Title

**Abstract**

The appointed goal of this project and the scope of this documentation is the implementation of a simple snake game solely written in C++ using the OpenGL interface and GLFW library. The challenge of the project was to create a 3D space and to implement a playable game within, without using an already existing game engine or any other prepared parts. Everything you see and everything that works behind the scene is implemented by us. Even so smaller code segments may have been tipped 1:1 from examples, the project as a whole is only relying on our efforts and knowledge.

**Introduction**

**The Goal**

The purpose of the project is to implement a program related to the scope of Computer Graphics and Physical Modelling. To be more precise a simple snake game was decided to be implemented. But as mentioned before without the help or usage of an already existing game engine. So, for this matter not only the product itself is the goal, but also the means to achieve it.

**Requirement Definition**

There are plenty of resources to call upon if there is a program to be implemented. Since no one of the project team is fluent in machine coding we have to use a higher programming language hence an IDE. It was decided to use CodeBlocks as an as simple as possible solution, with an also simple MINGW compiler.

An also important requirement is not to use an existing game engine. Which means that we are able to use anything like library’s and interfaces to create one.

The requirement for the game itself a pretty straight forward since we are reinventing the wheel. The game has to be capable of providing the usually snake game rules as there be:

Nonoptional:

-The snake moves constantly in jerky leaps

-One can only control the direction of the movement

-There are four directions

-One can not turn on the spot, 180°

-If the snake happens to bite itself the game is over

-If the snake happens to touch the borders of the playing ground the game is over

-There is always one random point to on the playing ground

-The point may be eaten by the snake

-Points eaten add a snake segment to the end

As also Optional assets:

-The border of the playing ground is represented by a laser fence

-More than four directions e.g. use of the 3D space.

-Model Loading

**The Process**

**Recherche**

Due to the little time we had, our first approaches were to watch records of other people’s 3D games implementation in hope to learn enough to mimic their efforts and to produce an own one. The lack of sufficient documentation made this one futile. Fortunately, there were plenty of sources on YouTube where people shared their effort to create little games in C++ while using OpenGL in detail.

**Problems**

The current state of our project was proceeded by several attempts, failures and hence resets of the project. The first and most severe problem we encountered is the language itself.

Because C++ other than Java does not run on every system independently good as it does on the machine at that it is written and compiled on. C++ has many implementation dependencies and all this different software changed overtime and might not even be compatible through the whole development process.

For example, in the final project a bug happens to appear even though till now all known dependencies were correctly implemented and tested for that matter. The documentations of the current versions showed everything should be compatible but that was obviously not the case. What happened was that our OpenGL version was the currently newest one but our OpenGL Math Library were older since it was downloaded by a link within an old tutorial that pointed to a likewise old repository.

After a little digging the error turned out to be an old and forgotten missing downward combability. During the development of the OpenGL Math Library they changed the expected parameter integer value of a method to be in degrees while the older version provided it in radians. The next two OpenGL Version had pointed out this change but by know it was forgotten and the exception got removed, hence no error message while we were building our project.

We also had to 🡨--------------------------

**The Final Product**

**Shader**

Shader are small programs that are written in their own language depending on the project they are belonging to. In this case it is an OpenGL project hence shaders are written in GLSL a C like language. Shader can either be implemented in the source file itself as a C string and then read in by a shader compiler or saved in an external file.

Shader are running on the GPU of the system, if provided and necessary for rendering. They are independent from each other and concerning the scope of this project are rather simple. They usually have only two kinds of variables as there be input and output and passing them along.

**Textures**

Textures are easy ways to provide some sort of fancy surface images for simple objects. In OpenGL. They are stored as external files a then loaded in the unsigned int variable within the program. From there they can applied to any object may it fit the texture or not. OpenGL provides many options to align textures to their objects.

**3D Space**

Implementing the 3D space were for sure the biggest challenge. The course of this project was to build always upon the last step at the beginning an OpenGL windows was created, then filled with simple objects and textures got implemented. But everything happened in 2D space. The step to 3D space required to rewrite all shaders and the shader header as well as much of the code itself. All of that while it was nearly impossible to check for smaller accomplished steps. It was a huge rebuild of the entirely project and if it did not work there were little chance for debugging the whole project. So it took several attempts to convert the 2D environment to 3D.

Now we have four transformations implemented.

1. Local to World Space
2. World to View Space
3. View to Clip Space
4. Clip to Screen Space

The Local Space is a space in which the object exists solely by itself defined by its parameters. After the first transformation the object exists in a relation to the origin. After the transformation to the View Space every object exists as it is perceived by a viewer. The Clip Space is the space is a -1.0 to 1.0 matrix everything beyond is not to be rendered. The Last transformation converts the -1.0 to 1.0 space into a 0 to X and 0 to Y space depending on the screen resolution. Giving us a pretty good perception of a 3D space.

**Game Logic**

The game logic was actually pretty quickly implemented. There is a reason game engines are such a big market and that is because is saves you a lot, the bigger part even, of work. Sure, there were problems to overcome but after all this was still the smallest working package.

**Debugging**

We had a lot of debugging to do through out the whole project. Since we had started a few times new from scratch the scope here will only regard the last and working approach.

Usually debugging is always a little troublesome but manageable. But we faced the problem of using our own shader and they are not really communicative. For example, if the shader does not get the correct bitstream, the offset is wrong or the code not working you are going to see nothing in your window. The shader also won’t tell you what is wrong, probably not even that he had ran into an error. And that is really troublesome since you would have to look for each of the examples in a different location.

Maybe there is a way to implement shader that tell you what is wrong but this is our first project of this kind and we didn’t and don’t know how.

**Conclusion**

This project was one of the harder one but because of that also one of the educationally. It gave us a really good perception of what and how a game engine is doing. As a matter of fact retrospective we shouldn’t have named this project after snake but more like “A basic game engine and a simple game within” since the setup was the way bigger task. In the end there is only to say even though it gave us all some real headache it was totally worth it.

**Lookout**

As a future project we could try to get our game more closely to the unity implementation. That would mean that we had to implement a camera method which we already started working on but didn’t finish and model loading, which we would like to have but didn’t even touched because we were so much behind schedule. After all it is possible even feasible and after my troubles with the transformation matrix’s I would assume that the lighting is probably going to be the worst part of it.

References

While some of the tutorials linked here are dropped due to compatibility problems. There is a good to fair chance that some of its code made it to the final project. Nevertheless the least contribution is teaching a little bit about c++ and opengl along the way.

https://learnopengl.com/Introduction

<https://www.youtube.com/watch?v=CKqfzgowG9U>

<https://www.youtube.com/watch?v=CKqfzgowG9U>

<https://stackoverflow.com/>

https://www.3dgep.com/introduction-opengl/