CG

CGP600 – AE2 Individual project

Ethan Bruins – Q12192287

Solent University

Computer Games Software Development

Contents

[Introduction 3](#_Toc535326806)

[Implementation 3](#_Toc535326807)

[Camera 3](#_Toc535326808)

[Collision 3](#_Toc535326809)

[Level Generation 3](#_Toc535326810)

[Lighting 4](#_Toc535326811)

[Dissolve Shader 4](#_Toc535326812)

[Reflection Model 4](#_Toc535326813)

[Enemy Pathfinding 4](#_Toc535326814)

[Mini-map 4](#_Toc535326815)

[Testing 4](#_Toc535326816)

[Ad-hoc testing table 5](#_Toc535326817)

[Conclusion 7](#_Toc535326818)

[References 8](#_Toc535326819)

[Software Design Document – Appendix A 8](#_Toc535326820)

[Game Design – Ethan and Paulo 8](#_Toc535326821)

[Game Brief 8](#_Toc535326822)

[Game Inspirations 8](#_Toc535326823)

[Bibliography - Appendix B 10](#_Toc535326824)

[Source Code – Appendix C 10](#_Toc535326825)

[Camera.h 10](#_Toc535326826)

[Camera.cpp 11](#_Toc535326827)

[Enemy.h 16](#_Toc535326828)

[Enemy.cpp 16](#_Toc535326829)

[Entity.h 18](#_Toc535326830)

[Entity.cpp 18](#_Toc535326831)

[GameManager.h 19](#_Toc535326832)

[GameManager.cpp 21](#_Toc535326833)

[InputHandler.h 36](#_Toc535326834)

[InputHandler.cpp 36](#_Toc535326835)

[LightManager.h 38](#_Toc535326836)

[LightManager.cpp 40](#_Toc535326837)

[main.h 40](#_Toc535326838)

[main.cpp 41](#_Toc535326839)

[maths.h 44](#_Toc535326840)

[maths.cpp 45](#_Toc535326841)

[MiniMap.h 48](#_Toc535326842)

[MiniMap.cpp 49](#_Toc535326843)

[Model.h 52](#_Toc535326844)

[Model.cpp 55](#_Toc535326845)

[Movable.h 63](#_Toc535326846)

[Movable.h 63](#_Toc535326847)

[objfilemodel.h 64](#_Toc535326848)

[objfilemodel.cpp 65](#_Toc535326849)

[ParticleFactory.h 71](#_Toc535326850)

[ParticleFactory.cpp 73](#_Toc535326851)

[Player.h 82](#_Toc535326852)

[Player.cpp 83](#_Toc535326853)

[Scene\_Node.h 84](#_Toc535326854)

[Scene\_Node.cpp 86](#_Toc535326855)

[SkyBox.h 97](#_Toc535326856)

[SkyBox.cpp 98](#_Toc535326857)

[text2D.h 103](#_Toc535326858)

[text2D.cpp 104](#_Toc535326859)

[TimeHandler.h 109](#_Toc535326860)

[TimeHanlder.cpp 109](#_Toc535326861)

[Dissolve\_Shader.hlsl 110](#_Toc535326862)

[MiniMap\_Shader.hlsl 112](#_Toc535326863)

[Model\_shader.hlsl 112](#_Toc535326864)

[Particle\_Shader.hlsl 115](#_Toc535326865)

[Reflect\_Shader.hlsl 115](#_Toc535326866)

[SkyBox\_Shader.hlsl 116](#_Toc535326867)

[Text2d\_Shaders.hlsl 117](#_Toc535326868)

# Introduction

In this report I will go into detail about the project I created in C++ with DirectX 11 for advanced games programming. The original design for this project can be found in the appendix under [Appendix A](#_Software_Design_Document).

# Implementation

In this section I will go into detail about the parts that I implemented in this project. The source control for this project is at the following link: <https://github.com/Westerveld/CGP600_AE2>. Bear in mind that it is a private repository, so you need me (Ethan Bruins) to either log into the repository, or add you as a collaborator.

## Camera

For the camera, I had designed the game to use a third person perspective. The way I did this was to determine a target for the camera to look at. I then used this target to determine the cameras position and rotation. I also used the target to calculate the forward, right and up vectors relative to the camera. This would allow the player to use the cameras forward direction to move the player forward in relative space.

An additionally camera I made was a top down perspective one. This will be used with the mini map UI that I had previously talked about in my software design in [Appendix A.](#_Additional_Functionality)

## Collision

For my collision implementation I didn’t want to just have spherical colliders as this would result in a very clunky game if I had custom models. Instead I’ve gone for a joint Axis Aligned Boundary Box and Spherical collision. This allows me to have some freedom as to what type of colliding each object has. For implementation I took my knowledge of AABB collision and combined it with Sphere collision detection to allow for both collision types to collide with each other.

*No collision* *Collision Detected*

The way it works is we check which point on the cube is closest to the sphere, either its min or max. and then we calculate the square of that distance. We then added up each of these values and compare it to the square of the radius of the sphere. This then returns true if there is an overlap.

## Level Generation

For my level generation, I decided to have a text file for input. This text file would be read when the level needed to be generated and would consist of several letters and symbols. I then converted these letters and symbols to represent parts of the world such as the enemies, player and walls. This was done using an ‘ifstream’ and a vector of strings to store the input into. I then iterated through the vector to determine where everything should go.

## Lighting

For my lighting, initially I had only ambient lighting and directional. For my ambient and directional lights, I used the Doron Feinstein’s (2013) version which incorporated the Blinn-Phong lighting equation to give a more realistic look to the lights. I also tried to incorporate both point and spot lights into the project, but I was unable to correctly implement them.

## Dissolve Shader

I wanted to incorporate a bit more into my shaders than just having lighting. As I had created a dissolve shader in Unity, I thought I would give it a go in DirectX. I used the HLSL function ‘clip()’ to discard pixels from being rendered if a threshold was met. I also went a step further and added an emissive edge to the dissolve. This allowed me to create an effect like the one Federico (2018) did in his tutorial. This dissolve shader was used within my model class for models of type dissolve

## Reflection Model

I took the original code provided for the skybox and went on to create a reflection model. The way I did this was to add another model type to my class called shiny. I then added a second texture to the shader to allow for the skybox to be sent over. I then multiplied these two textures to blend the textures together to allow for both the original texture and the skybox to be seen onto the sphere.

## Enemy Pathfinding

For the enemy’s pathfinding, I set up a simple waypoint system. This system would check the distance between the enemy and the next waypoint. When they were close to the next waypoint the current waypoint value would be increased. If the current waypoint value was over the total waypoint count, the waypoint value would be set to 0. The pseudocode for this function looks like the following:

|  |
| --- |
| Check if we are at a waypoint  If we are, change our waypoint to the next one in the list  If we aren’t, move towards our current waypoint  Update our collision tree  Check if we are colliding with anything  If we are, move slightly back from our last movement |

## Mini-map

I took the time to try and make a mini-map show on the HUD, but unfortunately was unable to get it to render. I knew that I had to get the renderer to render to a texture and then display that to the screen. But I was unable to get this implementation to work correctly. If I had known this would be an issue, I would have allotted more time to this feature.

## Movable Objects

I was able to add in movable objects by adding a trigger Boolean to the scene node class. I then only checked for a collision if the scene node wasn’t a trigger. I also added another function to the scene node, that allowed me to return which scene node was colliding with the current object. I used this to then push the object away from the colliding object when they collided.

# Testing

For my testing, I will be using a couple of different testing methods. Firstly, I will use an ad-hoc test table for each of the functions I create. After I have finished programming the game, I will then use a black box testing method to ensure it does what is necessary.

## Ad-hoc testing table

In this table I will go through the functions created in my program to ensure they give the desired result. If they do not, I will make a note of it and correct the function.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Function to Test | Expected Result | Actual Result | Changes to be made |
| 1 | GameManager::InitialiseGraphics() | Set up the graphics for the program | Correctly sets up the graphics for the direct x program |  |
| 2 | GameManager::SetUpDirectX() | Set up Direct X for the program | Correctly sets up the directx device, context, back buffer and z buffer |  |
| 3 | GameManager::CheckInputs() | Check for inputs and distribute them to the classes | Uses the input class to determine which button is pressed. Also handles mouse position and button press |  |
| 4 | GameManager::Update() | Update the logic of all classes | Calls the classes that have game logics update functions |  |
| 5 | GameManager::Render() | Render all the models, skybox and text to screen | Renders models updated locations |  |
| 6 | Scene\_Node::AddChildNode() | Add the inputted node to the children vector | Correctly adds the node as child of the master node |  |
| 7 | Scene\_Node::DetachNode() | Remove the inputted node from the children vector |  |  |
| 8 | Scene\_Node::Execute() | Render the node and its children | Gets the location of the object and renders the model, if they have one. Also does the same for the child nodes |  |
| 9 | Scene\_Node::UpdateCollisionTree() | Updates the locations of the nodes to ensure they collide correctly | Correctly updates the world center location of the boundary box or sphere |  |
| 10 | Scene\_Node::CheckCollision() | Checks for a collision between objects that have models | A sphere wasn’t colliding with a cube, but a cube could collide with a sphere | Turns out there was an operator error. Had to switch the location of a + and - |
| 10 – Test 2 | Scene\_Node::CheckCollision() | Checks for a collision between objects that have models | Correctly returned if there was a collision between objects |  |
| 11 | Scene\_Node::~Scene\_Node() | Clean up references and empty the vector of children nodes | Kept breaking when deleting | Turns out I had 2 references to the same node, and when one was being deleted it created an error in the vector. Simple solution – delete the duplicate code |
| 11 – Test 2 | Scene\_Node::~Scene\_Node() | Clean up references and empty the vector of children nodes | Correctly cleans up |  |
| 12 | Player::Update() | Updates the players game logic | Correctly moves the player to the ground if jumped |  |
| 13 | Player::MoveForward() | Moves the player along their forward vector | Moves the player in their forward vector |  |
| 14 | Player::MoveRight() | Moves the player along their right vector | Moves the player in the right vector |  |
| 15 | Player::CalculateForwardVector() | Caclulcate the forward vector based on the camera | Correctly calculates the forward direction |  |
| 16 | Player::CalculateRightVector() | Calculates the right vector based on the cameras position | Correctly calculates the forward direction |  |
| 17 | Enemy::Update() |  |  |  |
| 18 | Enemy::AddWaypoint() |  |  |  |
| 19 | Enemy::Move() |  |  |  |
| 20 | Enemy::NextWaypoint() | Change the currentWaypoint value by 1. If over the size of the waypoints, put it to 0 | The currentWaypoint was going over the size of the m\_waypoints. | Changed the comparison from > to >=. |
| 21 | Enemy::AtWaypoint() | Check if the position of the enemy is equal to the current waypoint | Was always return false. | Changed the way the comparison was done. Now checking if the m\_position is equal to m\_wayPoints[m\_currentWaypoint] |
| 21 –  Test 2 | Enemy::AtWaypoint() | Check if the position of the enemy is equal to the current waypoint | Was still returning false. | Changed it to check the distance between the vectors with a subtraction. Then if it was within 0.05f of the location, return true |
| 21 -Test 3 | Enemy::AtWaypoint() | Check if the position of the enemy is equal to the current waypoint | Returned true if the way points position |  |
| 22 | Reflect\_shader.hlsl ModelPS() | Render out the pixels of the skybox onto the sphere | Wasn’t rendering out what soever | Change the Texture2D to TextureCube to allow for the cubemap to be passed through |
| 22 – Test 2 | Reflect\_Shader.hlsl ModelPS() | Render out the pixels of the skybox onto the sphere | Correctly rendered. |  |

# Conclusion

In this project I was successfully able to incorporate some of the features specified in my Software Design Document. Along the way I learned a lot about the DirectX 11 API and about C++ programming in general. One key thing I learned about were buffers. Buffers are a way for the C++ code to interact with shaders. Unfortunately, I was unable to try and get a geometry buffer to work but had read about it from several resources. I was however able to get a constant buffer working with my shaders. This helped me set up things like lighting, reflections and clipping pixels. I used this method to create my own dissolve texture, which clipped pixels based on a noise texture sample.

Another thing that I’ve improved on during this project is my ability to do text input for level creation. I had previously tried this in game engines like Unity but had never really done so in C++. The good thing is that it is relatively easy to do, and plan to use it in different projects that I do.

I also had a go at setting up a timer class to properly get the frames per second and delta time of the application. These could then be used with my other classes such as the player and enemy, to ensure their movement wasn’t sporadic. I was then able to display the frames per second onto my HUD using the provided text class. Within this text class I also added in alpha transparency. This allowed the text to not have a black background around it when rendering to the screen.

# References

## Software Design Document – Appendix A

### Game Design – Ethan and Paulo

This document will help clarify the mechanics that will be implemented in the second assignment for this module. Below are two lists with the project requirements and the additional functionality we will use. After that will be a brief of the game

#### Project requirements

* 3D Game
* Player should be able to move around
* Environment should have static and moving obstacles
* Some obstacles should be moveable
* Simple Textures
* Simple Lighting
* Collisions
* NPCs represented by models
* NPCs collide with objects and perform action (change direction)
* Interaction between player and NPC

#### Additional Functionality

* Player can jump
* Enemies have simple pathfinding
* Mini-map UI
* Advanced shader for dissolving the player when they die
* 3rd person camera

### Game Brief

We have chosen to create a 3rd person puzzle game in DirectX 11 with C++. The game will consist of a minimum of 5 levels with each level having increasing difficulty. The game will have two modes, timed and unlimited time. In the timed mode users will have a certain amount of time to complete each level, whereas the unlimited time mode will have no time limit. Both modes will have a leader board showing the fastest players in each.

The user will be able to move around the level using basic movement as well as jumping, this will be done using the ‘WASD’ and ‘Space Bar’ keys. The user will be able to rotate the camera around the player to see around the level, this will be done with mouse input.

When the user completes a level, there will be a loading screen to transition them into the next level. There will also be some particle effects to pronounce the fact they have completed the level.

### Game Inspirations

We had some influences when thinking of the design for the game. The first influence we had was Portal. Portal is a great example of a puzzle game as it gives players a sense of accomplishment when they create a room. We will not be using the same mechanics as portal but it’s a great inspiration for our level design.



*Figure 1 Portal 2, image source -* [*https://www.rockpapershotgun.com/2007/10/10/rps-verdict-portal/*](https://www.rockpapershotgun.com/2007/10/10/rps-verdict-portal/)

Another influence that we talked about while designing was the game Q.U.B.E 2. Q.U.B.E 2 has good art and level design which could be incorporated into our design. They also have a great way of using lighting to direct the player in the correct direction for the puzzle.



*Figure 2 QUBE 2, Image source -* [*https://www.gamasutra.com/view/news/316731/How\_the\_QUBE\_2\_devs\_built\_a\_better\_massive\_3D\_puzzle\_labyrinth.php*](https://www.gamasutra.com/view/news/316731/How_the_QUBE_2_devs_built_a_better_massive_3D_puzzle_labyrinth.php)

## Bibliography - Appendix B

FEDERICO, 2018. *Dissolve Shader Tutorial - HLSL and Unity's Shader Graph - Febucci*[viewed 13/01/ 2019]. Available from: <https://www.febucci.com/2018/09/dissolve-shader/>

FEINSTEIN, D., 2013. *HLSL Development Cookbook.*1st ed. Birmingham, England: Packt Publishing