AGP AE1 Assesment

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*

user stories during the designing process have been the element of our down fall as we didn’t do the use stories in the first place and resulted in multiple redo for the Gantt chart. The project manager aspect of the game had been weak. As user stories are done after Gantt chart and work break down structure there are multiple version of a WBS and Gantt chart. We did discussion on the design one the game with the core requirement and started the project with class diagram and flow chart. By that point we knew a basic flwo and what we need for the game with that knowledge, the Gantt chart were made without user stories and

# 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

## Spot Light

Research done on how to implement spotlight had been great as there are multiple example and tutorial on the internet show how and the theory of spotlight. As follow and in the appendix will have the figure and picture showing the cone and the formula to calculate the light to be limited in a cone shape hence turning directional light to a spot light. In fig 1.1 in a representation of the cone of light and in fig1.2 will have the explanation of the light and math of the cone and light calculation.

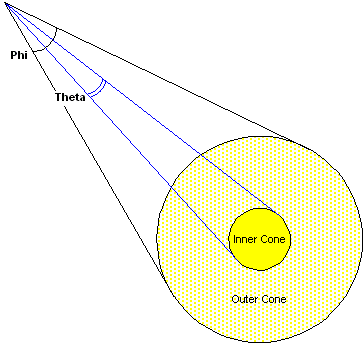


Fig1.1

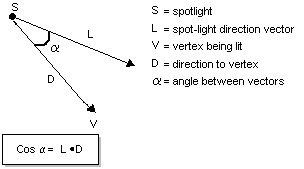


Fig1.2

finalColor \*= pow(max(dot(-lightToPixelVec, light.dir), 0.0f), light.cone);

the formula highlighted above is the formula to calculate the to define in cone light turning a point light in existing code and turning the point light to a cone and forming a spotlight. Further explanation is in appendix (1.1)

## Anisotropic filtering

anisotropic filtering is a technique use to output a better solution for texture that are going to be view at angle on object in the game or scene. Anti-aliasing is an isotropic filtering so anisotropic filtering from what the research be said is that anisotropic filtering is the opposite of anti-aliasing.

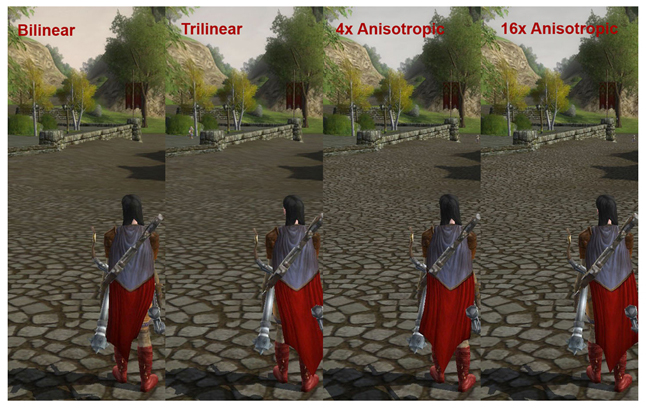


Fig2.1

## Mip mapping

## Deferred Rendering

# 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.

# 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 were the tasks of my partner 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.

// first per mail then git

// 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.

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

## Spotlight

### FIG1.1 IMAGE FROM WEBSITE ||

<https://docs.microsoft.com/en-us/windows/desktop/direct3d9/light-types> REF ON(02/11/2018)

### FIG1.2 IMAGE FROM WEBSITE ||

<https://docs.microsoft.com/en-us/windows/desktop/direct3d9/light-types> REF ON(02/11/2018)

### APPENDIX 1.1 WEBSITE ||

<https://www.braynzarsoft.net/viewtutorial/q16390-21-spotlights> REF ON(02/11/2018)

<http://www.directxtutorial.com/Lesson.aspx?lessonid=9-4-9> REF ON(02/11/2018)

<http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter05.html> REF ON(02/11/2018)

## Anisotropic Filtering

### FIG2.1 IMAGE FROM WEBSITE||

<https://www.geforce.com/whats-new/guides/aa-af-guide#1> REF ON(02/11/2018)

### APPENDIX 2.1 WEBSITE ||

<https://docs.microsoft.com/en-us/visualstudio/debugger/graphics/point-bilinear-trilinear-and-anisotropic-texture-filtering-variants?view=vs-2017>

REF ON(02/11/2018)

<https://www.braynzarsoft.net/viewtutorial/q16390-11-textures>REF ON(02/11/2018)

<https://www.geforce.com/whats-new/guides/aa-af-guide#1> REF ON(02/11/2018)

# Appendices

## Appendix 1.1

### SpotLight Factors

Spotlights are basically just point lights but with a direction. Because of this, we will build directly off our implimentation of the point light, adding only a direction, and a value representing the size of the cone of our spotlight.

Defining the Cone of Light

Here is the only new equation we will put into our effect file. This will create a cone that our light will shine through.

The first thing is find the angle between our lights direction, and the direction from the light to the pixel. We use the dot product to find this.

Then we use the max function to make sure we do not get dot product less than 0.0f. A dot product less than 0.0f would be behind the position of the spotlight, therefore shining light not only in the front of the spotlight, but also behind the spotlight.

Finally, we use the pow function with the light.cone value as the exponent. This will complete the equation which defines our cone of light.

finalColor \*= pow(max(dot(-lightToPixelVec, light.dir), 0.0f), light.cone);

|| Website-from <https://www.braynzarsoft.net/viewtutorial/q16390-21-spotlights>

### directxtutorial

“Setting up a spot light is exactly like setting up a point light, but with the new properties included. The four properties added here from the code above are phi, theta, falloff and direction. Also, the type was changed to D3DLIGHT\_SPOT.”

||Website- from <http://www.directxtutorial.com/Lesson.aspx?lessonid=9-4-9>

### Adding a Spotlight Effect

Another commonly used extension for the Basic lighting model is making the light a spotlight instead of an omnidirectional light. A spotlight cut-off angle controls the spread of the spotlight cone, as shown in Figure 5-18. Only objects within the spotlight cone receive light.

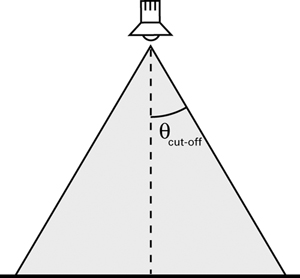


Figure 5-18 Specifying a Spotlight Cut-Off Angle

To create the spotlight cone, you need to know the spotlight position, spotlight direction, and position of the point that you are trying to shade. With this information, you can compute the vectors V (the vector from the spotlight to the vertex) and D (the direction of the spotlight), as shown in Figure 5-19.

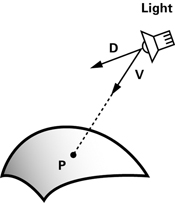


Figure 5-19 Vectors for Calculating the Spotlight Effect

By taking the dot product of the two normalized vectors, you can find the cosine of the angle between them, and use that to find out if P lies within the spotlight cone. P is affected by the spotlight only if **dot(V, D)**is greater than the cosine of the spotlight's cut-off angle.

Based on this math, we can create a function for the spotlight calculation, as shown in Example 5-8. The function **C5E8\_spotlight**returns 1 if P is within the spotlight cone, and 0 otherwise. Note that we have added **direction**(the spotlight direction—assumed to be normalized already) and **cosLightAngle**(the cosine of the spotlight's cut-off angle) to the **Light**structure from Example 5-6.

|| Website from <http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter05.html>

## Appendix 2.1

### Anisotropic Filtering

**Anisotropic Texture Filtering**  code like this: C++Copy

D3D11\_SAMPLER\_DESC sampler\_description;

// ... other sampler description setup ...

sampler\_description.Filter = D3D11\_FILTER\_ANISOTROPIC;

sampler\_description.MaxAnisotropy = 16;

d3d\_device->CreateSamplerState(&sampler\_desc, &sampler);

d3d\_context->PSSetSamplers(0, 1, &sampler

|| Website from

<https://docs.microsoft.com/en-us/visualstudio/debugger/graphics/point-bilinear-trilinear-and-anisotropic-texture-filtering-variants?view=vs-2017>

### MaxAnisotropy

“MaxAnisotropy - Clamping value used if D3D11\_FILTER\_ANISOTROPIC or D3D11\_FILTER\_COMPARISON\_ANISOTROPIC is specified in Filter. Valid values are between 1 and 16.” --||website from <https://www.braynzarsoft.net/viewtutorial/q16390-11-textures>

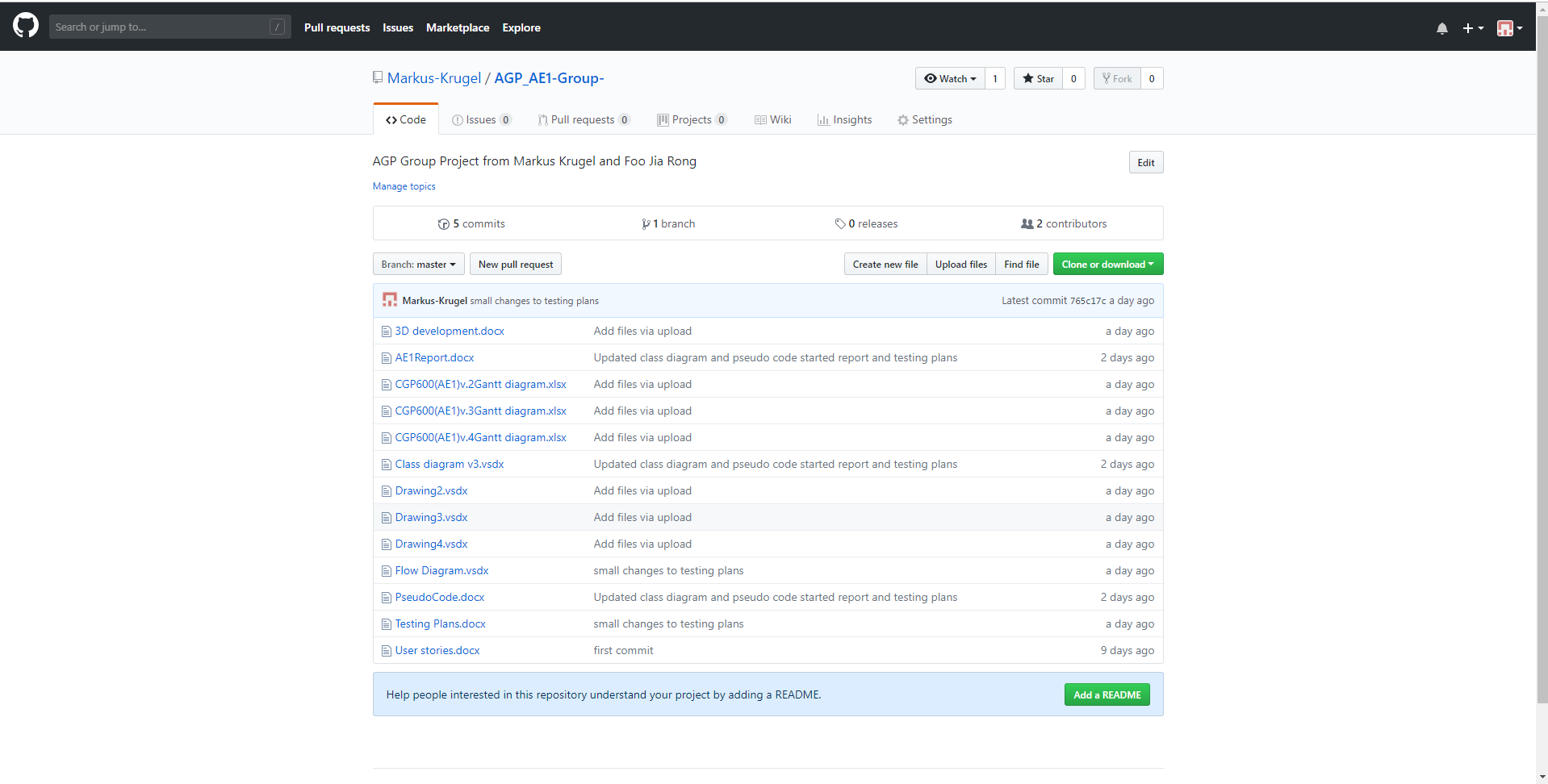
### geforce

“Anisotropic filtering exists to provide superior image quality in virtually all cases at the slight expense of performance. By the computer science definition, anisotropy is the quality of possessing dissimilar coordinate values in a space, which applies to any texture not displayed as absolutely perpendicular to the camera. As previously mentioned, bilinear and trilinear filtering suffer from resultant quality loss when the sampled textures are oblique with the camera due to both methods obtaining texel samples from mipmaps assuming that the mapped texel is perfectly square in the rendered space, which is rarely true. This quality loss is also related to the fact that mipmaps are isotropic, or possessing identical dimensions, so when a texel is trapezoidal there is insufficient sampling in both directions. To solve this, anisotropic filtering scales either the height or width of a mipmap by a ratio relative to the perspective distortion of the texture; the ratio is dependent on the maximum sampling value specified, followed by taking the appropriate samples. AF can function with anisotropy levels between 1 (no scaling) and 16, defining the maximum degree which a mipmap can be scaled by, but AF is commonly offered to the user in powers of two: 2x, 4x, 8x, and 16x. The difference between these settings is the maximum angle that AF will filter the texture by. For example: 4x will filter textures at angles twice as steep as 2x, but will still apply standard 2x filtering to textures within the 2x range to optimize performance. There are subjective diminishing returns with the use of higher AF settings because the angles at which they are applied become exponentially rarer.”--|| Website from <https://www.geforce.com/whats-new/guides/aa-af-guide#1>

## Proof for backup methology

Link to our Github repository:

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



## Diagrams

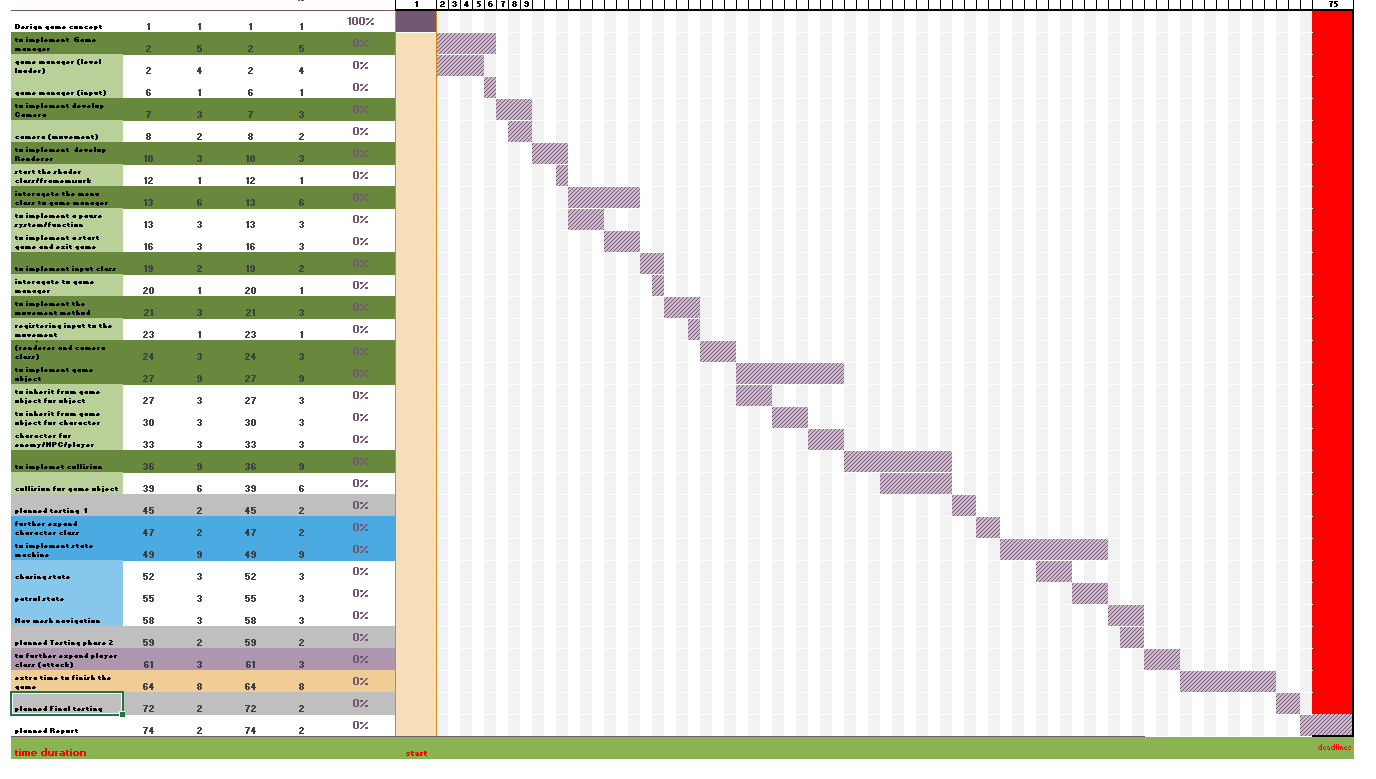
### User Stories



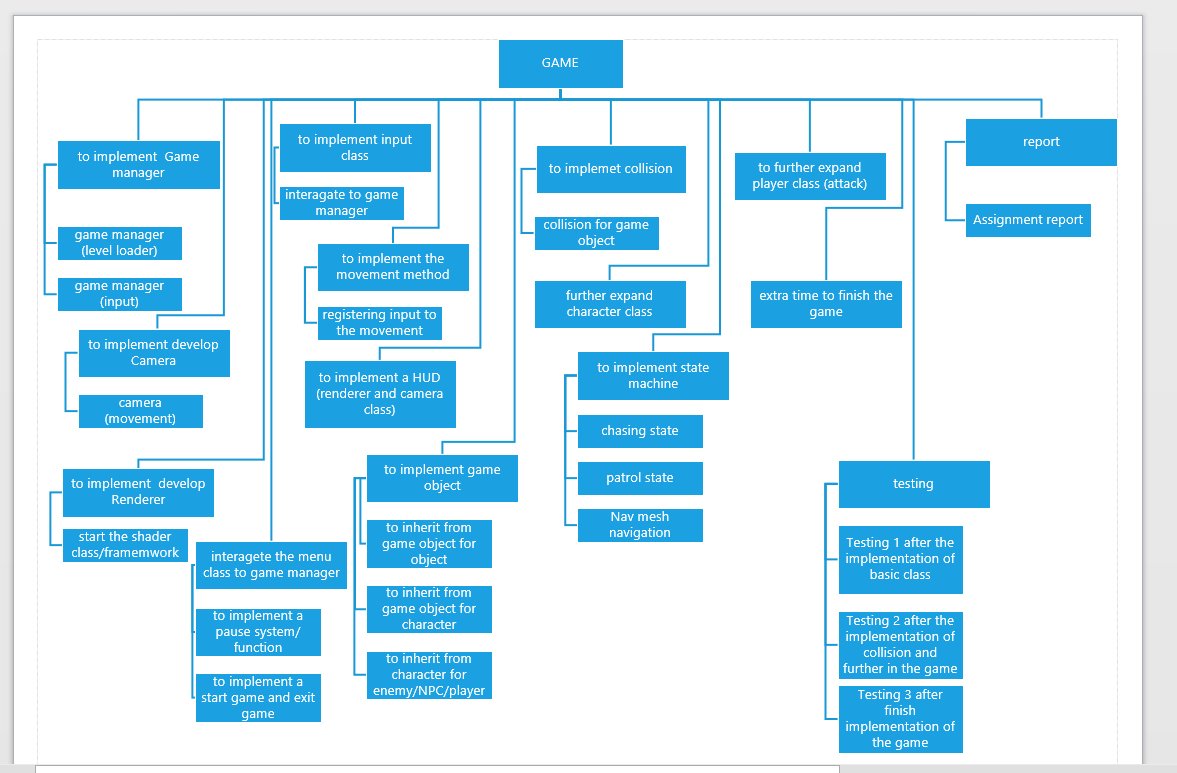
### Class Diagram

### Flow Diagram

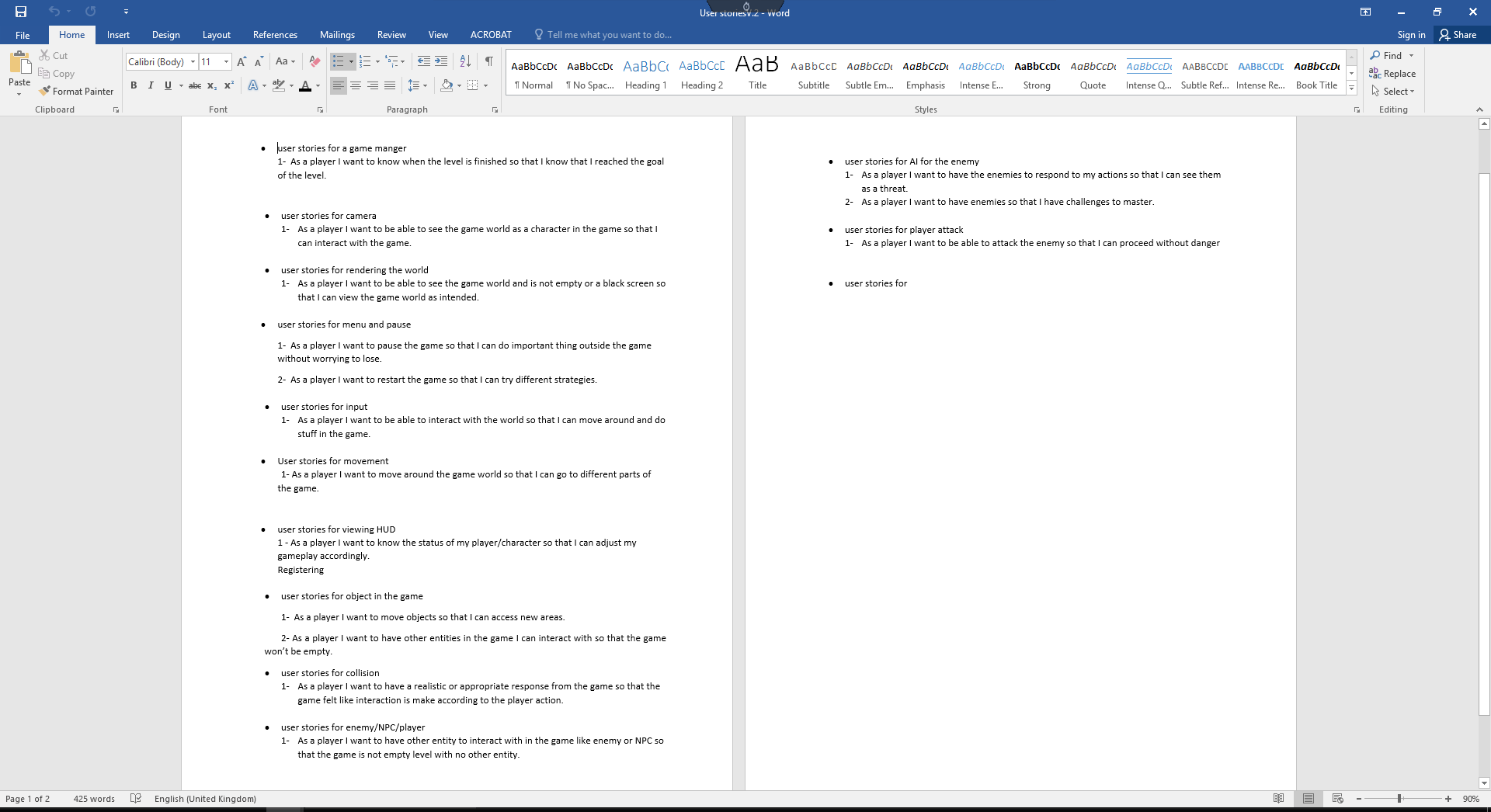
### Gantt Chart



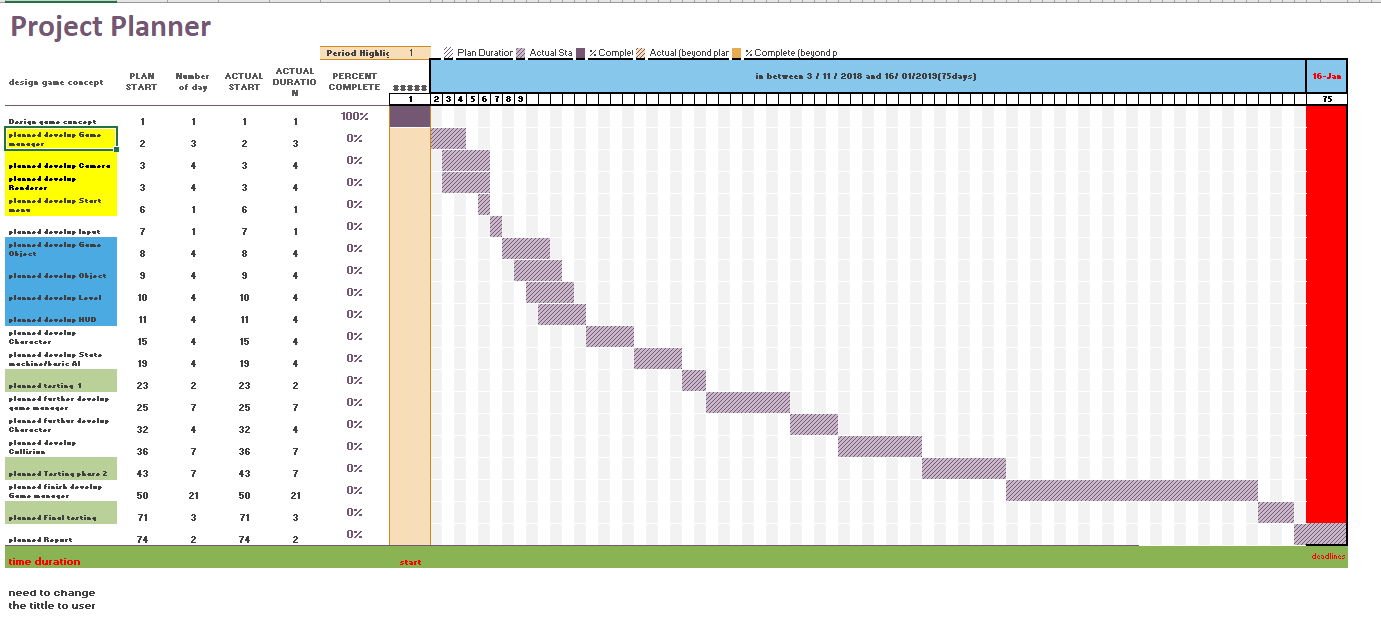
### Work Breakdown Structure



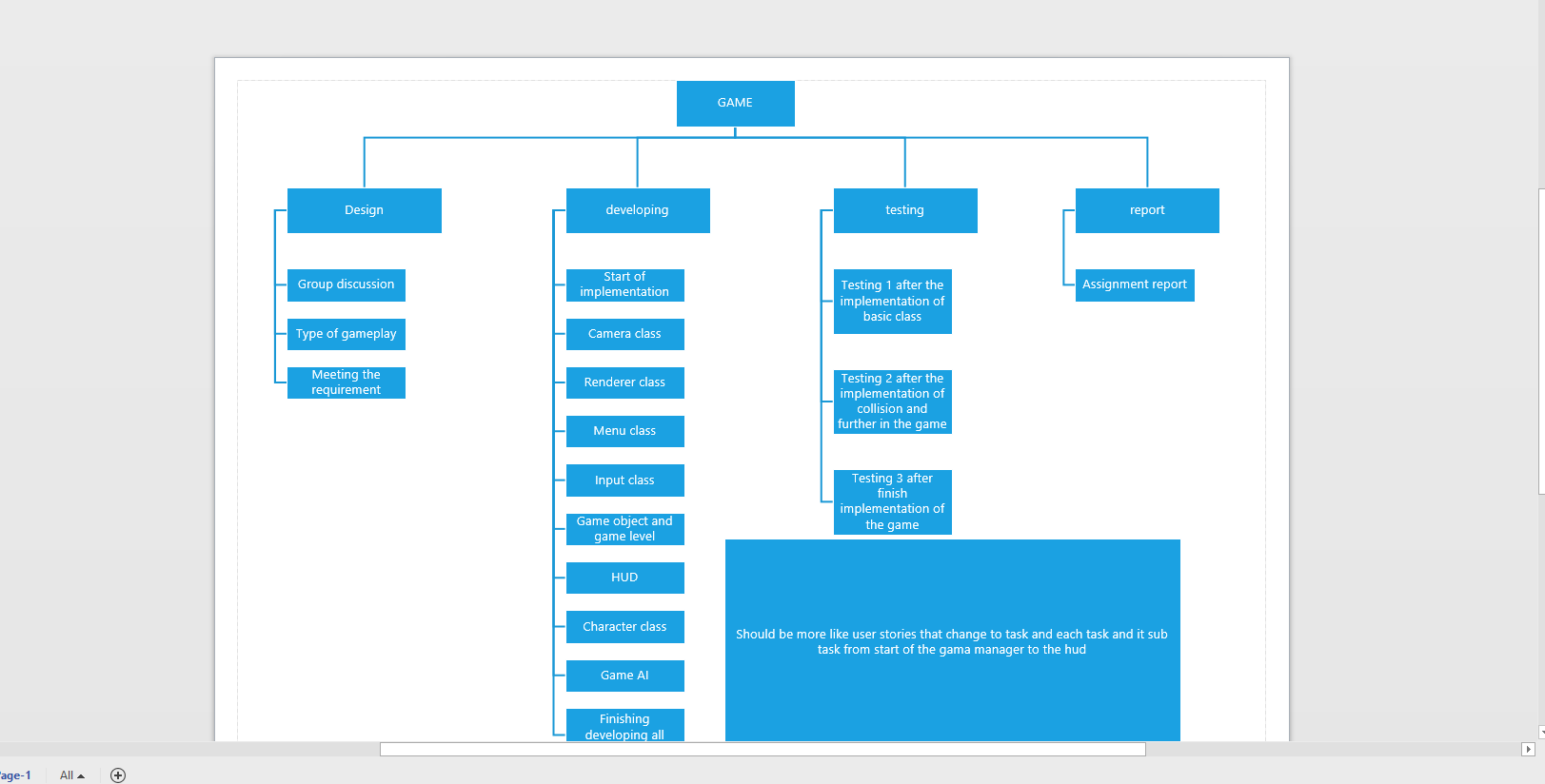
### fail version of the previos diagram



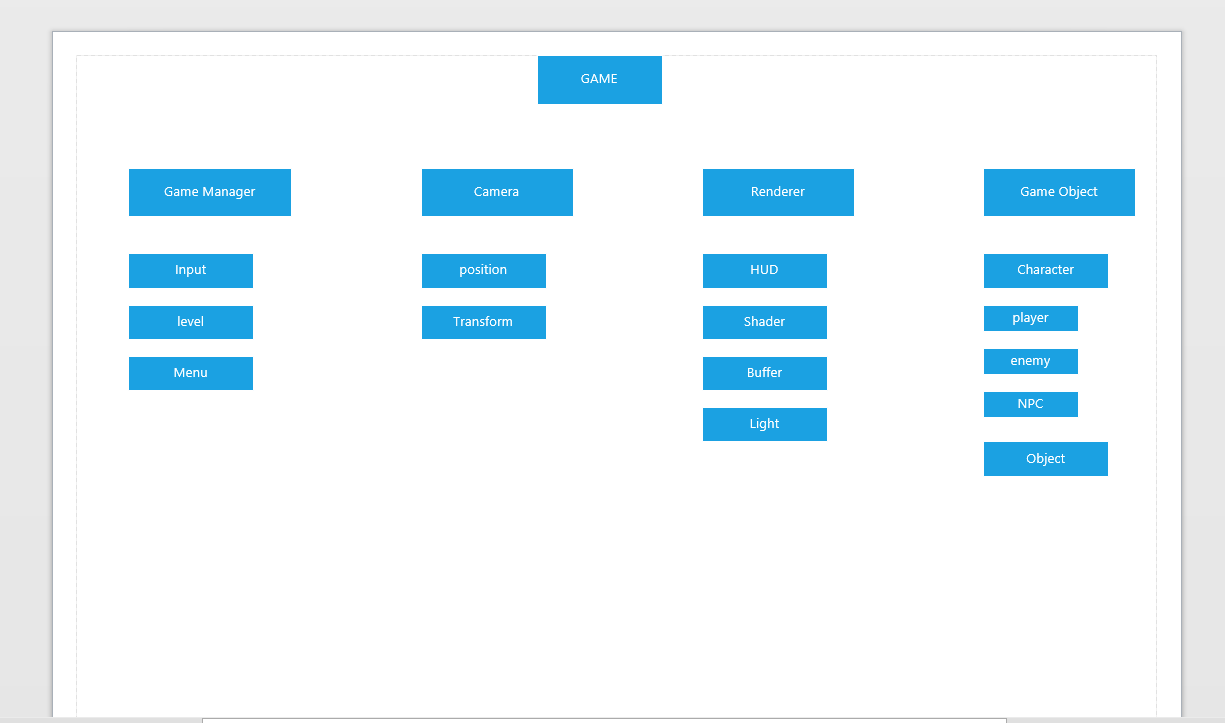
User stories



Gantt chart



Work breakdown structure



Work breakdown structure