**Creating a population simulator**

Contents page:

1. **Project proposal form**
2. **Why did I choose to make a population simulator?**
   1. What is a population simulator?
   2. Why Rust?
3. **What was my plan for the final project?**
4. **Progress**
   1. Research
      1. Research analysis
   2. App progress
   3. Problems…
      1. Problems with multithreading
      2. Problems with general optimisations
5. **The outcome**
   1. How to run the application
   2. Code analysis
6. **Evaluation of what I achieved and a reflection**
   1. What did I learn and achieve?



*By Pawel Zdunowski*

**1. Project Proposal form**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Learner Name** | **Pawel Zdunowski** | |  | **Learner number** |  |  |
|  |  | | |  | |  |
| **Centre Name** | **The Saint John Henry Newman School** | |  | **Centre Number** |  |  |
|  |  | | |  | |  |
| **Teacher Assessor** |  | |  | **Date** | **13/12/22** |  |
|  |  | | |  | |  |
| **Unit** |  | |  |  | |  |
|  |  | | |  | |  |
| **Proposed project title** | | **How do populations grow?** | | | |  |

|  |
| --- |
| **Section One: Title, objectives, responsibilities** |
| **Title or working title of the project (in the form of a question):**  How do populations grow?  **Project objectives (e.g., what is the question you want to answer? What do you want to learn how to do? What do you want to find out?)**  I would love to learn about, how and why populations of people grow and why how/they evolve. This will include details such as fertility, mortality, epidemics and lovers. I also want to use rust (a programming language as my base for the application which will be made), this will expand my knowledge of it.  **If it is a group project, what will your role or responsibilities be?**  N/a |

|  |
| --- |
| **Section Two: Reasons for choosing this project** |
| **Reasons for choosing the project (e.g., links to other subjects you are studying, personal interest, plans, knowledge/skills you want to improve, reasons why the topic is important):**  I have always wanted to create a simulation that can be a somewhat accurate model of how populations grow and become what they are today. I think that it is important to know how humanity has got to the point that is now and why/how it changed drastically. We would most probably not exist in the way we are now without things occurring. Also as mentioned above I would like to further my knowledge of the programming language ‘rust’ which I could use for a job in the future. |

|  |  |
| --- | --- |
| **Section Three: Activities and timescales** | |
| **Activities to be carried out during the project (e.g., research, data collection, numerical analysis, writing, preparing for the presentation, etc)**   * 1. Graph analysis * 2. Research for different averages: fertility, mortality and epidemic details * 3. Adding features, fixing problems with the code | **How long will this take?**   * 1. 3 weeks * 2. A month /a month and a half * 3. The remaining time / at least 2 months |
| **Milestone one:**   * A large amount of research, completion of v0.3 of the PopSimRust project (the name of the programming repository)   **Target date (set by tutor-assessor):**  **Milestone two:**   * A large amount of research, completion of v0.6 of the PopSimRust project (the name of the programming repository)   **Target date (set by tutor-assessor):** | |
| **Section Four: Resources** | |
| **What resources will you need for your research, data collection, writing up and presentation (e.g., equipment, ICT, libraries, books, journals)**   * The Internet * A computer * An IDE (an application that helps programmers develop software code efficiently) * My programming project repository: *PopSimRust on GitHub:* [*https://github.com/NotAF0e/PopSimRust*](https://github.com/NotAF0e/PopSimRust)   **What areas of research will you cover?**   * The characteristics of a population * How to create a user interface * Why populations grow: size, spread, etc, due to factors: fertility, mortality, epidemics | |

|  |
| --- |
| **Comments and agreement from tutor-assessor** |
| **Comments (optional):**  **I confirm that the project is appropriate.**  **Agreed: (name) (date)** |
| **Comments and agreement from the proposal checker** |
| **Comments (optional):**  **I confirm that the project is appropriate.**  **Agreed: (name) (date)** |

**2. Why did I choose to make a population simulator?**

2.1. What is a Population simulator?

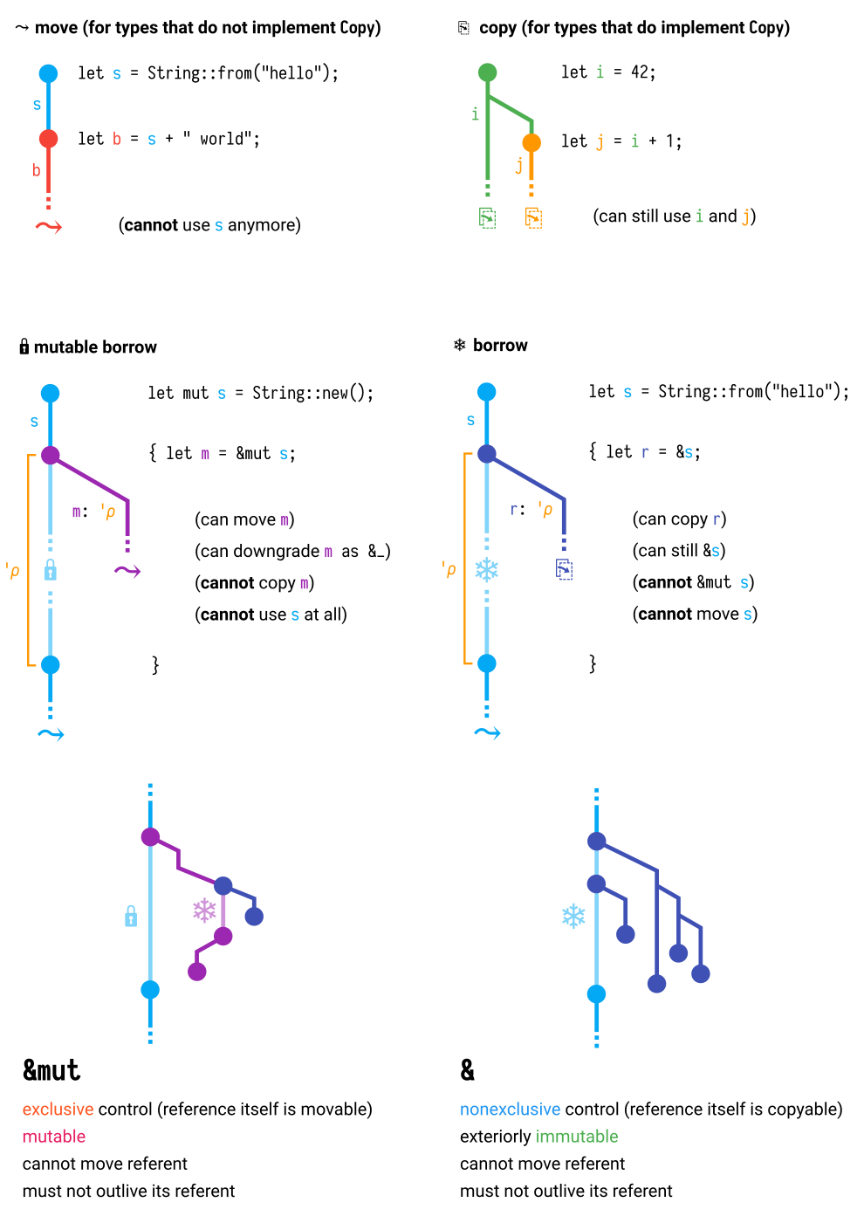
A population simulator is a computer program that can model the behaviour of groups of individuals within a defined system, like a country or even the entire planet. They are used in a variety of fields, including epidemiology, ecology, and sociology, to help researchers understand how complex systems of interacting people behave over time. By simulating the behaviour of groups of individuals, population simulators can provide insights into how different factors, such as social networks, environmental variables, or policy interventions, affect the behaviour of the overall system.

2.2. Why Rust?

I chose to use *Rust:* [***https://www.rust-lang.org/***](https://www.rust-lang.org/), a blazingly fast programming language as my foundation for this project for many reasons. One of them is its community; the community which surrounds this language is incredibly strong and helpful which would be very important if I faced dire problems along the way. This community also has created a massive quantity of “crates” which are mini codebases. These “crates” can help with a multitude of things such as: creating a UI. These will help me, in terms of my final project, because they will greatly increase the speed at which I can create new features. Here is a link to the *crate website:* [***https://crates.io/***](https://crates.io/)

The second reason why I chose Rust as my programming language was because of its superior speed compared to others like *JavaScript:* [**https://www.javascript.com/**](https://www.javascript.com/). Even though the language is quite hard to learn, I was certain that the speed of the simulation was more important than the time I would spend learning the language. This speed would allow for larger population sizes and huge complex features (if I had time). But also, because I wanted to challenge myself by learning a more complex programming language than Python for use in possible future apps.

I chose to create a population simulator for several reasons. 2 years ago, I began learning *Python:* [***https://www.python.org/***](https://www.python.org/) which is a simple programming language to create slow but useful applications. I created a few small apps/games, one of these projects being a population simulator. I was not happy with the final result though because the simulation which this app could provide was very unrealistic and the app lacked a user interface (UI). I also had many ideas for features which would require a full rewrite of the codebase due to the project’s lack of modularity in its design. Some of these features include epidemics, controlling the simulation and some start settings.

The final reason why I chose Rust was due to its modernity. This programming language has many features which make it stick out from the crowd and make it more of a joy to code in. One of these features is a borrow checker. I will keep its definition brief, but it is a trait of Rust which only allows data from one variable to be given to another. To combat this though, there is a system where the programmer can borrow one’s variable’s data from another. This trait eliminates the issues of memory management in code which are very difficult to diagnose and often lead to huge problems. This was beneficial for me, even though hard to learn because issues in my code will only be caused by me, not the memory. This, therefore, increases – again - the speed at which I could focus on making my plan for the project a reality. Here is a diagram describing the borrow checker’s many attributes:

Before the beginning of the HPQ, in November last year, I began recreating the python project in its entirety in Rust (after learning some of it using *rustlings:* [***https://github.com/rust-lang/rustlings***](https://github.com/rust-lang/rustlings)), but the app was somehow still slow and still lacked a UI. This may have been partly due to my limited skill and knowledge of Rust. I chose to create a detailed population sim for this HPQ project because I could spend more time on it at home and also be able to familiarise myself more with Rust, which began to interest me.

Another reason why I chose this project and did not choose to do something else was that creating a population sim is simple at heart. But if there was time, then I would be able to add many layers to its features. This made the final project very mailable and I would only add very complex features (like the ones spoken about in the first paragraph) if I had time.

Also, I have never worked on a huge programming project so I knew that this one could allow me to learn what it is like. This is very important to me because I need experience when I continue programming but at a job. Large projects like these, completed from start to finish by one person are what usually make an employer hire. This is *proven here:* [***https://www.educative.io/blog/how-to-plan-a-coding-project***](https://www.educative.io/blog/how-to-plan-a-coding-project) at “What can projects do for you?”.

Finally, it would be more flexible than an essay. As I could disable entire features of the project if they were not ready before the deadline, whereas in an essay you cannot be as lenient because all paragraphs must stay intact. This ensured that the final project was of the highest quality. It also ensured that I am not stressed by the task at hand and can work more productively.

**I chose to create a population simulation as my HPQ project for these reasons:**

* A learning experience to improve my programming skill in Rust

Why rust?

Its community

Its speed

And its modernity

* Being able to iterate a design which I had made previously.
* A taste of what managing large projects is like
* Greater flexibility than other projects leading to a higher quality outcome.

**3. What was my plan for the final project?**

The plan for my final project was, in short, to have a semi-accurate simulation of population growth and the various details that come with that task. I wanted every person in the simulation to be a simulated entity with various attributes. Some of these include age, name, sex, fertility, a lover, epidemic values and a random value seed. Most of these attributes would be used to determine the actions of these simulated beings. One of these is reproduction. The way that reproduction would work would be by randomly assigning a lover to a person. A baby would only be born under strict circumstances such as certain fertility values, age and the strength of love that these 2 people have for each other.

Another idea was to implement realistic values for different population factors, instead of guessing what looked correct, which would require a lot of research, to add them appropriately. Fertility is the average number of children a woman has in her lifetime, and I planned to implement this by assigning multiple values of "fertility" that change throughout a woman's lifetime. These "fertility" values would be combined with a randomizer to determine if a baby should be born that month. This approach could work well because the average of all values is equal to a certain fertility rate, just like in real life. I describe this later in 5.2.

The next, but challenging, prospect that I had for this project was to add a UI to the application so that users can interact with it, with less struggle. Without a UI, users would need to enter commands through a command line interface which is a lot less user-friendly. I planned to use a Rust crate for the GUI (graphical user interface) called *EGUI****:*** [***https://github.com/emilk/egui***](https://github.com/emilk/egui). On the next page is my rough graphical plan for how it would look.

Population amount

Other info

Controls

Other controls

Table with all people and some details about them

Bottom bar with various settings

Other info and controls

The final idea was to increase the speed of the simulation using different ways of code optimisation. I thought that *multithreading:* [***https://www.internalpointers.com/post/gentle-introduction-multithreading***](https://www.internalpointers.com/post/gentle-introduction-multithreading)would have been a good solution to this as it is parallel meaning two tasks could run at the same time instead of being bound to one central simulation loop. This means that I would have been able to decouple the rendering of the UI and the simulation of the population into 2 threads which would send information to each other.

*From now on I will be referring to the project as PopSimRust*

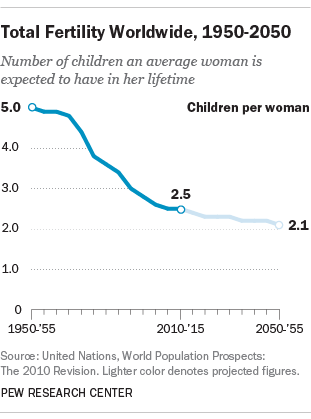
**4. Progress**

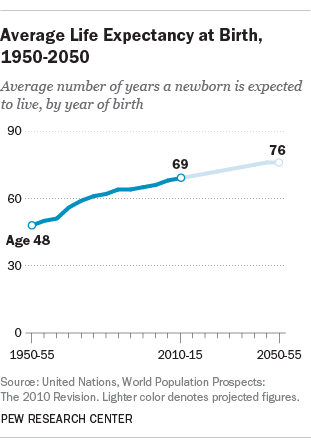
4.1. Research

First, I began the project with about 1 month of research as I knew I would need a lot more time to complete a fully working application. Here are all the topics which I researched and some information about them in a bibliography:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Source Title** | **Type of source** | **Where is it from?** | **Is it credible? How do you know?** | **What information have you learnt?**  **What questions does it raise?** | **Link** |
| Main factors driving population growth | Article with data (graphs) | Pew research centre | There is detailed data from outside trusted sources. But the source is a bit outdated (2015) | The main factors driving population growth are:   * Fertility (Number of children for a woman to have in a lifetime) * Life expectancy * Migration (Amount of people leaving the country per year)   My question is what are the detailed reasons for these drivers? | [**https://www.pewresearch.org/religion/2015/04/02/main-factors-driving-population-growth/**](https://www.pewresearch.org/religion/2015/04/02/main-factors-driving-population-growth/) |
| OECD data | A website with data (graphs) | OECD data | Very updated data | I have gained more info for the above, like for fertility, it has helped with the data which was needed for my simulation | **https://data.oecd.org/pop/fertility-rates.htm** |
| The R number | A website | Gov UK | It is a government website so it is very credible | Talks about the r number which is a factor in epidemics | [**https://www.gov.uk/guidance/the-r-value-and-growth-rate**](https://www.gov.uk/guidance/the-r-value-and-growth-rate) |
| EGUI docs | Coding documentation | Rust documentation | It is created by the EGUI developers so it is very credible and always up to date. | It has taught me how to create a good graphical user interface for the simulation. But I will still (potentially) need to learn how to use threads to multithread my code to increase the speed of the sim. | [**https://docs.rs/egui/latest/egui/**](https://docs.rs/egui/latest/egui/) |
| Multithreading - From Python to Rust | A video | YouTube | Uses the Rust docs so very credible | Teaches how to use multithreading. This will help me as I failed on my first attempt | [**https://www.youtube.com/watch?v=nPG-xcS\_AKQ**](https://www.youtube.com/watch?v=nPG-xcS_AKQ) |
| The Rust Lang Forums | A forum website | The Rust Lang Team | It is by the creators of the Rust programming language. So very credible. | See 4.3.2 | [**https://users.rust-lang.org/**](https://users.rust-lang.org/) |

4.1.1. Research analysis

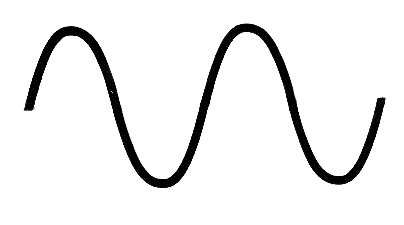
The source “Main factors driving population growth” (secondary research) was very helpful for many reasons, here are some of these reasons:

* It provided important average statistics. It described fertility and how the average worldwide fertility changed from 1950 to 2010. I was considering adding dynamic fertility based on some of the extra information in the report but I realized that that was out of the time scope of the project. It provided average fertility for 2010 of 2.5. Which is approximately the value which I then implemented.
* Another important statistic was the average life expectancy from birth which was 69 years old the average in 2010. In my simulation life expectancy would be weighted, meaning that there is a certain chance of dying at all ages which would change the older a person gets. This is explored in 5.2.
* I did also gain some knowledge about migration from the report. But as you can see from the full list of changes done to PopSimRust “[07/02/23] Removed immigration system as it was causing many issues”. These issues include errors with lovers who did not leave the population together and unnatural-looking population graphs. These graphs were strange as they did not reassemble the *exponential population growth formula:* [***https://math.libretexts.org/Bookshelves/Applied\_Mathematics/Book%3A\_College\_Mathematics\_for\_Everyday\_Life\_(Inigo\_et\_al)/04%3A\_Growth/4.02%3A\_Exponential\_Growt****h*](https://math.libretexts.org/Bookshelves/Applied_Mathematics/Book%3A_College_Mathematics_for_Everyday_Life_(Inigo_et_al)/04%3A_Growth/4.02%3A_Exponential_Growth).

The next source “OECD data” (secondary research) was quite useful for gaining more insight into fertility. It contains the average fertility for many countries which gave me more realistic values I could use myself in the simulation. Again, I theorised of ways I could make the values change dynamically but again; it was out of the project's time scope.

The next source is “The R number” (secondary research). This was the only source which I used for teaching myself how epidemics would be simulated due to time constraints. It was very useful as it described that the r number is the number of people on average who are infected by another person. I then implemented this number when creating the epidemic system.

After that, there are the “EGUI docs” (secondary research) which were a part of how I learnt how to use the EGUI crate. It contains all the important features and even has tutorials. For example, I used it when I was struggling to make a graph/plot. Here is the information that it provided for that task:

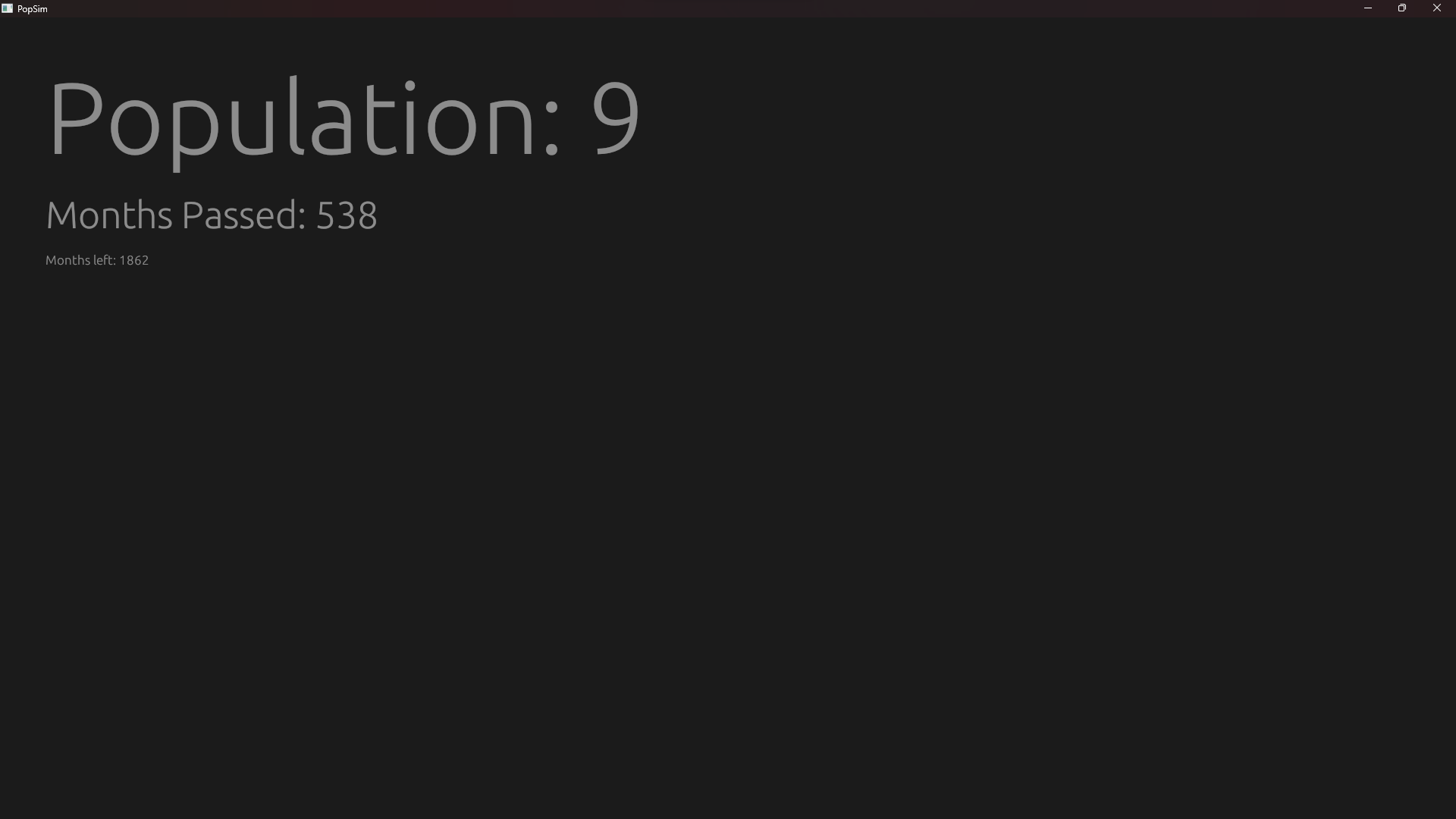


The above code would create a sine wave (shown in the image on the right) plot with 1000 points.

The source “Multithreading - From Python to Rust” (secondary research) was a very helpful video which used code from the rust docs to teach what threads are and how to use them in a Rust program efficiently.

The source “Rust Lang Forums” (primary research) refers to all posts which I created on the Rust forums to find out answers to questions that I could not find on the internet, their effects can be seen in 4.4.2.

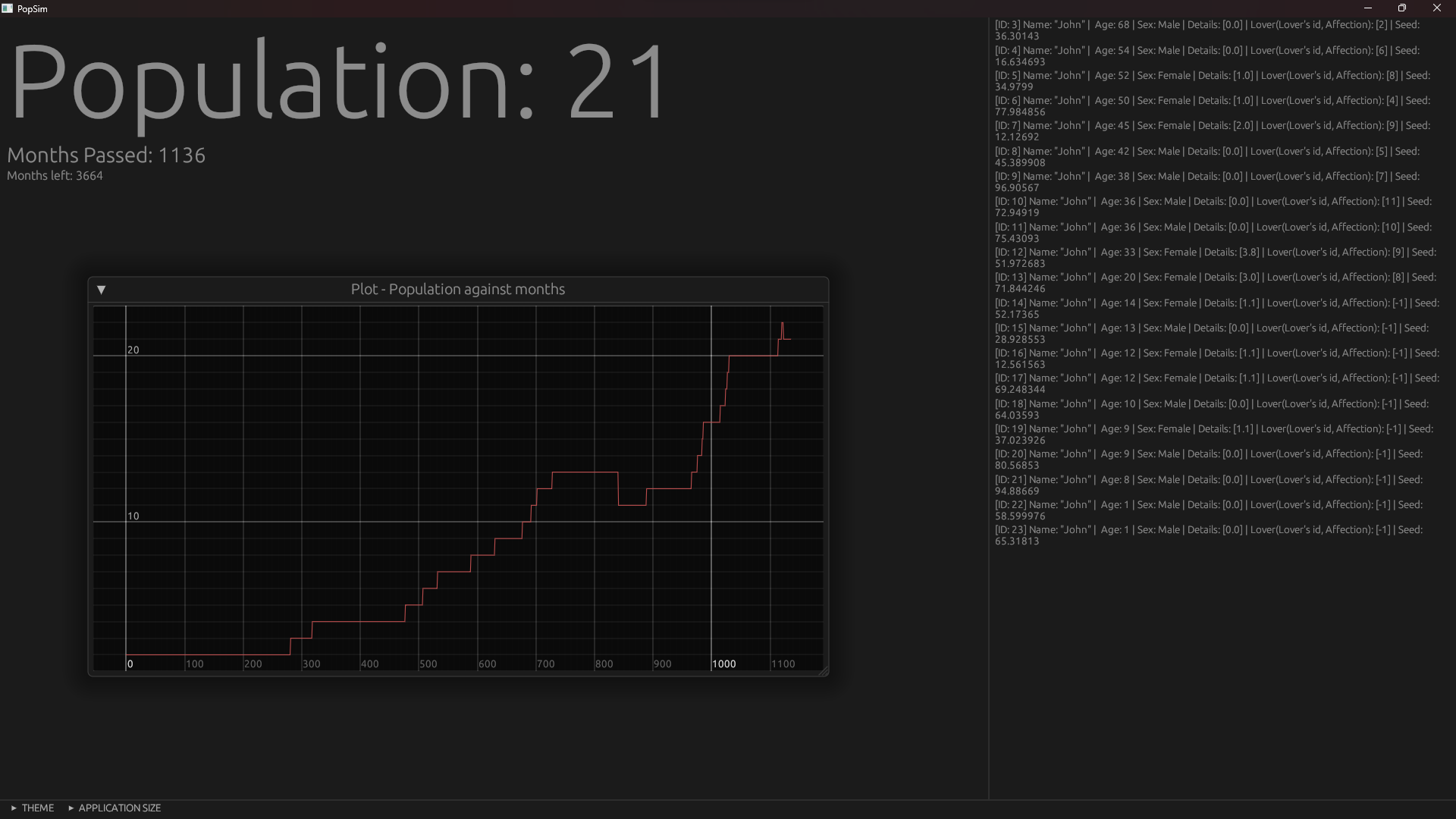
4.3. App progress

V0.1 was the first version of PopSimRust. It was very lacklustre and still relied partly on the test which I had created before starting the HPQ to test Rust with population simulators. It only displayed the population, months passed and months left. There were many issues with this version such as still using the console to display a table to list people and problems with resizing the window. Screenshot:

V0.2 was the 3rd total version (missed v0.125 as changes were minimal) and was already an improvement with the complete removal of the console in favour of a table which displays all people. This update also included a start screen to select the number of months to simulate, the bottom bar from the graphical plan and in that bar, an application size slider and finally a theme switcher. Screenshot:



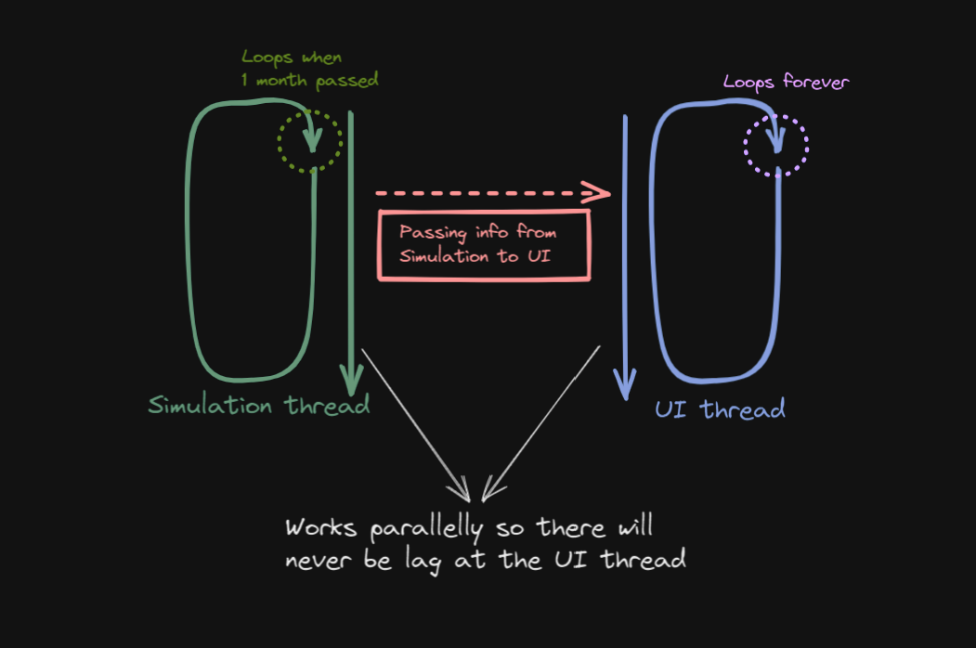
The next major version was V0.3 and this was where the graphical plan began to become a reality. I had to restructure some of my code to allow for a graph to exist but after I did, it was worth it. The graph provided a graphical representation of how large the population size was every month. Also, the addition of fertility now meant that the population would only create babies if the woman was ready to be pregnant with one. This single feature drastically increased the realism of PopSimRust and was another step to my final project plan. Screenshot:



Changes from V0.3 to V0.7 and the final version outcome (V0.7) can be seen in 5.

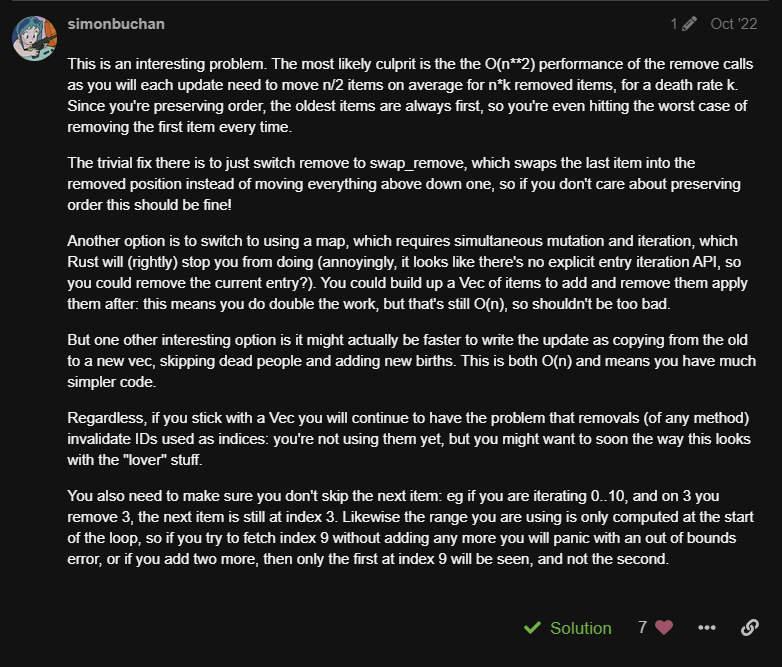
4.4. Problems…

4.4.1. Problems with multithreading

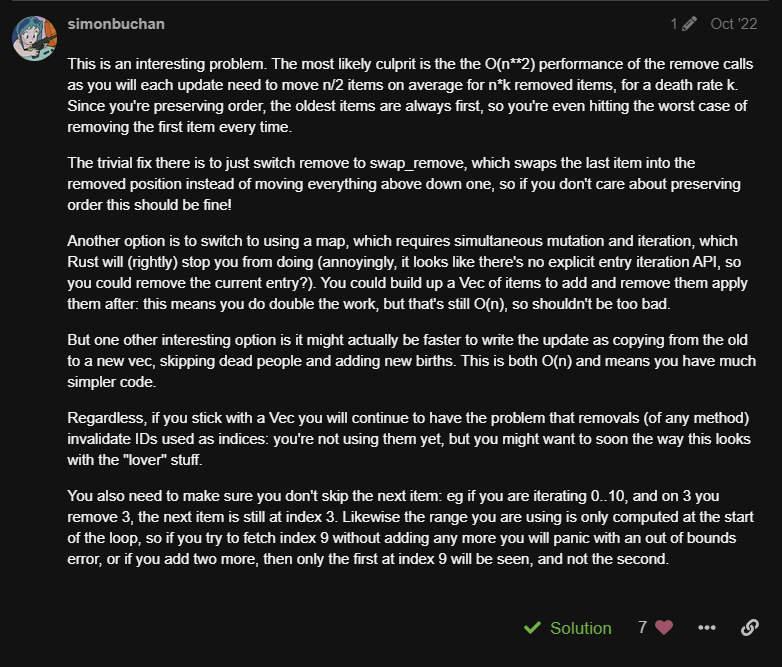
But this journey of progress was not all smooth sailing. After having issues with the UI performance at large population sizes, I decided to add one very large system that I mentioned earlier in the project plan. I would attempt to add multithreading to my project. The first time I tried; my code would simply error out when running most likely due to the lack of knowledge I still had about rust. The way I attempted to add multithreading was by having 2 threads: The “UI thread” (UT) and the “Simulation thread” (ST). These threads would work in parallel. The parallel nature of threads meant that whenever a month of the simulation went by, the ST would send its variables to the UT for it to display them. I was able to successfully decouple the simulation into 2 threads but couldn’t understand how to send information between them.

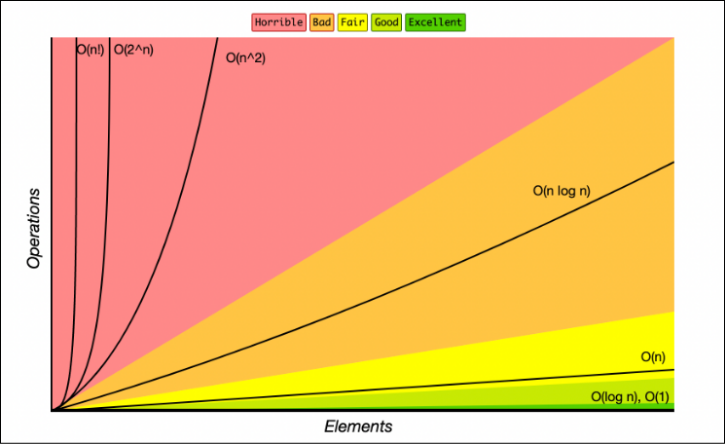
On the second attempt, I was able to add some debug information about some people as a test, which worked as intended. But when I tried to move the rest of the code to the system which I had created, it would crash the program but yield no errors. I did troubleshoot for a while but after some time, realized it would be too big of an undertaking to fix. I would instead focus on optimisation of the already existing simulation code. Here is a *YouTube video****:*** [**https://www.youtube.com/watch?v=Vsmzc51p4yE**](https://www.youtube.com/watch?v=Vsmzc51p4yE) showing a timelapse that I recorded of the troubleshooting processes that I went through but then still failing in the end. In the video you can see me, after failing to add Mutexes (A type of data which can be transferred between threads), consulting the Rust community for help. This happens at 2:18/4:10 in the video. This timelapse is about 6h of work in 4 minutes. I also created a *discussion thread on the Rust forums:* [***https://users.rust-lang.org/t/how-to-separate-the-update-of-the-egui-gui-and-background-program-processes/84944***](https://users.rust-lang.org/t/how-to-separate-the-update-of-the-egui-gui-and-background-program-processes/84944). But it was not very helpful because only one person replied.

4.4.2. Problems with general optimisations

But there was more than just one problem along the journey. The second was optimisation. Here is another link to a *Rust forums post****:*** [***https://users.rust-lang.org/t/population-sim-with-exponentially-bad-performance/82934***](https://users.rust-lang.org/t/population-sim-with-exponentially-bad-performance/82934). This time many people replied and they were all very helpful. Here is the suggestion which helped the most (some information is removed for readability):

USERNAME REDACTED

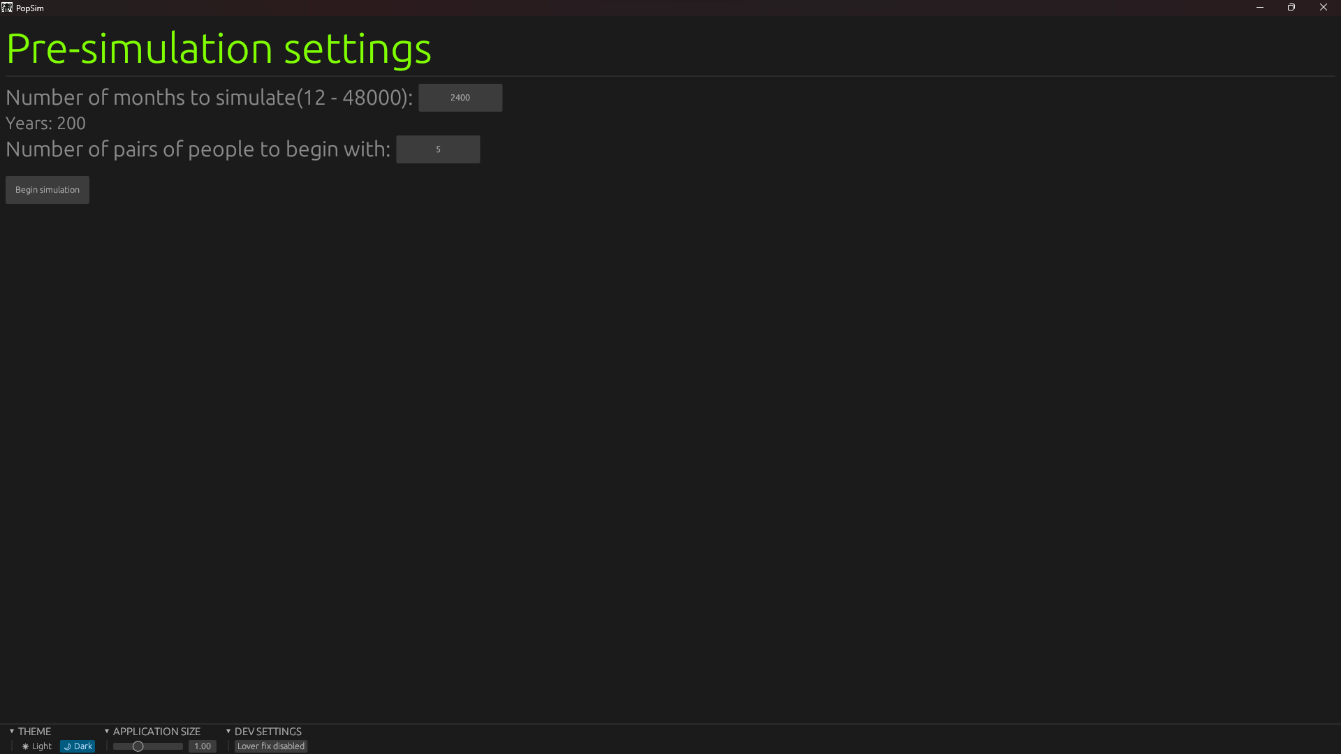


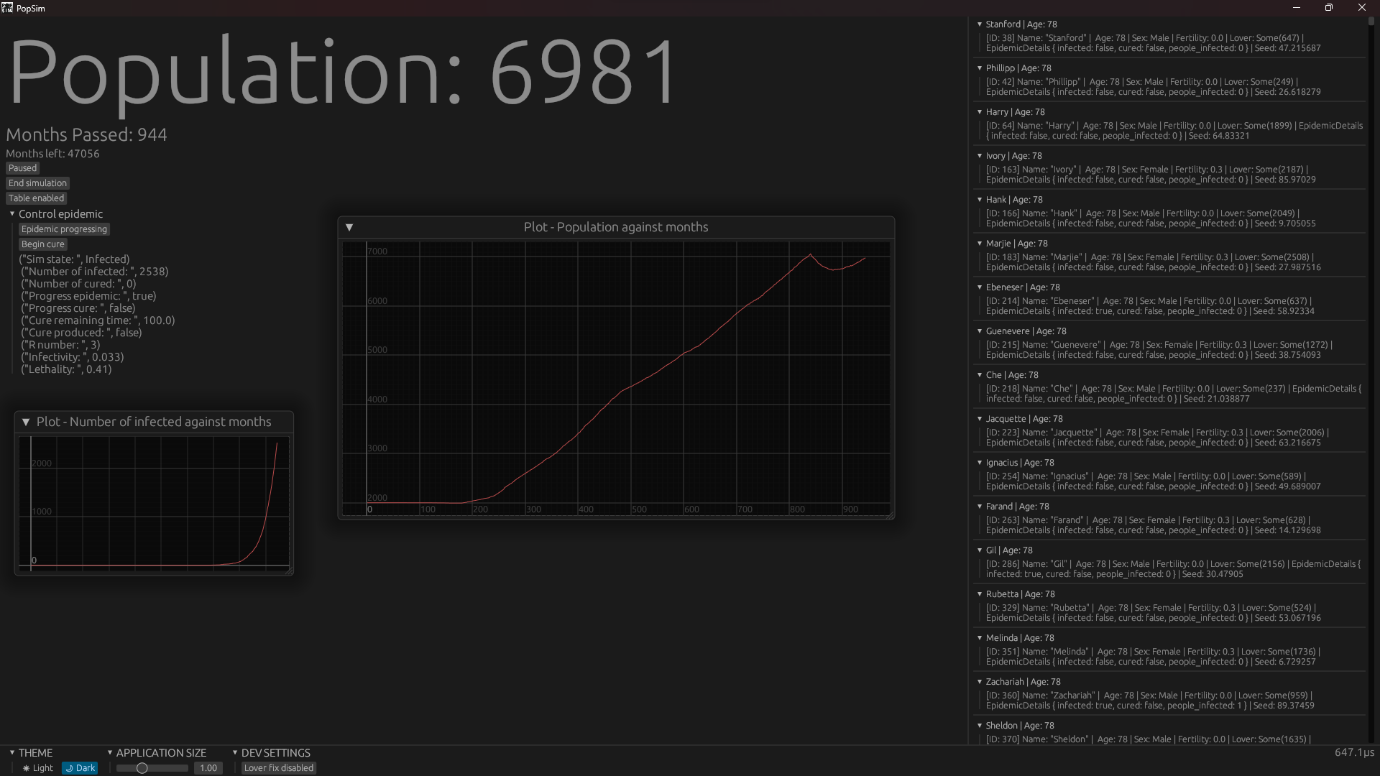
The box in red highlights the main culprit of my optimisation problems. It is an O(n\*\*2) time complexity (in this example: O being deletions, n being the number of people). This means that every single person that needs to be removed from the list of people takes a time of O(n\*\*2). This is bad because if removing 2 people takes 4secs then doing the same with 10 people would yield a removing time of 100secs which is immensely slow. I fixed this problem with a modified version of what is in the green box. It involves setting the age of a person to “None” and then removing all people all at once who have the age of “None” at the end of each month. This fixed my problem and now the number of people in my simulation would not affect performance (it was just O(n)).

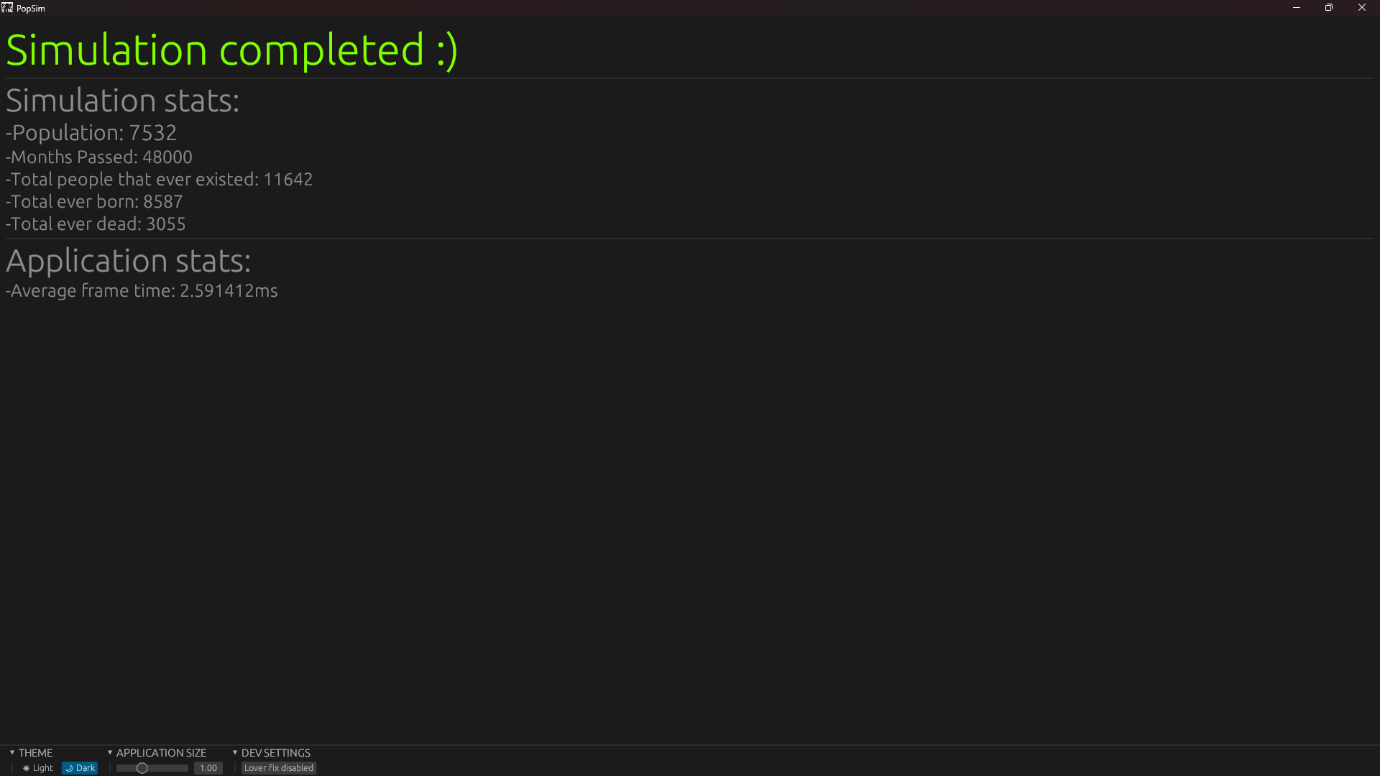
All progress from “Project activity log”:

|  |  |
| --- | --- |
| **Date** | **Comments** |
| ??/11/22 | I finished creating the base code needed for the final project. Note: this code will have to be modified nearly entirely to be an accurate model of how populations grow. The information needed will be derived from research. |
| 26/11/22 | I released version 0.1.25 which has a more polished base of a graphical user interface for me to work off of. |
| 14/12/22 | I released version 0.2 which has some major features which can be read here: [**https://github.com/NotAF0e/PopSimRust/releases/tag/GUI-alpha-v0.2**](https://github.com/NotAF0e/PopSimRust/releases/tag/GUI-alpha-v0.2)  Note: the application is not trusted by windows and there will be a message on windows pcs that states:  *Windows protected your PC*  *Microsoft Defender SmartScreen prevented an unrecognised app from starting. Running this app might put your PC at risk.*  *Click “show more options” then “run” if you would like to run the application. This applies to all versions as I would need to receive a trusted application certificate from Microsoft which is not feasible.* |
| 19/12/22 | I attempted to learn threading to increase the speed of my code by decoupling the rendering of the graphical user interface and simulation calculations. Threading is a very complex idea in every programming language. This led to me failing to refactor my code to use threading and leaving that for later. |
| 22/12/22 | Started some research for the bibliography. |
| 27/12/22 | More research for some missing info (for fertility). |
| 29/12/22 | I refactored the way my code handled sexes for a more streamlined programming workflow and added fertility to determine a more accurate representation of how many babies a woman would have on average. |
| 31/12/22 | I added a graph to show the population against months. It was incredibly troublesome to get working but now it is finished. |
| 05/01/23 | I changed the formatting of the releases section on my project repository. And I will take a second attempt to multithread my code. |
| 05/01/23 | I had help from many outside people on different forums and was able to compile the threaded code, but sadly the application crashes on start and there are no errors meaning it will be very difficult to find the issue. I may not even add threading to the final project as a colossal amount of time is probably needed to get it working. |
| 18/01/23 | There were many changes to the code structure and added some features: see [**https://github.com/NotAF0e/PopSimRust/blob/master/pop\_sim\_gui/patch\_list.md**](https://github.com/NotAF0e/PopSimRust/blob/master/pop_sim_gui/patch_list.md) for more detail. |
| 18/01/23 | I am planning to add an emigration simulation as my next feature. |
| 21/01/23 | Released version v0.3.5 with incredible optimisations and Frame counter. |
| 23/01/23 | Here is more info: [**https://github.com/NotAF0e/PopSimRust/releases/tag/GUI-alpha-v0.3.5**](https://github.com/NotAF0e/PopSimRust/releases/tag/GUI-alpha-v0.3.5) |
| n/a | I moved onto a different way of keeping progress as it would take less time and be briefer and to the point. This involves a markdown file (text file) in the code editor which I can edit while editing code. This also ensures that the progress kept is more accurate.  **The full list of every change done to code: (also available here:** [**https://github.com/NotAF0e/PopSimRust/blob/master/pop\_sim\_gui/patch\_list.md**](https://github.com/NotAF0e/PopSimRust/blob/master/pop_sim_gui/patch_list.md))  # Format of patch\_list.MD:  - type [Date] Feature/Fix info  ## Types:  - B: Bug fix (fixing an issue)  - F: Feature addition  - O: Other  These are changes regarding: pop\_sim\_gui!  ## Actual file:  - O [07/11/23] Added patch\_list.md to pop\_sim\_gui  - B [07/11/23] Person table will no longer show will a population of 0  - F [07/11/23] Added spacers into the table of people  - F [07/11/23] Fixed a bug with adding values to the population/months graph before removing dead people  - O [07/11/23] Added male names.txt and female\_names.txt to pop\_sim\_gui  - F [07/11/23] People are now given a random name from either male\_names.txt or female\_names.txt on birth dependent on their sex  - O [10/11/23] Formatted all of pop\_sim\_gui/main.rs  - O [17/01/23] Added to-do to pop\_sim\_gui/main.rs  - B [18/01/23] Fixed people not spawning in set pairs  - O [18/01/23] Changed code structure for easier programming  - F [18/01/23] Added temporary close table button as it was causing performance issues  - F [18/01/23] Table v2 is out! The table looks nicer, and info is easier to read  - F [18/01/23] Added a frame time counter to show simulation performance  - B [20/01/23] Fixed a strange table id problem  - F [21/01/23] Optimised the app a ton. It is now up to 1500x faster.  - F [21/01/23] Improved the frame time counter: it is now on the bottom bar. Also, adjusted the bottom bar text size to accommodate the frame counter  - F [25/01/23] Began adding the immigration system  - B [25/01/23] Fixed id issues  - F [27/01/23] Finished the immigration system  - F [28/01/23] Added an icon  - O [01/02/23] Streamlined the code a bit  - F [01/02/23] Added simulation end window  - F [07/02/23] Began adding the random death causes system  - O [07/02/23] Removed immigration system as it was causing many issues  - B [07/02/23] Finally fixed a lover check bug which has been plaguing the project for a month  - F [19/02/23] Added a button which disables the lover check fix as it is incredibly badly optimised  - O [19/02/23] Updated some project dependencies in Cargo.toml  - F [20/02/23] Implemented a better way to do the lover fixes - Now each 100 months  - F [20/02/23] Added better buttons which are easier to code with and show 2 different states  - F [23/02/23] Made progress on the new simulation start screen  - F [23/02/23] Improved the frame time calculator and added the average frame time to the end screen  - O [23/02/23] Changed some code structure which improved the performance of the simulation a bit  - O [23/02/23] Began adding the code structure for more sim stats  - O [25/02/23] Changed code structure further: the code is now split into 2 files  - O [25/02/23] Deprecated pop\_sim\_gui\_threaded  - O [27/02/23] Even more code structure changes  - F [28/02/23] Began adding the structure needed for the epidemic system  - F [28/02/23] Continued with the epidemic system with epidemic initialization and epidemic ending  - F [01/03/23] Fixed the readme a bit  - B [02/03/23] Fixed many issues which occurred with the epidemic system  - F [02/03/23] An epidemic can now be started and will automatically stop when nobody is infected  - F [06/03/23] Infection of the epidemic system is complete, now all that remains is a better death system and a curing system  - F [06/03/23] Added an end sim button  - O [06/03/23] Reformatted and restructured the code a bit for ease of programming  - F [08/03/23] I greatly improved the realism of the epidemic system and added UI with displays all epidemic stats as well as a graph  - O [11/03/23] Removed a non-needed crate from the project  - F [11/03/23] Increased the realism of epidemics and began adding curing  - B [11/03/23] Fixed an issue with the epidemic system which stopped the whole population from becoming infected after some died  - F [12/03/23] Further fixed the above issue with a rebalancing of values from minimal research  - F [13/03/23] Users can now customise the start of the simulation with different sliders  - B [15/03/23] Fixed a UI formatting issue  - F [15/03/23] Continued with the curing system  - F [15/03/23] Added most of the lethality system. Users can customize how lethal an epidemic is to a person.  - F [22/03/23] Completed the lethality system and improved the life expectancy weighted ages |

**5. The outcome**

Start screen:

Simulation screen:

End screen:

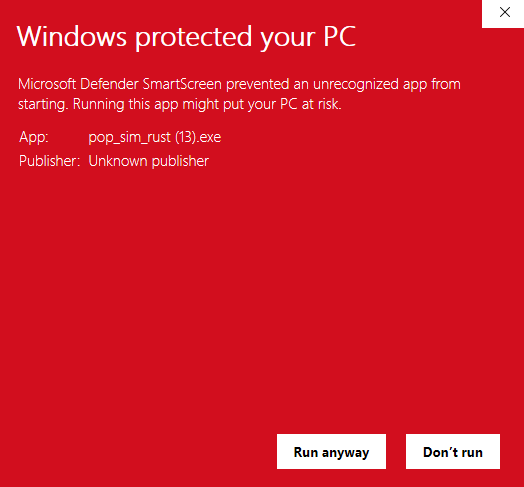
From these screenshots, many changes are evident for the final completed version (v0.7). The first is the great amount of polish added to the different screens: The start screen now hosts a bit more information and a converter from months to years. In the Simulation screen, many things have been polished up and improved, for example, the new formatting of the person table and the drop-down menus to be able to change what is displayed on the interface. One feature which helped a lot with the optimisation is the simulation update time at the bottom right-hand corner. This gave me an indication of if I had optimised the simulation or not, by showing the time between simulation updates. Finally, the end screen now displayed more information which could not be seen on the simulation screen to avoid clutter, such as total people and average frame time.

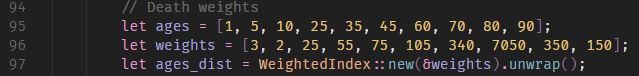
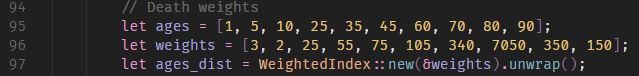
Another very evident change is the addition of an epidemic system (as of writing this, most of it is complete but curing is still not fully done). This system is a semi-realistic rendition of epidemics with various variables such as R number, infectivity and lethality. This system is not as fleshed out as I would have liked but I think it’s still a feature which gives the simulation some intractability and more depth. The way it works is quite simple: if a person is infected then the epidemic chooses another person randomly from the person list and if the variables above and some others are true then infects that person. This is not entirely realistic because proximity is not in consideration, nor is quarantine. I did not do as much research as for the main simulation due to time constraints (the repercussions of this are in 6.1.). But I think it is still quite convincing.

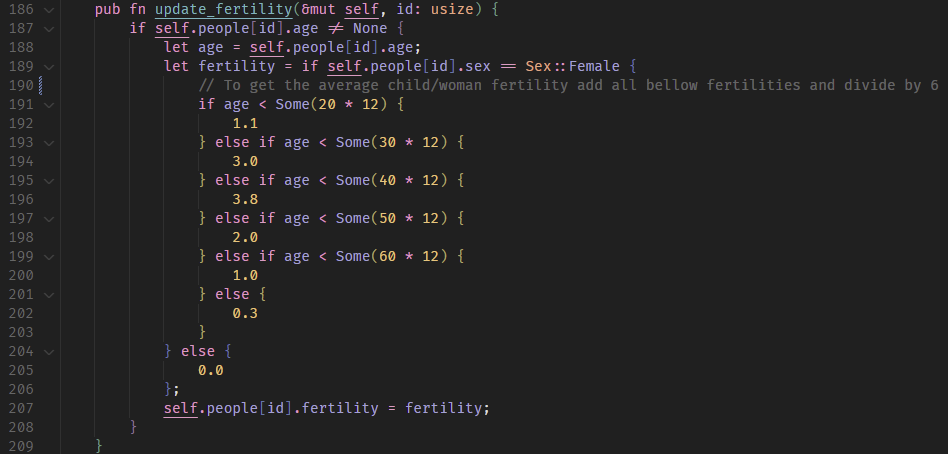
The final addition is not very noticeable but it provides more realism in sacrifice for a huge amount of performance loss every 100 months. Under the dev(eloper) settings there is a button “Lover fix”. This was added due to another huge issue which I encountered: A person loving their lover even if they had passed away. This issue only occurred sometimes during simulations, but in large population sizes, it is noticeable in the form of creating babies from nothing. I tried every possible way to fix this issue and I think I may have fixed it a few times but it always came back for some reason. This button enables a check every 100 months which removes lovers from a person's love list if they do not exist.

5.1. How to run the application

Running the app is quite simple:

1. Open the file called “PSR\_GUI” in that file there will be a folder called “PopSimRust”
2. Drag the “PopSimRust” folder anywhere then open the file called “psr\_gui”.
3. There may be a warning: If this error appears then press “show more options” then “run anyway”. This error shows up due to the app not having a *Windows-signed safety certificate:*[***https://learn.microsoft.com/en-us/windows/win32/win\_cert/certification-requirements-for-windows-desktop-apps***](https://learn.microsoft.com/en-us/windows/win32/win_cert/certification-requirements-for-windows-desktop-apps).
4. When in the application, set the settings to your liking and enjoy the simulated goodness of the populations 😊

5.2. Some code analysis

Above you can see 2 lists of ages and weights. The first age is correspondent to the first weight, the second age to the second weight, etc. The ages\_dist is a randomised distribution created from these 2 lists which is used to estimate, using randomness, when a person should die. The weights work by being a percentage of every weight added together. So, the first weight is a percentage of 3/8155 = 0.00036787247.

The function above first checks if the person's age is not equal to None, and if it is, it calculates their fertility value based on their sex and age. If the person is female, the function uses a series of conditions to determine their fertility value based on their age. The fertility values assigned to each age bracket are based on real-world data about how the fertility of women changes over time. For example, a woman below 20 years old has a fertility value of 1.1, while a woman between 30 and 39 years old has a fertility value of 3.8. If the person is not female, the fertility value is set to 0. Once the fertility value is determined, it is assigned to the person's fertility attribute.

**6. Evaluation of what I achieved and a reflection**

6.1. What did I learn and achieve?

In the end, I learned a lot about many different things, I gained this knowledge by succeeding and failing. In my opinion, the most important thing I learned was how to program in a new language, Rust. There is much for me to still learn from the language like being able to successfully implement multithreading and using a framework for the next project I make, such as *Bevy:*[**https://bevyengine.org/**](https://bevyengine.org/) which would increase the performance of the application and streamline adding new features. I think that having the knowledge that I have gained so early on in a programming career is great. Employers will more likely choose me for a job and there is a larger variety of programs I will be able to create. Or potentially even improve those that I have created already such as PopSimRust.

Another important truth which I learned from this HPQ, is that time management for a huge project like this one is very challenging. This led to many difficult decisions, for example giving up on multithreading and not being able to create a better and finished epidemic system. But this insight will be very useful to me in future projects as I will know that I may need to lower the size of a project to accompany a deadline.

Also, I learned about how populations grow and evolve with many details such as life expectancy, fertility, mortality, epidemics and many more. These have not only been very interesting to research but also gave me context for why the world population is so high in the present day. This has interested me for many years and the factors which led to this huge population are now uncovered for me. I could use this knowledge in the future to create an even more robust population simulator which has a population spread system and other more complex factors.

I would say that my plan at the start is very similar to the outcome that I have created. Here are all the things which I achieved on a programmatical level. Points in green were planned/not planned and added. Points in red are for those planned but not added:

* People are simulated at an individual level instead of incrementing a number for the population count. These simulated attributes are:
  + Age
  + Sex
  + Lovers
  + Fertility
  + Epidemic details
  + Random value seed
* A UI which allows users to watch the simulation grow and evolve in real-time with controls for the start settings and settings to change during the simulation:
  + Graphs
  + Some start settings
  + Real-time people table and other elements such as population counter
  + A dark and light theme switcher for viewing pleasure
  + Application size slider
* Multithreading to decouple the simulation performance from the UI performance
* Using real-world statistics to create a simulation more grounded in reality. Also implementing the way these values work in the real world in the simulation. This includes:
  + Epidemics:
    - Infection
    - R number
    - Lethality
    - Curing
  + Fertility
  + Age expectancy
  + A lover system to assign people to others that they will reproduce with.