3D Solar System -"Space Travel"

Final report

Aleksandra Musial
Zami Nizam

Goldsmiths University of London
Creative Computing Creative - Projects

May 2019

CONTENT

3D Solar System -"Space Travel"

To access our project:

#1 Download our project repository from **GitLab**

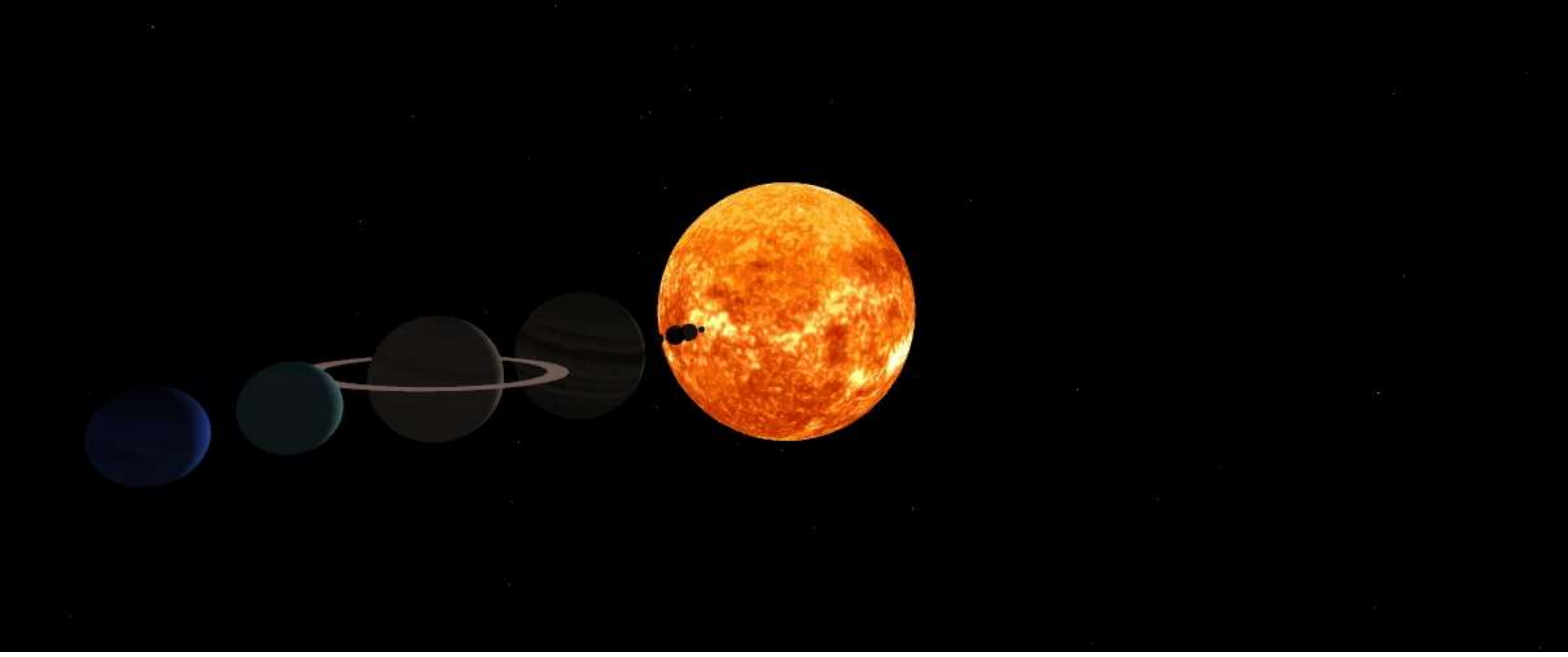
#2 The project uses OpenFrameworks library, so create the new project with project generator.

#3 Include ofxGui and ofxAssimpModelLoader addons.

#4 Move all the '*.h*' and '*.cpp*' files inside the src file of newly project.

#5 Move all the pictures and 3D model inside bin/data folder.

#6 It might take a moment to run the project.



#1 PROJECT DESCRIPTION

At first, we named our project “Space Travel”, but we have realized we will not be able to write the code for the whole galaxy. That is why we have decided to make our project only about Solar System, so we have changed the name to “3D Solar System – Space Travel”.

Our project simulates a 3D Solar System, which purpose is to educate people more about the Solar System. From the feedbacks we received so far on our final version, seems like we have been able to impress our potential users. The reason we decided to go for something like this is because of the various documentaries containing 3D animation which costs thousands of GB’s to render and are so interesting to watch and know about. However, the documentaries are very hard to understand for the young minds and so our goal is to let the young ones as well as adults, grow fond of the space and the Solar System. They can understand it better and learn about it while enjoying and playing with it. We also inserted a spaceship cockpit UI to make them feel like they are piloting a spaceship, one quite futuristic with holograms.

Originally, we wanted the Solar System to be as realistic as possible, so we even created the paper prototype of scaled real distances and we wanted to adjust them. We would need to bring closer the further away planets, even while scaled, so we just decided to set up the way it looks neat and clear. It is mostly a visualization for a younger audience, so for them the realism is not that necessary, more important is how interesting it looks like. However, for a simulation like this realism with distances may mean gigabytes of data and would require longer to accomplish.

For this assignment we used 3D graphics and OpenGL elements to achieve all the objects and effects such as lighting, 3D space, objects, 3D models, camera, black hole etc. We know a Black hole is not a part of the solar system, so we thought of removing it since it challenges the purpose of the app a bit. However due to the recent success of the “Event Horizon” telescope for taking the first ever picture of a black-hole, we decided to keep it as an Easter egg, as a tribute to “Powehi” the name given to the EHT’s black-hole of the M87 Galaxy.

Our Solar System has three modes:

#1 User can feel like he is navigating the spaceship and can travel around the planets.
#2 User can follow one of eight planets and at the same time can read the description of the planet.
#3 User can see the whole Solar System and rotate it in the divergent directions.

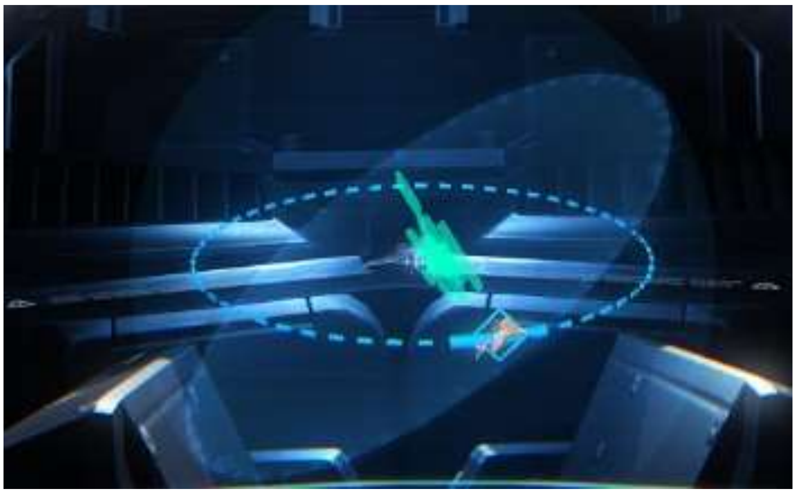
#2 USERS EVALUATION

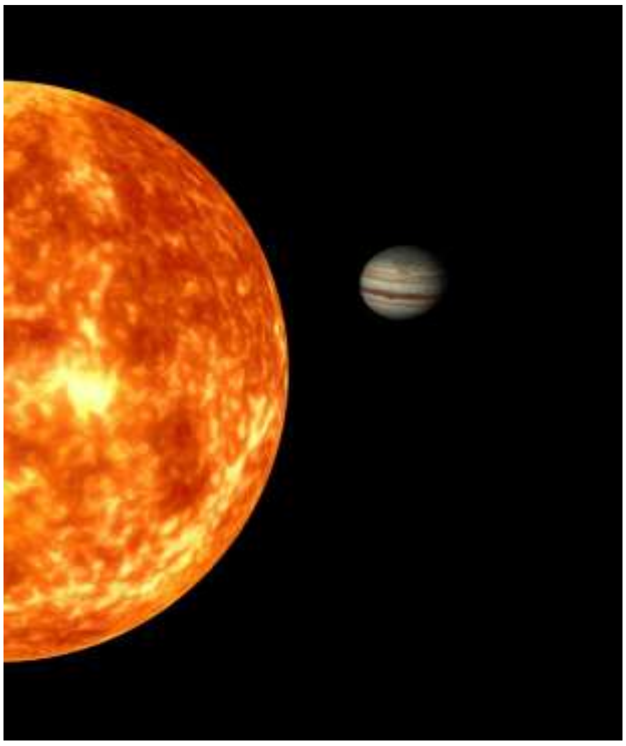
While we were working on this project we could see ourselves how important the users' opinions are to set the right direction for the work and essentially how valuable they are to solve various disagreements in the team. We know it was said so many times that setting the target audience is one of the most important jobs to do at the early stage of the project. We know this app will be mostly for users who still attend to school and are below 15 years old. Despite this knowledge, we asked for opinions mostly older audience: developers and programmers, but not potential users. The lesson from this part is to try always to reach out to target audience, which may have needs, others than us computing people.

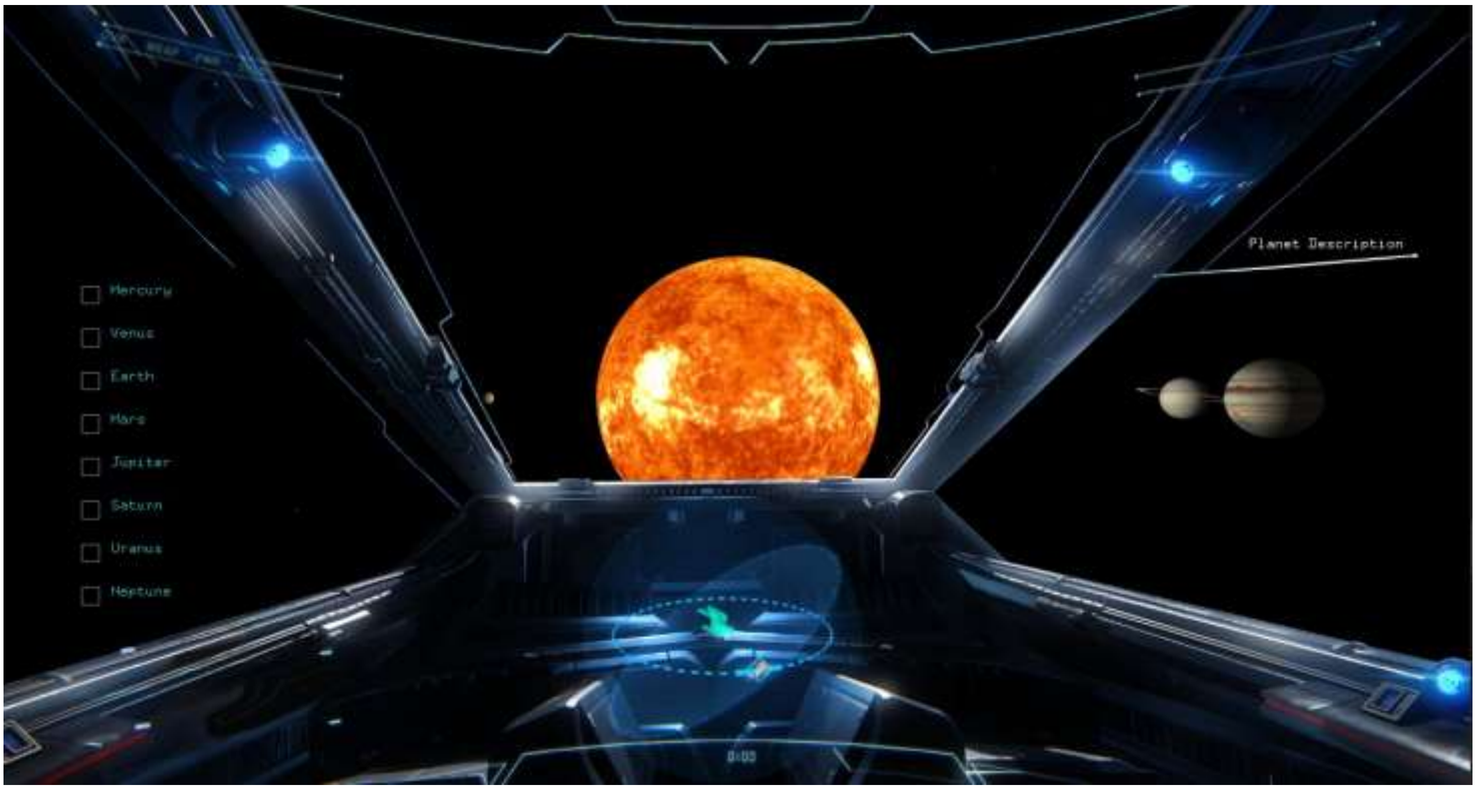
READ MORE:

Users opinion – week 3
Users opinion – week 8
Users opinion – final project

week3-users opinion.pdf   week8-users opinion.pdf   Users opinion - final.pdf


Spaceship 3D hologram model


The Sun and Jupiter



#3 TOOLS AND PROJECT MANAGEMENT

Tools

C++ - OpenFrameworks

We both use Windows Systems on our computers, so we implemented our code in Visual Studio 2017, using the OpenFrameworks library. Even though our knowledge of C++ is not advanced, we were able to create Solar System with our basic skills.

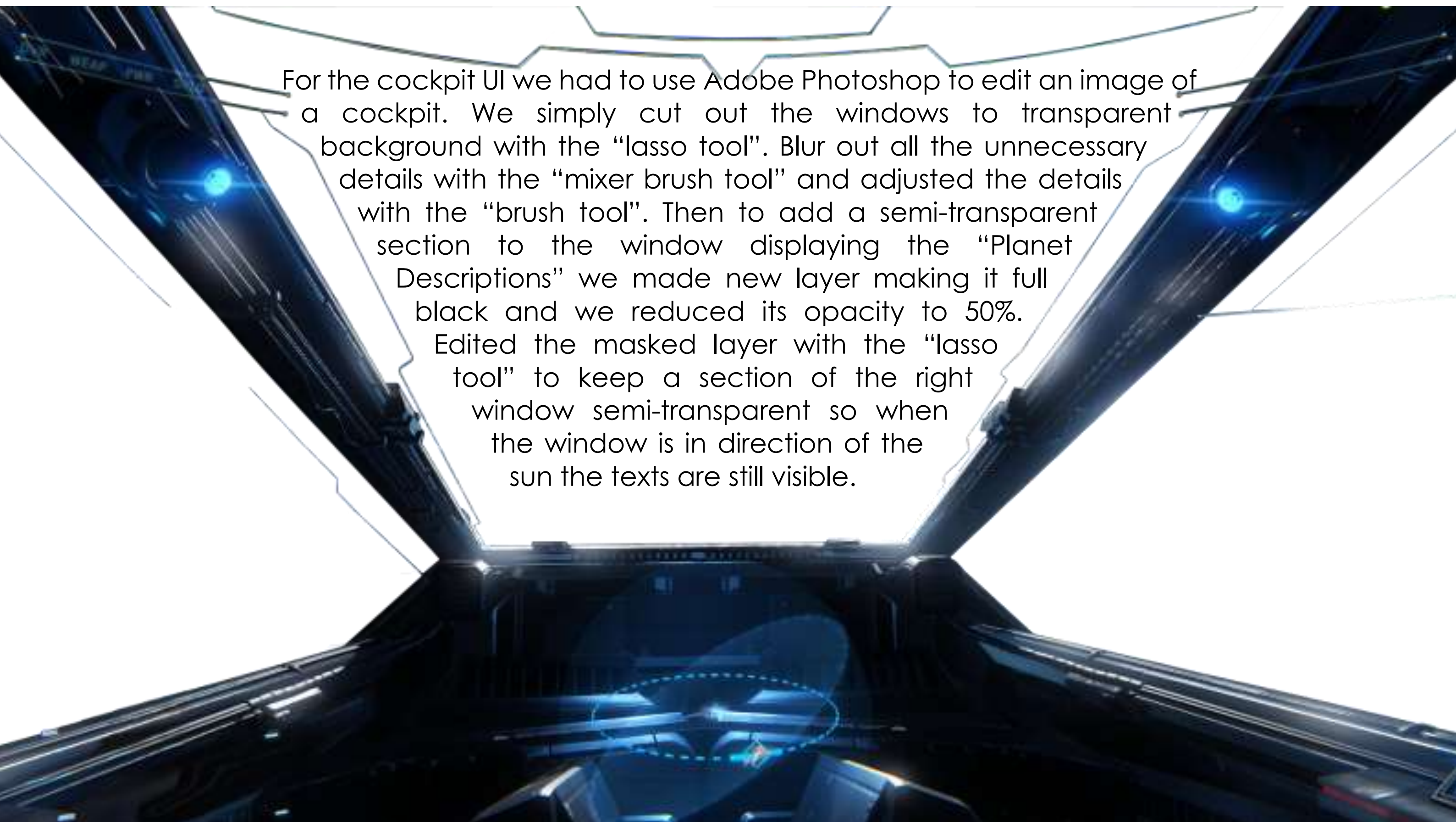
ofxGui – OpenFrameworks addon

For Graphical User Interface we decided to use the *ofxGui* addon from the OpenFrameworks library, because we had worked with it before at one of the Principles and Applications of Programming labs. We used the *ofxButton* to create the simple buttons, so the user can see descriptions of the planets and instructions as well. What is more, while the button to show description is pressed, we toggle it to get closer to the planet, which descriptions is currently being viewed. We wanted to use the *ofxPanel* but refrained from doing so because *ofxPanel* does not have built in function or way to implement spacing between buttons. In addition, the panel moves only while mouse is being dragged and we did not want that either. So, we went with buttons directly by making the back-ground of the buttons transparent making them look like holograms projected by the ship.

ofxAssimpModelLoader – OpenFrameworks addon

Next, we wanted to use a hologram of a spaceship as part of the cockpit, so our users could feel more like inside of a futuristic spaceship. We had to use *ofxAssimpModelLoader* instead of designing a spaceship hologram ourselves to not slow down the program. Moreover, instead of wrapping the 3D model with a texture we decided to just fill it up with a semi-transparent colour. To simulate a live feed of the ships vitals as shown in many sci-fi films and games, we added the rotation in all three X, Y and Z axis.

Adobe Photoshop



For the cockpit UI we had to use Adobe Photoshop to edit an image of a cockpit. We simply cut out the windows to transparent background with the "lasso tool". Blur out all the unnecessary details with the "mixer brush tool" and adjusted the details with the "brush tool". Then to add a semi-transparent section to the window displaying the "Planet Descriptions" we made new layer making it full black and we reduced its opacity to 50%. Edited the masked layer with the "lasso tool" to keep a section of the right window semi-transparent so when the window is in direction of the sun the texts are still visible.

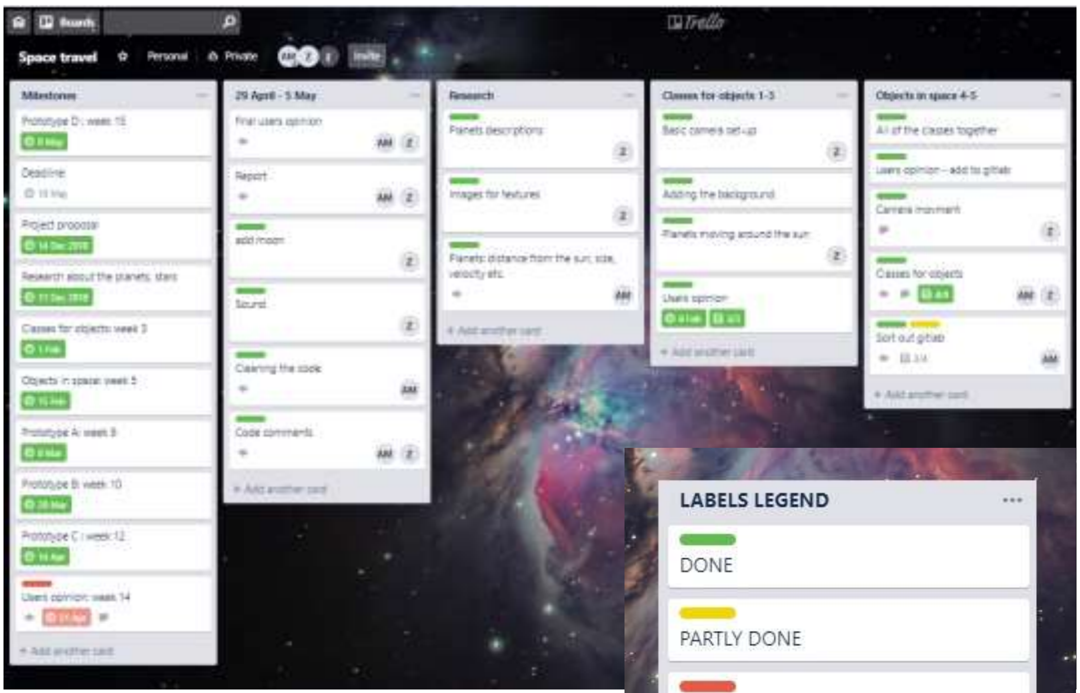
Project management

Maintaining such a huge project is not possible without long-term detailed planning. To do so, we were using GitLab repository, Trello board and Excel sheet to keep track of our hours. Apart from that we also created file with the rules for names of commits code plan or general rules for coding. Those files are accessible on our GitLab repo.

Trello board

Trello is an extremely useful tool which helped us to manage our project effectively. At first we set our milestones and scheduled deadlines for all of them, so we could keep track of time as well. Then we created separate cards for each of the milestones, where we added lists of tasks to do. Next, we split them up between us two. Unfortunately, even though we thought our plans were not complicated and we wold not spend much time on them, we struggled to finish all of them. We did not include possible problems with code, GitLab and that we will have to do that many assignments for other courses.

You can access our Trello board ***here****. On the right there is labels legend, will help you to understand our work better.*



Space travel · Personal · Private · Invite

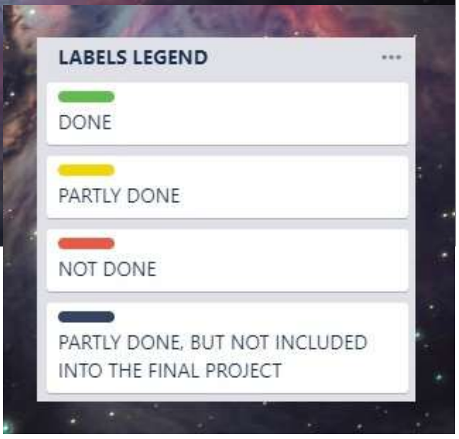
Milestones
- Prototype D: week 15 — 🟩 [illegible]
- Deadline — [illegible]
- Project proposal — 🟩 14 Dec 2018
- Research about the planets, stars — 🟩 [illegible] Dec 2018
- Classes for objects: week 3 — 🟩 1 Feb
- Objects in space: week 5 — 🟩 15 Feb
- Prototype A: week 8 — 🟩 8 Mar
- Prototype B: week 10 — 🟩 28 Mar
- Prototype C: week 12 — 🟩 14 Apr
- 🟥 Users opinion: week 14 — 🟥 [illegible]

29 April - 5 May
- Final users opinion — AM Z
- Report — AM Z
- 🟩 add moon — Z
- 🟩 Sound — Z
- 🟩 Cleaning the code — AM
- 🟩 Code comments — AM Z

Research
- 🟩 Planets descriptions — Z
- 🟩 Images for textures — Z
- 🟩 Planets: distance from the sun, size, velocity etc. — AM

Classes for objects 1-3
- 🟩 Basic camera set-up — Z
- 🟩 Adding the background
- 🟩 Planets moving around the sun — Z
- 🟩 Users opinion — [illegible]

Objects in space 4-5
- 🟩 All of the classes together
- 🟩 Users opinion – add to gitlab
- 🟩 Camera movement — Z
- 🟩 Classes for objects — AM Z
- 🟩🟨 Sort out gitlab — 3/4 — AM

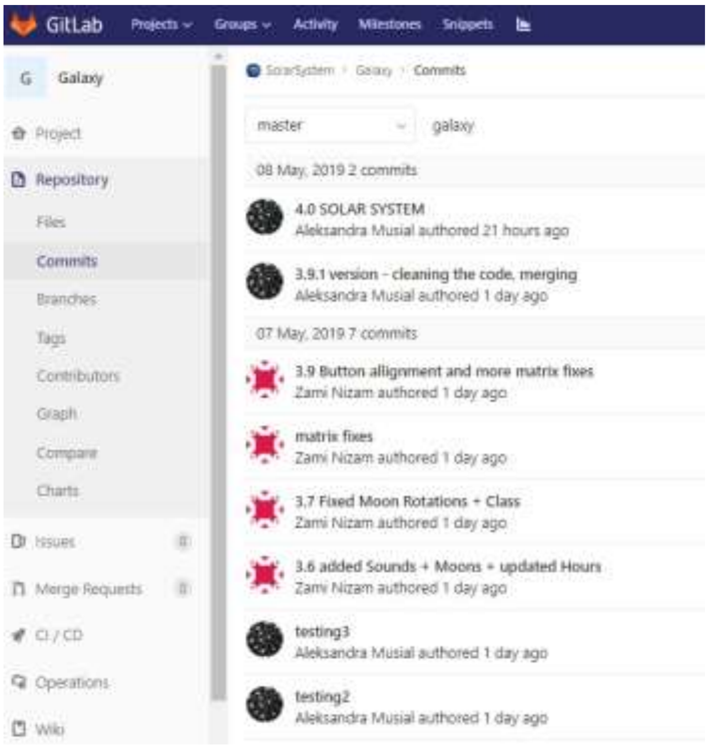


LABELS LEGEND

- 🟩 DONE
- 🟨 PARTLY DONE
- 🟥 NOT DONE
- ■ PARTLY DONE, BUT NOT INCLUDED INTO THE FINAL PROJECT

GitLab

Git is a helpful and necessary control version system to manage such a complex project, especially while doing it as a team. At the same time, it is complicated and difficult for new programmers. It took us a lot of time to make everything work well. Our biggest mistake was that at the beginning we were adding all the files which were created by the Visual Studio, so our repo grew to 2GB and was difficult to maintain. We kept having merge conflicts. While we decided to add the '.gitignore' file everything seemed to work fine, until we noticed that Zami was not able to pull or push from the repo. We tried to fix it for more than two weeks, so at that time instead of implementing our Solar System, we were just reading forums, websites and were asking lab assistants, lecturers and our colleagues for help. At the end it appeared to be a pity bug, which we had not noticed before. At least now, we are this one experience ahead.



08 May, 2019 2 commits

4.0 SOLAR SYSTEM
Aleksandra Musial authored 21 hours ago

3.9.1 version - cleaning the code, merging
Aleksandra Musial authored 1 day ago

07 May, 2019 7 commits

3.9 Button allignment and more matrix fixes
Zami Nizam authored 1 day ago

matrix fixes
Zami Nizam authored 1 day ago

3.7 Fixed Moon Rotations + Class
Zami Nizam authored 1 day ago

3.6 added Sounds + Moons + updated Hours
Zami Nizam authored 1 day ago

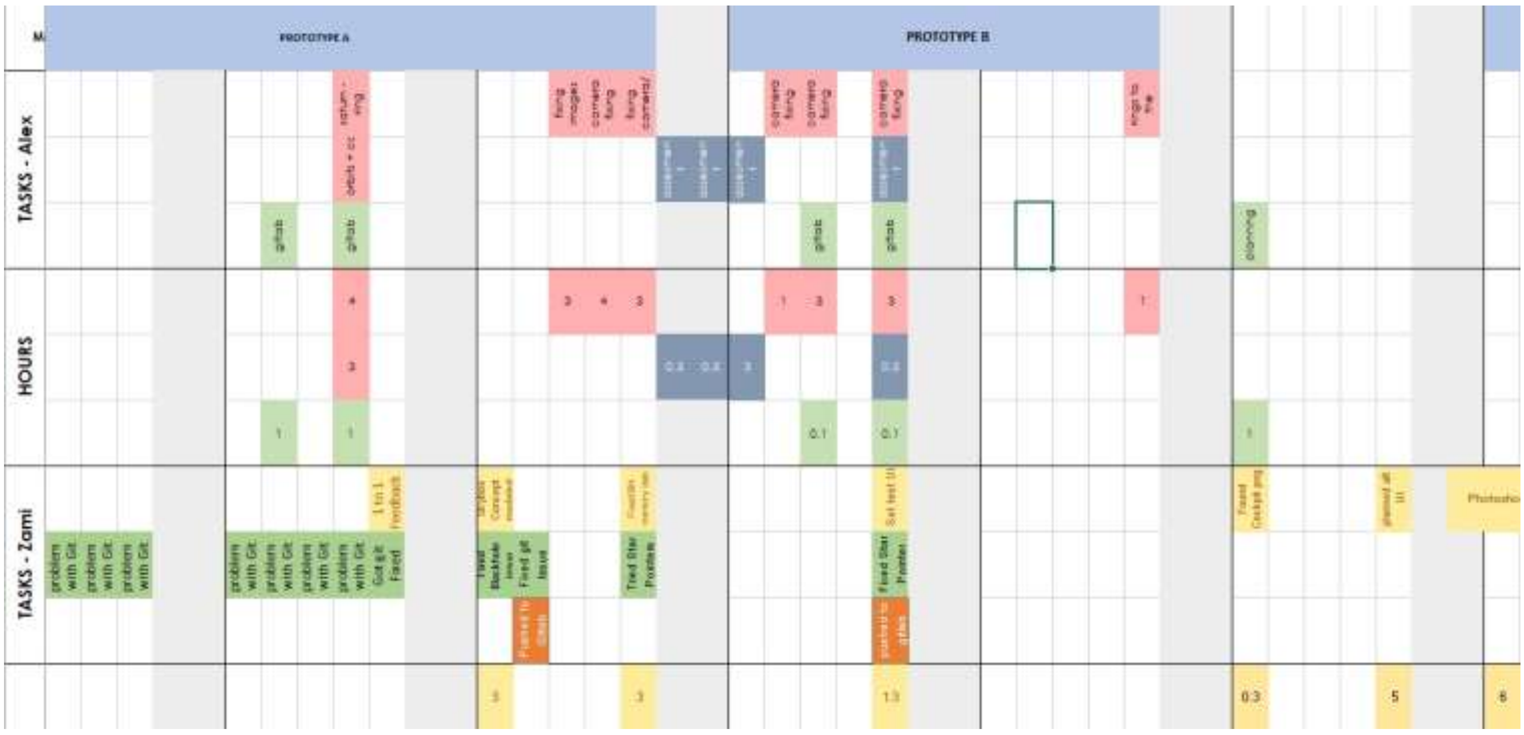
testing3
Aleksandra Musial authored 1 day ago

testing2
Aleksandra Musial authored 1 day ago

We created branches, because we were working on different modules. We added four version tags to our project, which seems only one is working and on our local repositories we have much more of them, but we were not able to manage them. We tried to use issues tracker, but we gave up this idea, since most of the time, we were working on different parts of code and we always discussed any problems when we were in the lab or at our meetings.

Microsoft Excel

In our project proposal we wrote we will spend around 10-15 hours on our project. We had 16 weeks to finish it, so we should spend on this project 160-240 hours. It did not go that well, as we mentioned it in the 'Trello board' part. The hours sheet will help us in the future to see how to maintain our projects better, because it is clearly visible that one week we could work on our project every day and the next week not at all. Life is a rollercoaster, but even though we should always find a moment to even think how to solve one of the problems in our code.



PROTOTYPE A

PROTOTYPE B

TASKS - Alex

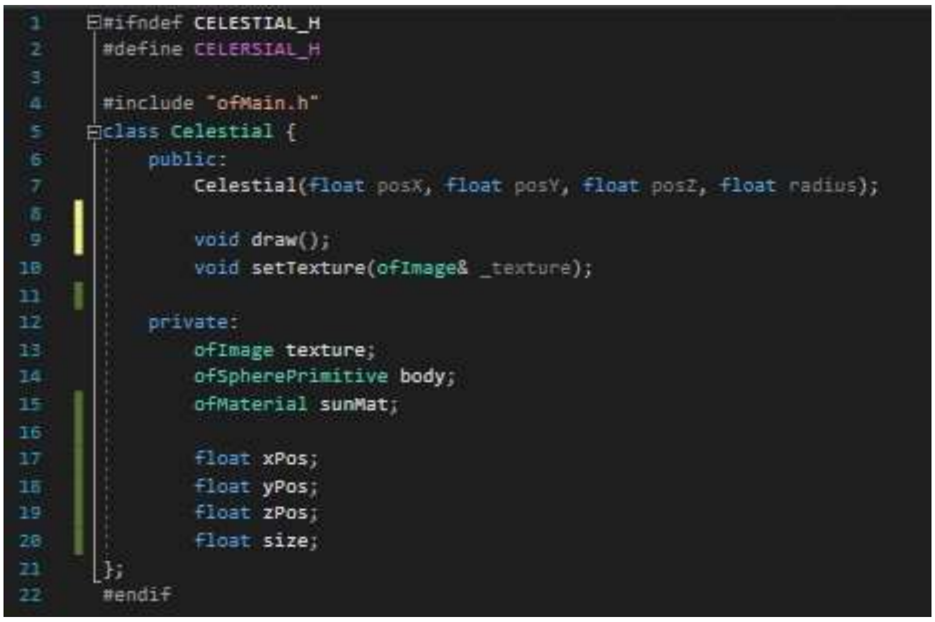
HOURS

TASKS - Zami

Check out our Excel sheet in the repository.

#4 STRUCTURE OF THE CODE

We approached to write our code using Object-Oriented Programming. Firstly, we used one of the basic concepts of OOP, which is creating the separate classes for objects e.g. planet, star or celestial. The objects are being declared inside the offApp.h, so in ofApp.cpp the data members and function members can be accessed. We also use the encapsulation concept, which means most of our class members are private: e.g. positions. To do so we implemented this code.



```
#ifndef CELESTIAL_H
#define CELERSIAL_H

#include "ofMain.h"
class Celestial {
    public:
        Celestial(float posX, float posY, float posZ, float radius);

        void draw();
        void setTexture(ofImage& _texture);

    private:
        ofImage texture;
        ofSpherePrimitive body;
        ofMaterial sunMat;

        float xPos;
        float yPos;
        float zPos;
        float size;
};
#endif
```

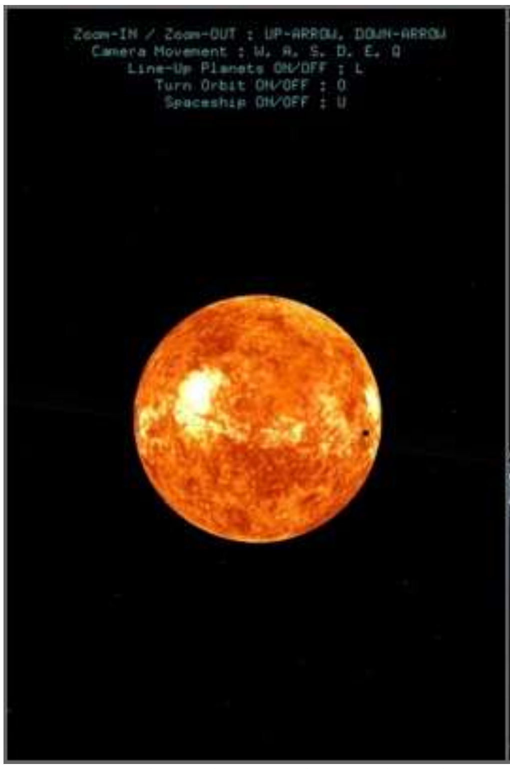
PLANET CLASS

This is the first class we implemented, and it is being used to create all the eight planets in the Solar System. Primarily, the rotation of the planets was set inside the draw function of the ofApp.cpp file, using the *ofxRotateY*(). Later, we realized, the positions of the planets are not being updated, which meant we could not use *lookAt()* function to properly calculate the distance between camera position and the planets. That is way we had implemented the code to move the planets in the circular move. Planet class also allows us to draw orbits and set the position of the planets to one line.

CELESTIAL CLASS
This class looks almost the same as the planet class. We could just inheritance from Planet class to create this one. Sun class sets the material to emissive, so it appears as a source of light for planets. Unlike planets, sun is not rotating.

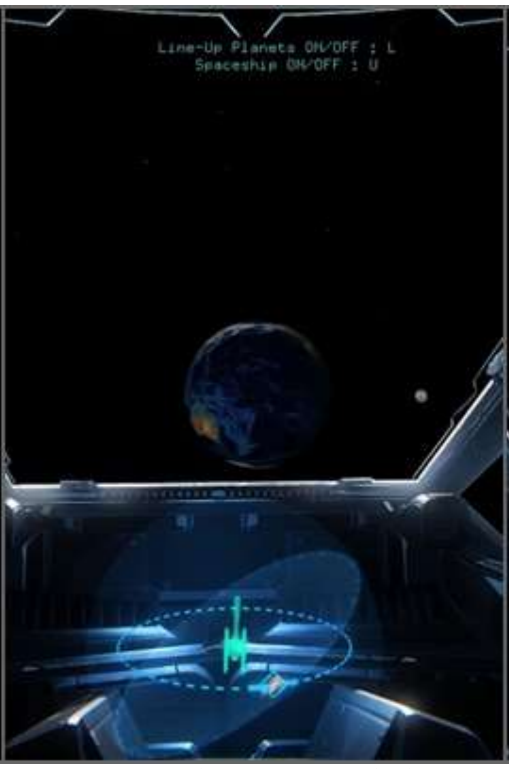
STAR CLASS
Basic class which draws just one star. We create more stars inside the ofApp file. Firstly, creating the empty vector of type Star, then setting the random values for position and size of stars, and at the end drawing them inside the draw function. Star system was more complicated than we thought it will be, since drawing many objects significantly slows down the program. Eventually, we just decided to draw less stars and we used pointers.

INSTRUCTION CLASS
It is a simple class, which we wrote to clean up the code in ofApp.cpp file. Depending on the mode we have three various instructions. The reason behind it, is we simply were not able to implement *every function in each mode, so we just had to unable them.*

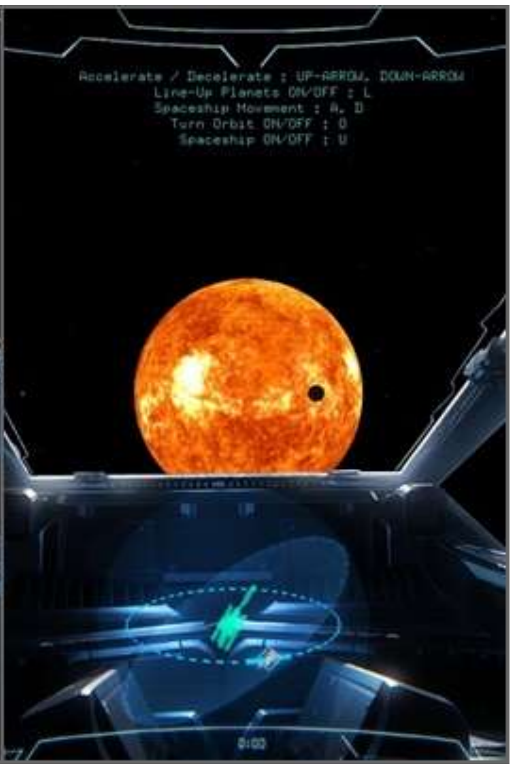


Zoom-IN / Zoom-OUT : UP-ARROW, DOWN-ARROW
Camera Movement : W, A, S, D, E, Q
Line-Up Planets ON/OFF : L
Turn Orbit ON/OFF : O
Spaceship ON/OFF : U

User can feel like he is navigating the spaceship and can travel around the planets.



User can follow one of eight planets and at the same time can read the description of the planet.



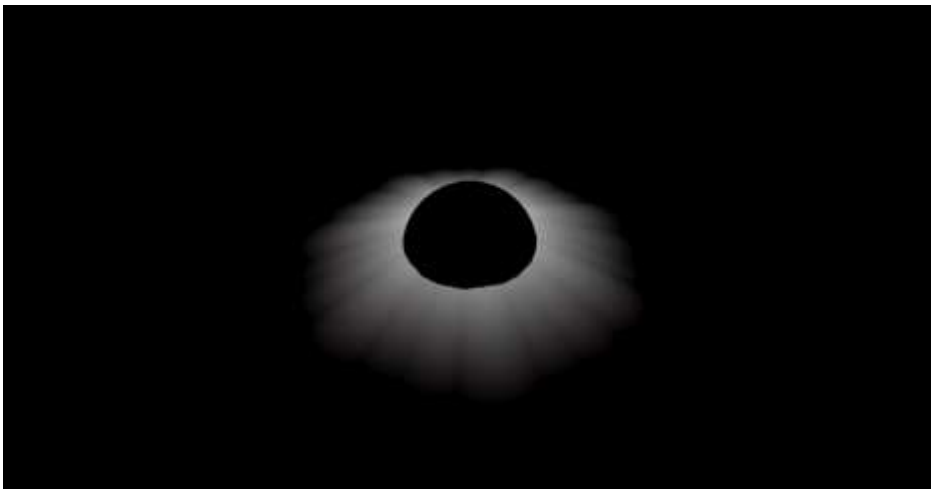
Accelerate / Decelerate : UP-ARROW, DOWN-ARROW
Line-Up Planets ON/OFF : L
Spaceship Movement : A, D
Turn Orbit ON/OFF : O
Spaceship ON/OFF : U

User can see the whole Solar System and rotate it in the divergent directions.

OFAPP CLASS
This is the brain of our project. We set here the lighting, camera movement and we call the member functions on the objects. We also create and draw here our User Interface. Also, the sounds and 3D model are being loaded. In update function all the magic happens that allows us to change the position of camera, change the position of the planets to set them in one line and either if descriptions will be shown or not. In the draw function camera begins, but it not ends at the end of the function, because we wanted to load Spaceship image, buttons, instructions and descriptions on top of the Solar System, they must be outside of the camera. Not using the *ofxEasyCam* was challenging, it took us lot of time to find the way to move camera the way we wanted to. We could achieve that using the *ofxCamera*, part of the *ofxNode* library. The *rollDeg()* function helps us to rotate Solar System from left to right relatively to current z-axis position, *orbitDeg()* helps us to orbit camera around the origin, 1000 far away from it. Key pressed function plays a key role in our code. It allows user to interact with the Solar System. There also exists the function which allows us to tokenize the description words. In general, our ofApp class is quite expanded and sometimes logic behind it could be much simpler, than it is now.

MOON
Moon is a vital part when it comes to both realism and visuals. Adding it was not a piece of cake, as it's supposed to rotate around the Earth and around the Sun. To do so we wanted to use our planet class, but because it is set to rotate around the sun we could not use it to work as a moon. After a lot of research, we found out about the "orbit" function for the *ofPrimitives*. Then, we were able to achieve movement around Earth, but we had to use a 3D primitive directly. It is not a very efficient way, if we would like in the future to customise or add more moons. The only option that remained was to create a class. After that, a new challenge appeared, since we were not very familiar with the "orbit" function and if we are to use a moon class we have to re implement the orbit function within the moon class so that when we call to the moon object in our "ofApp.cpp" we have access to the "orbit" function. Now understanding the functionality of the "orbit" function may not have been this hard if the documentation on OpenFrameworks was a little bit better. However, we were able to figure it out and implement it in our moon class now we have a perfectly working moon class which in future if needed can be used to implement more moons to our app. At present we only setup one moon for the Earth, priorities! Because if we were to set moons to all planets the program would become very heavy on memory, because planets like Jupiter and Saturn have 69 and 62 moons each. And we decided to go with one moon.

BLACKHOLE CLASS
The black-hole class uses a spiral mesh object rotating. We had to flip it upside down to imitate the event horizon of the black hole. We took the code for the spiral mesh object from a Graphics project we did. And we added a black sphere in the middle as the hole. To implement it to the code we had to change the colouring and convert it to a class. We adjusted the code to reduce memory consumptions like reducing the number of segments a bit.



#5 PITFALLS AND CONTINGENCIES

Unanimously, we can say all ours concerns from the project proposal came true. Object clicking, description mapping, adding moons, memory consumptions or even camera movement. Below there are some example problems we struggled with during the process of creating this project.

Memory leak – Black Hole
The black hole class had a memory leak. It caused the memory consumption to reach gigabytes and then it crashed the app. We did not know what was causing it, so we had to go over the code several times and finally figured it out and fixed it. Shockingly, it was just one line of code being called in the wrong function; in the draw function making it draw repeatedly, taking so much memory that at the end it crashed.

Excessive memory consumption of Stars
Due to the high number of stars the app was running extremely slow: 7-8 fps. At first, we decided to reduce the number of stars being drawn and set the resolution of the star objects to smaller amount. Additionally, instead of randomly setting the positions in floats, we change it to integers, which also saved some memory. What worked perfectly was using the pointers. At the end we achieved a more than a 2X boost in performance as the app now runs in about 20 fps, which is great.

Button class
To make the code clearer, we created the button class, which was meant to store the size, colour, positions and the state of the buttons, triggering the right event. We were able to draw the buttons in the ofApp.cpp file but we could not change states of them. They simply did not work. We could try using lambda expressions, as in one of our DJ GUI implementations in Principles and Applications of Programming lab, but we simply ran out of time. Currently, we are just creating the buttons in ofApp.h and calling them in ofApp.cpp.

Coordinates system
Our main idea for the user interaction with planets was easily mouse pressed function. We did not realize at the beginning, that we will have to manage four coordinates systems at the same time to do so. Camera's position and planet's positions in 3D, mouse position in 2D, and then somehow make work all of them in 2D screen. Rotation function, which takes only one line of code, was the main reason we were not able to manage all the coordinates together. Simply, because it rotates the whole coordinate system, so the positions of the camera and planets were not changing. It took us long time to figure it out and still it was not the end. We had to calculate the distance between the camera and screen, and furthermore calculate the length between planets and that distance. At the end we were able to implement it, but just for one planet. We did not have time and not much free memory to do the same thing for eight planets. Especially, because many lines of code were just hard-coded.

Rings
We planned to add rings to all the planets that have them: Jupiter, Saturn Neptune and Uranus. We could not find one function in OpenFrameworks to create torus. What we found was the code which was built out of many rectangles and then was creating the torus, which remarkably slowed down the program. Considering this, we decided to add only one, Saturn ring.

#6 FUTURE DESIGN

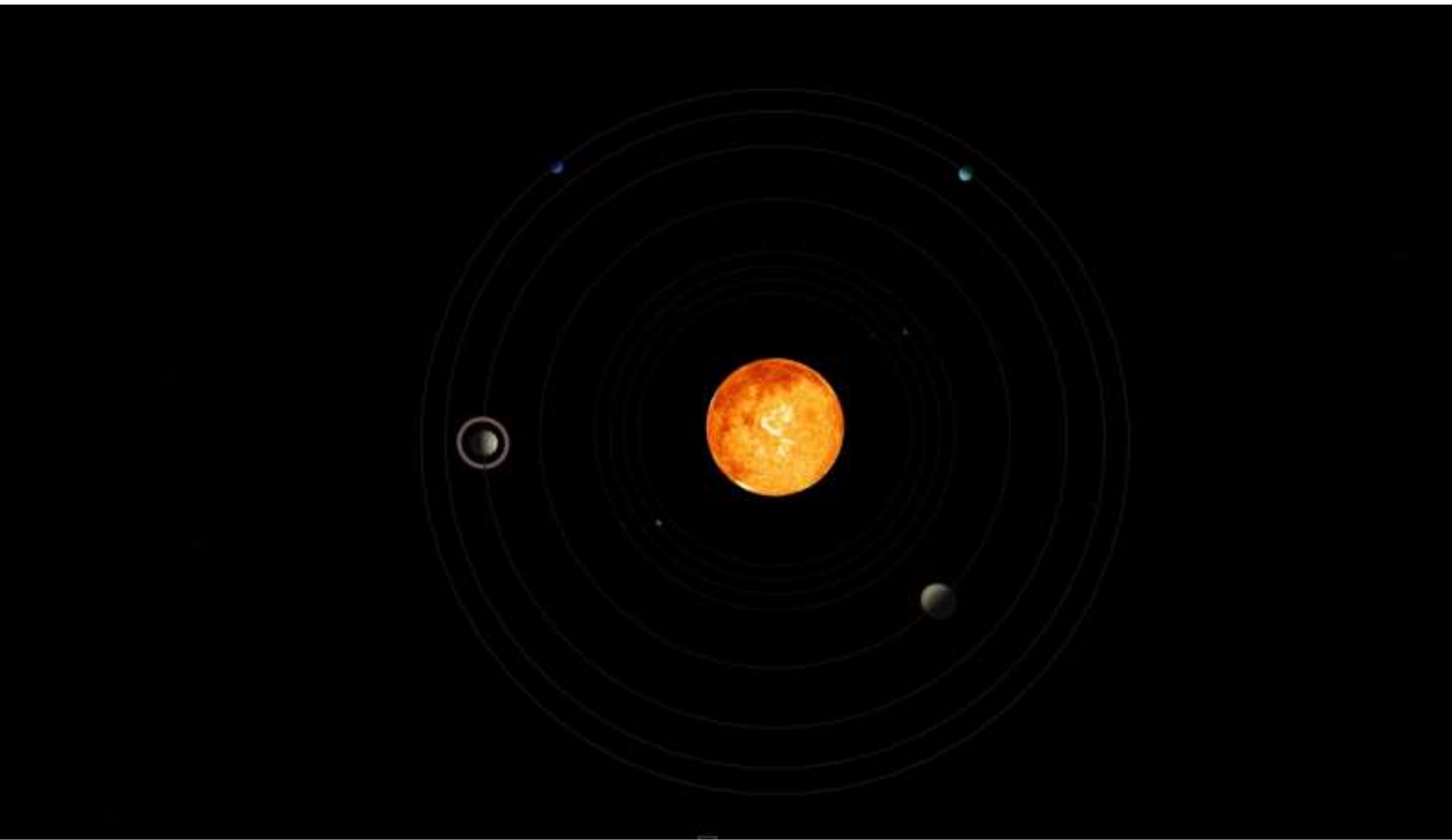
For future design improvements we would like to implement click on the planet function, as we were close to achieve that. If we had more time we would improve GUI, but most likely write our own GUI implementation with better looking buttons and panels. Adding more stars with their own little emissive lights would make our Solar System brighter and pleasant to look at. Working on User Interface would be a huge part of our future coding, we would like to add for example animating buttons.

From technical aspects we would like to compress the code. Even though we changed many functions, the code is repetitive in many modules and could be maintain more neatly. Regrettably, we did not have a chance to use much of our knowledge from Graphics class. That is the reason why, if we have more time we would like to write our own shader instead of texture for the Sun. There is so many futures we would like to implement, but this time lets be more realistic than in our project proposal.

#7 CONCLUSION

To sum up, programming in C++ was challenging and more complicated than for example the game which we made last year in JavaScript. Apart from managing how the app will look like, we had to optimize memory, make the code as much efficient as possible and adjust the plan to users' opinions. This project helped us not only to stretch our C++ coding skills, but also how to organise project, work better as a part of the team, keep track of the time and that sometimes the best solution is just give up on some tasks or evaluate them to less complicated and be on time for the deadline. We think it is essential lesson in computing industry.

Even though, we were convinced we planned the project thoroughly, if we had done more, the process would be much simpler. Instead of just preparing the code structure, we could write the pseudocode as well, it would make our logic coherent and we would not have to adjust the same lines of code so many times. There is plenty of aspects we would like to change, but the essence is that we have learnt, and we will do the next project better.



#8 REFERENCES AND BIBLIOGRAPHY

Camera rotation adaptation:

(2015, May 9). Retrieved 10 May 2019, from OpenFrameworks website: https://forum.openframeworks.cc/t/simple-camera-positioning-and-lookat/19514/2

Circular movement of the planets:

(n.d.). Retrieved 10 May 2019, from: https://openframeworks.cc/ofBook/chapters/animation.html#functionbasedmovement

Torus code to create a ring:

(2011, September 5). Retrieved 10 May 2019, from OpenFrameworks website: https://forum.openframeworks.cc/t/torus-not-displyed-solved/7242/3

Textures, images, sounds:

Solar System Scope. (n.d.). Retrieved 10 May 2019, from Solar System Scope website: https://www.solarsystemscope.com/

Earth, N. V. (2009, August 11). January, Blue Marble Next Generation w/ Topography and Bathymetry [Collection]. Retrieved 10 May 2019, from: https://visibleearth.nasa.gov/view.php?id=73580

Uranus Texture Map 2017. (n.d.). Retrieved 10 May 2019, from DeviantArt website: https://www.deviantart.com/magentameteorite/art/Uranus-Texture-Map-2017-727689611

The Sound FX Channel. (n.d.). *Sci Fi Computer Interface sound effects*. Retrieved from: https://www.youtube.com/watch?v=hTuUYMLJCpY

ANDULAIRAH Of EARTH. (n.d.). *Actual Sounds From Space! NASA Voyager Recordings*. Retrieved from: https://www.youtube.com/watch?v=giU92w_S3V0&t=1s

free 3ds mode qa test. (n.d.). Retrieved 10 May 2019, from: https://www.turbosquid.com/3d-models/free-3ds-mode-qa-test/710119

CGTrader - 3D Model Store. (n.d.). Retrieved 10 May 2019, from CGTrader website: https://www.cgtrader.com/