2018

[Document subtitle]

Markus Krugel and foo jia rong

Southampton Solent University | [Company address]

Contents

[Contents 1](#_Toc528915080)

[Game Description 2](#_Toc528915081)

[User stories 2](#_Toc528915082)

[Diagrams 2](#_Toc528915083)

[Class Diagram 2](#_Toc528915084)

[Flow Diagram 3](#_Toc528915085)

[Gantt Char 4](#_Toc528915086)

[Work Breakdown Structure 4](#_Toc528915087)

[3D Development Techniques 4](#_Toc528915088)

[Point Light 4](#_Toc528915089)

[Anisotropic filtering 4](#_Toc528915090)

[Mip mapping 4](#_Toc528915091)

[Deferred Rendering 4](#_Toc528915092)

[Testing Plans 4](#_Toc528915093)

[Report Markus Krugel 4](#_Toc528915094)

[Design Process 4](#_Toc528915095)

[Reflection 5](#_Toc528915096)

[Report Foo Jia Rong 5](#_Toc528915097)

[References 5](#_Toc528915098)

[Appendices 5](#_Toc528915099)

[Proof for backup methology 5](#_Toc528915100)

[Diagrams 5](#_Toc528915101)

[User Stories 5](#_Toc528915102)

[Class Diagram 5](#_Toc528915103)

[Flow Diagram 5](#_Toc528915104)

[Gantt Chart 6](#_Toc528915105)

[Work Breakdown Structure 6](#_Toc528915106)

# Game Description

*by Markus Krugel*

Our game is a 3D exploration game displayed in a third person view, where the camera is positioned behind the player, which also follows the player. References for this game are games like “Super Mario 64” and “Banjo Kazooie”.

The game starts with a start menu, where the player can select via mouse click if he wants to start the game or exit the game.

After the player selects to start the game, the level with its objects, enemies and the player will spawn. The player can move around the world using the WASD keys and space bar for jumping.

Additionally, the player can pause the game using the enter button. If the game is paused a pause menu will appear where the player can select via mouse click if they want to continue or to go back to the start menu.

Inside the world are also enemies which patrol around in certain areas and chase the player when the player gets in the sight area of the enemy, but they will also lose interest in the player when the player is too far away from him. Once the enemy arrived at the player’s position they will attack the player. The player can also attack the enemy using the “F” Key. If the enemy’s life is under zero they will die and drop an item, which can be collected by the player by simply walking over it. Otherwise if the player dies the screen will get black for a couple of seconds and a game over text will be shown. After that the player will return in the start menu.

There will be static objects, e.g. walls, in the world but also objects which can be moved around by colliding with it. Furthermore, there are objects like coins which can be collected by the player.

An optional goal is to implement NPC characters, which are talking to you when you press “E” in front of them.

# User stories

*by Foo Jia Rong*

fa

# Diagrams

## Class Diagram

*by Markus Krugel*

For every entity which will be displayed on the screen we use the “GameObject” class. Inside this class we store the basic information about the entity like position, model and the collisionBox.

The classes “Character” and ”Object” are derived from this “GameObject” class. The “Object” class is used for the objects in the game which can be static, movable or can be collected. The “Character” class stores additional values like the amount of health and the damage output. Furthermore, the class is being divided into 3 different classes: “NPC”, “Player” and “Enemy”. Whilst the “NPC” class only have their dialogue text variable and a method to execute this dialogue the “Player” class is for the player character and handles his movement, attack and other additional actions. The “Enemy” class have a simple AI in the form of a state machine. There he has his standard action, patrolling, and his second state where he chases and attacks the player.

We have a “Renderer” class to handle the rendering of the game. This class does have an object of the “HUD” and the “Camera“ so that he can factor both in the rendering process. The camera class allows us to easily create different cameras.

The “GameManager” class is the main class for the game. The class handles in every update tick the game and cares about the different aspects of the game, for example he updates the entities and handles the player input.

We have the “Collision” class to detect if different entities are colliding and to handle the outcome of collisions.

Moreover, the “Level” class stores a list of the entities, which are inside the level. By having a level class, it is easier to create different levels with their own collection of entities.

Inside the “Input” class we register the inputs of the player so that the GameManager can react to the player’s input.

The “StartMenu” and “PauseMenu” classes are for the different menus where the player can be. The PauseMenu derives from the StartMenu so that it can uses the methods for exiting the game and to start the level and add another function to continue to the game from his paused state.

## Flow Diagram

*by Markus Krugel*

The flow diagram displays the general flow we intend to implement to our game.

In the beginning we create our GameManager class, which will handle the logic of the game. Right after that we are creating our Renderer and Camera so that the player can see the game.

Thereafter the StartMenu will be created, where the player can decide between two options via mouse click. Either they choose to exit the game or they choose to start the game.

When the player decides to play the game, the level and the HUD will be created. The next step is to populate the level with the entities, like the player, enemies and the objects.

Now the update function inside the GameManager class will care about the main loop of the game. Inside this function the current state of the game will update constantly. There are three scenarios to break outside of the normal game loop.

The first option will be when the player dies. Then the screen will turn black for a couple of seconds and a game over message is being displayed. After that we are returning to the StartMenu state.

As a second option to break outside the normal game loop is to pause the game. Then the PauseMenu will be displayed and the update function of the GameManager stops to update the level. Inside the PauseMenu there will be again a decision between two options for the player. At the one hand he can go back to the StartMenu and on the other hand he can continue the game. If they choose to continue the PauseMenu will disappear and the update function will continue to do his work.

The last option is to finish the game. Thereafter the player lands once again in the StartMenu.

## Gantt Char

*by Foo Jia Rong*

## Work Breakdown Structure

*by Foo Jia Rong*

# 3D Development Techniques

## Point Light

## Anisotropic filtering

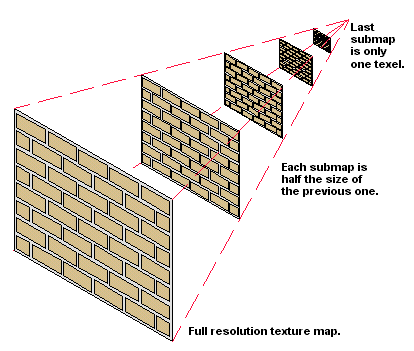
## Mip mapping

The basic concept of mip mapping is to use different sizes of a texture depending on the distance between the object and the camera. The original texture is labelled level 0. Every following level would be half of the size of the previous level.

The main use for this technique is to improve the image quality by removing the moiré patterns which appear when the texel count is higher than the pixel count to be rendered.

Another usage for this technique is to allow for a level of detail system.

The only downside of this technique is that you need 1/3 more memory per texture you are using.



There are different concepts of creating mip maps.

The first concept is called “Nearest Neighbour”. In this simple concept we are simply taking the nearest located texel colour for our the next mip level. For this method you can this formular to sample from the previous level:

*ci(x,y) = ci-1(x·2,y·2)*

or this formular to sample from the original texture:

*ci(x,y) = c0(x·2i,y·2i )*

Another concept we can use is the “Boxfilter”. Here we are using the average colour of a 2x2 pixel square for our next mip level. The formular for this looks like this:

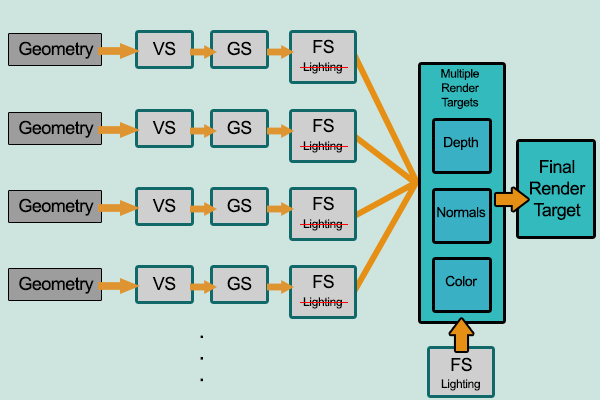
*C1(x,y) = ¼ ( ci-1(x·2,y·2) + ci-1(x·2+1,y·2) + ci-1(x·2,y·2+1) + ci-1(x·2+1,y·2+1) )*

<http://cf.ydcdn.net/latest/images/computer/MIPMAP.GIF>

## Deferred Rendering

The deferred rendering consists of two main passes. In the first pass, the “Geometry Pass” we are outputting the geometric and material data, in the form of different buffers like color, depth and normal, to the “Geometric Buffer”, also called “G-buffer”.

After that we add the lighting buffers like diffuse and specular to the G-Buffer in the “Lighting Pass”.



The main advantage of deferred rendering is that you can use many light objects in your scene without having to worry about the performance because it does not need the lighting on every object themselves.

On the other hand, the base concept of deferred rendering has multiple disadvantages, for example you cannot use transparent objects or you cannot use multiple materials. Otherwise there are different modifications of this concept to solve these problems. Like you can combine it with forward rendering which only cares about the transparent objects or you can use the deferred lighting modification to use multiple materials.

To summarize deferred rendering is optimal when want to work with many light sources and you do not want to use semi-transparent objects or different materials.

# Testing Plans

*by Markus Krugel*

One of the first aspect that we control is that our game is rendered correct. To test that we will create different simple objects in different positions, e.g. one can block the sight of another, and see if they are displayed right.

Subsequently we can test both of our key inputs and our camera at the same time. We will test if by pressing certain keys the camera is moving and rotating accordingly to our input.

To test our menus and our mouse input during the menus we can do an action when we click at certain points or objects.

For our enemy AI we will position the player and the enemy in certain distances to see if he will chase our player when they are in range. We also control then that the enemy will lose interest when the player left them behind.

When we start doing the combat we are first going to print out debug messages to see if the player and the enemy is losing life but after we implemented our HUD we also test then that the HUD will update accordingly to our changed health value.

To see if our collisions are working we will move different objects towards each other and then stop them when they are colliding.

# Report Markus Krugel

## Design Process

Initially we inspected the required tasks for this project. After that we collected our ideas of the game together and the different aspects we have to accomplish for the creation of the game, like the need for collision detection, the registration of the player input and the rendering.

Afterwards we created together the base of the class diagram and the flow of the general game. Subsequently we started to divide the tasks. While I improved the class diagram and created the flow diagram, the tasks of my partner were to create the gantt chart and the work breakdown structure. Furthermore, we both researched two 3D development techniques each. I decided to research about mip mapping and deferred rendering whereas my partner looked up about point light and anisotropic filtering.

In the beginning we send our ideas and files via mail to each other. After a while we opened up a git repository in github where we upload our files.

// user stories

## Reflection

The communication with my partner went smoothly. Both of us went to our planned meetings and even when somebody could not come he informed his partner about it and we figured out another date to meet.

For the next time I would try to start earlier to inform myself about the different 3D development techniques because of the complexity of these aspects.

One problem we had was that we collected our user stories pretty late so that we had to revise our gantt chart and the user stories so that they fit to each other.

# Report Foo Jia Rong

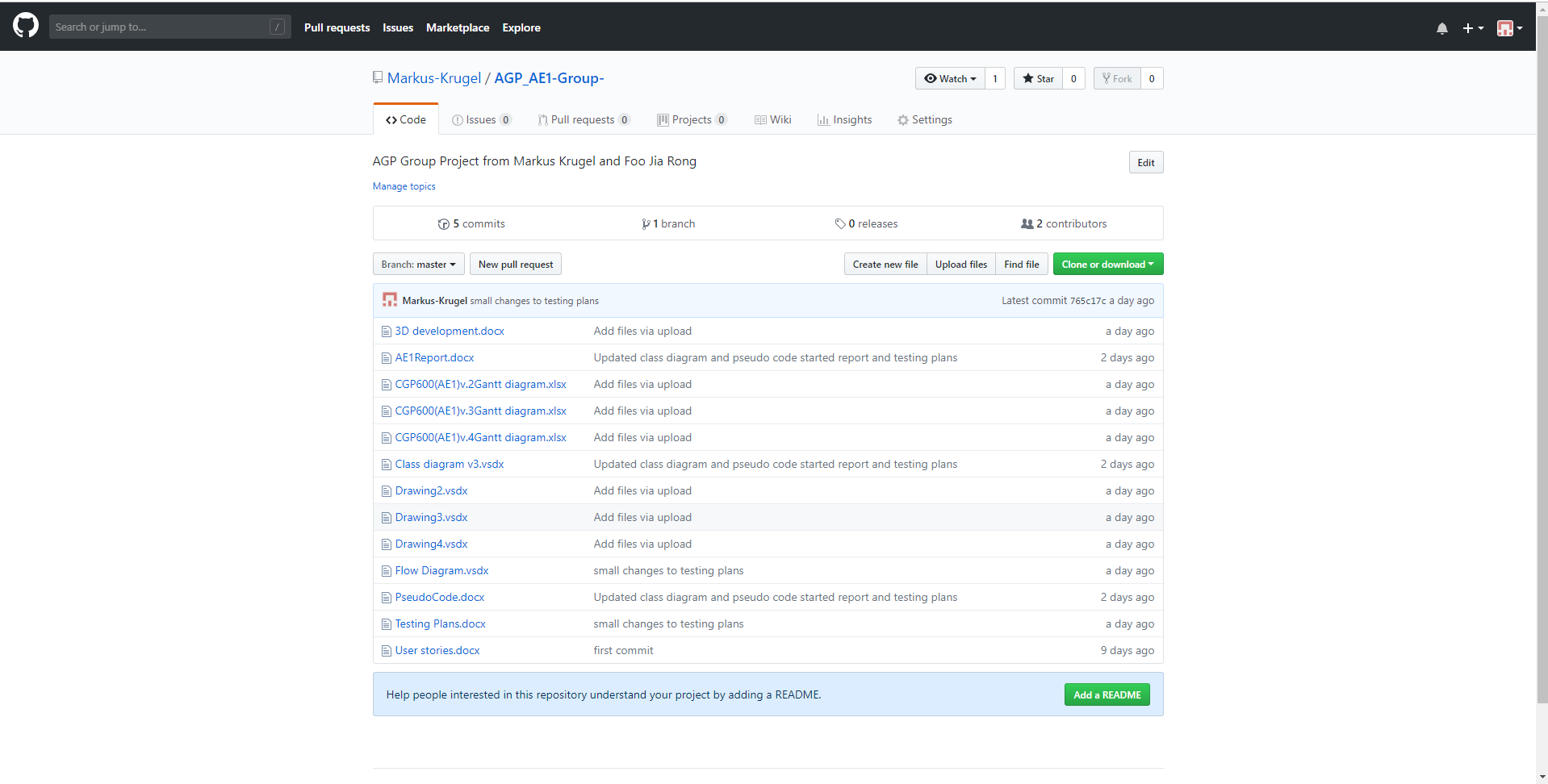
# References

# Appendices

## Proof for backup methology

Link to our Github repository:

<https://github.com/Markus-Krugel/AGP_AE1-Group->



## Diagrams

### User Stories

### Class Diagram

### Flow Diagram

### Gantt Chart

### Work Breakdown Structure