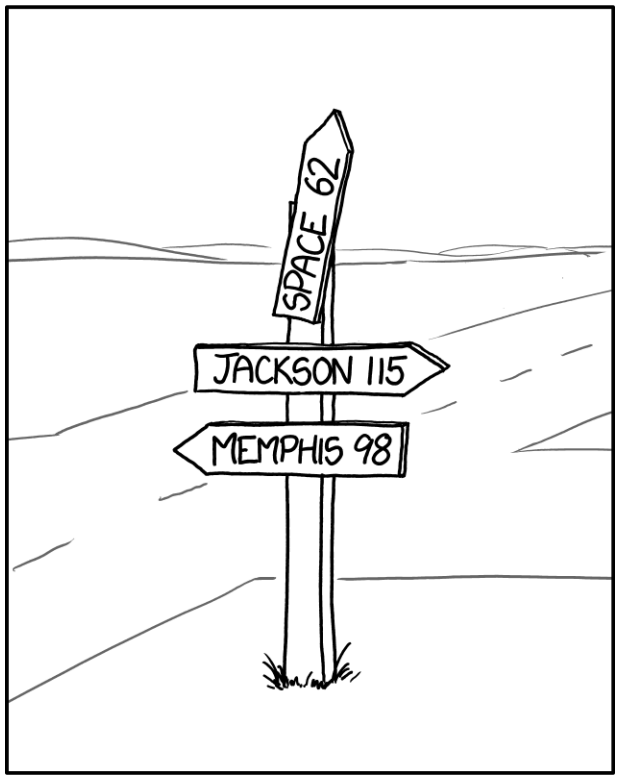
Report

Kurt Russel's Teapot "StarDwarf"

Elie "Dr. Rodney McKay" Halimi
Samuel "Chaussure" Corno
Ngoc Phuong Anh Duong

May 23rd 2018



Contents

1 Our project

This project is an orbital simulator called StarDwarf. This software thus model trajectories of moving objects depending on one another.
For the third defense, we had to add a three-dimensional representation of space, and thus let the user run simulations in three-dimensions.
We will detail in this report, the objectives we have attained and the parts where we have failed.

2 Progress on the project

2.1 Reminder of the schedule

Tasks/Person	**Elie**	**Samuel**	**Anh**
Physics		●	❍
Gui	●	●	●
Collision handling		❍	●
saving/loading systems	●	❍	
installation	❍	●	
soundtrack	●		
Website	❍		●

● : Holder of the task
❍ : Substitute of the task

2.2 Planning

Tasks/Period	**1st presentation**	**2nd presentation**	**3rd presentation**
Physics	✶✶	✶✶✶	
Gui	✶	✶✶	✶✶✶
Collision handling	✶	✶✶✶	
Saving/loading systems	✶	✶✶	✶✶✶
installation			✶✶✶
soundtrack		✶	✶✶✶
Website	✶	✶✶	✶✶✶

Completion of the task : ✶ : started ✶✶ : advanced ✶✶✶ : finished

2.3 GUI

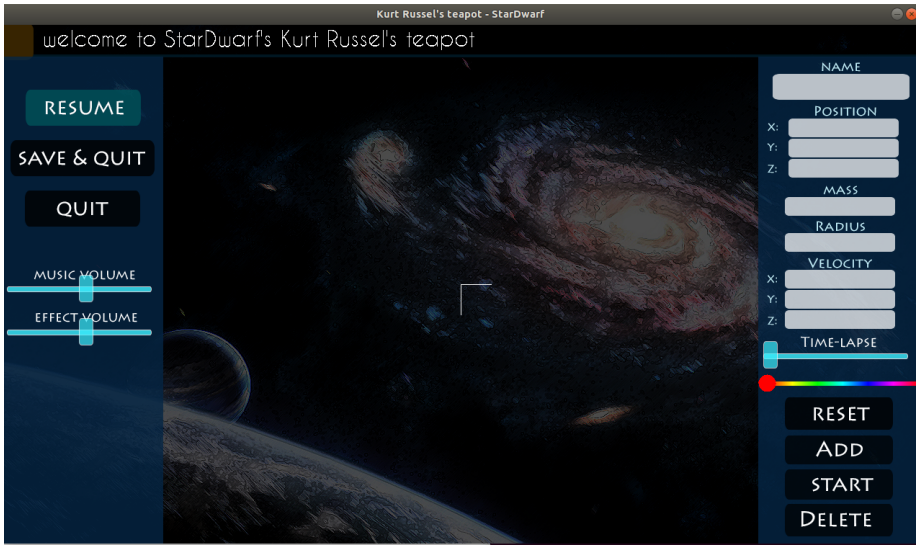
2.3.1 Hashtable

As there were so many buttons and windows, we implemented the hashtable structure to store our buttons, windows, sliders, texts and drawing conditions as dictionaries, which avoided global variables.

2.3.2 Main menu

The item menu was created and fixed in main menu to let the user be able to interact with the system through some tools and variables by modifying the items such as name, mass, radius, position, velocity, time-lapse slider, color-picker palette, "reset", "add", "start" and "delete". In other words, it not only shows the variables of the items but also takes the inputs from the user and add them to the items. The fact that user can add inputs in this menu is allowed by clicking on the specific text box and then typing the relating characters.

The pause menu's graphical design also saw a significant change compared to the second defense. Not only that, it is now located at the left of the main menu and involves two more volume sliders.



Main menu's graphical version

In terms of the item's variables, the struct "text" was created in order to hold and to display the information of the item's variable. It also allows the user to change the variable of the item through these input spaces.

```c
struct text{
  char *text;
  int nbchar;
  int active;
  int item;
};
```

Struct text

The "text" pointer points to the string of the information. The "nbchar" counts the number of characters (char) in the text in order to verify whether the text reaches its limit or not. The "active" condition is to know if the user is modifying this text box. The "item" condition check as it is the text that use for user input and output.

We also created some new "struct" such as a slider, which is used for the "time-lapse" function, to modify the volume and palette, which are used for the color picker.

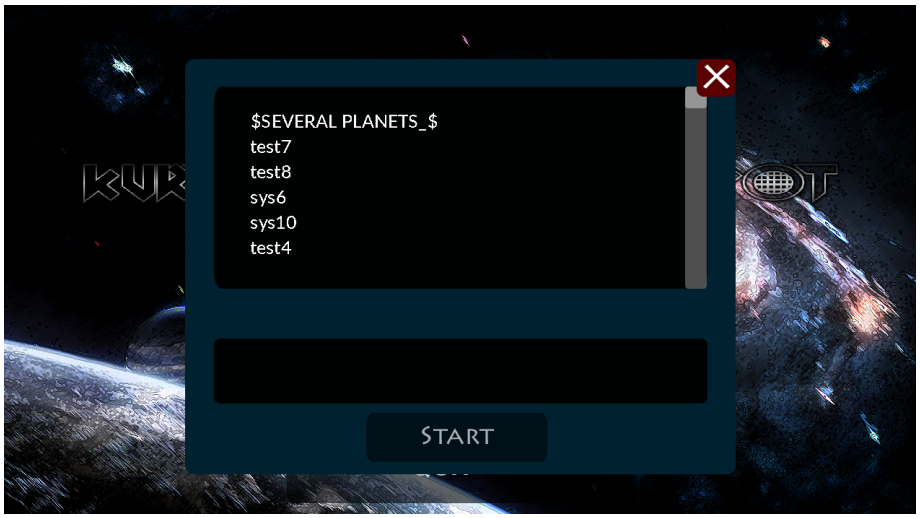
```c
struct slider{

  int visible;
  int event;
  int active;
  int prelight;

  SDL_Rect rect_bar;
  SDL_Rect rect_token;

  int mousepos;
  int mousedown;
  int horizontal;
  int maxlength;
  int curlength;
  void *maxvalue;
  void *minvalue;

  struct window *window;
  struct image *bar;
  struct image *unselected;
  struct image *selected;
};
```

```c
struct palette{
  int visible;
  int event;
  int active;
  int r;//radius of the picker

  SDL_Color *color;
  int pos;

  SDL_Rect rect_palette;
  SDL_Rect rect_picker;

  int mousepos;
  int mousedown;

  struct window *window;

};
```

Struct slider and palette

2.3.3 Load menu

We used the struct slider to implement our scrollbar, showing as many files as the directory contains.

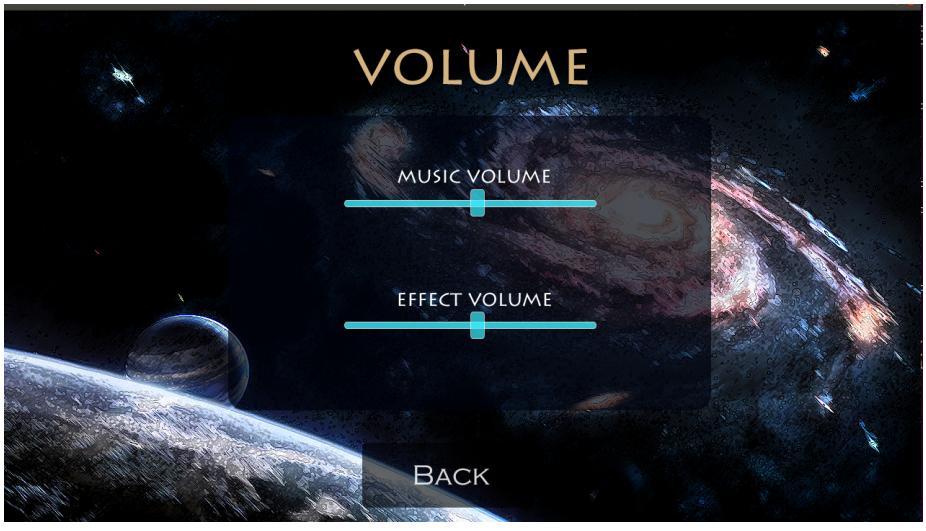


This is our load menu.

The load menu is mainly about the streaming directory by using opendir(3) and readdir(3), and applying the list structure to store and find all of the names.

2.3.4 Volume menu

In this menu, we used the slider structure to create music volume slider and effect volume slider.



This is our volume menu.

2.4 3D and Camera

During the time between the second and the final defense, our software went from a 2D software to a 3D software. This was achieved using vector spaces, orthogonality, and the fact that our simulated items could be considered as perfect round spheres.

2.4.1 The Camera and projection structures

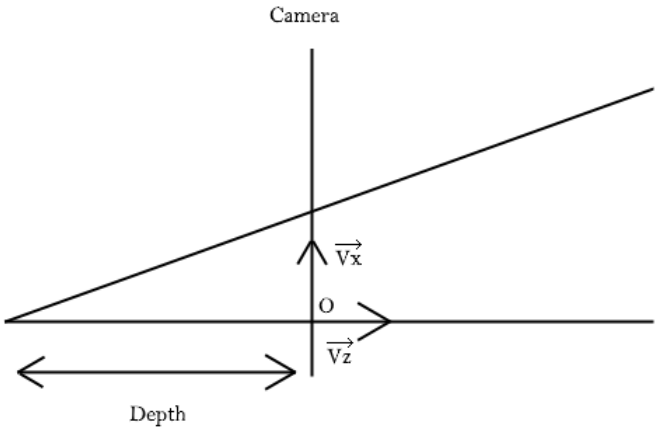
Two new structures were created to make the 3D possible: The camera and the projection.

```
struct projection
{
        struct item *item;
        struct vector position;
        float size;
        float distance;
        int shown;
        struct list next;
        struct list prev;
};
```

```
struct camera
{
        struct vector position;
        struct vector origin;
        struct vector Vx;
        struct vector Vy;
        float depth;
        size_t nb_proj;
        struct list projections;
        float center_X;
        float center_Y;
        float mouse_x, mouse_y;
        int event_type;
};
```

Struct projection and camera

The camera represents where the user is looking from. It contains its position, the position of the point around which it will rotate, called the origin point, an inclusive list of projections, and a depth representing the focal point of the camera. It also contains two vectors, $\vec{Vx}$ and $\vec{Vy}$, which are orthonormal basis for the plane the camera is on. They are relative to the position of the camera and they thus always consider it to be the reference of the space they're in. We can thus model the camera as being a vector space, which, if considered in only two dimensions, could be represented like the following figure.



A 2D representation (thus ignoring the y axis) of the Camera

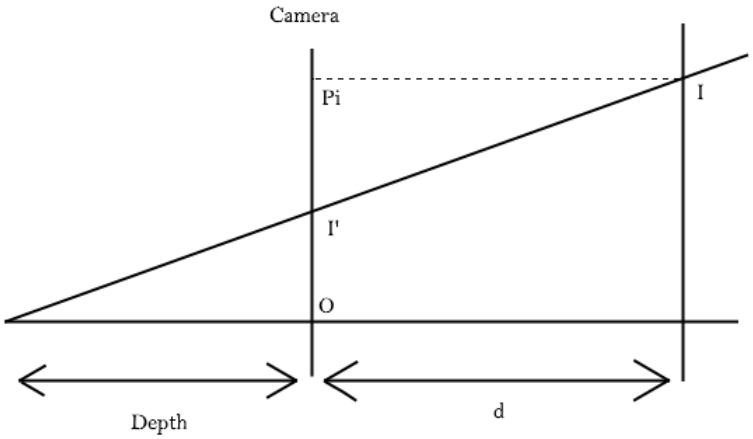
Note: Even though $\vec{V}z$ is not stored, it can be computed from the position of the camera and the position of the origin.

Each projection represents the 2D image corresponding to the 3D object it represents. It thus contains its position, in 2D, its size and a pointer to its item. At each frame, every projection is updated to reflect the new point of view of the items the user gets, either because the camera moved, or because the items moved.

When the software needs to draw an item, it thus only has to update the projection and to display the position and the size given by the projection structure.

2.4.2 Computing a projections

Computing a projection is done through a mix of Thales's Theorem and orthogonal projections. Using the previous model for the camera, we can now model the projection I' of the item I on the camera by the following figure :



A 2D representation (thus ignoring the y axis) of the projection of the item I on the Camera

Since the camera is a vector space with an orthonormal basis, we can easily compute the orthogonal projection Pi of I on the camera. Then, using Thales's Theorem, we get:

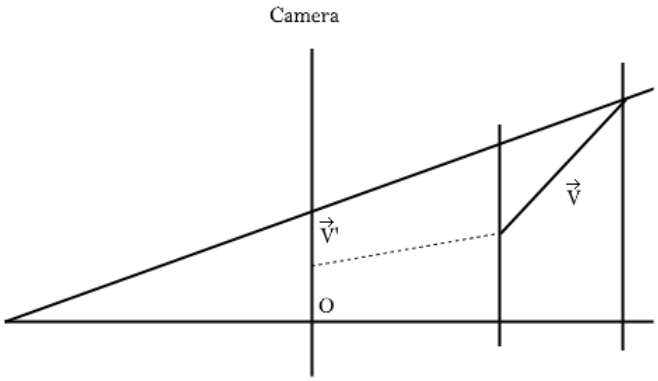
$$Ratio = \frac{Depth}{Depth + d} = \frac{OI'}{OPi} \iff OI' = OPi * Ratio$$

Note: The value d is obtained by computing the magnitude of the orthogonal projection of I on $\vec{V}z$.

This method only gives the position of the projection. To compute its size (the projection's), we define it as being the size of the item multiplied by the same ratio.

2.4.3 Displaying the Vectors

The software will display the velocity of every item as well as the basis of the space at the origin point. To compute their positions, a similar methods as for the item is used, but the starting point and the ending point of the vector both use different ratios, as shown below:

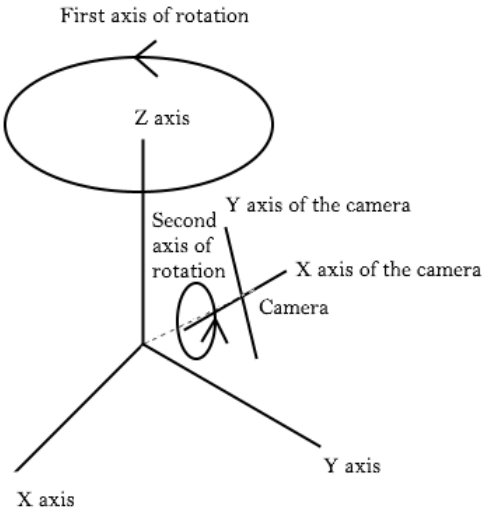


A 2D representation (thus ignoring the y axis) of the projection of a vector $\vec{V}$ on the camera

As it can be seen, to compute the projection of $\vec{V}$ on the camera, we have to apply the previously described method to both starting and ending points of the vector.

2.4.4 Moving the camera

Translation of the camera is simply done by adding $k1\vec{V}x$ and $k2\vec{V}y$ to both positions of the camera and the origin, with k1 and k2 two reals decided by how much it is needed to move the camera. This allows the user to always move the camera in a parallel way to what he sees. The hard part is rotating the camera. Since the user controls the camera through the mouse, which only has two axes, a rotation on a three-dimensional space cannot be achieved by a simple multiplication of 3 rotation matrices. Instead, we use what is called a "dolly camera", and make the camera move only along two axes: One global, depending on the basis of the space the camera is in, and one local, depending on the camera's own basis. We thus get the following:



A 3D representation of the rotation of the camera

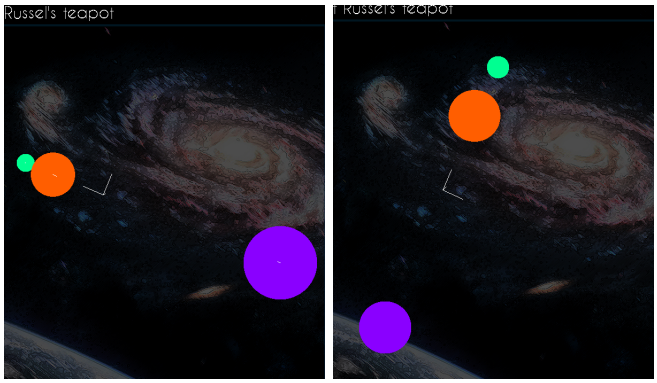
This method has the advantage of both allowing a complete control of the camera with only two axes, and also the convenience of never rotating on a third axis over time and manipulation, because $< \vec{Vz}, \vec{Camera}.Vx >= 0$ is always verified. We achieve such a rotation by using a simple rotation matrix for the Z axis, and a simple rotation matrix to which we apply basis changing matrices for the X axis.
Since both our basis are orthonormal bases, we can use easy to compute matrices for the basis changing matrices. But since our software will perform approximations on operations on floats, the vectors will stop being orthonormal, even by a slight amount, and thus the matrices will amplify this effect. To avoid that problem, we apply Gramm-Schmidt algorithm before each rotation.

2.4.5 Interacting with items through the camera

The software allows the user to interact with items directly displayed on screen: the user can select, drag and drop items, and when the user wants to create a new item, he has to click on the position he wants. This is done by reversing the method used for the projection of items.
The hard part is knowing on which plan we want to move or create our items. For the creation of items, we use the plan, which is parallel to the camera that goes through the origin vector, and for the movement of the item, we use the plan, which is parallel to the camera that goes through the actual position of the item.

2.4.6 Result

This result in making the software fully functional in 3D: the user can observe and interact with all items through the three dimensions and can move the camera freely, with rotations, translations, and zooms. Below are two screenshots of the same items at different angles:



Struct projection and camera

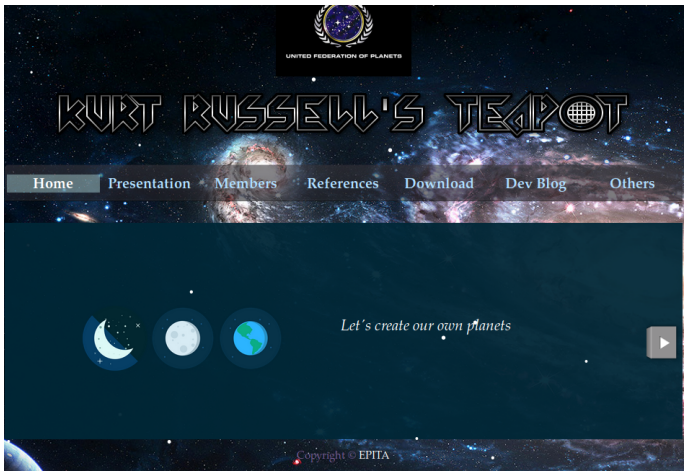
This part was interesting to make because it was a direct application of several of our math courses (Linear Algebra, Orthogonality ...) and gives the results in a concrete, visible, graphical way.

2.5 Website

In this task, we only needed to update all of the information and changes during the progress such as "devblog", "references" and "presentation".

This is our devblog page.

We also added a short introduction for our beautiful program.



This is our home page.

2.6 Music

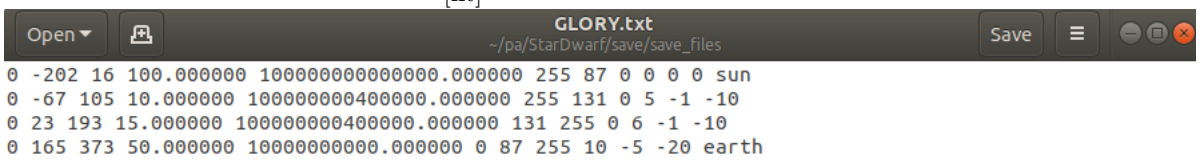
In this task, as said in the previous report and defense, we couldn't find any music that would be suited for our software for the second defense.
However, as we weren't all focused this time on the GUI, we were able to find some suitable sounds and background musics, which have been added to our software. Indeed, we wanted some sounds and musics that would not require any rights and that would also be free. We finally found on the website opengameart.org some musics and sounds effects, which are in the public domain.
We then proceeded in adding these sounds into our software using SDL_Mixer as well as adding a slider in order to modify the sound volume of the background music and one slider for the sound effects.

2.7 save and load system

In this task, the goal was to let the user save a system in order for him to pick up where he left off and run his simulations again. For the first defense, we actually had a loading system, which only took care of the position and the radius of a planet.
For the third defense, we decided to have a working saving and loading system, which would take care of all the important variables such as the position (in three dimensions (x, y ,z in Cartesian coordinates), the size, the mass, the color of a planet (r, g and b) as well as the velocity vector (still in Cartesian coordinates). Indeed, the saving system takes a name as well as the current system as argument and creates a file with the given name in /save/save_files/ under "name.txt". This file contains on each line an object (whether it is a sun or a planet (this depends on the mass and its radius)). Each line contains in order: the position, the size, the mass, the color of the planet and the velocity

vector as you can see on the picture.

[ht]



picture showing the inside of a file representing a saved system

As you can see in the picture below, we have modified our code so as to make it more efficient, more readable and easier to maintain.

```
int color0 = 0;
int color1 = 0;
int color2 = 0;
int velocity0 = 0;
int velocity1 = 0;
int velocity2 = 0;
char *label = "";

while(str != NULL)
{
        switch(counter)
        {
                case 1:
                        x = (float) atoi(str);
                        val[0] = x;
                        counter++;
                        break;
                case 2:
                        y = (float) atoi(str);
                        val[1] = y;
                        counter++;
                        break;
                case 3:
                        z = (float) atoi(str);
                        val[2] = z;
                        counter++;
                        break;
                case 4:
                        size = (double) atol(str);
                        counter++;
                        break;
                case 5:
                        mass = (double) atol(str);
                        counter++;
                        break;
                case 6:
                        color0 = atoi(str);
                        counter++;
                        break;
                case 7:
                        color1 = atoi(str);
                        counter++;
                        break;
                case 8:
                        color2 = atoi(str);
                        counter++;
                        break;
                case 9:
                        velocity0 = atoi(str);
                        counter++;
                        break;
                case 10:
                        velocity1 = atoi(str);
                        counter++;
                        break;
                case 11:
                        velocity2 = atoi(str);
                        counter++;
                        break;

                case 12:
                        label = str;
                        break;
                default:
                        break;
        }

        str = strtok(NULL, " ");
```

[ht] picture showing the inside of a file representing a saved system

3 What could be done

As all the main structures of the software have already been implemented, all upgrades would not be essential to our software as it works on its own. However, improvements can be made on the Physic part as we are not using for instance relativity in our software (whether special relativity or general relativity) or even add a way to distinguish correctly a sun from a planet by adding components (atoms) that compose a stellar object, or even add other stellar object such as black holes...
What we could also do would be to add on the website all the different save files in order for any user on Internet to run simulations from other people. Furthermore, this would be a nice way to share simulations. This might require the use of a database (likely MySQL database).
Any other improvements could be done to make the software more ergonomic, easier for the user to run simulations, save systems... An improvement in the Physic part would require to add variables to the save and load functions as they might be an essential part of the simulation. For instance, if we were to add the electromagnetic force in Cartesian coordinates, we would essentially need to add 3 variables in the load and in the save functions. Fortunately, our save and load system is fairly easy to maintain and would not require much in order to add such improvement into our software.
Although we have given up on adding user-created forces as it would be a hassle adding them to the GUI using SDL, the code implementing them is still in our (engine) libraries, and can thus be used to implement these forces based.
Last but not least, we could try again to add Mandelstam's variables in order to simulate the collisions of particles and generalize that to our stellar objects (thus handling the collision (fusion or small asteroids) and even add in that case some animations when there happens to be a collision).

4 Conclusion

We are proud to have been part of this team as we have all learned about project organization, knowledge on physics especially related to astrophysics. Furthermore, we have gained an incredible amount of experience in programming especially regarding the use of the SDL graphic library instead of GTK3. This project (StarDwarf) has held up to its promises as it can run simulations on trajectories of planets depending on some gravity fields as well as some velocity vectors. It also can simulate some kind of collisions between different stellar objects. Moreover, the user can load and save simulations as well as listen to a background music.
We have had fun creating this software and think that it is fun to play with. We did not have much problems organizing this projects as well as problems interacting with each other. We hope that you enjoy our software as much as we have had while creating it.

5 Appendix

5.1 Links

To create our simulator, we used various softwares, websites until now :

- **Git**: it is an indispensable software for the project since it allows to share files and codes. We use the website Github reachable at the address https://github.com to host our git repository.
- **Overleaf**: it is the software that we will use in order to write our reports in LaTeX.
- **Discord**: an "All-in-one voice and text chat" software to ease our communication.
- **SDL**: A library allowing us to create a graphical interface. (SDL2)
- **Trello**: A collaboration tool that organizes projects into boards.
- **XKCD**: A website containing small comics that we use for our pictures in all of our reports.

Our website is :

- https://eliehalimi.github.io/KurtRusselsteapot/