different colours and showing their trail 70](#_Toc2534143)

[3.3.5 A stable orbit formed out of the chaotic beginning 72](#_Toc2534144)

[3.3.6 Demonstrating Kepler’s second law: 72](#_Toc2534145)

[3.3.7 Changing the y axis of the graph plotter 73](#_Toc2534146)

[3.3.8 Drawing the graph of the net force against time: 73](#_Toc2534147)

[4 Testing 74](#_Toc2534148)

[4.1 Functionality testing 74](#_Toc2534149)

[4.1.1 Testing the physics 74](#_Toc2534150)

[4.1.2 Testing the file management 74](#_Toc2534151)

[4.1.3 Testing the GUI 75](#_Toc2534152)

[4.2 Feature testing 75](#_Toc2534153)

[4.3 Panels 75](#_Toc2534154)

[4.3.1 Simulation features 75](#_Toc2534155)

[4.3.2 Graph testing 76](#_Toc2534156)

[4.3.3 Extension testing 76](#_Toc2534157)

[5 Evaluation 77](#_Toc2534158)

[5.1 Completeness 77](#_Toc2534159)

[5.2 Improvements and expansions 78](#_Toc2534160)

# Analysis

## Introduction

“Gravity explains the motions of planets, but it cannot explain who sets the planets in motion.”

– Sir Isaac Newton

In my project I intend to let the user be the person who sets the planets in motion.

My project will be an application designed to help a-level physics students to visualise physics principles based around gravity and help them understand the maths going on to make the ideas work. I have chosen to do this as physics is a subject I am currently studying at A level and as someone who may wish to use the application I intend to develop I think this would be a good choice of project.

## Background

Currently, educational applications and platforms for physics tend to be one of two types. The first type teaches exactly to the curriculum providing only tests, and quizzes designed to replicate exam questions. Websites which use this method include the following:

<https://2017.integralmaths.org/my/> <https://www.kerboodle.com/users/login?user_return_to=/app>.

However, this method neglects to visualise what is actually going on; the student may understand how to answer questions of the exact form they have been taught but without fully understanding how or why it works, meaning they could come across a question in their exam that wants them to use a basic principle they understand but the format is not what they are used to, so they struggle to answer it. The other type is one that demonstrates principles by showing how certain principles work visually. This is done through simulations, graphs, diagrams etc. Some souluations that do this are:

<https://www.myphysicslab.com/> <https://hermann.is/gravity/>

These are great for students to be able to visualise and understand the principles of physics, but their main flaw is that they sometimes don’t relate what they are demonstrating back to the mathematics behind the graphs and diagrams that often distract from the point it was trying to get across.

The application I will attempt to develop will be to help the student visualise gravity and orbits as well as showing the mathematical equations that are used so that a student learning with it has a better chance of fully understanding the topic of gravity and the mathematics behind it.

## Current solutions

Educational physics simulations can be found as websites. Here are some examples.

* <https://www.myphysicslab.com/>
  + This website includes many different physics simulations such as springs, pendulums and chains. There is a wide variety of types of simulations, however within the simulations themselves, there is little freedom of customisation. For example, the mars moon simulation is the one that demonstrates gravity however in this simulation you are limited to just the objects given. This is a common issue in this application. You are given a simulation to interact with, but you can’t create the simulation itself. However as a teaching tool it is very useful since it has graph plotting features and shows the equations that are related to the simulation.
* <https://hermann.is/gravity/>
  + This website focuses more on gravity, the topic I would like to cover. It shows the orbits of objects as they are affected by gravity. The user can add as many objects as they want giving the user more freedom to simulate whatever they want. However there is a lack of teaching tools making it more of a fun game to mess around with than an actual educational tool. There are no graph plotters or information about the mathematical formulas involved.
* <https://www.kerboodle.com/users/login?user_return_to=/app>.
  + This website provides online textbooks for students to read as well as allowing the teacher to set assignments and quizzes that the students can complete on the website. It lacks the visual, interactive experience that engages the student sometimes making it harder to grasp the concepts being taught.
* <https://2017.integralmaths.org/my/>
  + This website allows students to read about the maths they will require for their particular course as well as print off exam style questions and doing online tests. It is a website for maths but I have included it as the mechanics in maths crosses over with physics. This website is a much more mathematics based one, which is to be expected since it is a website for teaching a-level maths. However, when teaching mechanics, although providing examples of questions relating to real life, there is still a distinct lack of visual explanation that links the maths they are teaching to the real situation that they are modelling.

After considering these current solutions I have chosen certain features that work well and put them together to create a more user friendly and educational application. Inspired by <https://www.kerboodle.com/users/login?user_return_to=/app> and <https://2017.integralmaths.org/my/>, I will allow the user to see the equations related to the simulation. From <https://www.myphysicslab.com/>, I will let the user draw graphs of the current simulation. Finally, from <https://hermann.is/gravity/>, I will allow the user to add in multiple objects into the simulation.

In the application the user will be see the display split into different panels. Each panel will have a different purpose such as tool box, simulation display, graph plotter etc. The graphs can be used to compare different physical variables like the force acting on an object, or its velocity. The user will be able to customise their simulation to suit what they wish to experiment with.

## User Requirements

The users of this program are:

* A-level Physics students
* Physics teachers

In this section, I intend to find out what specific features my target users are looking for. This will allow me to design the program around the users needs. To do this I asked students from my physics class in school as well as teachers from both physics and maths at my school.

I asked the likely users of my project some questions as to what key features should be implemented for them to get the most out of the program. I have then analysed this feedback and then used this to help me with writing the final objectives. To ask them I first explained with a brief introduction, what the program was supposed to do. I then asked the specific questions about what they think would be good to implement. These questions were:

**Students questions**

Q1. What sort of interface would you expect from the program?

Q2. What features of the simulation would be useful to include?

**Teachers question**

Q3. What would be the best principles to teach students that require more visual understanding?

**Student Feedback**

The application should include:

Question 1:

* A user-friendly graphical menu system
* The use of buttons and switches
* Panels or windows to separate out the display into sections

Question 2:

* Adjustable options (e.g. change size of planet, change colour etc)
* Multiple objects in a simulation
* The option to view statistics and variables of the simulation
* An objects measures (velocity and force were mentioned by the students in this case) should be shown visually.
* Showing an objects path as it moves
* Demonstrations of a-level physics principles
* Some way to graph the simulation

**Teacher Feedback**

**General feedback**

An application that demonstrates the principles visually would help them to learn more quickly than an application that just asked students test questions.

**Specific feedback**

* It should demonstrate Kepler’s second law – since this is very visual and a simulation like this is the perfect way to teach students the law (Kepler’s second law is explained later)
* It should be able to draw the trail of the objects to show that the objects follow an elliptical path

### Analysing the feedback

In this section I will analyse the feedback that was given by the students and teachers

**Students Feedback Analysis**

“A user-friendly graphical menu system” This requires me to use a module that allows me to draw shapes, take mouse and keyboard input and animate.

“The use of buttons and switches” I could create classes called button and switch used for letting the user interact with the program. This would allow me to create multiple buttons and switches without defining how they function each time I create one.

“Panels or windows to separate out the display into sections” To do this I would have to design a layout (see the design section) to put the main simulation screen, information section, tool bar etc.

“Adjustable options” The user should be able to change things like the radius if the planets, the colour of the planets.

“Multiple objects in a simulation” The user should be able to add many objects into the simulation

“ The option to view statistics and variables of the simulation” The user should be able to view the information about the objects that they are adding into the simulation in metric units.

“An objects measures (velocity and force were mentioned by the students in this case) should be shown visually” An Idea for this could be drawing lines from each object representing vectors.

“Showing an objects path as it moves” Drawing a line behind the objects as they move

“Demonstrations of a-level physics principles” Specific examples are discussed in teacher feedback analysis

“Some way to graph the simulation” Take values of two of an object’s variables as coordinates. Repeat this for different points in time and draw lines between them to draw the graph.

**Teachers Feedback Analysis**

“It should demonstrate Kepler’s second law – since this is very visual and a simulation like this is the perfect way to teach students the law (Kepler’s second law is explained later)” For this I would have to draw the area between an object and the path of an orbiting object and state that the area is constant for constant periods of time anywhere in the orbit.

“It should be able to draw the trail of objects to show that the objects follow an elliptical path” This can be done by drawing a line from the object to its previous positions. When testing this it should show that the path is elliptical.

## Programming Language

I have chosen to write my code in python 3. This is because it is the programming language that I am most familiar with, it has lots of libraries and modules that I can use, and it uses object orientated programming. One of the libraries that I would like to use is Pygame. This will allow me to create the graphical user interface that is required as well as animating the simulation. I have had some basic experience in Pygame which will be enough for me to complete this project but it will also be a great opportunity to learn more about programming in Pygame and will give transferable skills to other languages as well. The object orientated programming is essential for the creation of unlimited objects, reducing repetition and helping readability of the code

## Objectives

### Initial Objectives

The initial objectives are a rough guide to the aims I wish to complete by the end of the project.

1. A simulation display showing the objects interacting with each other.
2. Add multiple objects into a simulation.
3. Objects obey Newtons laws of gravitation.
4. A graphical user interface allowing the user to interact with the simulation.
5. A graph plotter
6. A section showing the mathematical principles used in the simulation
7. The option to save and open simulations

### Final Objectives

The Final objectives are the more specific requirements for my project.

1. The GUI will include:
   1. Tool bar
   2. Time settings
   3. Graph plotter
   4. The simulation display
   5. The information panel
   6. Learn panel
2. Within the simulation, the user will be able to:
   1. Add new objects of different densities, volume, position and velocity
   2. View the stats of each object visually by:
      1. showing velocity by a line of a certain size and direction
      2. showing the ID of an object on top of that object
      3. showing the force acting on the object by a line of certain size and direction
      4. showing the previous positions of the object with a trail that it makes as it moves
   3. Affect the speed that the simulation runs using:
      1. Pause
      2. Play
      3. Increase speed
      4. Decrease speed
3. The simulation will calculate the following:
   1. the gravitational force an object has on another object
   2. The acceleration of an object with the force calculated before
   3. The velocity of an object in the next frame based on the acceleration
   4. The position of an object in the next frame based on the velocity
   5. When objects collide, their masses combine
   6. When objects collide, their densities average out according to each mass and each density.
   7. I would have to do all of these calculations for vectors in the x and y directions since I will be dealing with 2 space dimensions.
   8. Colliding objects should also conserve momentum
4. The graph will be able to display the relationship between two variables of selected objects:
   1. Velocity x and y
   2. Net Force
   3. Time

### Extension Objectives

The Extension Objectives are additional features that I may wish to add in later stages depending on the completeness of the project.

1. Instructions for the simulation
2. Coloured objects
3. Saving graphs

## Acceptable Limitations

I will allow myself some acceptable limitations that may appear in the final program. Some of these include:

Just a two-dimensional simulation – a 3D simulation would be better but orbits occur in a 2D plane so this is not necessary and would increase the time complexity of the position calculations from O(n2) to O(n3), not to mention the fact that Pygame does not let you draw in 3D directly and you would have to create sphere drawing functions as well as dealing with perspective in a 3D universe. Therefore, would be more viable to make a 2D simulation instead of 3D.

Although Newtonian gravity is an accurate model for most occasions, in some circumstances a more accurate model would be Einstein’s theory of general relativity. However, I will not be including this as it is beyond a-level physics so is not needed by my target audience nor do I have the knowledge myself to implement this.

The collision of objects does not have to be complex at all either as this is not required by the a-level specification. Colliding objects becomes very complex if you consider all possible outcomes. One case is when two objects merge into each other. This is the most likely option to happen (and the way I will be dealing with collisions) because as the mass of two objects increases their gravitation attraction to each other increases but the strength between molecules that makes up the objects structures remains the same. This all means that when astronomical objects collide they almost always break up into small pieces and often fall together to merge into one object. However, if when two objects collide, they bounce off each other, there is still a lot to calculate. This would mean I would have to define a coefficient of restitution (how bouncy one material is with another) between each object making n2 extra variables to store (n = number of objects and each object has one value with every other object). There may also be a combination of both options where some mass is transferred from one object to the other, but the rest is deflected off.

One final limitation would be certain features of a graphical user interface. I plan to make all features that are necessary to the program but since I am building the GUI from scratch making classes for panels, title bars, buttons and switches, an acceptable limitation would be more advanced GUI features like resizable panels and the ability to create new panels etc.

# Documented Design

This section will be split into 3 parts these are:

* Overview – looking at the program as a whole
* Physics – looking at how calculations for the actual simulation are going to be done
* GUI – designing how the program will be displayed and how the user will interact with the program.
* Other – designing other things like search algorithms, saving simulations etc

## Overview

### Hierarchy chart

The following hierarchy chart will show the splitting up of the sections of code. The code won’t be fully segmented like this because the code will not be that clearly separated as it makes sense to do all calculations and displaying for the simulator panel at the same point in the code before moving on to calculations and displaying for the graph plotter. Also, due to layering of drawing on the screen there has to be certain ordering to the code so that objects on the screen are overlapping correctly.

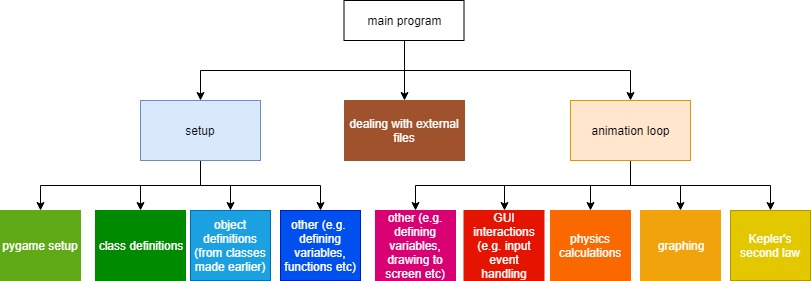
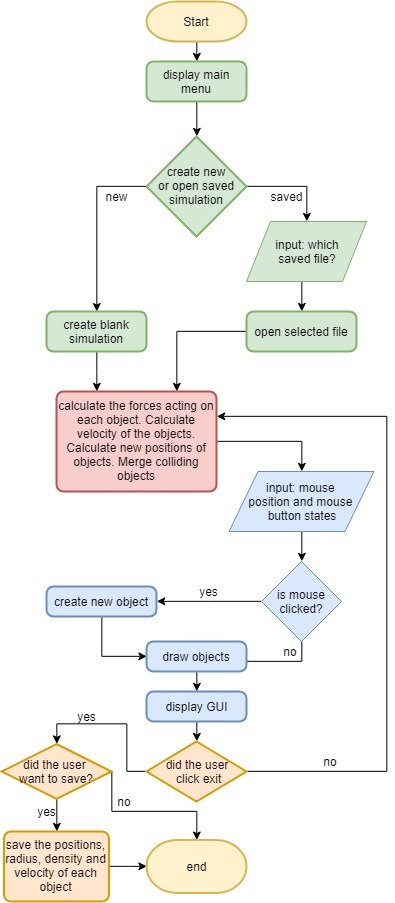


Figure 8

### Basic flowchart



This diagram shows main flow of the program with the most basic features (simulation display, opening saved simulations and saving simulations).

## Physics

### Physics flowchart

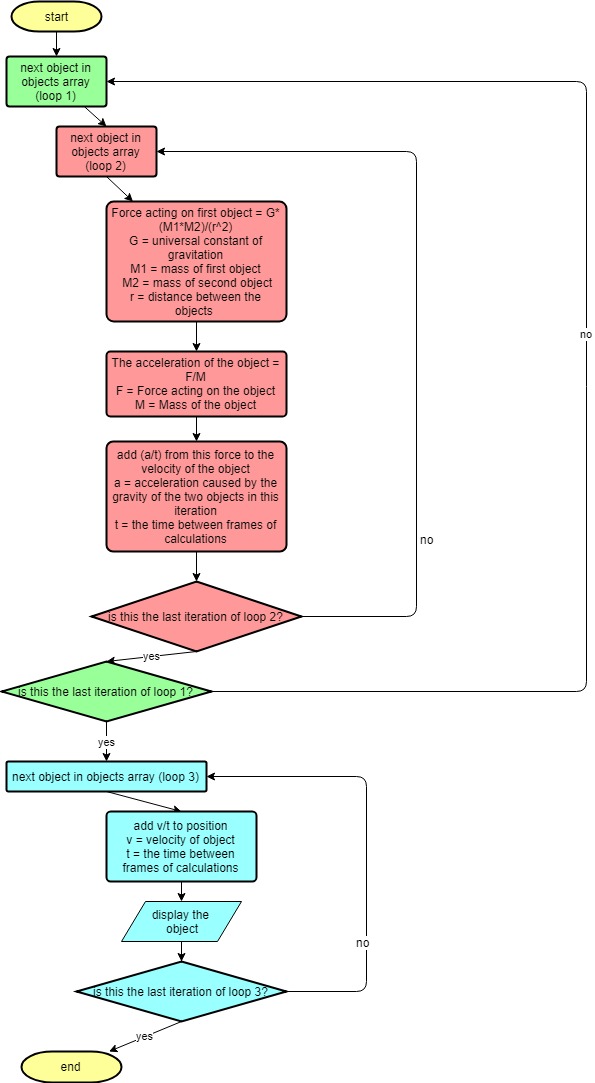


diagram 4

### Physics calculations

To be able to simulate the motion of astronomical objects I will have to apply newtons laws of motion amongst other physical and mathematical ideas. The most crucial and the equation at the heart of the program will be Isaac Newton’s law of gravitation. This model for gravity is accurate for speeds that are small in comparison to the speed of causality (the speed of light). One of my acceptable limitations to my project is that I would not consider general relativity in my calculations. I have chosen to do this partly because it would be beyond my knowledge of physics but also because the time complexity is already large because I will need a nested loop. The first loop would be to iterate through all objects in the simulation and calculate the total force acting on each object. The second loop would iterate again through all the objects to calculate the gravitational force acting on the object that it is currently on in the first loop. With all of this to calculate, general relativity would add another whole layer of complexity that would significantly increase the time complexity.

Newtons law of gravitation:

F = Force acting on the object (Newtons)  
G = Gravitational constant (6.67\*10-11)  
m1 = Mass of the first object (Kg)  
m2 = Mass of the second object (Kg)  
r = Distance between the centres of the objects (m)

To calculate the position of an object I will use the current position in meters, current speed in meters per second of the object and the time since the last calculation (1/FPS \* time adjustment factor) in seconds.

Equation for position:

x2 = New position (m)  
 x1 = Current position (m)  
 v = Current velocity (ms-1)  
 t = Time interval (s)

To find the velocity of the object I will need the current velocity in meters per second, current acceleration in meters per second2 and the time since the last calculation in seconds.

Equation for velocity:

v2 = New velocity (ms-1)  
 v1 = Current velocity (ms-1)  
 a = Current acceleration (ms-2)  
 t = Time interval (s)

To find the acceleration of the object I will need the Force acting on the object in Newtons, and the mass of the object

Equation for acceleration:

a = New acceleration of the object (ms-2)  
 F = Force acting on the object (N, calculated by the equation for gravity)  
 m = Mass of the object (Kg)

### Physics pseudo code

The following section is the pseudo code based on the basic flowchart, the physics flowchart and the calculations section.

###Creating Object Class###

CLASS Object{

DEF initialise(Self, ID, ObjectColour, ObjectRadius, PositionX, PositionY, VelocityX, VelocityY, density){

Self.ID = ID

Self.colour = ObjectColour

Self.displayRadius = ObjectRadius

Self.displayPosX = PositionX

Self.displayPosY = PositionY

Self.velX = VelocityX

Self.velY = VelocityY

Self.metricRadius = MetersPerPixel \* ObjectRadius

Self.metricPosX = MetersPerPixel \* PositionX

Self.metricPosY = MetersPerPixel \* PositionY

Self.density = denisty

Self.MetricMass = pi\*(4/3)\*(self.metricRadius^3)\*self.density

Self.netForce = 0

}

DEF displayObject(self){

DRAW circle(Self.colour, Self.displayPosX, Self.displayPosY, Self.displayRadius)

}

DEF updateDisplayUnits(self){

Self.metricRadius = ((3\*self.metricMass)/(4\*pi\*Self.density))^(1/3)

Self.displayRadius = self.metricRadius/MetersPerPixel

Self.displayPosX = self.metricPosX/MetersPerPixel

Self.displayPosY = self.metricPosY/MetersPerPixel

}

DEF changePositions(self){

Self.metricPosX += (self.velX\*Timeinterval)  
 Self.metricPosY += (self.velY\*Timeinterval)

Self.displayPosX = self.metricPosX/MetersPerPixel

Self.displayPosY = self.metricPosY/MetersPerPixel

}

}

###Using Objects class###

defaultRadius = 10

G=6.67\*10^-11

Objects = []

animationLoop = True

ClickedLastFrame = False

ID = 0

defaultDensity = 5500

WHILE animationLoop{

Clicked = DETECT LEFT MOUSE BUTTON DOWN

X = GET HORIZONTAL CURSOR POSITION

Y = GET VERTICAL CURSOR POSITION

FOR Self IN objects{

Self.netForce = 0

For Other IN objects{

If Other IS NOT Self{

PX = Self.velX\*Self.MetricMass + Other.velX\*Other.MetricMass

PY = Self.velY\*Self.MetricMass + Other.velX\*Other.MetricMass

x = self.metricPosX-Other.metricPosX

y = self.metricPosY-Other.metricPosY

distance = SQRT((x^2)+(y^2))

force = (G\*Self.MetricMass\*Other.MerticMass)/(distance^2)

Self.netForce+= force

acc = force/self.metricMass

Self.velX -= acc \* (x/distance) \* Timeinterval

Self.velY -= acc \* (y/distance) \* Timeinterval

IF distance <= (Self.metricRadius+Other.metricRadius){

totalMass = Self.MetricMass + other.MetricMass

SelfPercent = Self.MetricMass/totalMass

OtherPercent = Self.MetricMass/totalMass

combinedDensity = (SelfPercent\*self.density) + (OtherPercent\*Other.density)

objects.REMOVE(Other)

Self.density = combinedDensity

Self.velX = PX/Self.MetricMass

Self.velY = PY/Self.MetricMass

}

}

}

}

FOR Self IN objects{

Self.changePos()

Self.updateUnits()

Self.displayObject()

}

DRAW circle (blue, X, Y, defaultRadius, fill=FALSE):

IF Clicked AND NOT ClickedLastFrame{

initialY = Y

initialX = X

}

ELSE IF Clicked AND ClickedLastFrame{

DRAW circle(grey, inititalX, initialY, defaultRadius)

DRAW line(green, initialX, initialY, X, Y)

DRAW circle(yellow, X, Y, defaultRadius, fill = FALSE

}

ELSE IF NOT Clicked AND ClickedLastFrame{

Objets.append(Object(ID, grey, defaultRadius, initialX, initialY, initialX-X, initialY-Y, defaultDensity))

}

ClickedLastFrame = Clicked

WAIT (TimeInterval)

}

## GUI

### GUI design

The first GUI that the user will see is the main menu.

In the menu are two buttons to either open a file or enter a new simulation

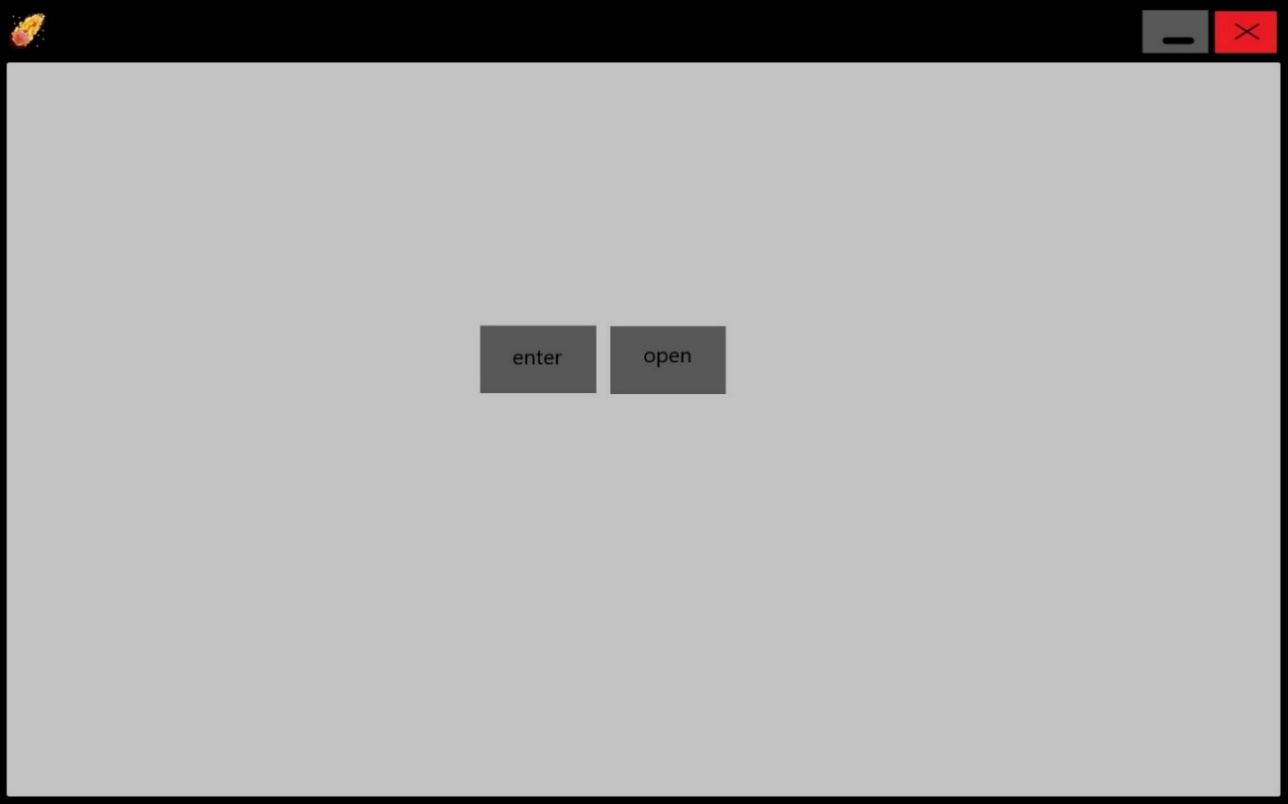


Figure 1

When the user presses open it shows all of the file that have been saved as well as the pre-set simulations.

A screenshot of a cell phone

Description generated with very high confidence

Figure 2

### Panels

The panels discussed in the objectives will include:

* Tool bar
* Information panel
* Time panel
* Simulation display
* Graph plotter
* Learning panel

#### Simulation Display

This is the most important panel which shows the objects that the user has added in. The objects are displayed as circles, gravity is modelled at its centre of mass and has a uniform density throughout the object.

#### Tool Bar

This will have switches to turn on different tools. One tool will be show velocity switch which will draw a line from each object with the size indicating the magnitude of the velocity and the direction showing the direction of the velocity. Another tool is the show force. This will draw a line from each object showing the direction and magnitude of the force acting on the object. Show ID is a tool that will display a number on each object. This number is a unique identifier of each object. The final tool will be the show trail. This will draw the path that the object has taken allowing the user to see how the object moves more easily.

#### Information Panel

This panel holds information about the current moment in the simulation. It will display numbers relating to what the user has set. One of these is the current radius. This is the radius of the cursor on the simulation. The cursor has a ring drawn around it to indicate the size of the object that would be made if they were to add one in. In the simulation the user will be able to change the radius by scrolling. Another is the current density. This is the density in Kg/m^3 that the object will be if the user were to add one in. The user should be able to change this by holding some keys down and scrolling. The current scale gives the distance that each pixel in the simulation represents. This is measured in meters per pixel. The user can change this by holding some keys and scrolling which will have the effect of zooming in or out. The final piece of information is the current speed of time which has no units as it is just the ratio of normal time compared to the simulation time. This measures how many seconds pass on the simulation for every second that passes in real time. This can be changed by using the buttons in the time panel.

#### Time Panel

The time panel will hold the switch for pause/play, so the user can see the simulation at one moment in time. It will also contain the buttons for speeding up and slowing down time. The last button in the time panel is the reset button which removes all objects from the simulation and resets the information in the information panel. The speeding up and slowing down time will not be available when the graph is being drawn.

#### Graph plotter

Used to show a comparison between two variables. The graph plotter consists of an x axis and a y axis in which to plot points to. Most variables with the exception on time are dependent on an object in the simulation. By default, the object to measure will be the object with an ID of 0. The user should be able to change this by clicking on the axis and a screen will appear asking which object they want to change to and which variable to measure.

#### Learning panel

This will be used to show certain equations and run a demonstration. The equations that will be shown are the equations for motion, and the equations for gravity. The demonstration is Kepler’s second law. This is only available when exactly two objects are in the simulation. It draws the area between the path of one object and the center of mass of the other object over a fixed period of time. It then repeats this showing that these areas are always equal at any point in the orbit.

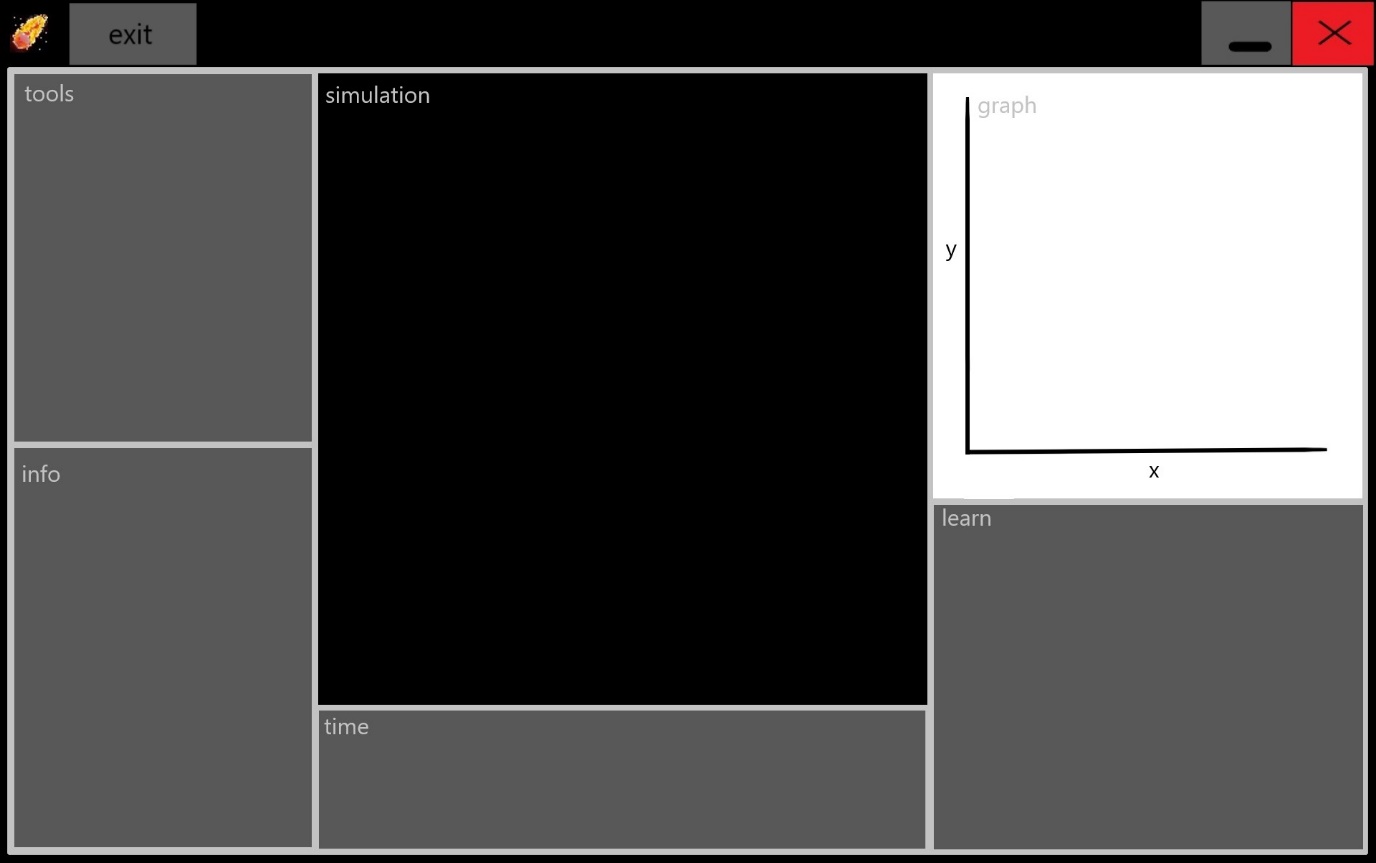
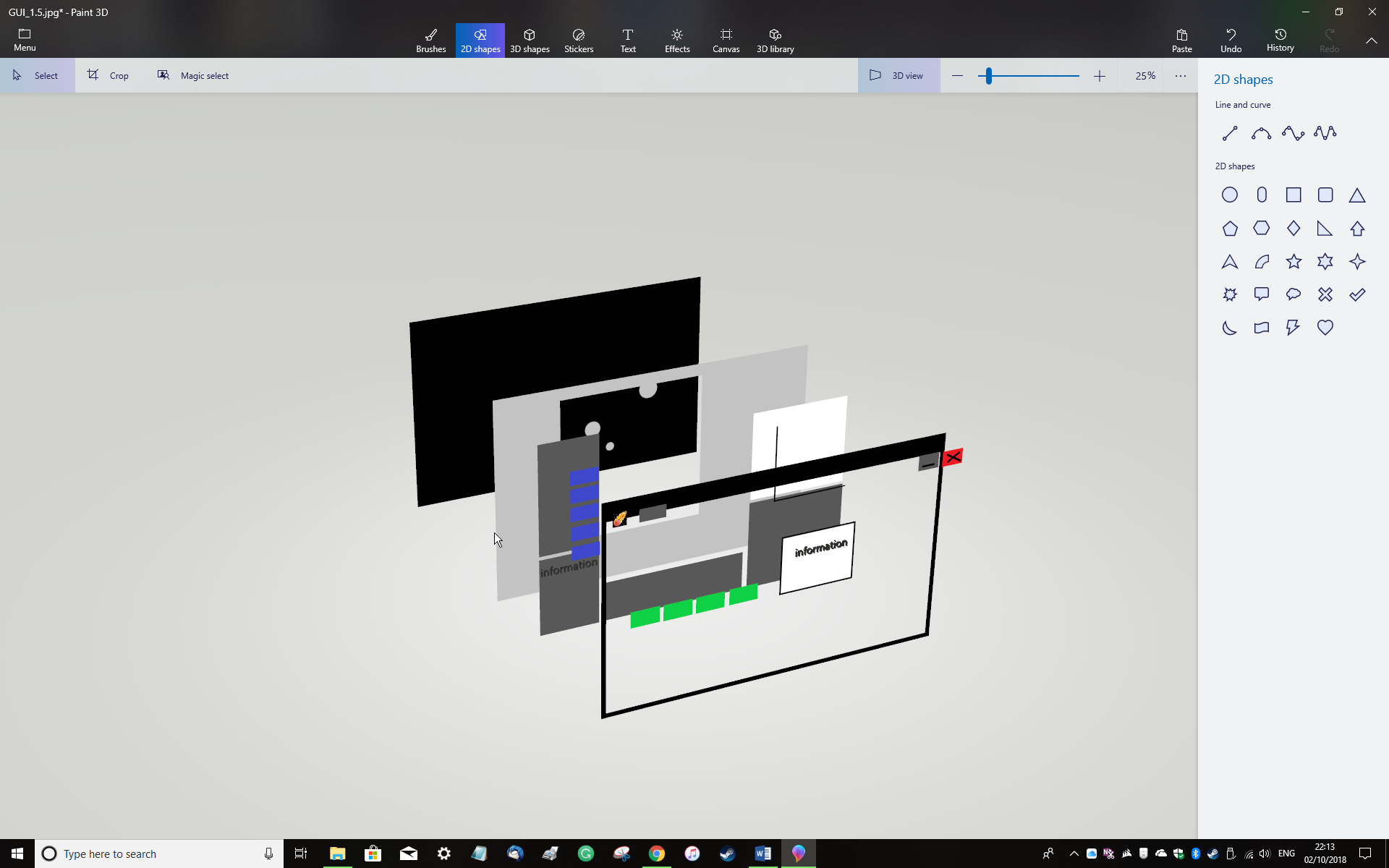
 

Figure 3

Panel to teach the physics principles used in the program

Pause/play, speed up and slow down

Lets the user view stats about the current simulation

Lets the user change features of the simulation

close

minimise

Shows a graph of variables of the users choosing

Shows the simulation with the objects

Returns the user to the main menu

Figure 4

The image above shows the layering of the panels, buttons, switches etc. I designed it so that when the objects moved off the simulation panel they pass underneath the other panels preventing the objects from covering the GUI.

The rest was designed so that buttons, switches, text and images were visible by being layered on top of the panels.

The way to order the layers in the program would be objects in the foreground get displayed after the objects in the background have been displayed.

### GUI Flowcharts

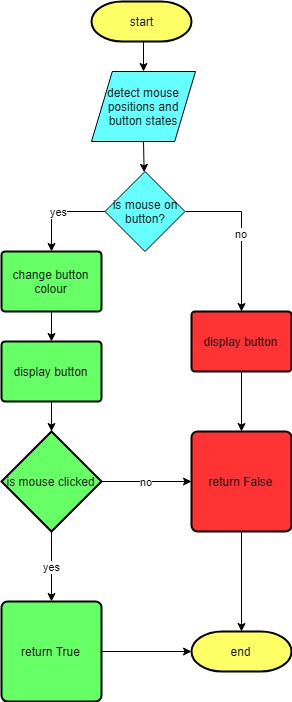


diagram 2

This flowchart shows how I would program a button.

A close up of text on a white background

Description generated with high confidenceThis flowchart shows how I would program the switch.

diagram 3

GUI pseudo code  
###Creating Button class###

CLASS Button{

DEF initialise(self, buttonName, buttonColour, PositionX, PositionY, SizeX, SizeY){

Self.name = buttonName

Self.colour = buttonColour

Self.PosX = PositionX

Self.PosY = PositionY

Self.SX = SizeX

Self.SY = SizeY

}

DEF displayButton(Self, NewColour){

DRAW reactangle(NewColour, Self.PosX, Self.PosY, Self.SX, Self.SY)

DRAW text(Self.name, black, Self.PosX, Self.PosY)

}

DEF detectClicks(Self, Clicked, X, Y){

IF(X>Self.PosX AND X<Self.PosX+Self.SX AND Y>Self.PosY AND Y<Self.PosY+Self.SY AND Clicked) {

RETURN white, TRUE

}

ELSEIF (X>Self.PosX AND X<Self.PosX+Self.SX AND Y>Self.PosY AND Y<Self.PosY+Self.SY){

RETURN white, FALSE

}

ELSE{

RETURN self.colour, FALSE

}

}

}

###Using Button class###

Clicked = DETECT RIGHT MOUSE BUTTON STATE

X = HORIZONTAL MOUSE POSITION

Y = VERTICAL MOUSE POSITION

B1 = Button(“Name1”, red, 50, 50, 50, 30)

Colour1, B1Clicked = B1.detectClicks(Clicked, X, Y)

B1.displayButton(Colour1)

Timeinterval = 1/120

IF (B1Clicked == True) {

DO CODE FOR B1 IS PRESSED

}

###Creating Panel class###

CLASS Panel{

DEF initialise(self, PanelName, PanelColour, PositionX, PositionY, SizeX, SizeY){

Self.name = PanelName

Self.colour = PanelColour

Self.PosX = PositionX

Self.PosY = PositionY

Self.SX = SizeX

Self.SY = SizeY

}

DEF displayPanel(Self){

DRAW reactangle(Self.colour, Self.PosX, Self.PosY, Self.SX, Self.SY)

DRAW text(Self.name, black, Self.PosX, Self.PosY)

}

}

###Using Panel class###

P1 = Panel(“Name2”, grey, 10, 10, 300, 600)

P1.displayPanel()

## Other

### Saving simulations

I also plan to let the user save simulations. This will involve storing data about all the objects in the simulation. To do this I plan to use a plain text file. To organise the data I will need a 2-dimensional list. The outer list will contain other lists within it. Each of these inner lists will be used to represent each object in the simulation. The inner lists themselves must hold all the necessary data that will be needed to re-create the simulation when the user wants to open them again. This data will be the main attributes of the objects themselves. They will be (\*metric units):

* The position in the x axis\*
* The position in the y axis\*
* The x velocity component\*
* The y velocity component\*
* The radius of the object\*
* The density of the object\*
* The ID of the object

I will not need to store any forces acting on each other since this can be calculated from the other data that is saved when it is loaded again.

To create this new list, I will iterate through the list containing all of the objects and for each object I will add its necessary attributes to a list using listname.append(object.attribute). Once I have created this inner list of attributes I will use append again to add the inner list to the outer list. This will then repeat as I iterate through each the list of all objects. The final newly created list will be the one that I will save.

Once I have created this 2-dimensional list, I will then be able to convert it to a string which I can then write to a new text file. When the user wants to open the simulation again, the program will #open the text file as a string. Since this string is in the format of a list, I can use the built-in function eval, to convert it to a list recognised by python. When I have converted it to a list, I can loop over it creating objects with initial attributes that are the same as the ones I have saved.

[[object 1 attributes], [object 2 attributes], [object 3 attributes], [object 4 attributes]]

Each of the inner lists will be structured as:

[ID, mass, x position, y position, x velocity, y velocity, density]

### Object Search Optimisation

Since the simulation supports many objects, it would be quite inefficient to do a linear search on all objects to find just one of them. The IDs of the objects are an integer and would be in order in the list of objects since I would append new objects to the end with each new object having an ID one higher than the last. However, objects can get destroyed by colliding with a larger object so although the object IDs are in order, the IDs are not necessarily consecutive numbers. This means that a binary search would be suitable for this. Binary search is generally more efficient than linear search because it reduces the number of objects that it looks at in the list before finding the required object.

Linear search (not used in this instance) iterates through the list checking if the ID of the object is equal to the ID of the object we are looking for. E.g. looking for 23:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| object | 0 | 3 | 5 | 7 | 8 | 11 | 14 | 23 | 77 | 99 | 100 |
| attempt | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |  |  |

Table 1

Found in 8 repetitions

Binary search on the other hand looks at the middle object and if the ID of the object we are looking for is higher it cuts the list down to just the upper half, and if the ID is lower, it cuts the list down to the lower half. The process repeats on the new list until the object is found. E.g.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| object | 0 | 3 | 5 | 7 | 8 | 11 | 14 | 23 | 77 | 99 | 100 |
| attempt |  |  |  |  |  | 1 |  | 3 | 2 |  |  |

Table 2

Found in 3 repetitions

If we were searching for 0, linear search would work better since it starts on 0 and binary search starts in the middle. However, there are more cases where binary search is more efficient than linear search therefore, I will use binary search instead of linear search for looking for a specific object in the array.

### Object orientated programming

In my project there will be many opportunities for me to use object orientated programming. Creating a class for the objects in the simulation will be essential for organising the objects attributes.

The next table shows the classes that will be required

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name of class | Use | Attributes | Functions | Examples of objects created from this class |
| Object | The class that creates and displays simulation objects and updates values. | \*there will be metric and display versions of these   * X/Y Position\* * Mass\* * ID * Colour - extension * X/YVelocity\* * Radius\* * Density | * Initialisation * Display object * Update units * Changing positions * Show ID | These are created each time the user clicks on the simulation screen |
| Button | A button will allow the user to click on it to interact with the program. It is used for one time clicking or for a click and hold option | * Name * Colour * X position * Y position * Size X/Y | * Initialisation * Display button * Detect clicks | * Close button * Minimise button * Enter button * Open button |
| Panel | Used to segment the display into different sections based on their use | * Type * Colour * X/Y position * Size X/Y | * Initialisation * Display panel | * Tool bar * Simulation panel * Time panel * Graph panel |
| Switch | Similar to a button but instead of a one-time action, it is a toggle for a specific action | * Name * Colour * X/Y position * State (on/off) * On and off text * Size X/Y * Was clicked in last frame | * Initialisation * Display Switch   (click detection is done in outside of the class) | * Pause and play switch * Togglable simulation features like showing the ID, showing the velocity of objects etc |

*Table 3*

### Other External Files

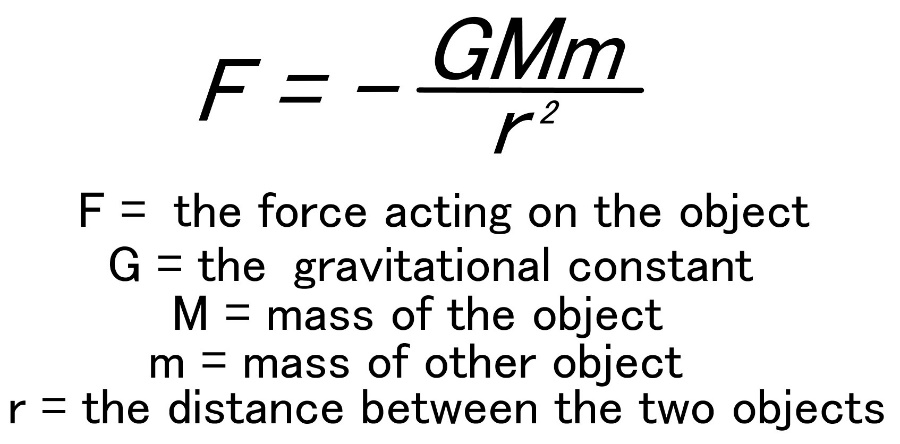
The other files that I will need to keep in the program folder (the location in which the program itself is stored) are images. These images can be seen below

Figure 5

This image will be able to be seen in the learn panel when the user clicks on the gravity button

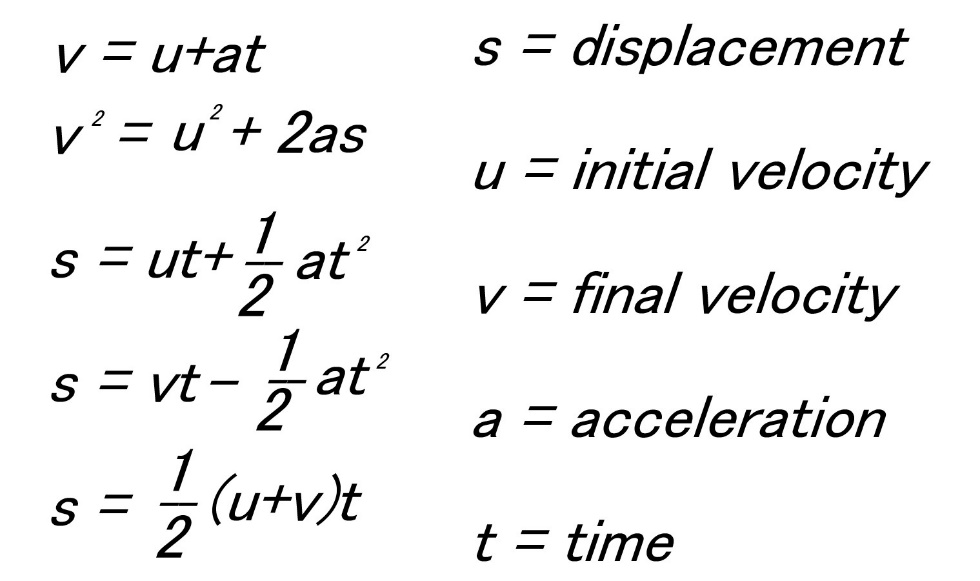


Figure 6

This image will also be able to be seen in the learn panel when the user has clicked on the SUVAT button



Figure 7

This image is the icon that will be able to be seen in the top left corner at all times that the program is running.

### Libraries and modules

The python libraries I will be using are:

* Sys
  + exiting
* Random
  + creating random numbers
* Time
  + waiting a period of time
* os
  + getting file names in a directory
* Math
  + the constant pi
  + the function sqrt()
* Pygame
  + Dealing with images
    - loading images
    - transforming images
    - displaying images
    - setting the window icon
  + dealing with text
    - creating fonts
    - creating text with those fonts
    - displaying text
  + drawing GUI
    - drawing rectangles
    - drawing lines
  + drawing objects
    - drawing circles
  + animating
    - updating display
    - waiting between each frame
  + detecting input events
    - using scroll wheel
    - mouse buttons
    - key presses

# Technical Solution

## The Code

To describe what is happening in the program I have made the following commenting format:

#### title

Whole program

### comment on large section of code

Large section of code e.g. a class definition or a large while loop

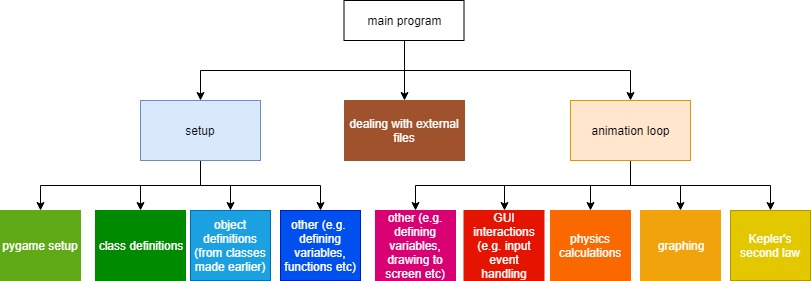
## comment on a small section of code

Small section of code e.g. a small for loop or a large if statement

# one to two line comment

One to two lines of code

Here is the hierarchy chart again. The colours of each box correspond to the coloured line to the left of the code.



1. ####orbit program
3. ###setup
4. ##importing required libraries
5. **import** sys #for exiting python
6. **import** time #for waiting periods of time
7. **import** math #for quickly performing mathematical functions
8. **import** pygame #for all graphics, animation and event handling during the program
9. **import** os #used later for listing the files in a folder
10. **import** ctypes #see next line of code
12. #compenstates for the adjustments made by some computers with high DPI displays that squash application displays down
13. ctypes.windll.user32.SetProcessDPIAware()
15. ###getting pygame ready
16. ##initialising pygame
17. pygame.init()
18. pygame.display.init()
20. ##defining the pyagame display
21. #sets the screensize to the resolution of the monitor used and puts the display in full screen
22. display = pygame.display.set\_mode((0, 0), pygame.FULLSCREEN)
24. #sets display width to the screen's width resolution
25. width = int(((pygame.display.Info()).current\_w))
26. #sets display height to the screen's height resolution
27. height = int(((pygame.display.Info()).current\_h))
29. #loads the image for the icon
30. icon = pygame.image.load('icon.jpg')
31. #sets the program icon to the image we just loaded in
32. pygame.display.set\_icon(icon)
33. #re-scales the icon to fit in the corner of the page
34. icon = pygame.transform.scale(icon, (30, 30))
36. #loads the image for the gravitation formulas used later to teach about newtons law of gravitation
37. gravitationFormula = pygame.image.load('formulas.jpg')
38. #loads the image for the suvat formulas used later to teach about motion
39. suvatFormula = pygame.image.load('suvat.jpg')
41. #loads the image of the colour wheel used to select object colours
42. colourWheel = pygame.image.load('ColourWheel.png')
43. #loads the lightness scale - also used to select object colours
44. lightness = pygame.image.load('lightness.jpg')
46. ##setting up the timing
47. #sets the number of frames that pass each second to be 120
48. FPS = 120
49. #sets the pygame animation clock up
50. FPSClock = pygame.time.Clock()
52. ##predefining colours for later
53. #the colours are defined in the following format:
54. #colour = (red(0-255), green(0-255), blue(0-255))
55. white = (255, 255, 255)
56. grey1 = (192, 192, 192)
57. grey2 = (128, 128, 128)
58. grey3 = (64, 64, 64)
59. grey4 = (32, 32, 32)
60. black = (0, 0, 0)
61. blue = (0, 0, 255)
62. red = (255, 0, 0)
63. green = (0, 255, 0)
64. green2 = (0, 150, 0)
65. yellow = (255, 255, 0)
66. cyan = (0, 255, 255)
67. magenta = (255, 0, 255)
68. orange = (255, 128, 64)
70. #tells pygame to get the font system for making text ready
71. pygame.font.init()
73. #these are the fonts I have defined. The first parameter "consolas" is the text style that I have chosen
74. #the second parameter, an integer, is the font size
75. ButtonFont = pygame.font.SysFont("Consolas", 15)
76. PanelFont = pygame.font.SysFont("Consolas", 16)
77. SwitchFont = pygame.font.SysFont("Consolas", 14)
78. InputBoxFont = pygame.font.SysFont("Consolas", 13)
79. ObjectFont = pygame.font.SysFont("Consolas", 12)
80. LearnFont = pygame.font.SysFont("Consolas", 24)
82. ###the class that defines the button - a part of the user interface that allows the user to select the button by clicking.
83. ###When it is being clicked, the corresponding value is true, when it is not clicked it is false
84. **class** Button:
85. ##Initialisation method
86. #the self parameter means that variable with self. infront is unique to an individual button. The other parameters are the ones that we input to create as many buttons as we want later
87. **def** \_\_init\_\_(self, name, colour, posX, posY, sizeX, sizeY):
88. #this defines the name of the button tha will be displayed
89. self.name = name
90. #this defines the default colour that the button will be
91. self.colour = colour
92. #this defines the position in the x axis the button will be on the screen
93. self.posX = posX
94. #this defines the position in the y axis the button will be on the screen
95. self.posY = posY
96. #this defines the width of the button
97. self.sizeX = sizeX
98. #this defines the height of the button
99. self.sizeY = sizeY
101. ##method to display the button
102. #The newColour will change the colour of the button we use this later to indicate to the user if the mouse is on the button.
103. #self references all the variables unique to the object tha we defined earlier
104. **def** displayButton(self, newColour):
105. #draws the main rectangular part of the button to the display based on the values we initialised it with and the updated values
106. pygame.draw.rect(display, newColour, ((self.posX, self.posY), (self.sizeX, self.sizeY)))
107. #defines the text and the colour of the text to be put on the button
108. ButtonText = ButtonFont.render(self.name, 1, black)
109. #adjusting the position of the text on the button depending on the buttons length
110. adjustBy = len(self.name)
111. #draws the text to the display so that it is on the button in the correct place
112. display.blit(ButtonText, ((0.5\*self.sizeX)+self.posX-(4\*adjustBy), (0.5\*self.sizeY)+self.posY-10))
114. ##method to detect if the mouse if over the button and if it has been clicked
115. #clicked a variable asking whether or not the left mouse button is down at this instant
116. #X is the position in the x axis on the screen of the mouse
117. #X is the position in the y axis on the screen of the mouse
118. #self references all the variables unique to the object tha we defined earlier
119. **def** buttonClicked(self, clicked, X, Y):
120. #asks if the mouse is within the reigon of the button and the mouse button is down
121. **if** (X >= self.posX) **and** (X <= (self.posX+self.sizeX)) **and** (Y >= self.posY) **and** (Y <= (self.posY+self.sizeY)) **and** clicked:
122. #button is clicked so make the display colour white and set button clicked to true
123. **return** white, True
124. #asks if the mouse is within the reigon of the button and the mouse button is up (since if it was down it would have gone through the if statement)
125. **elif** (X >= self.posX) **and** (X <= (self.posX+self.sizeX)) **and** (Y >= self.posY) **and** (Y <= (self.posY+self.sizeY)):
126. #button is not clicked but the mouse is on the button so still display white but  set button clicked to false
127. **return** white, False
128. #if none of the previous conditions are then the mouse must not be on the button. whether or not the button is clicked is irrelevant
129. **else**:
130. #don't change the button colour and set button clicked to false
131. **return** self.colour, False
133. ###the class that defines the panel
134. ###panels create different sections on the display so the user can easily know what they are interacting with.
135. **class** Panel:
136. ##initialisation method
137. #self creates variables unique to each panel. The other parameters are assigned values when we create the individual panels.
138. **def** \_\_init\_\_(self, type, colour, posX, posY, sizeX, sizeY):
139. #defines the name given at the top of the panel
140. self.type = type
141. #defines the colour of the panel
142. self.colour = colour
143. #defines the x position of the panel
144. self.posX = posX
145. #defines the y position of the panel
146. self.posY = posY
147. #defines the width of the panel
148. self.sizeX = sizeX
149. #defines the height of the panel
150. self.sizeY = sizeY
152. #method to display panel
153. #self references the variables that are unique to each panel
154. **def** displayPanel(self):
155. #draws the rectangle with with the colours, positions and sizes we initialised them with
156. pygame.draw.rect(display, self.colour, ((self.posX, self.posY), (self.sizeX, self.sizeY)))
157. #defines the text that will be displayed and its colour.
158. PanelText = PanelFont.render(self.type, 1, grey1)
159. #displays the text at the top of the panel
160. display.blit(PanelText, (self.posX+4, self.posY))
162. ###the class that defines the switch - a part of the user interface that allows the user to toggle things in the program.
163. ###this is done by clicking the switch
164. **class** Switch:
165. ##initialisation method
166. #self creates variables unique to each switch. The other parameters are assigned values when we create the individual switches.
167. #the switch has default sizes of x=46 and y=24 but can be changed before initialisation
168. **def** \_\_init\_\_(self, name, colour, posX, posY, state, On, Off, sizeX = 46, sizeY = 24):
169. #defines the name of the switch
170. self.name = name
171. #defines the default colour of the switch
172. self.colour = colour
173. #defines the x position of the switch
174. self.posX = posX
175. #defines the y position of the switch
176. self.posY = posY
177. #defines the initial state of the switch
178. self.state = state
179. #defines the name of the on position
180. self.On = On
181. #defines the name of the off position
182. self.Off = Off
183. #defines the width of the switch
184. self.sizeX = sizeX
185. #defines the height of the switch
186. self.sizeY = sizeY
187. #defines the boolean value of it's clicked state last frame
188. self.wasClicked = False
190. ##method to display the switch
191. #self references the variable that are unique to each individual switch
192. **def** displaySwitch(self):
193. #draws the rectangle that will be displayed behind the switch
194. pygame.draw.rect(display, grey3, (self.posX-2, self.posY-2, self.sizeX, self.sizeY))
195. #asks if the state is false
196. **if** **not** self.state:
197. #since the switch is off draw the rectangle in the default colour and the off position
198. pygame.draw.rect(display, self.colour, (self.posX, self.posY, (self.sizeX/2)-4, (self.sizeY)-4))
199. #defines the text to be the name of the off position
200. SwitchText1 = ButtonFont.render((self.Off), 1, grey1)
201. #displays the text of the off position
202. display.blit(SwitchText1, (self.posX+(9\*len(self.name)), self.posY-25))
203. **else**:
204. #since the switch is on draw the retangle as white and in the on position
205. pygame.draw.rect(display, white, (self.posX+self.sizeX/2, self.posY, (self.sizeX/2)-4, (self.sizeY)-4))
206. #defines the text to be the name of the on position
207. SwitchText1 = ButtonFont.render((self.On), 1, grey1)
208. #displays the text of the on position
209. display.blit(SwitchText1, (self.posX+(9\*len(self.name)), self.posY-25))
210. #creates the text for the name of the switch
211. SwitchText2 = ButtonFont.render((self.name + ":"), 1, grey1)
212. #displays the name text
213. display.blit(SwitchText2, (self.posX, self.posY-25))
215. #creates an initial value for the number of metric meters in every pixel on the simulation
216. metersPerPixel = 500000
217. #creates an initial value for the number of simulation seconds that pass every frame 10000 simulation seconds every second is (10000/fps) simulation seconds every frame
218. Timeinterval = 10000/FPS
219. #defines the x position of the center of the universe created. This is the point around which you will be able to zoom in the x axis
220. centerX = width/3
221. #defines the y position of the center of the universe created. This is the point around which you will be able to zoom in the y axis
222. centerY = height/3
224. ###the class that defines the objects in the simulation - these are the planets that the user will put into the simulation
225. **class** Object:
226. ##initialisation method
227. #self allows each object to have it's own unique variables. The other parameters are defined later when the objects are added in by the user
228. **def** \_\_init\_\_(self, ID, colour, radius, posX, posY, velX, velY, density):
229. #defines the ID of each object - used to tell the difference between objects
230. self.ID = ID
231. #defines the colour of the object
232. self.colour = colour
233. #defines the radius of the circle that will be drawn onto the screen
234. self.drawRadius = radius
235. #defines the x position that the circle will be drawn onto
236. self.drawPosX = posX
237. #defines the y position that the circle will be drawn onto
238. self.drawPosY = posY
239. #defines the x velocity that the object will have
240. self.velX = velX
241. #defines the y velocity that the object will have
242. self.velY = velY
243. #defines the net force in the x direction. The force acting on the object will change as it interacts by gravity with other objects
244. self.netX = 0
245. #defines the net force in the y direction. The force acting on the object will change as it interacts by gravity with other objects
246. self.netY = 0
247. #defines the density of the object (this is mass/volume)
248. self.density = density
249. #defines the x position in meters used for calculations later
250. self.metricPosX = metersPerPixel\*self.drawPosX
251. #defines the y position in meters used for calculations later
252. self.metricPosY = metersPerPixel\*self.drawPosY
253. #defines the radius of the object in meters used for calculations later
254. self.metricRadius = metersPerPixel\*self.drawRadius
255. #definest the mass of the object in kilograms used for calculations later
256. self.metricMass = math.pi\*(4/3)\*(self.metricRadius\*\*3)\*self.density
257. #sets the list holding previous points the object has been to be empty
258. self.trail = []
259. #sets the net force to 0
260. self.netForce = 0
262. ##display method
263. #self references all variables unique to each object
264. **def** displayObject(self):
265. #draws a circe based on the draw radius, draw positions and colour
266. pygame.draw.circle(display, self.colour, (round(self.drawPosX+centerX), round(self.drawPosY+centerY)), round(self.drawRadius))
268. ##method to display the ID of teh object on the object
269. **def** showID(self):
270. #creates the text of the ID that will be displayed on top of the objects
271. ObjectText = ObjectFont.render(str(self.ID), 1, red)
272. #displays the ID in the position of the object
273. display.blit(ObjectText, (self.drawPosX-5+centerX, self.drawPosY-5+centerY))
275. ##method that updates the units that were defined earlier
276. **def** updateUnits(self):
277. #changes the metric radius based on the mass and density - this changes when objects collide
278. self.metricRadius = ((3\*self.metricMass)/(4\*math.pi\*self.density))\*\*(1/3)
279. #updates the draw radius to match the metric radius
280. self.drawRadius = self.metricRadius/metersPerPixel
281. #updates the x draw position to match the metric x position - this changes when objects have a non-zero x velocity
282. self.drawPosX = self.metricPosX/metersPerPixel
283. #updates the y draw position to match the metric y position - this changes when objects have a non-zero y velocity
284. self.drawPosY = self.metricPosY/metersPerPixel
286. ##method that
287. **def** changePos(self):
288. #changes the metric X position based on it's x velocity and the time between frames
289. self.metricPosX += (self.velX\*Timeinterval)
290. #changes the metric y position based on it's y velocity and the time between frames
291. self.metricPosY += (self.velY\*Timeinterval)
292. #updates the x draw position to match the metric x position - this changes when objects have a non-zero x velocity
293. self.drawPosX = self.metricPosX/metersPerPixel
294. #updates the y draw position to match the metric y position - this changes when objects have a non-zero y velocity
295. self.drawPosY = self.metricPosY/metersPerPixel



300. ###function to call for exiting the program
301. #takes the parameter exit (true is if they do want to exit, false is if they don't)
302. **def** exiting(exit):
303. #asks if the user wants to exit
304. **if** exit:
305. #closes pygame
306. pygame.quit()
307. #closes python
308. sys.exit()
310. ###GUIs - these are being defined by creating objects with the classes made earlier
312. ##panels
313. #tool panel - will have switches and a colour chooser for customising the simulation
314. pTools = Panel("Tools", grey2, 5, 34, int(width/8), int(height/2))
315. #colour wheel - not a panel but is an image that sits on top of the tool bar panel
316. colourWheel = pygame.transform.scale(colourWheel, (int(pTools.sizeX\*(3/4)),int(pTools.sizeX\*(3/4))))
317. #lightness - not a panel but is an image that sits on top of the tool bar panel
318. lightness = pygame.transform.scale(lightness, ((int(pTools.sizeX/5)-10, int(pTools.sizeX\*(4/5)))))
319. #information panel - will have general information about the current simulation
320. pInfo = Panel("Info", grey2, 5, int(pTools.posY+pTools.sizeY+2), int(width/8), int(height/2)-46)
321. #simulation panel - will contain the objects and is the main visual part of the whole program
322. pSimulation = Panel("Simulation", black, int(pTools.posX+pTools.sizeX+2), 34, int(width/2), 4\*height/5)
323. #time panel - switches and buttons to pause/play, speed up, slow down and reset the simulation
324. pTime = Panel("Time", grey2, int(pTools.posX+pTools.sizeX+2), int(pSimulation.posY+pSimulation.sizeY+2), int(width/2), (height-int(pSimulation.posY+pSimulation.sizeY+12)))
326. ##these define the boundary that the graph panel will be in
327. topGraph = 34
328. leftGraph = int(((5\*width)/8)+9)
329. rightGraph = int(((5\*width)/8)+9)+int(((3\*width)/8)-12)
330. bottomGraph = int(height/2+34)
332. #graph panel - will contain the graphing feature
333. pGraph = Panel("Graph 1", white, leftGraph, topGraph, rightGraph-leftGraph, bottomGraph-topGraph)
334. #exit panel - the panel that only shows up when the user wants to exit the simulation -also contains simulation saving features
335. pExit = Panel("Exit", grey4, int((width/2)-300), int((height/2)-100), 600, 300)
336. #learn panel - shows the equations of motion, gravity and keplers second law
337. pLearn = Panel("Learn", grey2, int(((5\*width)/8)+9), (height/2)+36, int(((3\*width)/8)-12), (height/2)-46)
338. #gravitation formula - not a panel but is one of the images that will be displayed on the learn panel
339. gravitationFormula = pygame.transform.scale(gravitationFormula, (int(((3\*width)/8)-112), int((height/2)-146)))
340. #suvat formula = not a panel but is one of the images that will be displayed on the learn panel
341. suvatFormula = pygame.transform.scale(suvatFormula, (int(((3\*width)/8)-112), int((height/2)-146)))
342. #instruction panel - the panel that will contain instructions on how to use the program
343. pInstructions = Panel("instructions", white, int(pTools.posX+pTools.sizeX+2), 34, int(width/2), 4\*height/5)
344. #the panel that will appear when the user wants to change the x axis of the graph
345. pChangeX = Panel("Change x axis", grey2, leftGraph, topGraph, rightGraph-leftGraph, bottomGraph-topGraph)
346. #the panel that will appear when the user wants to change the Y axis of the graph
347. pChangeY = Panel("Change y axis", grey2, leftGraph, topGraph, rightGraph-leftGraph, bottomGraph-topGraph)
349. ##window and top bar buttons
350. #the button for exiting the program
351. bExit = Button("x", red, width-52, 1, 50, 30)
352. #the button th minimise the program
353. bMin = Button("\_", grey1, width-103, 1, 50, 30)
354. #the button to enter a new simulation
355. bEnter = Button("Enter", grey2, (width/2)-150, (height/2)-40, 100, 80)
356. #the button to open a saved simulaton
357. bOpen = Button("Open", grey2, width/2, (height/2)-40, 100, 80)
358. #the button to go back to main menu
359. bMainMenu = Button("Exit", grey2, 36, 3, 50, 26)
360. #the button to display the first set of instructions
361. bInstructions = Button("Instructions", grey2, 100, 3, 120, 26)
362. #the button to display the second set of instructions
363. bInstructions2 = Button("more Instructions", grey2, 235, 3, 170, 26)
364. #the button that resets the simulation
366. ##time buttons
367. bReset = Button("reset", red, pTime.posX+95, pTime.posY+48, 86, 44)
368. #the button that speeds up the flow of time
369. bIncrease = Button(">>+", cyan, bReset.posX+bReset.sizeX+5, pTime.posY+48, 86, 44)
370. #the button that slows down the flow of time
371. bDecrease = Button(">>-", cyan, bIncrease.posX+bIncrease.sizeX+5, pTime.posY+48, 86, 44)
372. ##exiting buttons
374. #the button that exits and saves the current simulation
375. bSave = Button("save", green, int((width/2)-250), int((height/2)+100), 100, 50)
376. #the button that exits and does not save the current simulation
377. bDontSave = Button("Dont save", red, int((width/2)-50), int((height/2)+100), 100, 50)
378. #the button that cancles the exit of the simulation
379. bCancel = Button("Cancel", grey3, int((width/2)+150), int((height/2)+100), 100, 50)
381. ##learn panel buttons
382. #the button that displays the formula for gravity in the learn panel
383. bGravity = Button("Gravity", green2, pLearn.posX+110, pLearn.posY+18, 100, 30)
384. #the button that displays the formulas of motion in the learn panel
385. bSuvat = Button("Motion", green2, pLearn.posX+5, pLearn.posY+18, 100, 30)
386. #the button that starts the demonstration of keplers second law
387. bKepler = Button("Keplers 2nd law", green2, pLearn.posX+215, pLearn.posY+18, 160, 30)
388. #the button that lets you change the x axis of the graph
390. ##graphing buttons
391. bX = Button("x", green, pGraph.posX+(pGraph.sizeX/2)-20, pGraph.posY+pGraph.sizeY-25, 40, 20)
392. #the button that lets you change the y axis of the graph
393. bY = Button("y", green, pGraph.posX+5, pGraph.posY+(pGraph.sizeY/2)-20, 20, 40)
394. #VX - not a button but is the text that will be displayed if any of the axes are the x velocity of an object
395. VX = "Velocity X (m/s)"
396. #the button that changes an axis to be the x velocity of an object
397. bXvel = Button(VX, yellow, pGraph.posX+30, pGraph.posY+30, 200, 50)
398. #VX - not a button but is the text that will be displayed if any of the axes are the y velocity of an object
399. VY = "Velocity Y (m/s)"
400. #the button that changes an axis to be the y velocity of an object
401. bYvel = Button(VY, yellow, pGraph.posX+30, pGraph.posY+90, 200, 50)
402. #T - not a button but is the text that will be displayed if any of the axes are Time
403. T = "Time (s)"
404. #the button that changes an axis to time
405. bTime = Button(T, yellow, pGraph.posX+30, pGraph.posY+150, 200, 50)
406. #F - not a button but is the text that will be disp[layed if any of the axes are force
407. F = "Net Force (10^24 N)"
408. #the button that changes an axis to net force
409. bForce = Button(F, yellow, pGraph.posX+30, pGraph.posY+210, 200, 50)
411. #the button that starts the graphing
412. bStartGraph = Button("Start Graphing!", red, pGraph.posX+(pGraph.sizeX/2+120), pGraph.posY+50, 160, 30)
413. #the button that stops the graphing
414. bStopGraph = Button("Stop Graphing!", red, pGraph.posX+80, pGraph.posY+10, 160, 30)
416. ##switches
417. #switch to pause and play
418. sPausePlay = Switch("||/>", green, pTime.posX+5, pTime.posY+50, False, "||", " >", 86, 44)
419. #switch to shoiw the IDs of the objects
420. sShowID = Switch("Show ID", yellow,  pTools.posX+5, pTools.posY+50, False, "on", "off")
421. #switch to show the velocities of the objects
422. sShowVel = Switch("Show velocity", yellow, pTools.posX+5, pTools.posY+100, False, "on", "off")
423. #switch to show the net forces acting on the objects
424. sShowFor = Switch("Show net Force", yellow, pTools.posX+5, pTools.posY+150, False, "on", "off")
425. #switch to show the previous path of the objects
426. sShowTrail = Switch("Show trail", yellow, pTools.posX+5, pTools.posY+200, False, "on", "off")
427. #the list holding the switches
428. switches = [sPausePlay, sShowID, sShowVel, sShowFor, sShowTrail]




434. ##assembling the Window GUI
435. **def** WindowGUI(clicked, X, Y, clickedLastFrame):
436. ##the rectangle and 3 lines draws the outline of the program that will have the window buttons on
437. pygame.draw.rect(display, black, (0,0,width,32))
438. pygame.draw.line(display, black, (0,0), (0, height), 2)
439. pygame.draw.line(display, black, (width,0), (width, height), 2)
440. pygame.draw.line(display, black, (0,height), (width, height), 18)
441. #displays the logo in the top right of the screen
442. pygame.Surface.blit(display, icon, (0,0))
443. #asks if the mouse button was not clicked in the last fram
444. **if** **not** clickedLastFrame:
445. #runs the method fo detectinmg button clicks on the exit button
446. colour1, bExitClicked = bExit.buttonClicked(clicked, X, Y)
447. #runs the method fo detectinng button clicks on the minimise button
448. colour2, bMinClicked = bMin.buttonClicked(clicked, X, Y)
449. #runs the method for displaying the minimise button
450. bMin.displayButton(colour2)
451. #runsa the method for displaying the exit button
452. bExit.displayButton(colour1)
453. #runs the exit procedure z
454. exiting(bExitClicked)
455. #asks if the minimise button was clicked
456. **if** bMinClicked:
457. #minimises the program
458. pygame.display.iconify()
459. **else**:
460. #displays the button as grey
461. bMin.displayButton(grey1)
462. #displays the button as red
463. bExit.displayButton(red)
465. ###function for drawing the background of the display
466. **def** background():
467. ##leaves a space where the simulation will be displayed so that the objects that move past the edge will go under the background
468. pygame.draw.rect(display, grey1, (0, 0, width, pSimulation.posY))
469. pygame.draw.rect(display, grey1, (0, 0, pSimulation.posX, height))
470. pygame.draw.rect(display, grey1, (0, pSimulation.posY+pSimulation.sizeY, width, height))
471. pygame.draw.rect(display, grey1, (pSimulation.posX+pSimulation.sizeX, 0, width, height))
473. #variable that tells us if we are in the main menu
474. inMainMenu = True
475. #variable that tells us if the left mouse button is clicked
476. clicked = False
478. ###program loop
479. **while** True:
480. ##getting ready for main menu
482. #list - holds all of the planets inside of it
483. objects = []
484. #variable - tells us if the simulation is pasued
485. paused = False
487. #variable - the mouse button state in the last frame
488. ClickedLastFrame = False
489. #variable - the mouse button state in this frame
490. clicked = False
492. #variable - the number of planets in the simulation
493. number = 0
494. #variable - the number of frames the decrease flow of time button was held for
495. heldD=0
496. #variale - the number of frames the increase flow of time button was held for
497. heldI=0
499. #variable - is the user trying to open a file
500. opening = False
501. #list - contains the file names of the simulations that have been saved
502. fileNames = []
503. #list - contains the buttons that allow you to select the file to open
504. buttons = []
505. #list - contains the data of the simulation that the user is trying to open
506. data = []
507. #updates filenames to include all files in the saves folder
508. fileNames = os.listdir("saves\\")
509. #variable - is the scroll bar clicked on
510. activated = False
512. #the ratio between RGB values for the colour picker
513. currentHUE = [0.7, 0.7, 0.7]
514. #the multiplier that changes the RGB ratios to an actual colour value
515. reduce = 255
517. ##loop that creates buttons based on the files in filenames
518. #enumerate keeps hold of the number of loops in count and the current value from fileNames in filename
519. **for** count, filename **in** enumerate(fileNames):
520. buttons.append(Button(filename, grey2, (width/2)-350, (count\*60)+40, 300, 50))
522. ### main menu animation loop
523. **while** inMainMenu:
524. #fills the background with grey
525. display.fill(grey1)
526. #gets x coordinate of mouse
527. X = list(pygame.mouse.get\_pos())[0]
528. #gets y coordinate of mouse
529. Y = list(pygame.mouse.get\_pos())[1]
530. ##event handling
531. #iterates through all possible events
532. **for** event **in** pygame.event.get():
533. #asks if the event is quit
534. **if** event.type == quit:
535. #quits the program
536. exiting(True)
537. #asks if event is a key press
538. **if** event.type == pygame.KEYDOWN:
539. #asks if the key is escape
540. **if** event.key == pygame.K\_ESCAPE:
541. #quits the program
542. exiting(True)
543. #updates clicked to be the left mouse button state
544. clicked = list(pygame.mouse.get\_pressed())[0]
545. #displays the boarder of the window
546. WindowGUI(clicked, X, Y, ClickedLastFrame)
548. ##open or enter?
549. #asks if the use does not want to open a saved simulation
550. **if** **not** opening:
551. #runs method to detect button clicks on enter button
552. colour3, bEnterClicked = bEnter.buttonClicked(clicked, X, Y)
553. #runs method for displaying the enter button
554. bEnter.displayButton(colour3)
555. #asks if the enter button is clicked
556. **if** bEnterClicked:
557. #changes inMainMenu to false since the user has clicked to enter the simulation
558. inMainMenu = False
560. #runs method to detect button clicks on open button
561. colour11, bOpenClicked = bOpen.buttonClicked(clicked, X, Y)
562. #runs method to display the open button
563. bOpen.displayButton(colour11)
564. #asks if the open button is clicked
565. **if** bOpenClicked:
566. #changes opening to true so that we can select a file to open
567. opening = True


571. **else**:
572. #runs method to detect button clicks on the back button
573. colour27, bMainMenuClicked = bMainMenu.buttonClicked(clicked, X, Y)
574. #runs method to display the back button
575. bMainMenu.displayButton(colour27)
576. #asks if back button clicked
577. **if** bMainMenuClicked:
578. opening = False
580. #iterates through buttons - the list that holds each file's button
581. **for** item **in** buttons:
582. #runs the method that detects button clicks on the current button in the loop
583. colour12, buttonClicked = item.buttonClicked(clicked, X, Y)
584. #runs the method for displaying the bcurrent button in the loop
585. item.displayButton(colour12)
586. #asks if that button was clicked
587. **if** buttonClicked:
588. #opens the file corresponding to tha button that was clicked
589. file = open("saves\\" + item.name, "r")
590. #eval turns a string in the format of an list into an actual list and this is assigned to data
591. data = eval(file.readline())
592. #changes inMainMenu to false since the user has clicked to open a simulation
593. inMainMenu = False
595. # this updates the screen so anything drawn to the display is actually shown on the users monitor
596. pygame.display.update()
597. # waits for 1/FPS seconds
598. FPSClock.tick(FPS)
600. #waits for half a second
601. time.sleep(0.5)
603. ##loop adds the data from the file opened (if there is any) to the objects list
604. #iterates through data
605. **for** item **in** data:
606. #asks if the planets ID is greater than the number of objects
607. **if** item[0] > number:
608. #sets number equal to the ID of that object +1
609. number = item[0]+1
610. #calculates the metric radius from the
611. metricradius = (((3\*item[1])/(4\*math.pi\*item[6]))\*\*(1/3))
612. #calculates the radius of which to draw the circle
613. Radius = metricradius/metersPerPixel
614. #adds the planet to objects
615. objects.append(Object(item[0], grey1, Radius, (item[2]/metersPerPixel), (item[3]/metersPerPixel), item[4], item[5], item[6]))
617. ##setting up simulation
619. #the density of any new object that is added in
620. newDensity = 5500
621. #the radius of any new object that is added in
622. newRadius = 10
623. #currently zooming is set to false
624. zoom = False
625. #currently changing density is set to false
626. changeDensity = False
627. #default graph x variable is set to [T, -1]. T is Time and -1 indicates that the varable chosen is not object specific
628. VarX = [T, -1]
629. #default graph y variable is set to [VX 0]. VX is X velocity and 0 indicates that the vaiable chosen will be based on object 0
630. VarY = [VX, 0]
631. #currently changing X axis is set to False
632. changeXaxis = False
633. #current;y chaning Y axis is set to False
634. changeYaxis = False
635. #Started the graph is set to False
636. started = False
637. #the metric X coordinates for the graph will be stored in this list
638. XMetricCoords = []
639. #the metric Y coordinates for the graph will be stored in this list
640. YMetricCoords = []
641. #currently running keplers second law demo is set to False
642. runKepler = False
643. #graph is stopping is set to False
644. stop = False
645. #Objects required for the x axis of the graph is set to false
646. objectXExists=False
647. #Objects required for the Y axis of the graph is set to false
648. objectYExists=False
650. ###running the simulation
651. **while** **not** inMainMenu:
652. ##inputs
653. #gets x coordinate of mouse
654. X = list(pygame.mouse.get\_pos())[0]#gets x coordinate of mouse
655. #gets Y coordinate of mouse
656. Y = list(pygame.mouse.get\_pos())[1]#gets y coordinate of mouse
657. ##event handling
658. #iterates through all possible events
659. **for** event **in** pygame.event.get():
660. #asks if the event is a key press
661. **if** event.type == pygame.KEYDOWN:
662. #asks if the key pressed is escape
663. **if** event.key == pygame.K\_ESCAPE:
664. #quits the program
665. exiting(True)
666. #asks ifthe key that was pressed is the space bar
667. **if** event.key == pygame.K\_SPACE:
668. #toggles the pause variable
669. paused = **not** paused
670. #toggles the pause switch
671. sPausePlay.state = **not** sPausePlay.state
672. #asks if the event is a mouse button action
673. **elif** (event.type == pygame.MOUSEBUTTONDOWN):
674. #asks if mouse wheel is scrolling down
675. **if** event.button == 5:
676. #asks if we are not currently changing the density
677. **if** **not** changeDensity:
678. #asks if we are currently zooming
679. **if** zoom:
680. #reduces metersPerPixel by a factor of 1.05
681. metersPerPixel /= 1.05
682. #asks if newRadius is less than or equal to 150 so that the input circle is not too big
683. **if** newRadius <= 150:
684. #increases newRadius by a factor of 1.05
685. newRadius \*= 1.05
686. **else**:
687. #increases new density by a factor of 1.05
688. newDensity \*= 1.05
689. #asks if mouse wheel is scrolling up
690. **if** event.button == 4:
691. #asks if we are not currently changing density
692. **if** **not** changeDensity:
693. #asks if we are currently zooming
694. **if** zoom:
695. #increases metersPerPixel by a factor of 1.05
696. metersPerPixel \*= 1.05
697. #asks if newRadius is less than 3 so the input circle is not too small
698. **if** newRadius >= 3:
699. #reduces newRadius by a factor of 1.05
700. newRadius /= 1.05
701. **else**:
702. #reduces newDensity by a factor of 1.05
703. newDensity /= 1.05
705. #asks if the event is quit
706. **elif** event.type == quit:
707. #exits the program
708. exiting(True)
710. #makes a list of all keys that are presses
711. keys = pygame.key.get\_pressed()
712. #asks if the left shift key is pressed and the z key is pressed
713. **if** keys[pygame.K\_LSHIFT] **and** keys[pygame.K\_z]:
714. #sets zoom to true
715. zoom = True
716. #sets run kepler to false
717. runKepler = False
718. #sets show trail to false
719. sShowTrail.state = False
721. #sets change density to false
722. changeDensity = False
723. #sets the center x to the mouse x position
724. centerX = X
725. #sets the center y to the mouse y position
726. centerY = Y
727. #asks if the left shift key is pressed and the d key is pressed
728. **elif**  keys[pygame.K\_LSHIFT] **and** keys[pygame.K\_d]:
729. #sets change density to True
730. changeDensity = True
731. #sets zoom to False
732. zoom = False
733. **else**:
734. #sets zoom to false
735. zoom = False
736. #sets change density to False
737. changeDensity = False
738. #updates clicked to be the left mouse button state
739. clicked = list(pygame.mouse.get\_pressed())[0]
741. #runs the method that displays the simulation panel
742. pSimulation.displayPanel()
744. ##next 4 lines defines the reigon that objects can be added into
745. leftEdge = pSimulation.posX
746. topEdge = pSimulation.posY
747. rightEdge = pSimulation.posX+pSimulation.sizeX
748. lowerEdge = pSimulation.posY+pSimulation.sizeY
750. #asks if the simulation is not paused
751. **if** **not** paused:
752. #iterates through objects - the list of planets
753. **for** self **in** objects:
754. self.netForce = 0
755. #iterates through objects - a nested loop
756. **for** other **in** objects:
757. #asks if other object does not equal self object
758. **if** other != self:
759. ##calculating measures of the objects motion
760. #calculating the momentum in the x direction - used for collisions later
761. momentumX = self.velX\*self.metricMass + other.velX\*other.metricMass
762. #calculating the momentum in the y direction - used for collisions later
763. momentumY = self.velY\*self.metricMass + other.velY\*other.metricMass
765. #x is the distance in the x direction between the two objects
766. x = self.metricPosX-other.metricPosX
767. #y is the distance in the y direction between the two objects
768. y = self.metricPosY-other.metricPosY
769. #here I use pythagoras theorem to find the magnitude of the distance between the centers of the two objects
770. distance = math.sqrt((x\*\*2)+(y\*\*2))
772. #This equation finds the force acting on self object through gravity
773. #although it is equal to the force acting on other object, it is calculated when the other object is calculating it's forces when other becomes self (in the outer for loop)
774. force = 6.67\*(10\*\*(-11))\*(self.metricMass\*other.metricMass)/(distance\*\*2)
775. #adds the force to the net force attribute
776. self.netForce += force
777. #calculating acceleration of the object from the force and the mass of the object
778. acc = force/self.metricMass
780. #calculating the force in the X direction - used for showing the force on the simulation later
781. self.netX = (x/distance \* force)/(10\*\*22)
782. #we divide by 10 to the power of 22 because the number wouyld be so big that it would crash if we tried to draw a line of that size - also it wouldn't fit on the screen
783. #calculating the force in the X direction - used for showing the force on the simulation later
784. self.netY = (y/distance \* force)/(10\*\*22)
786. #calculating the x velocity after the acceleration has taken place
787. self.velX -= acc \* (x/distance) \* Timeinterval
788. #calculating the Y velocity after the acceleration has taken place
789. self.velY -= acc \* (y/distance) \* Timeinterval
791. ##detecting collisions
792. #asks if the distance between the two objects is les than the sum of the radii of the two objects - if they are touching or closer
793. **if** distance <= (self.metricRadius + other.metricRadius):
794. ##calculating the density of the object after the collision
795. #the total mass is calculated
796. totalMass = self.metricMass+other.metricMass
797. #the percentage of the total that self object is, is calculated here
798. selfPercent = self.metricMass/totalMass
799. #the percentage of the total that other object is, is calculated here
800. otherPercent = other.metricMass/totalMass
801. #the density of the combined object is calculated here
802. combinedDensity = ((selfPercent\*self.density)+(otherPercent\*other.density))
804. #asks if the self radius is greater than the other radius
805. **if** self.metricRadius > other.metricRadius:
806. #adds other's mass onto self
807. self.metricMass += other.metricMass
808. **try**:
809. #removes other from objects
810. objects.remove(other)
811. **except**:
812. **pass**
813. #assigns the combined density to self
814. self.density = combinedDensity
815. #calculates the x velocity of the object after a collision
816. #since momentum of two objects in a closed system remains constant, we use the momentum that was calculated earlier
817. self.velX = momentumX/self.metricMass
818. #calculates the y velocity of the object after a collision
819. self.velY = momentumY/self.metricMass
821. **else**:
822. #adds self's mass onto other
823. other.metricMass += self.metricMass
824. **try**:
825. #removes self from the list of objects
826. objects.remove(self)
827. **except**:
828. **pass**
829. #assigns the combined density to other
830. other.density = combinedDensity
831. #calculates the x velocity of the object after a collision
832. #since momentum of two objects in a closed system remains constant, we use the momentum that was calculated earlier
833. other.velX = momentumX/other.metricMass
834. #calculates the y velocity of the object after a collision
835. other.velY = momentumY/other.metricMass
837. #calculating new positions of the objects
838. #iterates through objects
839. **for** self **in** objects:
840. #asks if the simulation is paused
841. **if** **not** paused:
842. #as it is paused, don't move any objects
843. self.changePos()
844. #update any units that need to be updated
845. self.updateUnits()
846. #asks if we are not currently zooming
847. **if** **not** zoom:
848. #asks if the switch for showing the trail of each object is on
849. **if** sShowTrail.state:
850. #adds the position of each object to the
851. self.trail.append((self.drawPosX+centerX, self.drawPosY+centerY))
852. #asks if the length of the trail is greater than 1000
853. **if** len(self.trail)>1000:
854. #deletes the point at the oldest end of the trail
855. **del** self.trail[0]
856. #asks if the length of the trail is greater than 1
857. **if** len(self.trail)>1:
858. #draws the trail
859. pygame.draw.lines(display, self.colour, False, self.trail, 2)
860. **else**:
861. #resets the trail
862. self.trail = []
864. #display the objects
865. self.displayObject()
866. #asks if the switch for showing the velocity of the objects is on
867. **if** sShowVel.state:
868. #draws a line from self object to a point in space - the line will point in the direction of the velocity and the length is always in proportion to the velocity of the object
869. pygame.draw.line(display, magenta, (int(self.drawPosX+centerX), int(self.drawPosY+centerY)), (int(self.drawPosX+(self.velX/100)+centerX), int(self.drawPosY+(self.velY/100)+centerY)))
870. #asks if the switch for showing the net force acting on each object is on
871. **if** sShowFor.state:
872. #draws a line from self object to a point in space - the line will point in the direction of the force and the length is always in proportion to the force acting on the object
873. pygame.draw.line(display, cyan, (self.drawPosX+centerX, self.drawPosY+centerY), ((int(self.drawPosX-self.netX)+centerX), int(self.drawPosY-self.netY)+centerY), 2)
874. #asks if the switch for showing the ID of each object is on
875. **if** sShowID.state:
876. #displays the ID of the object
877. self.showID()
878. #sets netX to 0
879. self.netX = 0
880. #sets netY to 0
881. self.netY = 0
883. ##this section is used for adding objects into the simulation
884. #asks if the cursor is within the simulation boundary
885. **if** (X-newRadius>leftEdge) **and** (X+newRadius<rightEdge) **and** (Y-newRadius>topEdge) **and** (Y+newRadius<lowerEdge):
886. #draws a blue circle around the cursor - used to indicate the size and position of the object to be added in next
887. pygame.draw.circle(display, blue, (X,Y), int(newRadius), 2)
888. #asks if the mouse is clicked in this frame but also not clicked in the last frame
889. **if** clicked **and** **not** ClickedLastFrame:
890. #sets clicked last frame to true so that when next frame comes around we know that the mouse has been held
891. ClickedLastFrame = True
892. #sets initialY equal to the Y position of the mouse so we know where the planet is to be placed
893. initialY = Y
894. #sets initialX equal to the x position of the mouse
895. initialX = X
896. #asks if the mouse is clicking in this frame and in the last frame
897. **elif** clicked **and** ClickedLastFrame:
898. #sets clicked last frame to true so that when next frame comes around we know that the mouse has been held
899. ClickedLastFrame = True
900. #draws a circle where it will be positioned
901. pygame.draw.circle(display, currentcolour, (initialX, initialY), int(newRadius))
902. #draws a line from the mouse to the initial position to indicate the speed and direction of the object
903. #the direction is inverted so that the mouse and line don't cover where the planet will be travelling
904. pygame.draw.line(display, green, (initialX, initialY), (X, Y), 3)
905. #draws a yellow circle around the mouse
906. pygame.draw.circle(display, yellow, (X, Y), int(newRadius), 2)
907. #asks if the mouse has just been released
908. **elif** **not** clicked **and** ClickedLastFrame:
909. #sets clicked last frame to false
910. ClickedLastFrame = False
911. #adds the planet into the simulation by appending it to the list of planets
912. objects.append(Object(number, currentcolour, int(newRadius), initialX-centerX, initialY-centerY, (initialX-X)\*100, (initialY-Y)\*100, newDensity))
913. #increments the number of objects by 1
914. number += 1
915. #asks if not recent clicks have been made
916. **elif** **not** clicked **and** **not** ClickedLastFrame:
917. #sets clicked last frame to false
918. ClickedLastFrame = False
920. #asks if we want to run the keplers second law demo
921. **if** runKepler:
922. **try**:
923. #draws the area set out by the point list for the blue area
924. pygame.draw.polygon(display, blue, pointList1)
925. **except**:
926. **pass**
927. **try**:
928. #draws the area set out by the point list for the green area
929. pygame.draw.polygon(display, green, pointList2)
930. **except**:
931. **pass**
933. #draws the background to the non-simulation panels
934. #as stated before it is in front of the simulation panel so planets pass underneath it
935. background()
937. #displays the tool panel
938. pTools.displayPanel()
939. #displays the information panel
940. pInfo.displayPanel()
941. #displays the time panel
942. pTime.displayPanel()
943. #displays the graph panel
944. pGraph.displayPanel()
945. #displays the learn panel
946. pLearn.displayPanel()
947. ### Graphing
949. ##drawing the axes
951. #defines the origin position of the graph
952. originX = pGraph.posX+50
953. originY = pGraph.posY+pGraph.sizeY-50
955. #draws X axis
956. pygame.draw.line(display, black, (originX, pGraph.posY+50), (originX, originY), 2)
958. #draws button X
959. colour17, bXClicked = bX.buttonClicked(clicked, X, Y)
960. bX.displayButton(colour17)
962. #draws Y axis
963. pygame.draw.line(display, black, (originX, originY), (pGraph.posX+pGraph.sizeX-50, originY), 2)
965. #draws button Y
966. colour18, bYClicked = bY.buttonClicked(clicked, X, Y)
967. bY.displayButton(colour18)
969. ##this section shows what the x and y axes are representing
970. #this is used so that measures that dont belong to an object get the right label
971. **if** VarX[0] != "Time (s)":
972. #text telling the user what variables are being measured on the X axis
973. Xaxis = PanelFont.render("x axis = " + VarX[0] + " of object " + str(VarX[1]), 1, black)
974. #displaying that text
975. display.blit(Xaxis, (pGraph.posX+(pGraph.sizeX/2)-100, pGraph.posY+5))
977. **else**:
978. #text telling the user what variables are being measured on the X axis
979. Xaxis = PanelFont.render("x axis = " + VarX[0], 1, black)
980. #text telling the user what variables are being measured on the X axis
981. display.blit(Xaxis, (pGraph.posX+(pGraph.sizeX/2)-100, pGraph.posY+5))
983. **if** VarY[0] != "Time (s)":
984. #text telling the user what variables are being measured on the Y axis
985. Yaxis = PanelFont.render("y axis = " + VarY[0] + " of object " + str(VarY[1]), 1, black)
986. #text telling the user what variables are being measured on the Y axis
987. display.blit(Yaxis, (pGraph.posX+(pGraph.sizeX/2)-100, pGraph.posY+20))
989. **else**:
990. #text telling the user what variables are being measured on the Y axis
991. Yaxis = PanelFont.render("y axis = " + VarY[0], 1, black)
992. #text telling the user what variables are being measured on the Y axis
993. display.blit(Yaxis, (pGraph.posX+(pGraph.sizeX/2)-100, pGraph.posY+20))
995. #asks if the change x axis button is clicked
996. **if** bXClicked:
998. changeXaxis = True
999. #asks if the change Y axis button is clicked
1000. **if** bYClicked:
1001. #sets the changeYaxis variable to true
1002. changeYaxis = True
1004. #creates an empty list used later for storing the buttons with the IDs of the objects
1005. IDButtons = []
1007. #adjusts the position of the buttons
1008. adjust = 30
1009. **for** self **in** objects:
1010. IDButtons.append(Button(str(self.ID), orange,  pGraph.posX+250, pGraph.posY+adjust, 200, 20))
1011. adjust += 30
1013. size = (len(objects)\*(50))+30
1015. #does the user want to change the Y axis and is the graph not started
1016. **if** changeXaxis **and** **not** started:
1017. #display the change x panel
1018. pChangeX.displayPanel()
1020. #detect mouse click on the X velocity button
1021. colour19, bXvelClicked = bXvel.buttonClicked(clicked, X, Y)
1022. #display button
1023. bXvel.displayButton(colour19)
1025. #detect mouse click on the Y velocity button
1026. colour20, bYvelClicked = bYvel.buttonClicked(clicked, X, Y)
1027. #display button
1028. bYvel.displayButton(colour20)
1030. #detect mouse click on the time button
1031. colour21, bTimeClicked = bTime.buttonClicked(clicked, X, Y)
1032. #display button
1033. bTime.displayButton(colour21)
1035. #detect mouse click on the force button
1036. colour26, bForceClicked = bForce.buttonClicked(clicked, X, Y)
1037. #display button
1038. bForce.displayButton(colour26)
1040. #iterate through the buttons
1041. **for** n **in** IDButtons:
1042. #detect mouse click on n button
1043. colour23, IDButtonClicked = n.buttonClicked(clicked, X, Y)
1044. #is variable on x axis not time
1045. **if**(VarX[0] != T):
1046. #is the button within the graph panel
1047. **if** n.posY+n.sizeY < pGraph.posY+pGraph.sizeY:
1048. #display the button
1049. n.displayButton(colour23)
1050. #is the button clicked
1051. **if** IDButtonClicked:
1052. #set the object for the x axis to the name of the button
1053. VarX[1] = int(n.name)
1054. #stop changing the x axis
1055. changeXaxis = False
1056. **else**:
1057. #is the button within the graph panel
1058. **if** n.posY+n.sizeY< pGraph.posY+pGraph.sizeY:
1059. #display the button but blanked out
1060. n.displayButton(grey4)
1062. #is the button for the X velocity clicked
1063. **if** bXvelClicked:
1064. #set the type of variable for the x axis to the x velocity
1065. VarX[0] = VX
1066. #is the object of the x axis equal to -1
1067. **if** VarX[1] == -1:
1068. #set the object of the x axis to 0
1069. VarX[1] = 0
1070. #stop changing the x axis
1071. changeXaxis = False
1073. #is the button for the Y velocity clicked
1074. **if** bYvelClicked:
1075. #set the type of variable for the x axis to the Y velocity
1076. VarX[0] = VY
1077. #is the object of the x axis -1
1078. **if** VarX[1] == -1:
1079. #set the object of the x axis to 0
1080. VarX[1] = 0
1081. #stop chaning the x axis
1082. changeXaxis = False
1084. #is the button for time clicked
1085. **if** bTimeClicked:
1086. #set the x axis variable to time
1087. VarX[0] = T
1088. #set the x axis object to -1
1089. VarX[1] = -1
1090. #stop changing the x axis
1091. changeXaxis = False
1093. #is the button for Force clicked
1094. **if** bForceClicked:
1095. #set the x axis variable to net force
1096. VarX[0] = F
1097. #is the object of the x axis -1
1098. **if** VarX[1] == -1:
1099. #set object of the x axis to 0
1100. VarX[1] = 0
1101. changeXaxis = False

1104. #does the user want to change the Y axis and is the graph not started
1105. **if** changeYaxis **and** **not** started:
1106. #display the change y axis panel
1107. pChangeY.displayPanel()
1109. #detect click on X velocity button
1110. colour19, bXvelClicked = bXvel.buttonClicked(clicked, X, Y)
1111. #display the button
1112. bXvel.displayButton(colour19)
1114. #detect click on Y velocity button
1115. colour20, bYvelClicked = bYvel.buttonClicked(clicked, X, Y)
1116. bYvel.displayButton(colour20)
1118. #detect click on time button
1119. colour21, bTimeClicked = bTime.buttonClicked(clicked, X, Y)
1120. bTime.displayButton(colour21)
1122. #detect click on force button
1123. colour26, bForceClicked = bForce.buttonClicked(clicked, X, Y)
1124. bForce.displayButton(colour26)
1126. #iterate through the buttons
1127. **for** n **in** IDButtons:
1128. #detect n button being clicked
1129. colour23, IDButtonClicked = n.buttonClicked(clicked, X, Y)
1130. #is the variable for the y axis not time
1131. **if**(VarY[0] != T):
1132. #is n button within the graph panel
1133. **if** n.posY+n.sizeY< pGraph.posY+pGraph.sizeY:
1134. #display the button
1135. n.displayButton(colour23)
1136. #is the button clicked
1137. **if** IDButtonClicked:
1138. #set the object for the y axis equal to the name of the button
1139. VarY[1] = int(n.name)
1140. #stop changing the y axis
1141. changeYaxis = False
1142. **else**:
1143. #is the button within the graph panel
1144. **if** n.posY+n.sizeY< pGraph.posY+pGraph.sizeY:
1145. #display the button but blanked out
1146. n.displayButton(grey4)
1148. #is the x velocity button clicked
1149. **if** bXvelClicked:
1150. #set the variable for y axis to x velocity
1151. VarY[0] = VX
1152. #is the object for the y axis -1
1153. **if** VarY[1] == -1:
1154. #set object for y axis to 0
1155. VarY[1] = 0
1156. #stop changing y axis
1157. changeYaxis = False
1159. #is the Y velocity button clicked
1160. **if** bYvelClicked:
1161. #set variable for y axis to Y velocity
1162. VarY[0] = VY
1163. #is the object for the y axis -1
1164. **if** VarY[1] == -1:
1165. #set object for y axis to 0
1166. VarY[1] = 0
1167. #stop changing Y axis
1168. changeYaxis = False
1170. #is the time button clicked
1171. **if** bTimeClicked:
1172. #set variable for y axis to time
1173. VarY[0] = T
1174. #set object for y axis to -1
1175. VarY[1] = -1
1176. #stop changing Y axis
1177. changeYaxis = False
1179. #is the button for Force clicked
1180. **if** bForceClicked:
1181. #set the y axis variable to net force
1182. VarY[0] = F
1183. #is the object of the y axis -1
1184. **if** VarY[1] == -1:
1185. #set object of the y axis to 0
1186. VarY[1] = 0
1187. changeYaxis = False
1189. #detects the start graphing button being clicked
1190. colour22, bStartGraphClicked = bStartGraph.buttonClicked(clicked, X, Y)
1192. #asks has the graphing started
1193. **if** started:
1194. #object for X axis has not been verified so set to false
1195. objectXExists=False
1196. #object for Y axis has not been verified so set to false
1197. objectYExists=False
1198. #asks is objects not empty
1199. **if** objects != []:

1202. ##binary search - the IDs of the objects will be in order so binary search works - it is also more efficient than linear search
1203. #variable that tells us if the search is finished
1204. found = False
1205. #if the object we are looking for exists it will always remain in the nextSearch list
1206. nextSearch = objects
1207. #loops untill the object is found(or untill it is found that it doesn't exist)
1208. **while** **not** found:
1209. #asks if nextSearch has more than 1 object
1210. **if** len(nextSearch) > 1:
1211. #finding halfway point in the list
1212. index = int(len(nextSearch)/2)
1213. **if** VarX[1] < nextSearch[index].ID:
1214. #discards the higher section
1215. nextSearch = nextSearch[:index]
1216. #asks if the ID we are looking for is greater than the one we are looking at right now
1217. **elif** VarX[1] > nextSearch[index].ID:
1218. #discards the lower section
1219. nextSearch = nextSearch[index:]
1220. **else**:
1221. #object is found
1222. found = True
1223. #returns the index of the object in the list of objects
1224. Xindex = objects.index(nextSearch[index])
1225. #object exists
1226. objectXExists = True
1227. **else**:
1228. #asks if the object we are looking at is the one we are looking for
1229. **if** nextSearch != []:
1230. **if** VarX[1] == nextSearch[0].ID:
1231. #object is found
1232. found = True
1233. #returns the index of the object in the list of objects
1234. Xindex = objects.index(nextSearch[0])
1235. #object exists
1236. objectXExists = True
1237. **else**:
1238. #object was not found but we finished the search
1239. found = True
1240. #object cannot exist
1241. objectXExists = False
1242. **else**:
1243. #object was not found but we finished the search
1244. found = True
1245. #object cannot exist
1246. objectXExists = False

1249. #variable that tells us if the search is finished
1250. found = False
1251. #if the object we are looking for exists it will always remain in the nextSearch list
1252. nextSearch = objects
1253. #loops untill the object is found(or untill it is found that it doesn't exist)
1254. **while** **not** found:
1255. #asks if nextSearch has more than 1 object
1256. **if** len(nextSearch) > 1:
1257. #finding halfway point in the list
1258. index = int(len(nextSearch)/2)
1259. **if** VarY[1] < nextSearch[index].ID:
1260. #discards the higher section
1261. nextSearch = nextSearch[:index]
1262. #asks if the ID we are looking for is greater than the one we are looking at right now
1263. **elif** VarY[1] > nextSearch[index].ID:
1264. #discards the lower section
1265. nextSearch = nextSearch[index:]
1266. **else**:
1267. #object is found
1268. found = True
1269. #returns the index of the object in the list of objects
1270. Yindex = objects.index(nextSearch[index])
1271. #object exists
1272. objectYExists = True
1273. **else**:
1274. #asks if the object we are looking at is the one we are looking for
1275. **if** nextSearch != []:
1276. **if** VarY[1] == nextSearch[0].ID:
1277. #object is found
1278. found = True
1279. #returns the index of the object in the list of objects
1280. Yindex = objects.index(nextSearch[0])
1281. #object exists
1282. objectYExists = True
1283. **else**:
1284. #object was not found but we finished the search
1285. found = True
1286. #object cannot exist
1287. objectYExists = False
1288. **else**:
1289. #object was not found but we finished the search
1290. found = True
1291. #object cannot exist
1292. objectYExists = False
1294. #asks if there is an object to be measured for the x axis
1295. **if** objectXExists:
1296. #asks if the variable is the x velocity of the object
1297. **if** VarX[0] == VX:
1298. #adds the x velocity to the list of coordinates
1299. XMetricCoords.append(objects[Xindex].velX)
1300. #asks if the variable is the y velocity of the object
1301. **elif** VarX[0] == VY:
1302. #adds the y velocity to the list of coordinates
1303. XMetricCoords.append(objects[Xindex].velY)
1304. #asks if the variable if the net force acting on the object
1305. **elif** VarX[0] == F:
1306. ##adds the net force to the list of coordinates
1307. XMetricCoords.append(objects[Xindex].netForce/(10\*\*24))
1308. #asks if there is an object to be measured for the x axis
1309. **if** objectYExists:
1310. **if** VarY[0] == VX:
1311. YMetricCoords.append(-objects[Yindex].velX)
1312. **elif** VarY[0] == VY:
1313. YMetricCoords.append(-objects[Yindex].velY)
1314. **elif** VarY[0] == F:
1315. YMetricCoords.append(-objects[Yindex].netForce/(10\*\*24))

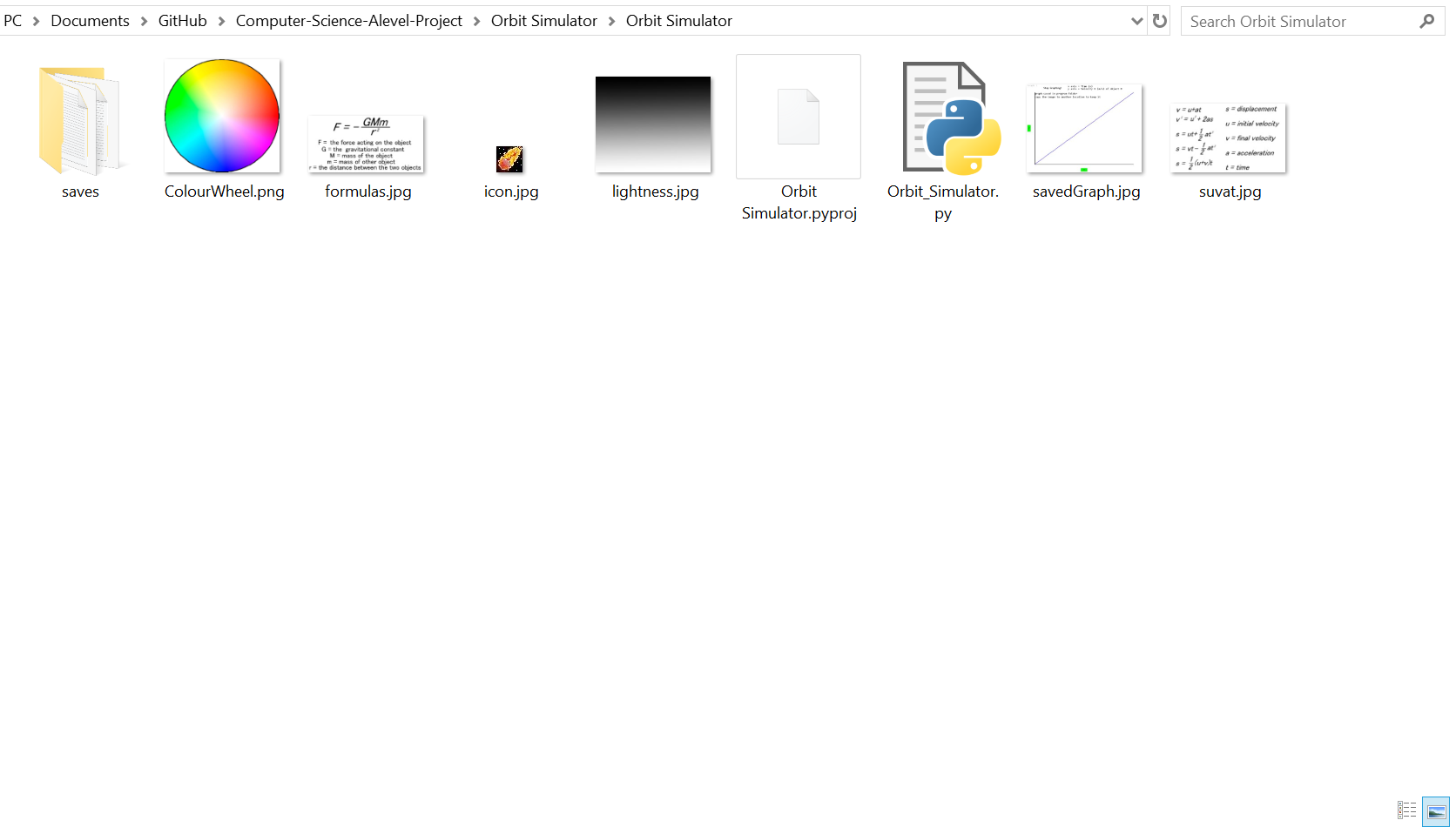


1320. #coordinates to be displayed on the screen
1321. screenCoords = []
1322. #x coordinates to be displayed on the screen
1323. XGraphCoords = []
1324. #y coordinates to be displayed on the screen
1325. YGraphCoords = []
1326. #is the simulation not paused and is the graph not stopped
1327. **if** **not** paused **and** **not** stop:
1328. #is the Y axis variable Time
1329. **if** VarY[0] == T:
1330. #adds the current time to the list of coordinates
1331. YMetricCoords.append(-timeElapsed)
1332. #object does exist
1333. objectYExists = True
1334. #is the X axis variable Time
1335. **if** VarX[0] == T:
1336. #adds the current time to the list of coordinates
1337. XMetricCoords.append(timeElapsed)
1338. #object does exist
1339. objectXExists = True
1341. #asks if object X does't exist or if object Y doesn't exist
1342. **if** **not** objectXExists **or** **not** objectYExists:
1343. #stops the graphing
1344. stop = True
1346. #minimum x coordinate
1347. minMetX = min(XMetricCoords)
1348. #text to be displayed at the start of the x axis
1349. lowerX = PanelFont.render(str(round(minMetX, 1)), 1, black)
1350. #displays the text
1351. display.blit(lowerX, (originX, originY+30))
1353. #maximum x coordinate
1354. maxMetX = max(XMetricCoords)
1355. #text to be displayed at the end of the x axis
1356. upperX = PanelFont.render(str(round(maxMetX, 1)), 1, black)
1357. #displays the text
1358. display.blit(upperX, (originX+pGraph.sizeX-150, originY+30))
1360. #size of the line in the x direction
1361. lineSizeX = maxMetX-minMetX
1363. #minimum y coordinate
1364. minMetY = min(YMetricCoords)
1365. #text to be displayed at the start of the y axis
1366. upperY = PanelFont.render(str(round(-minMetY, 1)), 1, black)
1367. #maximum y coordinate
1368. display.blit(upperY, (originX-50, originY-pGraph.sizeY+100))
1370. #maximum y coordinate
1371. maxMetY = max(YMetricCoords)
1372. #text to be displayed at the end of the y axis
1373. lowerY = PanelFont.render(str(round(-maxMetY, 1)), 1, black)
1374. #displays the text
1375. display.blit(lowerY, (originX-50, originY-15))
1377. #size of the line in the y direction
1378. lineSizeY = maxMetY-minMetY
1379. #asks if both objects exists
1380. **if** objectXExists **and** objectYExists:
1381. #sets the width of the graph equal to the width of the panel-100
1382. graphWidth = pGraph.sizeX-100
1383. #sets the height of the graph equal to the height of the panel-100
1384. graphHeight = pGraph.sizeY-100
1386. #asks if the line size in the x direction is greater than 0
1387. **if** lineSizeX > 0:
1388. #calculates the width ratio
1389. WRatio = lineSizeX/graphWidth
1390. **else**:
1391. WRatio = 1
1393. #asks if the line size in the y direction is greater than 0
1394. **if** lineSizeY > 0:
1395. #calculates height ratio
1396. HRatio = lineSizeY/graphHeight
1397. **else**:
1398. HRatio = 1
1400. #loops for the number of coordinates
1401. **for** n **in** range(len(XMetricCoords)):
1402. #converts metric to graph coorodinate and adds the results to the lists of graph coordinates
1403. XGraphCoords.append((XMetricCoords[n]/WRatio)-(minMetX/WRatio))
1404. YGraphCoords.append((YMetricCoords[n]/HRatio)-(maxMetY/HRatio))
1406. #loops for the number of coordinates
1407. **for** n **in** range(len(XGraphCoords)):
1408. #combines the x and y coordinates
1409. screenCoords.append(((XGraphCoords[n]+originX), YGraphCoords[n]+originY))
1411. #sets saveCoords equal to the screen coordinates
1412. savedCoords = screenCoords
1413. **try**:
1414. #draws the line onto the graph
1415. pygame.draw.lines(display, blue, False, savedCoords, 2)
1416. **except**:
1417. **pass**
1419. #detects stop graphing button being clicked
1420. colour24, bStopGraphClicked = bStopGraph.buttonClicked(clicked, X, Y)
1421. #displays the stop graphing button
1422. bStopGraph.displayButton(colour24)
1423. #asks is the stop graphing button clicked
1424. **if** bStopGraphClicked:
1425. #graphing is not started
1426. started = False
1427. #graphing is stopped
1428. stop = True
1429. #the area to be caputured is defined here
1430. area = pygame.Rect(pGraph.posX, pGraph.posY, pGraph.sizeX, pGraph.sizeY)
1431. #creates a subsurface of the display
1432. subSurface = display.subsurface(area)
1433. #saves an image called savedGrapg.jpg with the most recent graph in it
1434. pygame.image.save(subSurface, "savedGraph.jpg")
1436. #is the variable for the y axis Time
1437. **if** VarY[0] == T:
1438. #adds the time interval onto the time elapsed
1439. timeElapsed += Timeinterval
1441. #is the variable for the x axis Time
1442. **if** VarX[0] == T:
1443. #adds the time interval onto the time elapsed
1444. timeElapsed += Timeinterval
1445. #asks do both objects exist
1446. **elif** objectXExists **and** objectYExists:
1447. #displays the button to start graphing
1448. bStartGraph.displayButton(colour22)
1449. #resets the time elapsed
1450. timeElapsed = 0
1451. **else**:
1452. #if the objects list is not empty
1453. **if** objects != []:
1454. #iterate through objects
1455. **for** self **in** objects:
1456. #is the object equal to the variable for the X axis
1457. **if** self.ID == VarX[1]:
1458. #object does exist
1459. objectXExists = True
1460. #is the object equal to the variable for the Y axis
1461. **elif** self.ID == VarY[1]:
1462. #object does exist
1463. objectYExists = True
1464. #asks if not paused and not stopped
1465. **if** **not** paused **and** **not** stop:
1466. #asks if the Y axis variable is Time
1467. **if** VarY[0] == T:
1468. #there isn't an object required for Time so object for Y axis does exist
1469. objectYExists = True
1470. #asks if the Y axis variable is Time
1471. **if** VarX[0] == T:
1472. #there isn't an object required for Time so object for X axis does exist
1473. objectXExists = True
1475. #asks if the start graph button is clicked
1476. **if** bStartGraphClicked:
1477. #sets started to true
1478. started = True
1479. #resets the graph values for x
1480. XMetricCoords = []
1481. #resets the graph values for y
1482. YMetricCoords = []
1483. #sets the time elapsed to 0
1484. timeElapsed = 0
1486. #detects gravity button being clicked
1487. colour15, bGravityClicked = bGravity.buttonClicked(clicked, X, Y)
1488. #displays gravity button
1489. bGravity.displayButton(colour15)
1491. #detects suvat button being clicked
1492. colour16, bSuvatClicked = bSuvat.buttonClicked(clicked, X, Y)
1493. #displays suvat button
1494. bSuvat.displayButton(colour16)
1496. #detects the keplers second law button being clicked
1497. colour25, bKeplerClicked = bKepler.buttonClicked(clicked, X, Y)
1498. #displays the keplers second law button
1499. bKepler.displayButton(colour25)
1501. #asks if the gravity button is clicked
1502. **if** bGravityClicked:
1503. #displays the gravitation formula
1504. display.blit(gravitationFormula, (pLearn.posX+50, pLearn.posY+50))
1505. #asks if the suvat button is clicked
1506. **elif** bSuvatClicked:
1507. #displays the motion formulas
1508. display.blit(suvatFormula, (pLearn.posX+50, pLearn.posY+50))
1509. #asks if keplers second law button is clicked
1510. **elif** bKeplerClicked:
1511. #asks if demonstration is not running
1512. **if** **not** runKepler:
1513. #asks if the length of objects is not 0 or 1
1514. **if** **not**(len(objects) == 0 **or** len(objects) == 1):
1515. #asks if object0's mass is greater than object1's mass
1516. **if** (objects[0].metricMass) > (objects[1].metricMass):
1517. #this is the ID of the orbiting object
1518. testObject = 1
1519. #this is the ID of the center object
1520. centerObject = 0
1521. **else**:
1522. #this is the ID of the orbiting object
1523. testObject = 0
1524. #this is the ID of the center object
1525. centerObject = 1
1526. #resets the point lists
1527. pointList1 = [(objects[centerObject].drawPosX+centerX, objects[centerObject].drawPosY+centerY)]
1528. pointList2 = [(objects[centerObject].drawPosX+centerX, objects[centerObject].drawPosY+centerY)]
1529. #sets the amount of time passed to 0
1530. timePassed = 0
1531. #run kepler is set to true
1532. runKepler = True
1533. #there have been 0 repeats
1534. reps = 0
1535. **else**:
1536. #creates text to instruct the user on how to use the learn panel
1537. learnText = LearnFont.render("click on the buttons above to see the formulas", 1, grey3)
1538. #displays the text
1539. display.blit(learnText, (pLearn.posX+30, pLearn.posY+100))
1541. #asks if the number of objects is not 2 and kepler is running
1542. **if** len(objects)!=2 **and** runKepler:
1543. #creates error text
1544. learnText = LearnFont.render("There must be exactly 2 objects", 1, black)
1545. #creates more error text
1546. learnText2 = LearnFont.render(" in the simulation for this!", 1, black)
1547. #displays error text
1548. display.blit(learnText, (pLearn.posX+30, pLearn.posY+150))
1549. #displays more error text
1550. display.blit(learnText2, (pLearn.posX+30, pLearn.posY+200))
1551. #kepler demonstration is stopped
1552. runKepler = False
1554. ##this section runs the kepler demonstration
1555. #asks if we want to run the kepler demonstration
1556. **if** runKepler:
1557. #creates the first line of text
1558. learnText = LearnFont.render("A radius vector joining any two objects, sweeps", 1, black)
1559. #creates the second line of text
1560. learnText2 = LearnFont.render("out equal areas in equal lengths of time.", 1, black)
1561. #creates the third line of text
1562. learnText3 = LearnFont.render("green area = blue area", 1, black)
1563. #displays the first line of text
1564. display.blit(learnText, (pLearn.posX+30, pLearn.posY+150))
1565. #displays the second line of text
1566. display.blit(learnText2, (pLearn.posX+30, pLearn.posY+200))
1567. #displays the third line of text
1568. display.blit(learnText3, (pLearn.posX+30, pLearn.posY+250))
1570. #asks if the time passed is greater than1 second
1571. **if** timePassed < FPS:
1572. #appends the position of the object to the point list
1573. pointList1.append(((objects[testObject].drawPosX+centerX),(objects[testObject].drawPosY+centerY)))
1574. #asks if the time passed is greater than 2 seconds but less than 3 seconds
1575. **if** timePassed >2\*FPS **and** timePassed<3\*FPS:
1576. #adds the position of the object to the other point list
1577. pointList2.append(((objects[testObject].drawPosX+centerX),(objects[testObject].drawPosY+centerY)))
1578. #asks if the time passed is greater than 3.3 seconds
1579. **if** timePassed > 3.3\*FPS:
1580. #sets time passed to 0
1581. timePassed = 0
1582. #resets the point lists to only have the center points of the obejcts in the lists
1583. pointList1 = [(objects[centerObject].drawPosX+centerX, objects[centerObject].drawPosY+centerY)]
1584. pointList2 = [(objects[centerObject].drawPosX+centerX, objects[centerObject].drawPosY+centerY)]
1585. #adds 1 to repeats
1586. reps += 1
1588. #increases time Passed by 1 (measured in frames)
1589. timePassed+=1
1591. #asks if the number of repeats if greater than 4
1592. **if** reps > 4:
1593. #sets run kepler to false so it stops the demonstration
1594. runKepler = False
1595. #resets the point lists to only have the center points of the obejcts in the lists
1596. pointList1 = [(objects[centerObject].drawPosX+centerX, objects[centerObject].drawPosY+centerY)]
1597. pointList2 = [(objects[centerObject].drawPosX+centerX, objects[centerObject].drawPosY+centerY)]
1598. #sets repeats to 0
1599. reps = 0

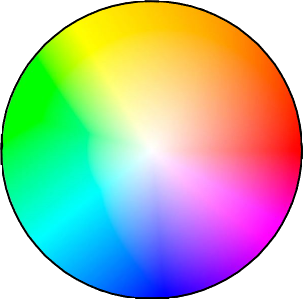
1602. ##displaying the switches
1603. **for** switch **in** switches:
1604. #asks if the left mouse button is clicked and the cursor is within the switch
1605. **if** clicked **and** X>(switch.posX-2) **and** Y>(switch.posY-2) **and** X<(switch.posX+82) **and** Y<(switch.posY+42):
1606. #sets switch was clicked to True
1607. switch.wasClicked = True
1608. #asks if the switch was clicked in the last fram but is not clicked in this frame
1609. **elif** switch.wasClicked **and** **not** clicked:
1610. #toggles the switch
1611. switch.state = **not** switch.state
1612. #sets switch was clicked to False
1613. switch.wasClicked = False
1614. #displays the switch
1615. switch.displaySwitch()
1617. #draws a grey rectangle for the colour wheel to be drawn on top of
1618. pygame.draw.rect(display, grey1, (pTools.posX+3, sShowTrail.posY+45, pTools.sizeX-8, int(pTools.sizeX\*(4/5)+10)))
1619. #draws a black outline for the rectangle
1620. pygame.draw.rect(display, black, (pTools.posX+3, sShowTrail.posY+45, pTools.sizeX-8, int(pTools.sizeX\*(4/5)+10)), 2)
1621. #displays the colour wheel onto the rectangle
1622. display.blit(colourWheel, (sShowFor.posX+3, sShowTrail.posY+55))
1623. #displays the lightness scale onto the rectangle
1624. display.blit(lightness, (pTools.posX+pTools.sizeX\*(4/5), sShowTrail.posY+50))
1625. #displays a black outline for the lightness scale
1626. pygame.draw.rect(display, black, (pTools.posX+pTools.sizeX\*(4/5), sShowTrail.posY+50, int(pTools.sizeX/5)-10, int(pTools.sizeX\*(4/5))), 2)
1628. #asks if the cursor is within the colour wheel
1629. **if** X>sShowFor.posX+3 **and** X<(int(pTools.sizeX\*(3/4)))+sShowFor.posX+3 **and** Y>sShowTrail.posY+55 **and** Y<sShowTrail.posY+55+(int(pTools.sizeX\*(3/4))):
1630. #asks if the left mouse button is clicked
1631. **if** clicked:
1632. #sets the current red value
1633. currentR = list(display.get\_at((X, Y)))[0]/255
1634. #sets the current green vlaue
1635. currentG = list(display.get\_at((X, Y)))[1]/255
1636. #sets the current blue value
1637. currentB = list(display.get\_at((X, Y)))[2]/255
1638. #puts the colour values in a list
1639. currentHUE = [currentR, currentG, currentB]
1641. #asks if the cursor is within the lightness scale
1642. **if** X>pTools.posX+pTools.sizeX\*(4/5) **and** X<pTools.posX+pTools.sizeX\*(4/5)+int(pTools.sizeX/5)-10 **and** Y>sShowTrail.posY+50 **and** Y<sShowTrail.posY+50+int(pTools.sizeX\*(4/5)):
1643. #asks if the left mouse button is clicked
1644. **if** clicked:
1645. #gets the lightness value
1646. reduce = list(display.get\_at((X,Y)))[0]
1648. #makes the current colour by multiplying the ratios of the RGB values by the lightness value
1649. currentcolour = (int(currentHUE[0]\*reduce), int(currentHUE[1]\*reduce), int(currentHUE[2]\*reduce))
1651. #draws a rectangle to show the current colour
1652. pygame.draw.rect(display, currentcolour, (pTools.posX+5, sShowTrail.posY+55+int(pTools.sizeX\*(4/5)+10), 50, 50))
1653. #draws a rectangle to outline the colour preview
1654. pygame.draw.rect(display, black, (pTools.posX+5, sShowTrail.posY+55+int(pTools.sizeX\*(4/5)+10), 50, 50), 2)
1656. #sets paused equal to the state that the pause button is in
1657. paused = sPausePlay.state
1659. ##creating text to be displayed on the information panel
1660. radiusLabel = PanelFont.render("Current Radius:", 1, black)
1661. radiusInfo = PanelFont.render("  " + str(int(metersPerPixel\*newRadius)) + " m", 1, black)
1662. densityLabel = PanelFont.render("Current Density:", 1, black)
1663. densityInfo = PanelFont.render("  " + str(int(newDensity))+" Kg/m^3", 1, black)
1664. scaleLabel = PanelFont.render("Current Scale:", 1, black)
1665. scaleInfo = PanelFont.render("  " + str(int(metersPerPixel))+" meters per pixel", 1, black)
1666. timeLabel = PanelFont.render("Current speed of time:", 1, black)
1667. timeInfo = PanelFont.render("  x"+str(int(round(FPS\*Timeinterval, 3))), 1, black)
1669. ##displaying the text onto the information panel
1670. display.blit(radiusLabel, (8, 50+int(height/2)))
1671. display.blit(radiusInfo, (8, 70+int(height/2)))
1672. display.blit(densityLabel, (8, 100+int(height/2)))
1673. display.blit(densityInfo, (8, 120+int(height/2)))
1674. display.blit(scaleLabel, (8, 150+int(height/2)))
1675. display.blit(scaleInfo, (8, 170+int(height/2)))
1676. display.blit(timeLabel, (8, 200+int(height/2)))
1677. display.blit(timeInfo, (8, 220+int(height/2)))
1679. #detects the reset button being clicked
1680. colour5, bResetClicked = bReset.buttonClicked(clicked, X, Y)
1681. #displays the reset button
1682. bReset.displayButton(colour5)
1684. #asks if the reset button is clicked
1685. **if** bResetClicked:
1686. #deletes all objects by setting the list of objects to an empty list
1687. objects = []
1688. #number of planets is now 0
1689. number = 0
1690. #density is reset
1691. newDensity = 5500
1692. #radius is reset
1693. newRadius = 10
1694. #meters per pixel is reset
1695. metersPerPixel = 500000
1696. #center X is reset
1697. centerX = width/3
1698. #center Y is reset
1699. centerY = height/3
1700. #Time interval is reset
1701. Timeinterval = 10000/FPS
1703. #asks if the graphing is not started
1704. **if** **not** started:
1705. #allows the increase time interval to be clicked
1706. colour6, bIncreaseClicked = bIncrease.buttonClicked(clicked, X, Y)
1707. **else**:
1708. #increase time interval button can't be clicked
1709. colour6 = grey4
1710. bIncreaseClicked = False
1711. #displays the increase time interval button
1712. bIncrease.displayButton(colour6)
1714. #asks if the increase time interval button is clicked and asks if it has not been held or has been held for a second
1715. **if** bIncreaseClicked **and** (heldI == 0 **or** heldI > FPS):
1716. #multiplies the time interval by 2
1717. Timeinterval \*= 2
1718. #adds 1 to the held variable
1719. heldI+=1
1720. #asks if the button is not clicked
1721. **elif** **not** bIncreaseClicked:
1722. #sets the held variable to 0
1723. heldI=0
1725. #asks if the graphing is not started
1726. **if** **not** started:
1727. #allows the decrease time interval to be clicked
1728. colour7, bDecreaseClicked = bDecrease.buttonClicked(clicked, X, Y)
1729. **else**:
1730. #decrease time interval button can't be clicked
1731. colour7 = grey4
1732. bDecreaseClicked = False
1733. #displays the decrease time interval button
1734. bDecrease.displayButton(colour7)
1736. #asks if the decrease time interval button is clicked and asks if it has not been held or has been held for a second
1737. **if** bDecreaseClicked **and** (heldD == 0 **or** heldD > FPS):
1738. #divides the time interval by 2
1739. Timeinterval /= 2
1740. #adds 1 to the held variable
1741. heldD+=1
1742. #asks if the button is not clicked
1743. **elif** **not** bDecreaseClicked:
1744. #sets the held variable to 0
1745. heldD=0
1747. #displays the boarder
1748. WindowGUI(clicked, X, Y, ClickedLastFrame)
1750. #detects the main menu button being clicked
1751. colour4, bMainMenuClicked = bMainMenu.buttonClicked(clicked, X, Y)
1752. #displays the main menu button
1753. bMainMenu.displayButton(colour4)
1755. #detects the instruction button being clicked
1756. colour13, bInstructionsClicked = bInstructions.buttonClicked(clicked, X, Y)
1757. #displays the instructions button
1758. bInstructions.displayButton(colour13)
1760. **if** bInstructionsClicked:
1761. #displays the instructions panel
1762. pInstructions.displayPanel()
1764. ##tool bar instructions
1766. #creates the next instruction
1767. InsToolBar = PanelFont.render("1) Tool bar - This panel gives you options to affect the simulation screen", 1, black)
1768. #displays the next instruction
1769. display.blit(InsToolBar, (pInstructions.posX+5, pInstructions.posY+30))
1770. #draws a line from the instruction to the switch
1771. pygame.draw.line(display, black, (pInstructions.posX+5, pInstructions.posY+30), (pTools.posX+50, pTools.posY+10), 3)
1773. #creates the next instruction
1774. InsShowID = PanelFont.render("a) Show ID - View the ID of each object", 1, black)
1775. #displays the next instruction
1776. display.blit(InsShowID, (pInstructions.posX+20, pInstructions.posY+90))
1777. #draws a line from the instruction to the switch
1778. pygame.draw.line(display, yellow, (pInstructions.posX+20, pInstructions.posY+90), (sShowID.posX, sShowID.posY), 3)
1780. #creates the next instruction
1781. InsShowVel = PanelFont.render("b) Show velocity - View the velocity of each object", 1, black)
1782. #displays the next instruction
1783. display.blit(InsShowVel, (pInstructions.posX+20, pInstructions.posY+120))
1784. #draws a line from the instruction to the switch
1785. pygame.draw.line(display, yellow, (pInstructions.posX+20, pInstructions.posY+120), (sShowVel.posX, sShowVel.posY), 3)
1787. #creates the next instruction
1788. InsShowFor = PanelFont.render("c) Show net force - View the net force acting on each object", 1, black)
1789. #displays the next instruction
1790. display.blit(InsShowFor, (pInstructions.posX+20, pInstructions.posY+150))
1791. #draws a line from the instruction to the switch
1792. pygame.draw.line(display, yellow, (pInstructions.posX+20, pInstructions.posY+150), (sShowFor.posX, sShowFor.posY), 3)
1794. #creates the next instruction
1795. InsShowTrail = PanelFont.render("c) Show net trial - see the path that each object traces", 1, black)
1796. #displays the next instruction
1797. display.blit(InsShowTrail, (pInstructions.posX+20, pInstructions.posY+180))
1798. #draws a line from the instruction to the switch
1799. pygame.draw.line(display, yellow, (pInstructions.posX+20, pInstructions.posY+180), (sShowTrail.posX, sShowTrail.posY), 3)
1801. #Info panel instructions
1803. #creates the next instruction
1804. InsInfo = PanelFont.render("2) Information Panel - This lets you view information about the current time in the simulation", 1, black)
1805. #displays the next instruction
1806. display.blit(InsInfo, (pInstructions.posX+5, pInstructions.posY+300))
1807. pygame.draw.line(display, black, (pInstructions.posX+5, pInstructions.posY+300), (pInfo.posX, pInfo.posY), 3)
1809. #creates the next instruction
1810. InsCurrRad =  PanelFont.render("a) Current radius is the radius of the pointer used to insert objects in m", 1, black)
1811. #displays the next instruction
1812. display.blit(InsCurrRad, (pInstructions.posX+20, pInstructions.posY+330))
1813. pygame.draw.line(display, black, (pInstructions.posX+20, pInstructions.posY+330), (50, 50+int(height/2)), 3)
1815. #creates the next instruction
1816. InsCurrDens =  PanelFont.render("b) Current density is the density selected for the next object in Kg/m^3", 1, black)
1817. #displays the next instruction
1818. display.blit(InsCurrDens, (pInstructions.posX+20, pInstructions.posY+360))
1819. pygame.draw.line(display, black, (pInstructions.posX+20, pInstructions.posY+360), (50, 100+int(height/2)), 3)
1821. #creates the next instruction
1822. InsCurrScale = PanelFont.render("c) Current Scale is the number of meters in a pixel", 1, black)
1823. #displays the next instruction
1824. display.blit(InsCurrScale, (pInstructions.posX+20, pInstructions.posY+390))
1825. pygame.draw.line(display, black, (pInstructions.posX+20, pInstructions.posY+390), (50, 150+int(height/2)), 3)
1827. #creates the next instruction
1828. InsCurrTime = PanelFont.render("d) Current speed of time is how quickly the simulation is going", 1, black)
1829. #displays the next instruction
1830. display.blit(InsCurrTime, (pInstructions.posX+20, pInstructions.posY+410))
1831. pygame.draw.line(display, black, (pInstructions.posX+20, pInstructions.posY+410), (50, 200+int(height/2)), 3)
1833. #Time Panel instructions
1835. #creates the next instruction
1836. InsTime = PanelFont.render("3) Time Panel - adjust the flow of time", 1, black)
1837. #displays the next instruction
1838. display.blit(InsTime, (pInstructions.posX+5, pInstructions.posY+500))
1839. pygame.draw.line(display, black, (pInstructions.posX+5, pInstructions.posY+510), (pTime.posX, pTime.posY), 3)
1841. #creates the next instruction
1842. InsPause = PanelFont.render("a) Pause/Play switch - stop and start the flow of time", 1, black)
1843. #displays the next instruction
1844. display.blit(InsPause, (pInstructions.posX+20, pInstructions.posY+530))
1845. pygame.draw.line(display, green, (pInstructions.posX+20, pInstructions.posY+540), (sPausePlay.posX+30, sPausePlay.posY), 3)
1847. #creates the next instruction
1848. InsReset = PanelFont.render("b) Reset button - deletes all objects on the screen", 1, black)
1849. #displays the next instruction
1850. display.blit(InsReset, (pInstructions.posX+35, pInstructions.posY+560))
1851. pygame.draw.line(display, red, (pInstructions.posX+35, pInstructions.posY+570), (bReset.posX+30, bReset.posY), 3)
1853. #creates the next instruction
1854. InsIncrease = PanelFont.render("c) >>+ - speed up the flow of time", 1, black)
1855. #displays the next instruction
1856. display.blit(InsIncrease, (pInstructions.posX+50, pInstructions.posY+590))
1857. pygame.draw.line(display, cyan, (pInstructions.posX+50, pInstructions.posY+600), (bIncrease.posX+30, bIncrease.posY), 3)
1859. #creates the next instruction
1860. InsDecrease = PanelFont.render("d) >>- - slow down the flow of time", 1, black)
1861. #displays the next instruction
1862. display.blit(InsDecrease, (pInstructions.posX+65, pInstructions.posY+620))
1863. pygame.draw.line(display, cyan, (pInstructions.posX+65, pInstructions.posY+630), (bDecrease.posX+30, bDecrease.posY), 3)
1865. colour14, bInstructions2clicked = bInstructions2.buttonClicked(clicked, X, Y)
1866. bInstructions2.displayButton(colour14)
1868. #asks if the second instructions button is clicked
1869. **if** bInstructions2clicked:
1870. #displays the instructions panel
1871. pInstructions.displayPanel()
1873. #creates sub title text
1874. Ins1 = PanelFont.render("How to use the simulation:", 1, black)
1875. #displays subtitle text
1876. display.blit(Ins1, (pInstructions.posX+5, pInstructions.posY+30))
1878. #creates the next instruction
1879. Ins2 = PanelFont.render("1) Move the mouse over the black simulation screen. you should see a blue circle where the mouse is.", 1, black)
1880. #displays the next instruction
1881. display.blit(Ins2, (pInstructions.posX+5, pInstructions.posY+60))
1883. #creates the next instruction
1884. Ins3 = PanelFont.render("2) Use the scroll wheel to change the size of the blue circle", 1, black)
1885. #displays the next instruction
1886. display.blit(Ins3, (pInstructions.posX+5, pInstructions.posY+90))
1888. #creates the next instruction
1889. Ins4 = PanelFont.render("3) Left click on the simulation screen to add an object the same size as the blue circle", 1, black)
1890. #displays the next instruction
1891. display.blit(Ins4, (pInstructions.posX+5, pInstructions.posY+120))
1893. #creates the next instruction
1894. Ins5 = PanelFont.render("4) Left click and drag on the simulation screen to adjust the object's initial velocity", 1, black)
1895. #displays the next instruction
1896. display.blit(Ins5, (pInstructions.posX+5, pInstructions.posY+150))
1898. #creates the next instruction
1899. Ins6 = PanelFont.render("5) You can add multiple objects into the simulation at a time. They will automatically iteract by gravity", 1, black)
1900. #displays the next instruction
1901. display.blit(Ins6, (pInstructions.posX+5, pInstructions.posY+180))
1903. #creates the next instruction
1904. Ins7 = PanelFont.render("6) Holding down shift and z will let you zoom in and out by using the scroll wheel", 1, black)
1905. #displays the next instruction
1906. display.blit(Ins7, (pInstructions.posX+5, pInstructions.posY+210))
1908. #creates the next instruction
1909. Ins8 = PanelFont.render("7) Holding down shift and d will let you change the density of the next object using the scroll wheel", 1, black)
1910. #displays the next instruction
1911. display.blit(Ins8, (pInstructions.posX+5, pInstructions.posY+240))
1913. #creates the next instruction
1914. Ins9 = PanelFont.render("8) Click the buttons on the graph panel to start graphing, change axis variables", 1, black)
1915. #displays the next instruction
1916. display.blit(Ins8, (pInstructions.posX+5, pInstructions.posY+240))
1918. #asks if the main menu button isw clicked
1919. **if** bMainMenuClicked:
1920. #sets done equal to false
1921. done = False
1922. #loops untill done is equal to True
1923. **while** **not** done:
1924. #iterates through all possible pygame events:
1925. **for** event **in** pygame.event.get():
1926. #asks if the type of event is quit
1927. **if** event.type == quit:
1928. #quits pygame
1929. pygame.quit()
1930. #quits python
1931. sys.exit()
1932. #gets x coordinate of mouse
1933. X = list(pygame.mouse.get\_pos())[0]
1934. #gets y coordinate of mouse
1935. Y = list(pygame.mouse.get\_pos())[1]
1936. #gets left mouse button state
1937. clicked = list(pygame.mouse.get\_pressed())[0]
1938. #displays the exit panel
1939. pExit.displayPanel()
1941. #calls method for detecting button click on the cancel button
1942. colour8, bCancelClicked = bCancel.buttonClicked(clicked, X, Y)
1943. #displays the cancel button
1944. bCancel.displayButton(colour8)
1946. #calls method for detecting button click on the dont save button
1947. colour9, bDontSaveClicked = bDontSave.buttonClicked(clicked, X ,Y)
1948. #displays the dont save button
1949. bDontSave.displayButton(colour9)
1951. #calls method for detecting button click on the save button
1952. colour10, bSaveClicked = bSave.buttonClicked(clicked, X, Y)
1953. #displays the save button
1954. bSave.displayButton(colour10)
1956. #asks if the cancel button has just been clicked
1957. **if** bCancelClicked:
1958. #sets done to true to end the loop
1959. done = True
1960. #creates confirmation text
1961. cancelText = PanelFont.render("ok", 1, grey1)
1962. #displays confirmation text
1963. display.blit(cancelText, (width/2, height/2))
1964. #asks if the don't save button is clicked
1965. **elif** bDontSaveClicked:
1966. #creates confirmation text
1967. dontSaveText = PanelFont.render("ok", 1, grey1)
1968. #displays confirmation text
1969. display.blit(dontSaveText, (width/2, height/2))
1970. #sets done to true to end the loop
1971. done = True
1972. #sets in main menue to true so that we go back to the main menu
1973. inMainMenu = True
1974. #resets the objects
1975. objects = []
1976. #changes number to be 0
1977. number = 0
1978. #asks if the save button is clicked
1979. **elif** bSaveClicked:
1980. #sets done to true
1981. done = True
1982. FileName = "simulation"+str(len(fileNames)+1)
1983. #creates a new file with the new file name
1984. file = open("saves\\"+FileName+".txt", "w")
1985. #sets data to be an empty list
1986. data = []
1987. #iterates through all of the planets in the simulation
1988. **for** item **in** objects:
1989. #appends the data of the item that the loop is currently on to the data
1990. data.append([item.ID, item.metricMass, item.metricPosX, item.metricPosY, item.velX, item.velY, item.density])
1991. #writes the data to the file
1992. file.write(str(data))
1993. #saves and closes the file
1994. file.close()
1995. #creates the confirmation text
1996. SaveText = PanelFont.render("saving as: "+FileName, 1, grey1)
1997. #displays the confirmation text
1998. display.blit(SaveText, (width/2-100, height/2))
2000. #sets in main menu to true to we can go to main menu
2001. inMainMenu = True
2002. #resests the objects
2003. objects = []
2004. #changes number to be 0
2005. number = 0
2006. **else**:
2007. #creates a prompt text
2008. promptText = PanelFont.render("Do you want to save this siulation?", 1, grey1)
2009. #displays the prompt text
2010. display.blit(promptText, (width/2-100, height/2))
2012. # this updates the screen so anything drawn to the display is actually shown on the users monitor
2013. pygame.display.update()
2014. # waits for 1/FPS seconds
2015. FPSClock.tick(FPS)
2016. #waits for 1.5 seconds
2017. time.sleep(1.5)
2018. #asks if the left mouse button is clicked
2019. **if** clicked:
2020. ClickedLastFrame = True
2021. **else**:
2022. ClickedLastFrame = False
2023. # this updates the screen so anything drawn to the display is actually shown on the users monitor
2024. pygame.display.update()
2025. # waits for 1/FPS seconds
2026. FPSClock.tick(FPS)

## Other files

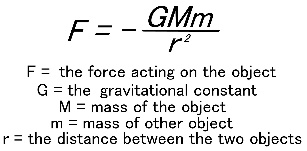
The code is not the only file to make up the whole project. Within the program folder, there are many different files. This contains images that are used in the program, the saves folder as well as the python file itself.

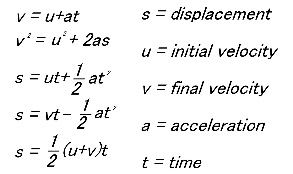


The images that are needed are:

I use this colour wheel to let the user select the RGB values of the object that they wish to add in. To do this I have used a Pygame function ( display.get\_at((x, y)) ) that returns the colour values of the pixel at that location on the screen. I can then divide those by 255 and multiply them by the lightness value (explained by the next image) to get the final colour values.

I use this lightness scale to let the user select the lightness of the colour of the object that they wish to add in. I use display.get\_at((x, y)) to get this value. Then I can multiply this by the RGB values to get the final colour values.

This image is part of the “learn” panel. When the user clicks and holds on the gravity button in the learn panel this image shows allowing the user to see the mathematics of gravity.

This image is also part of the “learn” panel but shows the calculations for motion.

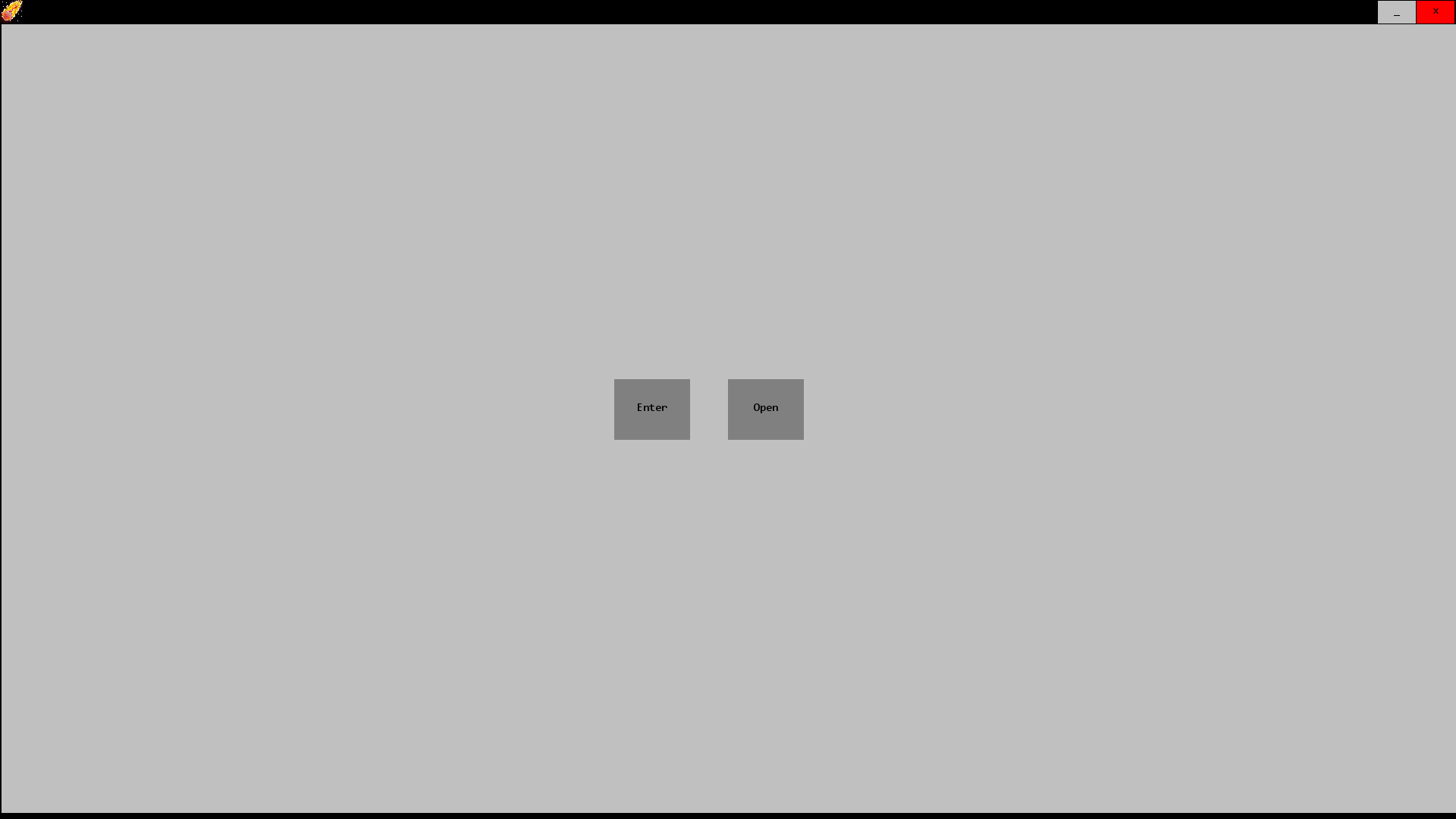
This image is just a logo that I use in the top left corner.

## Running the Final Program

This section will show screenshots of the program giving an idea of what the program looks like before the testing section has started.

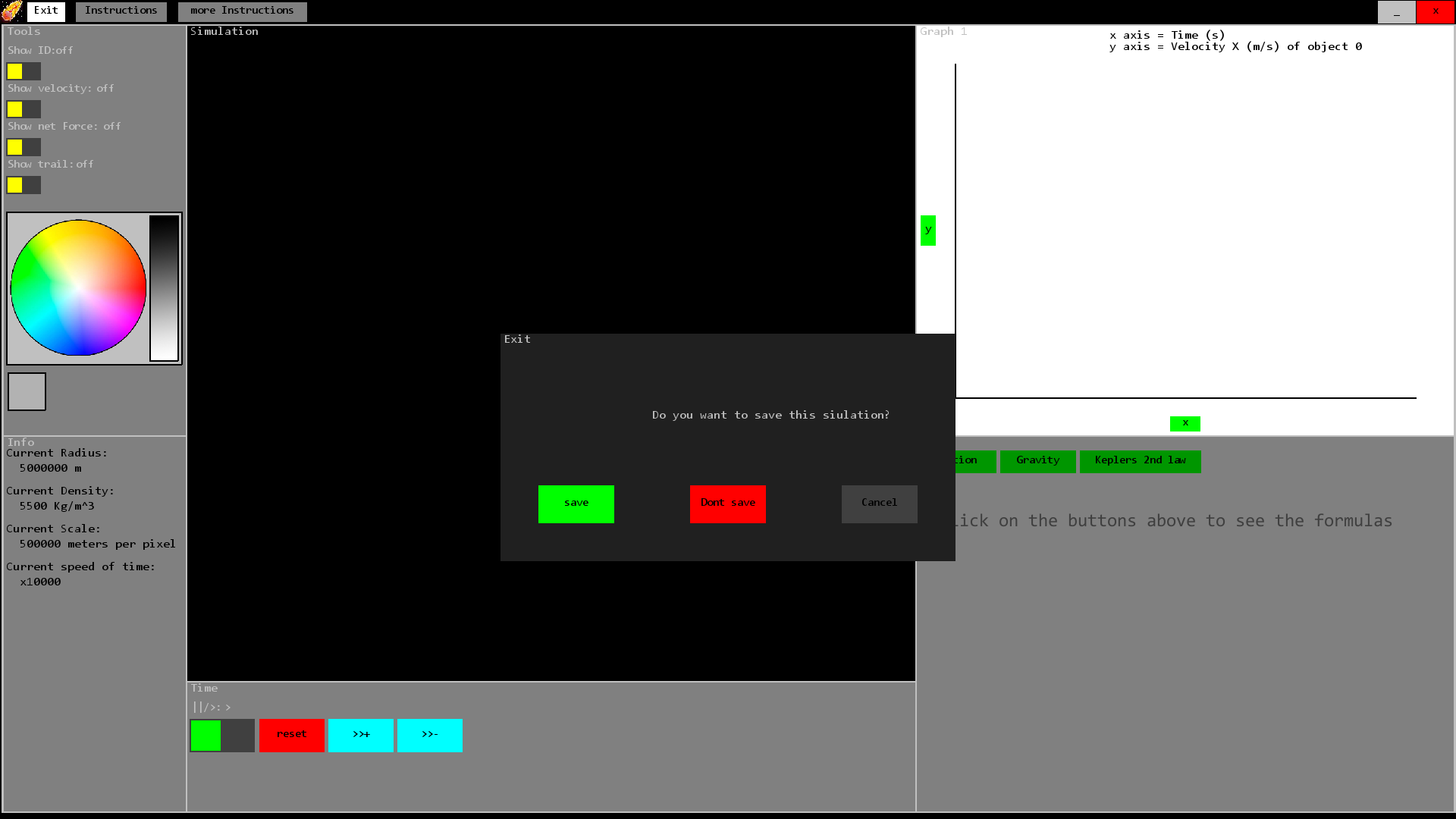
### Starting up

This is the screen that displays when the user has started up the program. It shows a bar across the top with a close and minimise button at the top right. The screen also show an enter button for a new simulation or open for a saved simulation.



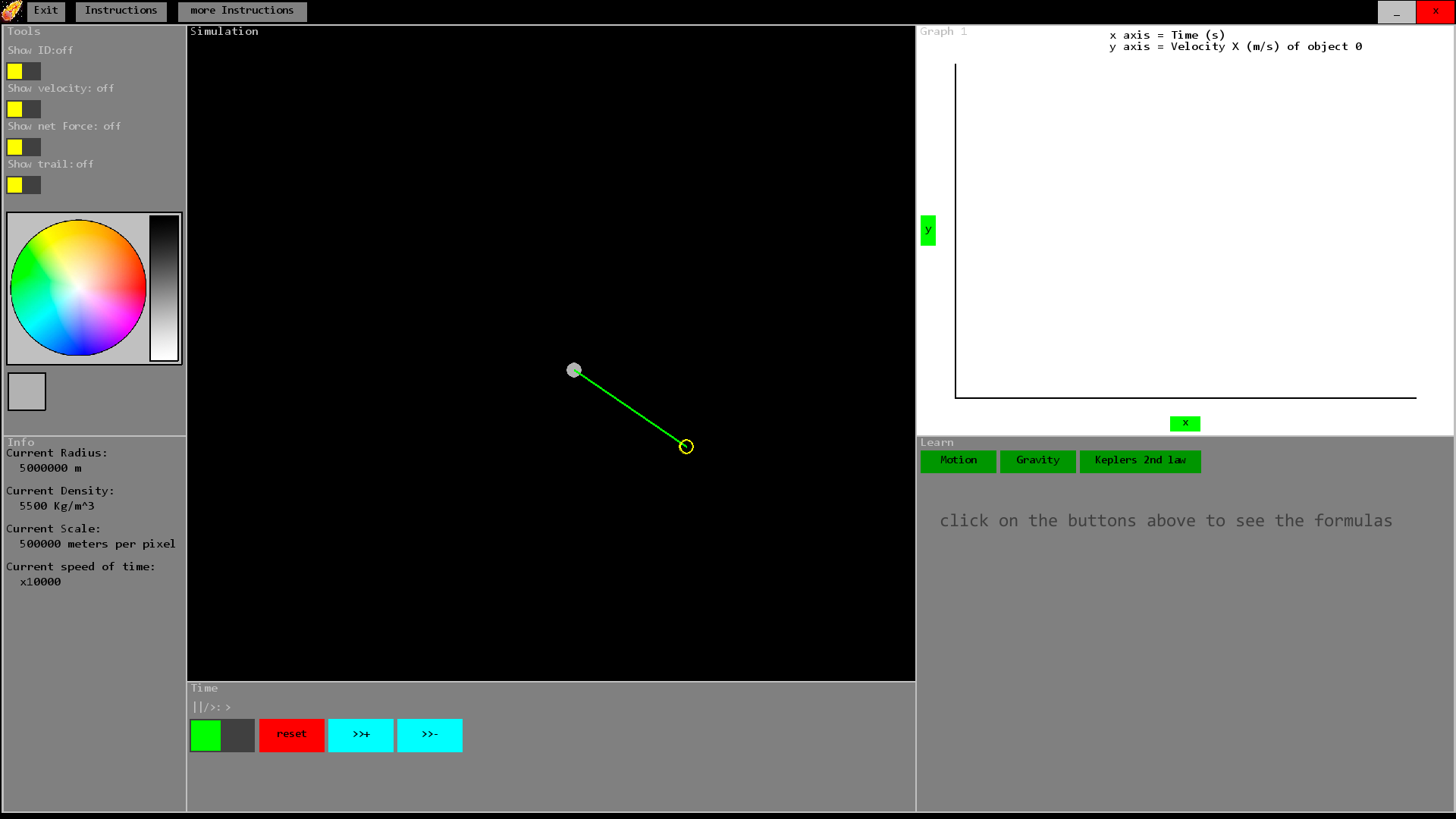
### Pressing the exit button

Once the user has entered a new simulation, they can press exit. This shows the saving options. The options shown are save(green), don’t save(red) and cancel (grey).



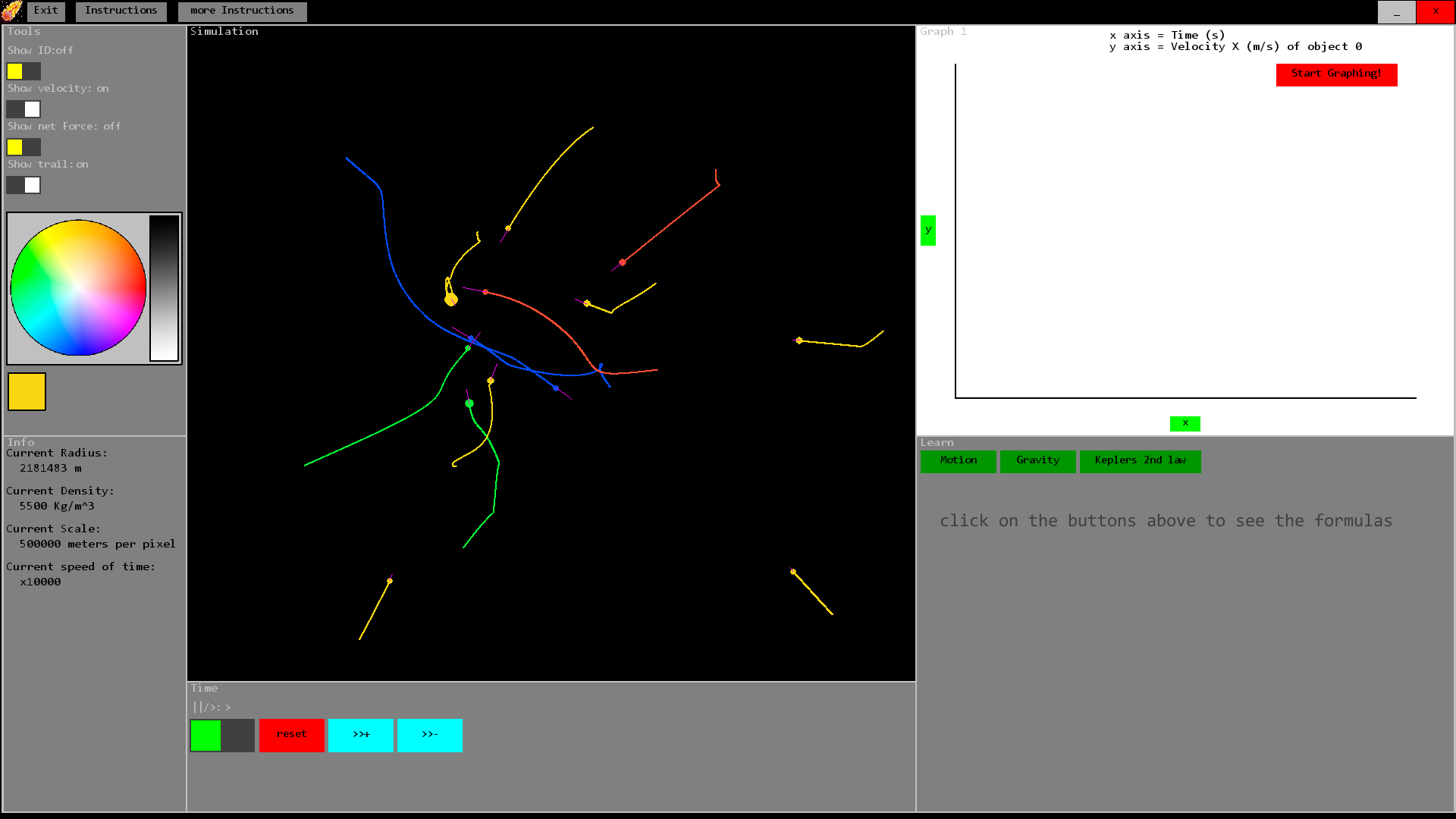
### Adding an object in:

To add an object in the user clicks, drags and releases. The distance the user drags the mouse indicates the speed of the object. The direction the mouse is dragged is the opposite to the direction of the speed of the object.



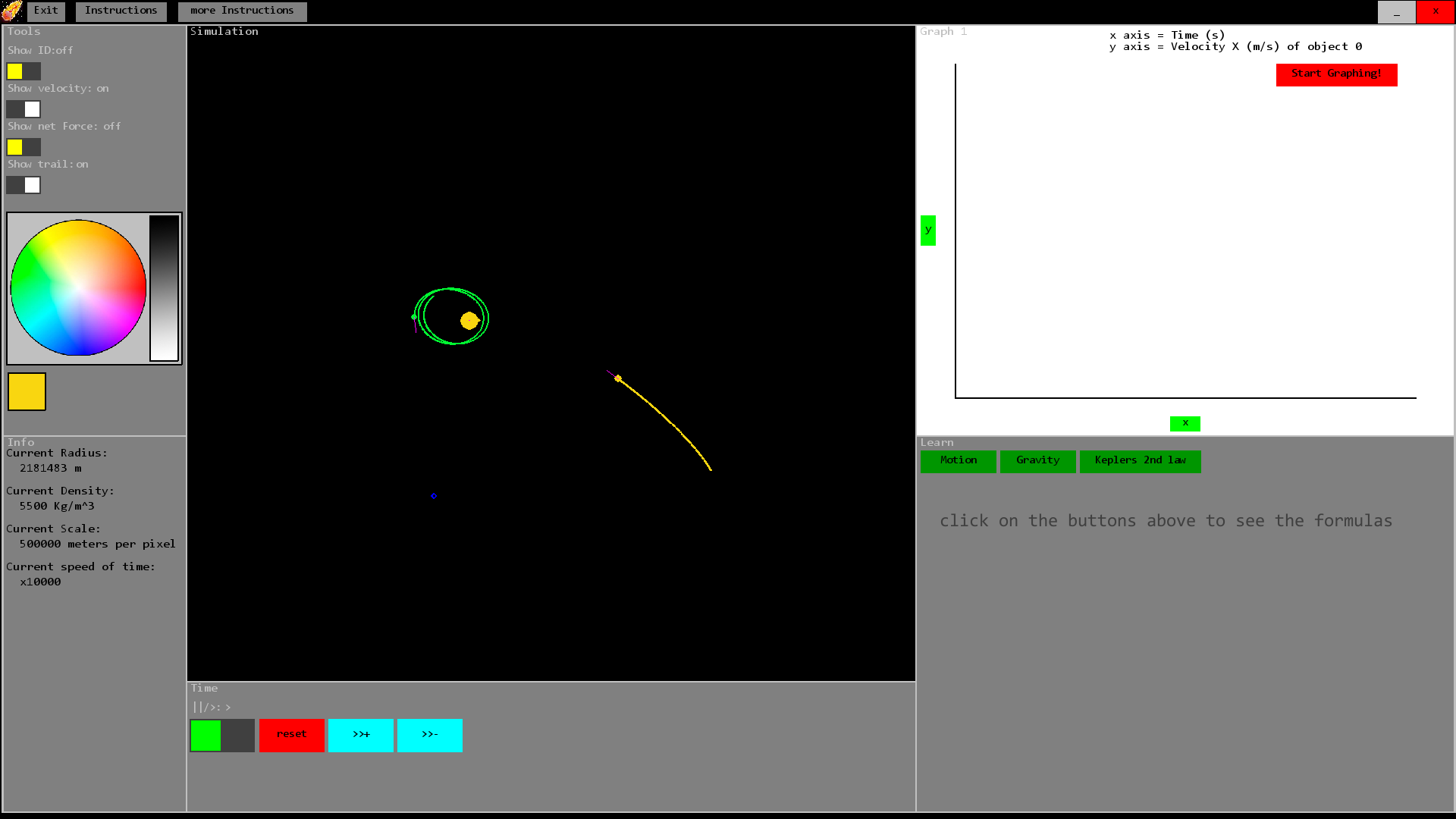
### Added multiple objects of different colours and showing their trail

In this simulation I added in lots of objects to see how they interact. The object have been given different colours. There is a switch on the tool bar that turns on the trails for the objects so you can see where the objects have been.

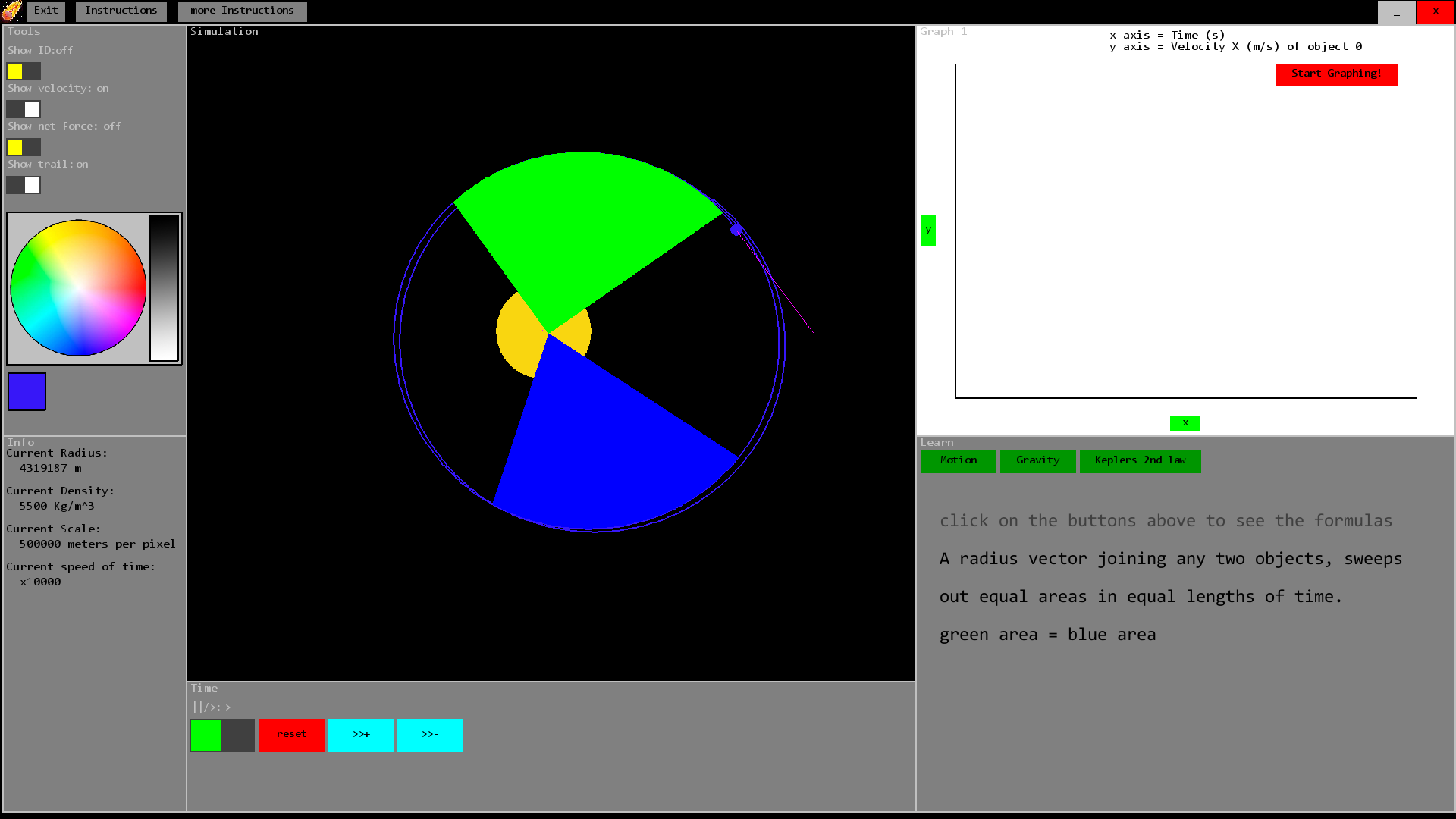


### A stable orbit formed out of the chaotic beginning

This is the result of the previous simulation setup. A small green object has formed a stable elliptical orbit around a larger yellow object.

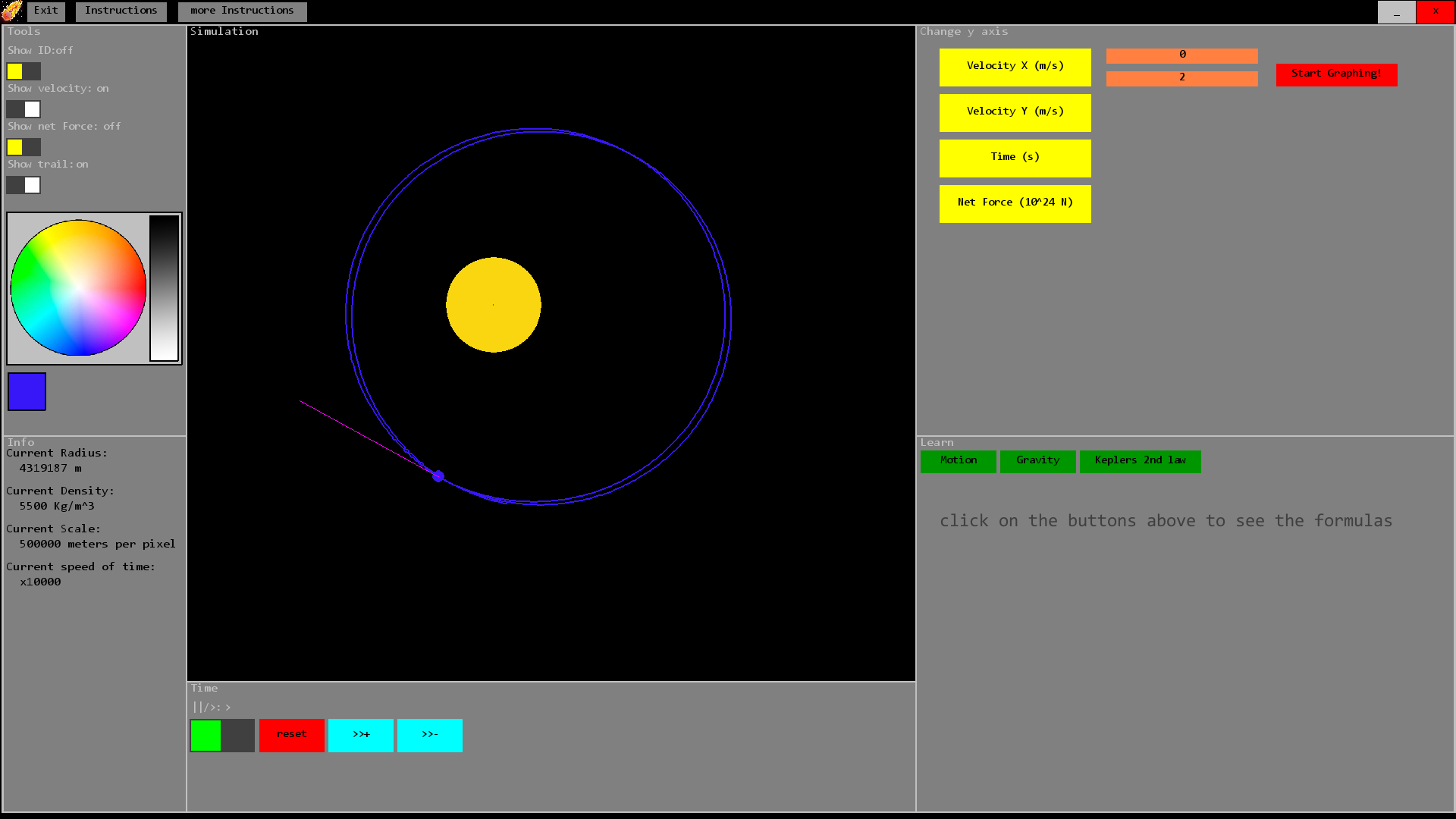


### Demonstrating Kepler’s second law:

After setting up a simulation with a stable orbit of two objects, I clicked the Kepler’s second law button. It draws two shapes showing the areas created by two objects over equal lengths of time. There is also a description of the law in the learn panel.

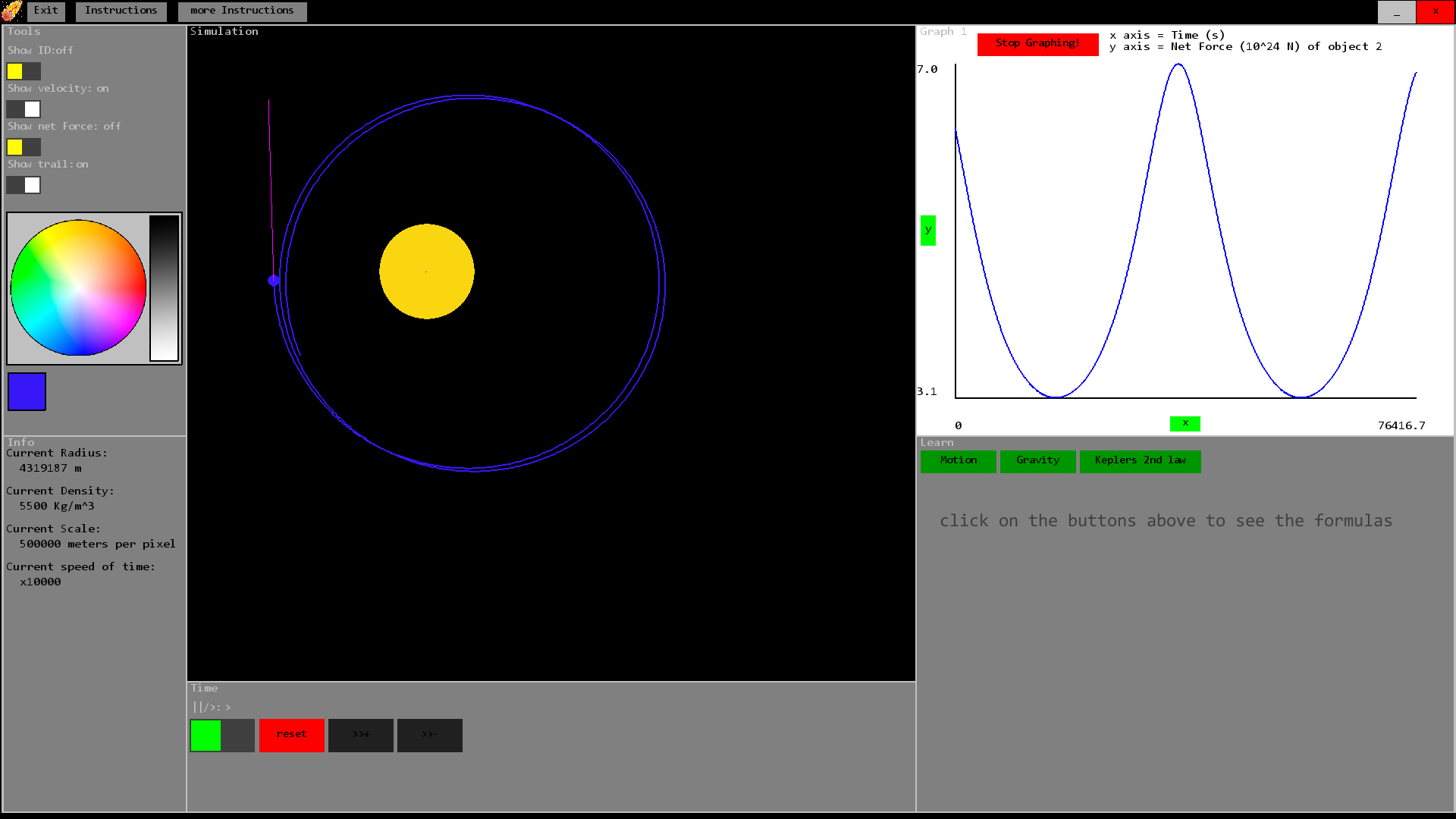
### Changing the y axis of the graph plotter

This screenshot shows a setup of a small blue object orbiting a large yellow object. The user has clicked change y axis to change the object or the variable for the y axis.



### Drawing the graph of the net force against time:

This screen shows the graph of force acting on object 2 against time.



# Testing

## Functionality testing

### Testing the physics

To test the accuracy of the simulation I will have to see if it correctly demonstrates the physics principles that I am trying to represent. The following list shows what I will be testing in this section.

1. Objects attract – Due to gravity any number of objects will accelerate towards the combined center of gravity
2. Shape of orbit – This is a clear indicator that the model for gravity is correct
   1. The shape should be elliptical or circular
   2. The graph of the velocity in one direction should make a form of sine wave. This will be a standard sine wave for a perfect circle but since most objects are at least slightly elliptical either the peaks will be squashed and the troughs stretched or the peaks will be stretched and the troughs squashed depending on the direction of the orbit.
   3. The area of the sector swept out by the path of one object orbiting another remains constant for equal time intervals. I can test this using the Kepler’s second law feature that demonstrates this principle.
3. Conservation of momentum – When two objects (both with 0 initial velocities) are added into the simulation, they should have 0 final velocity when they have collided. This is due to the conservation of momentum which states that in a closed system, the total momentum remains constant. To test this I can use the graphing feature to see how fast an object is moving and prove that the objects both had velocities of 0 at the start and after the collision the combined object has velocity 0.

To see the evidence for these tests please see video 1 at the web address:

<https://www.youtube.com/watch?v=AvtnBNN2HIU>

### Testing the file management

This section will be testing how the program loads and saves files in the program folder The following list shows what I will be testing in this section.

This section only contains tests for bugs, efficiency and reliability not for completing the objectives. None the less, this is still an important area to test. These were not included in the objectives since the objectives are reliant on these tests indirectly.

1. Saving and opening simulations
   1. This test will make sure that the program saves the files correctly
   2. This will show the format of the saved simulation file
2. Loading images – The program folder contains images that are used in the program. This test will make sure that they are loaded correctly
3. Saving Graphs – When the user stops the graphing, the program should save an image of that graph in the program folder.

To see the evidence for these tests please see video 2 at the web address:

<https://www.youtube.com/watch?v=PNO6idKms9E>

### Testing the GUI

This section only contains tests for bugs, efficiency and reliability not for completing the objectives. None the less, this is still an important area to test. These were not included in the objectives since the objectives are reliant on these tests indirectly.

This section will make sure that panels are displayed correctly, buttons and switches work and others. The following list shows what I will be testing in this section.

1. Title bar – This should contain a logo in the top left corner, a button to minimise and a button to close the program
2. Panels – Test used to make sure that the panels are displayed correctly for different screen sizes
3. The buttons function properly – This will test if the buttons are displayed correctly as well as if they work.
4. The switches function properly – This will ensure that switches are displayed correctly and test that they work

To see the evidence for these tests please see video 3 at the web address:

<https://www.youtube.com/watch?v=MxdmQuSJVj8>

## Feature testing

This section contains test for most of the objectives

### Panels

This test will involve making sure that all of the panels that I set out to include are in the final program

1. Tool bar – This should contain switches to toggle features (e.g. trail) in the simulation
2. Time settings – Buttons are used here to affect how time flows in the simulation
3. Graph plotter – Allows the user to draw graphs of the motion of the objects
4. Simulation display – The main screen to watch the objects in the simulation
5. Information panel – The panel that gives information about the simulation
6. Learn panel – The panel that teaches the basic physics principles that are used in the simulation

### Simulation features

This test looks at the different features that are included in the main simulation

1. Add new objects of different densities, volume, position and velocity
2. View the stats of each object visually by:
   * 1. showing velocity by a line of a certain size and direction
     2. showing the ID of an object on top of that object
     3. showing the force acting on the object by a line of certain size and direction
     4. showing the previous positions of the object with a trail that it makes as it moves
3. Affect the speed that the simulation runs using:
   1. Pause/play
   2. Increase speed of time
   3. Decrease speed of time

### Graph testing

This test makes sure that all graphs work.

1. Velocity in x and y direction
2. Net force
3. Time

### Extension testing

This test makes sure that the extension objectives that I have included are complete

1. Instructions are included
2. The user can change the colour of the next object
3. The user can save graphs that they draw

To see the evidence for these feature tests, please see the video at the following web address:

<https://www.youtube.com/watch?v=-ERTcPO1KB0>

# Evaluation

## Completeness

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Objective**  **number** | **Objective name** | **Complete (y/n)** | **Test to prove it** | **Video**  **(these tests can be seen in the following videos)** | **Line in technical solution** |
| 1a | Tool Bar | Y | Feature testing: Panels – 1 | 4 | 314 – created  420-426 – put switches in |
| 1b | Time Settings | Y | Feature testing: Panels – 2 | 4 | 324 – created  368-371 – put in buttons |
| 1c | Graph plotter | Y | Feature testing: Panels – 3 | 4 | 333 – created  948-1486 – plotting graph |
| 1d | Simulation display | Y | Feature testing: Panels – 4 | 4 | 322 – created  750-918 - simulating |
| 1e | Information panel | Y | Feature testing: Panels – 5 | 4 | 320 – created  1660-1678 – text displayed |
| 1f | Learn Panel | Y | Feature testing: Panels – 6 | 4 | 337 – created  381-387 – put in buttons |
| 2a | Add new objects of different densities, volume, position and velocity | Y | Feature testing: simulation – 1 | 4 | 883-918 – allows objects to be added in |
| 2b | Show an objects stats visually. | Y | Feature testing: simulation – 2 | 4 | 839-881 – showing trail, velocity and forces. |
| 2c | Allow the user to pause, play, increase and decrease speed of time | Y | Feature testing: simulation – 3 | 4 | Button/switch created at (369, 371, 418) |
| 3a | Calculates force acting on each object | Y | Physics testing 1 and 2 | 1 | 774 |
| 3b | Calculates the acceleration of each object | Y | Physics testing 1 and 2 | 1 | 778 |
| 3c | Calculates the velocity of each object | Y | Physics testing 1 and 2 | 1 | 786-789 |
| 3d | Calculates the position of each object | Y | Physics testing 1 and 2 | 1 | Method defined at 287-295  Method called at 843 |
| 3e | Masses of collided objects combine | Y | Physics testing 3 | 1 | 806 and 823 |
| 3f | Calculates the final density of each collided object. | Y | Physics testing 3 | 1 | 797-802 |
| 3g | All vector calculations are done in 2 space dimensions | Y | Physics testing 1, 2 and 3 | 1 | Nested loops 753 (outer loop) and 756 (inner loop) |
| 3h | Colliding objects conserve momentum | Y | Physics testing 3 | 1 | 760-763 |
| 4a | Graphs velocity x and y | Y | Feature testing: graph – 1 | 4 | 1064-1083 and 1150-1169 – setting axis to velocity.  1300 adding velocity to the list of data points |
| 4b | Graphs net force | Y | Feature testing: graph – 2 | 4 | 1095-1102 and 1181-1188 – setting axis to force.  1308 adding Force to the list of data points |
| 4c | Graphs time | Y | Feature testing: graph – 3 | 4 | 1086-1092 and 1172-1178 – changing axis to time.  1338 adding time period to the total time elapsed. |
| **Extension number** | **Extension name** | **Complete (y/n)** | **Test to prove it** |  |  |
| 1 | Instructions for the simulation | Y | Feature testing: extension – 1 | 4 | 1767 - 1917 |
| 2 | Coloured objects | Y | Feature testing: extension – 2 | 4 | 1630-1655 – using colour wheel |
| 3 | Saving graphs | Y | Feature testing: extension – 3 | 4 | 1425-1435 – graph being saved |

Table 3

**Student feedback (end user)**

When asking students who tested the program I received quite positive feedback. Students said that it was very user friendly and the instructions within the program were useful in helping to understand how to use it. It was also described as being a useful educational tool. This is great feedback to hear since the point of the project was that it was an educational tool. The students also liked how it allowed them to visualise the ideas they had been taught in their physics classes and it helped them understand Newtonian gravity better. Students also said that the graph drawing feature was great for learning about circular motion since the motion of an object in a circle can be described by sine waves. Possible improvements that were suggested was to be able to re-size the panels.

**Teacher feedback (client)**

I also spoke with physics teachers about the project. They also liked the program and one of the teachers asked for a copy of the program to use in class.

This feedback has been helpful to me showing that the project has been successful in being an educational tool for a-level physics students learning about circular motion and gravity. Teachers agree that it would be useful for in-class activities and students have also said that it would be useful to them too.

## Improvements and expansions

The program is successful in that it has completed the objectives I set at the start. However, there are still improvements that could be made.

For example, I could make it 3D instead of 2D. If I were to make the 3D graphics from scratch within the Pygame module I would need to do some research into how perspective works on a 2D computer screen. Alternatively, I could use an existing 3D graphics engine. This would require a different graphics module to Pygame or even just making it in a 3D engine outside of python such as <https://unity3d.com/> . This would need to be combined with a different programming language such as JavaScript or C#. Whether or not I use an existing graphics engine, I would still need to make calculations in 3D. When I calculate vector measures, I must repeat the process again for two dimensions, so for 3D I would have to repeat the calculations once more. In the program I also have certain object attributes that are dimension specific such as posX and posY. For all these dimension specific variables I would need equivalent attributes for the ‘Z coordinates’ e.g. posZ. Once all vectors are calculated, I would just insert the positions of the objects into the graphics engine or my own function and it would draw the objects in those positions.

I could also implement a feature that lets you resize the panels and create more panels containing more graphs or simulation displays etc. The resizing could be done by changing the values of the widths and heights of the panels. When text is displayed of the panel I would need a scroll bar to hold all of the text

Another improvement that I could make would be to add different types of object. In earlier versions of the program I experimented with adding an image over the object to make it look like certain planets. This was problematic because it would severely slow down the speed when the user tried to scale the objects since transforming the image many times every second would be very inefficient compared to transforming a circle (which involved only adjusting the radii of the objects). However, I could have added the ability to make objects of different types. Instead of having just planets, I could have made stars, black holes, asteroids etc. These would have different properties such as emitting light or having an irregular shape. With different types of objects, a database would be a useful to store saved simulations. Each type of object could be a new entity and the attributes would be the different properties mentioned earlier. I didn’t add this into the program because although educational and useful, it was not part of the gravity section in the physics a-level. However, an expansion of the project could be to make a series of applications for different simulations which would cover the different topics that did involve stars, blackholes, asteroids etc.