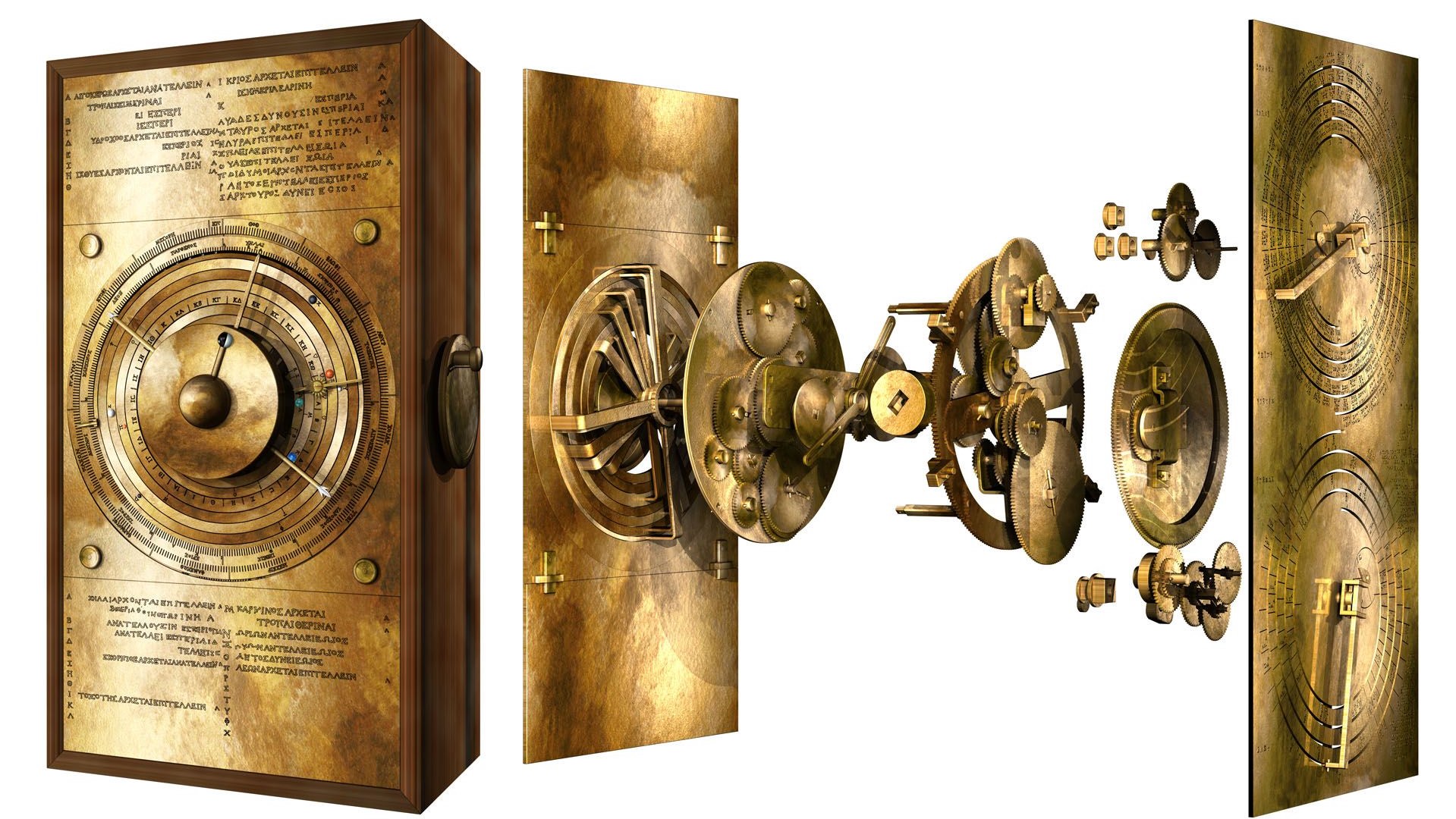
Antikythera Mechanism Simulation

Candidate Name: Kostas Kanakaris

Candidate Number: 2426

Centre Name: Heckmondwike Grammar School

Centre Number: 37463



Contents

[Analysis: 2](#_Toc323778466)

[Introduction: 2](#_Toc982603842)

[Case Study 1: 3](#_Toc659480108)

[Case Study 2: 3](#_Toc685333936)

[Client Overview: 4](#_Toc367577951)

[Interview: 5](#_Toc1797091884)

[Analysis of the Interview: 6](#_Toc145753517)

[Contact with experts: 7](#_Toc1733011128)

[Mind Map of Preliminary Solution: 8](#_Toc819947094)

[Systems Flowchart: 8](#_Toc284505363)

[Analysis of Potential Solutions: 9](#_Toc1518065719)

[Mathematical Background and Research: 10](#_Toc809735948)

[Final Analysis Objectives: 12](#_Toc1521157578)

[Design: 13](#_Toc747641436)

[Overview: 13](#_Toc158036530)

[General UI: 13](#_Toc1045082916)

[Design (Generation): 15](#_Toc339787726)

[Fields (inputs) for planets: 15](#_Toc464530289)

[Fields (inputs) for stars: 17](#_Toc379561279)

[Class Diagram: 18](#_Toc826186496)

[Pseudocode Design (Generation of planet): 18](#_Toc2123486638)

[Advanced Mathematics: 20](#_Toc1039623445)

[Polar Notation: 20](#_Toc296580923)

[Matrices: 20](#_Toc594217528)

[Validation: 22](#_Toc1709070593)

[Parsing: 24](#_Toc311872470)

[Regex: 27](#_Toc705326314)

[Technical Solution: 28](#_Toc859856340)

[MainWindow.xaml: 28](#_Toc996937234)

[MainWindow.xaml.cs: 35](#_Toc505633450)

[Classes (Folder containing classes): 49](#_Toc1158279300)

[BackgroundEntities.cs: 49](#_Toc1331744449)

[Body.cs: 50](#_Toc1879186435)

[EntityRegex.cs: 51](#_Toc55971346)

[GaseousPlanet.cs: 53](#_Toc823268468)

[Parser.cs: 55](#_Toc1212376576)

[Star.cs 57](#_Toc1770584123)

[TerrestrialPlanet.cs 59](#_Toc456830644)

[World.cs: 62](#_Toc1513749379)

[Assets (Folder containing images): 64](#_Toc531585255)

[border.png 64](#_Toc497979339)

[callippicdial.png 64](#_Toc1765633935)

[corinthian.png 65](#_Toc1429185516)

[deathstar.png 65](#_Toc732052554)

[dyson.png 66](#_Toc1843073217)

[english.png 66](#_Toc1699961040)

[exeligmosdial.png 67](#_Toc1502263842)

[olympiaddial.png 67](#_Toc1564632466)

[sarosdial.png 68](#_Toc1519671485)

[sothic.png 68](#_Toc970434235)

[bin: 69](#_Toc186319486)

[Debug: 69](#_Toc1554343415)

[App.xaml: 71](#_Toc1345999205)

[App.xaml.cs: 72](#_Toc85875254)

[packages.config: 72](#_Toc1696288327)

[Testing: 72](#_Toc1129740773)

[Formulation of Testing Strategy: 73](#_Toc2098043727)

[Testing Evidence: 73](#_Toc1848909802)

[Evaluation: 90](#_Toc1241503987)

[Feedback from client and primary users: 90](#_Toc2045415634)

[Objective Evaluation: 91](#_Toc308192461)

# **Analysis:**

## **Introduction**:

This project will be a 2D simulation of the Antikythera mechanism - the first ever analogue computer, which can be used to help students understand how the ancient mechanism worked. However, I will also add the ability to generate your own solar systems, to give the simulator increased interactivity and usability and a modern twist. The simulation will therefore offer a user-friendly interface where students can not only observe the intricate movements of the Antikythera mechanism but also actively participate in creating their own solar systems. Guided tooltips will assist users in navigating the simulation and understanding its features, ensuring accessibility for learners of all levels. This includes input labels explaining what datatype should be used.

I will decompose this simulation into 2 overarching segments later in the design and technical solution phase: the procedural generation of celestial bodies and their interactions with each other, and then the dials and mechanisms to do with time-based developments.

The simulation should have 4 secondary dials: the Olympiad dial, the Saronic dial, the Callippic dial and the Exeligmos dial, along with the primary Month dial. I want the Month dial to have 3 languages: English, Sothic and Corinthian, as the traditional Antikythera mechanism did. There should be a separate Planet Generate and Star Generate panel on the left as well for generating our own solar systems, but our own real solar system should be included as a preset.

I will make a quick table that outlines what I will aim to accomplish. I will make a more detailed objectives one later in the design phase.

## **General Objectives:**

|  |  |
| --- | --- |
| Features: | Reason for inclusion: |
| Solar System Modelling | Central and most important part of the simulation. Without the solar system being graphically represented the simulation will be severely limited in its educational potential. This also requires A-level further maths knowledge such as vectors and matrices. |
| Central dial & Sub-dials | This was the core of the Antikythera mechanism's mathematics being applied. I can define a constant for each of the dials and give values in relation to each other. |
| Generate Custom Planets | Custom planet generation is also very important to add diversity to the solar systems that can be generated. Already a problem that I anticipate is the complex nature of terrestrial planets as they have many parameters that define their appearance. |
| Generate Custom Stars | Custom star generation is ostensibly less important, however with counteracting gravitational fields it could be challenging to model the planets too. I will try to add custom stars, however it may be too difficult to model certain star configurations (i.e binary and trinary systems). |
| Load Presets | This is important as it will allow users to save and open other users' custom solar systems which could improve usability and increase collaboration in the classroom. |

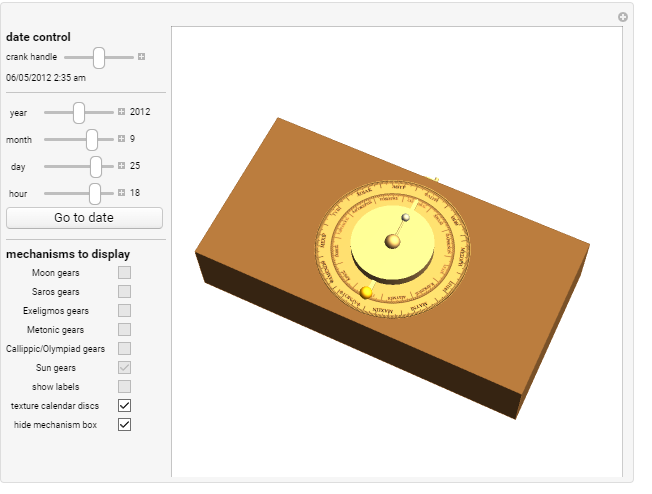
### Case Study 1:

All Antikythera mechanism simulations that I found were overly simplistic, such as this one: This simulation was programmed in Unity.



It is simplistic and does not provide a realistic simulation as it only shows the sun, moon and earth. There is no mention of other planets, despite the fact that they were known at the time. Also, the UI is very lacklustre, along with being difficult to navigate. I also think that the 3d element of it is distracting and is not required to accurately depict the mechanism. Furthermore, the dials are on the bottom. While this is historically accurate, on this simulation, you have to awkwardly pan the camera around to the bottom to see the dials, and you cannot look at the top planetarium simultaneously. Therefore, I have learned from this case study that I should have all the dials on the screen at once. One thing that this simulation does do correctly is the rotation slider. Compared to Case Study 2, which moves at a fixed rate, this simulation allows you to change the speed at which the dials move. This is also useful because it can let you speed up/slow down at will and even stop the simulation.

### Case Study 2:

This simulation is also 3D and is made by Wolfram Alpha. 

Wolfram is known for their complex mathematical knowledge engine. However, this algorithm is even simpler than the previous case study and is not intuitive to use. It is very choppy in terms of frames when running it. Also, you must manually crank the handle by moving the slider up and down. This is impractical and it also does not allow you to set the handle to move by itself at a fixed rate, like Case Study 1. Something that is unique and interesting, however, is the ability to input a date and see what the mechanism would look like. This could be an interesting implementation for my program, although I am not sure if it would work due to the procedurally generated solar systems. It could definitely be done for the solar system preset, though.

Overall, both of the case studies were helpful but for different reasons. It is reassuring to see that my project has at least been attempted in the past, albeit not very well. I may try to get in contact with the person who created the first model to ask some further questions, as he wrote a thesis on the Antikythera Mechanism.

## **Client Overview:**

My client is Mr Rieper, a computer science and physics teacher who is interested in the Antikythera mechanism. In lesson, he wants to teach more about our solar system. Also, he wants to be able to create his own planets and solar systems and represent them using the Antikythera mechanism. My primary users are Mr Rieper's pupils that will also be given access to the Antikythera mechanism simulator in the form of an executable file on their computers. This project's primary users are intended to be physics students, but history students and engineering students are secondary users also. This is because of the nature of the program - it uses primarily the subset of maths called mechanics to work which is very close to physics.

## **Interview:**

Me: "Why does the Antikythera mechanism interest you?"

Mr. Rieper: "Because it was one of the first analogue computers and as such is historically valuable. Also, it has many practical applications as nowadays many space simulations use many concepts that were included in the Antikythera mechanism.

Me: "Would you like all of the different cycles (Saronic, Exeligmos etc.)?"

Mr. Rieper: "Yes because it would help the students understand exactly how the mechanism worked and also it shows the relationship between the years, months and days. Some dials will obviously be longer like the Exeligmos dial in comparison to something like the Olympiad.

Me: "Would you like the English, Sothic or Corinthian dials?"

Mr. Rieper: "The English dial should be the basic dial but the other ones should be available as a selection because they were the original languages, and so this would increase the historical accuracy of the simulation.

Me: "Would you like to create your own solar systems?"

Mr. Rieper: "Being able to create custom solar systems would allow students to experiment with different planetary configurations and make their own worlds. I would specifically like to be able to make both gas giants and terrestrial planets with distinguishing characteristics."

Me: "What type of parameters would you like so that you can generate the solar systems?"

Mr. Rieper: "Solar mass, radius, life levels, and water levels. It would be great to have control over these factors to model both our solar system and own worlds. I also would like for an implementation of moons, along with a moon number that reveals how many moons each planet has."

Me: "Would you like the solar systems to be to scale?"

Mr. Rieper: "It might not always be practical due to space constraints or computational limitations. However, if feasible, having an option for scaled representations could work. Maybe through matrix multiplication you can implement zooming and panning."

Me: "Would you like for the planets’ orbits to be represented in polar form?"

Mr. Rieper: "Yes, representing the planets' orbits in polar form would align with the historical accuracy of the Antikythera mechanism. It would provide a realistic representation of celestial motion that works with the analogue nature of the device. The students could also hit two birds with one stone by learning how polar notation works if they read the code."

Me: "Why do you choose polar over cartesian?"

Mr. Rieper: "Because polar notation was the way that the planets were initially modelled. Also, I think that since the solar system will be modelled from the centre, it makes a lot of sense for the mechanism to use polar as polar is based off of an angle and then a radius."

Me: "How will you incorporate my Antikythera mechanism simulation into the classroom?"

Mr. Rieper: "I see the simulation being used as an executable when my class is in a computer room. I also think that it might be worth creating an online version of the simulation because students might want to access the simulation at home, and our email firewall policy stops users from sending executables to each other."

Me: "Do you believe the simulation should include a guide to assist users in navigating and understanding its features?"

Mr. Rieper: "Including tutorials or guides or tooltips would be very helpful, especially for this simulation as it is on a very niche topic and has a lot of dials and outputs that need to be explained. Maybe you could also make it so that the program has datatypes next to some fields to help users know what to input."

Me: "How important do you think it is for the simulation to be accessible across different devices and platforms?"

Mr. Rieper: "Pretty important, because simulations should be handicapped by the hardware, not by the software, so if it is able to be run on most machines, then obviously it is better even if low end computers can’t generate more than, say, 5 planets. But maybe it should be declared if the input is too ambitious in the guide.”

Me: "In what ways do you think the simulation could be expanded or enhanced in the future?"

Mr. Rieper: “Maybe you could add it so that satellites and rockets can be launched from planets and then modelled. Or maybe add some sort of educational quiz system to the algorithm. Also you could make it so that students can load their favourite solar systems and planets from movies and games.”

Me: "Thank You for this interview."

## **Analysis of the Interview:**

In summary, the interview with Mr. Rieper explained why the Antikythera mechanism simulation is intriguing. Mr. Rieper finds it fascinating due to its ancient engineering and astronomical significance. He believes the simulation should include all its cycles and dials for a complete understanding. Mr. Rieper also likes the idea of creating custom solar systems in the simulation, allowing students to experiment and learn more. He suggests including guides to help users navigate the simulation and making it accessible on different devices. Looking forward, Mr. Rieper thinks the simulation could be improved with additional features, like interactive explanations and virtual tours.

The Interview revealed many things to me. One of the most daunting tasks that I have to do now is to convert my coordinates from cartesian to polar and vice versa. This is because the Antikythera mechanism was designed with polar coordinates, which were expanded on by Hipparchus. This is difficult because polar notation is a year 2 A-level Further Maths topic. I will need to use arctan and the modulus of the points in cartesian to calculate the r and θ values, and I also need to integrate to calculate area if I am to use equations which was taught quite recently by my maths teacher. This is A-level further mathematics that will be incorporated into my solution, adding a layer of complexity that will set my program apart from others.

Mr. Rieper also wanted to be able to control life levels. I think that I am going to generate planets with a user input called 'Life'. With this user input, which could be an integer from 0 to 10, I will generate the planet's characteristics. For example, a randomly generated planet with a life level of 10 will be nearly completely green, while one with a level of 0 will be visually completely uninhabitable. I will do the same for variables called 'Water'. This will add interactivity. Procedural generation of planets based on inputs is also quite advanced. I also want to have temperature play a part in the generation of planets, but I will discuss this more in the design phase.

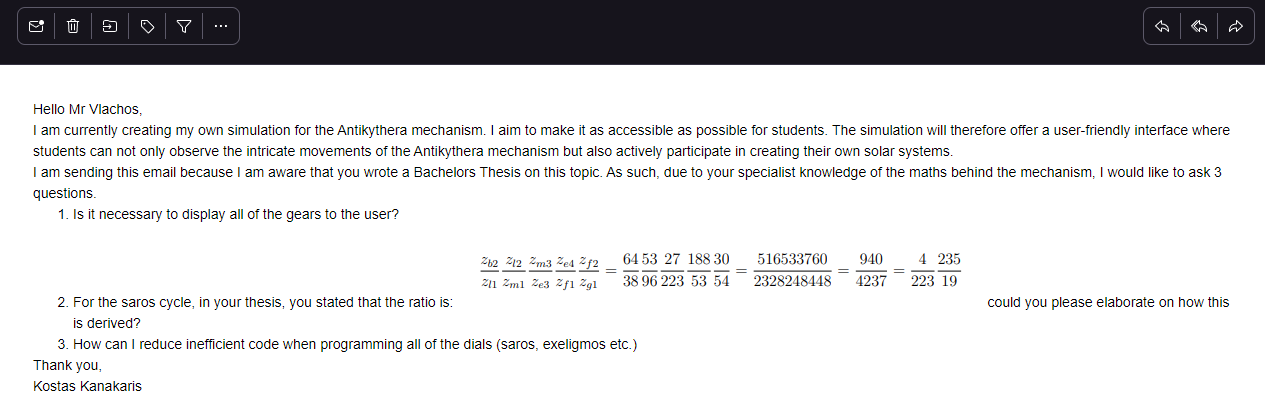
Both the Sothic and Corinthian dials can be implemented easily. I will probably have to design them in photoshop, however. XAML has limitations in terms of graphically representing text and warping it, so it will be nearly impossible to program it to skew and warp text.

I have to add tips and guides to my program too. I can use message boxes along with a help button to increase the accessibility of my program, especially since not many people are aware of how the Antikythera mechanism works. Also, I have to make sure that my application is portable as I am not sure what hardware will be used to run the executable. The program by extension should not be bloated and require unnecessary code (i.e inefficient iterations) to ensure that it is usable by everyone. I think that this may be a problem when it comes to randomly generating planets as there will be a lot of shapes involved when generating them.

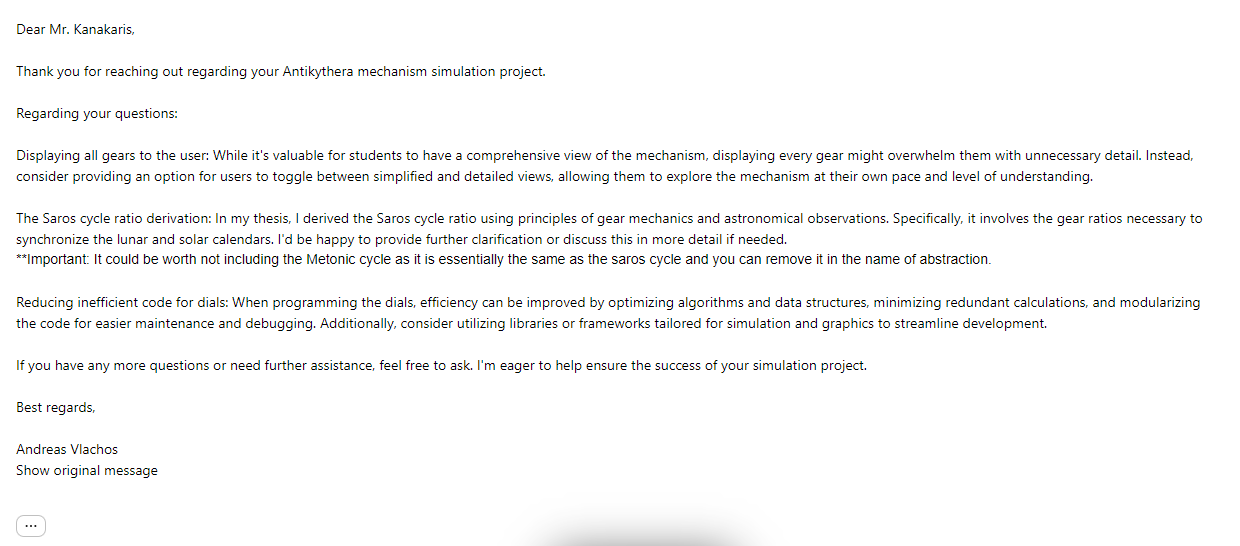
## **Contact with experts:**

Along with my interview of a teacher, as part of my NEA, I have decided to reach out to a specialist in the Antikythera mechanism and ask him some questions about how it functioned and what can be essentially omitted from the original algorithm without fundamentally altering the simulation. By abstracting parts of the problem, I can remove a lot of the algorithms that are unnecessary.

I sent an email to Andreas Vlachos, who wrote a thesis on the mathematical aspects of the Antikythera Mechanism.

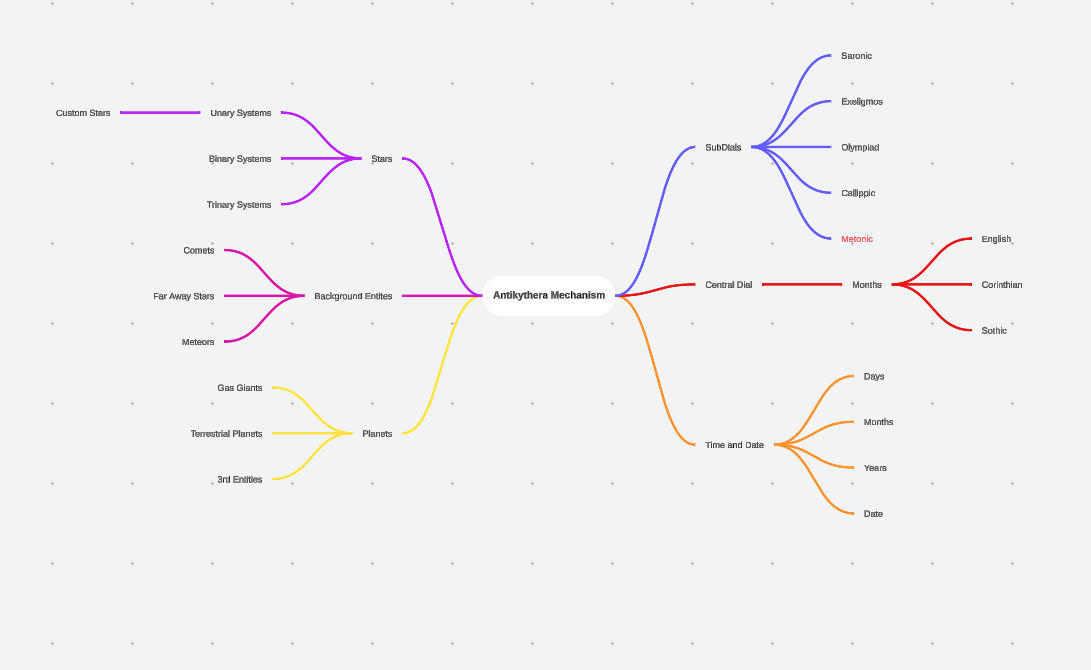


I received a response from him 3 days later, which was helpful in designing the dials as I will elaborate on during the design phase.

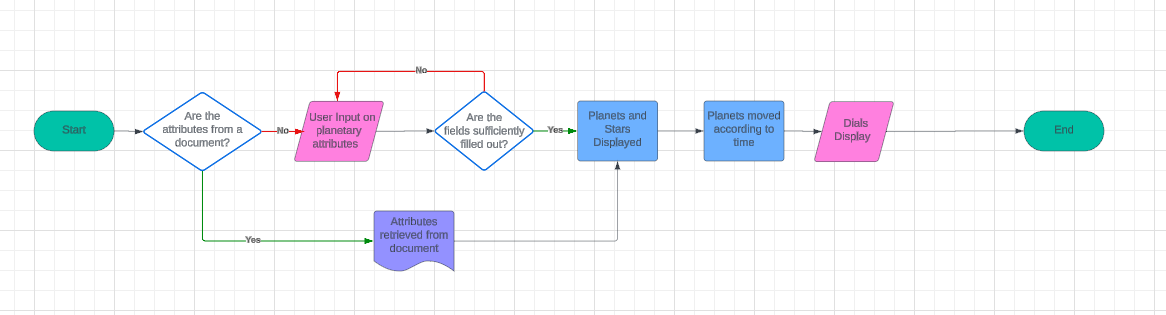


I understand now more about how the Saros cycle works by going back into the dissertation and rereading it more thoroughly. Furthermore, I have decided to remove the Metonic cycle from my program plan as it is made redundant by the inclusion of the more accurate Saronic cycle.

## **Mind Map of Preliminary Solution:**



## **Systems Flowchart:**



## **Analysis of Potential Languages:**

There are a few programming languages that I could use. I can program in quite a few languages well such as VB.NET, C# and JS. The following table analyses the advantages and disadvantages of using each language for this simulation:

|  |  |  |  |
| --- | --- | --- | --- |
| Language(s) | Overview | Advantages | Disadvantages |
| VB.NET | Offers simple syntax and visual design tools in Visual Studio. Supports advanced features like object-oriented programming  Has .NET Framework libraries, making it versatile for diverse applications. | Easy to understand syntax. I am also better at programming in VB.NET as I have used it more than C# | Not much support online. VB.NET online community is smaller e.g. on stack overflow. |
| C# | More complicated syntax than VB.NET. Also has visual design tools in Visual Studio. Supports advanced features like object-oriented programming.  Like VB.NET, it has .NET Framework libraries. | C# after being read through the CLR produces more efficient intermediate code than VB.NET.  A lot of support available online for C#. Supports lambda expressions which could be useful for my project. | Case sensitive - could cause certain errors. I am less confident in C# than in any of the other languages. |
| Web App languages (JS, HTML, CSS) | Enables the creation of dynamic and engaging web applications. Versatile and can be used to create responsive webpages. Back end can be developed with other languages and runtime environments like Node.js. | The only way to make a web app. Web apps are very good for accessibility as they do not require much hardware to run depending on how they are developed. | Not really required for my project: there will be little to no interaction between users apart from being able to share presets. Not required to be a web app. |

With all of these languages being considered in detail, I believe that C# should be the language that I use for this project. This is because I will also increase my knowledge of C# itself if I pick this language because I don't know the language as well as VB.NET. I do not need to make a web application since it is not going to require connectivity, so there is little point.

Now that I have picked C#, I have to pick a graphical framework. There are two main frameworks in Visual Studio for C#: WPF and Windows Forms. I have created another table detailing their differences:

## **Analysis of Potential Graphical Frameworks:**

|  |  |  |
| --- | --- | --- |
| Feature | WPF | Windows Forms |
| User Interface Definition | Uses XAML. | Uses drag and drop designer. |
| UI Features | Has animations, templates and styles. | Limited UI Features - fixed panels. |
| Data binding | Improved data binding between UI and data sources. | Less flexibility with data binding. |
| Animation support | Extensive support for animations. | A lot less support for animations |
| Community | Larger and more active community on Stack overflow | Much smaller and inactive community |
| Updates | Updated relatively frequently | No longer updated - obsolete |
| Resolution Independence | Not fixed resolution - based on computer I can design it to have different resolutions | Fixed resolution - decreases portability |

Windows forms is indeed easier to code due to the drag and drop feature, however I think that its decreased functionality will severely limit my ability to develop effectively on it. I also see the lack of resolution independence as a big problem because I aim to distribute my executable to many students, and so therefore it will be as portable as possible. With all of these factors taken into account, I have decided to use WPF.

## **Final Analysis Objectives:**

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Objective | Core/  Desirable | SMART |
| 1 | Displays our solar system when loaded from a preset and moves according to a slider which sets the speed at which the simulation moves. | Core | ✔ |
| 2 | Dials around the solar system move according to the aforementioned slider and visibly output the current position of the different dials using a needle. | Core | ✔ |
| 3 | Generate planet accurately based on inputs such as temperature, water and life levels. Randomly and procedurally generated – same inputs will never produce the same looking planet. | Core | ✔ |
| 4 | Generate stars based on different inputs. Generation of multiple stars may not be doable due to coinciding gravitational fields. | Core | ✔ |
| 5 | Have the main central dial in 3 different languages – English, Corinthian and Sothic based off maybe an array. Make it so that it also is output on the control panel on the right. | Core | ✔ |
| 6 | Import and Export custom solar systems. Possibly done using either custom algorithm or serialization like XML or Json or YAML. | Core | ✔ |
| 7 | A queue on the right that when a new entity (star or planet) is created, it adds the name of the new entity and displays the details of the planet. | Core | ✔ |
| 8 | Have background stars slowly moving in the background that improve the immersion. Make their speed be intrinsically connected to the speed slider. | Core | ✔ |
| 9 | Control panel on the right extracts all information from the dials and puts it all on the right which condenses the actual mathematics of the program. Includes specific dates and times that the algorithm is currently at. | Core | ✔ |
| 10 | Have fun settings and modes such as Lorax Mode which could change the appearance of a planet into a very colourful one, or for example a death star as a sun. | Desirable | ✔ |
| 11 | Ability to generate planet and star names based off of a set of predefined endings and vowel + consonant combinations, maybe with Regex. | Desirable | ✔ |
| 12 | Have all moons be represented. However, this might be too ambitious as some planets have hundreds of moons and so it could slow down the program. | Desirable | ✔ |
| 13 | Have custom orbits for custom solar systems. This may be difficult due to the many parameters in place and could be scrapped for complexity reasons. | Desirable | ✔ |
| 14 | Add custom controls to the canvas that allow for panning and zooming. | Desirable | ✔ |

# **Design:**

## **Overview:**

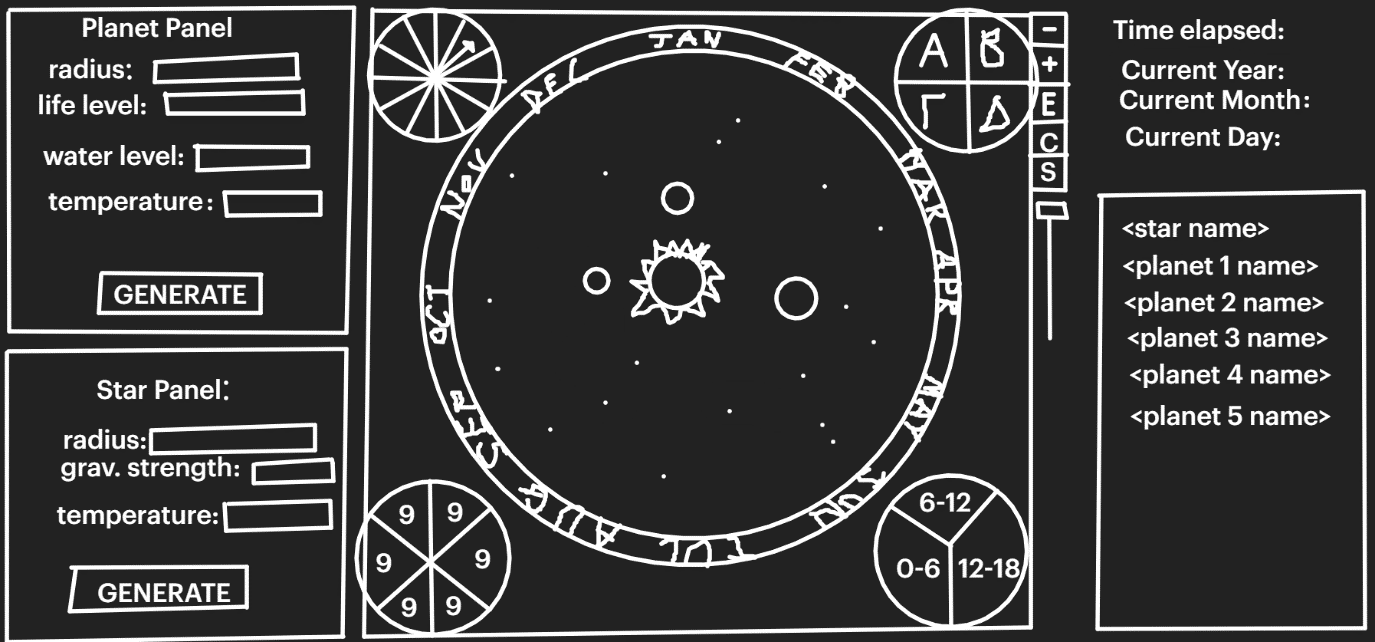
The design of this program, as I asserted earlier, can be broken down into two parts, the dial movements and the generation of solar systems.

I will be designing the UI using XAML which is a graphical framework in Visual Studio that can be used by C#. I will also be using UML to represent my algorithms' OOP structure.

## **General UI:**

The UI in my project will be very functional. I do not want for the UI to take up a large part of my project. As such, I will not install any aesthetic based frameworks such as Material Design in Visual Studio. I plan to use a variety of UI elements, as XAML calls them, such as buttons, text boxes and panels. The following table I made shows the UI elements I plan to use:

|  |  |  |
| --- | --- | --- |
| Element | Function | Usage in my project |
| Grid | Organizes content into rows and columns, allowing for easy layout and alignment. | A central grid on which everything will be placed. |
| TextBlock | Displays text content with various formatting options, such as font, colour, and alignment. | Many text blocks will be used for titles and outputs, as well as guiding the user with how to use the Antikythera Mechanism (i.e input types, etc.) |
| Button | Provides a clickable UI element that triggers an action or command when pressed. | Again, used very frequently for inputs and also for generation of planets and stars. |
| Image | Displays graphical content, typically images, within the user interface. | Used a lot for the dials: this is because I think that if I design images for the dials, it will be a lot easier that trying to warp text around a circle using XAML. |
| Canvas | Provides a surface for drawing graphics or arranging elements with absolute positioning. | Central canvas in the centre on which everything will occur. I may also place invisible canvases on top of planets to allow for increased detail. |
| Slider | Allows users to select a value from a range by sliding a thumb control along a track. | Primarily to be used for the speed slider. I am not sure what values to have as maximum and minimum - preliminary 1 and 0. |
| ListView | Displays a collection of items in a scrollable list, often used for presenting data. | I think that I will have a listview probably on the right that shows every single body in the project, apart from background entities. |
| Rectangles and Ellipses | Geometric shapes used for UI or to define boundaries within layouts. | Again, very common usage. I think rectangles will be primarily used as the pointers for the dials that rotate. The rotate transform function will be defined by a variety of constants that will be declared based on mathematics. |
| StackPanel | Arranges child elements in a single line, either horizontally or vertically. | Stack panels will be used for the Planet Generate and Star Generate Panels. I can also place children inside them which will help structure my UI. |
| RadioButton | Presents a choice among multiple mutually exclusive options, allowing users to select one. | I think that this will be primarily used for temperature levels. I will have 3 radio buttons for both Planet and Star, and call the radio buttons Low, Medium and High. |
| CheckBox | Represents a binary state, typically used for toggling an option on or off. | I will use this for activating some of the fun modes such as Lorax Mode if I decide to implement it. |

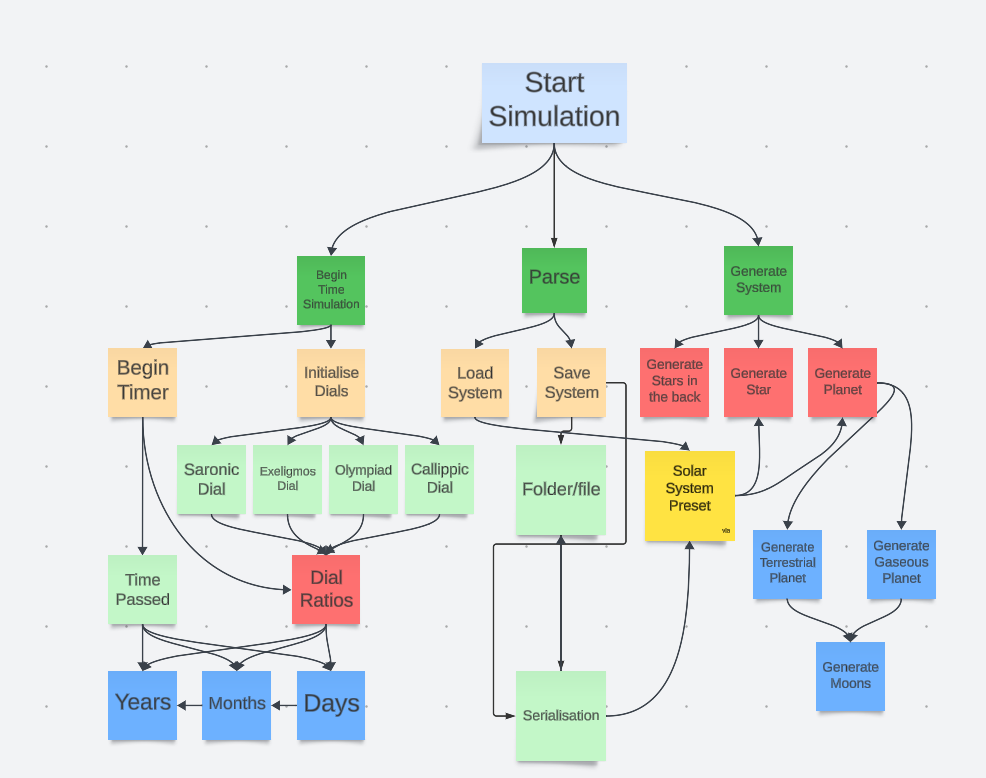


The UI is very clear. The central dial is the one with the months around it, and the canvas on which everything will be moving on is the black circle in the centre. The 4 dials in the corner are the Saronic, Exeligmos, Olympiad and Callippic cycles. I will explain these more in detail later when I investigate them one by one.

On the left are the generation panels. Based on certain inputs, my simulation will procedurally generate planets and stars. I decided to add two mini canvases as well on the panels, because I think that it is important that the user can see what the generated planet will look like before committing to adding it to the central canvas.

On the right are some controls. I have the toggle buttons for the central dial language and also have the buttons for zooming in and out in the canvas. I also have the speed slider.

## **Top-Down Diagram:**



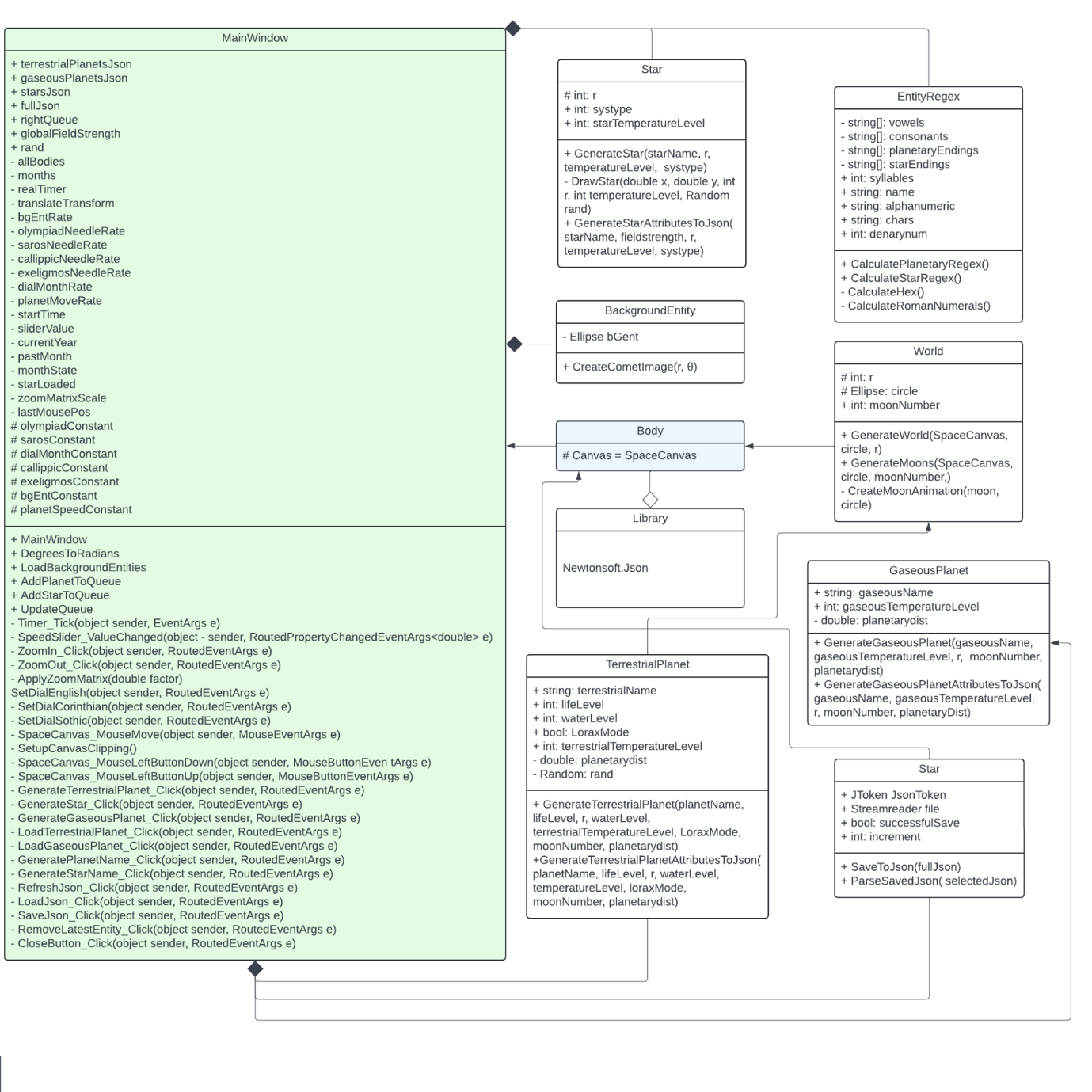
## **IPSO Chart:**

|  |  |  |
| --- | --- | --- |
| IPSO | Information | Source |
| Input | Planetary Generation:  - Radius  - Life Level  - Water Level  - Temperature  - Name | Interview |
| Input | Solar Generation:  - Radius  - Field Strength  - Temperature  - Name | Interview |
| Input | Date Generation:  - Day  - Month  - Year | Case Study 2 |
| Input | Potential preset files:  - Untitled-0.txt  - MattHowellSystem.txt  - customss.txt | Interview |
| Process | Compile dates into format that can easily be output by later stages. Maybe integer datatypes that can then be easily parsed. | Case Study 1 |
| Process | Apply mathematical algorithms to calculate ratios between the different needles and dials:  - Saronic Dial  - Exeligmos Dial  - Callippic Dial  - Olympiad Dial | Emailing Experts |
| Process | Internal parsing of preset files. | Interview |
| Process | Randomly/procedurally generate the planets using mathematics and random data types | Interview |
| Storage | Internal exporting of preset files which means saving to a specified folder on the user's machine or possibly on the web. | Interview |
| Storage | Saving of positions once the app is closed automatically and when opened again, temp files return the user to where they were in the simulation. Import from Process calculations | Case Study 1 |
| Storage | Storage of all entity information to carry out other processes like modifying them as the program is running. Import from Planet input and | Case Study 2 |
| Output | Output of position of dials, exactly like Case Study 1, but without any gears due to abstraction to make it easier for the user. Import from Dial Process. | Case Study 1 |
| Output | Procedural generation of planets, output graphical changes and highlight the random nature of how planets are formed. Import from generation process. | Interview |
| Output | User error fetching based on what their inputs are and what is lacking. Depends on planetary and solar inputs. | Case Study 1 |

## **Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Data Item | Data Type | Validation | Sample Data |
| Planet Name | String | <>”” | “Earth” |
| Planet Radius | Int | <>”” | 40 |
| Planet Temperature | Int (serves as a 0,1,2) | <>”” | 1 |
| Planet Moon Number | Int | <>”” | 3 |
| Life Level | Int | <>”” if terrestrial | 10 |
| Water Level | Int | <>”” if terrestrial | 5 |
| Star Name | String | <>”” | “Great Star” |
| Star Radius | Int | <>”” | 50 |
| Star Temperature | Int (serves as a 0,1,2) | <>”” | 0 |
| System Type | Int (serves as a 0,1,2) | <>”” | 2 |
| Entity Queue | Queue |  | Planet 2: MyPlanet - Life: 10 - Water/Lava: 15 - Radius: 40 - Temp: 1 - Moons: 2 |

## **Class Diagram:**

Unfortunately, the class diagram does not look very good because I used a lot of private fields and variables in the MainWindow.cs file which was easily my largest. Because of how XAML and WPF work, I can only have GUI components in that file, explaining the large size of the green class box. Therefore, all of the dial development was created on the MainWindow.cs file, which didn’t allow for

## **Design (Generation of solar system):**

### Regex:

Regex can also be used in my program. I am going to attempt to make a regex for generating suggestions for what to name a planet.

According to the international astronomical union, "the brightest member of a star system receives the letter "A". Distinct components not contained within "A" are labeled "B", "C", etc. Subcomponents are designated by one or more suffixes with the primary label, starting with lowercase letters for the second hierarchical level and then numbers for the third.[2] For example, if there is a triple star system in which two stars orbit each other closely with a third star in a more distant orbit, the two closely orbiting stars would be named Aa and Ab, whereas the distant star would be named B. For historical reasons, this standard is not always followed: for example Alpha Centauri A, B and C are not labelled Alpha Centauri Aa, Ab and B."

I created a quick outline of how I could do this with regex.

planetNamePattern = one of

{

- Starts with an uppercase letter,

- Followed by 2 to 5 lowercase letters,

- Followed by a hyphen ("-"),

- Followed by 2 to 3 digits

}

OR

{

- Starts with an uppercase letter,

- Followed by 4 to 6 lowercase letters,

- Followed by a space,

- Followed by 1 to 3 uppercase Roman numerals

}

OR

{

- Starts with an uppercase letter,

- Followed by 3 to 7 lowercase letters

}

foreach (UIElement element in SpaceCanvas.Children)

{

if (element is Ellipse bgent)

{

Canvas.SetLeft(bgent, Canvas.GetLeft(bgent) - rotationSpeed\*random.NextDouble()\*10);

if (Canvas.GetLeft(bgent) < -2500)

{

Canvas.SetLeft(bgent, 2500);

Canvas.SetTop(bgent, random.NextDouble() \* 2500 \* ((random.Next(1) \* 2) - 1 ));

}

}

}

### Validation:

I created some basic pseudocode for parsing GenerateTerrestrialPlanet. Since GenerateStar is very similar, I decided to only create one piece of pseudocode as I do not require both since it is quite clear what I am doing:

errorMessage = ""

if planetNameTextBox.Text is empty or null:

errorMessage += "\n- Planet name is required."

if lifeTextBox.Text cannot be parsed as an integer:

errorMessage += "\n- Life level must be a valid integer."

if radiusTextBox.Text cannot be parsed as an integer:

errorMessage += "\n- Radius must be a valid integer."

if waterTextBox.Text cannot be parsed as an integer:

errorMessage += "\n- Water level must be a valid integer."

if neither HighPlanetTemperature nor MediumPlanetTemperature is checked:

errorMessage += "\n- Temperature level must be selected."

if errorMessage is not empty:

Display MessageBox with message "Invalid input(s):" + errorMessage

return

I then wrote it in C#. I will go in depth on all of the combinations in the testing phase of the NEA:

string errorMessage = "";

if (string.IsNullOrEmpty(planetNameTextBox.Text))

{

errorMessage += "\n- Planet name is required.";

}

if (!int.TryParse(lifeTextBox.Text, out int lifeLevel))

{

errorMessage += "\n- Life level must be a valid integer.";

}

if (!int.TryParse(radiusTextBox.Text, out int r))

{

errorMessage += "\n- Radius must be a valid integer.";

}

if (!int.TryParse(waterTextBox.Text, out int waterLevel))

{

errorMessage += "\n- Water level must be a valid integer.";

}

if (HighPlanetTemperature.IsChecked == false && MediumPlanetTemperature.IsChecked == false)

{

errorMessage += "\n- Temperature level must be selected.";

}

if (!string.IsNullOrEmpty(errorMessage))

{

MessageBox.Show("Invalid input(s):" + errorMessage);

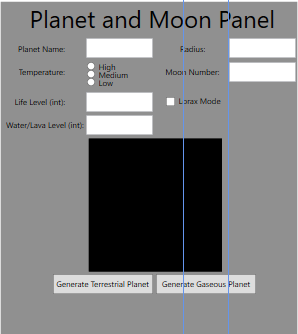
return;

}

As I said, I will show this in action during the testing phase.

### Fields (inputs) for planets:

My project requires inputs for generation of random planets:



There will be 6 + 1 fields for generating a terrestrial planet:

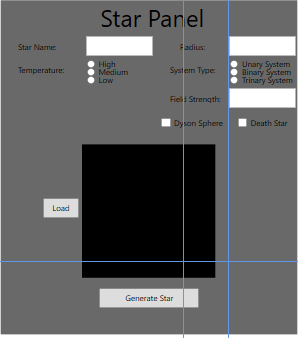
|  |  |  |
| --- | --- | --- |
| Field/Variable/Input | Field Description | Datatype |
| Planet Name | Unique identifier for the planet. | string |
| Temperature | Determines the temperature - radio button chosen to simplify development and reduce need of conversions - kelvin to celsius etc. | int (0,1,2 through a radio button) |
| Life Level | Determines the amount of grass and life when the planet is procedurally generated. | int |
| Water/Lava Level | Determines the amount of water generated when the planet is procedurally generated. | int |
| Radius | A double was chosen for the radius because it could be a decimal if the simulation is to be used precisely. | double |
| Moon Number | The moon number should ideally be kept low, and it indicates how many moons will be orbiting the planet. | int |
| Lorax Mode | When toggled, the planet is given unique and vibrant colouring similar to the planet in the Lorax. | bool |

For gas planets. there are 4+1 inputs:

|  |  |  |
| --- | --- | --- |
| Field/Variable/Input | Field Description | Datatype |
| Planet Name | Unique identifier for the planet. | string |
| Temperature | Determines the temperature - radio button chosen to simplify development and reduce need of conversions - kelvin to celsius etc. | int (0,1,2 through a radio button) |
| Radius | A double was chosen for the radius because it could be a decimal if the simulation is to be used precisely. | double |
| Moon Number | The moon number should ideally be kept low, and it indicates how many moons will be orbiting the planet. | int |
| Lorax Mode | When toggled, the planet is given unique and vibrant colouring similar to the planet in the Lorax. | bool |

As seen by my design, I have decided to incorporate the moons into my planet generation. I have done this because I believe that it is not necessary to have the user input the moon dimensions, and other features. At that point, the simulation becomes too finicky, and I don't believe that differing moon sizes will be particularly useful for procedurally generated planets. Because of this, I think that the algorithm will be much simpler and easier for the students to use. Furthermore, for large solar systems, having randomly generated moons will be very taxing on the computer, which directly goes against one of my goals which is the portability of this simulation.

### Fields (inputs) for stars:



There will be 5 + 2 fields for generating a star:

|  |  |  |
| --- | --- | --- |
| Field/Variable/Input | Field Description | Datatype |
| Star Name | Unique identifier for the Star. | string |
| Temperature | Determines the temperature - radio button chosen to simplify development and reduce need of conversions - kelvin to celsius etc. | int (0,1,2 through a radio button) |
| Field Strength | Determines the distance between all of the planets for their temperatures to be accurate. | int |
| Radius | A double was chosen for the radius because it could be a decimal if the simulation is to be used precisely. | double |
| Dyson Sphere | Allows for a Dyson sphere to be modelled. Dyson spheres are hypothetical megastructures that can extract the energy from a star. | bool |
| Death Star | When toggled, the star becomes a death star just for fun. | bool |

### Pseudocode Design (Generation of planet):

method GeneratePlanet

create a new Canvas

set spotCanvas width to circle width

set spotCanvas height to circle height

calculate grassNum as (circle width \* circle height \* lifeLevel^2) / 5000

loop i from 0 to grassNum:

create a new Ellipse grass

set grass width to a random value between 15 and 25

set grass height to grass width

if randomnumber is 1:

set grassSpot fill color to dark green

else if temperatureLevel is medium:

set grassfill color to lighter green

else:

set grassfill color to regular green

generate random grassSpotX and grassSpotY within circle boundaries

set left position of grass to grassSpotX

set top position of grass to grassSpotY

add grass to spotCanvas

calculate water as (circle width \* circle height \* waternumber)

loop i from 0 to water:

create a new Ellipse water

set water width to a random value between 1 and 25

set water height to waterwidth

if temperatureLevel is 0:

set water fill color to lighter blue

else if temperatureLevel is 1:

set water fill color to regular blue

else:

set water fill color to another shade of blue

generate random waterX and waterY within circle boundaries

set left position of water to waterX

set top position of water to waterY

add waterSpot to spotCanvas

create a new SolidColorBrush landBrush

if temperatureLevel is 0:

set stone color to light grey

else if temperatureLevel is 1:

set stone color to dark grey

else:

set stone color to black

create a new Ellipse stone covering the entire circle width

set stone fill color

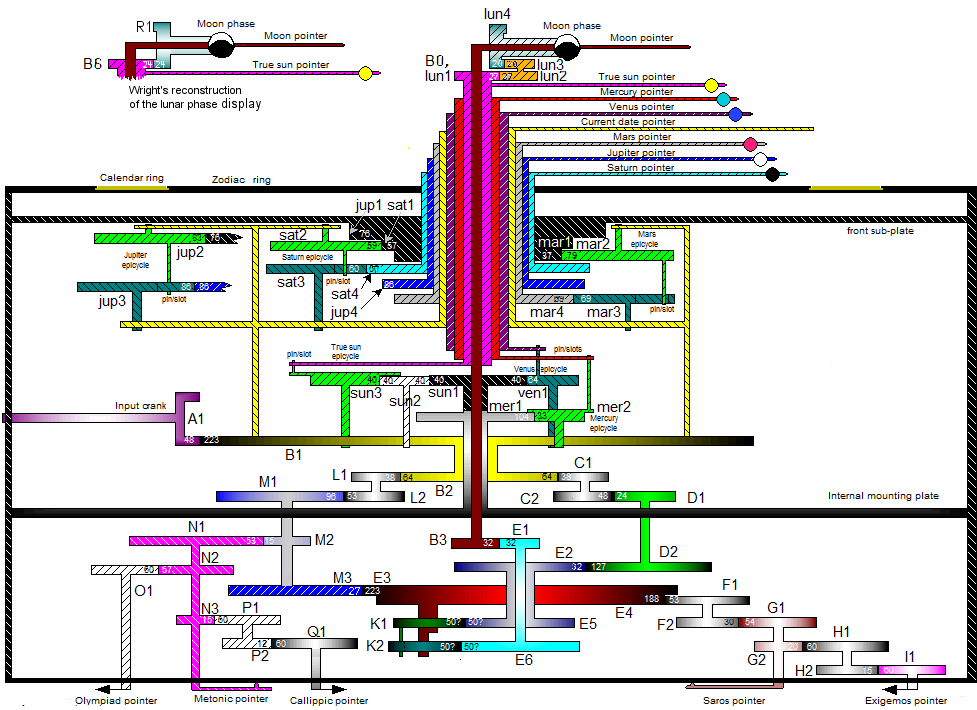
add stone and spotCanvas to grid

set circle fill to a VisualBrush with grid as content

## **Design (Dial Movement):**

### Mathematical Background and Research:

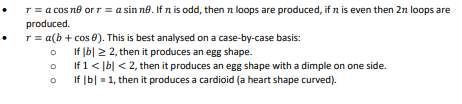
The Antikythera mechanism was found in 1900 by sponge divers off the coast of the Greek island of Antikythera, but it was only identified as the first ever analogue computer in 1902. It was scanned with an X-ray by the University of Cardiff in 2008 which determined a lot of the features that it had. Based on this X-ray and many other studies performed on it by archaeologists and computer scientists, the functionality of the mechanism has been understood as being used to calculate and predict astronomical phenomena, including the positions of celestial bodies such as the Sun, Moon, and planets, as well as eclipses. Due to the fact that Archimedes had roughly estimated what π was along with basic knowledge of calculus suggests that the ancient Greeks could have begun conceptualizing the elliptical orbits of celestial bodies as early as 200BC. Trigonometry also played a crucial role in the design and operation of the Antikythera Mechanism. The positions of celestial bodies were represented using angular measurements, and trigonometric functions such as sine and cosine were likely utilized to convert between angular measurements and linear displacements within the device's gear system. This implies that I might require the usage of Polar Coordinates (A-level further maths) to create my program.



This is a very complex diagram; it is easily broken down when using abstraction. Essentially, the entire solar system is connected to a central rod, which spins all of the different planets around it. the speeds at which they spin at are directly proportional to the teeth number of the gears. The Antikythera mechanism has two sides, however for the sake of simplicity, I will have it all on one side to prevent confusion because I want this to be intuitive for young students to understand how the first computer worked. This is an example of information hiding.

The Antikythera mechanism was geocentric, which means that Earth was at the centre of the diagram. However, to make it easier for the user to understand, and to be able to generate your own solar systems, I will have a more correct heliocentric system. I will allow the user to enter a variety of parameters for the stars, such as number of stars.

However, the code behind will be based on the historic model. The concept of a central Sun, with planets spinning around it attached to rods, is very close to how Polar coordinates work. In polar coordinates, there is a fixed "r", or radius, and the θ is increasing, which creates an orbit of a circle. I could then use advanced trigonometric algorithms to further fine tune the orbit, which could allow for orbits that are not perfectly circular. For example, to draw an "egg" orbit, I could use A level further maths techniques.



This is taken from the Physics and Maths tutor website, and details 3 types of orbits. Since I want the modulus b to be greater than or equal to 2, I could utilise this when I am programming.

### Advanced Mathematics:

My program will include 2 topics of maths that are A-level Further Maths standard, along with a lot of other maths. These Further Maths topics are, as I mentioned earlier:

- Polar notation (Used for rotating the planets around and generating them, along with modelling the needles of the dials)

- Matrices (Used for zooming in and out on my central canvas)

#### Polar Notation:

Trigonometric functions like sine, cosine, and tangent are essential for calculating the angular positions of celestial bodies relative to the observer on Earth. In the Antikythera Mechanism, gears with specific ratios encoded these trigonometric relationships to accurately model celestial movements. By utilizing polar notation, I can ensure that the angular positions can be efficiently represented.

Additionally, polar notation enables the representation of complex numbers, which can be used to model cyclical phenomena such as the oscillations of celestial bodies over time. This allows the simulation to predict astronomical events like lunar phases and planetary positions with remarkable accuracy, as the Antikythera Mechanism was capable of doing. However, complex numbers are very difficult to program accurately, and I don't believe that using them will provide any meaningful advantage. Generally including polar notation, however, will be a lot more accurate and closer to the historical way that the Antikythera mechanism worked.

To convert a number from cartesian to polar form, I can use the equations x^2+y^2 = r^2, x = rcosθ and y = rsinθ. r is the radius of the planet, and θ is the angle from the right x axis. I created a diagram on how polar notation could be used in my algorithm:

#### Matrices:

To implement zooming in my algorithm, the scale component of the transformation matrix could be modified based on user interactions, such as mouse scroll events and zoom in and out buttons. Increasing the scale enlarges the UI elements inside of the canvas, resulting in zooming in, while decreasing the scale reduces the canvas size, facilitating zooming out. By updating the scale component appropriately, the canvas can be zoomed in or out smoothly, enabling the students to explore different details of my Antikythera Mechanism at various levels of magnification. I think that when I write the code, I should keep the enlargement over time small, because otherwise the matrix transformations will cause the UI elements in the canvas to look choppy.

I created a pseudocode that would use matrices to zoom in and out a hypothetical canvas.

procedure zoomCanvas(canvas, zoomFactor): M

Matrix = createMatrix(zoomFactor)

for each UIElement in canvas.getUIElements():

currentPosition = UIElement.getPosition()

currentSize = UIElement.getSize()

newPosition = scalingMatrix \* currentPosition

UIElement.setPosition(newPosition) UIElement.setSize(newSize)

scalingMatrix = identityMatrix

scalingMatrix[0][0] = scaleFactor scalingMatrix[1][1] = scaleFactor return scalingMatrix

### Ratio design:

Internally, I can use ratios between each of the dials, which can be declared as constants. For between the saronic, exeligmos, callippic and olympiad dials, and of course the central dial, the ratios would be 1:4 for the olympiad, 1:76 for the callippic, 1:54 for the exeligmos and 1:18 for the saronic. These are all in relation to the central dial. I can then incorporate some polar coordinates to change the angles that all of the fixed distance (r) dial needles are:  
  
 dialMonthRotateTransform = dialMonth.RenderTransform;

dialMonthRotateTransform.Angle += dialMonthRate;

sarosCycleRotateTransform = SarosNeedle.RenderTransform;

sarosCycleRotateTransform.Angle += sarosNeedleRate;

olympiadRotateTransform = OlympiadNeedle.RenderTransform;

olympiadRotateTransform.Angle += olympiadNeedleRate;

callippicRotateTransform = CallippicNeedle.RenderTransform;

callippicRotateTransform.Angle += callippicNeedleRate;

exeligmosRotateTransform = ExeligmosNeedle.RenderTransform;

exeligmosRotateTransform.Angle += exeligmosNeedleRate;

### Slider:

The slider is also important to declare the needle rates. Because all of the needles are moving based on the slider value, and then the ratio/constant, I can simply multiply the constant by the slider value. I believe that XAML allows you to declare whatever slider value you want for maximum and minimum, which could prove very useful for fine tuning later.

olympiadNeedleRate =+ sliderValue \* olympiadConstant;

sarosNeedleRate =+ sliderValue \* sarosConstant;

callippicNeedleRate =+ sliderValue \* callippicConstant;

exeligmosNeedleRate =+ sliderValue \* exeligmosConstant;

dialMonthRate =+ sliderValue \* dialMonthConstant;

For planetary movement, I could probably do something similar as well. Maybe I can have a constant that is exactly defined by the radius and distance from the star.

## **Parsing:**

Initially, I was trying to see if I parse presets using my own algorithm that I had developed. For testing, to check whether or not my parsing worked, I used our solar system as a base model. I used abstraction and removed a lot of the other parameters to save time and just as a synoptic check to see if it worked or not.

However, I then learned about JSON as part of the Computer Science A-level (Client server model part of the specification) in February. While I have used XML serialization before, I have never used JSON, and therefore I decided to use JSON to parse so that Ι could increase my knowledge of other techniques. JSON also has many advantages compared to XML. It is easier for a human to read, easier to create and quicker for computers to parse.

I also wanted to use JSON because I want to learn how to use it when I am developing web applications as it is an integral part of communication between browsers and servers. Therefore, if I include it in my project, I will also gain valuable knowledge. Luckily, JSON is very easy for a human to read and understand, and as such, it did not take too long to understand.

I learned JSON syntax and created a test file, again for our solar system. I used Newtonsoft.Json library. Interestingly, Newtonsoft is the most popular NuGet package with over 4.4 billion downloads.

I also tried to use JSON to parse month dates. I created the following JSON:  
{

"months": {

"english": {

"1": "January",

"2": "February",

"3": "March",

"4": "April",

"5": "May",

"6": "June",

"7": "July",

"8": "August",

"9": "September",

"10": "October",

"11": "November",

"12": "December"

},

"corinthian": {

"1": "Poseidonios",

"2": "Aphroditios",

"3": "Artemisios",

"4": "Dionysios",

"5": "Hermaios",

"6": "Demetrios",

"7": "Kleonaios",

"8": "Hyperberetaios",

"9": "Eukleios",

"10": "Aristaios"

},

"sothic": {

"1": "Thoth",

"2": "Phaophi",

"3": "Athyr",

"4": "Choiak",

"5": "Tybi",

"6": "Mechir",

"7": "Phamenoth",

"8": "Pharmouthi",

"9": "Pachon",

"10": "Payni",

"11": "Epiphi",

"12": "Mesore"

}

}

}

However, this did not work due to the limitations of XAML. It was impossible to orient these around a circular path and make them rotate based on a slider. Also, a simple 2D array would suffice for this. While JSON was not useful this time, I am still going to use JSON to load and save planetary presets. Therefore, I decided to make my own images and place them over the canvas instead of trying to generate them using Json. The images are found in the Technical Solution Assets folder.  
  
  
I then will rotate the assets using the timer and the slider. I added a white to red gradient to symbolise the increase in speed towards the bottom.



This speed will also be intrinsically tied to the background movement of the stars in the background. BackgroundEntities, as I call them in my classes folder, will be generated each time I run the project. However, when I program this, it means that I need to get a balance of performance and panning, as the less stars generated, the more limited the panning of the canvas is. With 1000 stars, I can pan for a long time and still have BackgroundEntites present, while not requiring too many iterations to generate these so called 'UIElements'.

Because of how XAML works, it takes the top and left as 0,0. Therefore, I will have to use negative values to be able to generate some of the stars so that it looks like the solar system is in the center. Therefore, i use the formula: ((random.Next(1) \* 2) - 1 ) because it either generates a -1 or 1 and then multiply it by 2500 which is 5000/2.

I also will use .json files for loading solar systems and saving them as well. I created a .json record structure to help me visualise how .json could be used in my program. The json files will include all of the inputs of our solar system. Initially, when the program is first ran, the presets folder only has one file, which is sol.json, our own solar system, but the user can obviously save and load more.

### Record Structure:

|  |  |  |  |
| --- | --- | --- | --- |
| Record Element: | Json format | Json Example | Max Size (without strings) |
| Terrestrial Planets | "type": "TerrestrialPlanet",  "name": (string),  "lifelevel": (int),  "radius": (int),  "waterlevel": (int),  "temperature": (int),  "moonnumber": (int),  "lm": (bool),  "distancefromstar": (double) | "type": "TerrestrialPlanet",  "name": "Mercury",  "lifelevel": 0,  "radius": 10,  "waterlevel": 0,  "temperature": 2,  "moonnumber": 0,  "lm": false,  "distancefromstar": 80 | 32\*6+2+64 = 258B |
| Gaseous Planets | "type": "GaseousPlanet",  "name": (string),  "radius": (int),  "temperature": (int),  "moonnumber": (int),  "distancefromstar": (double) | "type": "GaseousPlanet",  "name": "Jupiter",  "radius": 60,  "moonnumber": 6,  "temperature": 1,  "distancefromstar": (double) | 32\*3+64 = 160B |
| Stars | {      "type": "Star”,      "name": (string),      "fieldstrength": (int),      "radius": (int),      "temperature": (int),      "systype": (int)    }, | {      "type": "Star",      "name": "Sun",      "fieldstrength": 10,      "radius": 60,      "temperature": 1,      "systype": 0    }, | 32\*4 = 128B |

## **Student Guide:**

The interface looks like this:  
A screenshot of a calendar

Description automatically generated

1. In the Planet and Moon Panel, students can add their own planets and numbers of moons. Generally, it is advisable to follow this table when deciding on what to choose of the inputs:

|  |  |
| --- | --- |
| Field Name: | Suggestions: |
| Name | Generate Suggested Planet Name button is found in the top left, you can use it in case you do not know what to name the planets. |
| Temperature | The temperature is one of three. Any is suitable, but consider the position you initialise the planets in; if you declare an ice planet first and then a lava planet, the lava planet will be further from the star yet will look visually warmer. |
| Radius | The radius should be less than the star’s radius, and ultimately is advised to be no bigger than 100. If generated bigger, it can cause issues with planetary spacing. Also, if you need larger/smaller planets, there are the zoom in/zoom out buttons. |
| Moon Number | The moon number should ideally be less than 5 because other wise there is a chance that moons overlap and are not displayed consistently. |
| Life Level | The Life Level should ideally be less than 30 as above that, the algorithm determining the procedural life generation will begin to increase exponentially as it is directly correlated to the radius as well and has a time complexity of O(2^n). |
| Water Level | Similar to the life level, however water level is initalised after life level, so if you are going for a very green planet, ensure that the water level is low, as if there is both a lot of water and a lot of grass, water takes precedence. |

1. Another panel is located directly below it which generates the stars:

|  |  |
| --- | --- |
| Field Name: | Suggestions: |
| Name | Generate Suggested Star Name button is found in the top left, you can use it in case you do not know what to name the stars. |
| Temperature | The temperature is one of three. Any is suitable, but if you are trying to accurately model a system, maybe consider the planet temperature and write the planetary strength accordingly because temperature often is directly correlated to gravitational field strength. |
| Radius | The radius should not really be any larger than 200, because, again, you can just zoom out and in if you want to be able to generate really large solar systems. |
| Field Strength | Essentially determines the distance of which the other stars will be spawned from each other and the central star. Generally, any larger than 50 can cause issues with displaying the planets properly. |
| Planet Type | A unary system is a system with one star. Binary and trinary systems therefore are simply two and three starred systems respectively. |

1. The dials are as follows:   
   - Callippic  
   - Saronic  
   - Exeligmos  
   - Olympiad  
   A calendar with numbers and a circle

   Description automatically generated
2. The central dial calculates the current month. The central dial’s language can be toggled based on the three buttons to the right of it: E, C and S. These stand for English, Corinthian and Sothic. The central dial moves anticlockwise because of the traditional nature of the Antikythera mechanism.
3. The central slider:   
   A close up of a colorful line

   Description automatically generated  
   This central slider is how you make the mechanism progress. The faster you push it forward, the faster the planetary movement and the Antikythera mechanism turning speed. The top of the slider represents the pause button as well, because it will be moving at 0x speed there. However, at the bottom which is red, the mechanism is moving at full speed.
4. On the right, there is the Json file loader. If you want to load presets, or other mechanisms that students have made and have shared with the class, you can download the Json files and then drag them into the Presets folder.   
   A screenshot of a computer

   Description automatically generated  
   As you can see here, the refresh button in the top left corner clears the stack and then reloads it, which allows you to see new additions without having to close down the program and open it again. The Save a system to .json file automatically saves the file as Untitled-<number>.json, which can then be renamed.

# **Technical Solution:**

Since my GUI is created using WPF XAML, I will include the XAML along with some snippets of the GUI. Further images of the GUI can be found in the Testing section. For checking what code associates to what part of the GUI, read the XAML, because it explains exactly which part of the GUI is what. I have included x:name for a lot of components to make it easier to follow.

## **MainWindow.xaml:**

|  |
| --- |
| <Window x:Class="MechanismSimulator.MainWindow"          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"          xmlns:d="http://schemas.microsoft.com/expression/blend/2008"          xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"          xmlns:local="clr-namespace:MechanismSimulator"          mc:Ignorable="d"          Title="MainWindow" Height="1080" Width="1920" WindowState="Maximized" WindowStyle="None">      <Grid>          <Grid.RowDefinitions>              <RowDefinition Height="481\*"/>              <RowDefinition Height="59\*"/>          </Grid.RowDefinitions>          <Grid.ColumnDefinitions>              <ColumnDefinition Width="286\*"/>              <ColumnDefinition Width="67\*"/>              <ColumnDefinition Width="139\*"/>              <ColumnDefinition Width="1428\*"/>          </Grid.ColumnDefinitions>            <TextBlock Panel.ZIndex="4" HorizontalAlignment="Left" Margin="44,69,0,0" TextWrapping="Wrap" Text="76" VerticalAlignment="Top" FontSize="18" Height="30" Width="Auto" Grid.Column="3"/>          <TextBlock HorizontalAlignment="Left" Margin="284,10,0,0" TextWrapping="Wrap" Text="Antikythera Mechanism Simulator" VerticalAlignment="Top" FontSize="24" Height="30" Width="368" Grid.Column="3"/>          <Button Content="Load a system from .json file" HorizontalAlignment="Left" Margin="1263,69,0,0" VerticalAlignment="Top" Click="LoadJson\_Click" Height="32" Grid.Column="3"/>          <Button Content="🗘" FontSize="20" HorizontalAlignment="Left"  Margin="1221,70,0,0" VerticalAlignment="Top" Click="RefreshJson\_Click" Height="32" Grid.Column="3" Width="32"/>          <Button Content="Save a system to .json file" HorizontalAlignment="Left" Margin="1221,220,0,0" VerticalAlignment="Top" Click="SaveJson\_Click" Height="32" Grid.Column="3"/>          <Button Content="Generate Suggested Planet Name" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top" Click="GeneratePlanetName\_Click" Height="32" Width="214"/>          <Button Content="Generate Suggested Star Name" HorizontalAlignment="Left" Margin="243,10,0,0" VerticalAlignment="Top" Click="GenerateStarName\_Click" Height="32" Grid.ColumnSpan="3" Width="214"/>            <Button Content="Close" HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,10,10,0" Click="CloseButton\_Click" Height="32" Width="61" Grid.Column="3"/>          <Button Panel.ZIndex="1" Content="Generate Terrestrial Planet" Click="GenerateTerrestrialPlanet\_Click" HorizontalAlignment="Left" Margin="74,478,0,0" VerticalAlignment="Top" Width="150" Height="30"/>          <Button Panel.ZIndex="3" Content="Clear Canvas" Click="ClearPlanetCanvas\_Click" HorizontalAlignment="Left" Margin="193,513,0,0" VerticalAlignment="Top" Width="80" Height="30"/>            <Button Panel.ZIndex="1" Content="Generate Gaseous Planet" Click="GenerateGaseousPlanet\_Click" HorizontalAlignment="Left" Margin="243,478,0,0" VerticalAlignment="Top" Width="150" Height="30" Grid.ColumnSpan="3"/>          <Button Panel.ZIndex="2" Content="Generate Star" Click="GenerateStar\_Click" HorizontalAlignment="Left" Margin="158,38,0,0" VerticalAlignment="Top" Width="150" Height="30" Grid.ColumnSpan="2" Grid.Row="1"/>            <Image x:Name="dialMonth" Panel.ZIndex="2" Source="assets/english.png" Height="1000" Width="1000" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="107,70,0,0" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.Column="2" Grid.ColumnSpan="2" Grid.RowSpan="2">              <Image.RenderTransform>                  <RotateTransform x:Name="dialRotateTransform" CenterX="500" CenterY="500" Angle="0"/>              </Image.RenderTransform>          </Image>          <Image x:Name="borderImage" Panel.ZIndex="1" Source="assets/border.png" Height="1000" Width="1000" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="107,70,0,0" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.Column="2" Grid.ColumnSpan="2" Grid.RowSpan="2"/>          <Image x:Name="olympiadDial" Panel.ZIndex="3" Source="assets/olympiaddial.png" HorizontalAlignment="Left" Height="160" Margin="803,75,0,0"  VerticalAlignment="Top" Width="160" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.Column="3"/>          <Image x:Name="sarosDial" Panel.ZIndex="3" Source="assets/sarosdial.png" HorizontalAlignment="Left" Height="160" Margin="803,905,0,0"  VerticalAlignment="Top" Width="160" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.Column="3" Grid.RowSpan="2"/>          <Image x:Name="callippicDial" Panel.ZIndex="3" Source="assets/callippicdial.png" HorizontalAlignment="Left" Height="160" Margin="113,75,0,0"  VerticalAlignment="Top" Width="160" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.Column="2" Grid.ColumnSpan="2"/>          <Image x:Name="exeligmosDial" Panel.ZIndex="3" Source="assets/exeligmosdial.png" HorizontalAlignment="Left" Height="160" Margin="113,905,0,0"  VerticalAlignment="Top" Width="160" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.Column="2" Grid.ColumnSpan="2" Grid.RowSpan="2"/>            <Label x:Name="realTimeLabel" Content="rtLabel" HorizontalAlignment="Left" Margin="1033,70,0,0" VerticalAlignment="Top" Height="30" Grid.Column="3"/>          <Label x:Name="yearLabel" Content="Years Passed: 0" HorizontalAlignment="Left" Margin="1033,100,0,0" VerticalAlignment="Top" Height="30" Grid.Column="3"/>          <Label x:Name="monthLabel" Content="mCount" HorizontalAlignment="Left" Margin="1033,130,0,0" VerticalAlignment="Top" Height="30" Grid.Column="3"/>          <Label x:Name="dayLabel" Content="dCount" HorizontalAlignment="Left" Margin="1033,160,0,0" VerticalAlignment="Top" Height="30" Grid.Column="3"/>            <Canvas              x:Name="SpaceCanvas"              Margin="107,70,460,10"              MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown"              MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp"              MouseMove="SpaceCanvas\_MouseMove" Width="1000" Height="1000" Grid.Column="2" Grid.ColumnSpan="2" Grid.RowSpan="2">              <Canvas.Background>                  <SolidColorBrush Color="Black"/>              </Canvas.Background>              <Canvas.LayoutTransform>                  <ScaleTransform ScaleX="1" ScaleY="1" />              </Canvas.LayoutTransform>          </Canvas>          <Canvas              Panel.ZIndex="3"              x:Name="PlanetCanvas"              Margin="132,275,21,487"              Width="200" Height="200" Grid.ColumnSpan="2">              <Canvas.Background>                  <SolidColorBrush Color="Black"/>              </Canvas.Background>          </Canvas>          <Canvas      Panel.ZIndex="3"      x:Name="StarCanvas"      Margin="132,785,21,95"      Width="200" Height="200" Grid.ColumnSpan="2" Grid.RowSpan="2">              <Canvas.Background>                  <SolidColorBrush Color="Black"/>              </Canvas.Background>          </Canvas>          <Image x:Name="DysonSphereSpace" Visibility="Hidden" Panel.ZIndex="3" Source="assets/dyson.png" HorizontalAlignment="Left" Height="160" Margin="388,490,0,0"  VerticalAlignment="Top" Width="160" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.Column="3"/>          <Image x:Name="DysonSphereCanvas" Visibility="Hidden" Panel.ZIndex="3" Source="assets/dyson.png" HorizontalAlignment="Left" Height="160" Margin="153,800,0,0"  VerticalAlignment="Top" Width="160" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.ColumnSpan="2"/>          <Image x:Name="DeathStarSpace" Visibility="Hidden" Panel.ZIndex="3" Source="assets/deathstar.png" HorizontalAlignment="Left" Height="160" Margin="388,490,0,0"  VerticalAlignment="Top" Width="160" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.Column="3"/>          <Image x:Name="DeathStarCanvas" Visibility="Hidden" Panel.ZIndex="3" Source="assets/deathstar.png" HorizontalAlignment="Left" Height="160" Margin="153,800,0,0"  VerticalAlignment="Top" Width="160" MouseLeftButtonDown="SpaceCanvas\_MouseLeftButtonDown" MouseLeftButtonUp="SpaceCanvas\_MouseLeftButtonUp" MouseMove="SpaceCanvas\_MouseMove" Grid.ColumnSpan="2"/>            <Button Content="+" HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,70,430,0" Width="30" Height="30" Click="ZoomIn\_Click" Grid.Column="3"/>          <Button Content="-" HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,100,430,0" Width="30" Height="30" Click="ZoomOut\_Click" Grid.Column="3"/>          <Button Content="E" HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,130,430,0" Width="30" Height="30" Click="SetDialEnglish" Grid.Column="3"/>          <Button Content="C" HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,160,430,0" Width="30" Height="30" Click="SetDialCorinthian" Grid.Column="3"/>          <Button Content="S" HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,190,430,0" Width="30" Height="30" Click="SetDialSothic" Grid.Column="3"/>          <Slider x:Name="speedSlider" HorizontalAlignment="Left" Margin="929,278,0,0" VerticalAlignment="Top" Width="120" RenderTransformOrigin="0.5,0.5" Minimum="0" Maximum="0.01"  ValueChanged="SpeedSlider\_ValueChanged" Grid.Column="3">              <Slider.RenderTransform>                  <TransformGroup>                      <ScaleTransform/>                      <SkewTransform/>                      <RotateTransform Angle="90"/>                      <TranslateTransform/>                  </TransformGroup>              </Slider.RenderTransform>          </Slider>              <Rectangle HorizontalAlignment="Left" Height="115" Margin="969,227,0,0"  VerticalAlignment="Top" Width="11" Grid.Column="3">              <Rectangle.Fill>                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                      <GradientStop Color="Red" Offset="1"/>                      <GradientStop Color="White" Offset="0"/>                  </LinearGradientBrush>              </Rectangle.Fill>          </Rectangle>          <Ellipse Panel.ZIndex="2"  HorizontalAlignment="Left" Height="160" Margin="803,75,0,0"  VerticalAlignment="Top" Width="160" Fill="#FFE49F3A" Grid.Column="3"/>          <Ellipse Panel.ZIndex="2"  HorizontalAlignment="Left" Height="160" Margin="803,905,0,0"  VerticalAlignment="Top" Width="160" Fill="#FFE49F3A" Grid.Column="3" Grid.RowSpan="2"/>          <Ellipse Panel.ZIndex="2"  HorizontalAlignment="Left" Height="160" Margin="113,75,0,0"  VerticalAlignment="Top" Width="160" Fill="#FFE49F3A" Grid.Column="2" Grid.ColumnSpan="2"/>          <Ellipse Panel.ZIndex="2"  HorizontalAlignment="Left" Height="160" Margin="113,905,0,0"  VerticalAlignment="Top" Width="160" Fill="#FFE49F3A" Grid.Column="2" Grid.ColumnSpan="2" Grid.RowSpan="2"/>          <ListBox x:Name="RightQueueBox" d:ItemsSource="{d:SampleData ItemCount=5}" Grid.Column="3" FontSize="11" Margin="1011,275,10,10" Grid.RowSpan="2" BorderThickness="2"/>          <ListBox x:Name="JsonListBox" d:ItemsSource="{d:SampleData ItemCount=5}" Grid.Column="3" Margin="1221,107,10,748" BorderThickness="2"/>            <Rectangle x:Name="OlympiadNeedle" Width="4" Fill="Black" Margin="881,155,543,727" Panel.ZIndex="3" Grid.Column="3">              <Rectangle.RenderTransform>                  <RotateTransform Angle="0" CenterX="2" CenterY="0"/>              </Rectangle.RenderTransform>          </Rectangle>          <Rectangle x:Name="SarosNeedle" Width="4" Fill="Black" Margin="881,18,543,15" Panel.ZIndex="3" Grid.Column="3" Grid.Row="1">              <Rectangle.RenderTransform>                  <RotateTransform Angle="0" CenterX="2" CenterY="0"/>              </Rectangle.RenderTransform>          </Rectangle>          <Rectangle x:Name="CallippicNeedle" Width="4" Fill="Black" Margin="52,155,1372,727" Panel.ZIndex="3" Grid.Column="3">              <Rectangle.RenderTransform>                  <RotateTransform Angle="0" CenterX="2" CenterY="0"/>              </Rectangle.RenderTransform>          </Rectangle>          <Rectangle x:Name="ExeligmosNeedle" Width="4" Fill="Black" Margin="52,18,1372,15" Panel.ZIndex="3" Grid.Column="3" Grid.Row="1">              <Rectangle.RenderTransform>                  <RotateTransform Angle="0" CenterX="2" CenterY="0"/>              </Rectangle.RenderTransform>          </Rectangle>          <Rectangle Panel.ZIndex="4" HorizontalAlignment="Left" Height="78" Margin="0,259,0,0" Fill="Red" Stroke="Red" VerticalAlignment="Top" Width="3" RenderTransformOrigin="0.5,0.5" Grid.Column="3">              <Rectangle.RenderTransform>                  <TransformGroup>                      <ScaleTransform/>                      <SkewTransform/>                      <RotateTransform Angle="-58.707"/>                      <TranslateTransform/>                  </TransformGroup>              </Rectangle.RenderTransform>          </Rectangle>          <StackPanel Grid.ColumnSpan="3"  Margin="10,70,37,392" Background="#FF909090">              <TextBlock TextWrapping="Wrap" Text=" Planet and Moon Panel" Width="Auto" HorizontalAlignment="Center" FontSize="36"/>            </StackPanel>          <RadioButton x:Name="HighPlanetTemperature" Content="High" GroupName="PlanetTemperature" HorizontalAlignment="Left" Margin="138,160,0,0" VerticalAlignment="Top"/>          <RadioButton x:Name="MediumPlanetTemperature" Content="Medium" GroupName="PlanetTemperature" HorizontalAlignment="Left" Margin="138,172,0,0" VerticalAlignment="Top"/>          <RadioButton x:Name="LowPlanetTemperature" Panel.ZIndex="2" Content="Low" GroupName="PlanetTemperature" HorizontalAlignment="Left" Margin="138,184,0,0" VerticalAlignment="Top"/>          <RadioButton x:Name="HighStarTemperature" Panel.ZIndex="2" Content="High" GroupName="StarTemperature" HorizontalAlignment="Left" Margin="138,657,0,0" VerticalAlignment="Top"/>          <RadioButton x:Name="MediumStarTemperature" Panel.ZIndex="2" Content="Medium" GroupName="StarTemperature" HorizontalAlignment="Left" Margin="138,669,0,0" VerticalAlignment="Top"/>          <RadioButton x:Name="LowStarTemperature" Panel.ZIndex="2" Content="Low" GroupName="StarTemperature" HorizontalAlignment="Left" Margin="138,681,0,0" VerticalAlignment="Top"/>          <RadioButton x:Name="UnarySystem" Panel.ZIndex="2" Content="Unary System" GroupName="SystemType" HorizontalAlignment="Left" Margin="0,657,0,0" VerticalAlignment="Top" Grid.Column="2"/>          <RadioButton x:Name="BinarySystem" Panel.ZIndex="2" Content="Binary System" GroupName="SystemType" HorizontalAlignment="Left" Margin="0,669,0,0" VerticalAlignment="Top" Grid.Column="2"/>          <RadioButton x:Name="TrinarySystem" Panel.ZIndex="2" Content="Trinary System" GroupName="SystemType" HorizontalAlignment="Left" Margin="0,681,0,0" VerticalAlignment="Top" Grid.Column="2"/>          <TextBox x:Name="planetNameTextBox" HorizontalAlignment="Left" Margin="138,125,0,0" VerticalAlignment="Top" Width="100" Height="30"/>          <TextBox x:Name="radiusTextBox" HorizontalAlignment="Left" Margin="0,125,0,0" VerticalAlignment="Top" Width="100" Height="30" Grid.Column="2"/>          <TextBox x:Name="moonNumberTextBox" HorizontalAlignment="Left" Margin="0,160,0,0" VerticalAlignment="Top" Width="100" Height="30" Grid.Column="2"/>          <TextBox x:Name="lifeTextBox" HorizontalAlignment="Left" Margin="138,205,0,0" VerticalAlignment="Top" Width="100" Height="30"/>          <TextBox x:Name="waterTextBox" Width="100" Height="30" Margin="138,240,48,692"/>          <TextBlock TextWrapping="Wrap" Text="Life Level (int):" Width="Auto" Height="15" Margin="31,212,176,735"/>          <TextBlock TextWrapping="Wrap" Text="Planet Name:" HorizontalAlignment="Left" Margin="36,132,0,0" VerticalAlignment="Top" Width="Auto" Height="15"/>          <TextBlock TextWrapping="Wrap" Text="Radius:" HorizontalAlignment="Left" Margin="279,132,0,0" VerticalAlignment="Top" Width="Auto" Height="15" Grid.ColumnSpan="2"/>          <TextBlock TextWrapping="Wrap" Text="Water/Lava Level (int): " HorizontalAlignment="Left" Margin="18,247,0,0" VerticalAlignment="Top" Width="Auto" Height="15"/>          <TextBlock TextWrapping="Wrap" Text="Temperature:" HorizontalAlignment="Left" Margin="37,167,0,0" VerticalAlignment="Top" Width="Auto" Height="15"/>          <TextBlock TextWrapping="Wrap" Text="Moon Number:" HorizontalAlignment="Left" Margin="257,167,0,0" VerticalAlignment="Top" Width="Auto" Height="15" Grid.ColumnSpan="2"/>          <CheckBox x:Name="LoraxModeBox" Content="Lorax Mode" HorizontalAlignment="Left" Margin="257,212,0,0" VerticalAlignment="Top" Width="Auto" Height="16" IsChecked="False" Grid.ColumnSpan="2"/>          <CheckBox x:Name="DeathStarBox" Panel.ZIndex="2" Content="Death Star" HorizontalAlignment="Left" Margin="12,745,0,0" VerticalAlignment="Top" Width="Auto" Height="16" IsChecked="False" Grid.Column="2"/>          <CheckBox x:Name="DysonSphereBox" Panel.ZIndex="2" Content="Dyson Sphere" HorizontalAlignment="Left" Margin="250,745,0,0" VerticalAlignment="Top" Width="Auto" Height="16" IsChecked="False" Grid.ColumnSpan="2"/>            <TextBox x:Name="starNameTextBox" Panel.ZIndex="2" HorizontalAlignment="Left" Margin="138,622,0,0" VerticalAlignment="Top" Width="100" Height="30"/>          <TextBox x:Name="starRadiusTextBox" Panel.ZIndex="2" HorizontalAlignment="Left" Margin="0,622,0,0" VerticalAlignment="Top" Width="100" Height="30" Grid.Column="2"/>          <TextBox x:Name="fieldStrengthTextBox" Panel.ZIndex="2" HorizontalAlignment="Left" Margin="0,701,0,0" VerticalAlignment="Top" Width="100" Height="30" Grid.Column="2"/>          <Button Panel.ZIndex="4" Content="Load" Click="LoadStar\_Click" HorizontalAlignment="Left" Margin="74,865,0,0" VerticalAlignment="Top" Width="53" Height="30"/>          <Button Panel.ZIndex="4" Content="Load T.P" Click="LoadTerrestrialPlanet\_Click" HorizontalAlignment="Left" Margin="74,360,0,0" VerticalAlignment="Top" Width="53" Height="30"/>          <Button Panel.ZIndex="4" Content="Load G.P" Click="LoadGaseousPlanet\_Click" HorizontalAlignment="Left" Margin="74,390,0,0" VerticalAlignment="Top" Width="53" Height="30"/>            <TextBlock TextWrapping="Wrap" Panel.ZIndex="2" Text="Star Name:" HorizontalAlignment="Left" Margin="36,630,0,0" VerticalAlignment="Top" Width="Auto" Height="15"/>          <TextBlock TextWrapping="Wrap" Panel.ZIndex="2" Text="Temperature:" HorizontalAlignment="Left" Margin="36,665,0,0" VerticalAlignment="Top" Width="Auto" Height="15"/>          <TextBlock TextWrapping="Wrap" Panel.ZIndex="2" Text="Radius:" HorizontalAlignment="Left" Margin="279,630,0,0" VerticalAlignment="Top" Width="Auto" Height="15" Grid.ColumnSpan="2"/>          <TextBlock TextWrapping="Wrap" Panel.ZIndex="2" Text="System Type:" HorizontalAlignment="Left" Margin="264,665,0,0" VerticalAlignment="Top" Width="Auto" Height="15" Grid.ColumnSpan="2"/>          <TextBlock TextWrapping="Wrap" Panel.ZIndex="2" Text="Field Strength:" HorizontalAlignment="Left" Margin="264,708,0,0" VerticalAlignment="Top" Width="Auto" Height="15" Grid.ColumnSpan="2"/>          <Button Panel.ZIndex="3" Content="Clear Canvas" Click="ClearStarCanvas\_Click" HorizontalAlignment="Left" Margin="193,73,0,0" VerticalAlignment="Top" Width="80" Height="30" Grid.Row="1"/>            <StackPanel Panel.ZIndex="1" Grid.ColumnSpan="3"  Margin="10,570,37,10" Background="DimGray" Grid.RowSpan="2">              <TextBlock  TextWrapping="Wrap" Text=" Star Panel" Width="Auto" HorizontalAlignment="Center" FontSize="36"/>          </StackPanel>          <TextBlock x:Name="SuggestedPlanetTextBox" HorizontalAlignment="Left" Margin="10,47,0,0" TextWrapping="Wrap" Text="Suggested Planet Name:" VerticalAlignment="Top" Width="228"/>          <TextBlock x:Name="SuggestedStarTextBox" HorizontalAlignment="Left" Margin="243,47,0,0" TextWrapping="Wrap" Text="Suggested Star Name:" VerticalAlignment="Top" Width="183" Grid.ColumnSpan="3"/>          <Button Grid.Column="3" Content="Remove Latest Planet" HorizontalAlignment="Left" Margin="1056,226,0,0" VerticalAlignment="Top"/>            </Grid>  </Window> |

### **MainWindow.xaml.cs:**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows;  using System.IO;  using System.Windows.Threading;  using System.Windows.Controls;  using System.Windows.Data;  using System.Windows.Documents;  using System.Windows.Input;  using System.Windows.Media;  using System.Windows.Media.Imaging;  using System.Windows.Navigation;  using System.Windows.Shapes;  using Newtonsoft.Json;  using MechanismSimulator.Classes;  using System.Collections;  using System.Globalization;  using System.Windows.Media.Media3D;  using System.Reflection.Emit;  using System.Threading;  using System.Text.RegularExpressions;  using System.Diagnostics.Eventing.Reader;  using Newtonsoft.Json.Linq;    namespace MechanismSimulator  {                     public partial class MainWindow : Window      {          List<JObject> terrestrialPlanetsJson = new List<JObject>();          List<JObject> gaseousPlanetsJson = new List<JObject>();          List<JObject> starsJson = new List<JObject>();          JArray fullJson = new JArray();          public string filen;          private List<Body> allBodies = new List<Body>();          private string[,] months = { { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December" },{ "Poseidonios", "Aphroditios", "Artemisios", "Dionysios", "Hermaios", "Demetrios", "Kleonaios", "Hyperetaios", "Eukleios", "Aristaios", "Diogenes", "Poseidonios" }, { "Thoth", "Phaophi", "Athyr", "Choiak", "Tybi", "Mechir", "Phamenoth", "Pharmouthi", "Pachon", "Payni", "Epiphi", "Mesore" } };          public Queue<string> rightQueue = new Queue<string>();          public double DegreesToRadians(double degrees)          {              return degrees \* Math.PI / 180.0;          }            private Point lastMousePos;            private Random rand = new Random();          private DispatcherTimer realTimer = new DispatcherTimer();          private TranslateTransform translateTransform = new TranslateTransform();          private double bgEntRate, olympiadNeedleRate, sarosNeedleRate, callippicNeedleRate, exeligmosNeedleRate, dialMonthRate, planetMoveRate;          private DateTime startTime = DateTime.Now;            protected const double olympiadConstant = 223/48;          protected const double sarosConstant = 1;          protected const double dialMonthConstant = -223/12;          protected const double callippicConstant = 1/4.75;          protected const double exeligmosConstant = 2/3.0;          protected const double bgEntConstant = 90/12;          protected const double planetSpeedConstant = 1;          private double sliderValue;          private int currentYear;          private int pastMonth;          private int monthState;          public bool starLoaded;          private double zoomMatrixScale = 1.0;          public int globalFieldStrength;          public MainWindow()          {                InitializeComponent();              LoadBackgroundEntities();                           SpaceCanvas.RenderTransform = new TransformGroup()              {                  Children = new TransformCollection()                  {                      translateTransform                  }              };                SpaceCanvas.MouseLeftButtonDown += SpaceCanvas\_MouseLeftButtonDown;              SpaceCanvas.MouseMove += SpaceCanvas\_MouseMove;              SpaceCanvas.MouseLeftButtonUp += SpaceCanvas\_MouseLeftButtonUp;              realTimer.Interval = TimeSpan.FromMilliseconds(1);              realTimer.Tick += Timer\_Tick;              realTimer.Start();            }          private void Timer\_Tick(object sender, EventArgs e)          {              TimeSpan elapsedTime = DateTime.Now - startTime;                           foreach (UIElement element in SpaceCanvas.Children)              {                  if (element is Ellipse bgent && bgent.Tag as string == "bgent")                  {                      Canvas.SetLeft(bgent, Canvas.GetLeft(bgent) - bgEntRate\*rand.NextDouble()\*10);                      if (Canvas.GetLeft(bgent) < -2500)                      {                          Canvas.SetLeft(bgent, 2500);                          Canvas.SetTop(bgent, rand.NextDouble() \* 2500 \* ((rand.Next(1) \* 2) - 1 ));                      }                  }                  else if (element is Ellipse planet && (planet.Tag as string == "terrestrialplanet" || planet.Tag as string == "gaseousplanet"))                  {                      Random internalrand = new Random();                                           double centerX = SpaceCanvas.ActualWidth / 2;                      double centerY = SpaceCanvas.ActualHeight / 2;                      double radius = Math.Min(centerX, centerY) - planet.Width / 2;                        double anglePerSecond = 0.1\*internalrand.NextDouble()+0.05;                      double currentAngle = (double)planet.GetValue(Canvas.LeftProperty) \* Math.PI / 180;                                             currentAngle += planetMoveRate \* anglePerSecond \* elapsedTime.TotalSeconds;                                             double newX = (centerX + planetMoveRate \* radius \* Math.Cos(currentAngle));                      double newY = (centerY + planetMoveRate \*radius \* Math.Sin(currentAngle));                                             Canvas.SetLeft(planet, newX);                      Canvas.SetTop(planet, newY);                  }              }                RotateTransform dialMonthRotateTransform = dialMonth.RenderTransform as RotateTransform;              dialMonthRotateTransform.Angle += dialMonthRate;              RotateTransform sarosCycleRotateTransform = SarosNeedle.RenderTransform as RotateTransform;              sarosCycleRotateTransform.Angle += sarosNeedleRate;              RotateTransform olympiadRotateTransform = OlympiadNeedle.RenderTransform as RotateTransform;              olympiadRotateTransform.Angle += olympiadNeedleRate;              RotateTransform callippicRotateTransform = CallippicNeedle.RenderTransform as RotateTransform;              callippicRotateTransform.Angle += callippicNeedleRate;              RotateTransform exeligmosRotateTransform = ExeligmosNeedle.RenderTransform as RotateTransform;              exeligmosRotateTransform.Angle += exeligmosNeedleRate;                realTimeLabel.Content = $"Real Time Elapsed: {elapsedTime.ToString(@"hh\:mm\:ss")}";              int monthIndex = (((int)Math.Floor((dialMonthRotateTransform.Angle \* 0.034) / (dialMonthRate - 1)))%12);              int dayIndex = (((int)Math.Floor((dialMonthRotateTransform.Angle) / (dialMonthRate - 1)))%365);              if ((monthIndex == 0) && (pastMonth != monthIndex))              {                  currentYear++;                  yearLabel.Content = $"Years passed: {currentYear}";              }              monthLabel.Content = $"Current Month: {months[monthState,monthIndex]}";              dayLabel.Content = $"Current Day: {dayIndex}";              pastMonth = monthIndex;          }          private void SpeedSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)          {             sliderValue = ((Slider)sender).Value;             olympiadNeedleRate =+ sliderValue \* olympiadConstant;             sarosNeedleRate =+ sliderValue \* sarosConstant;             callippicNeedleRate =+ sliderValue \* callippicConstant;             exeligmosNeedleRate =+ sliderValue \* exeligmosConstant;             bgEntRate =+ sliderValue \* bgEntConstant;             dialMonthRate =+ sliderValue \* dialMonthConstant;             planetMoveRate =+ sliderValue \* planetSpeedConstant;          }          private void ZoomIn\_Click(object sender, RoutedEventArgs e)          {              ApplyZoomMatrix(1.1);            }          private void ZoomOut\_Click(object sender, RoutedEventArgs e)          {              ApplyZoomMatrix(0.9);          }          private void ApplyZoomMatrix(double factor)          {                           zoomMatrixScale \*= factor;              Matrix scaleMatrix = new Matrix();              scaleMatrix.Scale(zoomMatrixScale, zoomMatrixScale);                                 foreach (UIElement element in SpaceCanvas.Children)              {                  element.RenderTransformOrigin = new Point(0.5,0.5);                  element.RenderTransform = new MatrixTransform(scaleMatrix);              }          }          private void SetDialEnglish(object sender, RoutedEventArgs e)          {              dialMonth.Source = new BitmapImage(new Uri("pack://application:,,,/Assets/english.png")); ;              monthState = 0;          }          private void SetDialCorinthian(object sender, RoutedEventArgs e)          {              dialMonth.Source = new BitmapImage(new Uri("pack://application:,,,/Assets/corinthian.png")); ;              monthState = 1;          }          private void SetDialSothic(object sender, RoutedEventArgs e)          {              dialMonth.Source = new BitmapImage(new Uri("pack://application:,,,/Assets/sothic.png")); ;              monthState = 2;          }                private void SpaceCanvas\_MouseMove(object sender, MouseEventArgs e)            {                if (SpaceCanvas.IsMouseCaptured)              {                  Point currentMousePos = e.GetPosition(SpaceCanvas);                  double dx = currentMousePos.X - lastMousePos.X;                  double dy = currentMousePos.Y - lastMousePos.Y;                      foreach (UIElement child in SpaceCanvas.Children)                  {                                           TranslateTransform translate = child.RenderTransform as TranslateTransform;                      if (translate == null)                      {                          translate = new TranslateTransform();                          child.RenderTransform = translate;                      }                        translate.X += dx;                      translate.Y += dy;                  }                    lastMousePos = currentMousePos;                  SetupCanvasClipping();                }            }            private void SetupCanvasClipping()          {              RectangleGeometry clippingRegion = new RectangleGeometry(new Rect(0, 0, SpaceCanvas.ActualWidth, SpaceCanvas.ActualHeight));              SpaceCanvas.Clip = clippingRegion;          }          private void SpaceCanvas\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)          {              lastMousePos = e.GetPosition(SpaceCanvas);              SpaceCanvas.CaptureMouse();          }          private void SpaceCanvas\_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)          {              SpaceCanvas.ReleaseMouseCapture();          }           public void LoadBackgroundEntities()          {                  BackgroundEntity bgent = new BackgroundEntity(SpaceCanvas, 2, 2);                  allBodies.Add(bgent);          }          private void GenerateTerrestrialPlanet\_Click(object sender, RoutedEventArgs e)          {              string errorMessage = "";                if (string.IsNullOrEmpty(planetNameTextBox.Text))              {                  errorMessage += "\n- Planet name is required.";              }                if (!int.TryParse(lifeTextBox.Text, out int lifeLevel))              {                  errorMessage += "\n- Life level must be a valid integer.";              }              if (!int.TryParse(moonNumberTextBox.Text, out int moonNumber))              {                  errorMessage += "\n- Moon number must be a valid integer.";              }              if (!int.TryParse(radiusTextBox.Text, out int r))              {                  errorMessage += "\n- Radius must be a valid integer.";              }                if (!int.TryParse(waterTextBox.Text, out int waterLevel))              {                  errorMessage += "\n- Water level must be a valid integer.";              }                if (HighPlanetTemperature.IsChecked == false && MediumPlanetTemperature.IsChecked == false && LowPlanetTemperature.IsChecked == false)              {                  errorMessage += "\n- Temperature level must be selected.";              }                if (!string.IsNullOrEmpty(errorMessage))              {                  MessageBox.Show($"Invalid input(s): {errorMessage}");                  return;              }              {                  int temperatureLevel = 0;                  if (HighPlanetTemperature.IsChecked == true)                  {                      temperatureLevel = 2;                  }                  else if (MediumPlanetTemperature.IsChecked == true)                  {                      temperatureLevel = 1;                  }                  string planetName = planetNameTextBox.Text;                  bool LoraxMode = false;                  if (LoraxModeBox.IsChecked == true)                  {                      LoraxMode = true;                  }                  TerrestrialPlanet terrestrialPlanet = new TerrestrialPlanet(PlanetCanvas, r);                  terrestrialPlanet.GenerateTerrestrialPlanet(planetName, lifeLevel, r, waterLevel, temperatureLevel, LoraxMode, moonNumber, 0);                }          }          public void AddPlanetToQueue(string planetName, int lifeLevel, int waterLevel, int radius, int temperature, int moonNumber)          {                           rightQueue.Enqueue($"Planet {rightQueue.Count}: {planetName} - Life: {lifeLevel} - Water/Lava: {waterLevel} - Radius: {radius} - Temp: {temperature} - Moons: {moonNumber}");                           UpdateQueue();          }          public void AddStarToQueue(string starName, int fieldStrength, int radius, int temperature, int systype)          {                           rightQueue.Enqueue($"Star: {starName} - Field Strength: {fieldStrength} - Radius: {radius} - Temperature: {temperature} - Star Number: {systype + 1}");                           UpdateQueue();          }          private void UpdateQueue()          {                           RightQueueBox.Items.Clear();                           foreach (var item in rightQueue)              {                  RightQueueBox.Items.Add(item);              }          }          private void ClearStarCanvas\_Click(object sender, RoutedEventArgs e)          {              StarCanvas.Children.Clear();          }          private void ClearPlanetCanvas\_Click(object sender, RoutedEventArgs e)          {              PlanetCanvas.Children.Clear();          }          private void LoadStar\_Click(object sender, RoutedEventArgs e)          {              string errorMessage = "";                if (string.IsNullOrEmpty(starNameTextBox.Text))              {                  errorMessage += "\n- Star name is required.";              }                if (!int.TryParse(fieldStrengthTextBox.Text, out int fieldStrength))              {                  errorMessage += "\n- Field strength must be a valid integer.";              }                if (!int.TryParse(starRadiusTextBox.Text, out int r))              {                  errorMessage += "\n- Radius must be a valid integer.";              }                if (UnarySystem.IsChecked == false && BinarySystem.IsChecked == false && TrinarySystem.IsChecked == false)              {                  errorMessage += "\n- System type must be selected.";              }                if (HighStarTemperature.IsChecked == false && MediumStarTemperature.IsChecked == false && LowStarTemperature.IsChecked == false)              {                  errorMessage += "\n- Temperature level must be selected.";              }                if (!string.IsNullOrEmpty(errorMessage))              {                  MessageBox.Show($"Invalid input(s): {errorMessage}");                  return;              }              else if (starLoaded == true)              {                  MessageBox.Show("A Star has already been generated.");                  return;              }                {                  int systype = 0;                  int temperatureLevel = 0;                  if (HighStarTemperature.IsChecked == true)                  {                      temperatureLevel = 2;                  }                  else if (MediumStarTemperature.IsChecked == true)                  {                      temperatureLevel = 1;                  }                  if (BinarySystem.IsChecked == true)                  {                      systype = 1;                  }                  else if (TrinarySystem.IsChecked == true)                  {                      systype = 2;                  }                  globalFieldStrength = fieldStrength;                  string starName = starNameTextBox.Text;                    Star  star = new Star(SpaceCanvas, r);                  starLoaded= true;                  star.GenerateStar(starName, r, temperatureLevel, systype);                  AddStarToQueue(starName, fieldStrength, r, temperatureLevel, systype);                  JObject starJson = star.GenerateStarAttributesToJson(starName, fieldStrength, r, temperatureLevel, systype);                  starsJson.Add(starJson);                  if (DeathStarBox.IsChecked == true)                  {                      DeathStarSpace.Visibility = Visibility.Visible;                  }                  else if (DysonSphereBox.IsChecked == true)                  {                      DysonSphereSpace.Visibility = Visibility.Visible;                  }              }          }          private void GenerateStar\_Click(object sender, RoutedEventArgs e)          {              string errorMessage = "";                if (string.IsNullOrEmpty(starNameTextBox.Text))              {                  errorMessage += "\n- Star name is required.";              }                if (!int.TryParse(fieldStrengthTextBox.Text, out int fieldStrength))              {                  errorMessage += "\n- Field strength must be a valid integer.";              }                if (!int.TryParse(starRadiusTextBox.Text, out int r))              {                  errorMessage += "\n- Radius must be a valid integer.";              }                if (UnarySystem.IsChecked == false && BinarySystem.IsChecked == false && TrinarySystem.IsChecked == false)              {                  errorMessage += "\n- System type must be selected.";              }                if (HighStarTemperature.IsChecked == false && MediumStarTemperature.IsChecked == false && LowStarTemperature.IsChecked == false)              {                  errorMessage += "\n- Temperature level must be selected.";              }                if (!string.IsNullOrEmpty(errorMessage))              {                  MessageBox.Show("Invalid input(s):" + errorMessage);                  return;              }              {                  int temperatureLevel = 0;                  int systype = 0;                  if (HighStarTemperature.IsChecked == true)                  {                      temperatureLevel = 2;                  }                  else if (MediumStarTemperature.IsChecked == true)                  {                      temperatureLevel = 1;                  }                  if (BinarySystem.IsChecked == true)                  {                      systype = 1;                  }                  else if (TrinarySystem.IsChecked == true)                  {                      systype = 2;                  }                      string starName = starNameTextBox.Text;                    Star star = new Star(StarCanvas, r);                    star.GenerateStar(starName, r, temperatureLevel, systype);                  if (DeathStarBox.IsChecked == true)                  {                      DeathStarCanvas.Visibility = Visibility.Visible;                  }                  else if (DysonSphereBox.IsChecked == true)                  {                      DysonSphereCanvas.Visibility = Visibility.Visible;                  }              }          }          private void LoadTerrestrialPlanet\_Click(object sender, RoutedEventArgs e)          {                string errorMessage = "";                if (string.IsNullOrEmpty(planetNameTextBox.Text))              {                  errorMessage += "\n- Planet name is required.";              }                if (!int.TryParse(lifeTextBox.Text, out int lifeLevel))              {                  errorMessage += "\n- Life level must be a valid integer.";              }                if (!int.TryParse(moonNumberTextBox.Text, out int moonNumber))              {                  errorMessage += "\n- Moon number must be a valid integer.";              }                if (!int.TryParse(radiusTextBox.Text, out int r))              {                  errorMessage += "\n- Radius must be a valid integer.";              }                if (!int.TryParse(waterTextBox.Text, out int waterLevel))              {                  errorMessage += "\n- Water level must be a valid integer.";              }                if (HighPlanetTemperature.IsChecked == false && MediumPlanetTemperature.IsChecked == false && LowPlanetTemperature.IsChecked == false)              {                  errorMessage += "\n- Temperature level must be selected.";              }                if (!string.IsNullOrEmpty(errorMessage))              {                  MessageBox.Show("Invalid input(s):" + errorMessage);                  return;              }              else if (starLoaded != true)              {                  MessageBox.Show("A Star must be generated first.");                  return;              }              {                  int temperatureLevel = 0;                  if (HighPlanetTemperature.IsChecked == true)                  {                      temperatureLevel = 2;                  }                  else if (MediumPlanetTemperature.IsChecked == true)                  {                      temperatureLevel = 1;                  }                  string planetName = planetNameTextBox.Text;                  bool LoraxMode = false;                  if (LoraxModeBox.IsChecked == true)                  {                      LoraxMode = true;                  }                  TerrestrialPlanet terrestrialPlanet = new TerrestrialPlanet(SpaceCanvas, r);                  double planetarydist = globalFieldStrength \* rand.NextDouble() + 100;                  terrestrialPlanet.GenerateTerrestrialPlanet(planetName, lifeLevel, r, waterLevel, temperatureLevel, LoraxMode, moonNumber, planetarydist);                  AddPlanetToQueue(planetName, lifeLevel,waterLevel,r,temperatureLevel,moonNumber);                  JObject terrestrialPlanetJson = terrestrialPlanet.GenerateTerrestrialPlanetAttributesToJson(planetName, lifeLevel, r, waterLevel, temperatureLevel, LoraxMode, moonNumber, planetarydist);                  terrestrialPlanetsJson.Add(terrestrialPlanetJson);              }          }          private void LoadGaseousPlanet\_Click(object sender, RoutedEventArgs e)          {              string errorMessage = "";                if (string.IsNullOrEmpty(planetNameTextBox.Text))              {                  errorMessage += "\n- Planet name is required.";              }                if (!int.TryParse(moonNumberTextBox.Text, out int moonNumber))              {                  errorMessage += "\n- Moon number must be a valid integer.";              }                if (!int.TryParse(radiusTextBox.Text, out int r))              {                  errorMessage += "\n- Radius must be a valid integer.";              }                if (HighPlanetTemperature.IsChecked == false && MediumPlanetTemperature.IsChecked == false && LowPlanetTemperature.IsChecked == false)              {                  errorMessage += "\n- Temperature level must be selected.";              }                if (!string.IsNullOrEmpty(errorMessage))              {                  MessageBox.Show("Invalid input(s):" + errorMessage);                  return;              }              else if (starLoaded != true)              {                  MessageBox.Show("A Star must be generated first.");                  return;              }              {                  int temperatureLevel = 0;                  if (HighPlanetTemperature.IsChecked == true)                  {                      temperatureLevel = 2;                  }                  else if (MediumPlanetTemperature.IsChecked == true)                  {                      temperatureLevel = 1;                  }                  string planetName = planetNameTextBox.Text;                    GaseousPlanet gaseousPlanet = new GaseousPlanet(SpaceCanvas, r);                  double planetarydist = globalFieldStrength \* rand.NextDouble() + 100;                    gaseousPlanet.GenerateGaseousPlanet(planetName, temperatureLevel, r, moonNumber, planetarydist);                  AddPlanetToQueue(planetName, 0, 0, r, temperatureLevel, moonNumber);                  JObject gaseousPlanetJson = gaseousPlanet.GenerateGaseousPlanetAttributesToJson(planetName, temperatureLevel, r, moonNumber, planetarydist);                  gaseousPlanetsJson.Add(gaseousPlanetJson);              }          }            private void GenerateGaseousPlanet\_Click(object sender, RoutedEventArgs e)          {              string errorMessage = "";                if (string.IsNullOrEmpty(planetNameTextBox.Text))              {                  errorMessage += "\n- Planet name is required.";              }                if (!int.TryParse(moonNumberTextBox.Text, out int moonNumber))              {                  errorMessage += "\n- Moon number must be a valid integer.";              }                if (!int.TryParse(radiusTextBox.Text, out int r))              {                  errorMessage += "\n- Radius must be a valid integer.";              }                if (HighPlanetTemperature.IsChecked == false && MediumPlanetTemperature.IsChecked == false)              {                  errorMessage += "\n- Temperature level must be selected.";              }                if (!string.IsNullOrEmpty(errorMessage))              {                  MessageBox.Show("Invalid input(s):" + errorMessage);                  return;              }              {                  int temperatureLevel = 0;                  if (HighPlanetTemperature.IsChecked == true)                  {                      temperatureLevel = 2;                  }                  else if (MediumPlanetTemperature.IsChecked == true)                  {                      temperatureLevel = 1;                  }                  string planetName = planetNameTextBox.Text;                    GaseousPlanet gaseousPlanet = new GaseousPlanet(PlanetCanvas, r);                  gaseousPlanet.GenerateGaseousPlanet(planetName, temperatureLevel, r, moonNumber, 0);              }            }            private void GeneratePlanetName\_Click(object sender, RoutedEventArgs e)          {              SuggestedPlanetTextBox.Text = $"Suggested Planet Name: {EntityRegex.CalculatePlanetaryRegex()}";          }          private void GenerateStarName\_Click(object sender, RoutedEventArgs e)          {              SuggestedStarTextBox.Text = $"Suggested Star Name: {EntityRegex.CalculateStarRegex()}";          }          private void RefreshJson\_Click(object sender, RoutedEventArgs e)          {              JsonListBox.Items.Clear();              string[] jsonFiles = Directory.GetFiles("Presets/", "\*.json");              foreach (string jsonFile in jsonFiles)              {                  JsonListBox.Items.Add(System.IO.Path.GetFileName(jsonFile));              }            }            private void LoadJson\_Click(object sender, RoutedEventArgs e)          {              string checknull = (string)JsonListBox.SelectedItem;                if (checknull != null)              {                  string selectedJson = JsonListBox.SelectedItem.ToString();                  Parser parser = new Parser(SpaceCanvas, this, starLoaded);                  parser.ParseSavedJson(selectedJson);              }              else              {                  MessageBox.Show("Please select a file to load.");              }          }          private void SaveJson\_Click(object sender, RoutedEventArgs e)          {              fullJson.Merge(starsJson);              fullJson.Merge(terrestrialPlanetsJson);              fullJson.Merge(gaseousPlanetsJson);              Parser parser = new Parser(SpaceCanvas, this, true);              parser.SaveToJson(fullJson);          }          private void RemoveLatestEntity\_Click(object sender, RoutedEventArgs e)          {              rightQueue.Dequeue();              UpdateQueue();          }          private void CloseButton\_Click(object sender, RoutedEventArgs e)          {              Close();          }      }  } |

## **Classes (Folder containing classes):**

### BackgroundEntities.cs:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows;  using System.Windows.Controls;  using System.Windows.Media;  using System.Windows.Media.Converters;  using System.Windows.Media.Imaging;  using System.Windows.Shapes;    namespace MechanismSimulator.Classes  {       internal class BackgroundEntity : Body          {              Random rand = new Random();          public BackgroundEntity(Canvas spaceCanvas, double r, double θ)                  : base(spaceCanvas)              {              SpaceCanvas = spaceCanvas;              for (int i = 0; i < 1000; i++)              {                  CreateCometImage(r, θ);              }                }              public UIElement Shape { get; set; }            private void CreateCometImage(double r, double θ)          {                Ellipse bGent = new Ellipse              {                  Width = r,                  Height = θ,                  Fill = Brushes.White              };                             Canvas.SetLeft(bGent, rand.NextDouble() \* 5000);              Canvas.SetBottom(bGent, rand.NextDouble() \* 5000);                             bGent.Tag = "bgent";              SpaceCanvas.Children.Add(bGent);          }      }  } |

### Body.cs:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Controls;  using System.Windows;  using System.Windows.Media;  using System.Windows.Media.Imaging;  using System.Windows.Shapes;    namespace MechanismSimulator.Classes  {      internal class Body      {          protected Canvas SpaceCanvas;            public Body(Canvas spaceCanvas)          {              SpaceCanvas = spaceCanvas;          }      }  } |

### EntityRegex.cs:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Controls;  using System.Text.RegularExpressions;  using System.Security.Policy;    namespace MechanismSimulator.Classes  {      internal class EntityRegex      {          private static Random rand = new Random();          private static string[] vowels = { "a", "e", "i", "o", "u" };          private static string[] consonants = { "b", "c", "d", "f", "g", "h", "j", "k", "l", "m", "n", "p", "q", "r", "s", "t", "v", "w", "x", "y", "z" };          private static string[] planetaryEndings = { "nus", "ter", "ria", "don", "rus", "tus", "tara", "thos", "lon", "ara", "thlon", "thon", " Skibidi" };          private static string[] starEndings = { "us", "a", "is", "ra", "on", "el", "um", "or", "al", "ix" };            public static string CalculatePlanetaryRegex()          {              int syllables = rand.Next(1, 4);              string name = "";                for (int i = 0; i < syllables; i++)              {                  if (rand.Next(2) == 0)                  {                      name += consonants[rand.Next(consonants.Length)];                  }                  name += vowels[rand.Next(vowels.Length)];              }                name = char.ToUpper(name[0]) + name.Substring(1);                name += planetaryEndings[rand.Next(planetaryEndings.Length)];                if (rand.Next(4) == 0)              {                  name += CalculateHex();              }              else if (rand.Next(4) == 0)              {                  name += CalculateRomanNumerals();              }                return name;          }          public static string CalculateStarRegex()          {              int syllables = rand.Next(1, 4);              string name = "";                for (int i = 0; i < syllables; i++)              {                  if (rand.Next(2) == 0)                  {                      name += consonants[rand.Next(consonants.Length)];                  }                  name += vowels[rand.Next(vowels.Length)];              }                name = char.ToUpper(name[0]) + name.Substring(1);                name += starEndings[rand.Next(starEndings.Length)];                return name;          }          private static string CalculateHex()          {              string alphanumeric = " ";              string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";                for (int i = 0; i < 3; i++)              {                  alphanumeric += chars[rand.Next(chars.Length)];              }                return alphanumeric;          }          private static string CalculateRomanNumerals()            {              Dictionary<int, string> romanNumerals = new Dictionary<int, string>              {                  {1,"I"},                  {4,"IV"},                  {5,"V"},                  {9,"IX"},                  {10,"X"},                  {40,"XL"},                  {50,"L"},                  {90,"XC"},                  {100,"C"},                  {400,"CD"},                  {500,"D"},                  {900,"CM"},                  {1000,"M"}              };              int denarynum = rand.Next(1, 1000);              string romanNumeral = " ";                foreach (var pair in romanNumerals.Reverse())              {                  while (denarynum >= pair.Key)                  {                      romanNumeral += pair.Value;                      denarynum -= pair.Key;                  }              }              return romanNumeral;          }          }  } |

### GaseousPlanet.cs:

|  |
| --- |
| using Newtonsoft.Json.Linq;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Controls;  using System.Windows.Ink;  using System.Windows.Media;  using System.Windows.Shapes;    namespace MechanismSimulator.Classes  {      internal class GaseousPlanet : World      {          public GaseousPlanet(Canvas spaceCanvas, int R) : base(spaceCanvas, R)          {              r = R;          }          public void GenerateGaseousPlanet(string planetName, int temperatureLevel, int r, int moonNumber, double planetarydist)          {              Random rand = new Random();                  SolidColorBrush baseColor = new SolidColorBrush();              if (temperatureLevel == 0)              {                  baseColor.Color = Color.FromRgb((byte)rand.Next(200, 256), (byte)rand.Next(230, 256), (byte)rand.Next(240, 256));              }              else if (temperatureLevel == 1)              {                  baseColor.Color = Color.FromRgb((byte)rand.Next(160, 180), (byte)rand.Next(150, 160), (byte)rand.Next(120, 150));              }              else if (temperatureLevel == 2)              {                  baseColor.Color = Color.FromRgb((byte)rand.Next(100, 256), 0, 0);              }              circle.Tag = "gaseousplanet";                           GenerateWorld(SpaceCanvas, circle, r, moonNumber);              Canvas.SetZIndex(SpaceCanvas, 0);              Canvas stripeCanvas = new Canvas();              stripeCanvas.Width = circle.Width;              stripeCanvas.Height = circle.Height;              circle.Fill = baseColor;              Canvas.SetLeft(circle, ((SpaceCanvas.ActualWidth - circle.Width) / 2) + planetarydist);              Canvas.SetTop(circle, ((SpaceCanvas.ActualHeight - circle.Height) / 2) + planetarydist);                           int numStripes = rand.Next(5, 10);              for (int i = 0; i < numStripes; i++)              {                  SolidColorBrush stripeColor = new SolidColorBrush();                  stripeColor.Color = Color.FromRgb((byte)(baseColor.Color.R + (byte)rand.Next(-20, 20)), (byte)(baseColor.Color.G + (byte)rand.Next(-20, 20)), (byte)(baseColor.Color.B + (byte)rand.Next(-20, 20)));                  Rectangle stripe = new Rectangle();                  Canvas.SetZIndex(stripe, 4);                  stripe.Width = 200;                  stripe.Height = 50 / numStripes;                  stripe.Fill = stripeColor;                      Canvas.SetTop(stripe, i \* (r \* 2 / numStripes));                  Canvas.SetLeft(stripe, 0);                  stripeCanvas.Children.Add(stripe);                  stripe.Tag = "gaseousplanet";              }              GenerateMoons(SpaceCanvas, circle, moonNumber);          }          public JObject GenerateGaseousPlanetAttributesToJson(string planetName, int temperatureLevel, int r, int moonNumber, double planetaryDist)          {              JObject gaseousPlanetJson = new JObject();              gaseousPlanetJson["type"] = "GaseousPlanet";              gaseousPlanetJson["name"] = planetName;              gaseousPlanetJson["radius"] = r;              gaseousPlanetJson["moonnumber"] = moonNumber;              gaseousPlanetJson["temperature"] = temperatureLevel;              gaseousPlanetJson["distancefromstar"] = planetaryDist;                return gaseousPlanetJson;          }      }  } |

### Parser.cs:

|  |
| --- |
| using Newtonsoft.Json;  using Newtonsoft.Json.Linq;  using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows;  using System.Windows.Controls;  using System.Windows.Documents;    namespace MechanismSimulator.Classes  {      internal class Parser : Body      {          protected MainWindow MainWindow;          protected bool StarLoaded;          public Parser(Canvas spaceCanvas, MainWindow mainWindow, bool starLoaded) : base(spaceCanvas)          {              SpaceCanvas = spaceCanvas;              MainWindow = mainWindow;              StarLoaded = starLoaded;          }            public void ParseSavedJson(string selectedJson)          {                JToken jsonToken;              using (StreamReader file = File.OpenText($"Presets/{selectedJson}"))              {                  using (JsonTextReader reader = new JsonTextReader(file))                  {                      jsonToken = JToken.ReadFrom(reader);                  }              }                foreach (var obj in jsonToken)              {                  string type = obj["type"].ToString();                    if (type == "Star")                  {                      StarLoaded = true;                      Star star = new Star(SpaceCanvas, (int)obj["radius"]);                      star.GenerateStar(obj["name"].ToString(), (int)obj["radius"], (int)obj["temperature"], (int)obj["systype"]);                      MainWindow.AddStarToQueue(obj["name"].ToString(), (int)obj["fieldstrength"], (int)obj["radius"], (int)obj["temperature"], (int)obj["systype"]);                    }                  else if (type == "TerrestrialPlanet")                  {                      TerrestrialPlanet terrestrialPlanet = new TerrestrialPlanet(SpaceCanvas, (int)obj["radius"]);                      terrestrialPlanet.GenerateTerrestrialPlanet(obj["name"].ToString(),                          (int)obj["lifelevel"],                          (int)obj["radius"],                          (int)obj["waterlevel"],                          (int)obj["temperature"],                          (bool)obj["lm"],                          (int)obj["moonnumber"],                          (double)obj["distancefromstar"]);                      MainWindow.AddPlanetToQueue(obj["name"].ToString(), (int)obj["lifelevel"], (int)obj["waterlevel"], (int)obj["radius"], (int)obj["temperature"], (int)obj["moonnumber"]);                    }                  else if (type == "GaseousPlanet")                  {                      GaseousPlanet gaseousPlanet = new GaseousPlanet(SpaceCanvas, (int)obj["radius"]);                      gaseousPlanet.GenerateGaseousPlanet(obj["name"].ToString(), (int)obj["temperature"], (int)obj["radius"], (int)obj["moonnumber"], (double)obj["distancefromstar"]);                      MainWindow.AddPlanetToQueue(obj["name"].ToString(), 0, 0, (int)obj["radius"], (int)obj["temperature"], (int)obj["moonnumber"]);                    }              }            }            public void SaveToJson(JArray fullJson)          {              bool successfulSave = false;              int increment = 0;              while (successfulSave == false)              {                  if(File.Exists($".Presets/Untitled-{increment}.json") == false)                  {                      File.WriteAllText($"Presets/Untitled-{increment}.json", fullJson.ToString());                      successfulSave = true;                  }                    increment = increment + 1;              }          }      }    } |

### Star.cs

|  |
| --- |
| using Newtonsoft.Json.Linq;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Controls;  using System.Windows.Media;  using System.Windows.Media.Imaging;  using System.Windows.Shapes;    namespace MechanismSimulator.Classes  {          internal class Star : Body          {              protected int r;              public Star(Canvas spaceCanvas, int R) : base(spaceCanvas)              {                  r = R;              }            public void GenerateStar(string starName, int r, int temperatureLevel, int systype)          {              Random rand = new Random();                    if (systype == 0)              {                                   DrawStar((SpaceCanvas.ActualWidth - r) / 2 + 20, (SpaceCanvas.ActualHeight - r) / 2 + 20, r, temperatureLevel, rand);              }              else if (systype == 1)              {                                   int spacing = 20;                  int totalWidth = 2 \* spacing + 2 \* r;                    DrawStar((SpaceCanvas.ActualWidth - totalWidth) / 2, (SpaceCanvas.ActualHeight - r) / 2, r, temperatureLevel, rand);                  DrawStar((SpaceCanvas.ActualWidth + totalWidth) / 2 - r, (SpaceCanvas.ActualHeight - r) / 2, r, temperatureLevel, rand);              }              else if (systype == 2)              {                                   double centerX = SpaceCanvas.ActualWidth / 2;                  double centerY = SpaceCanvas.ActualHeight / 2;                  double Ang = rand.NextDouble() \* 2 \* Math.PI;                  for (int i = 0; i < 3; i++)                  {                      double x = centerX + r \* Math.Cos(Ang + i \* 2 \* Math.PI / 3);                      double y = centerY + r \* Math.Sin(Ang + i \* 2 \* Math.PI / 3);                      DrawStar(x, y, r, temperatureLevel, rand);                  }              }          }            private void DrawStar(double x, double y, int r, int temperatureLevel, Random rand)          {              Ellipse starCircle = new Ellipse();              starCircle.Width = r \* 2;              starCircle.Height = r \* 2;              starCircle.StrokeThickness = 2;                             Color starColor = new Color();              if (temperatureLevel == 0)              {                  starColor = Color.FromRgb((byte)rand.Next(105, 200), 0, 0);              }              else if (temperatureLevel == 1)              {                  starColor = Color.FromRgb((byte)rand.Next(210, 256), (byte)rand.Next(106, 156), 0);              }              else              {                  starColor = Color.FromRgb((byte)rand.Next(200, 229), (byte)rand.Next(240, 244), (byte)rand.Next(250, 256));              }              starCircle.Fill = new SolidColorBrush(starColor);                             Canvas.SetLeft(starCircle, x - r);              Canvas.SetTop(starCircle, y - r);                             SpaceCanvas.Children.Add(starCircle);          }          public JObject GenerateStarAttributesToJson(string starName, int fieldstrength, int r, int temperatureLevel, int systype)          {              JObject starJson = new JObject();              starJson["type"] = "Star";              starJson["name"] = starName;              starJson["fieldstrength"] = fieldstrength;              starJson["radius"] = r;              starJson["temperature"] = temperatureLevel;              starJson["systype"] = systype;                return starJson;          }      }  } |

### TerrestrialPlanet.cs

|  |
| --- |
| using Newtonsoft.Json.Linq;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows;  using System.Windows.Controls;  using System.Windows.Media;  using System.Windows.Media.Imaging;  using System.Windows.Shapes;    namespace MechanismSimulator.Classes  {      internal class TerrestrialPlanet : World      {          public TerrestrialPlanet(Canvas spaceCanvas, int R) : base(spaceCanvas, R)          {              r = R;          }          public void GenerateTerrestrialPlanet(string planetName, int lifeLevel, int r, int waterLevel, int temperatureLevel, bool LoraxMode, int moonNumber, double planetarydist)          {                  Random rand = new Random();              GenerateWorld(SpaceCanvas, circle, r, moonNumber);              Canvas spotCanvas = new Canvas();              spotCanvas.Width = circle.Width;              spotCanvas.Height = circle.Height;              int grassNum = (int)(circle.Width \* circle.Height \* Math.Pow(lifeLevel, 2) / 5000);              for (int i = 0; i < grassNum; i++)              {                  Ellipse grassSpot = new Ellipse();                  grassSpot.Width = rand.Next(15, 25);                  grassSpot.Height = grassSpot.Width;                  if (LoraxMode == true)                  {                      int colorRand = rand.Next(1, 4);                      if (colorRand == 1)                      {                          grassSpot.Fill = new SolidColorBrush(Color.FromRgb(152, 251, 152));                        }                      else if (colorRand == 2)                      {                          grassSpot.Fill = new SolidColorBrush(Color.FromRgb(255, 182, 193));                        }                      else if (colorRand == 3)                      {                          grassSpot.Fill = new SolidColorBrush(Color.FromRgb(255, 255, 102));                        }                      else                      {                          grassSpot.Fill = new SolidColorBrush(Color.FromRgb(221, 160, 221));                        }                  }                  else if (temperatureLevel == 0)                      grassSpot.Fill = new SolidColorBrush(Color.FromRgb(0, (byte)rand.Next(30, 50), 0));                  else if (temperatureLevel == 1)                  {                      grassSpot.Fill = new SolidColorBrush(Color.FromRgb(0, (byte)rand.Next(50, 70), 0));                  }                  else                      grassSpot.Fill = new SolidColorBrush(Color.FromRgb((byte)rand.Next(30, 45), (byte)rand.Next(25, 30), (byte)rand.Next(10, 20)));                  double grassSpotX = rand.NextDouble() \* circle.Width;                  double grassSpotY = rand.NextDouble() \* circle.Height;                    Canvas.SetLeft(grassSpot, grassSpotX);                  Canvas.SetTop(grassSpot, grassSpotY);                    spotCanvas.Children.Add(grassSpot);                  grassSpot.Tag = "terrestrialplanet";              }                           int waterNum = (int)(circle.Width \* circle.Height \* Math.Pow(waterLevel, 2) / 5000);              for (int i = 0; i < waterNum; i++)              {                  Ellipse waterSpot = new Ellipse();                  waterSpot.Width = rand.Next(1, 25);                  waterSpot.Height = waterSpot.Width;                  if (temperatureLevel == 0)                  {                      waterSpot.Fill = new SolidColorBrush(Color.FromRgb((byte)rand.Next(200, 229), (byte)rand.Next(240, 244), (byte)rand.Next(250, 256)));                  }                  else if (temperatureLevel == 1)                  {                      waterSpot.Fill = new SolidColorBrush(Color.FromRgb(0, 0, (byte)rand.Next(100, 255)));                  }                  else                  {                      waterSpot.Fill = new SolidColorBrush(Color.FromRgb((byte)rand.Next(229, 255), (byte)rand.Next(0, 40), (byte)rand.Next(0, 20)));                  }                    double waterX = rand.NextDouble() \* circle.Width;                  double waterY = rand.NextDouble() \* circle.Height;                    Canvas.SetLeft(waterSpot, waterX);                  Canvas.SetTop(waterSpot, waterY);                    spotCanvas.Children.Add(waterSpot);                  waterSpot.Tag = "terrestrialplanet";                  }                             SolidColorBrush landBrush = new SolidColorBrush();              if (temperatureLevel == 0)              {                  landBrush.Color = Color.FromRgb(230, 230, 230);              }              else if (temperatureLevel == 1)              {                  landBrush.Color = Color.FromRgb(100, 100, 100);              }              else              {                  landBrush.Color = Color.FromRgb(20, 20, 20);              }              Ellipse stone = new Ellipse();              stone.Width = circle.Width;              stone.Height = circle.Height;              stone.Fill = landBrush;              stone.Tag = "terrestrialplanet";              circle.Tag = "terrestrialplanet";                           Grid grid = new Grid();              grid.Children.Add(stone);              grid.Children.Add(spotCanvas);                circle.Fill = new VisualBrush(grid);                             Canvas.SetLeft(circle, ((SpaceCanvas.ActualWidth - circle.Width) / 2) + planetarydist);              Canvas.SetTop(circle, ((SpaceCanvas.ActualWidth - circle.Width) / 2) + planetarydist);              GenerateMoons(SpaceCanvas, circle,moonNumber);            }          public JObject GenerateTerrestrialPlanetAttributesToJson(string planetName, int lifeLevel, int r, int waterLevel, int temperatureLevel, bool loraxMode, int moonNumber, double planetaryDist)          {              JObject terrestrialPlanetJson = new JObject();              terrestrialPlanetJson["type"] = "TerrestrialPlanet";              terrestrialPlanetJson["name"] = planetName;              terrestrialPlanetJson["lifelevel"] = lifeLevel;              terrestrialPlanetJson["radius"] = r;              terrestrialPlanetJson["waterlevel"] = waterLevel;              terrestrialPlanetJson["temperature"] = temperatureLevel;              terrestrialPlanetJson["moonnumber"] = moonNumber;              terrestrialPlanetJson["lm"] = loraxMode;              terrestrialPlanetJson["distancefromstar"] = planetaryDist;                return terrestrialPlanetJson;          }      }  } |

### World.cs:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Controls;  using System.Windows.Media;  using System.Windows.Media.Imaging;  using System.Windows.Shapes;  using System.Windows.Media.Animation;  using System.Windows;    namespace MechanismSimulator.Classes  {      internal class World : Body        {          protected int r;          protected internal Ellipse circle;          public World(Canvas spaceCanvas, int R) : base(spaceCanvas)          {              SpaceCanvas = spaceCanvas;              r = R;              circle = new Ellipse();          }          public static void GenerateWorld(Canvas SpaceCanvas, Ellipse circle, int  r, int moonNumber)          {                circle.Width = r \* 2;              circle.Height = r \* 2;              SpaceCanvas.Children.Add(circle);          }          public void GenerateMoons(Canvas SpaceCanvas, Ellipse circle, int moonNumber)          {              for (int i = 0; i < moonNumber; i++)              {                  Random moonRand = new Random();                  Ellipse moon = new Ellipse();                  moon.Width = moonRand.Next(3) + 2;                  moon.Height = moonRand.Next(3) + 2;                  moon.Fill = Brushes.Gray;                  SpaceCanvas.Children.Add(moon);                                     Canvas.SetLeft(moon, Canvas.GetTop(circle)/2);                  Canvas.SetTop(moon, Canvas.GetTop(circle)/2);                                     CreateMoonAnimation(moon, circle);              }          }          private void CreateMoonAnimation(Ellipse moon, Ellipse circle)          {              Random moonRand = new Random();              int timetaken = moonRand.Next(3) + 2;                DoubleAnimationUsingKeyFrames animationX = new DoubleAnimationUsingKeyFrames();              animationX.RepeatBehavior = RepeatBehavior.Forever;              DoubleAnimationUsingKeyFrames animationY = new DoubleAnimationUsingKeyFrames();              animationY.RepeatBehavior = RepeatBehavior.Forever;                             animationX.KeyFrames.Add(new LinearDoubleKeyFrame(Canvas.GetLeft(circle) + timetaken, TimeSpan.Zero));              animationY.KeyFrames.Add(new LinearDoubleKeyFrame(Canvas.GetTop(circle) + timetaken, TimeSpan.Zero));                             animationX.KeyFrames.Add(new LinearDoubleKeyFrame(Canvas.GetLeft(circle) + r\*2-5, TimeSpan.FromSeconds(timetaken)));              animationY.KeyFrames.Add(new LinearDoubleKeyFrame(Canvas.GetTop(circle) + r\*2-5, TimeSpan.FromSeconds(timetaken)));                animationX.KeyFrames.Add(new LinearDoubleKeyFrame(Canvas.GetLeft(circle), TimeSpan.FromSeconds(4)));              animationY.KeyFrames.Add(new LinearDoubleKeyFrame(Canvas.GetTop(circle), TimeSpan.FromSeconds(4)));                             Storyboard.SetTarget(animationX, moon);              Storyboard.SetTargetProperty(animationX, new PropertyPath(Canvas.LeftProperty));              Storyboard.SetTarget(animationY, moon);              Storyboard.SetTargetProperty(animationY, new PropertyPath(Canvas.TopProperty));                             Storyboard storyboard = new Storyboard();              storyboard.Children.Add(animationX);              storyboard.Children.Add(animationY);                             storyboard.Begin();          }      }    } |

## **Assets (Folder containing images):**

### border.png

|  |
| --- |
|  |

### callippicdial.png

|  |
| --- |
|  |

### corinthian.png

|  |
| --- |
|  |

### deathstar.png

|  |
| --- |
|  |

### dyson.png

|  |
| --- |
|  |

### english.png

|  |
| --- |
|  |

### exeligmosdial.png

|  |
| --- |
|  |

### olympiaddial.png

|  |
| --- |
|  |

### sarosdial.png

|  |
| --- |
|  |

### sothic.png

|  |
| --- |
|  |

## **bin:**

### Debug:

#### Sol.json:

|  |
| --- |
| [    {      "type": "Star",      "name": "Sun",      "fieldstrength": 10,      "radius": 60,      "temperature": 1,      "systype": 0    },    {      "type": "TerrestrialPlanet",      "name": "Mercury",      "lifelevel": 0,      "radius": 10,      "waterlevel": 0,      "temperature": 2,      "moonnumber": 0,      "lm": false,      "distancefromstar": 80    },    {      "type": "TerrestrialPlanet",      "name": "Venus",      "lifelevel": 15,      "radius": 25,      "waterlevel": 4,      "temperature": 2,      "moonnumber": 0,      "lm": false,      "distancefromstar": 130    },    {      "type": "TerrestrialPlanet",      "name": "Earth",      "lifelevel": 20,      "radius": 30,      "waterlevel": 6,      "temperature": 1,      "moonnumber": 1,      "lm": false,      "distancefromstar": 200    },    {      "type": "TerrestrialPlanet",      "name": "Mars",      "lifelevel": 20,      "radius": 20,      "waterlevel": 0,      "temperature": 2,      "moonnumber": 2,      "lm": false,      "distancefromstar": 250    },    {      "type": "GaseousPlanet",      "name": "Jupiter",      "radius": 60,      "moonnumber": 6,      "temperature": 1,      "distancefromstar": 320    },    {      "type": "GaseousPlanet",      "name": "Saturn",      "radius": 55,      "moonnumber": 8,      "temperature": 1,      "distancefromstar": 420    },    {      "type": "GaseousPlanet",      "name": "Uranus",      "radius": 25,      "moonnumber": 3,      "temperature": 0,        "distancefromstar": 500    },    {      "type": "GaseousPlanet",      "name": "Neptune",      "radius": 23,      "moonnumber": 2,      "temperature": 0,      "distancefromstar": 570    }  ] |

## **App.xaml:**

|  |
| --- |
| <Application x:Class="MechanismSimulator.App"               xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"               xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"               xmlns:local="clr-namespace:MechanismSimulator"               StartupUri="MainWindow.xaml">      <Application.Resources>          <Style x:Key="TemperatureButtonStyle" TargetType="ToggleButton">              <Setter Property="Margin" Value="5" />              <Setter Property="Padding" Value="10" />              <Setter Property="BorderThickness" Value="1" />              <Setter Property="BorderBrush" Value="DarkGray" />              <Setter Property="Background" Value="LightGray" />              <Setter Property="Foreground" Value="Black" />              <Setter Property="FontSize" Value="14" />              <Setter Property="Template">                  <Setter.Value>                      <ControlTemplate TargetType="ToggleButton">                          <Border Background="{TemplateBinding Background}"                              BorderBrush="{TemplateBinding BorderBrush}"                              BorderThickness="{TemplateBinding BorderThickness}"                              CornerRadius="5">                              <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" />                          </Border>                          <ControlTemplate.Triggers>                              <Trigger Property="IsChecked" Value="True">                                  <Setter Property="Background" Value="LightBlue" />                                  <Setter Property="Foreground" Value="White" />                                  <Setter Property="BorderBrush" Value="Blue" />                              </Trigger>                              <Trigger Property="IsMouseOver" Value="True">                                  <Setter Property="Background" Value="LightGray" />                                  <Setter Property="Foreground" Value="Black" />                                  <Setter Property="BorderBrush" Value="DarkGray" />                              </Trigger>                          </ControlTemplate.Triggers>                      </ControlTemplate>                  </Setter.Value>              </Setter>          </Style>      </Application.Resources>  </Application> |

### **App.xaml.cs:**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Configuration;  using System.Data;  using System.Linq;  using System.Threading.Tasks;  using System.Windows;    namespace MechanismSimulator  {      /// <summary>      /// Interaction logic for App.xaml      /// </summary>      public partial class App : Application      {      }  } |

## packages.config:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <packages>    <package id="Newtonsoft.Json" version="13.0.3" targetFramework="net472" />  </packages> |

# **Testing:**

## **Formulation of Testing Strategy:**

As part of the testing phase of my NEA, I have decided to break each test down into 4 categories:

|  |  |
| --- | --- |
| Testing Types | Explanation and Application |
| Modular Testing | Decomposition of mechanism into modules, which are tested individually to ensure they function correctly. |
| Black Box Testing | Testing the functionality of the software without considering its internal code structure. Essentially running the program as a student and focusing only on the outputs shown to me. |
| White Box Testing | Testing the internal structures or workings of the mechanism as a developer who knows how it should work.timer\_t |
| System Testing | Testing of the entire mechanism instead of just individual modules. Probably to be done with a YouTube Video. |

## **Testing Evidence:**

|  |  |
| --- | --- |
| Test Number and Type: | 1 - Black Box Testing |
| Objective: | 1 |
| Test Overview: | Loading of our Solar System as a Json and then displaying it on the screen. Further evidence on this objective (the planets moving according to the slider) will be in the System Test. |
| Inputs | No Inputs apart from user selecting Json, nothing can go wrong as the Json loaded is correct. Erroneous Json files will be explored later. |
| Expected Outcome | Planets being shown on the screen, along with moons and stars. Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus and Neptune displayed. Entities appear on the queue on the right. |
| Actual Outcome |  |
| Success/Failure | Success, all planets visibly modelled successfully and can be identified. |

\*Objective 2 will be tested in the System Test

|  |  |
| --- | --- |
| Test Number and Type: | 2.1- Modular Testing |
| Objective: | 3 |
| Test Overview: | Generation of custom solar system - Broken down into world and star classes, and then Terrestrial and Gaseous planet classes. This test is for Terrestrial Planet. |
| Inputs | Many inputs are here. There are 7 possible inputs just for this module. I will generate 8 planets to show the full extent of planetary generation, along with purposefully erroneous and extreme inputs.   |  |  |  | | --- | --- | --- | | Planet Number: | Input Type: | Inputs: | | Planet 1 | Typical | name - "Earth V2",  radius - 30,  temperature - 1,  moon number - 0,  life level - 20,  water/lava level - 5  lorax mode - false | | Planet 2 | Erroneous | name - "wrong datatypes",  radius - N/A,  temperature - "1",  moon number - 3,  life level - "chess.com"  water/lava level - 10  lorax mode - false | | Planet 3 | Extreme | name - "Bigworld",  radius - 10000,  temperature - 1,  moon number - 1,  life level - 0,  water level - 10  lorax mode - false | | Planet 4 | Typical | name - "Frostworld",  radius - 70,  temperature - 1,  moon number - 2,  life level - 0,  water level - 20  lorax mode - false | | Planet 5 | Erroneous | name - "cornball",  radius - N/A,  temperature - 1,  moon number - N/A,  life level - N/A,  water level - N/A  lorax mode - false | | Planet 6 | Typical | name - "LoraxPlanet",  radius - 30,  temperature - 1,  moon number - 0,  life level - 20  water level - 0  lorax mode - true | | Planet 7 | Extreme | name - "projectlagger",  radius - 40,  temperature - 1,  moon number - 0,  life level - 0,  water level - 2000  lorax mode - false | | Planet 8 | Typical | name - "truelavaworld",  radius - 50,  temperature - 2,  moon number - 0,  life level - 15,  water/lava level - 10  lorax mode - false | |
| Expected Outcome | Successful generation of the Typical planets, rejection of Erroneous data and an opportunity to re-enter information and a measured response to extreme data. |
| Actual Outcome | |  |  | | --- | --- | | Planet 1 |  | | Planet 2 |  | | Planet 3 |  | | Planet 4 |  | | Planet 5 |  | | Planet 6 |  | | Planet 7 |  | | Planet 8 |  | |
| Success/Failure | Mostly success. All typical planets generated correctly, and erroneous data was successfully denied with message boxes being outputted to let the user know exactly what went wrong.  The Extreme data was the most interesting. While my algorithm can handle quite large inputs depending on, of course, the hardware of the computer running the simulation, it struggles with large radiuses specifically. There is no reason for the user to input particularly large radiuses as you can zoom in and out on the canvas. In the design phase, I decided not to have any sort of contingency to stop extreme data from being inputted as it could handicap powerful machines. As seen in the planet 7 test, it required a lot of process memory which some machines might not have and might error. Therefore, it would be too complicated to implement machine-based recommendations on upper limits for generation - in my user guide, I have some general guides on what numbers to implement. |

|  |  |
| --- | --- |
| Test Number and Type: | 2.2- Modular Testing |
| Objective: | 3 |
| Test Overview: | Generation of custom solar system - Broken down into world and star classes, and then Terrestrial and Gaseous planet classes. This test is for Gaseous Planet. |
| Inputs | There are 5 possible inputs for this module. I will generate 5 gaseous planets, along with purposefully erroneous and extreme inputs.   |  |  |  | | --- | --- | --- | | Planet Number: | Input Type: | Inputs: | | Planet 1 | Typical | name - "gasplan",  radius - 60,  temperature - 1,  moon number - 6,  lorax mode - false | | Planet 2 | Erroneous | name - ,  radius - 30,  temperature - 1,  moon number - 0,  lorax mode - false | | Planet 3 | Erroneous | name - "gas",  radius - "fourteen",  temperature - 0,  moon number - 1,  lorax mode - false | | Planet 4 | Typical | name - "cornball2",  radius - 70,  temperature - 0,  moon number - 0,  lorax mode - false | | Planet 5 | Extreme | name - "break",  radius - 8888,  temperature - 1,  moon number - 4,  lorax mode - false | |
| Expected Outcome | Successful generation of the Typical planets, rejection of Erroneous data and an opportunity to re-enter information and a measured response to extreme data. |
| Actual Outcome | |  |  | | --- | --- | | Planet 1 |  | | Planet 2 |  | | Planet 3 |  | | Planet 4 |  | | Planet 5 |  | |
| Success/Failure | Successful apart from the Extreme, which makes sense as this is very close to test 2.1. There are no extra attributes to Gas Planets, so this is in essence a simpler version of test 2.1. |

|  |  |
| --- | --- |
| Test Number and Type: | 3 - Modular Testing |
| Objective: | 4 |
| Test Overview: | Generation of custom solar system - Broken down into world and star classes, and then Terrestrial and Gaseous planet classes. This test is for Star class. |
| Inputs | There are 5 possible inputs just for this module. I will generate 5 stars to show the full extent of planetary generation, along with purposefully erroneous and extreme inputs.   |  |  |  | | --- | --- | --- | | Star Number: | Input Type: | Inputs: | | Star 1 | Typical | name - "SolarV",  radius - 50,  field strength - 10  temperature - 1,  system type - 0, | | Star 2 | Typical | name - "DoubleSole",  radius - 40,  field strength - 20,  temperature - 2,  system type - 1, | | Star 3 | Typical | name - "Triplet",  radius - 30,  field strength - 30,  temperature - 0,  system type - 2, | | Star 4 | Extreme | name - "BlackHole",  radius - 30,  field strength - 1000,  temperature - 0,  system type - 0, | | Star 5 | Erroneous | name - "cornball3",  radius - "three",  field strength - N/A,  temperature - 0,  system type - 2, | |
| Expected Outcome | Successful generation of the Typical stars, rejection of Erroneous data and an opportunity to re-enter information and a measured response to extreme data. |
| Actual Outcome | |  |  | | --- | --- | | Star 1 |  | | Star 2 |  | | Star 3 |  | | Star 4 |  | | Star 5 |  | |
| Success/Failure | Mostly success. All stars generated correctly, and erroneous data was successfully denied with message boxes being outputted to let the user know exactly what went wrong.  For the Extreme value, despite having such a large gravitational field, it did not error, not just because I hadn't initialised any planets yet. All that that would do is bring the planets very close, but I haven't modelled any star swallowing planets, so it would not have a particularly interesting interaction. |

|  |  |
| --- | --- |
| Test Number and Type: | 3 - Black Box Testing |
| Objective: | 5 |
| Test Overview: | Changing of the language of the dials based on 3 buttons that act as radio buttons - E, C and S, which stand for English, Corinthian and Sothic. Also check that they are being output on the right. |
| Inputs | A simple button click that change the language accordingly. |
| Expected Outcome | Central dial changes language, along with control panel output change. Default language is English. |
| Actual Outcome |  |
| Success/Failure | Success, languages all have been successfully implemented with default being English. For dynamic changing, System Test will show dynamic changes while simulation is running. |

|  |  |
| --- | --- |
| Test Number and Type: | 4 - White Box Testing |
| Objective: | 6 |
| Test Overview: | Loading and saving Json, based off of bin/debug/Presets/...  Loading Json already proven from loading Sol.json which is our solar system. |
| Inputs | No Inputs apart from user selecting Json, nothing can go wrong as the Json loaded is correct. Not selecting any json will cause a prompt to select a Json. Json selection done based on box on the right. |
| Expected Outcome | Planets being shown on the screen being generated according to their Json attributes, along with being saved based on their program attributes to Json. |
| Actual Outcome |  |
| Success/Failure | Success, all planets visibly modelled successfully based on their values in their Json file. For dynamic generation, again, see System Test. |

|  |  |
| --- | --- |
| Test Number and Type: | 5 - Black Box Testing |
| Objective: | 7 |
| Test Overview: | Testing of a queue that is on the right which displays details of every body on the canvas excluding the background stars. Should be automatically added every time a new planet/star is generated, or when a Json is loaded, it should add all the planets and star on automatically. |
| Inputs | Inputs based on planetary and star inputs. Star and planet generation based on user inputs has already been tested in test 2.1 and 2.2. |
| Expected Outcome | Queue displays exactly what attributes each entity has, along with their position in terms of distance. Star goes at the top, and Star is not allowed to generate more than once. The first image is based off Json importing, the second is based off of generating the solar system yourself. |
| Actual Outcome |  |
| Success/Failure | Success, all planets visibly modelled successfully based on their values in their Json file. For dynamic generation, again, see System Test. |

\*Objective 8 will be exclusively tested in the System Test

\*Objective 9 will be exclusively tested in the System Test

|  |  |
| --- | --- |
| Test Number and Type: | 6 - Black Box Testing |
| Objective: | 10 |
| Test Overview: | Testing of the generation of fun modes to increase the enjoyability of the simulation. |
| Inputs | Inputs based on planetary and star inputs. Star and planet generation based on user inputs has already been tested in test 2.1 and 2.2. |
| Expected Outcome | Each of the 3 fun modes causes the algorithm to change appearance of certain planets and stars. |
| Actual Outcome |  |
| Success/Failure | Success, all fun modes generated. |

|  |  |
| --- | --- |
| Test Number and Type: | 7 - White Box Testing |
| Objective: | 11 |
| Test Overview: | Testing of the Regex being correct according to the preset rules of the IAU for planetary names. More generally, testing the generation of suggested planet and star names. |
| Inputs | No inputs apart from button clicks to generate regex. |
| Expected Outcome | Correct Regex outputted and in line with IAU rules. Could contain roman numerals or alphanumeric small strings at the end of the planet name. Star names should be ideally much smaller |
| Actual Outcome |  |
| Success/Failure | Success, all realistic names generated. Some have roman numerals, others alphanumeric strings. Stars have much simpler names as is common practice. |

\*Objective 12 will be exclusively tested in the System Test

\*Objective 13 was not fully met, however the gravitational field strength component was, and that will be exclusively tested in the System Test

\*Objective 14 will be exclusively tested in the System Test

|  |  |
| --- | --- |
| Test Number and Type: | 8 - System Testing |
| Objective: | Primarily Objectives 2, 8, 9, 12, 13, 14 The other objectives will also be tested |
| Test Overview: | A comprehensive test of the entire algorithm. This will include advanced planetary generation, exporting and importing and most importantly test the Timer\_Tick function which essentially underpins the entire time-based component to the simulation. |
| Inputs | Inputs based on planetary and star inputs. Star and planet generation based on user inputs has already been tested in test 2.1 and 2.2. |
| Expected Outcome | Generation of a completely independent solar system. Exporting this solar system. Then, being able to import it again to show the capabilities of Json. |
| Actual Outcome | https://youtu.be/jkXb39iaN\_A |
| Success/Failure | Overwhelming success, all different modules and parts of the algorithm work together to deliver the final completed simulation. |

# **Evaluation:**

## **Feedback from client and primary users:**

After completing this project, I showed it to my client Mr. Rieper. I showed him the algorithm and I gave him a few weeks to trial it out at his school and I held an interview with him after:

Me: Mr Rieper, tell me your overall thoughts of the solution.

Mr Rieper: I was very impressed by the solution even when I tested it out myself. I created a few custom solar systems, along with obviously loading our solar system which was included as a file. I really liked the fact that there was a help document which was very useful due to the complex nature of a simulation like this.

One of the best parts of this simulation was the fact that I could generate my own planets based on some inputs. I really enjoyed generating different types of planets like Lava planets, volcano planets, tundra planets, ocean planets and so forth. As all of the planets were randomly generated, no two worlds were the same visually. I spent a lot of time customising and making my own unique solar system, which I took a photograph of:

Me: What about your students? What did they think of the simulation?

Mr Rieper: They liked the file sharing aspect of it a lot. I think that the fact that they could send each other their .json files improved their collaboration together. The only problem that they had was that they thought that the slider was too slow: they had to wait a while for the simulation to actually move forward enough to see any meaningful impact on some of the dials. However, this was more of an issue of them being impatient, and I am certain in future iterations if you wanted to fix this it could be done very easily.

Me: Yes, all I would have to do is increase the maximum value for the slider. Also, another question, were there any problems with efficiency? I aimed to make my simulation as portable as possible, so I tried to reduce unnecessary iterations.

Mr Rieper: No, not really. There were a few students that tried to generate huge planets with radiuses in the thousands and with a lot of grass and water which caused the simulation to slow down significantly. However, this was the students' fault as the simulation clearly is not meant to simulate such large planets. I also think that the dedicated "close" button at the top right was helpful, as it worked even when accidentally generating far too large planets.

Me: Did you find any hiccups with putting erroneous inputs in my simulation?

Mr Rieper: No, not at all. In fact, I really liked the box that pops up because it told me exactly what was missing for me to generate the planet. For example, too many times I would try to generate a planet without clicking on one of the temperature buttons, and it would alert me to exactly what was missing. Also, I really appreciated the fact that some of the more ambiguous text boxes had the datatype next to them like Life Level for the terrestrial planets.

Me: Did you like the planet name suggestor?

Mr Rieper: I didn't really use it much, but I can definitely see its usage if your simulation was to be used as more of a tool. The Simulation helped me as a teacher as I took the opportunity to explain to my students exactly how exoplanets are named according to NASA.

Me: Were you expecting anything that was not included in the simulation?

Mr Rieper: I was expecting the Metonic cycle, as you included all the others, but that was a very small omission. I also was expecting for there to be a way to change the orbit shapes of planets. For example, some planets actually have elliptical orbits and as such cannot really be modelled with perfect circles. Also, I think that it was a little bit of an oversight to not include the gears.

Me: I completely understand all of your points. The reason I did not include the Metonic cycle was because the Metonic cycle is essentially identical to the Saronic cycle. The Saronic cycle is 0.029 years longer than the Metonic cycle, and so when you take into account the fact that they are 18 and 18.029 years respectively, I did not see the point of having two essentially identical dials. For the point about the orbit shapes, I agree that I could have included that. I believe that that would have been a good extension to my simulation, because of course there are many more factors that can be taken into account for a planet's orbit. However, for abstractive reasons, I decided to just have circular orbits. I did not include the gears for a similar reason as I did not want to flood the user with too much information. Thank you for your time, Sir.

## **Objective Evaluation:**

Green means fully achieved, yellow means partially achieved and red means not achieved.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number | Objective | Objective Implementation | Core/  Desirable | Achieved? |
| 1 | Displays our solar system when loaded from a preset and moves according to a slider which sets the speed at which the simulation moves. | Successful as I allowed my algorithm to load and save .json files which defined the format of the solar system. | Core | ✔ |
| 2 | Dials around the solar system move according to the aforementioned slider and output the current position of the different dials. | Successful as I implemented all of the dials excluding the Metonic which I explained why, and the slider is intrinsically tied to the Timer\_Tick procedure. | Core | ✔ |
| 3 | Generate planet accurately based on inputs such as temperature, water and life levels. Randomly and procedurally generated – same inputs will never produce the same looking planet. | Very successful as I also broke down the planets into gas and terrestrial planets and gave them separate. Successful random procedural generation of planets implemented. | Core | ✔ |
| 4 | Generate stars based on different inputs. Generation of multiple stars may not be doable due to coinciding gravitational fields. | Successful, more successful than initially thought as I included unary, binary and trinary systems as well. | Core | ✔ |
| 5 | Have the main central dial in 3 different languages – English, Corinthian and Sothic. Make it so that it also is output on the control panel on the right. | Successful, Dials defined by 2D array and then based on buttons the language is set. Also on the control panel it was successfully outputted. | Core | ✔ |
| 6 | Import and Export custom solar systems. To do this I need to develop my own parsing algorithm or maybe use serialization. | Importing and exporting was as previously mentioned done by using .json. Initial attempt at making own parsing algorithm was inefficient. | Core | ✔ |
| 7 | A queue on the right that when a new entity (star or planet) is created, it adds the name of the new entity and when the cursor is hovered over it displays the details of the planet. | Successful and this is also exported to .json when saved to the preset folder. I have all of the details of the planet which further helps for future usage. | Core | ✔ |
| 8 | Have background stars slowly moving in the background that improve the immersion. Make their speed be intrinsically connected to the speed slider. | Successful although I had a very weird bug with initialising the thousands of stars that took a long time to fix. Speed successfully related to the slider. | Core | ✔ |
| 9 | Control panel on the right extracts all information from the dials and puts it all on the right which condenses the actual mathematics of the program. | Implemented successfully but slightly barebones, which was due to the majority of the maths being done behind the UI in the classes. | Desirable | ✔ |
| 10 | Have fun settings and modes such as Lorax Mode which could change the appearance of a planet into a very colourful one, or for example a death star as a sun. | Successful but I made sure not to spend a lot of time on this step as it is simply to improve the fun element of the simulation. It took me around an hour to develop these modes. | Desirable | ✔ |
| 11 | Ability to generate planet and star names based off of a set of predefined endings and vowel + consonant combinations, maybe with Regex. | Very successful regex implementation that follows the internationally agreed standards for naming planets. Also was expanded to stars. | Desirable | ✔ |
| 12 | Have all moons be represented. However, this might be too ambitious as some planets have hundreds of moons and so it could slow down the program. | Successful generation of moons, but it was not random as it would have severe impact on efficiency. I could have made my code more clear by creating a separate class for moons, but I just used a procedure that would be inherited by both gas and terrestrial planets. | Desirable | ✔ |
| 13 | Have custom orbits for custom solar systems. This may be difficult due to the many parameters in place, and may even be completely scrapped for complexity reasons. | Not fully implemented unfortunately due to the sheer number of parameters and mathematics required to accurately model custom orbits. regular orbits based on field strength successfully implemented | Desirable | ✔ |
| 14 | Add custom controls to the canvas that allow for panning and zooming. | Successfully implemented by using matrices, which is an A level Further Maths technique. Panning developed with events based on mouse movement. I couldn't manage to implement a scroll wheel however, but the + and - buttons will do. | Desirable | ✔ |

## **Possible Extensions:**

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Extension | How Extension could be implemented | Difficulty Level |
| 1 | Develop a feature to simulate spacecraft missions, allowing users to launch probes or rovers to explore planets within the simulated solar system. | Based on my terrestrial planet class, a designer on the left of the UI could appear where the user could design their own probes and rovers, and send them to procedurally generated planets | 7/10 |
| 2 | Implement a multiplayer mode where multiple users can collaborate or compete within the same simulated solar system, allowing for shared creation and interaction. | Probably would require my simulation to become a web app for maximum collaboration potential. Would require a lot of my code to be built from the ground up again, even if I reused my classes. I think I would also have to redesign the UI. | 6/10 |
| 3 | Develop an educational module with interactive tutorials and quizzes to help users learn about astronomy concepts and celestial mechanics while using the simulation. | Could probably be a quite easy extension as it is simply a quizzing tool. Maybe I could use an external tool like Anki to create my own flashcards and then have simple quizzes. If I was to make this extension more complex, I could try implement my own difficulty-scaling algorithms to the questions. | 3/10 |
| 4 | Develop a mobile application version of the simulation, making it accessible to a wider audience and enabling users to explore the solar system on smartphones and tablets. | Would be very interesting to develop as I have never really developed a proper mobile application. I might have to limit some of the inputs since phones tend to have much lower processing power than desktops and laptops, and as such, I would have to downgrade some of the advanced logic of my WPF app. If I learned the syntax for mobile app development, it might not be that difficult | 5/10 |
| 5 | Implement a feature to simulate celestial events such as eclipses, transits, and conjunctions, enhancing the realism and educational value of the simulation. | Would require a lot more inputs from the user and generally a lot more code, especially if I was to do this for randomly generated and user-created solar systems. Overall, it would be a lot of work and a lot of maths and it would rival even commercial space simulators. | 10/10 |
| 6 | Add more presets such as nearby solar systems and solar systems from video games and film, which could help improve the entertainment element of the simulation. | Importing and exporting was as previously mentioned done by using .json. Therefore, to do this, all I would need is the data and I could write my own .json files and then load them. I would then place them in the assets folder. For custom sprites, I could just do what I did for the dyson sphere and death star by creating XAML images and turning them visible when required. | 2/10 |
| 7 | Add the procedural generation of entire solar systems based on probabilities and advanced physics. | Could be quite doable, but it depends on the level I do it. I think that if I was to determine probabilities of such solar systems existing and then output them, it could be manageable, but I would require a lot of physics | 4/10 |
| 8 | Have custom orbits for planets and stars based on their gravitational fields interacting with each other and other parameters to add realism. | Like extension 5, it would be quite difficult for larger systems, however if I was to do it for small systems it could be manageable. I think that with more and more planets and stars the complexity would increase exponentially. | 6/10 |