Earth Invaders

Final Project Computer Graphics Course

Jeroen Donkers
s1523430

@Leiden University, 2017

Student Number: s1523430
Guided by Dr. Michael S. Lew

FINAL PROJECT
COMPUTER GRAPHICS
LEIDEN UNIVERSITY

HTTP://PRESS.LIACS.NL/MLEW/CG2017.HTML

December 2017

1. Description

During the Computer Science course *Computer Graphics* I have been regularly learning basic theory and practice of computer graphics with strong focus to OpenGL. My final project was to use my gained skills to write a computer graphics interactive visual according to a certain theme. In this report I will explain and show the implementation of my take on the project.

1.1 Project specifics

The theme I chose to follow was making **Fireworks**. I had to make a visual of (realistic) fireworks, keeping in mind that it was important to have audio (eg. explosion sounds) integrated. The way of displaying the fireworks was

> "...up to my imagination."

and a basic implementation would earn me a sufficient grade, but

> "...the more elaborate, the better"

1.2 Idea

I was really happy that so much was left for my imagination. Working with this kind of freedom always gives me an extra boost to make the most of it.

I didn't want to just show some fireworks and be done with it. I wanted something more interactive than just being able to move the camera around the scene.
As I am making my first babysteps into the world of computer graphics just as the whole world was in the year 1978, I thought it was fitting to do a mockup of the game *Space Invaders* (1978). I would have OpenGL and a lot more memory at my disposal, so I could easily try to make a Space Invaders-like game in 3D. My idea was to show fancy fireworks when enemies explode.

1.3 Implementation

I chose to work with the OpenGL toolkit *GLUT*, and wrote my program in *C++14*.

To implement the game mechanics itself, I tried to stay close to the Space Invaders rules. You would move a character from left to right and shoot enemies coming at you from above.
The 3D aspect of the game really shows itself because of the *perspective mechanic*: When you pause the game, you move the camera instead of your character, and you would have to coordinate moving your character and the camera to shoot enemies using perspective.

1.4 Extra's

Enemy waves and boss waves

The goal of the game is to get as many points as possible before your health runs out. You get points by killing enemies or bosses before they hit the ground. Enemy waves get more difficult as the game progresses with more enemies spawning. Boss waves are always thesame, you have to fight one boss that always has thesame amount of health.

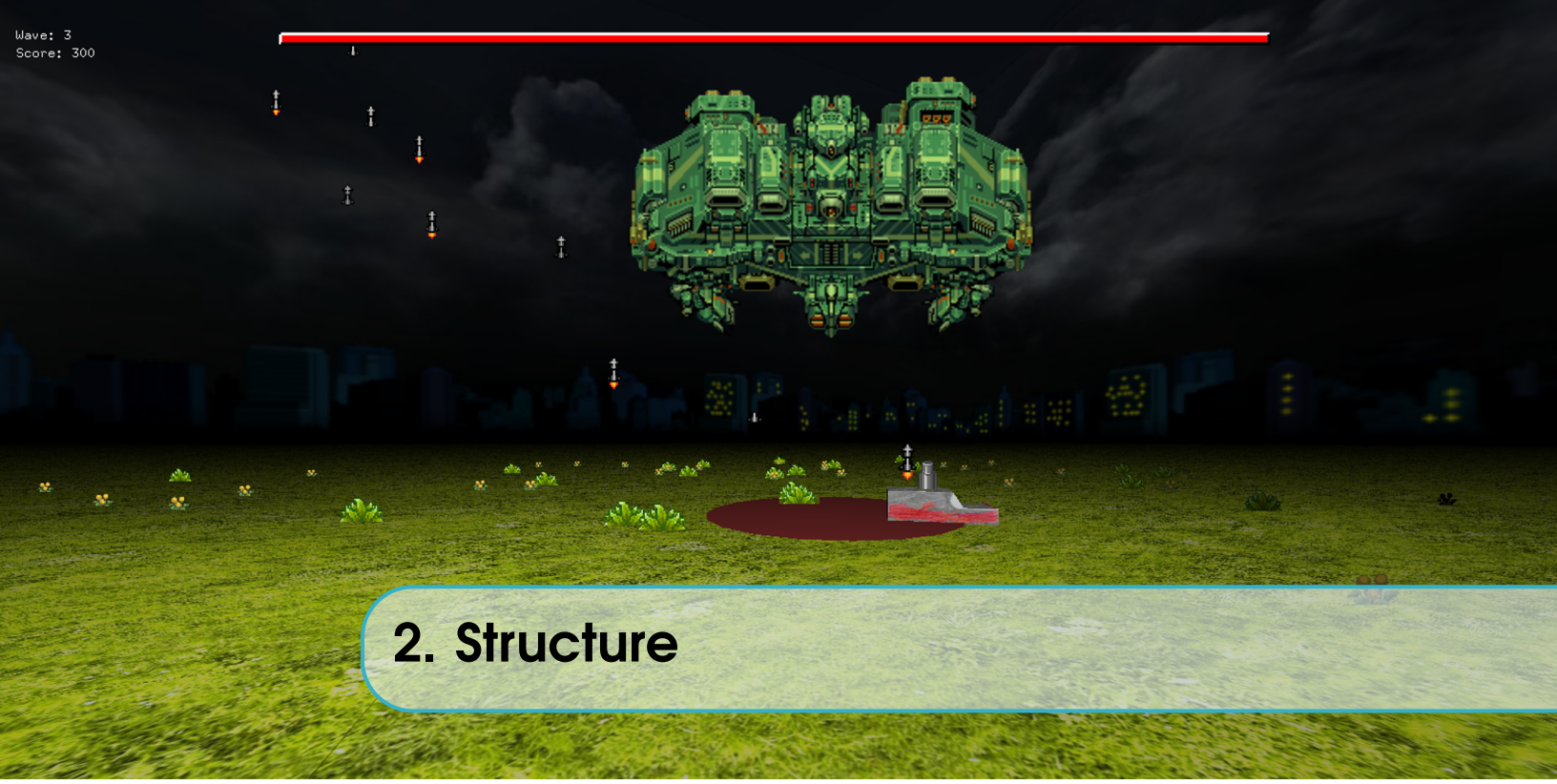
Highscores

There is a file in the project folder (`./HIGHSCORES.md`) that keeps track of the three best scores. If you got enough points to be in this highscore file, the game will tell you after you have finished the game, and you will be able to put your name in the terminal to be put in the highscore log.

HUD tooltip

When you drive too far off the screen, you get a helpful message saying you can press 'R' to reset your car. There are also messages when a boss wave is incoming (as seen in Figure 1.1) and when you finish a game and want to put in your highscore.

Figure 1.1: Warning! Boss incoming!



2. Structure

To keep everything clean and the project structure as organized as possible I made sure to write my project heavily object-oriented. This many files can be daunting to see at first, but having everything broken down into small parts really helps to keep things clear in the long run. My guess is that the filenames will probably speak for its function for most files, but in case it doesn't, I will explain in this chapter the use of every file in my source folders used to build my program.

2.1 Framework

All the main assets of the program are located in **Framework**. These are the most important and provide the building blocks for all other files.

main: The game window is created and the game (Game) and renderer (glut_rendering) are being initialised and called here.

glut_rendering: Handles all the rendering with GLUT. Draws the Game scene each frame.

Game: The game control flow is specified here. Everything from the scene and HUD to updating the scene objects is done here.

Fireworks: Creates multiple types of fancy fireworks.

2.2 Handlers

Some helpers that take care of specific things that are not necessarily as intertwined with the Game as the rest of the files.

HighscoreHandler: Reads and writes highscores.

InputHandler: Reads input, makes sure input is responsive.

LightHandler: Creates and handles openGL lights.

2.3 GraphicalObjects

All these files use the GraphicalObject class and are drawable objects in the scene.

GraphicalObject: Base class for drawable objects. Provides a simple interface between Game and all different objects.

Boss: Enemy in boss waves.

Cannon: The shootable cannon on top of the car.

Car: The player-controlled character.

Enemy: Killable enemies. Enemies are animated sprites (always facing the camera).

Explosion: A sprite animation of an explosion.

Ground: The ground with a texture.

Nature: Sprites for grass patches and flowers.

Rocket: A sprite animation of a rocket.

Skybox: The skybox is a textured cube to give the impression of a sky.

2.4 HUD

All objects related to the HUD, a 2D layer on top of the 3D scene that shows stats about the game.

HUD: Controls which items to draw and provides an interface between the Game and HUDitems.

HUDitem: A base class for any drawable object displayed on the HUD, with some simple drawing functions that objects can use alongside with the handling of the position of the object in relation with the screen width and height of the game window.

Health: Shows images of hearts as an health indicator.

BossHealth: Shows boss health bar on boss waves.

IntroMessage: Shows a tutorial message explaining how the game works.

WaveCounter: Shows a counter for the current wave.

PointCounter: Shows a counter for the current amount of points.

Tooltip: Shows a quick help message at the bottom of the screen.

2.5 Sound

A lot of files in this folder are from IrrKlang, an audio library specified in chapter 3. Here I only explain the files I created.

SoundEngine: Provides the program with sound. Loads and plays sound files located in src/media/.

2.6 Texturing

Animation: Loads a set of textures and handles iterating over the textures when binding the animation to OpenGL.

Sprites: Class where all textures and animations are initialised and saved.

Texture: Loads a texture in a texture buffer and binds the texture to OpenGL.

2.7 media

References to all media are included in the file `credits.md`, along with origin of all other files included in the project.

3. Libraries

3.1 SOIL 2008

Simple Open Image Library
Image library used for loading various image formats and putting it in a buffer, based on `stb_image`. Latest version and documentation can be found on `http://www.lonesock.net/soil.html`.

Files from the library are:
`lib/libSOIL.a`
`lib/SOIL.h`

3.2 Irrklang 1.5.0

Audio library used for loading and playing various audio formats. Latest version and documentation can be found on `https://www.ambiera.com/irrklang/`.

Files from the library are:
`lib/libIrrKlang.so`
`src/ik_ESoundEngineOptions.h`
`src/ik_SoundOutputDrivers.h`
`src/ik_EStreamMode.h`
`src/ik_IAudioRecorder.h`
`src/ik_IAudioStream.h`
`src/ik_IAudioStreamLoader.h`
`src/ik_IFileFactory.h`
`src/ik_IFileReader.h`
`src/ik_IRefCounted.h`
`src/ik_irrKlangTypes.h`
`src/ik_ISound.h`
`src/ik_ISoundDeviceList.h`
`src/ik_ISoundEffectControl.h`
`src/ik_ISoundEngine.h`
`src/ik_ISoundMixedOutputReceiver.h`
`src/ik_ISoundSource.h`
`src/ik_ISoundStopEventReceiver.h`
`src/ik_IVirtualRefCounted.h`
`src/ik_SAudioStreamFormat.h`
`src/ik_vec3d.h`
`src/ik_irrKlang.h`

4. Screenshots

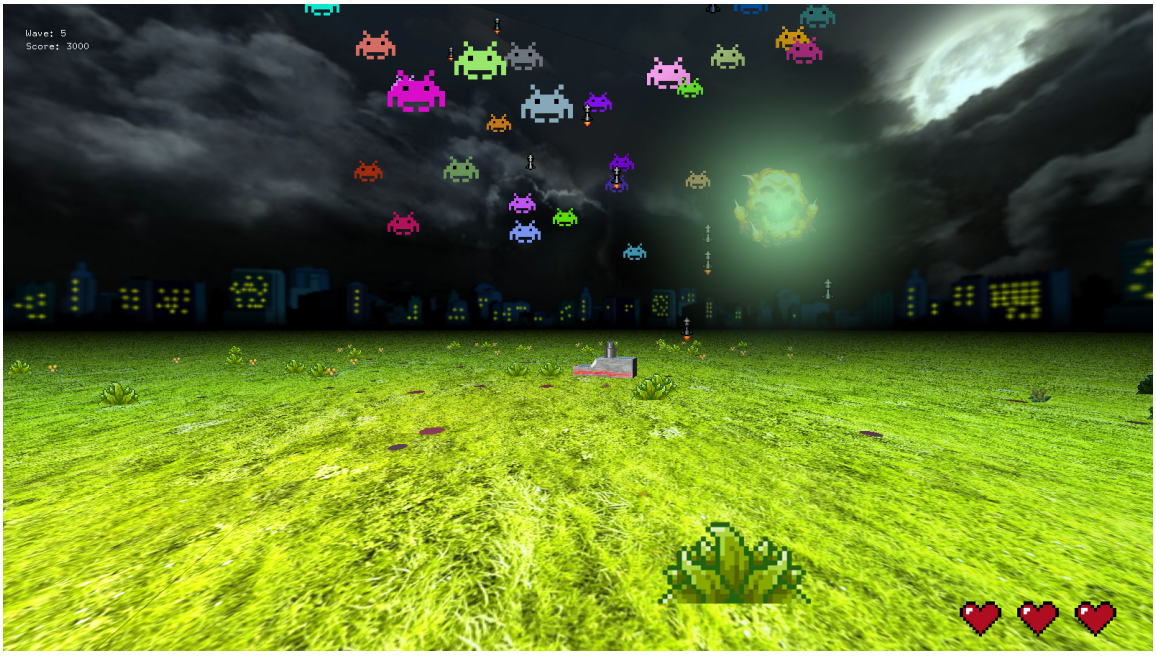


Figure 4.1: Fireworks glow.

Figure 4.2: Lots of cool explosions.

Figure 4.3: Spiraling Death Star-like fireworks.

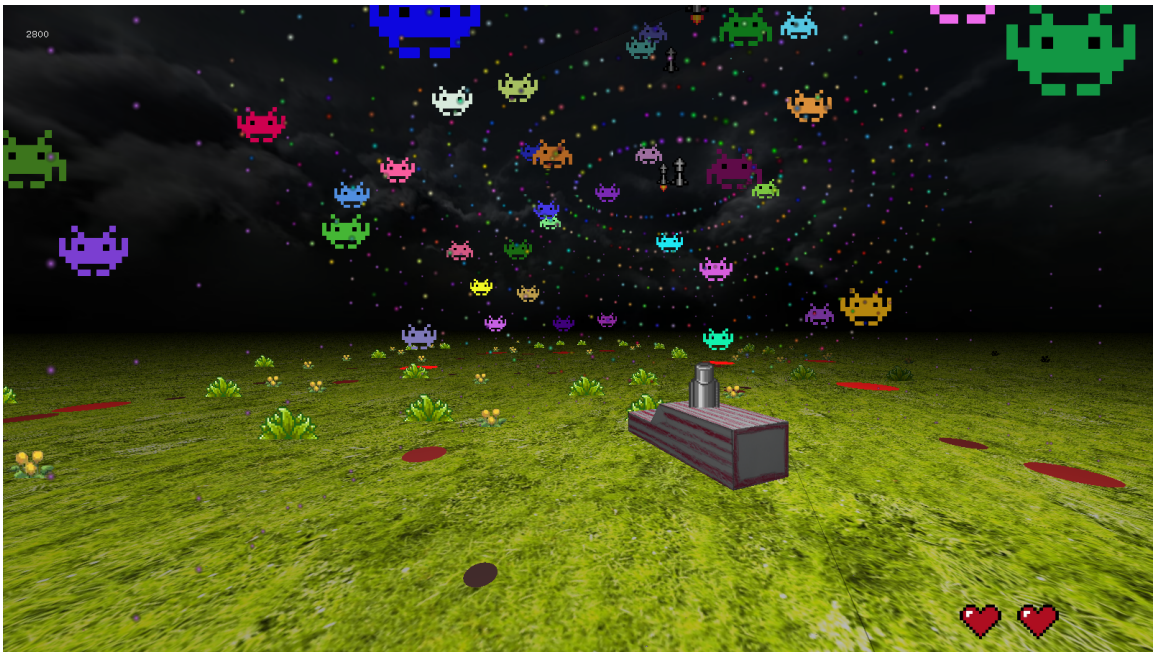


Figure 4.4: Overwhelmed.

Figure 4.5: Boss fight!

5. Discussion

5.1 Evaluation

While I was pretty succesful in most of what I was trying to achieve, I still encountered some problems during development which I would like to share along with my attempt to solve them:

Transparency

In OpenGL rendering, when an object A: 1. is in front of an object B. 2.(partially) blocks object B. 3. is drawn first. Then OpenGL will not render the part of B that is begind A. This causes no problem when A and B are solid objects, but it does when they have transparent textures, for which I have a lot of in my game. If object B is drawn first, object A is drawn over the rendered object B and causes no visual problem because the transparent parts will show the objects behind it.

Thus the game had to make sure objects furthest away from the camera are drawn first. It does this by calculating how far each object is from the camera and draw the objects from furthest to closest. This does not solve the problem completely though, as I only took the distance from the center of the objects into account, and not every vertex.

LIACS Computers

There was a problem with LIACS computers not rendering my scene correctly. I have tested other computers and laptops and they display my game perfectly, but for some odd reason all vertex normals were messed up and nothing was rendered or lighted correctly.

I changed all vertex normals to work fine and fixed the problem this way, however I couldn't figure out why this was suddenly happening only on LIACS computers. Maybe it was caused by having different versions of OpenGL, GLEW, and GLUT.

5.2 Further work

I did not implement OpenGL shaders because basic OpenGL lighting already seemed to work great for this project, however this is something to consider imlementing in the future, as this could open up some possibilities for fancier lighting effects for the fireworks.

As an actual game, Earth Invaders does not offer good gameplay. If you want to hit enemies accurately you would have to stare at the ground all the time to make sure your car is standing on a shadow of an enemy. The boss isn't all that exciting as it does not pose a bigger threat than small enemies. If this project was not just for showcasing OpenGL practices but supposed to be an actual game, there would be a lot of opportunity to make the gameplay mechanics a lot more polished.

I made the project using the old OpenGL pipeline as I was more familiar to it. As this works fine, the new pipeline (working with Vertex Buffer Objects, etc) is far more efficient and (more) up-to-date. Implementing this will improve performance and stability.

When resizing the game window, the HUD is not scaling accordingly. I chose not to do something about it as it would take a significant amount of time while I regard this problem as not in the scope of the assignment, but fixing this would help make more screen ratio's compatible.

5.3 Conclusion

Working with the old OpenGL pipeline was pretty succesful and while I had some trouble setting up things like textures at first I feel like I have a pretty good grasp of how everything works now. I am pretty excited about any follow-up knowledge regarding Computer Graphics as this course has certainly helped me getting my bearings in the field.

Thank you for reading, Jeroen Donkers