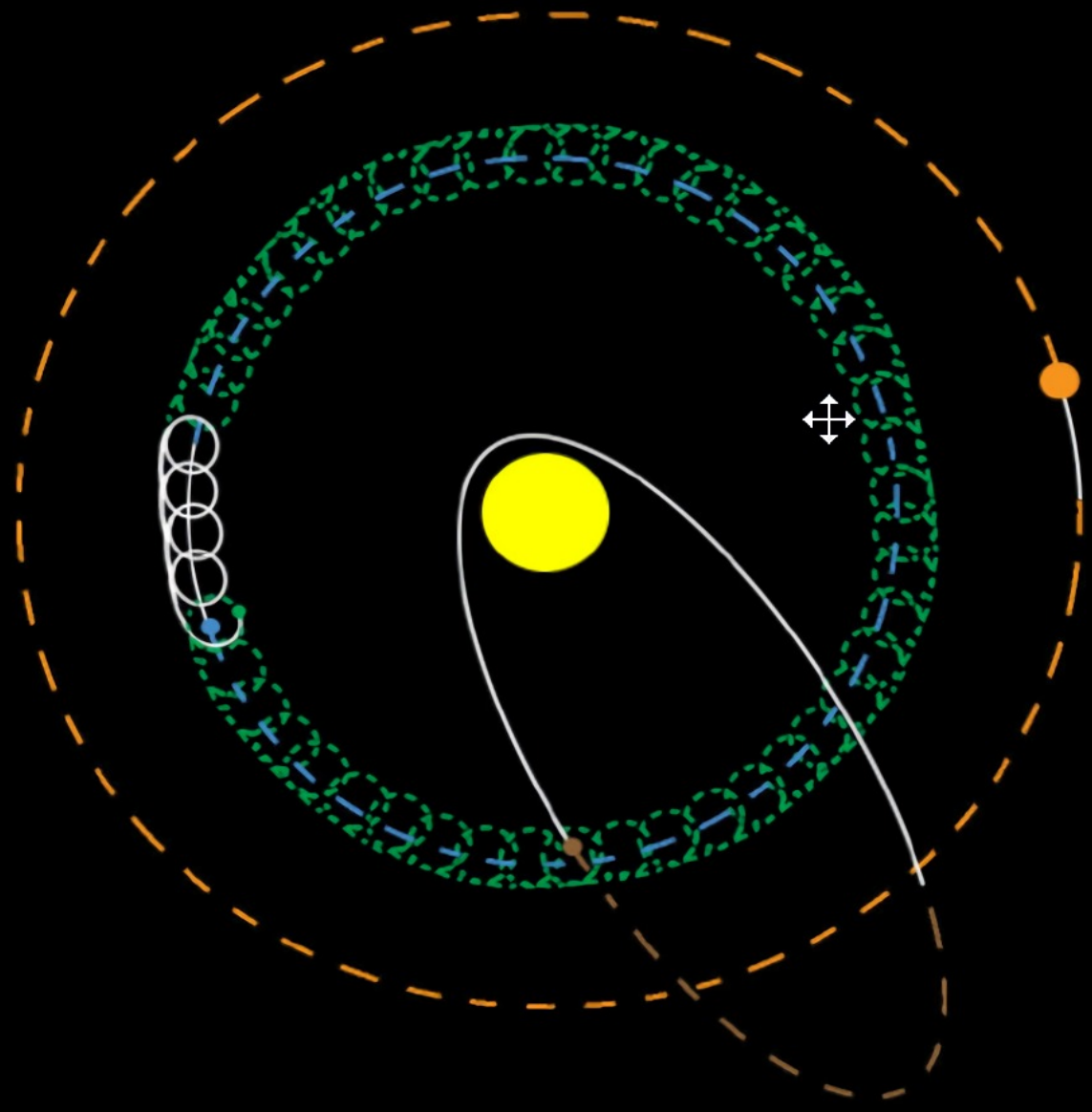
Computing Coursework



Python Celesital Simulation

Contents

Analysis

Introduction

In this project I will be addressing and fixing the issues behind typical 2D educational space simulators by creating my own for my school's Physics department. 2D simulations are used in both educational software and gaming software ranging from programs such as Conway's The Game of Life[1], a cellular simulation produced in 1970 all the way to the project that I am creating for the simulation of planetary objects such as solar systems and galaxies.

The project seems to have some challenging aspects including but not limited to optimising efficiency for low end hardware and having high enough accuracy for educational use in places such as Secondary schools and Universities.

Background research

Bingley Grammar School is an institution teaching upwards of 2000 students.
Our Physics department would like a program to help visualise customizable solar systems for the lower school's pupils. Teachers have tried using online flash based systems[2] [3] however find them lacking for a teaching environment due to the absence of some key features.

My goal with this project is to make a similar product to those mentioned before but with added features and is also open source. With the limited time I have I plan to incorporate all features that are needed and only some that are not essential to the running of the program.

End users

To design a program to a user's need I will interview potential end users to gather what the essential requirements are for this system. Generally speaking, this program is designed for students however can be used by anyone who has any interest in planetary interactions . For that reason I have called upon 2 teachers in the physics department and 2 students who are studying A-Level physics to be my interview targets as they are intended to be the direct end users of this system.

---

[1] **"Conway's Game of Life - Wikipedia." https://en.wikipedia.org/wiki/Conway's_Game_of_Life. Accessed 13 Mar. 2017.**
[2] **"Gravity - Hermann.is." https://hermann.is/gravity/. Accessed 13 Mar. 2017.**
[3] **"Gravity - nowykurier.com" http://www.nowykurier.com/toys/gravity/gravity.html. Accessed 13 Mar. 2017.**

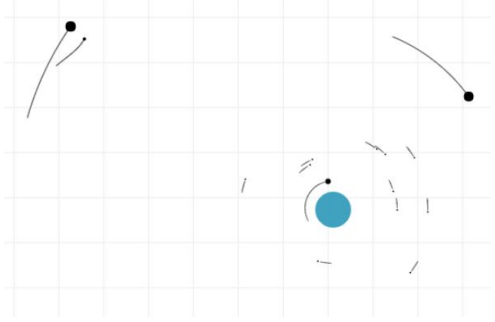
Similar systems

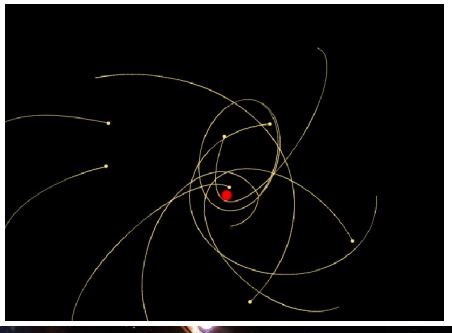
Previously mentioned websites such as hermann[4] and nowykurier[5] are online simulators that are written in either javascript or adobe flash.
This and the large absence of features are what call for this system to be implemented. The absence of features such as saving, undoing and pausing really restricts the educational side of these programs.
A lightweight system with these extra features with the ability to be built upon would be a perfect solution to the main issue that these tools don't solve.

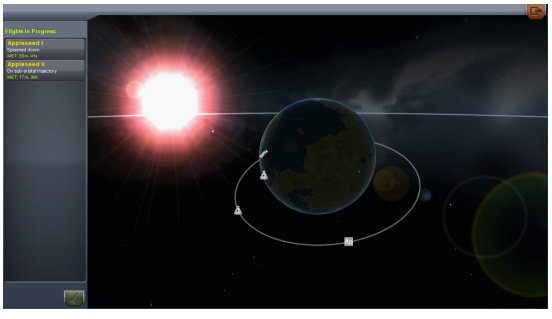
Part of the reason that these sites are so popular is how attractive the ui and output is. This is something worth aiming to do when producing this system as the better the output, the more useful the system is and the more pretty the output, the more interested students will inevitably be in the lesson.

Other similar systems include full fledged 3D simulators such as Kerbal space program[6] and Elite: Dangerous[7]. These system boast beautiful 3D graphics and hours of gameplay and fun whilst still being at least somewhat accurate to the laws of physics.
The two major issues around these being inspiration for this project is the fact that although they are accurate enough to be used in this fashion they are still games in the end and the fact that 3D simulation is way out of the timeframe for this project.

I aim to combine both the fun and attractive graphics of the 2 3D games and the educational and scientific output of the simulations to create a fun to use teaching tool for teachers in our school.







[4] "Gravity - Hermann.is." https://hermann.is/gravity/. Accessed 4 May. 2017.
[5] "Gravity - nowykurier.com - Nowy Kurier." http://www.nowykurier.com/toys/gravity/gravity.html. Accessed 4 May. 2017.
[6] "Kerbal Space Program." https://kerbalspaceprogram.com/. Accessed 4 May. 2017.
[7] "Elite Dangerous." https://www.elitedangerous.com/. Accessed 4 May. 2017.

8 9

Interviews

Teachers interview

Interviewees: Mrs Sheffield and Mr Bashley

Q: What is your current solution of your problem of teaching planetary interaction to students?

A: “Our current system regarding teaching about the solar system is to use physical props, diagrams and artists illustrations to explain how the solar system is and came to be. Alongside this, we have been experimenting with online simulators but they simply don’t have the tools we need to deliver the course content.”

E: The use of physical props only allows a very static kind of lesson as it doesn’t allow much changes. This shows the need for a variable system such as the one I have proposed. This approach may lead to those who have a more visual learning method to be left out due to the need for imagination.

Q: What “tools” do these online programs not provide?

A: “These online programs are simply more of a game rather than an education tool. They are missing things such as variable path lengths, lack of “load” and ”save” functions and also don't provide a way of pausing their output to allow explanations of what is going on. Alongside this, many programs don't allow colour coding of each planet which causes some orbits to be unrecognisable in cramped scenarios. ”

E: This answer gives me a lot of insight in what “unique selling points” a newly created program could have. In essence, the program should aim to hit all these as core objectives as this is the main reason why this program should be developed so far from what I’ve seen.

Q: What is the draw backs of the current “physical” system?

A: “The drawbacks of our system is that there is relatively little customizability in this section of the course. Many students cannot or refuse to recognize and be willing to learn about what many consider to be a rather boring subject.”

[8] "Kerbal Space Program Screenshots - Pics about space." https://pics-about-space.com/kerbal-space-program-screenshots?p=1. Accessed 4 May. 2017.
[9] "The Weekly 10 - Amazing Elite Dangerous Screenshots | Elite ...." https://community.elitedangerous.com/en/node/223. Accessed 4 May. 2017.

E: Making a fun, simple and visually stimulating system could help make the subject more interesting for student that are not particularly fond of what they are forced to learn. This is a strong objective as it directly colorates people's grades to something that

Q: Do you believe that a computer system can fix the issues of your current system?

A: "A well designed computer system would easily solve most of our problems regarding this subject."

E: This confirms my original belief that a computer system would be

Q: What features do you *require* a system to do?

A: "Well it would need to be at least somewhat accurate, which is a given. Secondly, It would need to be simple enough for the kids to understand its "output" as per say; a system without a simple output would be useless for the teaching environment it is aiming for and third, we would like it to cover the other points we have covered in this interview."

E: This shows the need for a nice user interface and a clean output. Some time should be dedicated to make a interface as simple yet powerful as possible to maximise the program's usefulness.

Q: What extra features would you like the system to do?

A: "Something along the lines of Pre-created solar systems and the ability to pause and 'set up' systems per se would be a great extra feature that comes to mind."

E: These points could be easily implemented and will add a great deal of ability to the program.

Student interview

Interviewees: Physics students Regan and James

Q: In your opinion, for a physics based simulation, what are the most important features?

A: “Accuracy is key however, if the information isn’t interpretable then what’s the point of running it in the first place?”

E: Reiterated what the teachers had said and it clearly seems very important to everyone using the program.

Q: How much good would a simulation do for your and lower school students studies?

A: “I think we and moreso the lower school will do great for our studies whilst adding a little joy to our boring Physics lessons.”

E: Confirmed what the teachers had suspicions about.

Q: What in your opinion would be some suitable additional features that would make the system work better for you?

A: “Planets exploding on impact sounds pretty cool to us. It would allow us to view the carnage that collisions would create in a planetary system”

E: This seems like it would be a stretch but also would increase the system's use case (use for impact simulation for example) but looks very out of scope for this kind of project.

Objectives:

From these interviews I have deduced a set of “Core” and “extension” objectives for me to complete. Core objectives being the objectives that are vital to the running of the program and made up of required properties that have been asked for of the system. Whereas “extension” being the objectives that I’d like to hit but is not essential for the system functionality but instead are more quality of life features.

Core Objectives:

- Accurate representation of 2D physics.
    - Using A-level physics equations such as FGrav we can accurately determine an object's velocity, acceleration and position in the plane.
- Display useful information (such as maximum velocity, acceleration)
    - Display this information on a sidebar that lists all the objects or place this information on a toggle-able option.
- Customisable.
    - Custom colours, masses of objects, object velocities etc.
    - Ability to change trail duration.
- Little/minor performance issues.
    - Working with 1000+ particles will likely be taxing on the hardware.
    - I will need to experiment with different refresh rate and integration methods to ensure that the program is efficient.
- Present planet's systems with a load/save system.
    - Will help teachers download or create example systems for illustrating teaching points.
    - Allows introducing parameters into a system that might be hard to set up on a whim.
- Play and pause system to allow accurate physics set ups.
    - Allows easy setup for systems for both teachers and students alike.
    - Lets teachers have chance to explain what is happening at a single point of time.

Extension objectives:

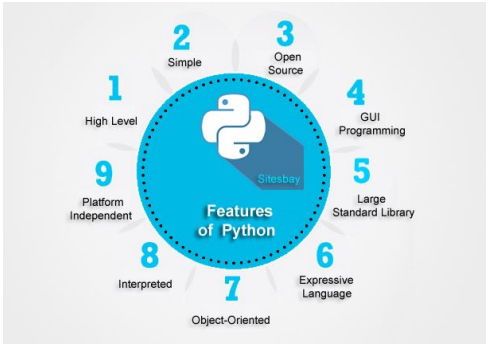
Alongside this, I decided on a set of extension objectives. These objectives may be hit but are heavily reliant on time and therefore are not the main focus of the project.

- Impact simulation.
    - Upon impact, objects will split into smaller masses and start affecting other objects with their own gravity.
- Auto orbit function.
    - Makes a planet automatically orbit the selected planet, allowing for easy creations of solar systems.
- Visualisation of forces on an object.
    - Shows why an object moves in the way that is does.
- Undo and redo function for ease of use.
    - Will require a stack and some design work to ensure that it functions correctly.

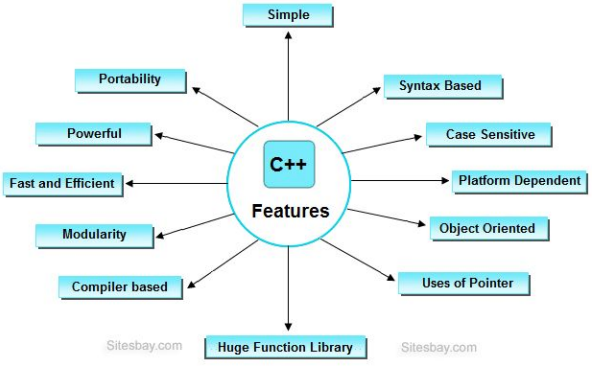
Solutions

First of all I must decide what language and libraries to use for the program:

Python vs other programming languages



10



11

There are many reasons to use python over languages such as C++. These reasons include but are not limited to:

1. Python is my most fluent language; this allows me to get my program quickly off the ground and quickly produce a working system. Alongside this, there will be no need to compile which will further reduce time spent programming.
2. Python scripts are usually <50kb which allows for easy transfer around an environment such as school (such as distribution by email).
3. Python scripts do not need to be installed and therefore do not require an administrator to install this program on every computer and instead can be distributed and updated on the fly.
4. Multiple GUI modules such as Tkinter and PYGTK are available in the default install of Python 3
5. More secure than distributing an executable that could potentially be manipulated or pose other security risks.

This is faster and less bloated than a flash/web based versions and possibly also compiled C binaries on our school's system and also allows for code modification for each potential use case throughout each different course. For example, the "classical mechanics" course taught at A-level

[10] **"Features of Python -Sitesbay." http://www.sitesbay.com/python/images/features-of-python.png. Accessed 16 Mar. 2017.**

[11] **"Features of C++ - Sitesbay." http://www.sitesbay.com/cpp/features-of-cpp. Accessed 16 Mar. 2017.**

has a different gravitational constant value compared to the A-Level physics course which has a far more accurate gravitational constant (6.81 vs 6.8).

A program with a suitable GUI, simplistic yet powerful controls and strong reliability with little to no performance drain would definitely be the best possible solution to the problem.

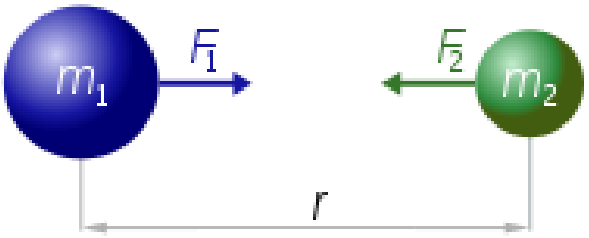
Libraries

One pressing issue is what GUI library to use for this program. Most of the time a library such as PyGame is the best solution for animation based software. However, PyGame and many others are not included in the default install of Python 3 which is essential for the portability aspect of the software. This in turn means that it is unsuitable for use in our case and instead the best solution is the Tkinter library.

Research

The first thing that I did was research into the required physics and mathematical equations required to create a program of this caliber.

First of all I researched into the newtonian gravity equation that is noted as :



$$F_1 = F_2 = G\frac{m_1 \times m_2}{r^2}$$

[12]

Based on this and Newton's law of universal gravitation, we can deduce that a planet's acceleration is:

(Mass 1 being the object that is moving and mass 2 being the object that is attracting it.)

((6.678) Mass1*mass2 / R**2) / Mass1
Due to F = Ma and therefore a = F/M[13]

Combining this and basic mathematics such as trigonometry along with a = f/m, we can conclude the Vx and Vy from out equation.

---

[12] **"Newton's law of universal gravitation - Wikipedia." https://en.wikipedia.org/wiki/Newton's_law_of_universal_gravitation. Accessed 16 Mar. 2017.**
[13] **"Newton's laws of motion - Wikipedia." https://en.wikipedia.org/wiki/Newton's_laws_of_motion. Accessed 16 Mar. 2017.**

Running this through a loop that does this to every planet except itself, we can figure out the new location by adding the Vx and Vy to its current x and y positions.

N Body problem:

In orbital mechanics, often mathematicians and physicians have struggled to compute the gravitational force between more that 2 masses have not been accurately assessed. This is called the n-body problem

The wikipedia article on n-body problem defines the issue as simply:

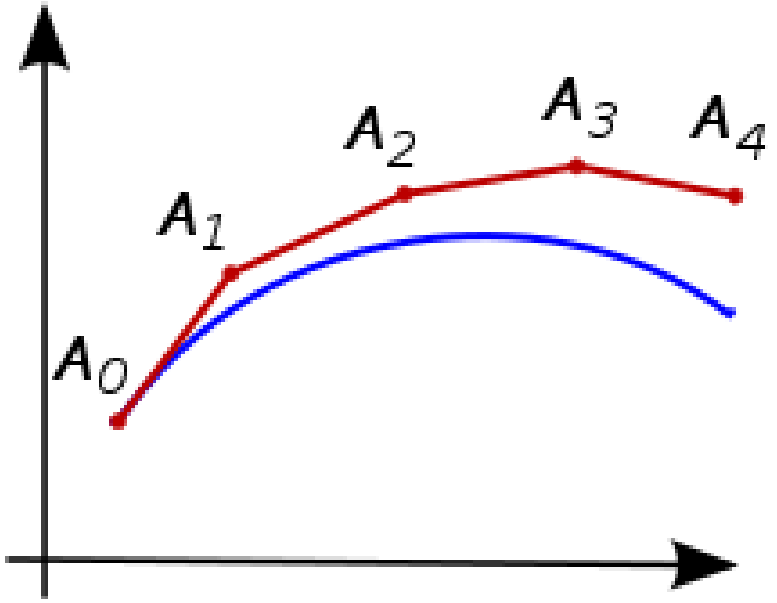
"Given the quasi-steady orbital properties (*instantaneous position, velocity and time*) *of a group of celestial bodies, predict their interactive forces; and consequently, predict their true orbital motions for all future times."*[14]

The main problem with this is that you can never find out the future position using a single equation for more than 2 objects. This is caused by highly chaotic behaviour which will not allow accurate representation through simple algebraic expression. Although many proposed solutions to this problem have been created[15] [16] through the means of creating "groups" of planets to be computed as one whole object, it is likely out of the scope for this kind of project considering my limited knowledge of physics outside of basic classical mechanics.

This therefore requires a more inefficient but far more plausible approach that is simply "brute forcing" using a heuristic method of computing every object with every other object. This solution far from perfect but it is actually doable in both mathematical terms and in time restricted terms for this scope of a project.

[14] "n-body problem - Wikipedia." https://en.wikipedia.org/wiki/N-body_problem. Accessed 29 Mar. 2017.
[15] "Protovis - N-Body Problem." http://mbostock.github.io/protovis/ex/nbody.html. Accessed 29 Mar. 2017.
[16] "N-body Gravity Simulation." http://seenjs.io/demo-gravity.html. Accessed 29 Mar. 2017.

Euler integration:



[17]

(Euler integration estimating a curve)

Euler integration is the default option on the program. This algorithm is very light but quite inaccurate as shown by the example on the page on[18]. A brief description of Euler is:

Previous Velocity + Newly added velocity = current velocity.

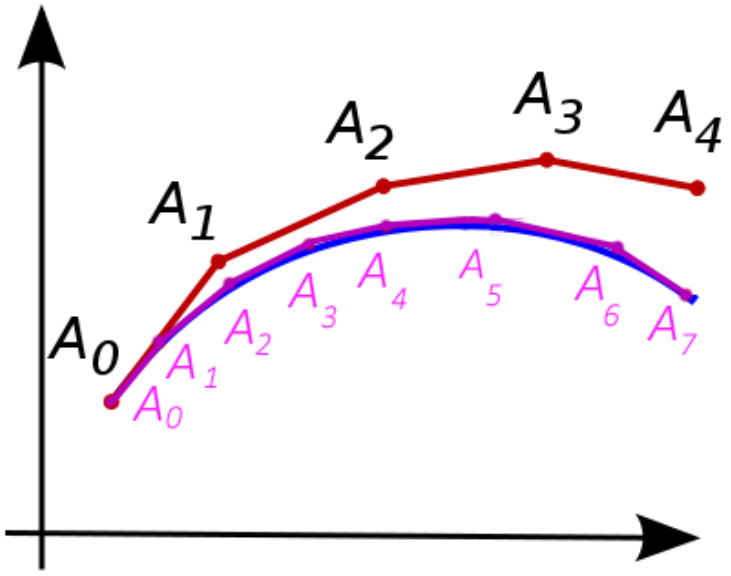
Simply put, you add velocity to the old velocity to find the new velocity. This allows very simple but inaccurate integration.

Although this produces somewhat inaccurate results this solution is rather useful as it is very simple and requires very little computing overhead. This allows the program to run alot faster in sacrifice of some minor imperfection of orbits.

---

[17] **"Euler method - Wikipedia." https://en.wikipedia.org/wiki/Euler_method. Accessed 16 Mar. 2017.**
[18] **"Integration by Example - Euler vs Verlet vs Runge-Kutta - Codeflow." 28 Aug. 2010, http://codeflow.org/entries/2010/aug/28/integration-by-example-euler-vs-verlet-vs-runge-kutta/. Accessed 16 Mar. 2017.**

Euler integration 8x:



[19]

The idea behind the Euler 8x integration method is that you divide the forces by 8 so that the planet must go through 8x the processing to cover the same distance but however is far more accurate.

Previous velocity + ⅛(newly added velocity) = current velocity

This is more accurate as the number of calculations are essentially multiplied by the number of strips that have been taken from the movement. In basic mathematics this is often used in something called the trapezium rule[20] which you use to approximate curves.

After my research I concluded that I should use Euler integration both because it is the most simple to implement and also the least computing overhead.

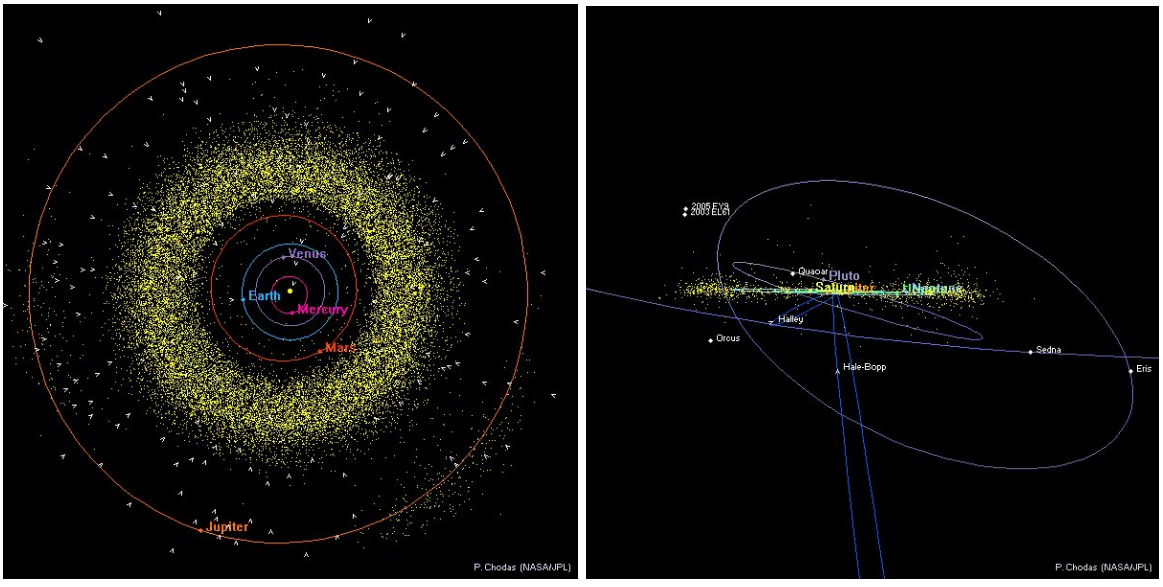
---

[19] **"Euler method - Wikipedia." https://en.wikipedia.org/wiki/Euler_method. Accessed 16 Mar. 2017.**
[20] "Trapezoidal rule - Wikipedia." https://en.wikipedia.org/wiki/Trapezoidal_rule. Accessed 6 Apr. 2017.

Acceptable limitations

This program is to be created in a small time period (50> hours) and therefore unfortunately means that I will have less opportunities to implement more complex objectives. However, these can be added later due to the modular structures of python and the use of object orientation and functions.
Some objectives such as splitting planets on impact may be too much of a stretch to hit. However, the core things that the user needs will definitely be included no matter what time constraints as long as it does not take away from the final product.

A limitation which I foresee being an large issue is small integration errors such as slightly inaccurate equations and minor rounding errors. These could cause perfect orbits to be hard to achieve as integration is usually not always 100% accurate.



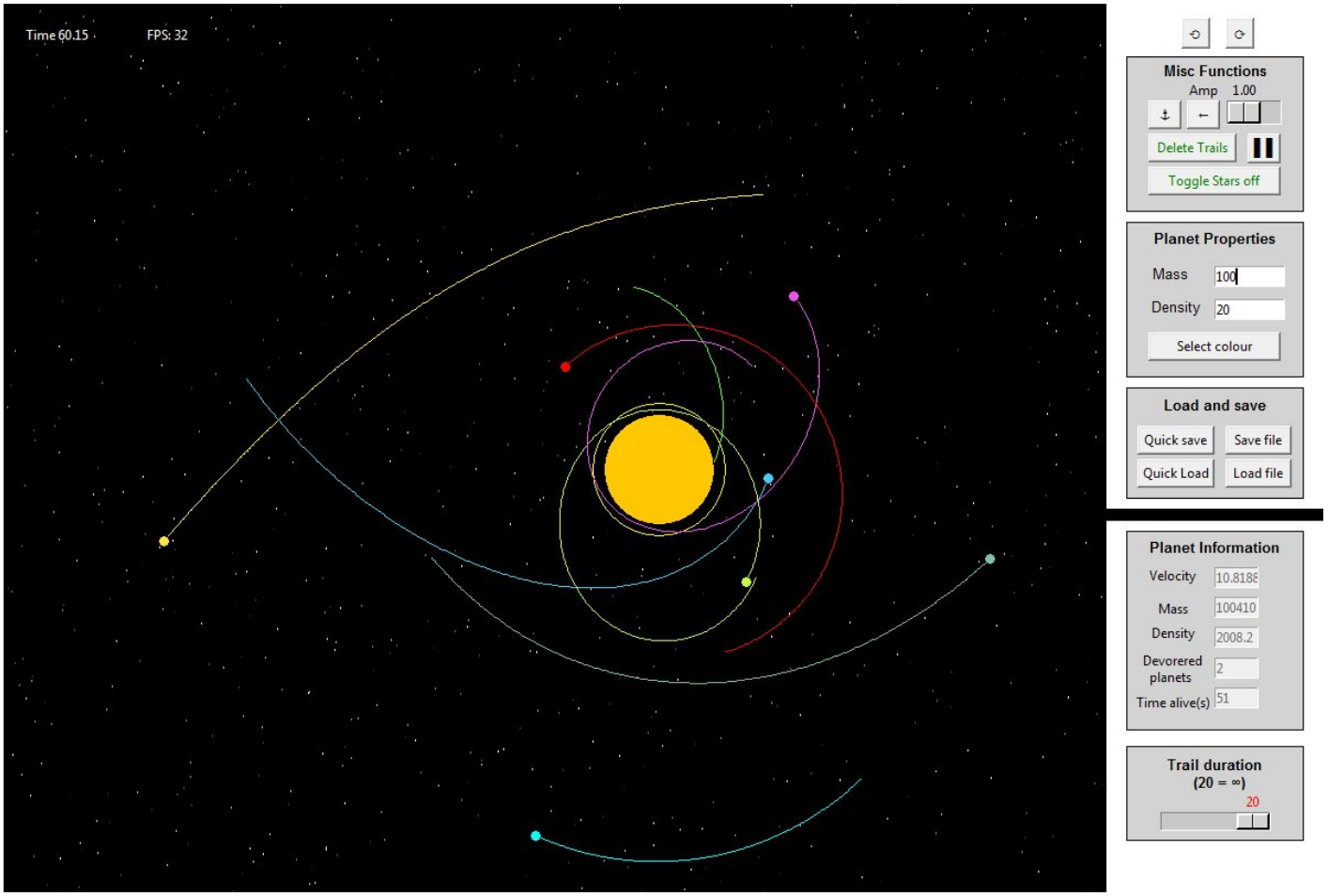
[21]

Mapping a lot of objects will likely bring a lot of bring a lot of performance issues even with the most efficient systems therefore it is a high priority to ensure that performance can be maximised on most systems. Objectives such as variable trail duration are vital to ensure acceptable performance on upmost computers.

[21] "Planetary orbits."http://cseligman.com/text/sky/orbits.htm. Accessed 16 Mar. 2017.

This is an issue which is very hard to avoid given the power of the computers that the staff use at our school alongside performance limitation in many of python's GUI libraries.

Documented Design



(Final product for show)

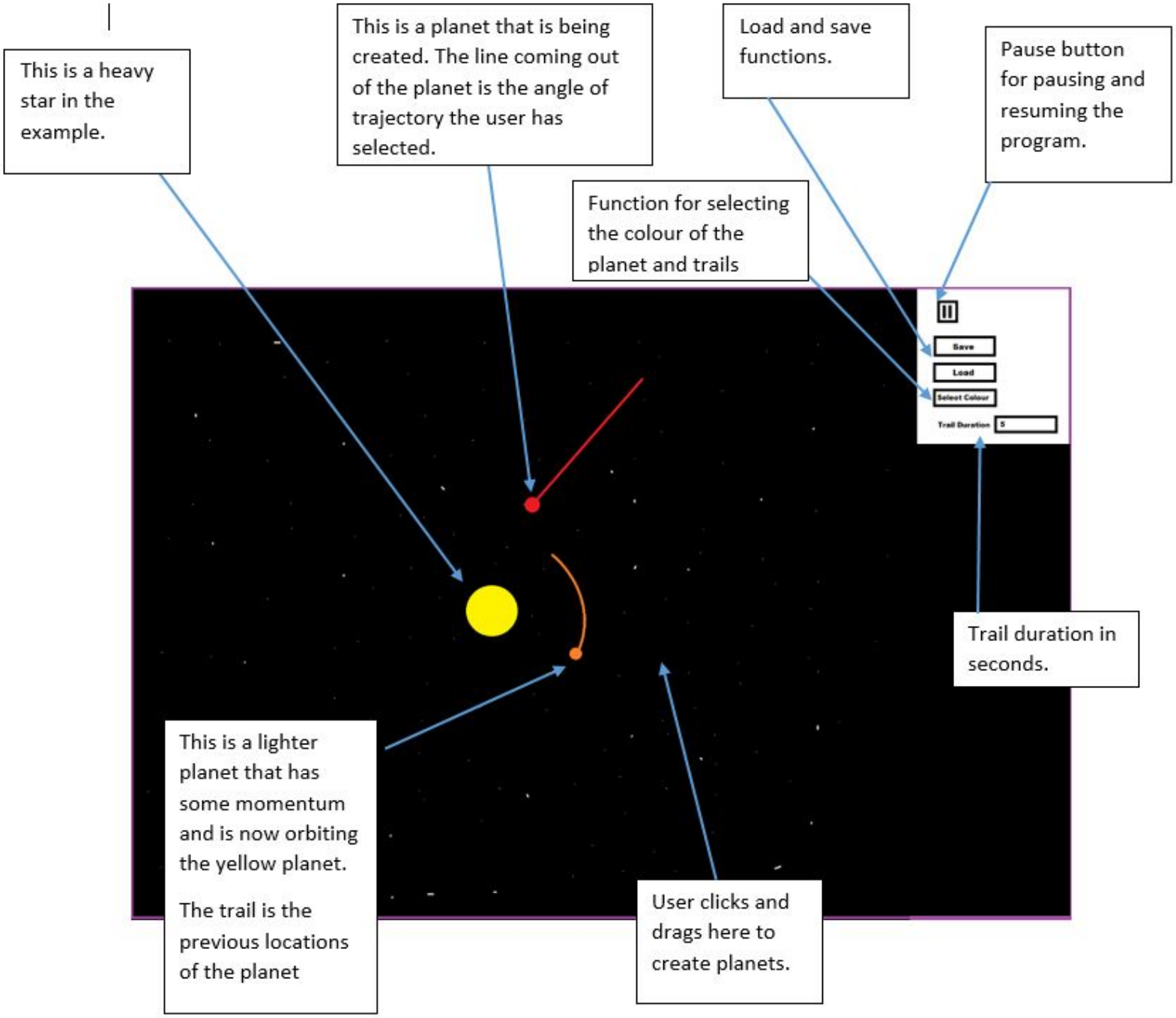
Introduction

The program I am designing is a 2D planetary gravity simulator in which the user can place and interact with planets and the program will show how they should interact with each other. Trails will be drawn and nearly everything can be customised to the users delight. This brings forward the idea of a single UI with plenty of functions for use in just one window. The aim of this is to cause as little confusion to the overall

usage of the application as possible and therefore require little explanation to get the user to understand how to build a workflow.

UI design

Main window:

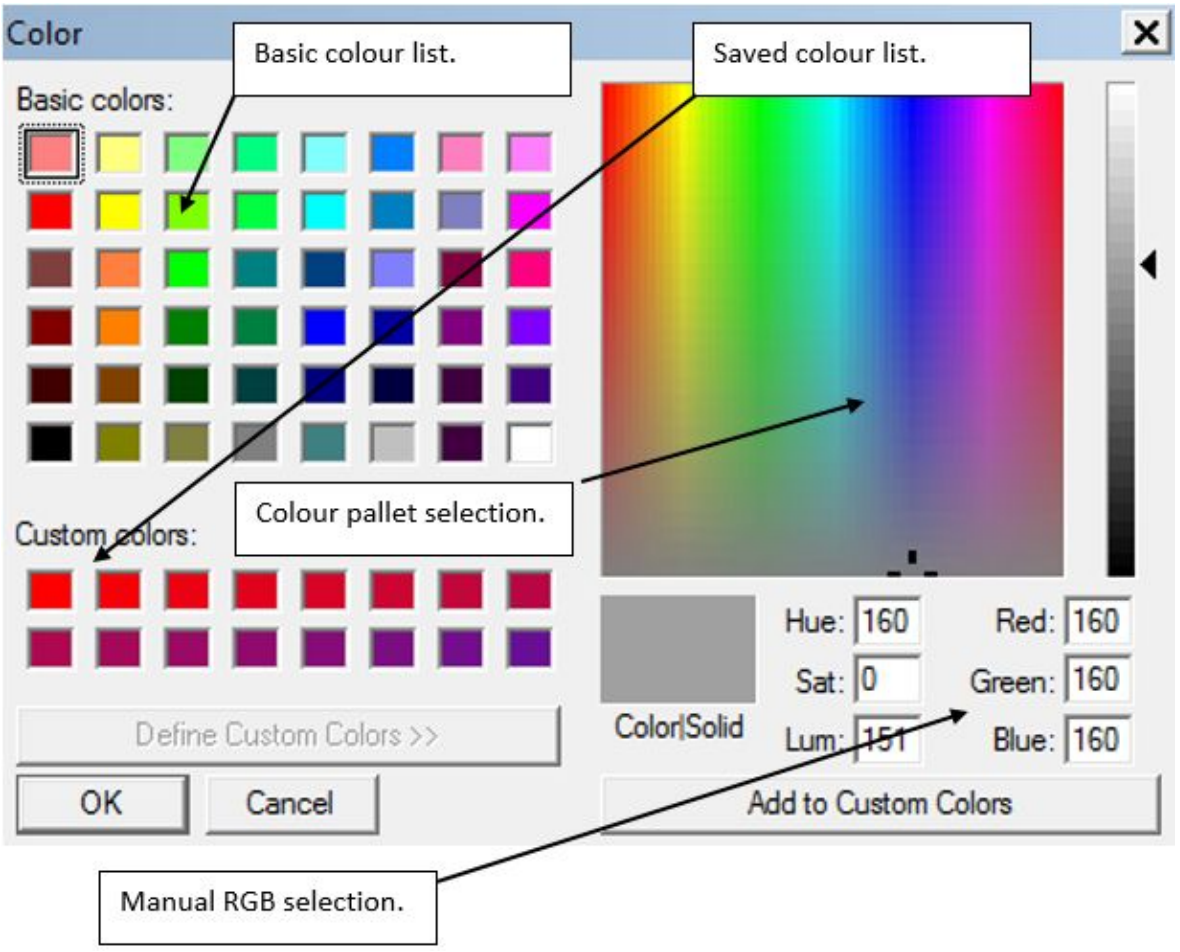


Tkinter based windows

Colour select

```
01. ui.planetcolour = askcolor()
```

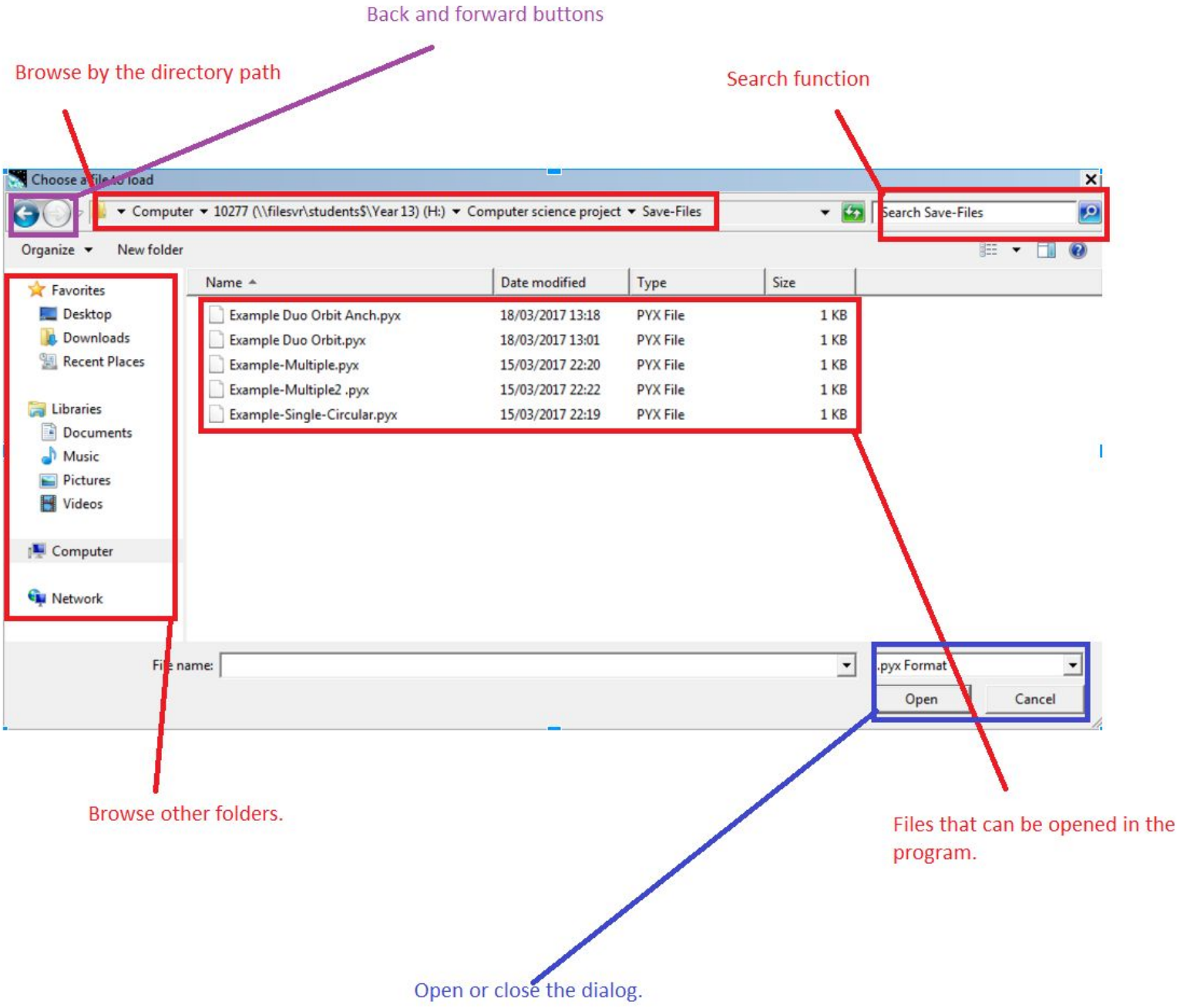
These windows uses a tkinter module to select colours so there is no need to design as it already exists.



Loading

Called using the line:

```
01. file = filedialog.askopenfilename(filetypes=[(".pyx Format","*.pyx")],title="Choose a file to load")
```



Saving

```
01. file = filedialog.asksaveasfile(filetypes=[(".pyx Format","*.pyx")],title="Choose a file to save",defaultextension=".gpy")
```

Technical solution

The core aspect of the program is split into subsections:

1. Creation of the UI
2. Creation of the planets
3. For loop to select every planet
4. A second for loop that selects every other planet
5. A function that moves the planets
6. A function that handles all the UI elements.

UI Creation

At the start of the program, the UI is initialised.

1. A 1200x1000 window is created.
2. Elements such as buttons are placed on the UI.
3. Timer starts.
4. User is handed control.

Creation of planets

The act of creating planets is a critical step.

1. User clicks on the window.
2. The ui elements representing the mass and density input is used to calculate size and gravitational influence.
3. The planet variables get appended to the array.
4. The planet is created on the UI.

Main for loop

This is the loop that selects each planet to calculate the forces that act upon that object.

1. Until every element has been selected run this loop.
2. If not go to 2nd for loop.

3. continue

Second for loop

In this loop, every other planet is selected and the forces are calculated by planet.

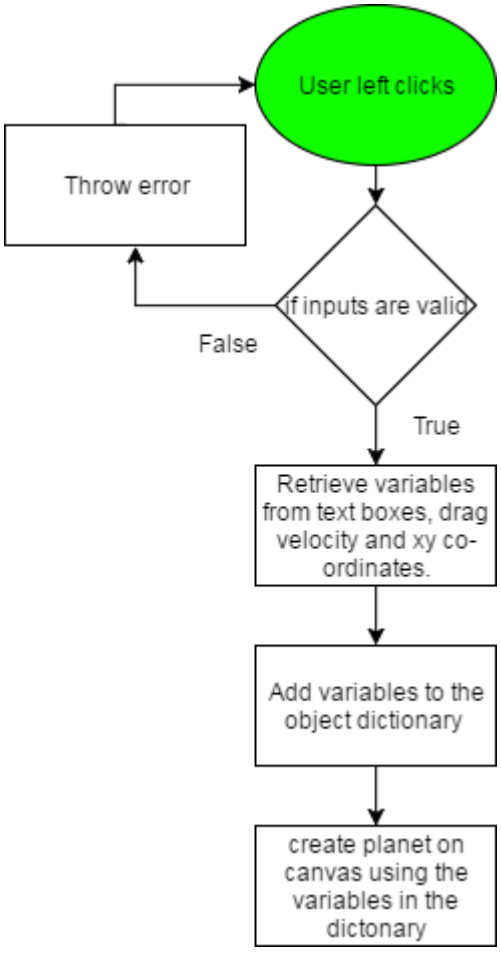
1. For loop to select every planet
2. If loop number is equal to the first planet then skip.
3. Calculate Fgrav between objects.
4. Add the result to the Vx and Vy of the object

Planet creation

Planet creation is one of the most simple functions inside this program.

All textbox defined variables are checked for invalidity before the inputs are accepted and used to create the planets.
First the variables are retrieved from the for mass text box, density text box and the planet colour variable. The radius of the object is then calculated from the statement mass/density.

The variables are combined with the x and y values of the click and added to the "objectdictionary" dictionary. That list is used to create the planet on the plane.



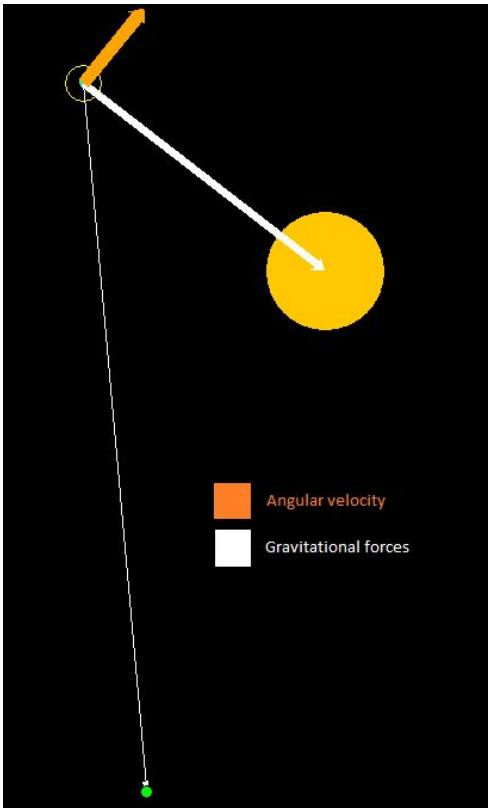
Planet manipulation

After right clicking the closest planet will be selected. The act of selecting a planet sets up the following functions:

- Anchor function

This adds the id of the planet to an array that are skipped from the main loop. This planet will apply force to other planets but not be moved by the other bodies. This exists as a solution to minor movements in large objects due to smaller object. EG: a solar system. Minor shifts in stars in reality have little effect on the planets orbits as the gravitational pull of those planets are near

nil. However, my program has exaggerated forces to allow for ease of use rather than full scientific usage. This means that the gravitational pull of planets are enough to throw orbits of their perfect elliptical ones. By anchoring a mass it allows for it to keep orbits similar to those in reality.



- Education mode

As terrible as the title of this function is; it enables the creation arrows that show how much force gravity is acting on an object. This can be used to show orbital forces that cause a planet to actually orbit.

To maintain a perfect orbit, velocity sideways must equal velocity caused by gravity with a net acceleration of 0m/s. This can be shown as an 90 degree angle between the two forces. The image shown on the left is the output from my program showing a near perfect orbit.

This was added after testing as the code was not copied between an earlier version of the code.

Physics Calculations

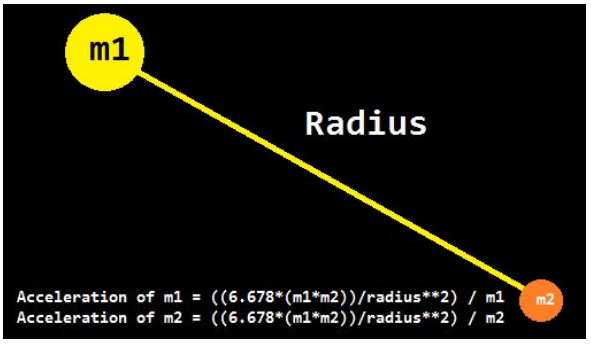
As stated in my analysis, the main equation that we will be using in this program is:
Force due to gravity = 6.871* (Mass1 + Mass2) / radius**2 [22]
Using this and the section on Maths calculations, we can determine the resultant forces on the planet caused by all the other planets.

[22] "Gravitational acceleration - Wikipedia." https://en.wikipedia.org/wiki/Gravitational_acceleration. Accessed 31 Mar. 2017.

Pseudo code

```
Function physics
    Radius,theta,xn,yn = maths(planet1,planet2)
    If radius is not 0 do
        Gravity = 6.871* (Mass1 + Mass2) / radius**2
        Accelerationx = gravity*cos(theta)
        Accelerationy = gravity*sin(theta)
If xn == True do
            Accelerationx = -(accelerationx)
        end
If yn == True do
    Accelerationy = -(accelerationy)
end
Return accelerationx,acceleorationy
    end
end
```

Visual



Python

```
def physics(planet1,planet2,objectxy,object2xy,OBD,speed):
    radiusbetweenplanets,theta,xn,yn = maths(objectxy,object2xy)
    if radiusbetweenplanets != 0:
        Fgrav = ((G*(int(OBD[planet1]["mass"])*(int(OBD[planet2]["mass"])))))/radiusbetweenplanets**2) / OBD[planet1]["mass"]
        accelerationx = Fgrav*math.cos(theta)
        accelerationy = Fgrav*math.sin(theta)
        if xn == True:
            accelerationx = -(accelerationx)
        if yn == True:
            accelerationy = -(accelerationy)

        cspeed = (speed.get()/1000)
        #Resolving (Right) (positive x)
        vx = accelerationx*cspeed #Mass is not used here because it results in a more "exaggerated" but more workable interactions.
        vy = accelerationy*cspeed #Essentially removes the need for larger numbers.
        #Resolving (Down) (positive y)
```

Maths Calculations

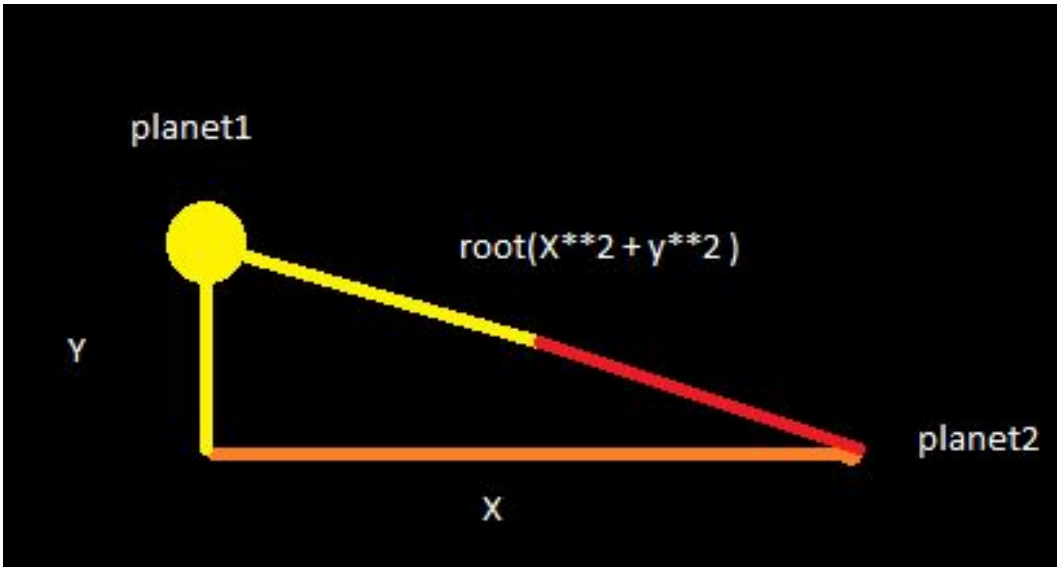
The maths module does some basic maths to calculate radius, angles and lengths based on coordinate geometry.

This module handles the following:

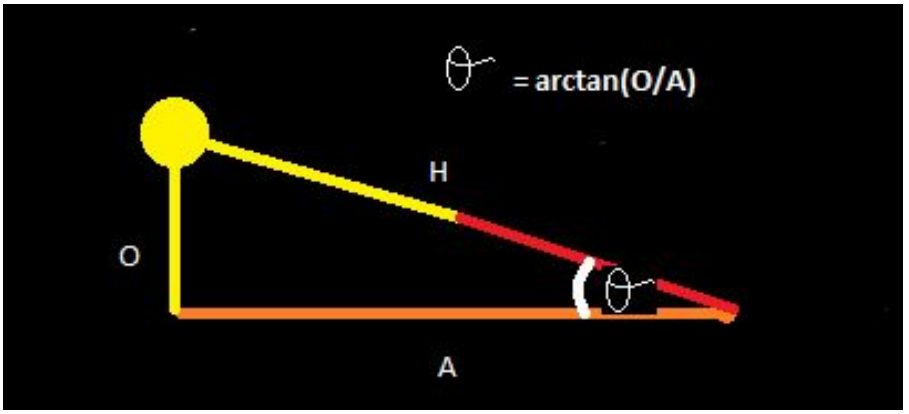
1. Calculating radius through pythagoras's theorem
2. Calculating the angle (theta) using trigonometry.

3. Calculate what direction the planet is traveling to append theta to.

Pythagoras function



Trig function



Negative function pseudo code

```
Xneg = False
Yneg = False
If changeiny <0:
    Yneg = True
If changeinx <0
    Xneg = True
```

Python code

```
def maths(objectxy,object2xy):
    a = int(objectxy[0] - object2xy[0])
    b = int(objectxy[1] - object2xy[1])
    radius = math.sqrt((a**2) + (b**2)) #Pythagorus theorem
    if radius != 0:
        if a == 0:
            theta = 0 #change in x is 0 and we dont want an error to be thrown.
        else:
            theta = abs(math.atan(b/a))
    else:
        theta = 0
    if -1 < radius < 1:
```

```
13.         radius = 1 #no div by 0
14.     if b > 0:
15.         yn = True
16.     else:
17.         yn = False
18.     if a > 0:
19.         xn = True
20.     else:
21.         xn = False
22.     return(radius,theta,xn,yn)
```

Connection between Maths and Physics modules

I decided early on in development that the maths and physics side of the equation should be split up to both simplify debugging process and to allow the maths module to be used for other purposes such as the trajectory drawing and the targeting on planet creation.

Bubble sort

An issue with the popping algorithm in my program that upon popping from the array, another planet will replace the row that the planet once resided upon. To fix this, the list to be deleted had to be ordered in descending order. This means that planets would not be deleted by accident as the highest lists are popped first, similar to removing the top pieces of a jenga stack rather than the bottom ones.

```
start array [9, 6, 5, 4, 2, 6, 7, 10, 3, 1, 8]
[1, 2, 3, 4, 5, 6, 6, 7, 8, 9, 10]
computed in 0.015599966049194336  seconds
```

```
start array [9, 6, 5, 4, 2, 6, 7, 10, 3, 1, 8]
[1, 2, 3, 4, 5, 6, 6, 7, 8, 9, 10]
computed in 0.015599727630615234 seconds
>>>
```

Bubble sort vs python sorted(array) performance wise

Collision detection

Using the Newton's third law "For every action, there is an equal and opposite reaction." Therefore we can equate that:
Momentum before impact = Momentum after impact[23]
So our equation for impact is collision is:

Mass1*InitialVelocity1 + Mass2*InitalVelocity2 = Mass1*FinalVelocity1 + Mass2*FinalVelocity2

M1u * M2u = M1v * M2v

[23] "Momentum - Wikipedia." https://en.wikipedia.org/wiki/Momentum. Accessed 16 Mar. 2017.

We can decide when objects collide by calculating whether the areas of the objects are clipped inside each other and then override the main equation to calculate a bounce or an explosion of the objects.

Pseudo code:

P1 = planet1
P2 = planet2
Ai = after impact
Pseudo

```
Until planets are all calculated Do
    If objects have collided Do
        If planets hit hard enough Do
            fragmentplanets()
        End
    End
    Else
        calculatetheta()
        P1mass*P1Velocity + P2*P2Velocity = P1mass *P1Velocityai +
P2mass*P2Velocityai
    End
End

Else Do
    NormalCalculation()
End
```

Python

```
def colide(planet1,planet2,radius):
    if radius <  OBD[planet2]["radius"] + OBD[planet1]["radius"]:
        if radius < OBD[planet2]["radius"]:
            ui.window.lower(OBD[planet2]["planet"])
        else:
            ui.window.lower(OBD[planet1]["planet"])
        return True
    else:
        return False


if colide(planet1,planet2,radius) == True:
                            if OBD[planet1]["radius"] >= OBD[planet2]["radius"]
and planet2 not in poplist:
                                tbd = planet2
                                tbnd = planet1
                                OBD[planet1]["mass"] += OBD[planet2]["mass"]
                                OBD[planet1]["planetsdevoured"] += 1
                            elif OBD[planet2]["radius"] >= OBD[planet1]["radius"]
and planet1 not in poplist:
                                tbd = planet1
                                tbnd = planet2
                                OBD[planet2]["mass"] += OBD[planet1]["mass"]
                                OBD[planet2]["planetsdevoured"] += 1
```

Loading

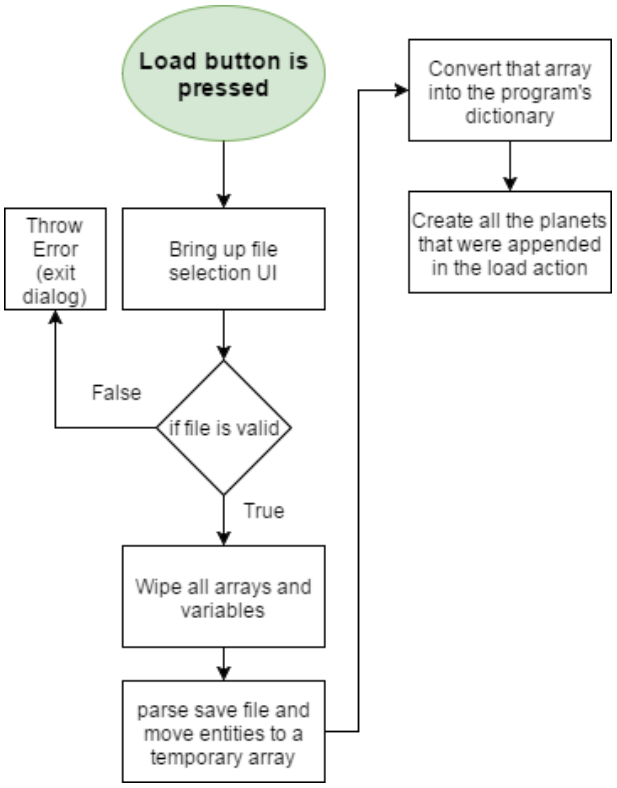
The program loads files using the load() function. The function translates a file under the file name ".pyx" and after ensuring the file is valid, translates that to a 2D array then to the object dictionary called "OBD".

The 2d array is a layout from an older version of my program that simply used 2d arrays to store variables.

Pseudo code

```
1. Function load()
2.       File = fileselected() #tkinterfunction
3.       File.open()
4.       wipeallvariables()
5.       For lines in file do
6.           temparray = file[lines]
7.       end
8.       temparray = objectdictionary
9.       createplanets
10. end
```

Flow diagram



Saving

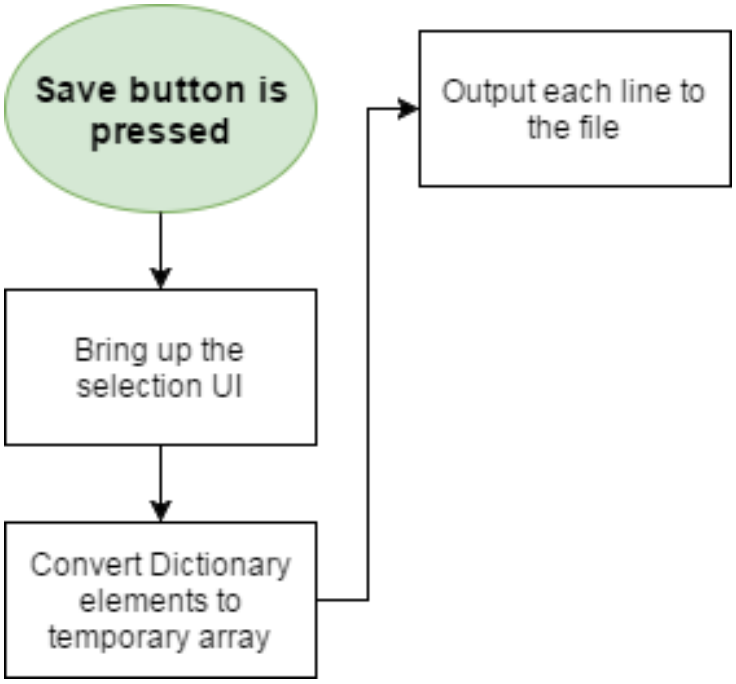
Saving is the direct opposite to the loading function, instead simply just converting the "OBD" dictionary to a 2d array and dumping it into a file specified by the user with a ".pyx" extension.

Pseudo code

```
1. Function save()
2.     file = fileselected() #tkinterfunction
3.     file.open()
4.     for line in objectdictionary
5.         temparray[line] = objectdictionary[x1,y1,etc...]
6.         file.output(temparray[lines])
```

```
7.      end
8. end
```

Flow diagram



Undo/redo

Stack

The stack class in my program deals with all stack related operations in the program. The main function of this is to allow undo and redo functions.

Summing this up, whenever a planet is deleted or created by any other function but the stack functions, the data of that planet is logged into the stack. A stack pointer points towards the last position of data that was processed.

Undo

When the undo button is pressed, the pointer moves downwards and reads what was logged. If it finds a “D” then the program will instead call the “createplanet” function with the data of that planet. When the program finds a “A” then the function calls “deleteplanet” which deletes the object by its ID on the canvas

Redo

Redo is very similar to the undo function but instead moves the pointer up the stack and instead deletes when it finds a “D" and creates when it finds an “A”.

Minor extra functions

Some extra things were needed to get the functions working in this regard:

- Planet identification numbers are changed upon creating a planet using the un/redo functions as the ID on the canvas no longer exists and must be changed to avoid essentially calling nothing .
- Self written pop and push algorithms as the python ones do not play nicely with stacks of this nature.
- A function to check if the stack is too large and if so either drop the last value on the list or append to the top of the stack.

Pseudo code

```
CLASS stack
    Function pop
        IF pointer => Do
            pointer = pointer + 1
        End
        ELSE
            OUTPUT “stack empty”
        End
    End
     Function push (data)
         IF pointer < stackmaximum Do
             pointer = pointer + 1
```

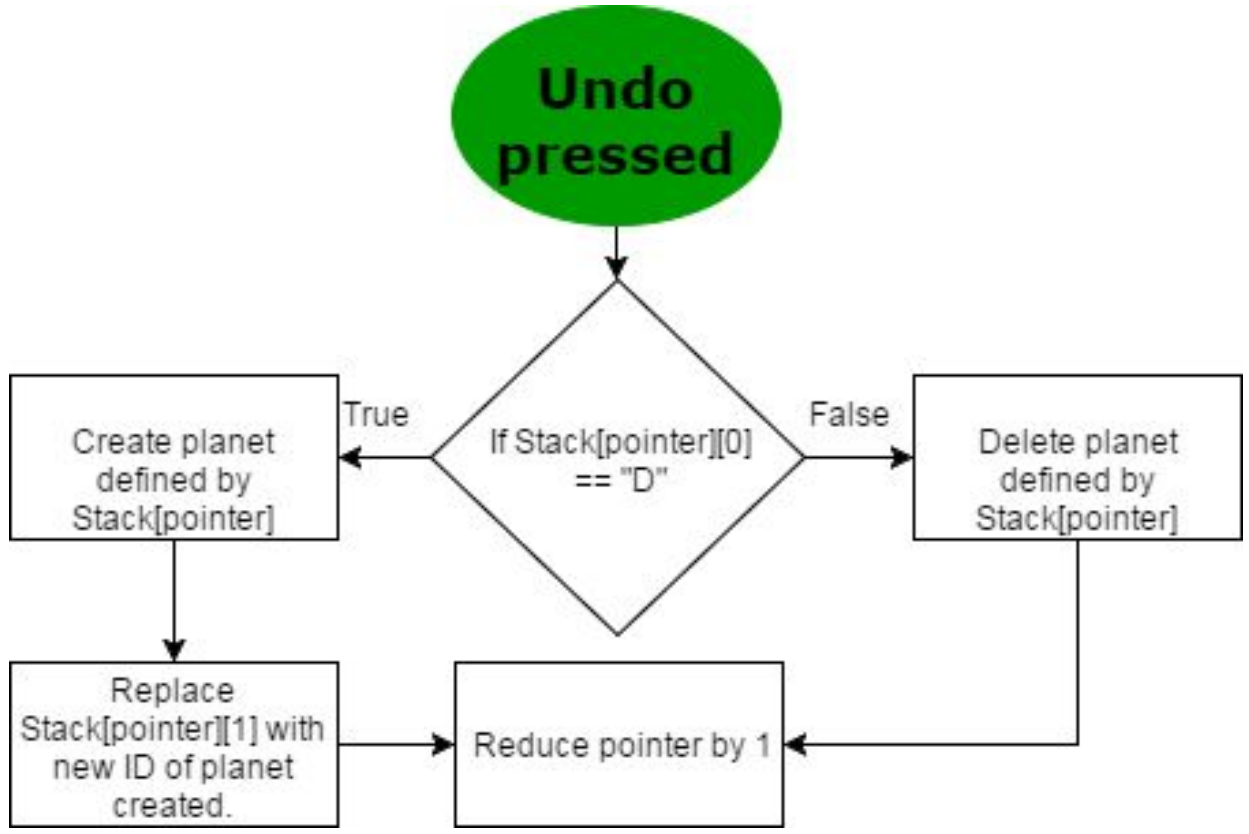
```
            stack[pointer]  = data
        ELSE
            OUTPUT “stack full”
    End
    Function undo
         IF lengthofstack > 0 Do
        IF stack[pointer][0] == “D” Do
            Createplanet(stack[pointer])
        End
        ELSE
            Deleteplanet(stack[pointer])
        End
         End
    End
    Function redo
           If lengthofstack != stackmaximum
        IF stack[pointer][0] == “A” then
            Createplanet(stack[pointer])
        End
        ELSE
                    Deleteplanet(stack[pointer])
        End
         End
     End
End
```

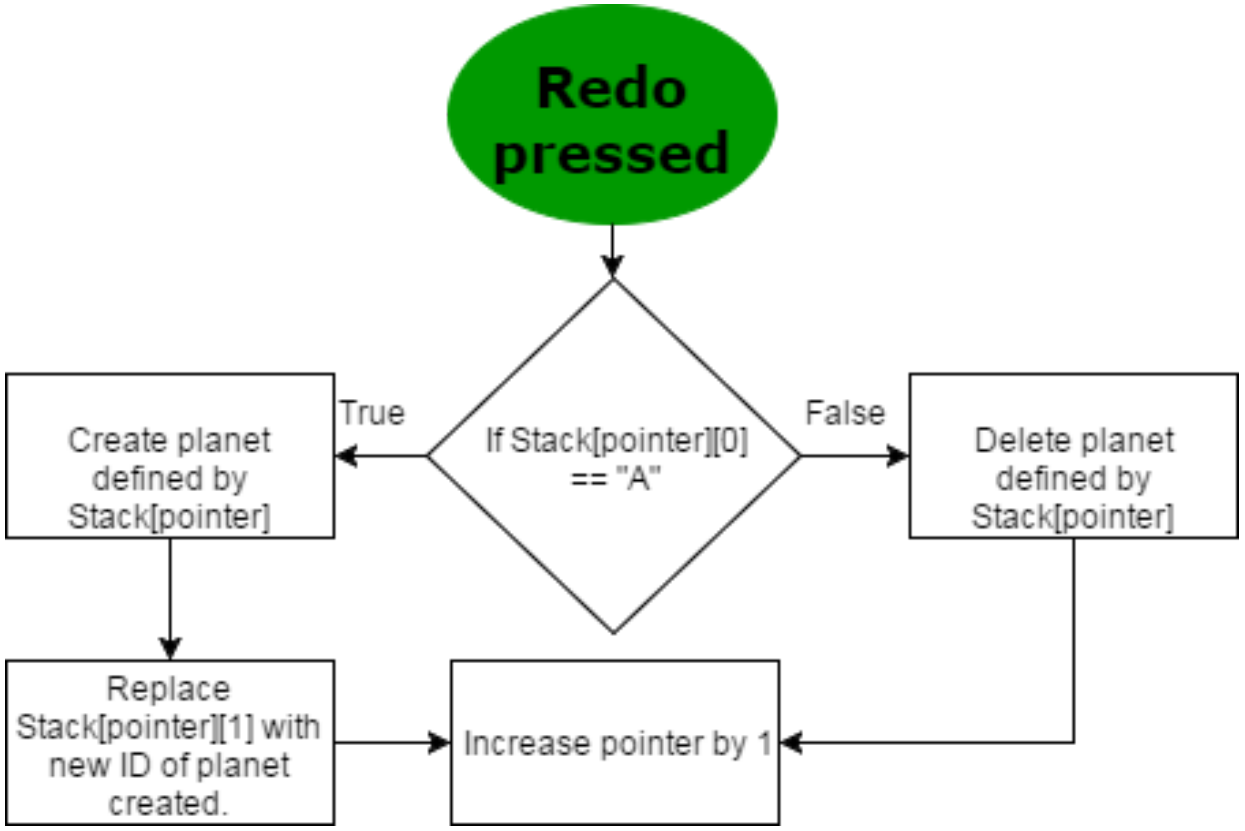
Python

```
class stack():
    def __init__(self,Maxlength,Maxwidth):
        self.StackArray = [["" for x in range(20)] for y in
   range(Maxlength)]#(Y,X) #2d array for planet variables and [0] to represent why
   it was added.
        self.StackMaximum = Maxlength -1
        self.StackPointer = -1
        self.CurrentData = []
        self.mod = 0
    def pop(self):
        if not self.isEmpty():
            self.CurrentData = self.StackArray[self.StackPointer]
```

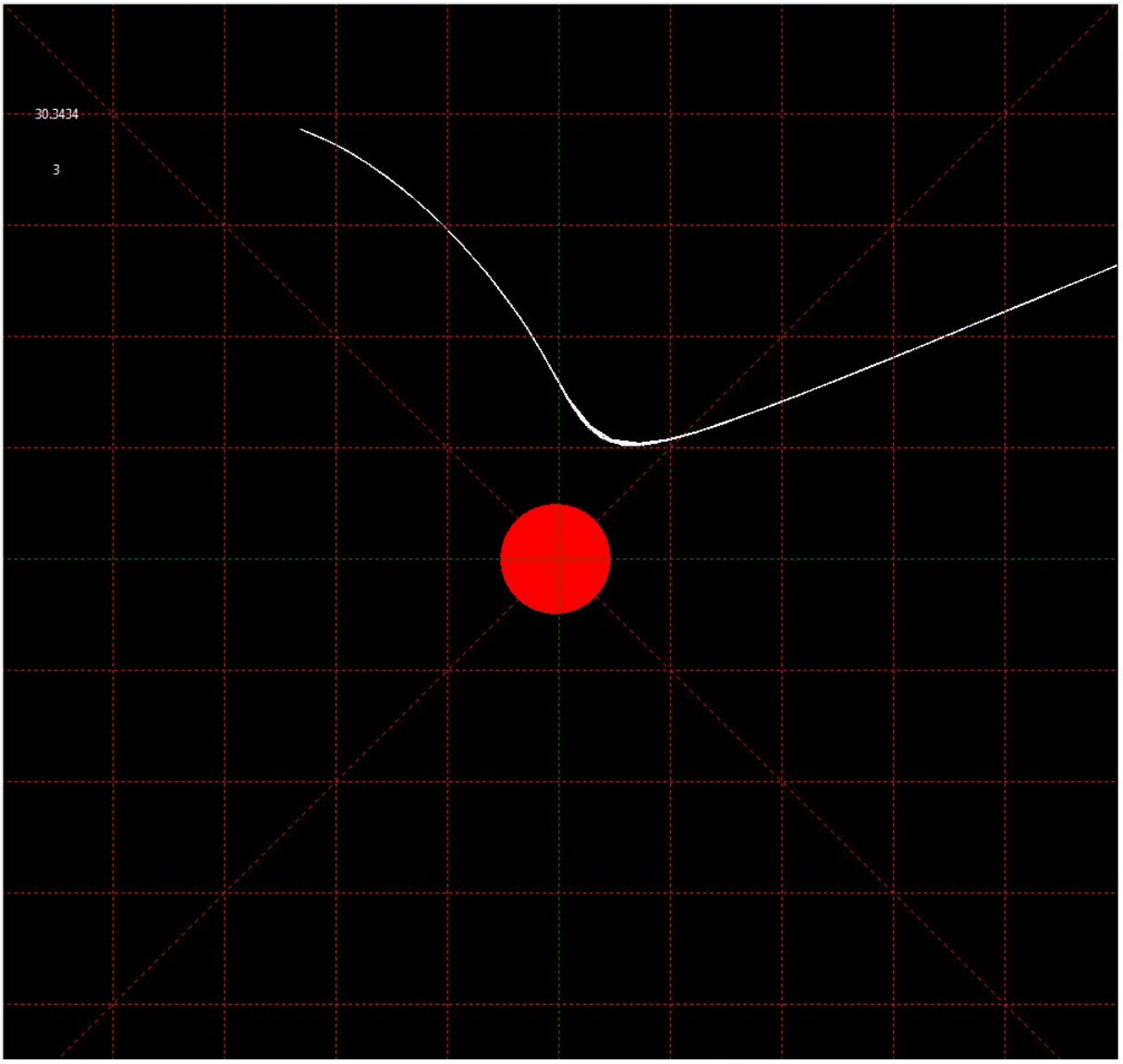
```
            self.StackPointer += -1
    def push(self,Data):
        if not self.isFull():
            self.StackPointer += 1
            self.StackArray[self.StackPointer] = Data
    def peek(self):
        print(self.StackArray[self.StackPointer])
    def isFull(self):
        if self.StackPointer >= self.StackMaximum:
            return True
        else:
            return False
    def isEmpty(self):
        if self.StackPointer < 0:
            return True
        else:
            return False

    def printStack(self):
print("--------------------------------------------------------------------------------
----------------------")
        for y in range(len(self.StackArray)):
            if self.StackMaximum - y == self.StackPointer:
                print("→",self.StackArray[len(self.StackArray)-1 - y],"←")
            else:
                print("|",self.StackArray[len(self.StackArray)-1 - y],"|")

    def fetchposition(self):
        for x in range(len(OBD)):
            if OBD[x]["planet"] == self.StackArray[self.StackPointer][1]:
                return(x)
    def replaceid(self,oldid,newid):
        for x in range(len(self.StackArray)):
            if self.StackArray[x][1] == oldid:
                self.StackArray[x][1] = newid
    def undo(self):
        if self.isEmpty() == False:
            self.mod = self.fetchposition()
            if self.StackArray[self.StackPointer][0] == "A":
                deleteplanet(self.mod,True)
            if self.StackArray[self.StackPointer][0] == "D":
                createplanet()
        self.pop()
    def redo(self):
        if not self.isFull()and self.StackArray[self.StackPointer+1][0] != "":
            self.StackPointer += 1
            if self.StackArray[self.StackPointer][0] == "A":
                createplanet()
            if self.StackArray[self.StackPointer][0] == "D":
                deleteplanet(self.mod,True)
```

Flow chart





Prototyping and development:



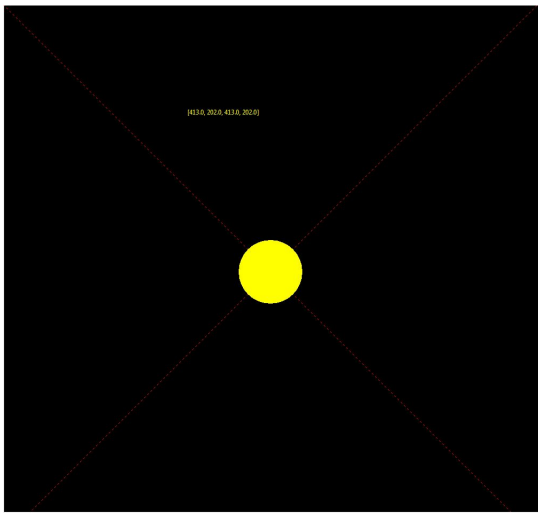
Early version with *large* mathematical errors.

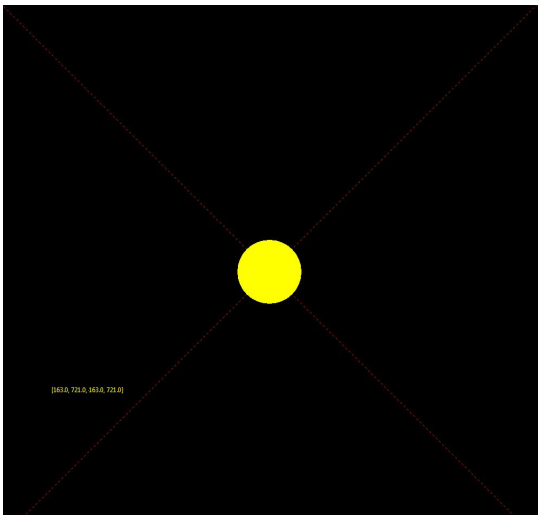
Through the transition from terminal based python to GUI based python I made a whole load of different prototypes.

The video for these prototypes can be found at https://youtu.be/VlUnG_ufb4s and screenshots are provided below.

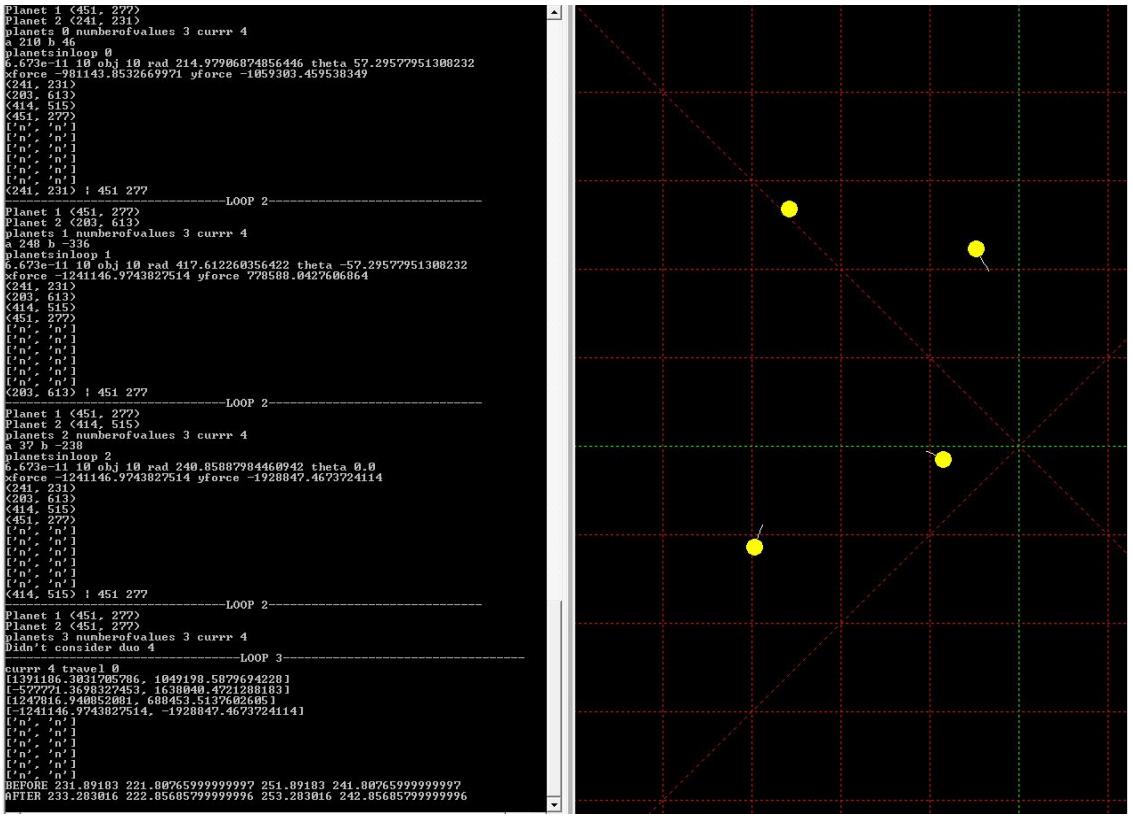
Proof of concept:
Time: 0:00

This is a program where a circle moves from the top left to the bottom right. This shows off the usefulness in our context of this engine. Shows off the ability for the program to move a planet across a plane at a high FPS rate.

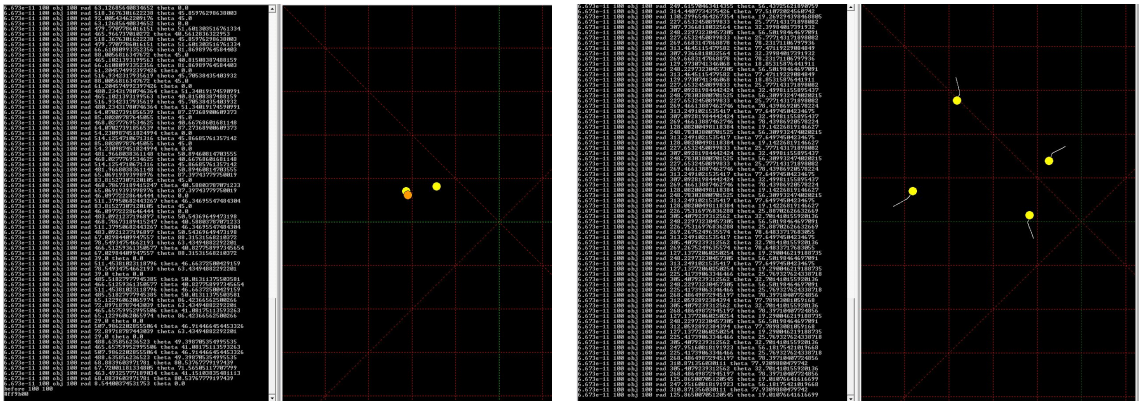




Second proof of concept:
Time: 0:08
This version shows off the tkinter aspects of the program. The program moves a point to a random value +-100 in both x and y axis. This is useful as it proves that this engine is sufficient enough for on-the-fly computation in this fashion. It also shows that the canvas can accurately pinpoint coordinates which would allow proper movement across the plane.



Broken prototype 1:
Time: 0:20
This prototype shows off the possibilities of interactions with multiple objects in the plane by using the first iteration of my mathematical and physics algorithms which were highly inaccurate.

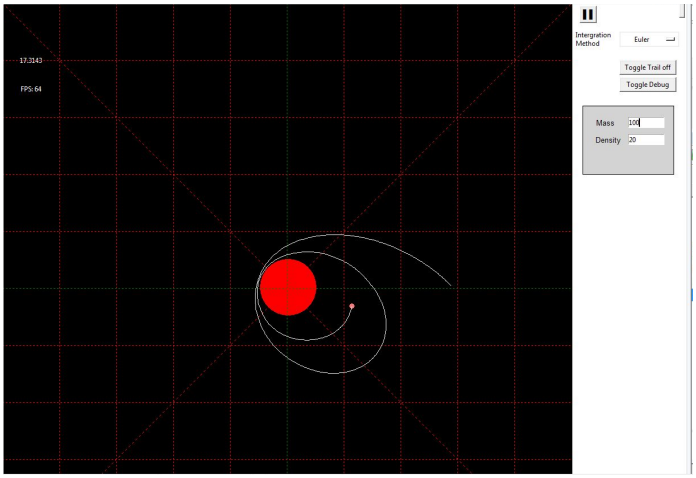


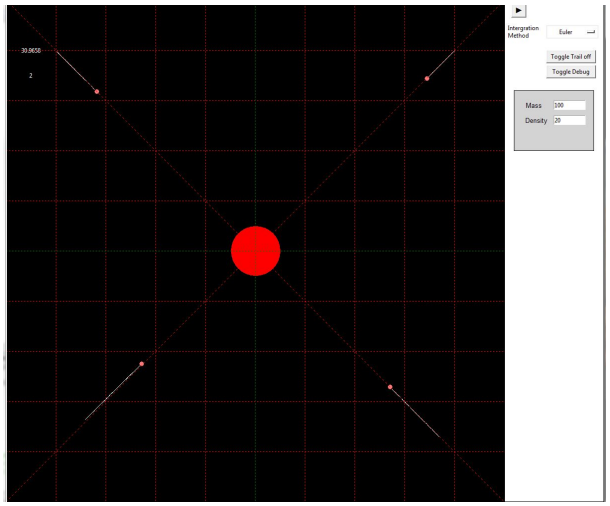
Partially working prototype 1 with bad collision detection:
Time: 0:50
Physics work somewhat accurately in this version. However, the experimental collision detection is not working and will need to be reworked.

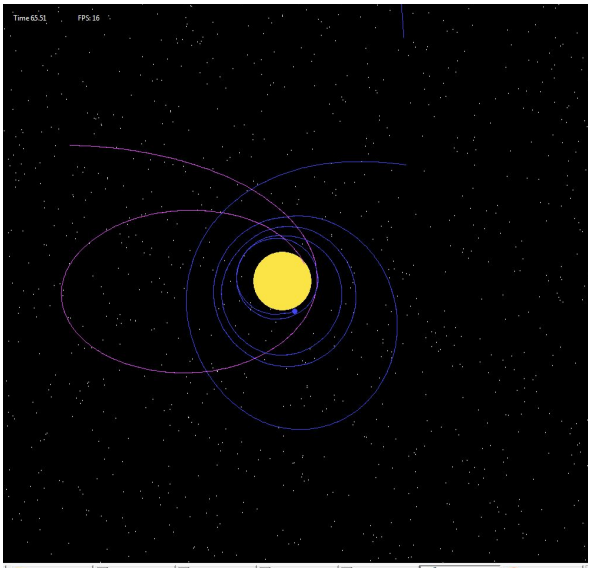
Multiple versions in between:

Time: 0:50 to 1:50

Here are a lot of undocumented versions that have somewhat working gravity. These versions are not well documented because most of them had little changes that weren't big enough improvements to count towards







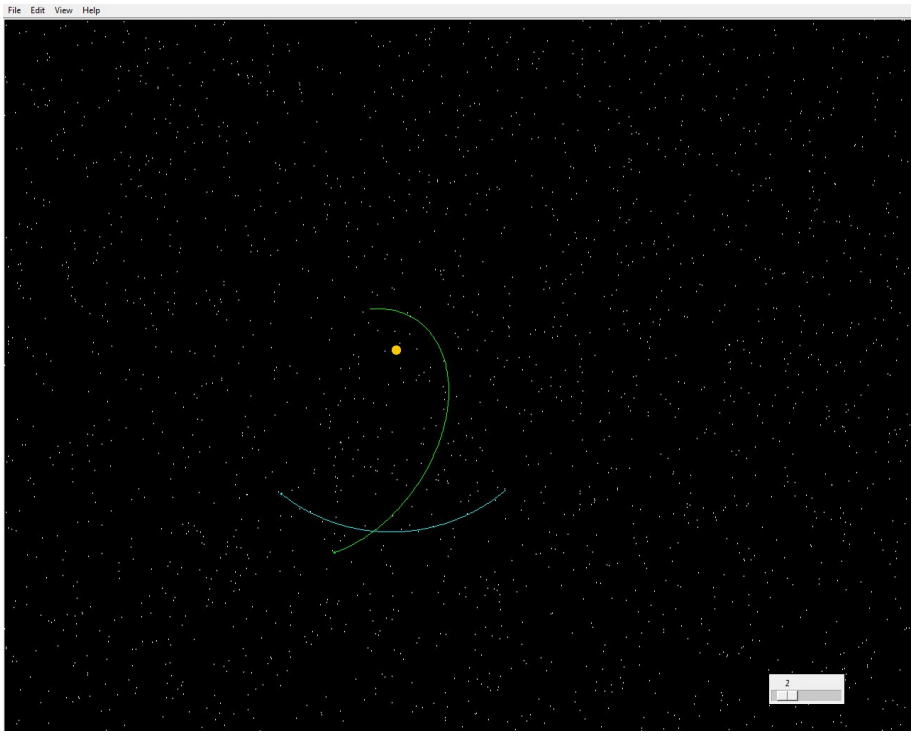
First working build:

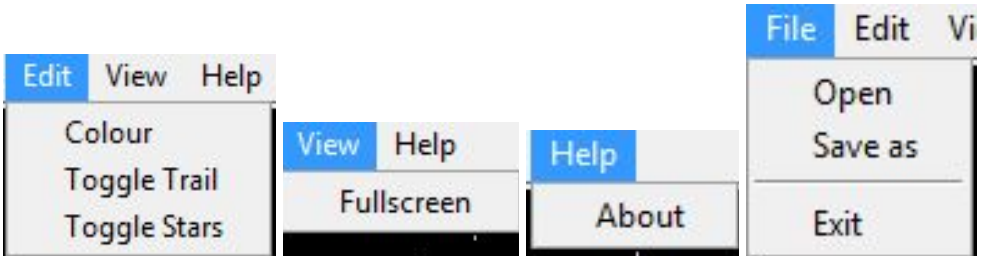
In this version I improved the UI a lot and added custom mass and density values for new planets.
In this version both Euler 8x and Euler integration are working in the program and all the subroutines that will contribute to the development of the other integration methods.

Alongside this I put a pause function in so that you can set up scenarios with very little effort.

The main issue of this version of the program is that the trail drawing function makes the program slow down to a standstill and when trails were not enabled the program would run far too quickly.
Alongside this, gravity was inaccurate due to an oversight in the program moving each planet one by one instead of computing movement of all objects then moving the planets all at once. ==========

Sidestep version: Zooming and panning





This is a version of my program I made which changes the following properties about the program:

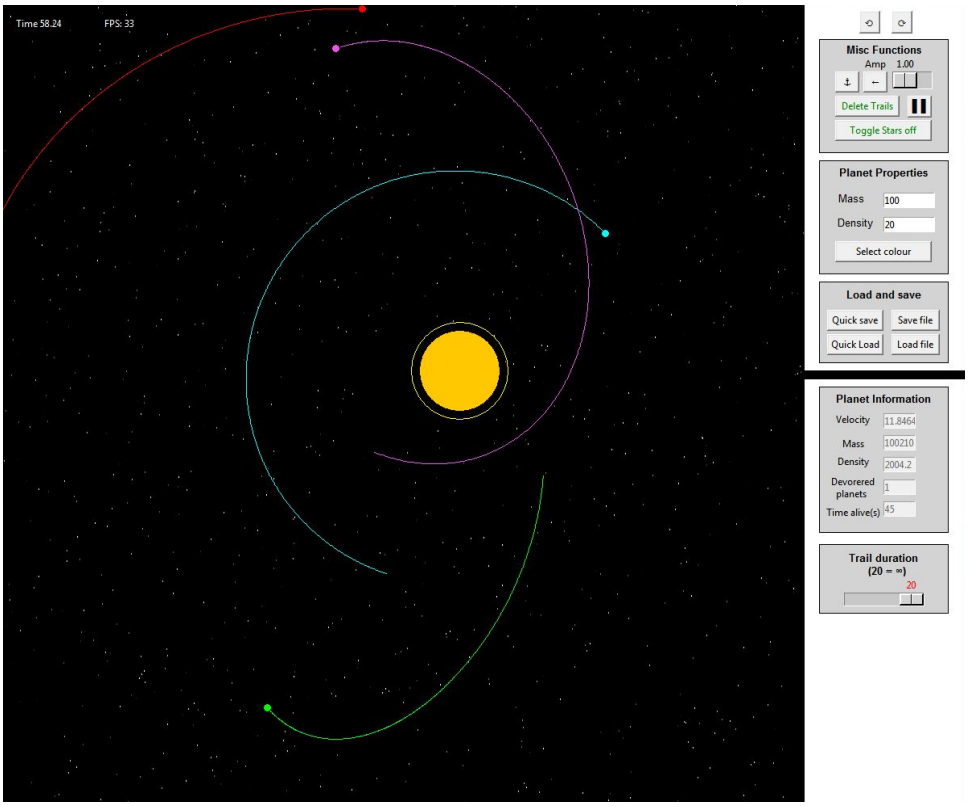
- Ability to take the program fullscreen
- Ability to zoom and pan the canvas
- Control functions UI was contained within a menu bar

This seemed like a step in the correct direction but this raised a number of issues:

- All previous canvas items relied on the x and y coordinates of the canvas so this broke a lot of the code which resulted in highly innacurate physics after zooming..
- Trails would temporarily draw oddly for a moment after zooming.
- Drawing stars would require much more processing power than python could provide.
- Zooming out

I decided that this far into development this would not be worthwhile and instead will work on hitting other objectives that will matter far more to the final product.

Final version of the program



In this version, all the algorithms are in place, working and the user interface and the stack parts of the program is fully object orientated.
The major difference between this version and the rest is a lot of UI related changes:

- A Lot more information is shown in the “planet information” box that was previously available to the user.
- Trails can have a variable lifespan if wanted.
- Saving, loading, quicksaving and quick loading are now implemented.
- Colour selection and star toggling is now a thing.

The entire UI section is contained in an object oriented class. This is used because it allows quick and easy access to UI elements which are constantly updating throughout the program. This would also allow further UI windows to be generated in future developments of the program if so desired.

Alongside this, the stack is object orientated to allow unrestricted access to the undo/redo stack all the way through the program. This reduces the amount of globals needed and vastly improves performance.

Testing

Unit testing

In this section I test if I have achieved what I set out to do in my analysis.

The entire testing is linked here https://www.youtube.com/watch?v=SHZZ8jZwW_4

Test no.	Description	Data type	Expected	Result	Time in video
1	Flinging a single planet in a single direction.	Typical	The planet flies in that direction.	Pass	0:00
2	Creating two planets that interact with each other	Typical	The planets interact properly.	Pass	0:08
3.	Planets collide correctly .	Typical	Planets collide and the smallest planet is obliterated then it's mass is added to the surviving planet.	Pass	0:17
4	The program pauses and resumes correctly	Typical	The program pauses then resumes.	Pass	0:30
5	Saving a system	Typical	The file saves properly	Pass	0:54
6	Loading a system	Typical	The file loads properly	Pass	1:05
7.	Quick save and quick load works appropriately	Typical	Quick save creates "tmp.pyx" then quickload loads "tmp.pyx"	Pass	1:10

8	Colours can be selected	Typical	When selecting the colour, the program should pause and afterwards, any planet should spawn with that colour.	Pass	1:23
9	Trails work correctly	Typical	Both timed trials and permanent trails are properly created from the movement of planets.	Pass	1:40
10	Undo and redo functions work accordingly	Typical	Planets are deleted and created according to their position in the stack	Pass	1:55
11	“Edu-mode” accurately shows gravitational forces	Typical	Width of lines increase near more influential places.	Pass	2:22
12	Planet anchor correctly works	Typical	Anchored planets do not move but apply forces to other planets	Pass	2:38
13	Force amplification works accordingly	Typical	Forces should be multiplied respectively	Pass	2:51
14	Delete functions work	Typical	Objects should cease existence	Pass	2:58
15	Mass and density only accept numerical values as inputs	Erroneous	The program should reject the input.	Pass	3:05
16	Attempting to create planets out of bounds	Extreme	Does not create the object.	Pass	3:32
17	Loading a file not created by the program	Extreme	Warns the user that it is not valid.	Pass	3:40

System testing

Here is my general system testing video where I basically recreate what a user might do on the program.
https://www.youtube.com/watch?v=LuS21359tzU

In my system testing I show most if not all of the objectives I have hit in my program such as loading, saving, creating planets and many more. By doing this kind of testing, I can test and present the workflow of this program and how a “power user” might use the program.

Evaluation

Limitations and oversights

The main issue I ran into regarding my setup was tkinter performance limitations when dealing with 200+ objects. Initially I was not worried about this because simply put I did not think that anybody would ever hit that limit. Unfortunately for me, when I came around to making the trail algorithm it turns out that creating a line for each time a planet moved hit that limit very quickly.

This issue is nicely outlined in article by explaining how tkinter updates it's canvas:

"The Canvas widget uses a straight-forward damage/repair model. Changes to the canvas, and external events such as Expose, all update a single "dirty rectangle". They also register an idle task (after_idle) which redraws the canvas when you get back to the main loop."[24]

This is a large problem for the program as each canvas object is to be completely redrawn on every frame of the program. In essence, this leads to a large performance overhead which could be a major issue on some older hardware.

This was a large oversight in my research as I did not believe that this would be such an issue that I would encounter near the end of my project.

If I could go back and change something I would likely use an engine such as PyGame or PyQT as they have support for "Stamping" objects on a canvas which leads to far less performance overhang.

Another issue I encountered was certain unicode values crashing both IDLE and Tkinter (IDLE is based on tkinter). This meant that some icons that I wanted to use was impossible as the program would simply crash trying to use the icons.
Turns out this bug has been around for 6 years[25] and nobody has been bothered enough to fix it.

This shows how dated the engine is as a simple bug like this has not been fixed after 6 years. This makes me question the longevity of the library and whether or not the engine will work with update versions of python in the future.

Objective Analysis

[24] "Notes on Tkinter Performance - effbot.org." http://effbot.org/zone/tkinter-performance.htm. Accessed 21 Mar. 2017.
[25] "Issue 13153: IDLE crashes when pasting non-BMP unicode char on Py3 ...." 11 Oct. 2011, https://bugs.python.org/issue13153. Accessed 21 Mar. 2017.

Core objectives

(all met objectives are shown in system testing)

Objective	Met?	Comments
Accurate representation of 2D physics.	Yes.	Gravity works fully and accurately enough to be used in a teaching environment. This is shown in most of my testing but especially in unit test 2 and unit test 6.
Display useful detailed information	Yes.	Program displays all useful variables for each planet on demand. This can be viewed in all my testing on the right hand side of the program window.
Customisable	Yes.	Every variable is customisable on creation of program and canvas. This is presented in Both unit test 2, unit test 8.
Low/little performance issues	Partial	Program runs fluidly when all diminishing functions are disabled when working with less than around 20 planets. Tkinter is the bottleneck here and the only solution is to rewrite the program. At 1:50 in the system testing and onwards demonstrates the issues that many planets causes in the program.
Load/save system.	Yes.	Small file sizes and save files perfectly load and save. On average the files are below 1 kb and even for large systems rarely exceed 5kb.
Play and pause function	Yes.	Program perfectly pauses and resumes. This is demonstrated in unit test 4.

Extension objectives:

Objective	Met?	Comments
Impact simulation	No.	This was simply out of the scope of the time frame that I had for the program.
Auto orbit functions	No.	Although I had anticipated that this would be simple to implement, It caused way too much hassle to be worth while and was scrapped mid development. If I had more time to research into the issues this caused it this would've been rather easy to do.
Force Visualisation	Yes.	The education mode visualises all forces acting on an object. This is shown in unit test 11.
Undo/redo functions	Yes.	Undo and redo functions work flawlessly. Shown in unit test 10.

User feedback

Student feedback

I contacted the both the students James and Regan that I initially interviewed as part of my analysis a week after I gave them the program and asked them to fill out a quick questionnaire.

Questionnaire

1. Does the program fully live up to what you requested? If not, please elaborate further.
2. Is the UI informative and visually interesting enough for yourselves and your peers?
3. Now that the project has been completed, what features do you think would be suitable additions to this system?
4. How does this compare to the other “competition” such as the online tools?

Praises and critiques

From the responses I received they seemed really satisfied with the final product but there was some main criticisms:

- The performance issues as only a “minor” issue as the program still produces output just a bit slower. This was my main concern creating the program and would be the first thing I would attempt to fix if I was to rewrite the program.

- How trails were not saved alongside the planetary positions in save files. I decided not to implement this as the save files would be both massive and I doubt most users wouldn't want a clean canvas after a save. However, I think it should be an option if I was to further improve the program.

Teacher feedback

I got back in contact with both the teachers (Mrs Sheffield and Mr Bashley) to question them on their opinions of the final product a week after supplying them with the finished product. I gave them a different questionnaire to the students:

1. Do you believe that the program fulfils the requirements that you outlined when I initially interviewed you?
2. Does the program output what is required for the teaching environment you create?
3. Now that you have the finished product, What more could be added to make the software better for your class?

Praises and critiques:

From the answers I received back they both strongly believe that the software is suitable for the class they are teaching and that the program's output was more than enough. However, they did have some idea's they'd wished that was added in hindsight:

1. The ability to rewind the simulation.

They mentioned that occasionally it is hard to talk about the simulation in complex situations as it might be simply to hectic to explain all at once. This problem is partly solved by the "quick save/load" function but could be worked upon to make the system even better for classroom work.

2. Trails delete whilst paused.

They told me that sometimes they would like to be able to see trails whilst paused so they can explain orbital patterns to students. Although this issue is mostly engined based, with the use of another engine it might not suffer from the same results.

Other than what has been previously stated in layman's terms: The final product satisfies the requirements of the intended users and has for the most part eradicated the main problem in their teaching environment.

Final analysis

After gathering all information ranging from user feedback to objective fulfillment I can strongly say that the system has solved our initial problem. The program hits all core objectives and most extention objectives which show the completeness of the solution.

The major limitations that I faced were graphics engine limitations and mainly time which prevented the sky from truly being the limit. If I had the time to implement what I pleased I could've added a whole lot more that could have brought it more into line with typical education software.

If I could go back and change any major decisions I would instead use a different GUI library such as PYGame which would result in a far more efficient and cleaner solution. However, this might have an impact on the portability of the program because it requires extra libraries.

Code appendix

> main.py

```
#setup
#Importing modules
from tkinter import * #enables use of the Tkinter UI, responsible for drawing of objects and generation of GUI
from tkinter import filedialog #Cannot run without this unless in IDLE
from tkinter.colorchooser import *
import time #needed for restriction of refresh rate (may not be needed)
import math #needed for physics calculation
import random #for some more "fun" parts of the program.

#constants
G = 6.6741  #simplfied
#2d arrays
OBD = [] #2D dictionary for planet variables
poplist = []
anchorlist = []
edu = False
def main():
    for planet1 in range(0,len(OBD)): #for more than one object
        if planet1 not in anchorlist:
            objectxy = [((OBD[planet1]["x0"]+OBD[planet1]["x1"])/2),((OBD[planet1]["y0"]+OBD[planet1]["y1"])/2)]
            for planet2 in range(0,len(OBD)):
                if planet2 != planet1:
                    object2xy = [((OBD[planet2]["x0"]+OBD[planet2]["x1"])/2),((OBD[planet2]["y0"]+OBD[planet2]["y1"])/2)]
                    if object2xy != objectxy:
                        ###Essentialy the collision detection###
                        radius,theta,xn,yn = maths(objectxy,object2xy)
                        if colide(planet1,planet2,radius) == True:
                            if OBD[planet1]["radius"] >= OBD[planet2]["radius"] and planet2 not in poplist: #condense this bit
                                tbd = planet2
                                tbnd = planet1
                                OBD[planet1]["mass"] += OBD[planet2]["mass"]
                                OBD[planet1]["planetsdevoured"] += 1
                            elif OBD[planet2]["radius"] >= OBD[planet1]["radius"] and planet1 not in poplist:
                                tbd = planet1
                                tbnd = planet2
                                OBD[planet2]["mass"] += OBD[planet1]["mass"]
                                OBD[planet2]["planetsdevoured"] += 1

                            try:
                                if tbd not in poplist and tbnd not in poplist:
                                    poplist.append(tbd)
                            except UnboundLocalError:
                                pass
                            ui.window.lower("star")
                            break
                        Euler(objectxy,object2xy,planet1,planet2,OBD,ui.speed)
            OBD[planet1]["lx"] = objectxy [0]
            OBD[planet1]["ly"] = objectxy [1]

def maths(objectxy,object2xy):
    a = int(objectxy[0] - object2xy[0])
```

```
52.     b = int(objectxy[1] - object2xy[1])
53.     radius = math.sqrt((a**2) + (b**2)) #Pythagorus theorem
54.     if radius != 0:
55.         if a == 0:
56.             theta = 0 #change in x is 0 and we dont want an error to be thrown.
57.         else:
58.             theta = abs(math.atan(b/a))
59.     else:
60.         theta = 0
61.     if -1 < radius < 1:
62.         radius = 1 #no div by 0
63.     if b > 0:
64.         yn = True
65.     else:
66.         yn = False
67.     if a > 0:
68.         xn = True
69.     else:
70.         xn = False
71.     return(radius,theta,xn,yn)
72.
73. def physics(planet1,planet2,objectxy,object2xy,OBD,speed):
74.     radiusbetweenplanets,theta,xn,yn = maths(objectxy,object2xy)
75.     if radiusbetweenplanets != 0:
76.         Fgrav =
    ((G*(int(OBD[planet1]["mass"])*(int(OBD[planet2]["mass"])))))/radiusbetweenplanets
    **2) / OBD[planet1]["mass"]
77.         accelerationx = Fgrav*math.cos(theta)
78.         accelerationy = Fgrav*math.sin(theta)
79.         if xn == True:
80.             accelerationx = -(accelerationx)
81.         if yn == True:
82.             accelerationy = -(accelerationy)
83.
84.         cspeed = (speed.get()/1000)
85.         #Resolving (Right) (positive x)
86.         vx = accelerationx*cspeed #Mass is not used here because it results in a
    more "exaggerated" but more workable interactions.
87.         vy = accelerationy*cspeed #Essentially removes the need for larger
    numbers.
88.         #Resolving (Down) (positive y)
89.
90.         #arrow for educationmode#
91.
92.         if edu == True and ui.planetselected == planet1:
93.             widthofgrav = abs((abs(vx) + abs(vy))*500)
94.             widthofvel = abs((abs(OBD[planet1]["dx"]) +
    abs(OBD[planet1]["dy"]))*5)
95.             if widthofgrav > 20:
96.                 widthofgrav = 20
97.             if widthofvel > 20:
98.                 widthofvel = 20
99.             gravline =
    ui.window.create_line(objectxy[0],objectxy[1],object2xy[0],object2xy[1],width =
    widthofgrav,fill = "White",tag="educat",arrow="last")
100.                velline =
    ui.window.create_line(objectxy[0],objectxy[1],objectxy[0]+OBD[planet1]["dx"]*50,o
    bjectxy[1]+OBD[planet1]["dy"]*50,width = widthofvel,fill =
    "orange",tag="educat",arrow="last")
101.                ui.window.lower(velline)#correct the ordering of objects.
102.                ui.window.lower("oval")
103.                ui.window.lower(gravline)
104.                ui.window.lower("t")
105.                ui.window.lower("star")
106.            return(vx,vy)
107.
108.    def Euler(objectxy,object2xy,planet1,planet2,OBD,speed):
109.        prevxy = objectxy
110.        vx,vy = physics(planet1,planet2,objectxy,object2xy,OBD,speed)
```

```
        OBD[planet1]["dx"] += vx
        OBD[planet1]["dy"] += vy

    def createplanet(rad,mass,x,y,R,G,B,cx,cy,theta,undo,id):
        global OBD
        OBD.append({"x0": x-rad,"y0": y-rad,"x1": x+rad, "y1": y+rad,"mass":
    mass,"RGB":('#%02x%02x%02x' % (int(R//1), int(G//1),
    int(B//1))),"dx":cx,"dy":cy,"lx":x,"ly":y,"radius":rad,"R":R,"G":G,"B":B,"planet"
    :0,"alivetime":time.time(),"Theta":theta,"planetsdevoured":0})
        slot = (len(OBD)-1)
        OBD[slot]["planet"] =
    ui.window.create_oval(OBD[slot]["x0"],OBD[slot]["y0"],OBD[slot]["x1"],OBD[slot]["
    y1"],fill=(OBD[slot]["RGB"]),tags="oval")
        tba =
    ["A",OBD[slot]["planet"],OBD[slot]["x0"],OBD[slot]["y0"],OBD[slot]["x1"],OBD[slot
    ]["y1"],OBD[slot]["mass"],OBD[slot]["RGB"],OBD[slot]["dx"],OBD[slot]["dy"],OBD[sl
    ot]["lx"],OBD[slot]["ly"],OBD[slot]["radius"],OBD[slot]["R"],OBD[slot]["G"],OBD[s
    lot]["B"],OBD[slot]["planet"],OBD[slot]["alivetime"],OBD[slot]["Theta"],OBD[slot]
    ["planetsdevoured"]]
        if undo == False:
            mainstack.push(tba)
        else:
            mainstack.replaceid(id,OBD[slot]["planet"])
        ui.window.lower("oval")
        ui.window.lower("star")

    def deleteplanet(mod,undo):
        try:
            if not undo:
                tba =
    ["D",OBD[mod]["planet"],OBD[mod]["x0"],OBD[mod]["y0"],OBD[mod]["x1"],OBD[mod]["y1
    "],OBD[mod]["mass"],OBD[mod]["RGB"],OBD[mod]["dx"],OBD[mod]["dy"],OBD[mod]["lx"],
    OBD[mod]["ly"],OBD[mod]["radius"],OBD[mod]["R"],OBD[mod]["G"],OBD[mod]["B"],OBD[m
    od]["planet"],OBD[mod]["alivetime"],OBD[mod]["Theta"],OBD[mod]["planetsdevoured"]
    ]
                mainstack.push(tba)
            poplist.append(mod)
            if mod in anchorlist:
                anchorlist.pop(anchorlist.index(mod))
            if mod == ui.planetselected:
                ui.window.delete("s")
                ui.planetselected = 0

        except IndexError:
            pass #deleting something that doesnt exist doesn't matter. Like wise,
    if the undo mod isnt found it will throw a IndexError.
    def colide(planet1,planet2,radius):
        if radius <  OBD[planet2]["radius"] + OBD[planet1]["radius"]:
            if radius < OBD[planet2]["radius"]:
                ui.window.lower(OBD[planet2]["planet"])
            else:
                ui.window.lower(OBD[planet1]["planet"])
            return True
        else:
            return False

    ###UI Related subroutines###

    def drawtrail(prevxy,objectxy,planet1):
        if 0 < objectxy[0] < 1000 and 0 < objectxy[1] < 1000: #no point drawing
    things the user wont see.
            trail =
    ui.window.create_line(prevxy[0],prevxy[1],objectxy[0],objectxy[1],fill=OBD[planet
    1]["RGB"],tags="t")
            ui.window.lower(trail)
            if ui.trailduration.get() != 20:

    ui.master.after(int((ui.trailduration.get())*1000),lambda:ui.window.delete(trail)
    )
```

```
159.
160.    def playpause(colourc):
161.        if colourc == True:
162.            if ui.paused != True:
163.                ui.prevpaused = ui.paused
164.                ui.playp["text"] = " ▸ "
165.                ui.paused = True
166.        elif colourc ==  False:
167.            if ui.playp["text"] == "▌▌  ":
168.                ui.playp["text"] = " ▸ "
169.            else:
170.                ui.playp["text"] = "▌▌  "
171.            ui.paused = not ui.paused
172.
173.    def safetypause(colourc):
174.        if ui.paused == True:
175.            playpause(colourc)
176.            ui.tobepaused = False
177.        else:
178.            ui.tobepaused = True
179.
180.    def getcolour():
181.        ui.prevpaused = ui.paused
182.        if ui.paused == True:
183.            askcolour(ui.prevpaused)
184.            ui.tobepausedcolour = False
185.        else:
186.            ui.tobepausedcolour = True
187.
188.    def askcolour(prevpaused):
189.        ui.planetcolour = askcolor()
190.        try:
191.            x = ui.planetcolour [1][0]
192.        except TypeError:
193.            ui.planetcolour = ((255,0,0),"#FF0000")
194.
195.    ###LOAD AND SAVE SYSTEM###
196.    def load(quick):
197.        #variables to be edited#
198.        if quick == False:
199.            try:
200.                file = filedialog.askopenfilename(filetypes=[(".pyx
    Format","*.pyx")],title="Choose a file to load")
201.            except AttributeError:
202.                pass #User closed the window.
203.        else:
204.            file =".tmp.pyx"
205.        try:
206.            with open(file,'r') as load:
207.                temparray = []
208.                ##Reset Variables##
209.                systemname = ("PYxis - System: {}".format(file))
210.                ui.master.wm_title(systemname)
211.                ui.window.delete("t")
212.                ###Delete array###
213.                for lines in range(len(OBD)):
214.                    deleteplanet(lines,False)
215.                popplanets()
216.                anchorlist = []
217.                ###set array to the file###
218.                lines = [line.split() for line in load]
219.                for line in range(0,len(lines)):
220.                    temparray.append(lines[line])
221.                ui.window.delete("oval")
222.                ui.window.delete("s")
223.                ###convert to float###
224.                for y in range(0,len(temparray)):
225.                    for x in range(len(temparray[y])):
226.                        try:
```

```
                            temparray[y][x] = float(temparray[y][x])
                        except ValueError:
                            pass #cant convert strings to float
            for lines in range(0,len(temparray)):
createplanet(temparray[lines][10],temparray[lines][4],((temparray[lines][0]+tempa
rray[lines][2])/2),((temparray[lines][1]+temparray[lines][3])/2),temparray[lines]
[11],temparray[lines][12],temparray[lines][13],temparray[lines][6],temparray[line
s][7],0,False,0)
                if temparray[lines][17] == "True": #saves the data as string
                    anchorlist.append(lines)
        except FileNotFoundError:
            print("FILE: No file was found.")
        except IndexError:
            print("FILE: Invalid File.")

    def save(quick):
        global OBD
        if quick == True:
            file = open(".tmp.pyx","w")
        else:
            try:
                file = filedialog.asksaveasfile(filetypes=[(".pyx
Format","*.pyx")],title="Choose a file to save",defaultextension=".gpy")
            except AttributeError:
                pass #User closed the window.
        for lines in range(len(OBD)):
            if lines in anchorlist:
                anchor = True
            else:
                anchor = False
            array =
[OBD[lines]["x0"],OBD[lines]["y0"],OBD[lines]["x1"],OBD[lines]["y1"],OBD[lines]["
mass"],OBD[lines]["RGB"],OBD[lines]["dx"],OBD[lines]["dy"],OBD[lines]["lx"],OBD[l
ines]["ly"],OBD[lines]["radius"],OBD[lines]["R"],OBD[lines]["G"],OBD[lines]["B"],
OBD[lines]["alivetime"],OBD[lines]["Theta"],OBD[lines]["planetsdevoured"],anchor]
            for entitiy in array:
                file.write(str(entitiy))
                file.write(" ")
            file.write(" \n")
        file.close()

    def popplanets():
        global poplist
        ##BUBBLESORT##
        for itterations in range(len(poplist)):
            for swaps in range(len(poplist)-1):
                if poplist[swaps]<poplist[swaps+1]:
                    poplist[swaps+1], poplist[swaps] =
poplist[swaps],poplist[swaps+1]
        for planet in range(len(poplist)):
            try:
                ui.window.delete(OBD[poplist[planet]]["planet"])
                OBD.pop(poplist[planet])
            except TypeError:
                pass #planet was destroyed between actions due to the stack
        poplist = []

    def selectobject(event):
        x = event.x
        y = event.y
        closest = ui.window.find_closest(x,y)
        try:
            if ui.window.gettags(closest)[0] == "oval":
                coords = ui.window.coords(closest)
                for planets in range(0,len(OBD)):
                    if coords ==
[OBD[planets]["x0"],OBD[planets]["y0"],OBD[planets]["x1"],OBD[planets]["y1"]]:
                        ui.planetselected = planets
```

```
285.        except:
286.            pass #planet is unavailable
287.    def deltrail():
288.        ui.window.delete("t")
289.
290.    def anchor():
291.        global anchorlist
292.        if ui.planetselected not in anchorlist:
293.            anchorlist.append(ui.planetselected)
294.        else:
295.            anchorlist.remove(ui.planetselected)
296.
297.    def educationmode():
298.        global edu
299.        if ui.education["text"] != "←":
300.            ui.education["text"] = "←"
301.        else:
302.            ui.education["text"] = "X"
303.        edu = not edu
304.
305.    class stack():
306.        def __init__(self,Maxlength,Maxwidth):
307.            self.StackArray = [["" for x in range(20)] for y in
    range(Maxlength)]#(Y,X) #2d array for planet variables and [0] to represent why
    it was added.
308.            self.StackMaximum = Maxlength -1
309.            self.StackPointer = -1
310.            self.CurrentData = []
311.            self.mod = 0
312.        def pop(self):
313.            if not self.isEmpty():
314.                self.CurrentData = self.StackArray[self.StackPointer]
315.                self.StackPointer += -1
316.        def push(self,Data):
317.            if not self.isFull():
318.                self.StackPointer += 1
319.                self.StackArray[self.StackPointer] = Data
320.        def peek(self):
321.            print(self.StackArray[self.StackPointer])
322.        def isFull(self):
323.            if self.StackPointer >= self.StackMaximum:
324.                return True
325.            else:
326.                return False
327.        def isEmpty(self):
328.            if self.StackPointer < 0:
329.                return True
330.            else:
331.                return False
332.
333.        def printStack(self):
334.
    print("-------------------------------------------------------------------------
    ----------------------")
335.            for y in range(len(self.StackArray)):
336.                if self.StackMaximum - y == self.StackPointer:
337.                    print("→",self.StackArray[len(self.StackArray)-1 - y],"←")
338.                else:
339.                    print("|",self.StackArray[len(self.StackArray)-1 - y],"|")
340.
341.        def fetchposition(self):
342.            for x in range(len(OBD)):
343.                if OBD[x]["planet"] == self.StackArray[self.StackPointer][1]:
344.                    return(x)
345.        def replaceid(self,oldid,newid):
346.            for x in range(len(self.StackArray)):
347.                if self.StackArray[x][1] == oldid:
348.                    self.StackArray[x][1] = newid
349.        def undo(self):
```

```
            if self.isEmpty() == False:
                self.mod = self.fetchposition()
                if self.StackArray[self.StackPointer][0] == "A":
                    deleteplanet(self.mod,True)
                if self.StackArray[self.StackPointer][0] == "D":
createplanet(self.StackArray[self.StackPointer][12],self.StackArray[self.StackPoi
nter][6],(self.StackArray[self.StackPointer][2]+self.StackArray[self.StackPointer
][4])/2,(self.StackArray[self.StackPointer][3]+self.StackArray[self.StackPointer]
[5])/2,self.StackArray[self.StackPointer][13],self.StackArray[self.StackPointer][
14],self.StackArray[self.StackPointer][15],self.StackArray[self.StackPointer][8],
self.StackArray[self.StackPointer][9],self.StackArray[self.StackPointer][18],True
,self.StackArray[self.StackPointer][1])
            self.pop()
        def redo(self):
            if not self.isFull()and self.StackArray[self.StackPointer+1][0] !=
"":
                self.StackPointer += 1
                if self.StackArray[self.StackPointer][0] == "A":
createplanet(self.StackArray[self.StackPointer][12],self.StackArray[self.StackPoi
nter][6],(self.StackArray[self.StackPointer][2]+self.StackArray[self.StackPointer
][4])/2,(self.StackArray[self.StackPointer][3]+self.StackArray[self.StackPointer]
[5])/2,self.StackArray[self.StackPointer][13],self.StackArray[self.StackPointer][
14],self.StackArray[self.StackPointer][15],self.StackArray[self.StackPointer][8],
self.StackArray[self.StackPointer][9],self.StackArray[self.StackPointer][18],True
,self.StackArray[self.StackPointer][1])
                if self.StackArray[self.StackPointer][0] == "D":
                    deleteplanet(self.mod,True)

    mainstack = stack(1000,18)

    class userinterface():
        def __init__(self):
            self.master = Tk() #master window
            self.window = Canvas(self.master, width=1200, height=1000) #generate
1200x1000 canvas
            self.master.resizable(width=False,height=False)
            self.window.pack()

            try:
                self.master.iconbitmap("icon.ico")
            except TclError:
                print("INFO: Icon not found. Continuing..")

            #Basic UI elements#
            self.starttime = time.time()
            self.lasttime = self.starttime
            self.window.configure(background="Black") #Main background
            self.optionsbackground =
self.window.create_rectangle(1001,0,1205,1000,fill="white") #white rectangle
behind the options
            self.curtime = self.window.create_text(50,30,fill = "White") #Time
running
            self.fps = self.window.create_text(150,30,fill = "White") #current
FPS
            self.ox = 0
            self.oy = 0
            self.otime = time.time()
            #Pause related#
            self.tobepaused = False
            self.tobepausedcolour = False
            self.paused = False
            self.prevpaused = False
            self.playp = Button(self.master,text="▌▌ ", command
=lambda:safetypause(False),font=("Helvetica", 12))
            self.playp.place(x=1130,y=120,width=30,height=27)

            #Delete trails#
```

```
398.            self.deltrailb = Button(self.master,text="Delete
    Trails",command=deltrail,fg="green",activeforeground="green")
399.            self.deltrailb.place(x=1040,y=120,width=80)
400.
401.            #Planet colour chooser#
402.            self.planetcolour = ((255,0,0),"#FF0000") #default planetcolour
403.            self.colourchoose = Button(self.master,text="Select
    colour",command=lambda:getcolour())
404.            self.colourchoose.place(x=1040,y=300,width = 120)
405.
406.            self.education = Button(self.master,text="←",command=educationmode)
407.            self.education.place(x=1075,y=90,width = 30)
408.
409.            self.planetanchor = Button(self.master,text="⚓",command=anchor)
410.            self.planetanchor.place(x=1040,y=90,width = 30)
411.
412.            #load and save#
413.            self.loadfunct = Button(self.master,text="Load
    file",command=lambda:load(False))
414.            self.loadfunct.place(x=1110,y=415,width=60)
415.
416.            self.quickloadfunct = Button(self.master,text="Quick
    Load",command=lambda:load(True))
417.            self.quickloadfunct.place(x=1030,y=415,width=70)
418.
419.            self.savefunct = Button(self.master,text="Save
    file",command=lambda:save(False))
420.            self.savefunct.place(x=1110,y=385,width=60)
421.
422.            self.quicksavefunct = Button(self.master,text="Quick
    save",command=lambda:save(True))
423.            self.quicksavefunct.place(x=1030,y=385,width=70)
424.
425.            #toggle stars#
426.            self.stary = Button(self.master, text="Toggle Stars on",
    command=self.startoggle,fg="Red",activeforeground="red")
427.            self.stary.place(x=1040,y=150,width=120)
428.
429.            ###Planet specific variables###
430.            self.window.create_rectangle(1020,50,1180,190,fill="Light Grey")
    #toggle function box
431.            self.window.create_rectangle(1020,675,1180,760,fill="Light Grey")
432.            self.window.create_rectangle(1020,200,1180,340,fill="Light Grey")
433.            self.window.create_rectangle(1020,350,1180,450,fill="Light Grey")
434.            self.window.create_rectangle(1001,460,1250,470,fill="BLACK")
435.            self.window.create_rectangle(1020,480,1180,660,fill="Light Grey")
436.            self.window.create_text(1100,63,text="Misc
    Functions",font=("Helvetica",10,"bold "))
437.            self.window.create_text(1100,366,text="Load and
    save",font=("Helvetica",10,"bold "))
438.            self.window.create_text(1100,215,text="Planet
    Properties",font=("Helvetica",10,"bold "))
439.
440.            #Mass#
441.
442.            self.mass = DoubleVar()
443.            self.mass.set(100)
444.            self.window.create_text(1060,247,text= "Mass",font=("Helvetica", 10))
445.            self.Mass = Entry(self.master,width=10,textvariable=self.mass)
446.            self.Mass.place(x=1100,y=240)
447.
448.            #Density#
449.
450.            self.density = DoubleVar()
451.            self.density.set(20)
452.            self.window.create_text(1065,277,text=
    "Density",font=("Helvetica",10))
453.            self.Density = Entry(self.master,width=10, textvariable=self.density)
454.            self.Density.place(x=1100,y = 270)
```

```
455.
456.            #Trail Duration#
457.
458.            self.trailduration = IntVar()
459.            self.trailduration.set(0)
460.            self.window.create_text(1100,700,text="Trail duration\n      (20 =
    ∞)",font=("Helvetica",10,"bold"))
461.            self.trailduration =
    Scale(self.master,from_=0,to=20,orient=HORIZONTAL,bg="light
    grey",highlightthickness=0)
462.            self.trailduration.place(x=1050,y=715)
463.
464.            #Force Amplification
465.
466.            self.speed = 1 #force amplification
467.            self.window.create_text(1090,80,text="Amp")
468.            self.speed =
    Scale(self.master,from_=1,to=5,resolution=0.01,variable=self.speed,orient=HORIZON
    TAL,length = 50,width=20,bg="light grey",highlightthickness=0)
469.            self.speed.place(x=1110,y=70)
470.            #Planet variable showing#
471.            self.planetselected = 0
472.
473.            self.window.create_text(1100,495,text="Planet
    Information",font=("Helvetica",10,"bold"))
474.
475.            self.window.create_text(1062,520,text="Velocity")
476.            self.planetvelocity = IntVar()
477.            self.showoffvelocity =
    Entry(self.master,width=6,textvariable=self.planetvelocity)
478.            self.showoffvelocity.place(x=1100,y=513)
479.            self.planetvelocity.set(0)
480.            self.showoffvelocity.configure(state="disabled")
481.
482.            self.window.create_text(1062,550,text="Mass")
483.            self.planetmass = IntVar()
484.            self.showoffmass =
    Entry(self.master,width=6,textvariable=self.planetmass)
485.            self.showoffmass.place(x=1100,y=540)
486.            self.planetmass.set(0)
487.            self.showoffmass.configure(state="disabled")
488.
489.            self.window.create_text(1062,572,text="Density")
490.            self.planetdensity = IntVar()
491.            self.showoffdensity =
    Entry(self.master,width=6,textvariable=self.planetdensity)
492.            self.showoffdensity.place(x=1100,y=567)
493.            self.planetdensity.set(0)
494.            self.showoffdensity.configure(state="disabled")
495.
496.            self.window.create_text(1062,605,text="Devorered\n  planets")
497.            self.planetsdevoured = IntVar()
498.            self.showoffdevoured =
    Entry(self.master,width=6,textvariable=self.planetsdevoured)
499.            self.showoffdevoured.place(x=1100,y=595)
500.            self.planetsdevoured.set(0)
501.            self.showoffdevoured.configure(state="disabled")
502.
503.            self.window.create_text(1062,635,text="Time alive(s)")
504.            self.planetalivetime = IntVar()
505.            self.showoffalive =
    Entry(self.master,width=6,textvariable=self.planetalivetime)
506.            self.showoffalive.place(x=1100,y=622)
507.            self.planetalivetime.set(0)
508.            self.showoffalive.configure(state="disabled")
509.
510.            #undo
511.            self.undo = Button(self.master,text="⟲",command=mainstack.undo,width
    = 2,height=-5,font=("Helvetica",10))
```

```
512.            self.undo.place(x=1070,y=15)
513.
514.            #redo
515.            self.redo =
    Button(self.master,text="↻",command=mainstack.redo,width=2,height=1,font=("Helvet
    ica",10))
516.            self.redo.place(x=1110,y=15)
517.
518.
519.
520.            ###    TITLE    ###
521.            alphabet = []
522.            for alpha in range (65,91): alphabet.append(chr(alpha))  #something
    like that
523.            self.systemname = ("PYxis - System:
    {}{}-{}{}{}".format(alphabet[random.randint(0,25)],alphabet[random.randint(0,25)]
    ,random.randint(0,9),random.randint(0,9),random.randint(0,9))) #Standard string
    consentraition methods leave ugly curly brackets.
524.            self.master.wm_title(self.systemname)
525.
526.        def startoggle(self): #make these shift all in one direction at some
    point
527.            if self.stary["text"] == "Toggle Stars off":
528.                self.window.delete("star")
529.                self.stary["text"] = "Toggle Stars on"
530.                self.stary.config(fg="Red",activeforeground="Red")
531.            else:
532.                for stars in range(0,1000):
533.                    for starsperline in range(0,1): #make this variable
534.                        ran = random.randint(0,1000)
535.                        colourcode = random.randint(0,255)
536.
    self.window.create_oval(ran,stars,ran,stars,outline=('#%02x%02x%02x' %
    (colourcode,colourcode, colourcode)),tags="star")
537.                self.stary["text"] = "Toggle Stars off"
538.                self.stary.config(fg="Green",activeforeground="Green")
539.
540.            self.window.lower("star")
541.        def clickfunct(self,event):
542.            try:
543.                floatmass = float(self.mass.get())
544.                floatdensity = float(self.density.get())
545.                nmass = round(abs(floatmass))
546.                ndensity = round(abs(floatdensity))
547.                if event.x < 1000:
548.                    self.ox = event.x
549.                    self.oy = event.y
550.
    self.window.create_oval(event.x+(nmass/ndensity),event.y+(nmass/ndensity),event.x
    -(nmass/ndensity),event.y-(nmass/ndensity),fill=self.planetcolour[1],tags="shotov
    al")
551.            except TclError:
552.                print("ERROR: Mass and/or density values are invalid.")
553.        def motion(self,event):
554.            self.window.delete("shot")
555.            colour = self.planetcolour[1]
556.            if event.x < 1000:
557.
    self.window.create_line(self.ox,self.oy,event.x,event.y,fill=colour,tags="shot",a
    rrow="last")
558.
559.        def release(self,event):
560.            if event.x < 1000:
561.                x = event.x
562.                y = event.y
563.                cx = event.x - self.ox
564.                cy = event.y - self.oy
565.                self.window.delete("shot")
566.                try:
```

```
                    lmass = int(self.mass.get()//1)
                    ldensity = int(self.density.get()//1)
                    if lmass < ldensity:
                        lmass = ldensity
                        self.mass.set(lmass)
                    radius = lmass / ldensity
                    if radius > 250:
                        radius = 250
                    end = [x,y]
                    start = [self.ox,self.oy]
                    rad,theta,xneg,yneg = maths(start,end)
                    vx = cx / lmass
                    vy = cy / lmass

createplanet(round(radius),lmass,self.ox,self.oy,self.planetcolour[0][0],self.pla
netcolour[0][1],self.planetcolour[0][2],vx,vy,theta,False,0)
                except TclError:
                    pass
            self.window.delete("shotoval")
        def select(self,line):
            if self.planetselected > len(OBD) - 1:
                self.planetselected = 0
            if line == self.planetselected:
                self.window.delete("s")
                self.selected = self.window.coords(OBD[line]["planet"])
                self.selectoval =
self.window.create_oval(self.selected[0]-10,self.selected[1]-10,self.selected[2]+
10,self.selected[3]+10,outline = "yellow",stipple="gray75",tags="s" )
                self.window.lower(self.selectoval)

    ui = userinterface()

    def updateUI():
        ui.window.delete("educat")
        if ui.planetselected in anchorlist:
            ui.planetanchor["text"] = "⛵"
        else:
            ui.planetanchor["text"] = "⚓"
        ui.window.itemconfig(ui.curtime,text=("Time",round(time.time() -
ui.starttime,2)))
        red = (round((255/20)*ui.trailduration.get()))
        ui.trailduration.config(fg='#%02x%02x%02x' % (red,round((255-red)/2),0))
        if len(OBD) > 0:
            ui.planetvelocity.set(math.sqrt(((OBD[ui.planetselected]["dx"])**2) +
((OBD[ui.planetselected]["dy"])**2))*1000)
            ui.planetmass.set(OBD[ui.planetselected]["mass"])
            ui.planetdensity.set(OBD[ui.planetselected]["mass"] /
OBD[ui.planetselected]["radius"])
            ui.planetalivetime.set(round(time.time() -
OBD[ui.planetselected]["alivetime"]))
            ui.planetsdevoured.set(OBD[ui.planetselected]["planetsdevoured"])
        try:
            ui.window.itemconfig(ui.fps,text=("FPS:",
round(1/(time.time()-ui.otime))))
        except ZeroDivisionError:
            pass #if the time change is too low
        ui.otime = time.time()

    def safepause():
        if ui.tobepaused == True or ui.tobepausedcolour == True:
            if ui.tobepausedcolour == True:
                playpause(True)
                askcolour(ui.prevpaused)
                playpause(False)
                ui.tobepausedcolour = False
            else: playpause(False)

    #keybinds

```

```
627.    ui.window.bind("<Button-1>",ui.clickfunct) #initial click
628.    ui.window.bind("<B1-Motion>",ui.motion) #click and drag
629.    ui.window.bind("<ButtonRelease-1>",ui.release) #release of click
630.    ui.window.bind("<Button-3>",selectobject)
631.    ui.window.bind("<Button-2>",lambda x:deleteplanet(ui.planetselected,False))
632.
633.    while True:
634.        if ui.paused != True:
635.            updateUI()
636.            main()
637.            if len(poplist) > 0:
638.                popplanets()
639.            safepause()
640.            ##Move the planets in time with the rest of the program.##
641.            #I decided not to function this to save on calling globals##
642.            for line in range(0,len(OBD)):
643.                ui.select(line)
644.                if line not in anchorlist:
645.
    ui.window.move(OBD[line]["planet"],OBD[line]["dx"],OBD[line]["dy"])
646.                    oldxy =
    [((OBD[line]["x0"]+OBD[line]["x1"])/2),((OBD[line]["y0"]+OBD[line]["y1"])/2)]
647.                    try:
648.
    OBD[line]["x0"],OBD[line]["y0"],OBD[line]["x1"],OBD[line]["y1"]=
    ui.window.coords(OBD[line]["planet"])
649.                    except ValueError:
650.                        pass
651.                    newxy =
    [((OBD[line]["x0"]+OBD[line]["x1"])/2),((OBD[line]["y0"]+OBD[line]["y1"])/2)]
652.                    if ui.trailduration.get() > 0:
653.                        drawtrail(oldxy,newxy,line)
654.        ui.window.update()
```