Title:

H663-03 Programming Project

Centre: 26360

Candidate: 8080

# Contents

[Contents 2](#_Toc92871368)

[Analysis 3](#_Toc92871369)

[Introduction 3](#_Toc92871370)

[Stakeholders 3](#_Toc92871371)

[Using computational methods 4](#_Toc92871372)

[Likely data structures/paradigms 5](#_Toc92871373)

[Likely interface/interaction 5](#_Toc92871374)

[Research 5](#_Toc92871375)

[Requirements 10](#_Toc92871376)

[Essential features 10](#_Toc92871377)

[Success criteria 10](#_Toc92871378)

[Limitations 12](#_Toc92871379)

[Design 12](#_Toc92871380)

[Decomposition 12](#_Toc92871381)

[Testing plan 27](#_Toc92871382)

[Development 27](#_Toc92871383)

[Final Testing 27](#_Toc92871384)

[Evaluation 28](#_Toc92871385)

[Appendices 28](#_Toc92871386)

[Final Code Listing 28](#_Toc92871387)

# Analysis

## Introduction

[DO LATER]

## Stakeholders

* Secondary school physics teachers
* Most physics teachers will have an in depth understanding of science and the physics surrounding gravity between different objects, because of this, I can make the systems in place to create simulations quite specific with a lot of the variables exposed to user control to allow for the teacher to set up a simulation to demonstrate exactly what they want to show. They will likely be above 30 years old as most of them will have spent significant time at University studying for their degree in their field. Potentially even staying longer for a PhD. However, the age range is large as physics teachers doesn’t have an age limit so they could potentially be very old. Because of this, the UI needs to be intuitive and easy to understand and read, the fonts will likely have to be quite simplistic and easy to read. Also, since they are less likely to be used to using the keyboard to control movement and other parts of the simulation, I would likely create UI elements to allow for the control of the camera as well as the various other functions that will be usable in the project. They will be large and the colour for the font will be adjustable to allow for more user control. Also, since teachers are usually quite busy, I will likely create a way to save configurations so that they could be loaded more easily. This would also allow teachers to create more complex systems without having to worry about having to recreate the same simulation each time. Creating a system for saving simulation information would also likely allow for a revert implementation allowing a teacher to demonstrate a certain part of a simulation multiple times.
* They will be teaching a class of students who are trying to learn about gravity between objects. These students will not understand the workings of gravity and will likely be using the project as a learning resource to get a better understanding. Because of this, the UI will likely need to be able to easily show the relevant information about planets in both a visual and a more exact way, likely text based as well. They will understand the English language and will be a lot younger, usually around 17 as the gravity topic is covered in year 13 physics. Because of this, the UI will need to likely be written in simpler terminology and in a way that makes it a lot easier to understand what is happening on the screen. These students may want to use the simulator themselves to they can better understand gravity between objects. Because of this, I will likely allow a lot of the UI to be adjustable to the user preference such as font colour, size (to an extent). Also, because they are significantly more likely to have a grasp of keyboard controls common in videogames, I would likely create keyboard shortcuts as well to allow for the quick and easy viewing of the simulation if you are used to that control scheme.

## Using computational methods

### Problem recognition

The problem with this project is that it needs to be able to do more than just calculate the physics. It also needs to be able to display the results in an easy to understand and grasp way. Making sure to demonstrate the necessary information for learning about gravity. Whilst also not being to be overloaded with information as to overwhelm those trying to use the program. And allowing the user to have full control of the simulation so they can create demonstrations to teach their class effectively.

### Decomposition

When working on this project, decomposition is going to be vital to the creation of this project as planets are there are quite a few separate parts to this project and keeping the separate parts separate, I should be able to keep the project more organised and create a more elegant and efficient solution. The project in total should be able to be split into 3 main sections, each of which will focus on 1 specific thing that needs to be able to happen within the simulation, with a few more things also created to connect the 3 sections into 1 working program.

### Abstraction

Since planets are massive, abstraction will also be a necessary component in the process of creating this project, because planets are very complex structures, I will need to create a model for the planets and the forces between them only containing the relevant data. Each planet will be defined by simple data relevant to its movement in space such as position, velocity, and mass. The simulation will also model the planets as perfect circles and won’t go into 3d, since that adds a potential unnecessary level of complexity to the simulation and may make it more confusing t

### See the source image Pipelining

Pipelining will be a very important part of this program running properly due to the nature of writing this project and the nature of LWJGL (Light Weight Java Game Library), the library I will be using to render the planets. LWJGL uses the GPU to handle rendering and uses a pipeline to render objects to the screen. This allows the developer using LWJGL to have nearly full control over the full rendering process. This will be useful for my project as it allows me to optimise the process of programming by only passing information down the pipeline to relevant points, so un-necessary information is not passed down when it isn’t needed. Furthermore, it allows me to modify specific parts of the program without having to worry about modifying other parts of the pipeline as each part of the pipeline functions independently.

### Visualisation

Since the program is designed as a visualisation of data. The data being the positions of generated planets as they interact with each other. Visualisation is going to be paramount to the success of this project. Since the planets are modelled as a circle, visualisation of the planets should be fairly simple, visualising the velocities of the planets could be done in real time by watching as the planets move, but the program could also draw an arrow on the planet to visualise the direction and magnitude of the velocity, a similar technique could be used to visualise the force acting on the planet. These both should be togglable in the event that the arrows begin to get intrusive.

## Likely data structures/paradigms

My design concept will heavily lend itself to the object-oriented paradigm, as there will be an unknown number of planets, all of which need to be able to simulate gravity and collision based on their own individual attributes. Furthermore, the program needs to be able to both add and remove planets quickly. And using an object-oriented approach for this will make managing all these a lot simpler. Also, I am using a library called LWJGL (Light Weight Java Game Library) to render the planets directly as it provides greater control over the rendering process. This allows for the rendering to be a lot more efficient and render at a much higher rate. However, due to the complexity involved in setting up and using LWJGL, an object-oriented approach makes it a lot easier to use LWJGL for any significant project.

## Likely interface/interaction

The main program will render the planets as circles with 1 colour. The circular planets are because most large celestial objects in real life are sphere-like and a circle is also the simplest shape to apply physics to. The 1 or 2 colours will be chosen by the user. This is so that, whilst the GUI is simple to render and look at. It is still visually appealing, so people still want to use it. When it comes to controlling various parts of the simulation, I would like to have both key bindings and GUI buttons for most of the functions. This is so that the users can control the simulation in whatever way they feel most comfortable with and they have a constant reminder of what they can do with the simulator. These buttons on the screen will have clear visual indicators to what state they are in to make it clear at a glance what the simulator is doing at any given point in time. The buttons would respond to being clicked on simply because this is convention with buttons in GUI and most text boxes would be editable when clicked on. This is as that would be a simple method of changing the value of something in relation to the planet.

## Research

### Asking stakeholders

I will be asking my stakeholders a questionnaire and using their responses to decide how I will be building the simulator. I will be asking these questions to my 2 physics teachers as well as another physics teacher that I used to have as they are the main stakeholders in this project. I will likely be asking these questions in the form of an online questionnaire, to allow me to easily send it to multiple teachers.

The questions will be:

##### Questions:

1. When learning how to use the simulator, what would be the easiest way to learn

-The controls are displayed as a pop-up that can be re-opened

-The opening menu has a “tutorial” option that takes you through the various controls

-Both

-Other (with text box to specify)

2. What tools would be useful for creating simulations (multiple choice)

-Generating a blank universe with no planets

-Generating a universe with a few starting pre-set systems to choose from

-The ability to save and reload user-made systems

3. What control scheme would be easiest to use

-Buttons displaying the functions that could be used (i.e., Delete planet in bottom left etc.)

-Key bindings (i.e. Press the delete key on your keyboard to delete a planet)

-Both

-Other (with text box to specify)

4. What information should be displayable to be used in teaching

-Forces applied by gravity (separate, as arrows)

-Direction of acceleration (as arrow)

-Direction of velocity (as arrow)

-The path the planet has taken (planets have a trail that slowly fades)

-Other (with text box to specify)

5. Would you want the gravitational constant to be fixed, or changeable by the user (so planets can be closer and shown in more detail)

-Yes

-No

-Maybe

6. What time-based tools would be useful in this simulator (multiple choice)

-Pausing the simulation

-Speeding up/slowing down the simulation

-Rewinding the simulation

-Restarting the simulation from the point it was initially generated

7. What colour scheme would you want

-Text box

8. How would you like planets to be differentiated from one another

-1 colour

-2 colours with a gradient between them

-Other (with text box to specify)

Questions 1, 2 and 3 are related to the user, and how they will use the simulator. I am trying to make this a useful tool so making it easy to use for physics teachers who may potentially not be that well versed with technology is important, therefore giving the simulator simple controls and an easy way to learn them as well as making the process of creating new simulations as trivial as possible are things I need to make sure are considered properly. Questions 4, 5 and 6 are related to the users using my program to teach physics. Since the simulator will be used as a teaching tool, making sure that any information the teachers think would be necessary can be displayed, and any information I may not necessarily thought of can be put in as well. Questions 7, & 8 are in relation to the UI and look of the program. If the program looks clunky and poorly made, people are likely not going to be able to gleam relevant information from it and people are much less likely to use it. Likewise, if the planets are difficult to differentiate, then the tool will be unhelpful as keeping track of a planet and demonstrating what it will do will be more difficult.

##### Results:

Amongst the teachers I asked, people generally wanted both a pop-up and a tutorial to learn to use the project, amongst them, only 1 person felt the ability to add a blank universe with no planets would be useful. And a couple pre-sets that were suggested were a realistic solar system, just the inner solar system, and just the sun, the moon and the earth. Most people found that buttons displaying the controls on the screen would be the easier method of controlling the simulator. But 2 people still decided that having key bindings would be useful to control the simulation, one person even saying they would prefer to use key bindings.

In terms of what the simulation would display, everyone thought that displaying the forces on the planets in vector form would be valuable, with most people feeling the direction of velocity would also be a valuable piece of information. Multiple people also said they wanted the planets to leave a “path” where they have gone so you can see if they have a stable orbit or not. It was also suggested to have the simulator display the masses of the planets, so that students could see them easier, and the distances between the planets. When deciding if people wanted the gravity constant to be changeable by the user, or fixed to a value, it was almost an even split between people with 2 wanting it to be changeable and 3 wanting it to be fixed. Everyone wanted the simulation to be paused, sped up/slowed down, and be rewindable, and all but 1 person wanted the simulation to be restarted to the point when the simulation was initially created.

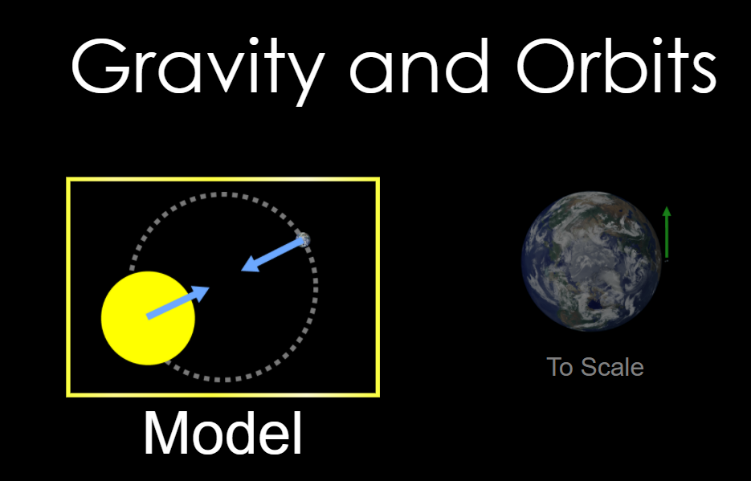
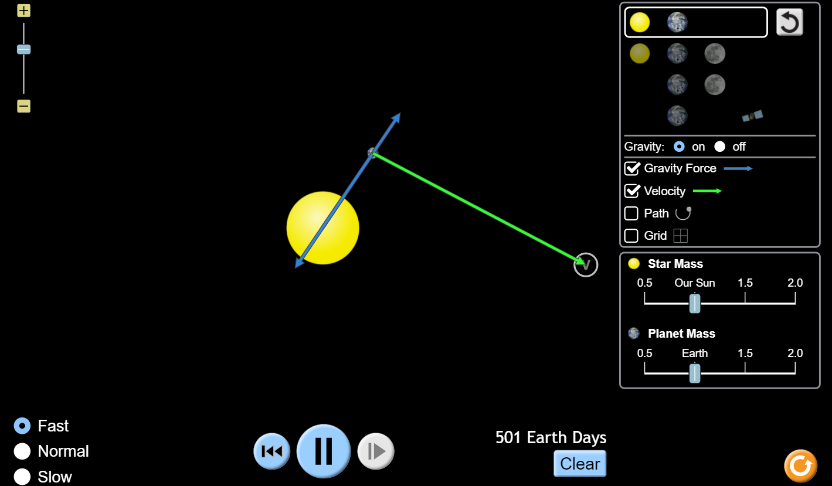
In terms of the look of the program, most people wanted the planets to simply be represented as 1 solid colour, with 1 person wanting it to be represented as 2, and someone suggested the ability to give planets names to more easily distinguish them. And Most people generally wanted a dark colour theme for the UI, with a bit of variance on the secondary colour, being either white, yellow or light green. One person suggested that there should be a higher contrast mode for accessibility.

### Existing designs

There are multiple online variations on a gravity simulator:

#### Phet interactive simulations

##### What it is

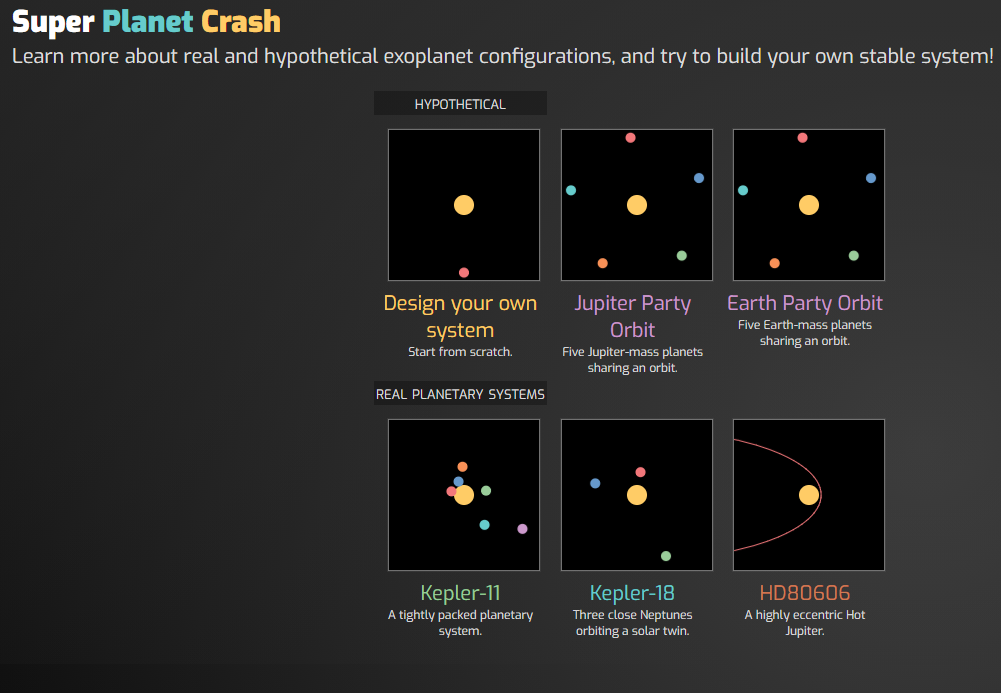
Phet interactive simulations, a part of the University of Colorado, have created a solar system simulator. It is online, so you can access it easily. When opening the website, it gives you 2 options between a pre-created, to-scale solar system between the sun, the earth and the moon, and a model system where they are generated a lot closer than is realistic. These are purely where the objects start and have no bearing on the actual simulation itself. When you load up the simulation, it gives you 4 object configurations based on models you are likely to find in real life, it also gives you a simple and effective UI showing everything that needs to be used, this is all cleanly spaced around the edges of the screen out for the way so the user can still see the simulation, the UI shows clearly which options can be enabled, disabled and what you have accessible, it allows you to demonstrate the vector of both the force of gravity and also the velocity. It also allows you to display other things on the screen such as the path the various objects have and a grid to allow you to get a better sense of scale. Around that it allows you to configure certain factors that will affect the physics such as the mass of the various objects and even turning gravity off. On the bottom of the screen, you can do various inputs to affect the speed of the simulation as well as pausing it completely, and resetting the simulation back to the start, as well as showing the time elapsed.

##### What I can apply

* I could create some starting pre-sets to allow users to more easily learn the various tools and parts of the application
* I could base the UI design for my simulator off the UI used in this simulator
* I can use the visualisation of gravity and velocity as arrows to allow for velocity and gravity to be better demonstrated by a teacher
* The use of arrows to represent information such as velocity or forces

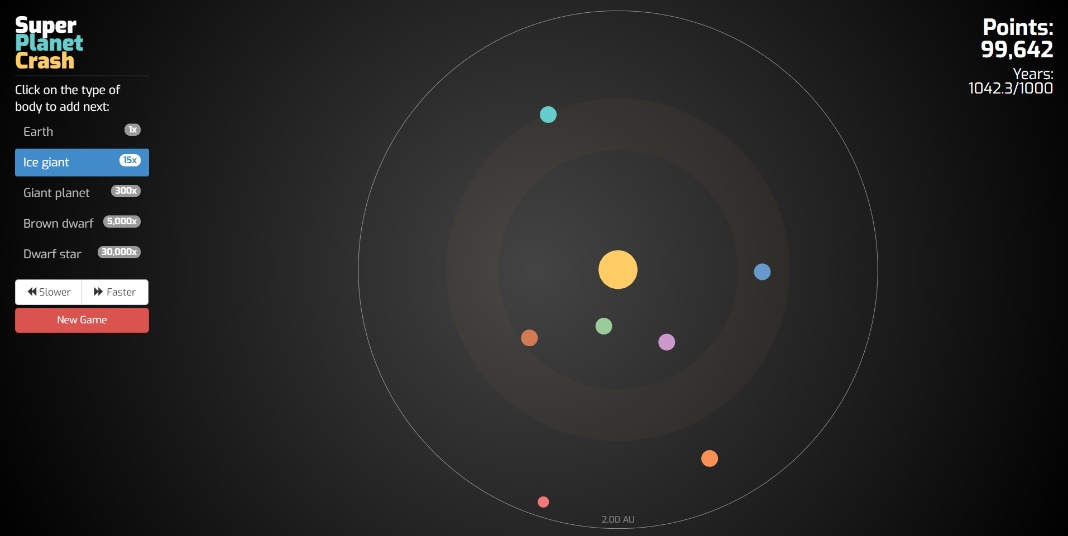
#### Super planet crash

##### What it is

Super planet crash is a game in which a solar system is accurately simulated in which you must try to balance multiple planets in a stable orbit. There are 6 different prebuilt planets of increasing mass and each of them carries an increasing point multiplier, the simulation allows for a varying number of planets, up to a maximum of 11 not including the central star that the planets all orbit. You must try to get as many points as possible over a course of 1000 (simulated) years or until one planet strays outside the area.

When the simulator is started up, you get given 6 potential options, 3 hypothetical models and 3 based off 3 real planetary systems. Selecting one of these generates the configuration as shown in the image. From there you can add additional planets to the system (up to 12). These planets orbit one central large body and are affected by each other’s gravity. Planets are added with a menu at the side and then the user clicks on the screen to generate the planet at their cursor. The planets will already have a pre-determined velocity so that they don’t just fall into the centre object. There is a tool in the bottom left corner that allows the user to speed up or slow down the simulation, as well as generate a new simulation. The planets also have a trail effect where it will draw a trail that will stay for a certain amount of time however this trail breaks if the speed of the simulation is high enough.

##### What I can apply

* Having several starting systems would be valuable for users
* Having a variable quantity of planets would allow for more freedom for users
* Having a trail for planets to show their orbits would allow users to demonstrate a stable orbit and what that looks like.
* Having the time be variable allows the less relevant parts to be skipped over
* The use of dark UI elements makes the whole simulator look more visually appealing

## Requirements

### Software

* JVM (Java runtime environment)
  + I will need this as the project is being coded in java
* OpenGL version 3.3 or above
  + I will need this as it is the graphics API that I am using to render the planets

### Hardware

* Monitor to view the program
* Mouse and keyboard to interact with the program
* …Gb of RAM to run the program
* …Gb of free storage space to store the program as well as any save files created
* …Processor to process the simulation
* A graphics card (integrated or otherwise) to render the simulation

## Essential features

The simulation needs to be able to simulate a variable number of planets, each with their own information on position, velocity, size and mass, to allow the users to be able to create their own teaching tools.

The simulation needs to be able to access and modify individual planet’s information, as well as more universal values such as the gravity constant, likely using a visual UI, to allow simulations to be created effectively, and as some stakeholders said they would find the ability to modify the gravity constant useful.

The simulation needs to be able to save, load and create new simulations to allow the user to more effectively use the program for demonstration purposes. This will also include several premade simulations as feedback suggested that users would like to have several simulations based on our solar system ready to use.

The simulation needs to have arrows drawn onto planets to allow for users to visually tell what is going on in the simulation, like the ones in the Phet interactive simulations.

The program needs to be able to effectively teach the user how to use itself so they can get more practical use out of the program.

The simulation needs to be able to run at varying speeds, including paused and backwards

The program will need to be able to render all relevant information to the screen in an easy-to-read way.

## Success criteria

### Quantitative

1. The program needs to be able to open a separate window
2. The program needs to be able to render: Rectangles, circles, images, text.
3. The program needs to be able to modify the colour of all rendered objects
4. The program needs to be able to detect where the user’s mouse is on the window and if the user’s mouse buttons are down.
5. The program needs to be able to detect what key inputs are happening at any given point.
6. The program needs to be able to use user input to detect if the user has clicked a button on the window
7. The program needs to be able to carry out a function specific to a button if that button has been clicked.
8. The program needs to be able to use user input to input from a list of acceptable characters into a text box, if it has been recently selected.
9. The program needs to run at a known rate.
10. The program needs to be able to change that rate at any given point
11. The program needs to be able to use the update rate in physics calculations to allow the simulation to run in real time.
12. The program needs to be able to add, remove, read and write to and from a database storing the simulations and planets
13. The program needs to provide a GUI to allow the user to effectively navigate the various pages of the program
14. The program needs to be able to calculate the position of a variable number of planets using known velocities and calculated acceleration
15. The program needs to be able to detect when 2 planets would intersect and alter their positions and velocities, so the planets collide properly.
16. The program needs to be able to calculate the force exerted on each planet by every other planet due to gravity, and convert that into a vector acceleration
17. The program needs to be able to calculate the physics, update the positions and render the scene at an acceptable rate (30 updates per second at a minimum)
18. The program needs to be able to render arrows to represent the vector velocity and force exerted on each planet.
19. The program needs to be able to add, remove and alter planets while a simulation is not running.
20. The program needs to provide a GUI to allow the user to alter planet information, or add and remove specific planets
21. The program needs to provide a GUI to allow the user to alter universal constants such as the gravity constant, with ability to read scientific notation.
22. The program needs to provide a GUI to allow the user to pause, speed up or slow down the running of the simulation
23. The program needs to provide a GUI to save built simulations, as well as load an already built simulation
24. The program needs to provide a list of pre-built simulations such as out current solar system
25. The program needs to provide a list at the user’s request on how to use the program
26. The program needs to provide a basic tutorial on using the program

### Qualitative

1. The program’s GUI needs to be readable, accessible and modifiable by the user.
2. The program needs to be simple and intuitive to use in its layout
3. The GUI needs to be clean and nice looking to the user
4. The different pages should be clear and distinct in their function, and their design and layout should reflect this.
5. The tutorial should be clear and easy to follow
6. The simulator should be clear and a valuable tool for learning about the interactions between objects due to gravity.
7. The collisions between objects should be reasonably realistic

## Limitations

* I will not be implementing a way to move planets around using the mouse as it would be potentially too complex and could make it more difficult to just click on a planet without displacing it slightly.
* I will not be implementing sound effects as they are unnecessary and would make the program more difficult to use as a teaching tool due to potential distractions
* I will be adding an upper limit to the number of planets you can generate at once so the user cannot generate so many planets that the program crashes. This upper limit will likely still be high, so users are unlikely to run into the upper limit unless they are trying to.
* The program will only run on a computer and won’t be on mobile. This is because I don’t have the necessary tools to develop on mobile. And since this is supposed to be a teaching tool, the program on mobile won’t be much use.

# Design

## Decomposition

### Navigating the program

Diagram

Description automatically generatedThis is a flowchart of how the user will navigate the main different areas of the program. There are multiple areas where the program can cycle between 2 or more menus, so it is important that menus have a state then open in which doesn’t persist between times opening. I’ve used the feedback from stakeholders to include several additional features I may not have otherwise such as the use of loading and saving of simulations and having a controls list and a tutorial. The program first starts in the main menu, this allows the user to more easily see the options available and access the tutorial and controls list more easily.

During development, the program will likely open to the “Build sim” section of the program, as that is the first part being developed, this is because the simulation building, and running is the core of the program and the most vital part. It is also the part of which most other features such as Loading, saving and creating a new simulation will be built on. The reason I am going to have the openable menu is because it allows the simulator to put all the parts it may need to navigate to in 1 place that isn’t intrusive the user. The main menu is the central “hub” where the user can navigate to access all the main features and is openable directly anywhere on the simulation, using this approach will make navigating the program simpler for the user. When loading a simulation, whether a new or pre-made one, the program will have to safely clean up the old simulation before loading the new one in its place. In multiple parts, such as between the building and running simulator and the controls list and the tutorial, the user will be able to move back and forth between 2 different sections. In the case of the simulator, being able to test a simulation to make sure it works quickly is going to be vital to allow the user to more effectively create simulations. And in the case of the controls list and the tutorial, being able to reference the controls list easily when trying to follow the tutorial would be helpful for the users to learn how to use the program.

The first part of the program I am going to be building is some of the engine processes that need to happen for the program to run, this is because they will lay the groundwork that the other parts of the program will be based off of. Then I will create the running of the simulator, since this is the core of the program and will be built upon by the building simulator that I will code next. Since this is the part the main menu will open into it is needed before working on the other sections, additionally, since the build and run sim can be thought of as 2 parts of a greater simulator. After the simulator is built the loading and saving of simulations will be created since that is closely related to the building simulations section of the program and I cannot create it earlier since it will have nothing to load the simulations into, after that, I will build the main menu as that is the central part that links up all the features the user will need to use, and since a larger amount of the options on the menu will be implemented, I will be able to test the menu navigates to the right area more effectively. After the menu, I will need to build the controls list and the tutorial for the user to be able to learn how to use the program. These are being done last as they require the build sim to already be created so I can write how to use it more effectively and they will be accessible from the menu. Which page is open is likely going to be controlled by a page manager class that will contain which page is open as well as methods to open new pages and start sections of the program with the relevant information that page will need, for example, the build and running simulation will need the list of planets they are displaying/simulating.

### Engine functions

There will be several parts to this project that need to be present in many of the separate parts of the program, these will be necessary for the running of the separate parts and will need to be built first.

#### GUI

The GUI is a very fundamental part of the project as it is the primary way the user will be interacting with the program. Because of this it is almost imperative I build this first. Since the API OpenGL doesn’t provide any GUI elements, I was required to create my own from scratch. For organisation reasons and since the GUI code uses a lot of OpenGL use I have decided to implement the GUI code in a separate library and import it into the project. The different types of GUI I need to implement are: static coloured rectangle, static images, text, buttons, text boxes, and checkboxes. To allow for inheritance and polymorphism within the different GUI elements, there will also be a base class “GUI” that will contain all the fields and methods that will be consistent across all GUI types, which can be overridden to include any additional methods that are needed. This means when I am using the GUI, the GUI for a specific page can be iterated through without having to do several iteration calls.

#### GUI loader

The GUI loader is a class that will contain all the different GUI layouts for all the different sections of the project, since the number of GUI objects will be fixed within each window of the program, the different GUI elements will be stored in arrays, and organised as each window had 1 array containing all the relevant GUI. I can store multiple different GUI classes in the same array because all the different GUI elements are all children of 1 parent class “GUI”. This allows all the GUI elements in a page to be iterated through when they are being rendered or checked for updates.

#### Page manager

Since the program will have several “pages” all with their own purpose, GUI and other functions, I am going to need a class to encapsulate the process of switching between different pages of the program, so the proper GUI can be loaded, and any other initialisation can be set up to allow the user to properly use the program. The PageManager class will be handing the loading and initialisation of the separate pages, including the retrieving of GUI from the GUI loader, resetting the GUI to default settings, retrieving the list of created planets in the case of the simulator and cleaning up anything that was in the old page.

#### Timer

The timer is going to be used to regulate the frame and tick rate of the program. And keep track of how much time has elapsed so that the simulator can run in real time, to allow the changes in the timestep to not affect the accuracy of the simulation, the update rate also needs to be modifiable, and the timer needs to be able to continue operating during this.

#### Database functions

The program is going to need to be able to store the simulations long term and for that I am going to be using SQL, to keep the code more organised I am going to encapsulate the database use within a separate class to allow me to better organise my project.

### Simulation

The simulation is a larger area of the code that can be split into several smaller sections: Building the simulation, running the simulation and file handling

#### Building the simulation

##### UI layout

Diagram

Description automatically generated

This is the layout for the GUI I will likely have, with some example planets drawn in for an example, the UI is designed to be clearly visible and readable, but also not intrusive or in the way of anything the user may want to do. The largest potential obstruction is the planet information as it takes up a larger chunk of space on the screen then everything else. Due to this the selected planet information will not be visible unless the user has a planet selected and will not be rendered and non-responsive until the user selects a planet. The delete selected planet will also not be rendered and be unresponsive when a planet isn’t selected as it is also planet specific. The rest of the UI will be constantly active when the build sim mode is active, as none of it does functions specific to a planet and will always be potentially required. The planets will be rendered underneath the UI and will not the given their own canvas, this is more of an aesthetic choice and because a lot less of the screen will be taken up when a planet isn’t selected, a lot of the time the space the canvas isn’t in will be un-used, and having the canvas resize when a planet is selected would be confusing and harder to follow. I’ve used a similar organisation to the Phet interactive simulation for my UI as the central area of the screen was more visible, and that is the place the user is going to be most comfortable looking.

##### Text Description automatically generatedPseudocode

Because of the modularity of the program, and the use of object-oriented methods to perform all the code in an organised manner, a lot of the pseudo code is made up of executing methods in other classes. Additionally, several classes will be present in other parts of the program such as timer or input. These will be defined during the initialisation code of the program and therefore do not need to be defined here. Additionally, the UI will be pre-created for this in the PageManager, a class I will create to handle the different pages and the planets will be loaded from either a saved configuration the user selected to load, or will just be empty if the user wanted to create a new simulation. I’ve also not got rendering in the pseudocode as that will be handled in either a main class or the PageManager class. Since I don’t want the user to be able to accidentally click on a planet whilst in the pause menu, the first thing the code does is check if the pause menu is open. If it isn’t, it checks if the left mouse button is down, since this is how the user will select UI elements and planets, if it is, it checks the mouse position, since I will be using pixel coordinates for both UI and the mouse position it is fairly simple to check if the mouse is within a UI box.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

`

For the planets however, since the camera moves and zooms independently of the planets, I need to transform the mouse position first into camera space which is between -1 and 1 on the horizontal and vertical axis. And then to world space by using the inverse of the camera’s transformation matrix. Since the planets are circles, not squares I check if the mouse is less than the planet’s radius away from the centre of the planet. If the user doesn’t have planet or UI element selected, it’s respective variable will be null. If the UI element is a functional UI (i.e. a button that does something, like create or delete a button) it carries out its function. After that it checks for any key presses, if it detects any, and the UI selected is a text input box, it checks if the key pressed is acceptable for a key box, if it is, and it is the backspace button, it removes the most recent

character, otherwise, it adds that character to the end of the string and updates the relevant information.

##### Key variables

|  |  |  |
| --- | --- | --- |
| Name | Data type | How it is used |
| Planets | Arraylist of Planet classes | Will be used to contain all the planets so that they can be iterated through when checking what planet has been selected, when planets are added or deleted, they will be added or removed from this Arraylist, so it needs to be dynamically sized. |
| UIElements | Arraylist of UIElement classes | Will be used to contain all the UI elements so they can be iterated through when rendering them and when finding what UI element has been selected, to carry out their function and so the user can modify planet information |
| PlanetSelected | Pointer to a planet class in Planets | Will be used to point towards the planet the user has selected, so the user can modify specific, individual planets. Will be null when UI or nothing is selected. |
| UISelected | Pointer to a UIElement class in UIElements | Will be used to point towards the UIElement the user has selected, so the UI can carry out its function and textboxes can be typed in. |
| changeMade | Boolean | Will be set to true if something has changed, so un-necessary re-calculations of planets aren’t done |
| manager | PageManager class | Will be used to manage which page the user will have open, navigate between pages and manage what different pages need so that the user can easily navigate between different pages |
| camera | Camera class | Will contain a position and scale vector to allow the user to change the position they are viewing the simulation from |
| timer | Timer class | Will contain the desired frame and update rate to allow the simulation to run at a fixed rate. |
| input | Input class | Will handle the OpenGL key and mouse inputs from the user to use the program. |
| window | Window class | Will handle the OpenGL window to allow the user to see what they are doing |

#### Testing the simulation

##### UI layout

This is the UI layout I will have for running the simulation, it is similar in layout to the building sim with the layout, this is to make the change between building and running the simulation as simple and intuitive as possible. I have kept several pieces of UI between both of them, such as the selected planet information, however, when running the simulation, the planet information will not be editable, as that would cause potential inconsistencies between different demonstrations of the simulation. It will still only be rendered when a planet is actually selected, as it takes up a lot of the screen and won’t actually have any information on it when no planet is selected. Additionally, it will feature a new button that allows the user’s camera to follow a planet as it moves, this is to allow the user to keep track of a planet and potentially demonstrate what orbits would look like from the perspective of one of the planets (for example, why it looks like the sun is orbiting us).

There are also now arrows on each of the planets to display information such as their velocity and acceleration. This is to visually represent the information relevant to planets in more context. This is because whilst the numbers can be useful to getting a more exact idea of what is going on, having arrows that point in the direction of velocity or acceleration can make the information much easier to understand. They will both be togglable with the UI in the bottom right of the screen to allow the user to get rid of them if they are irritating or distracting. Additionally, the UI at the top is significantly different from the simulation builder UI. The elements are smaller to fit the additional UI needed. The gravity constant has been moved there now. This is because it is no longer going to accept user input, so that simulations remain consistent, and moving the UI and changing the format written will signify to the user that it is a different piece of UI. The time controls have also been put in the top of the screen, just so it remains central to the view of the user. It will feature a pause button, as well as 2 buttons that will speed up or slow down the UI. These will be to allow the user to slow down and see more important interactions in more detail, or skip uneventful periods of time more quickly. Additionally, the stop button is in the centre to make it easier to locate.

##### Pseudocode

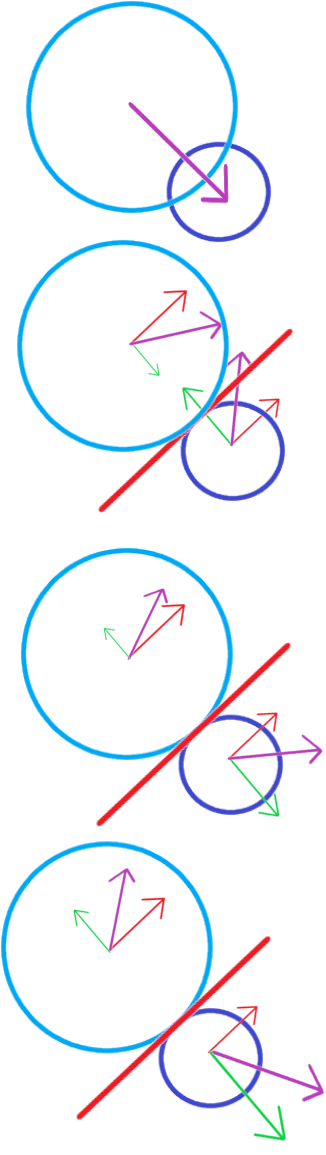
This code shares significant similarities with the building simulation code, mainly the bulk of the code is user inputs, because of this, when I am coding, I may put the general camera controls in the camera class itself, to better help organisation. I will likely keep the controls specific to the testing simulator in the testing simulator itself. Additionally, many of the functions will be in separate classes and called by the main testing loop. The code also shares the planet and UI select code as the building simulation code, but without the ability to input text, this is because there isn’t any UI that requires text input, only actual buttons, so checking if the button is a text input is unnecessary.

##### Key variables

|  |  |  |
| --- | --- | --- |
| Name | Data type | How it is used |
| Planets | Arraylist of planet classes | Will contain all the planets created in the testing sim and used by the Physics class to simulate the physics |
| UIElements | Arraylist of UIElement classes | Will contain all the UI related to the simulation tester and can be iterated through to take inputs to affect how the simulation runs, to convey information to the user, and to be rendered to the screen. |
| PlanetSelected | Pointer to Planet class in Planets | Will be used to get which planet the user currently has selected, so they can follow it, and see information relevant to it. |
| view | Integer | Will point towards the planet in Planets that the camera is supposed to be following, will be -1 for free roam mode |
| timeStep | float | Will be a scalar multiplier for how fast time passes in the simulation, when negative time will pass backwards |
| paused | boolean | Will be used to stop the physics of the simulation running when it is true, to allow for the simulation to stop |
| manager | PageManager class | Will be used to manage which page the user will have open, navigate between pages and manage what different pages need so that the user can easily navigate between different pages |
| camera | Camera class | Will contain a position and scale vector to allow the user to change the position they are viewing the simulation from |
| timer | Timer class | Will contain the desired frame and update rate to allow the simulation to run at a fixed rate. |
| input | Input class | Will handle the OpenGL key and mouse inputs from the user to use the program. |
| window | Window class | Will handle the OpenGL window to allow the user to see what they are doing |
| physics | Physics class | Will contain all the methods used to calculate the new velocity each planet should have after gravity and collisions have taken place |

#### Physics

The physics will be a background process in the simulation to control the velocities and therefore movements of the planets, it will handle the force of gravity between each planet and calculate the new momentums of 2 different planets in a collision. The physics will work in 3 steps



The physics finds the vector difference between the position of planetA and planetB, if the distance between them is less then the 2 radii added, they are colliding, it calculates the ratio of masses and moves both planets by a scalar found using how much they are intersecting and the ratio of masses, so that the larger planet is displaced less then the smaller planet. This is to stop the larger planet being as significantly affecting by potentially large amounts of planets hitting it

In a collision between 2 circles, it can be modelled as if each circle had collided with a line that is at a normal to both circles, to calculate the velocity after the collision has occurred you would need to find the vector components of the current velocity that are parallel and perpendicular to the wall. I found these using the vector difference calculated earlier normalized. In the collision, assuming the collision is perfectly elastic, the parallel component is kept constant, and double the perpendicular component is subtracted, to reflect the perpendicular velocity properly rather than just nullifying it, in an inelastic collision you would also multiply the perpendicular velocity by a scalar multiplier to reduce the overall velocity after the collision. however, that assumes that the wall is fixed, which it isn’t, and that both planets will have the same mass, which they may not.

To deal with this issue, I won’t double the perpendicular component before subtracting it, and will then use the other planet’s velocity to affect the new velocity, this will allow some of the momentum to be transferred rather than generated and should be more accurate to actual physics. And to deal with the variable masses I am using the mass ratio doubled to affect the amount each planet’s velocity will be a factor in the collision, so the larger mass planet will be more impactful than the smaller planet. It is doubled so the net velocity ratio between them equals 2 for both planets, rather then 1 so velocity isn’t lost in the collision.

##### Pseudocode

When creating the physics, I wanted each of the interactions between planets to affect both planets, rather than only affecting one. This is because when the planets interact, they reuse information frequently, so recalculating that same information again is inefficient. This reduces the number of iterations I need to perform and should improve the speed of the program with larger numbers of planets. The physics first needs to clear the acceleration of the previous update since it isn’t important anymore. It then iterates through each combination of planets calculating the acceleration each interaction has and resolving any collisions that have occurred. After which it iterates through the planets again to update their velocity. Because planets have their acceleration changed before they are used in the outer loop, the acceleration clearing needs to be written separately, but since the starting value for b is incrementing each time the loop is called, once the nested loop is finished, all interactions with that planet have been made, so the velocity can be updated then, rather then creating another loop.

##### Key variables

|  |  |  |
| --- | --- | --- |
| Name | Data type | How it is used |
| Planets | Arraylist of planet classes | This will be iterated through to apply the physics of the simulation to all planets. |
| gravityConstant | float | A constant scalar of the force created by gravity between planets |
| timeStep | float | A scalar of how quickly time will pass relative to real time |

#### Planets

I will create a class “planet” that will act as a container for all relevant data about a planet to be accessed by the simulation. I can do this because I am using an Object-Oriented language.

##### Key variables

|  |  |  |
| --- | --- | --- |
| Name | Datatype | What it is |
| mesh | Mesh class | Mesh class will be a wrapper to contain all relevant information pertaining to creating, rendering and cleaning up meshes. |
| position | Vector2 | The vector position of the planet relative to world space |
| velocity | Vector2 | The vector velocity of the planet relative to world space, will be used to influence position |
| acceleration | Vector2 | The vector acceleration of the planet relative to world space, will be used to calculate changes in velocity for that update and cleared in the next update |
| size | float | The radius of the planet, used in collision calculation and |
| mass | float | The mass of the planet, used in most physics calculations |
| colour | Vector3 | The colour of the planet the user selected, stored in vector form (x,y,z)->(r,g,b) |

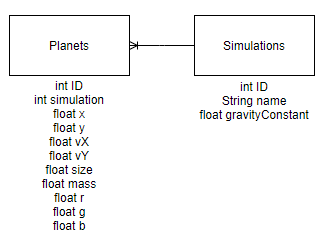
##### Key methods

|  |  |  |
| --- | --- | --- |
| Name | Returns | What it is |
| clearAcc() | null | Sets the acceleration of the planet to 0, so new acceleration can be calculated |
| addForce(Vector2 force) | null | Adds an acceleration to a planet using the equation acceleration=force/mass |
| updateVelocity(float timeStep) | null | Adds the acceleration multiplied by timeStep |
| render(Program program) | null | Renders the planet to the screen using the Program wrapper program |
| cleanup() | null | Cleans up the mesh before the planet is destroyed to prevent memory leaks |

### Simulation saving and loading

Because of the limitations of SQL datatypes, the file management systems will need to be able to convert between the usable planets in the simulation, and a format that is more saveable to a database.

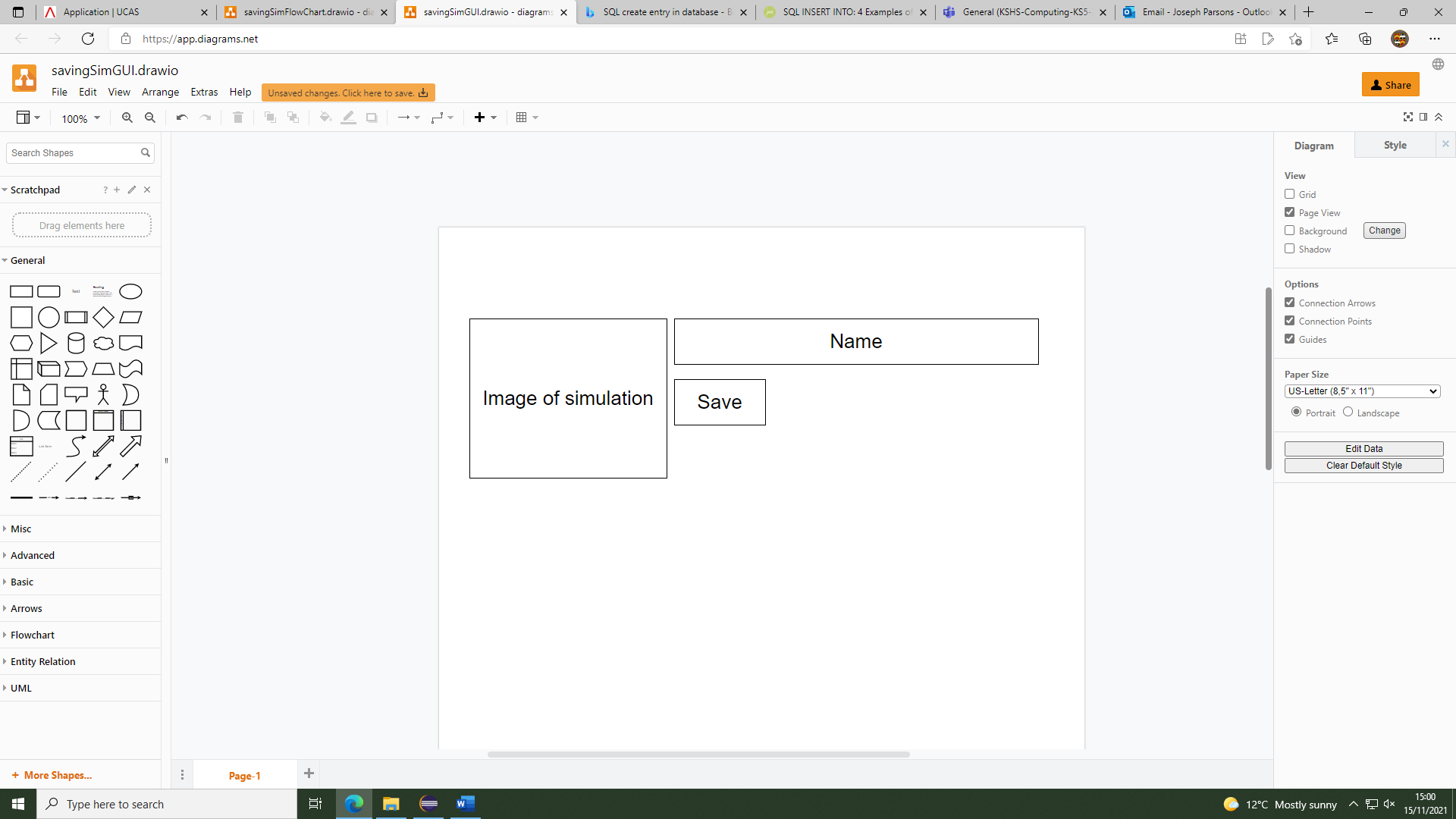
#### Storage format



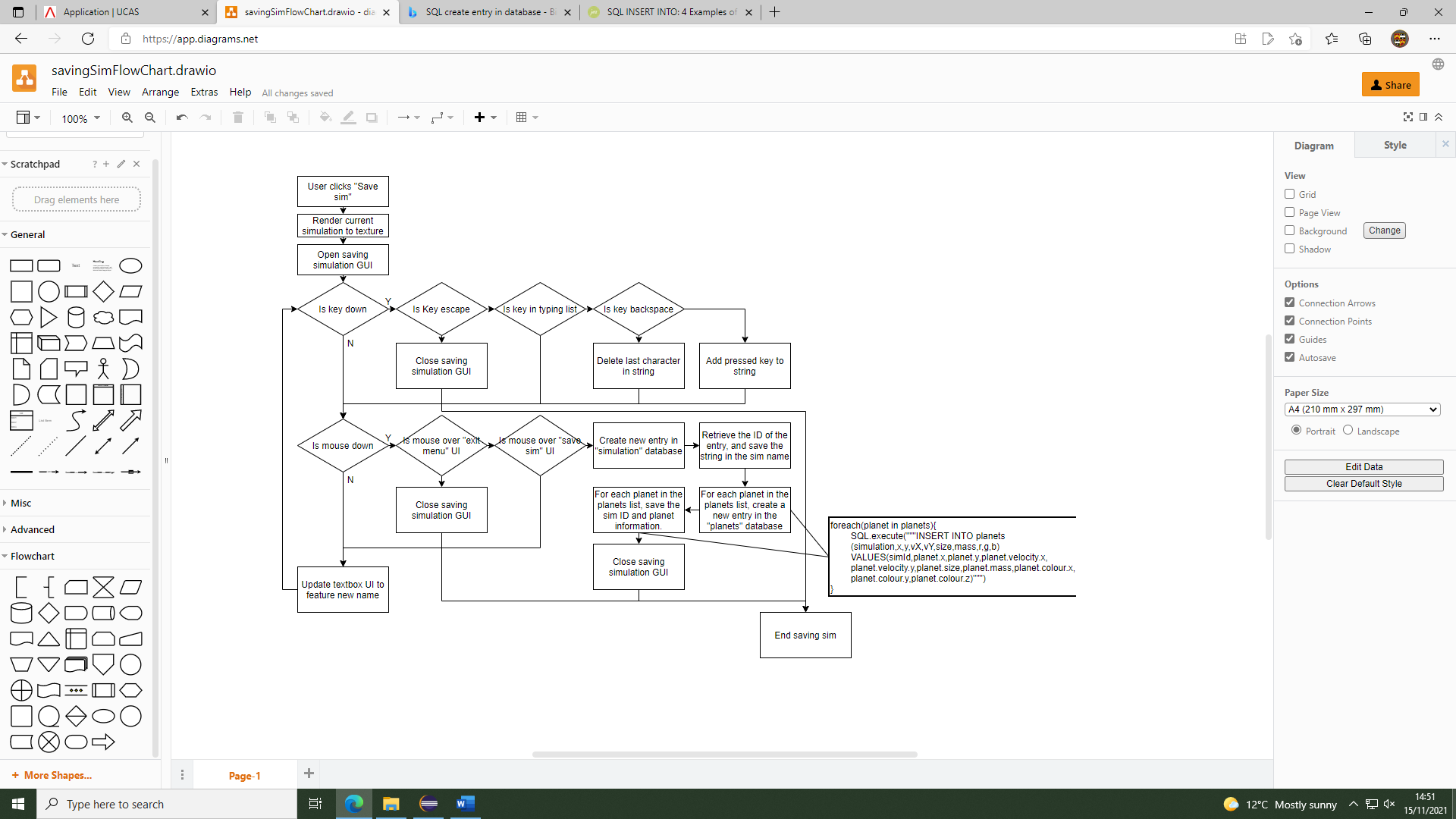
Since the planets and simulations are the only things that need to be saved, the relations table is relatively simple, with only 2 objects and a one-to-many relation between them. The planets table will have to be quite large, but that is mainly because SQL doesn’t support vectors because they are a class created in the library I am using, so I will have to convert between floating point values and the vector format. Planets overall will need to be able to retain their initial position (vector2), velocity (vector2), size, mass, and colour (vector3). Most of these values are used in the physics calculations, and colour is used to allow the users to be able to differentiate between separate planets in the simulation. They will also need to be able to store which simulation they are a part of, since they will have a many to one relationship with the simulations table. The simulations table needs to keep track of any values that are non-planet specific. So that will contain any non-planet specific information that needs to be saved. This is the name of the simulation to help users properly identify different simulations. The gravity constant for the simulation also needs to be saved, and since it is universal, it is saved in the simulation itself, rather then in a specific planet.

#### Saving simulation UI

The button the user must click to start saving a simulation will be in the main menu. Clicking this will open the saving pre-set GUI. This will feature a box to type in a name that users can use to better distinguish between different saved simulations, and a button for the users to click to save the simulation, I would also like to include a smaller image of the simulation in the GUI as well.

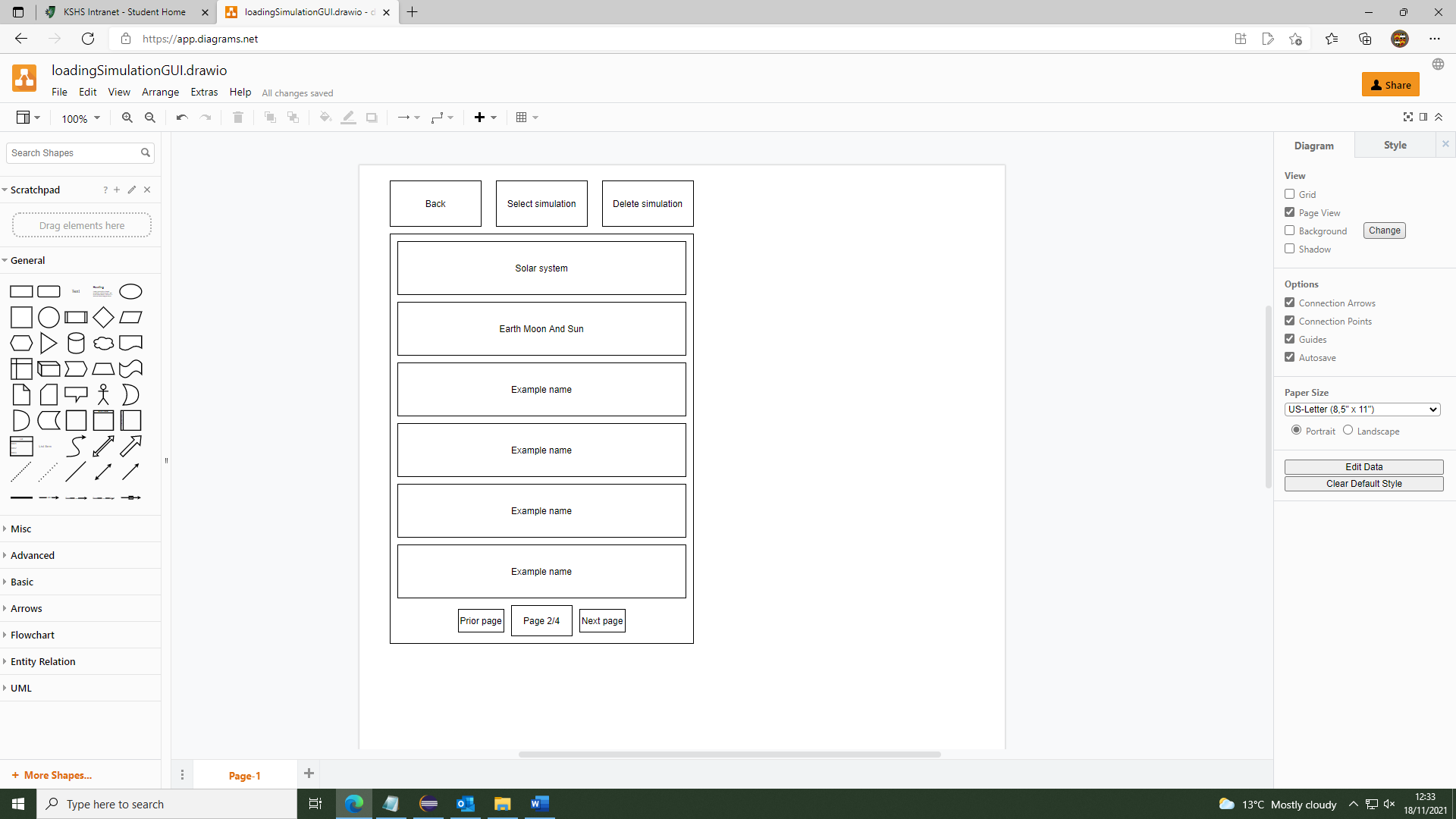


#### Saving simulation flowchart



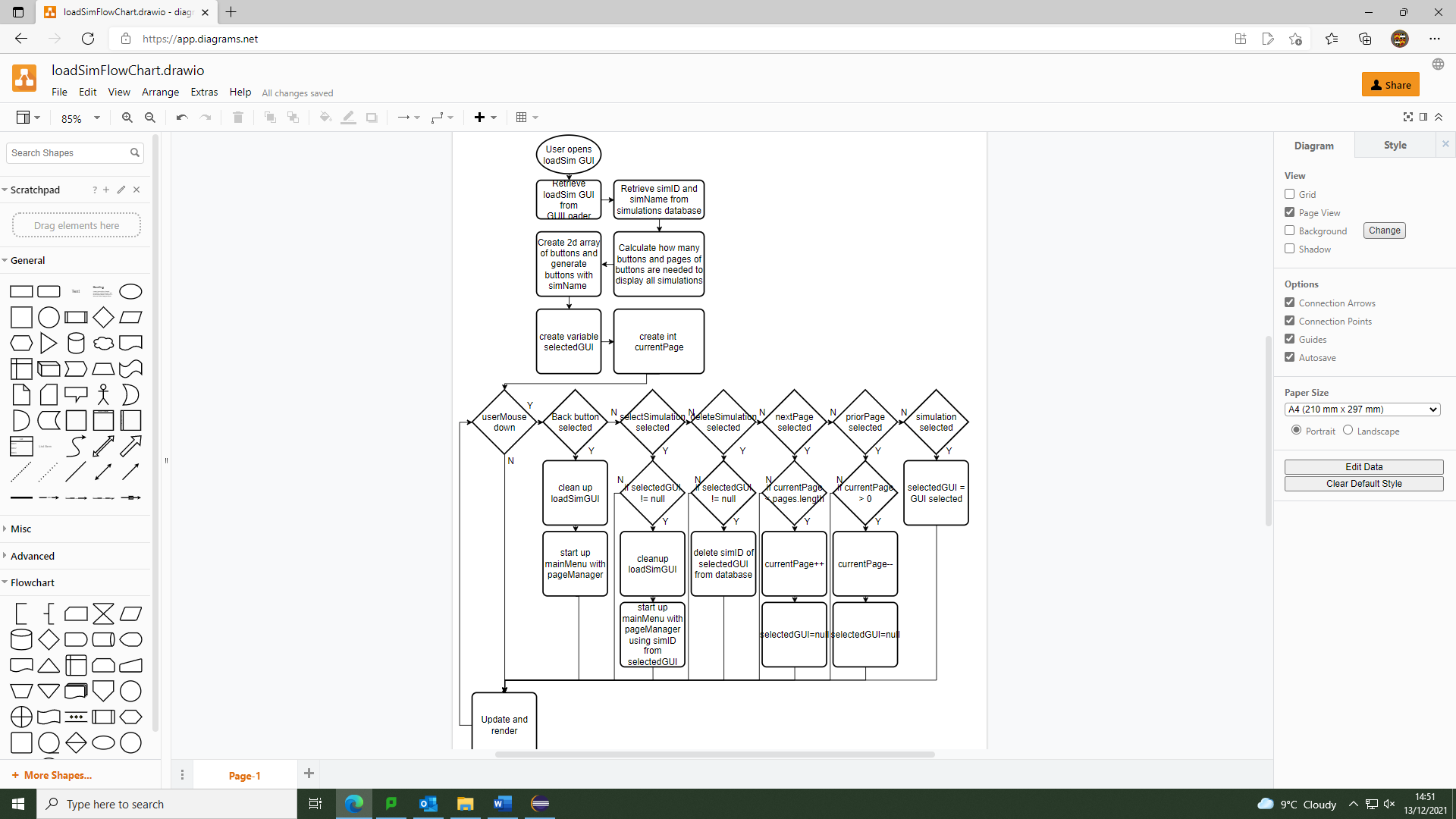
Saving the simulation is relatively simple, since there is only 1 text box, I don’t need to check if the text box is selected or not. Since a lot of the GUI and SQL will be handled by separate libraries. The code will be a lot simpler. The SQL that will be used to send the information to be saved will not send in the planet ID, since that is auto-incrementing, and is needed to store each planet’s unique ID. When the database tries to create a new simulation, it will need to retrieve the simulation ID so that planets can be properly linked to the correct simulation.

#### Loading simulation UI



The loading simulation UI will display all the different simulations, in a list similar to this, with their names in a readable size font. This is so that the users can easily tell which of their simulations they are selecting from the list. The simulations will all be separate buttons, with only 1 selectable at a time, this is to make the process of deleting a simulation more involved so that users cannot accidentally delete a simulation, I will be having all of the simulations done in pages, rather then having a scrollable GUI, this is to simplify the program as having the UI scrollable would be potentially difficult. The different simulations will be organised in order of creation, this is to make it easier for the user to know where a simulation they have created has gone.

#### Loading simulation flowchart

The loading simulation GUI should be fairly simple, mostly just going through the buttons on screen and checking if the user has selected them. When the program generates the 2d array of buttons, I will be able to use the position of the button in the 2d array to find the position of the corresponding simID that was retrieved from the database. Additionally, when detecting user input and rendering, I only need to iterate through the button son the current page, rather then the full array, since buttons on a separate page won’t appear or be selected. Additionally, since so many of the functions regarding the database and the GUI are encapsulated within separate classes, the actual code will primarily be utilising functions created by separate classes.

## Testing plan

During development I will be continually testing parts of the program as they are created to be sure that when problems occur, I know where these problems will be and will be more effectively able to solve them. Additionally, different sections of the project will be largely encapsulated and independent of each other, which will aid in testing as less of the code will be reliant on potentially non-functional code.

### GUI

The GUI testing will largely exist in isolation to the final product, due to it being fairly fundamental to the function of the rest of the code and being one of the primary methods I will be inputting data to the program in other parts of the simulation, it is imperative if functions properly.

|  |  |  |  |
| --- | --- | --- | --- |
| Test number | Success criteria | Details of test | Expected result |
| 1 | 1, 2, 3 | I will open the window of the simulator and have it render a rectangle, an image and some text | The window will open and the text will render without issue |
| 2 | 2, 4, 6 | I will have the program render a text box and input a short string of “TEST:002” on a keyboard, the text box will be set up to accept letters and numbers | The text box will be unselected and therefore will not allow keyboard input |
| 3 | 2, 3, 4, 5, 6, 8 | I will have the program render a text box as specified in test 2. Click on the box and that input the string “TEST: 003” on a keyboard | The text box will change colour when clicked to indicate it is selected, and fill in “TEST003” when the keyboard inputs are given, only accepting the characters specified during it’s creation as acceptable |
| 4 | 3, 4, 7 | I will have the program render a button. The button will be set up to print a statement “TEST: 004” when clicked, I will then clock the button | The program will print “TEST: 004” when the button is clicked |
| 5 | 3, 4, 7 | I will have the program render a switch button using 2 images, the switch will be set to print “TEST: 005” when active. I will then switch the button on and off repeatedly | The button will switch images every click as well as switching between being active and inactive. When active the program will repeatedly print “TEST: 005” |
| 6 | 2, 4, 6 | I will have the program render an button as specified in test 5. I will click in several places around the button but not on it directly | The button will not print anything since it isn’t clicked |

### Running simulations

Testing the simulation running will be done using planetary configurations I am creating specifically for the test, since I have yet to build the section of the program that will let the user build their own simulations. Since testing if the physics of the simulation seems realistic

# Development

# Final Testing

# Evaluation

# Appendices

# Final Code Listing

# 