# Analysis

## Background

GEMS Founders school is a private GEMS school teaching students from the ages of 4 to 17/18 (FS1 - Year 13). Courses offered include GCSEs and A-levels.

## Problem identification

(A-level Physics hosts one of the largest cohorts of any A-level in the school.)

While there already exist tools to simulate certain physical behavior, they often rely on the user knowing an exact scenario which they wish to test (usually by varying parameters). Because of this, online simulations often come in numbers to account for as many scenarios as possible, and by far the most popular collection of simulations is PhET.

PhET, and others, while incredible in their own right, fall short for when you wish to have more of a playground — a sandbox. A place to setup environments where you let everything follow from an initial condition.

As it stands, educators and students alike are left to probe their ideas only in thought experiments, unless they test it with a real apparatus. They can both attempt to calculate the states of a given scenario, but the more complicated said scenario, the more likely they are to be warded off by the idea of tedious calculations. This can leave uncertainty in both parties' minds about how the world works, and in the worst case, can leave incorrect ideas in the minds of educators that influence their pedagogy negatively.

The obvious solution, to some, would be to make use of Physics engines that are so often used in applications like games, where they are used to calculate information such as the position, velocity, and acceleration of a particle, among other things. These definitely exist, but because of the applications they were designed for, offer little use to those lacking experience with code. For this reason, a niche in the "market," wherein users with little to no programming knowledge can enlist computers to do the work for them and then visualize the results, has been left largely unfilled.

## Description of current systems

No current systems that achieve the same goals currently exist. The current way of teaching includes a combination of PhET (and similar online resources), the official exam board Textbook, and powerpoint presentations by the school's Physics department.

Presentations are how class lessons start, where the teacher goes over introductory concepts and lists questions on the following few slides to gauge how well the students have coped with them.

The notes students make on a given topic tend to be kept to themselves, but are seldom shared inside circles of friends.

After presentations come the textbook, which teachers often reference as a way to reinforce students' learning by assigning them the task of answering end of section questions and completing their notes. As an important note, this task is one which a sizable portion of students dismiss as unimportant and neglect to hand in.

Lastly, PhET simulations are used by some teachers to elicit educated guesses from students about a particular process or mechanism. These do not form a major component of the lesson plan, but nevertheless can allow some insight into the underlying principles of certain mechanisms and processes.

It may be of worthy mention that several students also study material from the mark schemes of earlier years' exams. As painstaking a process this can be, it also prepares the students in the best way to answer similar questions in their eventual exams.

## Research into physics engines

While mentioned previously, it would be useful to thoroughly explore what physics engines are and how they tend to be designed/work.

Physics engines are software used to approximate physical systems. The outputs of such engines are typically funnelled back into other parts of a/the program for another purpose, such as in computer graphics and video game development.

Physics engines come into two classes: real-time and high precision. The former, real-time physics engines, work with reduced accuracy and greater simplifications to deliver results to other parts of the system at an appropriate rate — their concept is similar to that of JIT (Just-in-time) compilers. The latter, high precision physics engines, contrast with real-time engines and work more similarly to standard compilers, where they perform all required calculations to higher precision (requiring more processing power) which is then passed off to other parts of the program; they do not work in real-time.

The oversimplified model of a Physics engine involves three core parts: user interaction; logic, calculation of state, and collision detection/collision response; and rendering.

'Calculation of state' is a simple process. It involves calculating data for the next frame using existing data and equations that govern how those data change, e.g., the speed of an object is 4 m/s initially, and changes at a rate of 2 m/s for every frame or second.

In either class of Physics engines, a lot of the computation required comes from the implementation of a collision detection (and a collision response) system. Collision detection is a classic problem in computational geometry where you attempt to detect when two or more objects intersect or "collide." There is no one-size-fits-all solution to the problem, and the implementation you choose and the optimizations you make will depend on your project. There generally exist two types of collision detection types: discrete and continuous. In discrete collision detections, you check every

frame for a collision and change the motion of an object based on conditionals. In continuous collision detections, you identify the frames in between which a collision occurs, and continuously interpolate motion to find the exact values of certain parameters at which it does.

After all calculations are said and done, the data is passed off to the rendering engine — that is often outsourced — where objects are rendered out to the user with the help of a multimedia framework. Both of these components are intricate and will likely not be bespoke for my project.

## Target demographic

The solution would be directly aimed to serve the needs of (GCSE and A-level) physics teachers and students.

## User needs

After conducting 3 interviews with 2 Physics teachers and a Physics student, these following 'wants' seem to be common:

- {insert}
- {more}

### Raw interviews below for reference:

**1. Mr. Sammi - teacher:**

| Question | Response |
|---|---|
| When looking for teaching resources, how often do you look for simulations? | |
| When you do look for simulations, is there anything you're looking for? | |
| Are there topics that you teach for which simulations are either incomplete, inefficient (not easily runnable on your laptop/PC), or simply non-existent? | |
| What is your programming background (concerning languages you may have some proficiency in)? | |
| Would simulations that require a more complicated interface (than those in PhET) — possibly having to deal with a few lines of code — be useful? If not, how much time would you be willing to learn the very basics of a specific language? | |
| Lastly, how would you like your simulation tool of choice to operate? On a website, as an app you can download, or as a simple command-line interface in which you run a command to run the simulation (this last one is simpler than it sounds)? | |

**2. Mr. Mursal - teacher:**

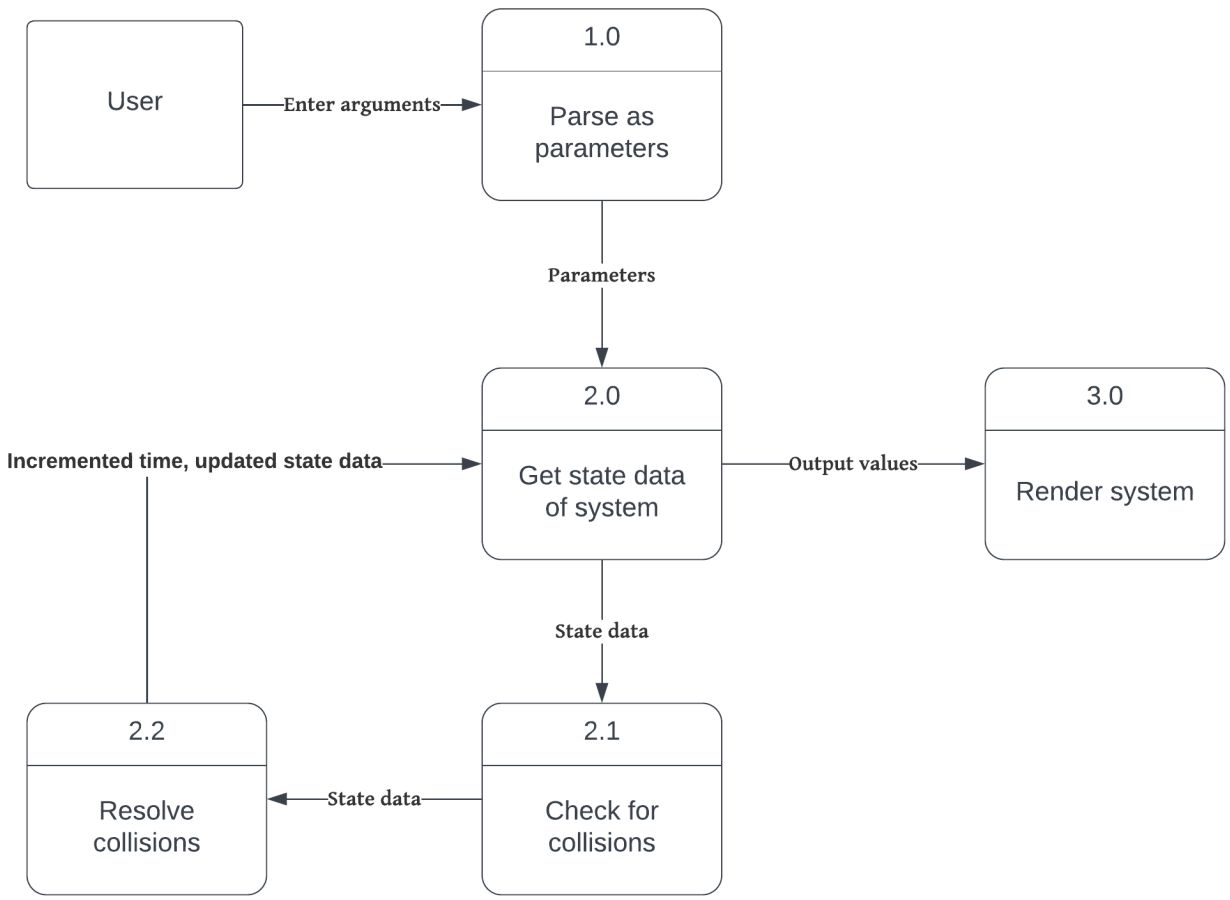
| Question | Response |
|---|---|
| When looking for teaching resources, how often do you look for simulations? | |
| When you do look for simulations, is there anything you're looking for? | |
| Are there topics that you teach for which simulations are either incomplete, inefficient (not easily runnable on your laptop/PC), or simply non-existent? | |
| What is your programming background (concerning languages you may have some proficiency in)? | |
| Would simulations that require a more complicated interface (than those in PhET) — possibly having to deal with a few lines of code — be useful? If not, how much time would you be willing to learn the very basics of a specific language? | |
| Lastly, how would you like your simulation tool of choice to operate? On a website, as an app you can download, or as a simple command-line interface in which you run a command to run the simulation (this last one is simpler than it sounds)? | |

**3. Siddhant - student:**

| Question | Response |
|---|---|
| When looking for learning resources, how often do you look for simulations? | I do like the idea of simulations, I don't use it extrememly often but I think it's extremely handy in concepts such as circuits, where the simulations helps you visualise the flow of electrons in a circuit |
| When you do look for simulations, is there anything you're looking for (convenience or otherwise)? | Ease of use for sure. I feel like a lot of simulations have a lot of fancy buttons and features that let you do all sorts of things which sometimes I do find overwhelming, so a simple UI that has the core features I want to simulate available is what I look for |
| Are there topics that you're taught for which simulations are either incomplete, inefficient (not easily runnable on your laptop/PC), or simply non-existent? | Nothing comes to mind right now |
| What is your programming background (concerning languages you may have some proficiency in)? | I've been doing python for the last 3.5 years and have completed a beginner course in C++ |

| Question | Response |
| --- | --- |
| Would simulations that require a more complicated interface (than those in PhET) — possibly having to deal with a few lines of code — be useful? If not, how much time would you be willing to learn the very basics of a specific language? | Like I said, I do prefer simple interfaces but having a complicated simulator is unquestionably useful especially for very curious students. Personally, I do have quite a cramped schedule right now due to it being the last year in high school, but I wouldn't mind spending 2 hours on weekends learning the basics |
| Lastly, how would you like your simulation tool of choice to operate? On a website, as an app you can download, or as a simple command-line interface in which you run a command to run the simulation (this last one is simpler than it sounds)? | Command line interface for sure. Downloading apps and looking for websites that do what you want is time consuming. If I can run a simulator quickly with just a few command lines, it would be very handy |

## Data flow diagram




## Project aims

- [ ] The program should be entirely functional as a lightweight app

- [ ] Allow the user to specify information about rigid bodies when prompted
- [ ] Update the user on progress as the final output is computed

- [ ] The program should be able to save the video output locally.
  - [ ] Let the user choose what file format they wish to save it as.
  - [ ] Add compression to reduce file size if the user so wishes.

- [ ] Allow the user to interact with the simulation in real-time
  - [ ] Let the user attach a spring to an object to propel it

- [ ] Show the user information about the simulated system
  - [ ] Velocity and acceleration information for the user
  - [ ] Display the time/frames passed

## Language considerations

- **Python**: being the language taught to me in my GCSEs and now A-levels, this was a natural consideration. Having syntax easy to interpret, logic easy to understand, and an extensive repository of external libraries makes Python a desirable choice for many applications, but not here. This is because physics engines need to run fast, something Python is not especially known for (some sources cite the language as being over 80 times slower than C). Due to this simple drawback, I decided against using Python.

- **C++**: Most physics engines today are written in C++ for its versatility and speed. This means that a plethora of resources are already available if you tried to write a Physics engine of your own. Despite this, however, I was swayed away from the language because of its syntax, difficulty to learn, and overall unpredictable, clunky behavior.

- **Rust**: Unlike Python, Rust is comparable to C in terms of speed (those same sources cite the language as being about 1.1x slow than C). This already makes it viable for the project. However, a feature of the language thanks to its design philosophy is memory safety. It is impossible to represent undefined behavior in Rust (without using an explicit unsafe block). This in general makes Rust code safer than the infamously hard-to-work-with C/C++ code, which is especially useful for a Physics engine as the program will be calculation-dense. In addition, Rust is sufficiently high-level to pivot to from a language like Python; it's syntax can be Pythonic. This is in sharp contrast to low-level langauges like C++ — which it does take some syntax from — where learning the language would act as a larger obstacle. Importantly, another convenience the design philosophy of Rust guarantees is backwards compatability: even as time passes, newer versions of Rust will still be able to compile older code, increasing the chosen solution's longevity.

## Solution

The chosen solution will consist of a lightweight, simple GUI built with Rust that provides a user front-end to interact with the Physics engine also implemented using Rust.

The final product itself will feature a GUI that allows the user to specify values of parameters to begin the simulation. As the engine performs the bulk of the calculations, a progress bar may be used to indicate the estimated work remaining. Once this progress bar has been filled, and the calculations are complete, a multimedia framework, such as ffmpeg, would be used to generate a video output in the format the user also selected beforehand.

While the video is playing back, state information such as objects' velocities may also be explicitly labelled to satiate the user's curiosity, and it may also prove useful for debugging purposes. Influencing those labels, real-time interactivity with the user with components such as springs would allow the user to experiment with the simulation in real-time.

It is clear to me that Rust is a better candidate for my needs, supplanting C++ with its memory-safety, backwards compatability, and shallower learning curve. There already exist plug-and-play GUI libraries that are simple to incorporate into my project. As for the math, Rust and C++ could be used interchangeably for the majority of the computation, as both languages allow for the creation of classes (more loosely in Rust) and functions. Considering both points of interest here, Rust is my language of choice.