Group 8

University of Hull  [Company address]

08356 – Games design and advanced graphics – report – Group 8

# Introduction

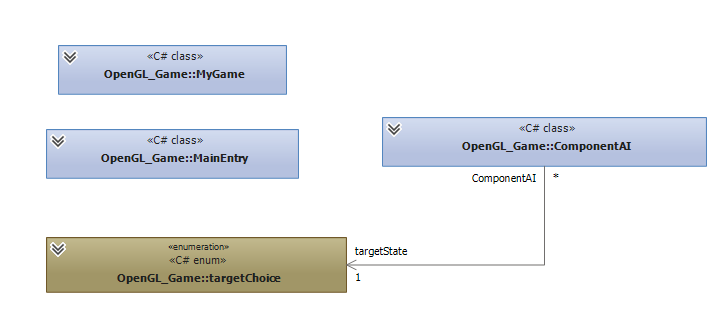
The following report will look at the creation and implementation of the 3d maze game created by our group, it will look at the design of our framework, an evaluation of how things went and also a discussion of the AI aspect of our report.

# Design

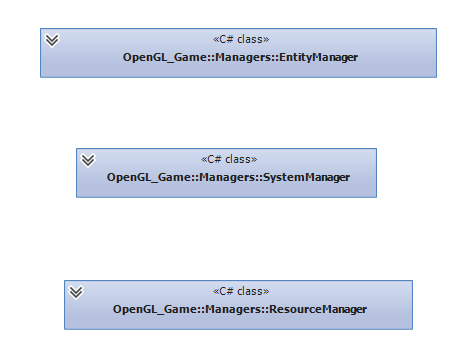
## UML Diagrams

This section of the report will show a collection of UML diagrams that form the basis of our code and will also provide an image of any classes that we have added to the framework ourselves.

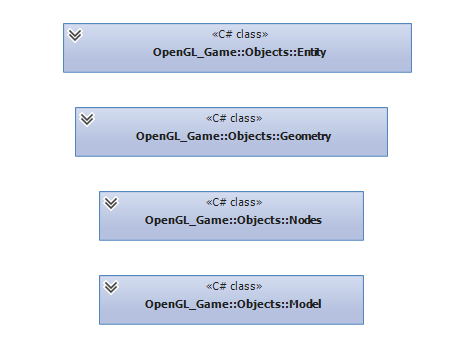
### OpenGL\_Game



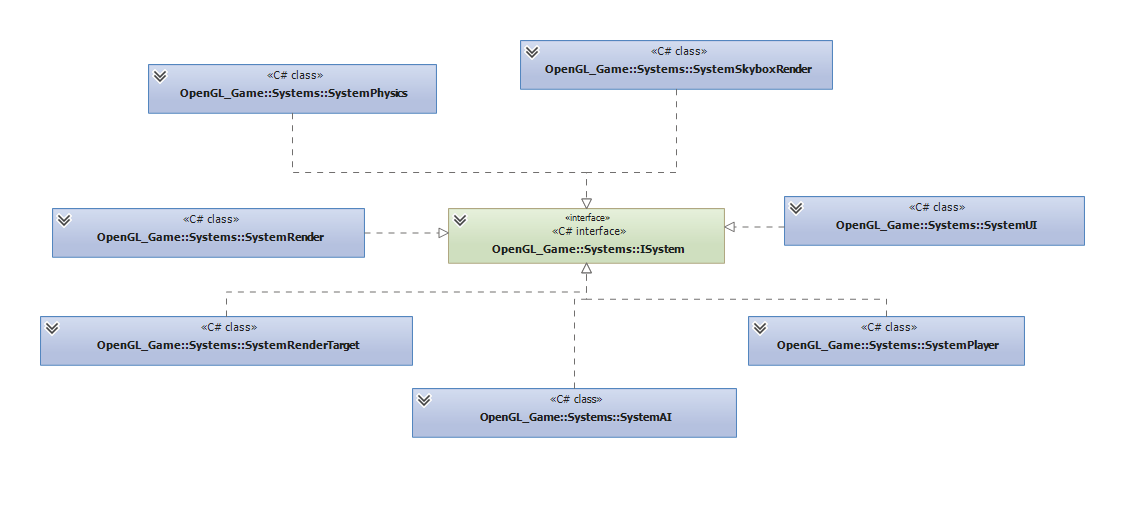
### OpenGl\_Game.Managers



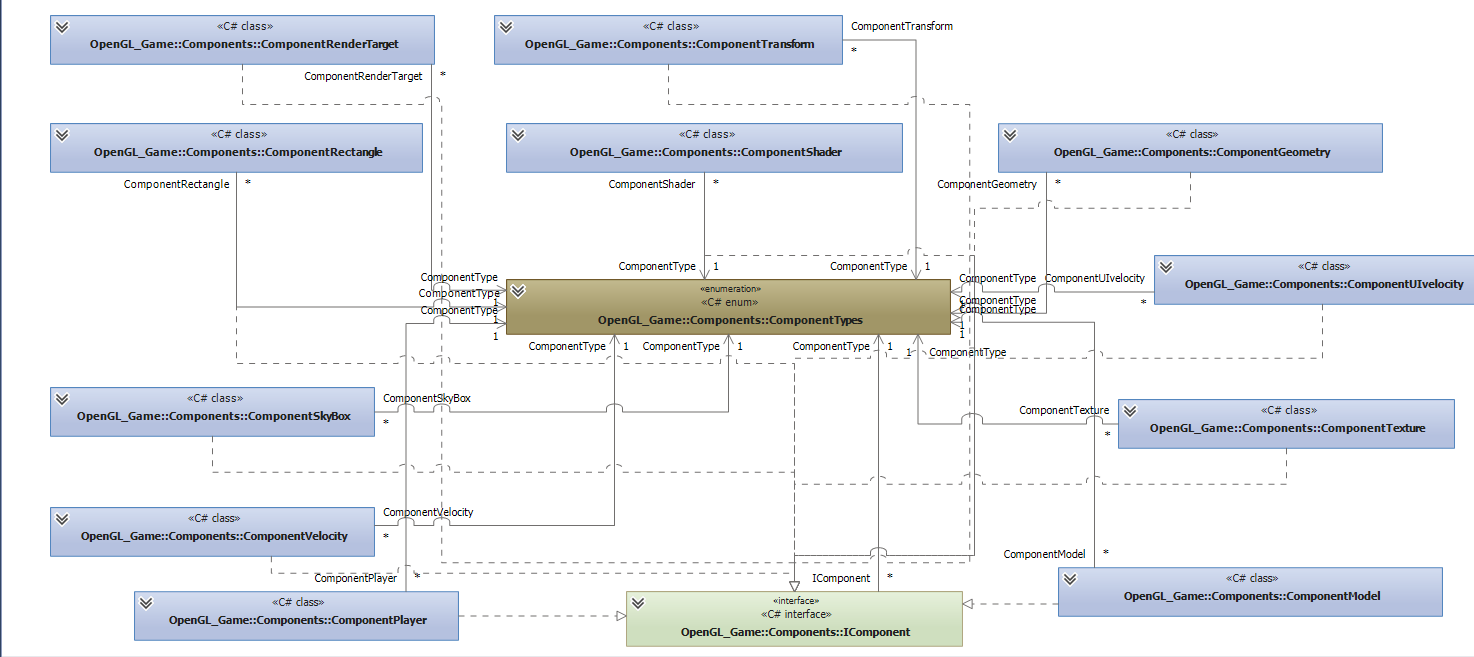
### OpenGL\_Game.Objects



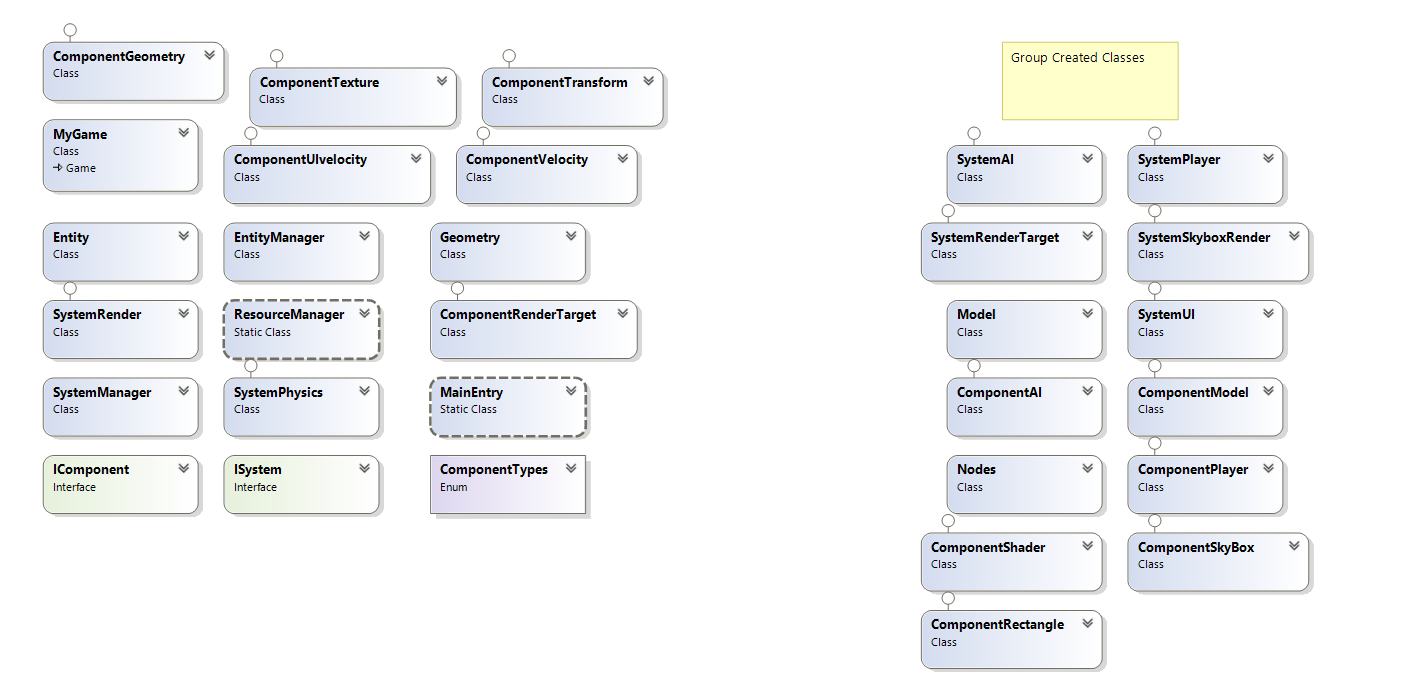
### OpenGL\_Game.Systems



### OpenGL\_Game.Component



### Full Class Diagram with Group Created Classes Shown



## Details of Classes We Added and the Reason for Their Inclusion in Our Design

This section of the report will discuss the classes that we have created ourselves, our reason for adding them into the game and how they improve the gameplay experience.

### Component & System AI

We created the component and system AI classes to hold all of the behaviour for our created artificial intelligence. The class itself holds the pathfinding data for the AI so that it moves around the map realistically and \*\*\*\*how it should, it also holds the data for how the AI should change behaviour when it sees the player, ensuring that the AI chases the player character when a user has been seen instead of continuing on the same path.

The system AI class also holds the function for updating bullets that will be fired by the AI robot, to ensure that when the player is seen it will be shot at, causing them to lose health, and also ensures that the robot doesn’t just move around the map firing at random intervals which could break the immersion for the user.

### System Player & Component Player

The system player and component player classes hold all of the information that is used by the player, the first thing that the system player class holds is the player’s health and ensures that when a player is shot by the AI they lose some health. It also holds the game over behaviour so that when a player’s health is reduced to zero they are greeted with the game over screen.

As well as the player health they also hold the information for the various keys available around the map, and also holds the function that will give the player additional keys that will be added to their inventory when they pick them up, adding them to the inventory and removing one from the game. When picking up the keys the Player function also checks to see if the player has collected all of the keys, if the player has all of the keys then the portal will open allowing them to complete the game.

### System Render Target

The render target was originally placed into the game due to the fact we were originally going to create the mini-map using a camera from the top-down view. We didn’t actually complete the mini-map using this class but decided to keep it in the game in case we could use it for something in the future as it could be a useful tool.

### System SkyBox Render

System SkyBox Render is used to render the skybox that we have created, the reason that we created a new render for the SkyBox was because it needed to take in ComponentSkybox to get our SkyBox top generate properly. This was essential to create the SkyBox for our game.

### Model & Component Model

The Model and Component classes that we have created holds all of our models and is used to load them in, it was essential that we created this so that the game was able to load all of our models in and create the atmosphere that we desired.

### System UI

The system UI class within our game holds all of the information that will that is displayed on the screen for the user, this includes the visual representation of the keys and the health which are both very important as they provide visual feedback to the user about how well they are doing.

It also holds the information for the mini-map which shows the user where they are and the location of the various keys around the map, this is important for the user to show the various positions and also provide feedback for the user.

### Nodes

The nodes class that we have created is essential because it holds the position of all of the nodes as well as all of the other nodes that it is connected too within a list, this is essential so that the AI knows where to move when it is moving around the map and that the pathfinding will work correctly so that it is always moving towards the nearest node.

### Component Shader

The Shader class that we created was implemented so that we could create our own shaders and implement them in MonoGame, this was essential so that we could create and implement our shaders and get the effects that we desired.

### Component SkyBox

The SkyBox class hold all of the information for our SkyBox that we have implemented, it holds the information for the faces that go into an array and are then implemented onto a textured cube. This provides us with our SkyBox which helps to add to the atmosphere of the game.

### Component Rectangle

The Rectangle class that we have holds a rectangle that we use when creating the UI as in order to draw a 2D element into our user interface window we first had to draw a rectangle.

### Libraries

We decided not to include libraries in our project as many classes were reliant on the main class of our project (MyGame) either directly or through a class it was associated with.

## Entity Diagrams

|  |
| --- |
| Player |
| ComponentTransform |
| ComponentPlayer |

|  |
| --- |
| Floor |
| ComponentGeometry |
| ComponentTexture |
| ComponentShader |

|  |
| --- |
| Wall |
| ComponentTransform |
| ComponentGeometry |
| ComponentTexture |
| ComponentShader |

|  |
| --- |
| SkyBox |
| ComponentTransform |
| ComponentGeometry |
| ComponentShader |
| ComponentSkyBox |

|  |
| --- |
| Portal |
| ComponentTransform |
| ComponentGeometry |
| ComponentTexture |
| ComponentShader |

|  |
| --- |
| RollingSquare |
| ComponentTransform |
| ComponentGeometry |
| ComponentTexture |
| ComposnentVelocity |
| ComponentShader |

|  |
| --- |
| BouncingOctahedron |
| ComponentTransform |
| ComponentGeometry |
| ComponentTexture |
| ComposnentVelocity |
| ComponentShader |

|  |
| --- |
| AI |
| ComponentTransform |
| ComponentGeometry |
| ComponentTexture |
| ComposnentShader |
| ComponentAI |

|  |
| --- |
| Minimap |
| ComponentTransform |
| ComponentTexture |
| ComponentRectangle |

|  |
| --- |
| MMKey |
| ComponentTransform |
| ComponentTexture |
| ComponentRectangle |

|  |
| --- |
| Health |
| ComponentTransform |
| ComponentTexture |
| ComponentRectangle |

|  |
| --- |
| Dot |
| ComponentTransform |
| ComponentTexture |
| ComponentRectangle |
| ComponentUIvelocity |

|  |
| --- |
| Bullet |
| ComponentTransform |
| ComponentGeometry |
| ComponentTexture |
| ComposnentVelocity |
| ComponentShader |

|  |
| --- |
| Key |
| ComponentTransform |
| ComponentModel |
| ComponentTexture |
| ComponentShader |

# Evaluation

The first main issue that we ran into when putting our game together was getting the wall and floor models to look accurate when we were putting them together. Initially when we looked into rendering the floors and walls we had problems with getting them to render in certain positions and also with them not being the correct size

To help make the implementation of the models within the game more accurate we decided to create a whole new component that would allow us to transform the object to our liking. The new component was called ‘ComponentTransform’ and allowed us to position, rotate and scale any object to make them more accurate to what we desired, helping to create a more realistic and more immersive maze, and also assisted of the creation of any future components making it a lot easier as we progressed.

The main problem that we ran into in the creation of our game was a stutter that happened within our game every two frames, it was most noticeable when moving around in the game and was something that definitely ruined the immersion for any potential user. Throughout the creation of the project we tried several different methods to try and remedy this issue, the first of which was to alter the amount of entities that we were using as we felt like even though it shouldn’t have been an issue due to the strength of the machines we were using both at home and at university it could have potentially been an issue with the machines having to load several entities in. After we lowered the amount of entities and testing on several different machines we found that it did improve the skipping issue however it was still present which told us that it was a problem with the code that we had and not the entities themselves, meaning that further investigation needed to diagnose the problem

The next step we took was to try and solve this problem was to use the console to check that the skipping was as consistent as it seemed when we were playing the game, the console showed that the game was performing as it should up until the skip and then after it had occurred and the skipping itself was consistent whether the player was moving or not, this led us to believe that the issue was to do with not the amount of entities but how they are being drawn which helped us focus our search for the issue.

As we now had an idea of a possible location of the problem we then began to search through the draw, commenting areas of code out to see if they improved our issue taking a trial and error approach to trying to solve the issue, and we eventually decided to change our basic effect to be an effect with shader, which at first seemed to fix the issue at first, but when we began to add the entities back in we found that the skip reappeared, although it was a lot more noticeable.

We then proceeded to look into the code and change the rasterization of our code so that it wasn’t something that happened on every single draw, which fixed our problem completely and removed the stuttering from our game.

Another issue that we ran into when creating the game arose when we tried to create the SkyBox and implement it into our game. We originally tried to implement a texture onto an entire SkyBox that would be placed over our game however we had problems with this due to the fact that cube mapping isn’t fully supported by the MonoGame framework.

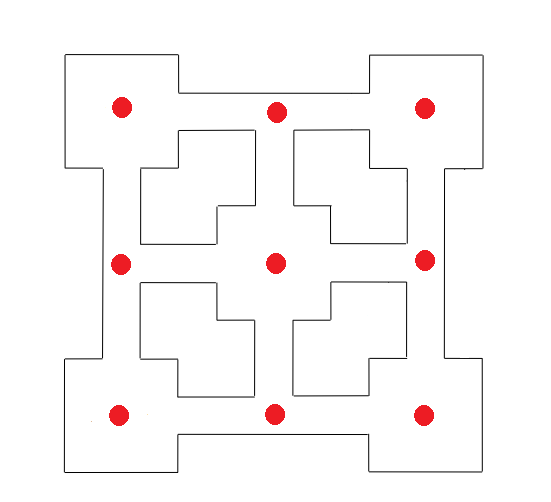
To overcome this we instead mapped the texture on to individual planes and attached them together to create a cube, which provided us with the effect that we desired without the problems that we were having when trying to implement a cube map.

# AI

The following section of the report will discuss the artificial intelligence that we have implemented into our game.

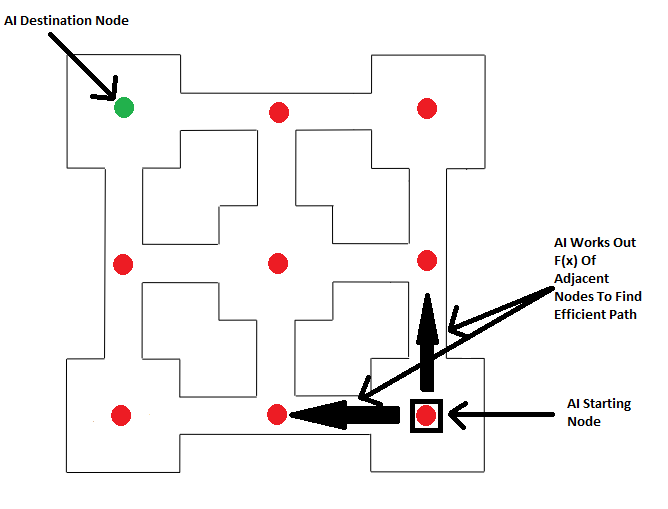
Our artificial intelligence follows a node based system which uses a simplified A\* algorithm to create the best possible path available for our robot when it is moving around the map that we have created.

The AI that we have created will start on a certain node that have been placed strategically around the map and will be given an end destination node from a list that it needs to reach to complete its objective.

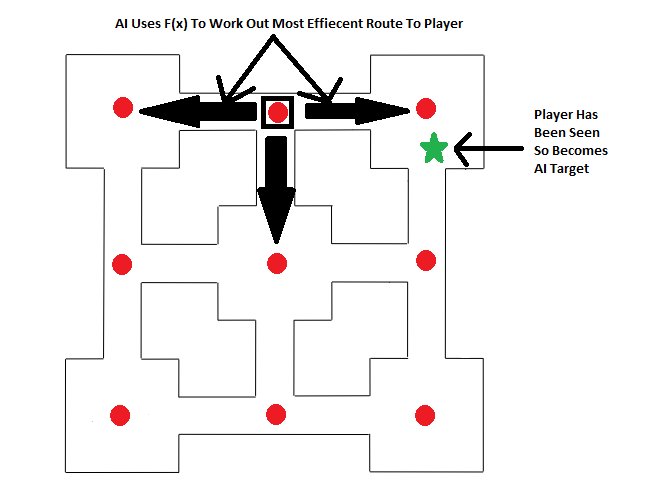


The AI, working from its current node will then work out the f(x) of all of the nodes that are adjacent to it and will then select the node that has the lowest distance cost(the closest to it) and move towards it, ensuring that it takes the most efficient path possible to reach its destination.

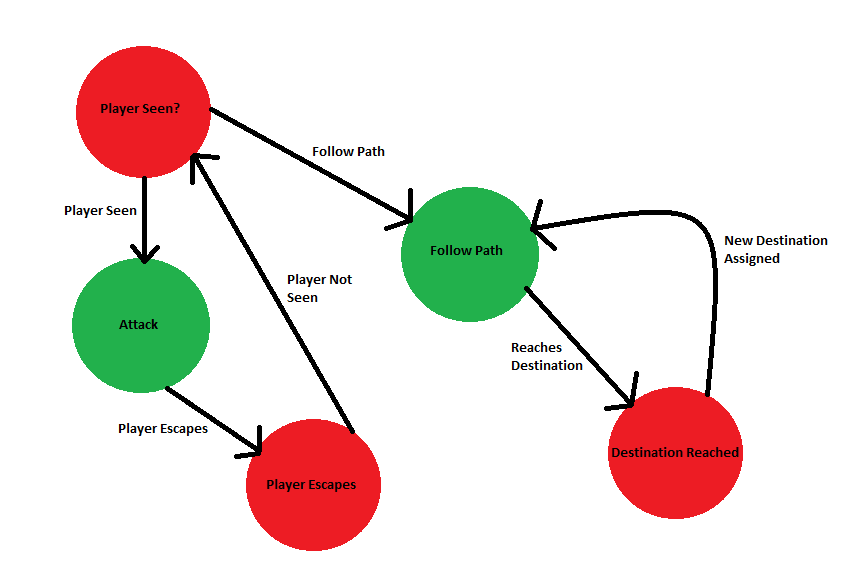
When the AI has reached its destination it will then be assigned a new end node which will be taken from the list and will then repeat this process until it once again reaches its desired location, this behaviour will be repeated until the AI’s motivation will change, which will be when it has the players character in its vision.



When the AI has seen the character its characteristics will change as it looks to lower the players health, it will look at the player position and work out the node closest to it using its f(x), it will then move towards the player following the closest nodes to them each time, until it either kills the player(which results in a game over) or the player manages to escape the AI’s vision for a set amount of time.



If the AI player manages to escape the players vision for a large amount of time then the AI will return to the closest node and once again begin its journey to the destination node working out the f(x) of the nearest adjacent node, until it either reaches its target or once again sees the player.



In evaluation we believe that the AI that we have implemented is one that is good for the game that we have created due to the shape of our map and the function that we need from our AI robot. Our map follows a square grid which means that the robot itself will always be heading towards the node closest to it and that the direction it takes will be the most efficient. However if the map was changed so that it wasn’t a set grid we may run into some issues, for example if we were to put a curved tunnel that led directly to the final node the AI wouldn’t take it into account due to the node positioning within the tunnel which could lead to it not taking the most efficient route

# Conclusion

This report has looked at the creation and implementation of the game that we have created, the framework, how the project went and how we overcame any problems that arose, and also how we implemented the AI.

# References

http://quest3d.com/forum/index.php?topic=64383.0;wap2

http://www.3dgep.com/texturing-lighting-directx-11/#Spotlights

http://rbwhitaker.wikidot.com/skyboxes-1

https://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.texturecube.aspx