CSC8507 – Technology Implementation Appendices

Link to GitHub of our Unity Prototype

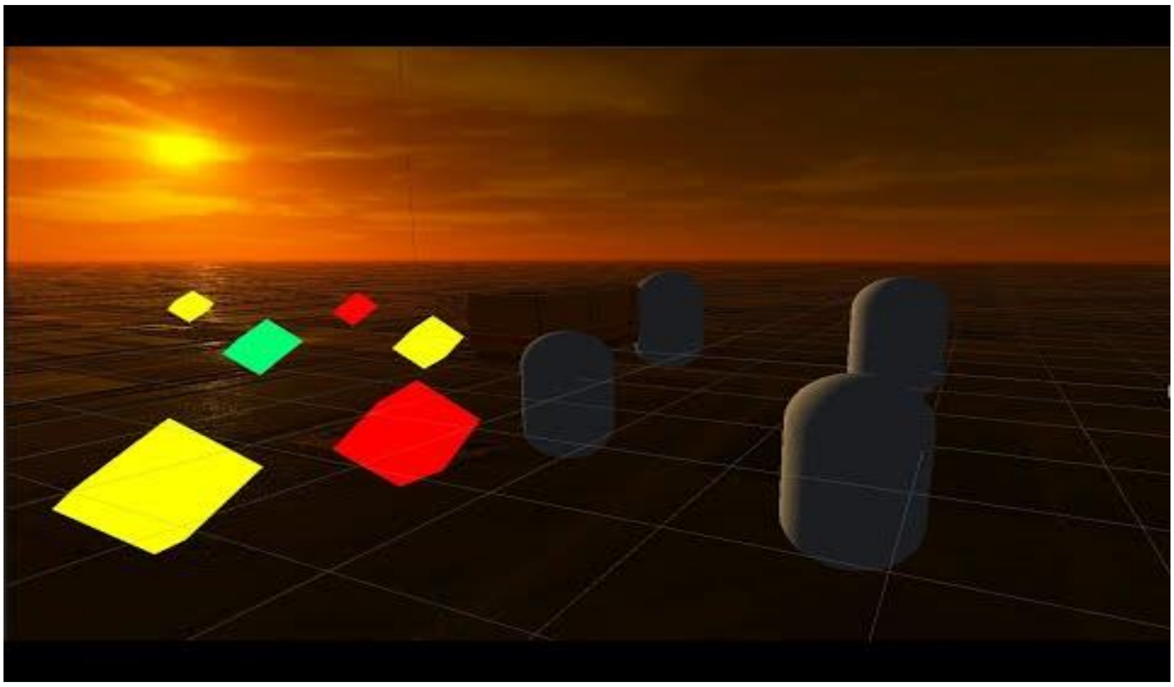
https://github.com/Group-Project-Team-6/Group-Project/tree/UnityGraphics/Group%20Demo

Unity Project of our Prototype

Group Demo.zip

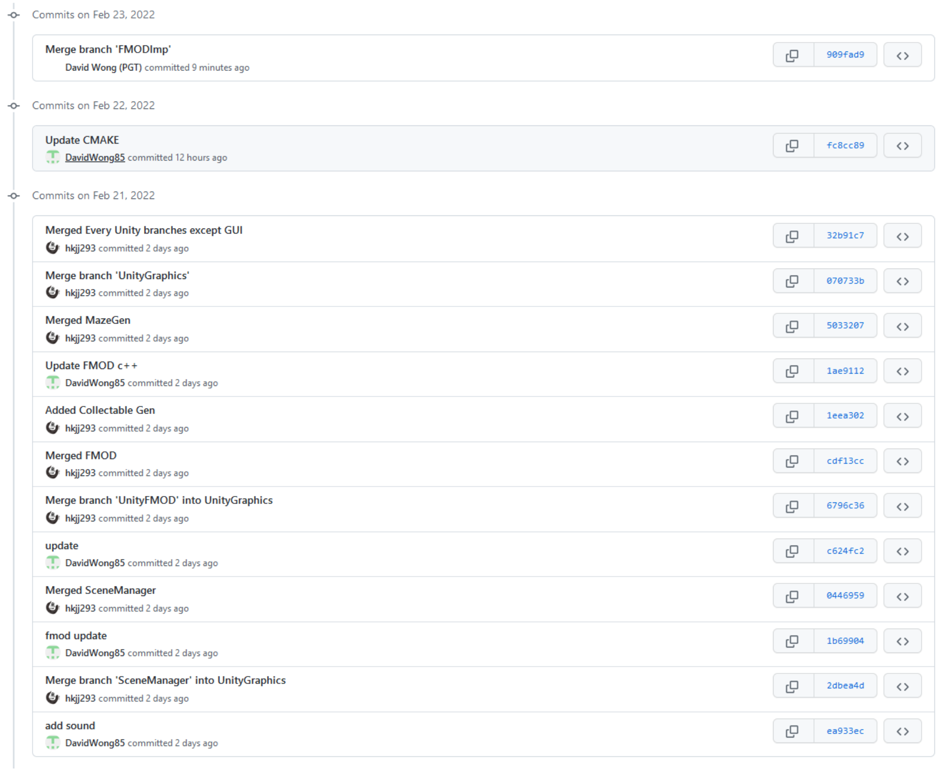
YouTube Link for our Game Play Demo

CSC8507 Splatooth 2042 Prototype



Appendix A – GitHub Commit Logs

Copy of Team 6’s GitHub commit logs from 23/02/2022


Commits on Feb 23, 2022

Merge branch 'FMODImp'
David Wong (PGT) committed 9 minutes ago
909fad9

Commits on Feb 22, 2022

Update CMAKE
DavidWong85 committed 12 hours ago
fc8cc89

Commits on Feb 21, 2022

Merged Every Unity branches except GUI
hkjj293 committed 2 days ago
32b91c7

Merge branch 'UnityGraphics'
hkjj293 committed 2 days ago
070733b

Merged MazeGen
hkjj293 committed 2 days ago
5033207

Update FMOD c++
DavidWong85 committed 2 days ago
1ae9112

Added Collectable Gen
hkjj293 committed 2 days ago
1eea302

Merged FMOD
hkjj293 committed 2 days ago
cdf13cc

Merge branch 'UnityFMOD' into UnityGraphics
hkjj293 committed 2 days ago
6796c36

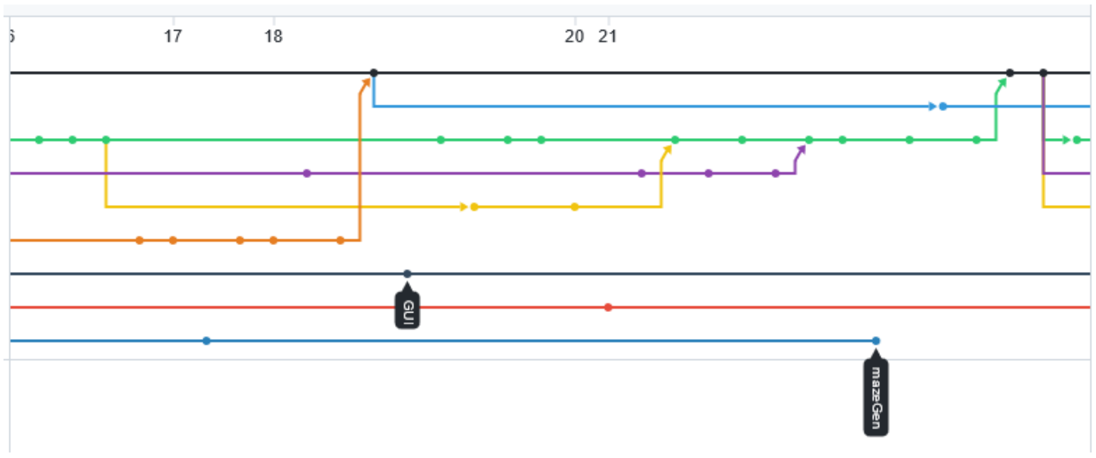
update
DavidWong85 committed 2 days ago
c624fc2

Merged SceneManager
hkjj293 committed 2 days ago
0446959

fmod update
DavidWong85 committed 2 days ago
1b69904

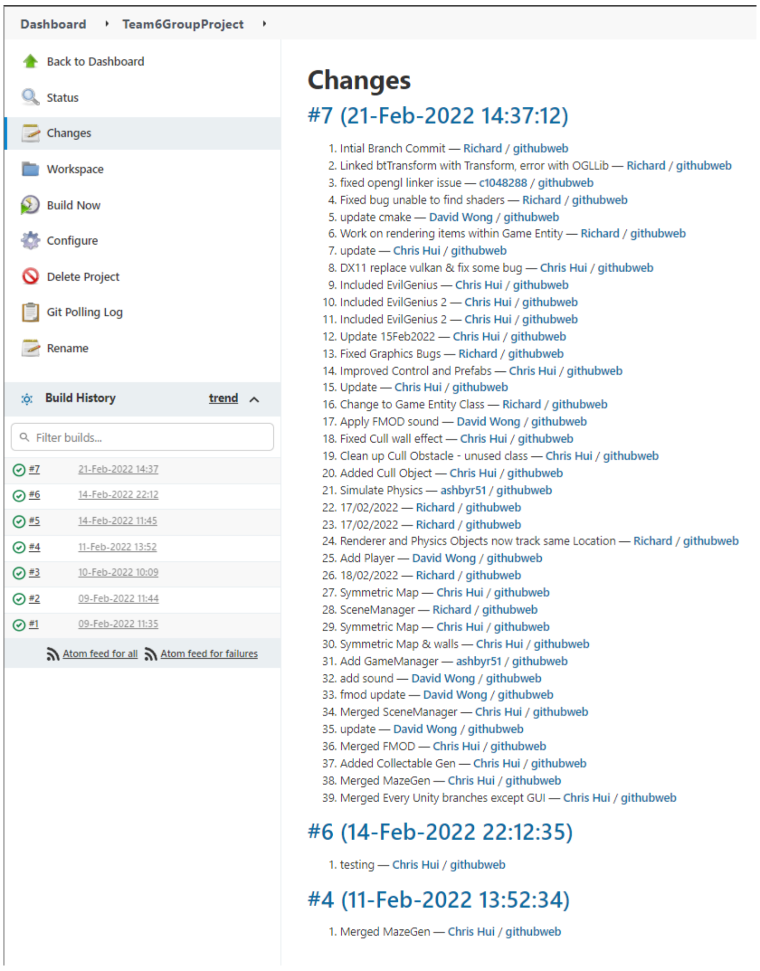
Merge branch 'SceneManager' into UnityGraphics
hkjj293 committed 2 days ago
2dbea4d

add sound
DavidWong85 committed 2 days ago
ea933ec



Appendix B – Jenkins Build Logs

Copy of Team 6's Jenkins commit logs from 23/02/2022



Dashboard ‣ Team6GroupProject ‣

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Configure
- Delete Project
- Git Polling Log
- Rename

Build History trend

Filter builds...

#7	21-Feb-2022 14:37
#6	14-Feb-2022 22:12
#5	14-Feb-2022 11:45
#4	11-Feb-2022 13:52
#3	10-Feb-2022 10:09
#2	09-Feb-2022 11:44
#1	09-Feb-2022 11:35

Atom feed for all Atom feed for failures

Changes

#7 (21-Feb-2022 14:37:12)

1. Intial Branch Commit — Richard / githubweb
2. Linked btTransform with Transform, error with OGLLib — Richard / githubweb
3. fixed opengl linker issue — c1048288 / githubweb
4. Fixed bug unable to find shaders — Richard / githubweb
5. update cmake — David Wong / githubweb
6. Work on rendering items within Game Entity — Richard / githubweb
7. update — Chris Hui / githubweb
8. DX11 replace vulkan & fix some bug — Chris Hui / githubweb
9. Included EvilGenius — Chris Hui / githubweb
10. Included EvilGenius 2 — Chris Hui / githubweb
11. Included EvilGenius 2 — Chris Hui / githubweb
12. Update 15Feb2022 — Chris Hui / githubweb
13. Fixed Graphics Bugs — Richard / githubweb
14. Improved Control and Prefabs — Chris Hui / githubweb
15. Update — Chris Hui / githubweb
16. Change to Game Entity Class — Richard / githubweb
17. Apply FMOD sound — David Wong / githubweb
18. Fixed Cull wall effect — Chris Hui / githubweb
19. Clean up Cull Obstacle - unused class — Chris Hui / githubweb
20. Added Cull Object — Chris Hui / githubweb
21. Simulate Physics — ashbyr51 / githubweb
22. 17/02/2022 — Richard / githubweb
23. 17/02/2022 — Richard / githubweb
24. Renderer and Physics Objects now track same Location — Richard / githubweb
25. Add Player — David Wong / githubweb
26. 18/02/2022 — Richard / githubweb
27. Symmetric Map — Chris Hui / githubweb
28. SceneManager — Richard / githubweb
29. Symmetric Map — Chris Hui / githubweb
30. Symmetric Map & walls — Chris Hui / githubweb
31. Add GameManager — ashbyr51 / githubweb
32. add sound — David Wong / githubweb
33. fmod update — David Wong / githubweb
34. Merged SceneManager — Chris Hui / githubweb
35. update — David Wong / githubweb
36. Merged FMOD — Chris Hui / githubweb
37. Added Collectable Gen — Chris Hui / githubweb
38. Merged MazeGen — Chris Hui / githubweb
39. Merged Every Unity branches except GUI — Chris Hui / githubweb

#6 (14-Feb-2022 22:12:35)

1. testing — Chris Hui / githubweb

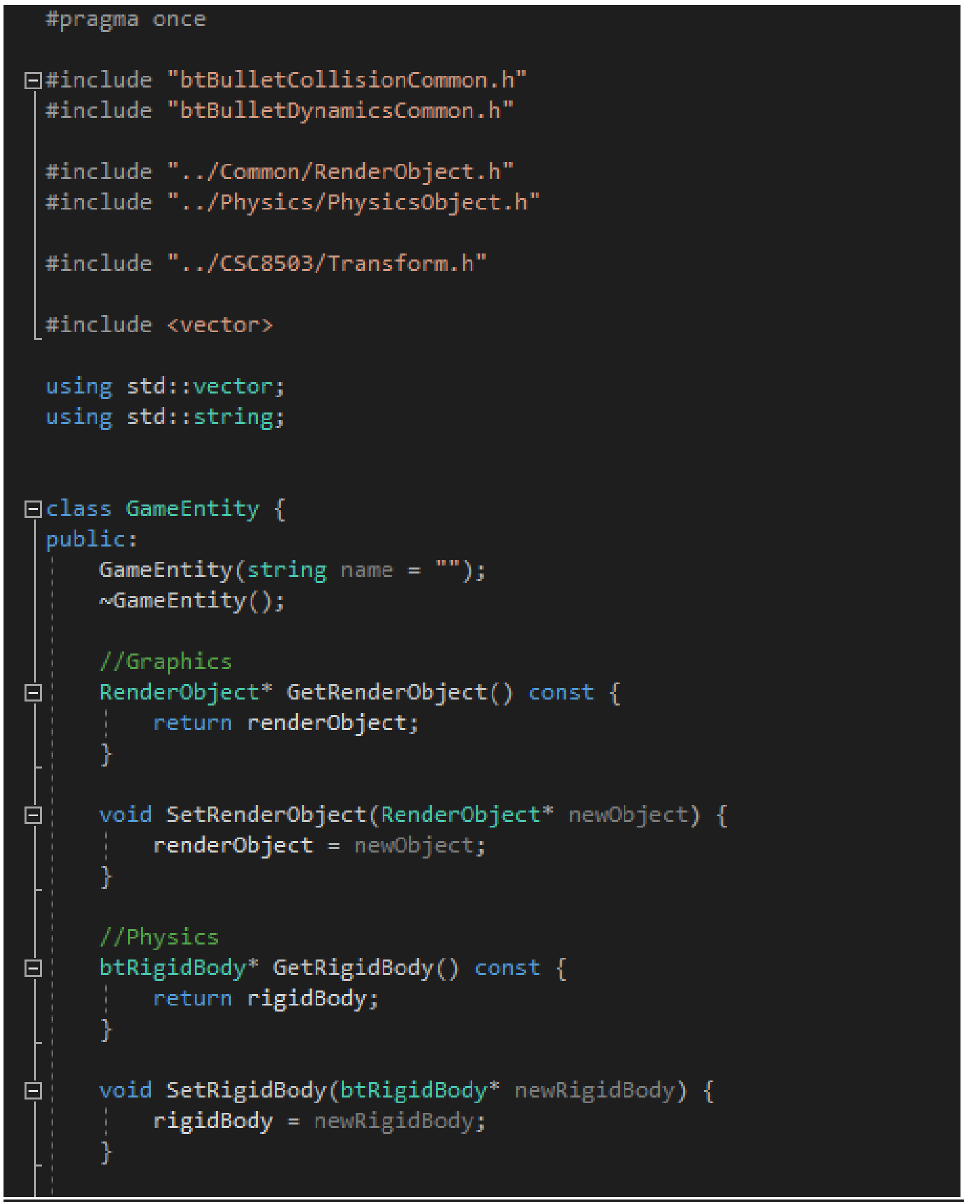
#4 (11-Feb-2022 13:52:34)

1. Merged MazeGen — Chris Hui / githubweb

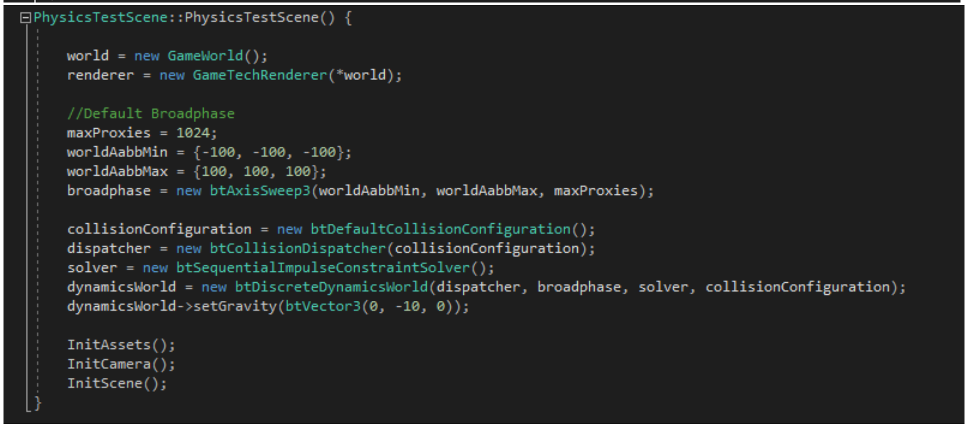
Appendix C – Code Snippets of Integrated middleware into Custom Engine

Physics – Bullet Physics

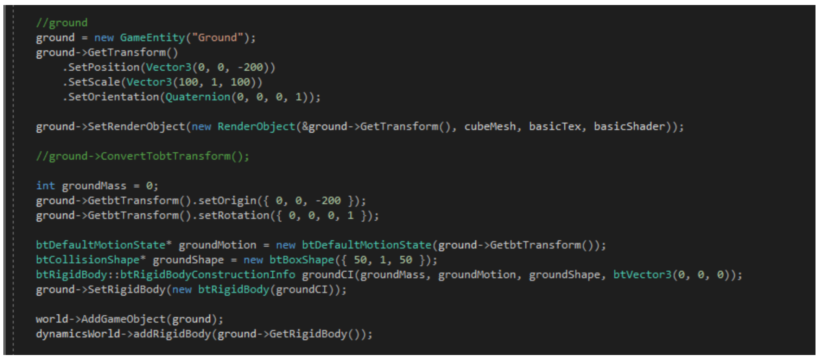
-Bullet Physics middleware accessed by two provided header files.
-Physics world created with a dynamic's world created in game's constructor
-Physics objects transformed accessed and shared in new game Entity class
-Simulation increments called in existing update loop.



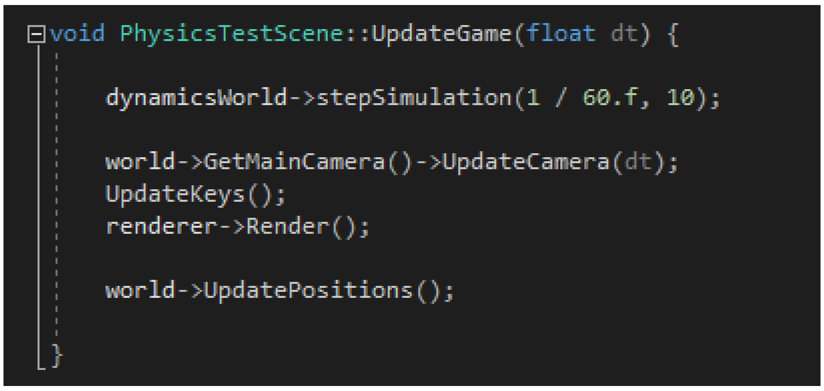
```cpp
#pragma once

#include "btBulletCollisionCommon.h"
#include "btBulletDynamicsCommon.h"

#include "../Common/RenderObject.h"
#include "../Physics/PhysicsObject.h"

#include "../CSC8503/Transform.h"

#include <vector>

using std::vector;
using std::string;


class GameEntity {
public:
    GameEntity(string name = "");
    ~GameEntity();

    //Graphics
    RenderObject* GetRenderObject() const {
        return renderObject;
    }

    void SetRenderObject(RenderObject* newObject) {
        renderObject = newObject;
    }

    //Physics
    btRigidBody* GetRigidBody() const {
        return rigidBody;
    }

    void SetRigidBody(btRigidBody* newRigidBody) {
        rigidBody = newRigidBody;
    }
```



```cpp
PhysicsTestScene::PhysicsTestScene() {

    world = new GameWorld();
    renderer = new GameTechRenderer(*world);

    //Default Broadphase
    maxProxies = 1024;
    worldAabbMin = {-100, -100, -100};
    worldAabbMax = {100, 100, 100};
    broadphase = new btAxisSweep3(worldAabbMin, worldAabbMax, maxProxies);

    collisionConfiguration = new btDefaultCollisionConfiguration();
    dispatcher = new btCollisionDispatcher(collisionConfiguration);
    solver = new btSequentialImpulseConstraintSolver();
    dynamicsWorld = new btDiscreteDynamicsWorld(dispatcher, broadphase, solver, collisionConfiguration);
    dynamicsWorld->setGravity(btVector3(0, -10, 0));

    InitAssets();
    InitCamera();
    InitScene();
}
```



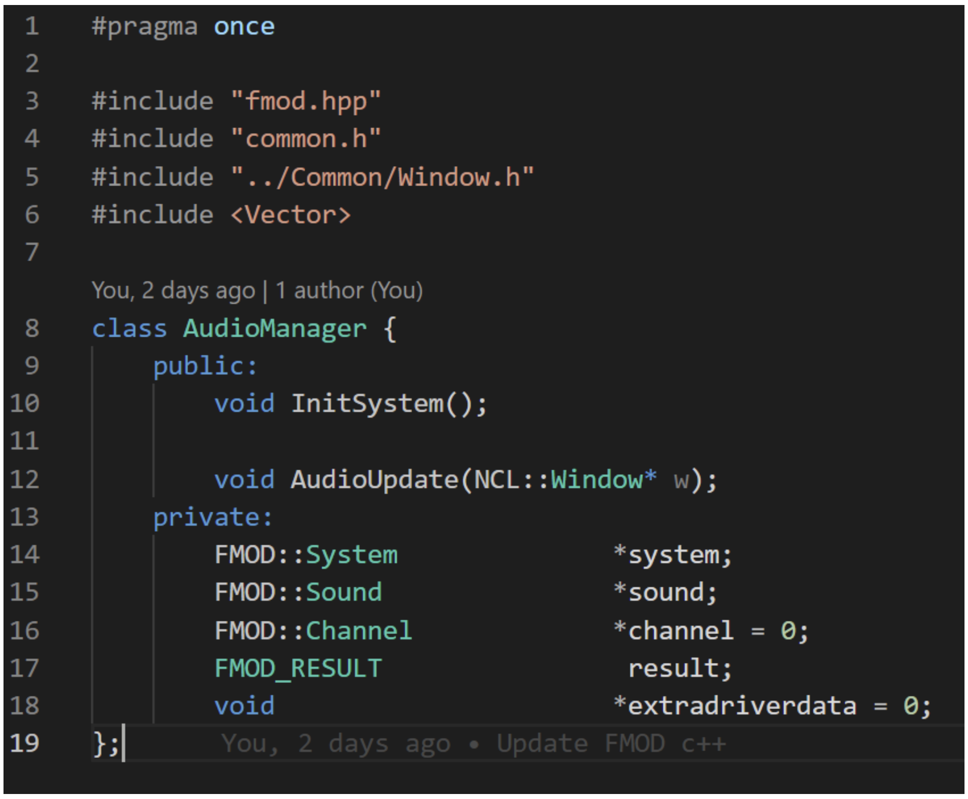
```cpp
//ground
ground = new GameEntity("Ground");
ground->GetTransform()
    .SetPosition(Vector3(0, 0, -200))
    .SetScale(Vector3(100, 1, 100))
    .SetOrientation(Quaternion(0, 0, 0, 1));

ground->SetRenderObject(new RenderObject(&ground->GetTransform(), cubeMesh, basicTex, basicShader));

//ground->ConvertTobtTransform();

int groundMass = 0;
ground->GetbtTransform().setOrigin({ 0, 0, -200 });
ground->GetbtTransform().setRotation({ 0, 0, 0, 1 });

btDefaultMotionState* groundMotion = new btDefaultMotionState(ground->GetbtTransform());
btCollisionShape* groundShape = new btBoxShape({ 50, 1, 50 });
btRigidBody::btRigidBodyConstructionInfo groundCI(groundMass, groundMotion, groundShape, btVector3(0, 0, 0));
ground->SetRigidBody(new btRigidBody(groundCI));

world->AddGameObject(ground);
dynamicsWorld->addRigidBody(ground->GetRigidBody());
```



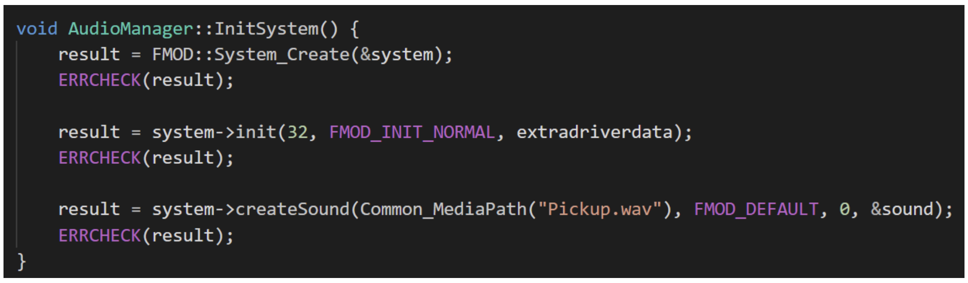
```cpp
void PhysicsTestScene::UpdateGame(float dt) {

    dynamicsWorld->stepSimulation(1 / 60.f, 10);

    world->GetMainCamera()->UpdateCamera(dt);
    UpdateKeys();
    renderer->Render();

    world->UpdatePositions();

}
```

Audio – FMOD

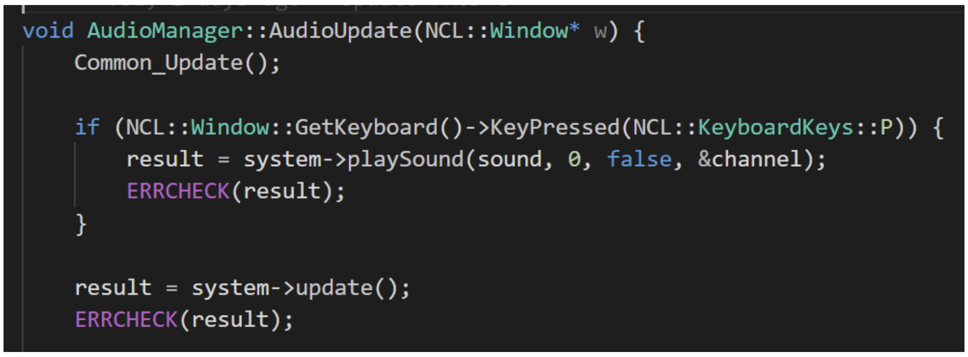
-“common.h” is an FMOD provided header
-ERRCHECK are built in debug log function
-Init system is called in start of main
-The AudioUpdate() function is called within main loop



```
#pragma once

#include "fmod.hpp"
#include "common.h"
#include "../Common/Window.h"
#include <Vector>

You, 2 days ago | 1 author (You)
class AudioManager {
    public:
        void InitSystem();

        void AudioUpdate(NCL::Window* w);
    private:
        FMOD::System            *system;
        FMOD::Sound             *sound;
        FMOD::Channel           *channel = 0;
        FMOD_RESULT              result;
        void                    *extradriverdata = 0;
};        You, 2 days ago • Update FMOD c++
```



```
void AudioManager::InitSystem() {
    result = FMOD::System_Create(&system);
    ERRCHECK(result);

    result = system->init(32, FMOD_INIT_NORMAL, extradriverdata);
    ERRCHECK(result);

    result = system->createSound(Common_MediaPath("Pickup.wav"), FMOD_DEFAULT, 0, &sound);
    ERRCHECK(result);
}
```

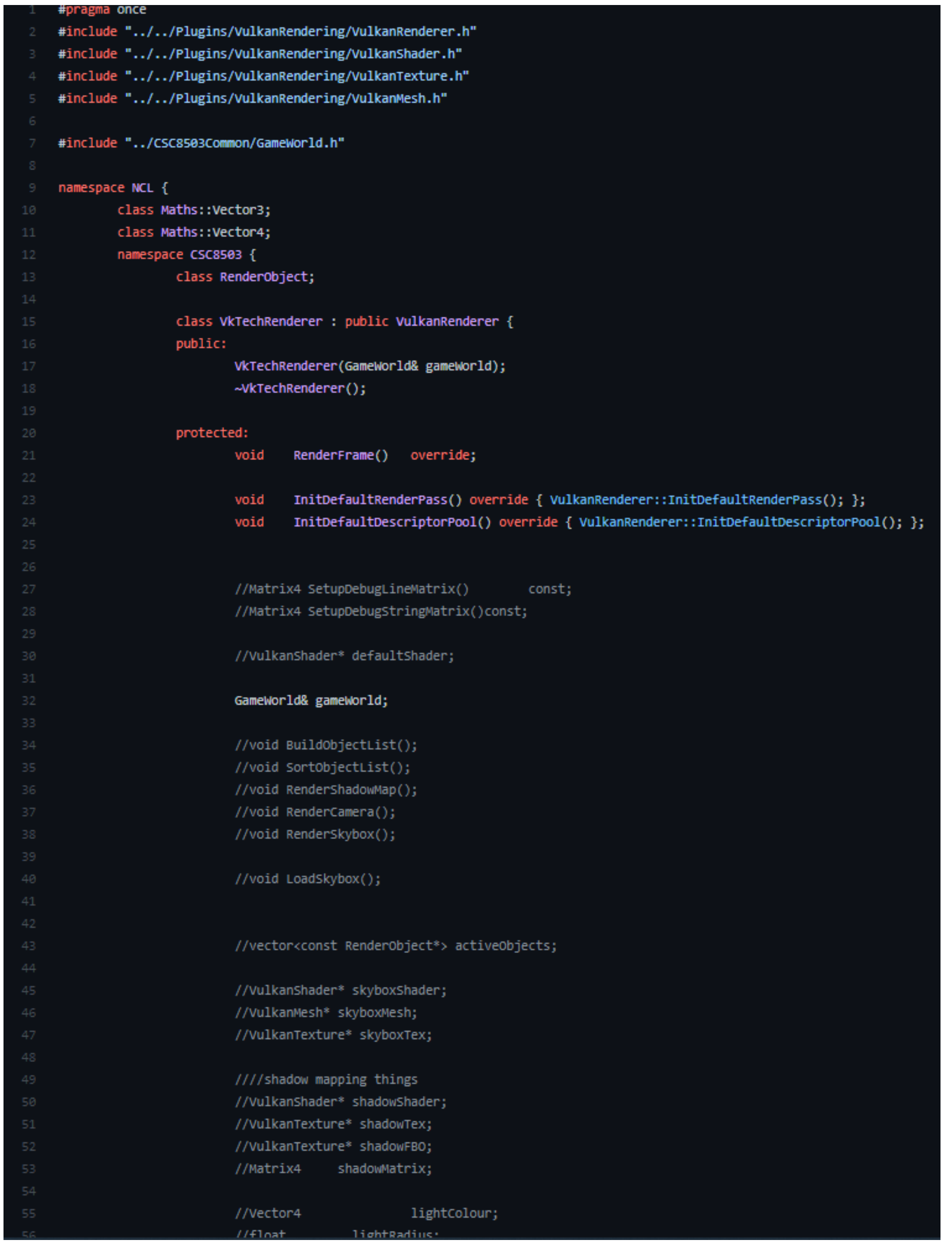


```
void AudioManager::AudioUpdate(NCL::Window* w) {
    Common_Update();

    if (NCL::Window::GetKeyboard()->KeyPressed(NCL::KeyboardKeys::P)) {
        result = system->playSound(sound, 0, false, &channel);
        ERRCHECK(result);
    }

    result = system->update();
    ERRCHECK(result);
```

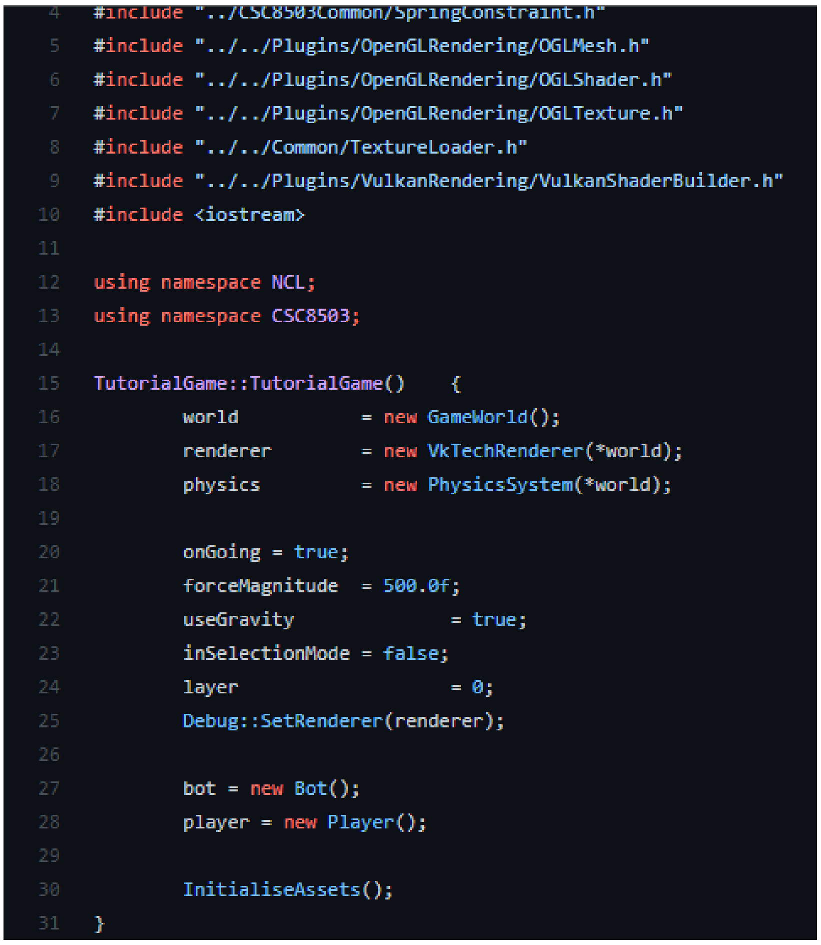
Rendering API – Vulkan

-Finish Technology Implementation Doc

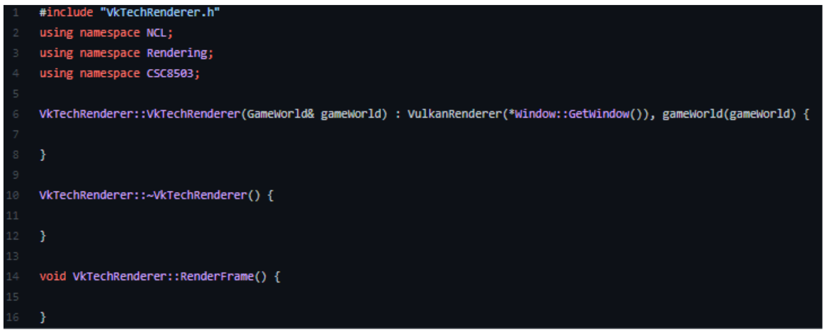
-New Class Vulkan Renderer with Draw Calls
-Initialised in Game Constructor
-Renderer called every frame in Update Game



```
#pragma once
#include "../../Plugins/VulkanRendering/VulkanRenderer.h"
#include "../../Plugins/VulkanRendering/VulkanShader.h"
#include "../../Plugins/VulkanRendering/VulkanTexture.h"
#include "../../Plugins/VulkanRendering/VulkanMesh.h"

#include "../CSC8503Common/GameWorld.h"

namespace NCL {
        class Maths::Vector3;
        class Maths::Vector4;
        namespace CSC8503 {
                class RenderObject;

                class VkTechRenderer : public VulkanRenderer {
                public:
                        VkTechRenderer(GameWorld& gameWorld);
                        ~VkTechRenderer();

                protected:
                        void    RenderFrame()   override;

                        void    InitDefaultRenderPass() override { VulkanRenderer::InitDefaultRenderPass(); };
                        void    InitDefaultDescriptorPool() override { VulkanRenderer::InitDefaultDescriptorPool(); };


                        //Matrix4 SetupDebugLineMatrix()        const;
                        //Matrix4 SetupDebugStringMatrix()const;

                        //VulkanShader* defaultShader;

                        GameWorld& gameWorld;

                        //void BuildObjectList();
                        //void SortObjectList();
                        //void RenderShadowMap();
                        //void RenderCamera();
                        //void RenderSkybox();

                        //void LoadSkybox();


                        //vector<const RenderObject*> activeObjects;

                        //VulkanShader* skyboxShader;
                        //VulkanMesh* skyboxMesh;
                        //VulkanTexture* skyboxTex;

                        ////shadow mapping things
                        //VulkanShader* shadowShader;
                        //VulkanTexture* shadowTex;
                        //VulkanTexture* shadowFBO;
                        //Matrix4     shadowMatrix;

                        //Vector4               lightColour;
                        //float         lightRadius;
```



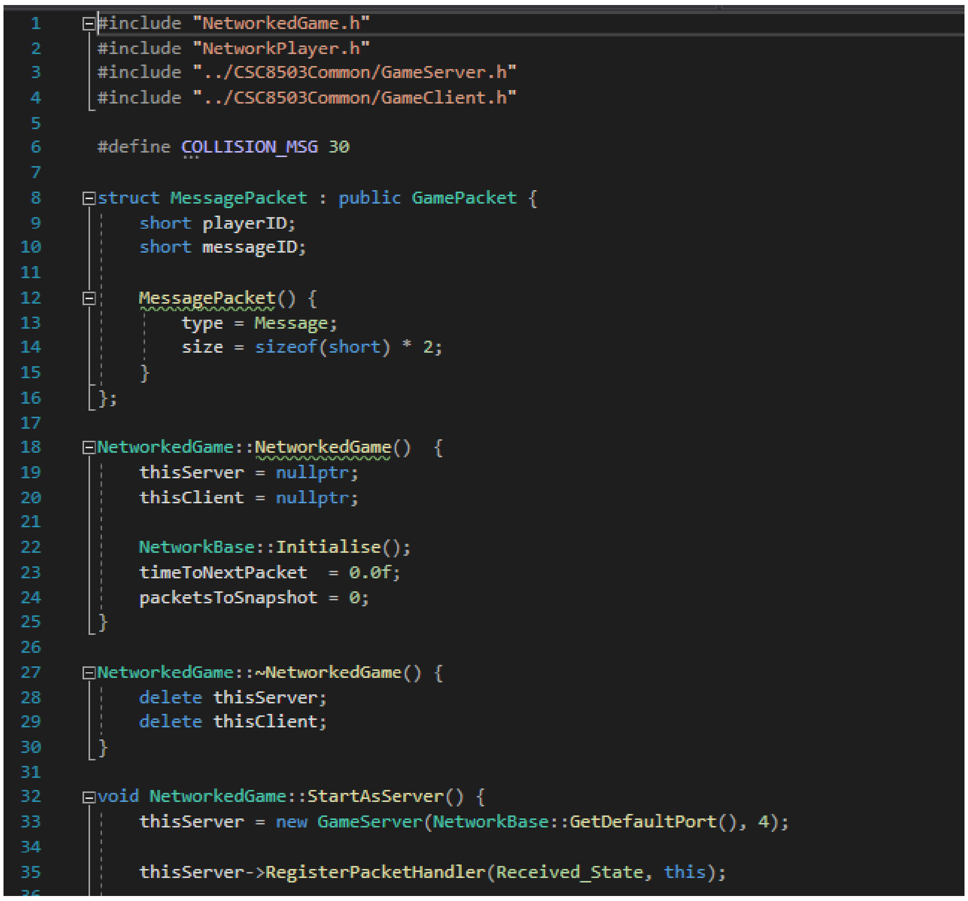
```
#include "../CSC8503Common/SpringConstraint.h"
#include "../../Plugins/OpenGLRendering/OGLMesh.h"
#include "../../Plugins/OpenGLRendering/OGLShader.h"
#include "../../Plugins/OpenGLRendering/OGLTexture.h"
#include "../../Common/TextureLoader.h"
#include "../../Plugins/VulkanRendering/VulkanShaderBuilder.h"
#include <iostream>

using namespace NCL;
using namespace CSC8503;

TutorialGame::TutorialGame()    {
        world           = new GameWorld();
        renderer        = new VkTechRenderer(*world);
        physics         = new PhysicsSystem(*world);

        onGoing = true;
        forceMagnitude  = 500.0f;
        useGravity              = true;
        inSelectionMode = false;
        layer                   = 0;
        Debug::SetRenderer(renderer);

        bot = new Bot();
        player = new Player();

        InitialiseAssets();
}
```



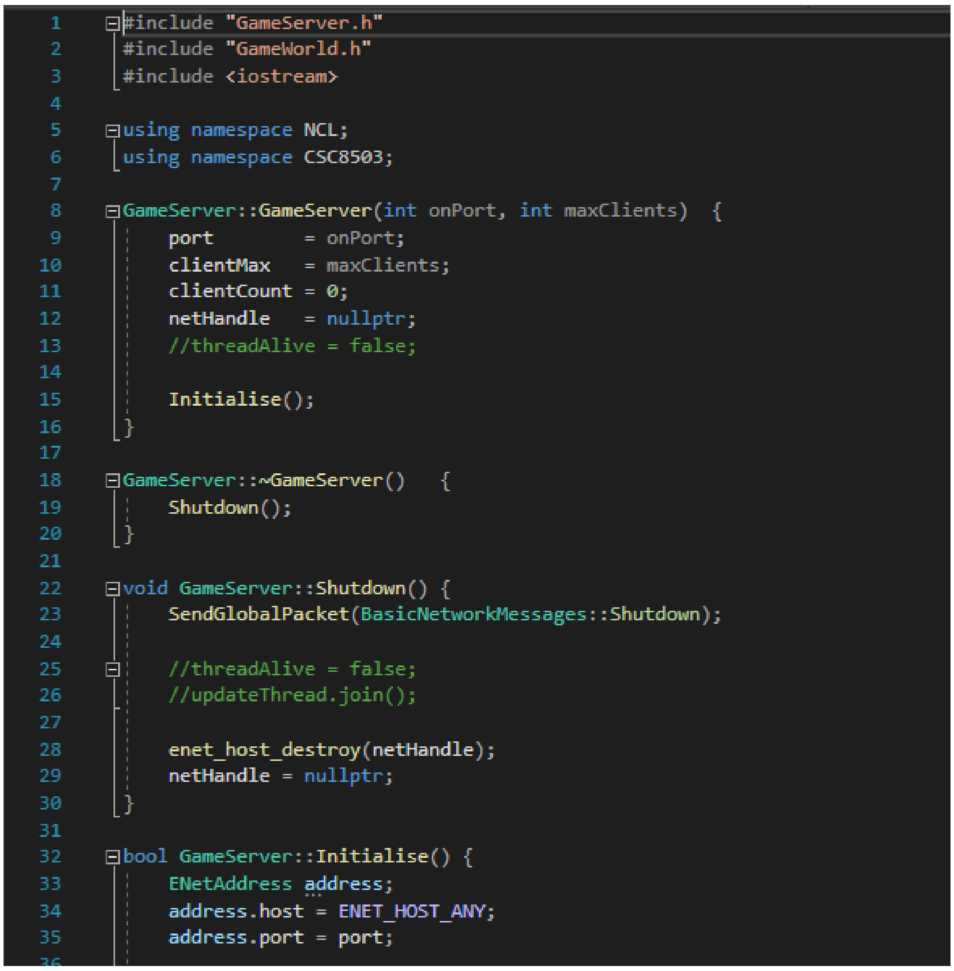
```
#include "VkTechRenderer.h"
using namespace NCL;
using namespace Rendering;
using namespace CSC8503;

VkTechRenderer::VkTechRenderer(GameWorld& gameWorld) : VulkanRenderer(*Window::GetWindow()), gameWorld(gameWorld) {

}

VkTechRenderer::~VkTechRenderer() {

}

void VkTechRenderer::RenderFrame() {

}
```

Networking - ENet (Srichand)

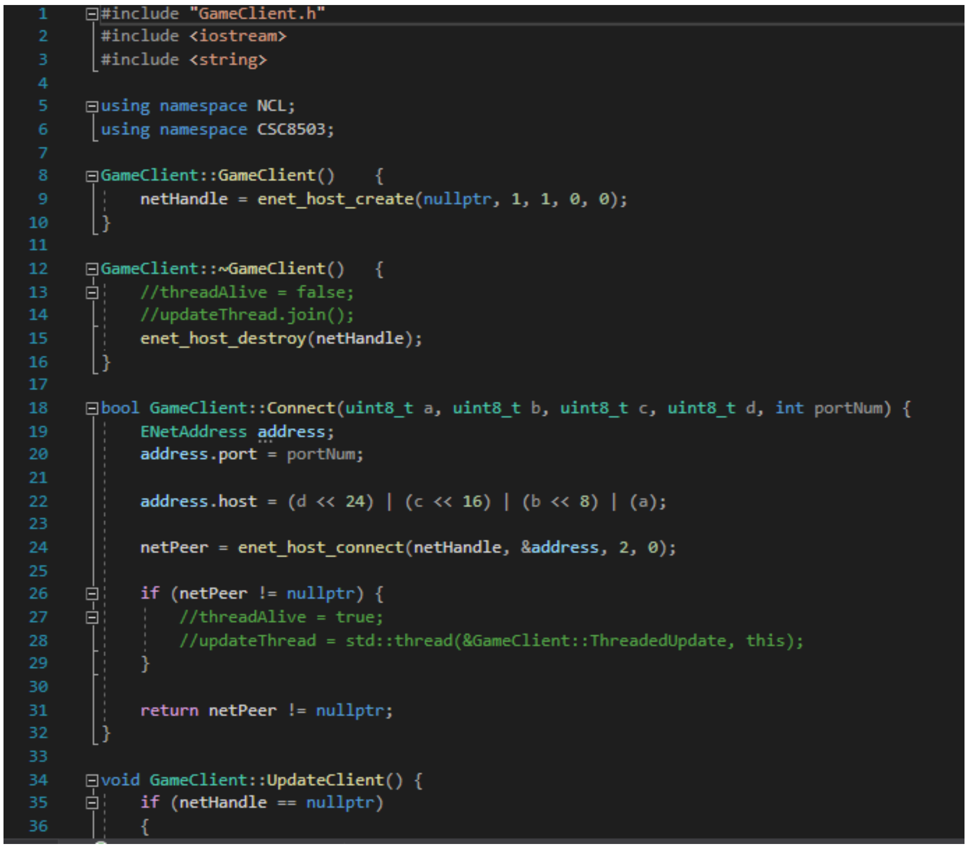
-ENet library is integrated into the Game Tech folder by copying the library files into the folder.
-The library functions are used in creating GameServer and GameClient files.



```cpp
#include "NetworkedGame.h"
#include "NetworkPlayer.h"
#include "../CSC8503Common/GameServer.h"
#include "../CSC8503Common/GameClient.h"

#define COLLISION_MSG 30

struct MessagePacket : public GamePacket {
    short playerID;
    short messageID;

    MessagePacket() {
        type = Message;
        size = sizeof(short) * 2;
    }
};

NetworkedGame::NetworkedGame()  {
    thisServer = nullptr;
    thisClient = nullptr;

    NetworkBase::Initialise();
    timeToNextPacket  = 0.0f;
    packetsToSnapshot = 0;
}

NetworkedGame::~NetworkedGame() {
    delete thisServer;
    delete thisClient;
}

void NetworkedGame::StartAsServer() {
    thisServer = new GameServer(NetworkBase::GetDefaultPort(), 4);

    thisServer->RegisterPacketHandler(Received_State, this);
```



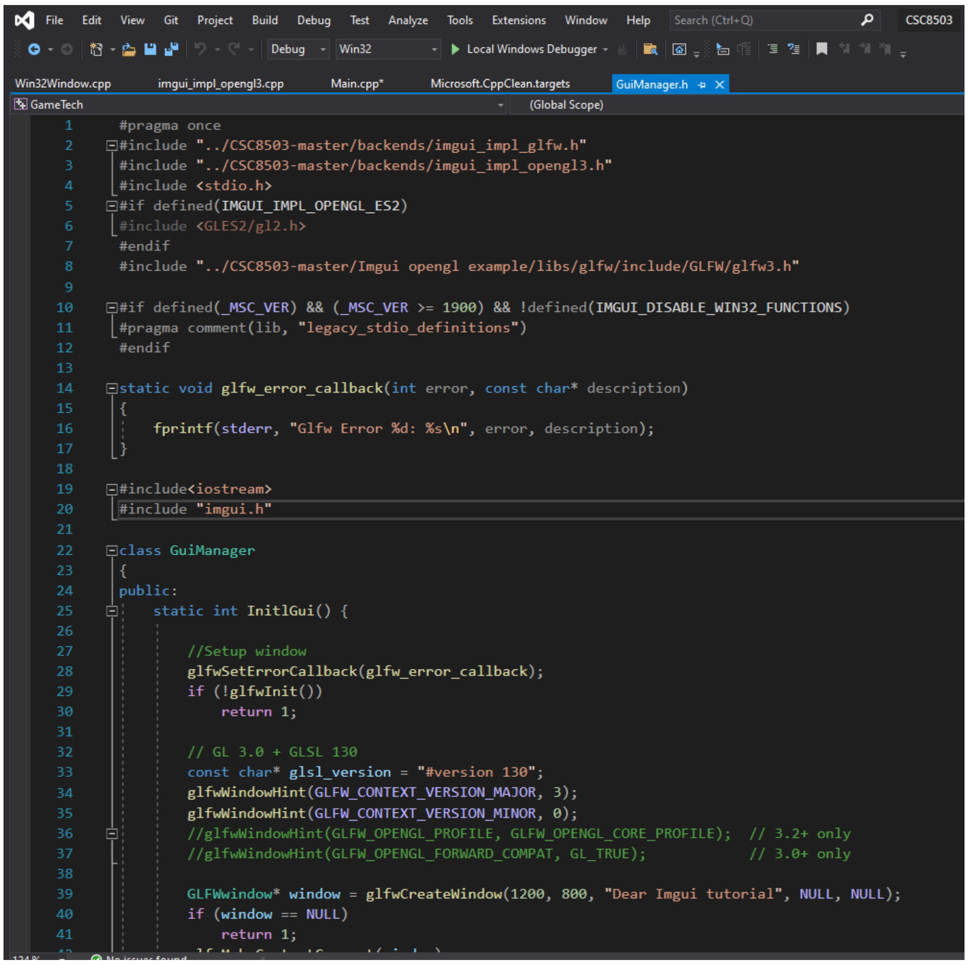
```cpp
#include "GameServer.h"
#include "GameWorld.h"
#include <iostream>

using namespace NCL;
using namespace CSC8503;

GameServer::GameServer(int onPort, int maxClients)  {
    port        = onPort;
    clientMax   = maxClients;
    clientCount = 0;
    netHandle   = nullptr;
    //threadAlive = false;

    Initialise();
}

GameServer::~GameServer()   {
    Shutdown();
}

void GameServer::Shutdown() {
    SendGlobalPacket(BasicNetworkMessages::Shutdown);

    //threadAlive = false;
    //updateThread.join();

    enet_host_destroy(netHandle);
    netHandle = nullptr;
}

bool GameServer::Initialise() {
    ENetAddress address;
    address.host = ENET_HOST_ANY;
    address.port = port;
```



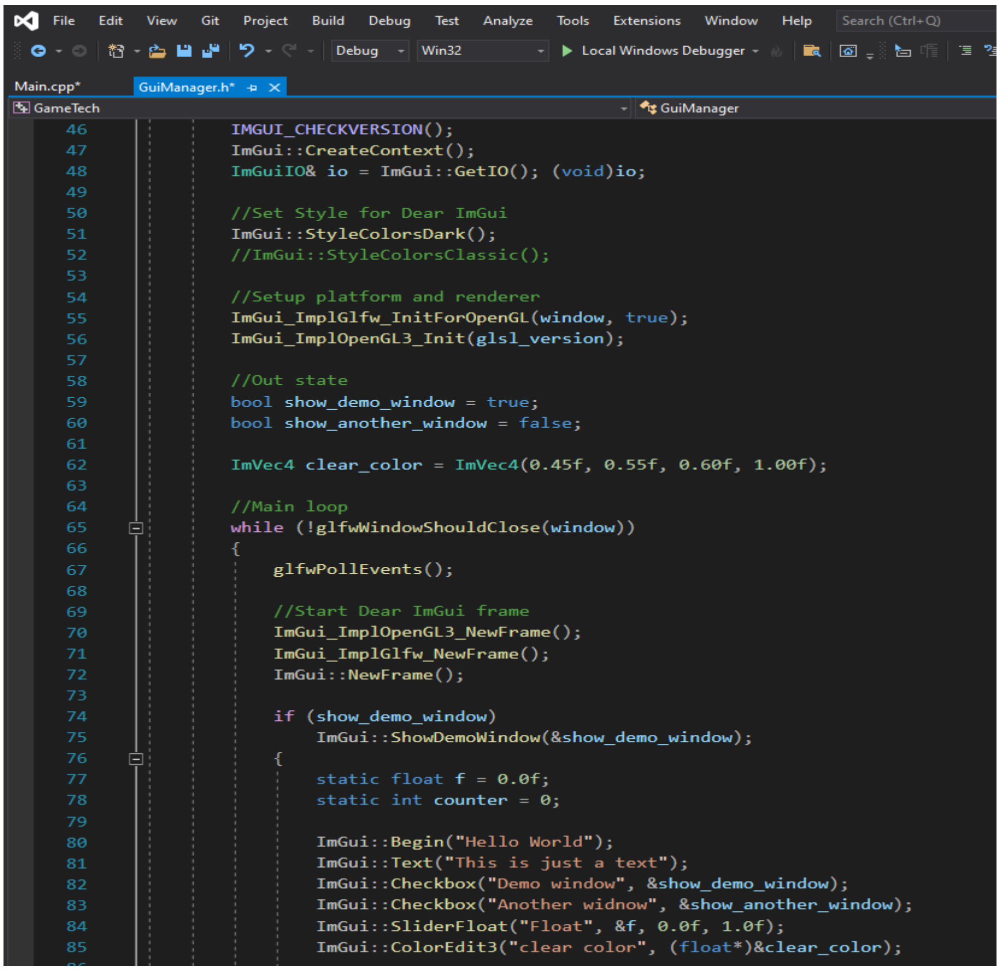
```cpp
#include "GameClient.h"
#include <iostream>
#include <string>

using namespace NCL;
using namespace CSC8503;

GameClient::GameClient()    {
    netHandle = enet_host_create(nullptr, 1, 1, 0, 0);
}

GameClient::~GameClient()   {
    //threadAlive = false;
    //updateThread.join();
    enet_host_destroy(netHandle);
}

bool GameClient::Connect(uint8_t a, uint8_t b, uint8_t c, uint8_t d, int portNum) {
    ENetAddress address;
    address.port = portNum;

    address.host = (d << 24) | (c << 16) | (b << 8) | (a);

    netPeer = enet_host_connect(netHandle, &address, 2, 0);

    if (netPeer != nullptr) {
        //threadAlive = true;
        //updateThread = std::thread(&GameClient::ThreadedUpdate, this);
    }

    return netPeer != nullptr;
}

void GameClient::UpdateClient() {
    if (netHandle == nullptr)
    {
```

GUI– ImGui

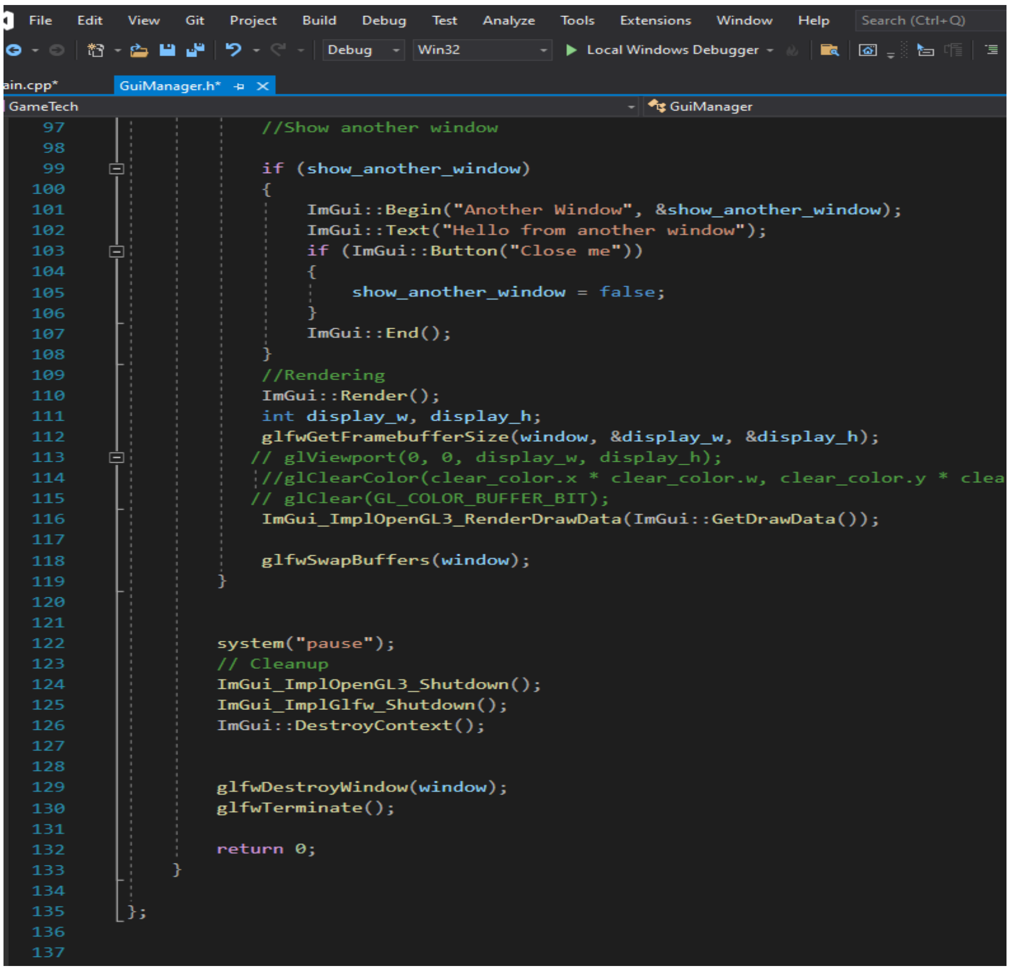
-GUI Manager class is created where imgui.h is defined to access the Imgui features
-Integrated a simple ImGui window using glfw and opengl3
-Imgui_impl_glfw and Imgui_impl_opengl3 invokes the backends used for this project
-Gui Manager here creates a function called InitlGui which is called in the main function



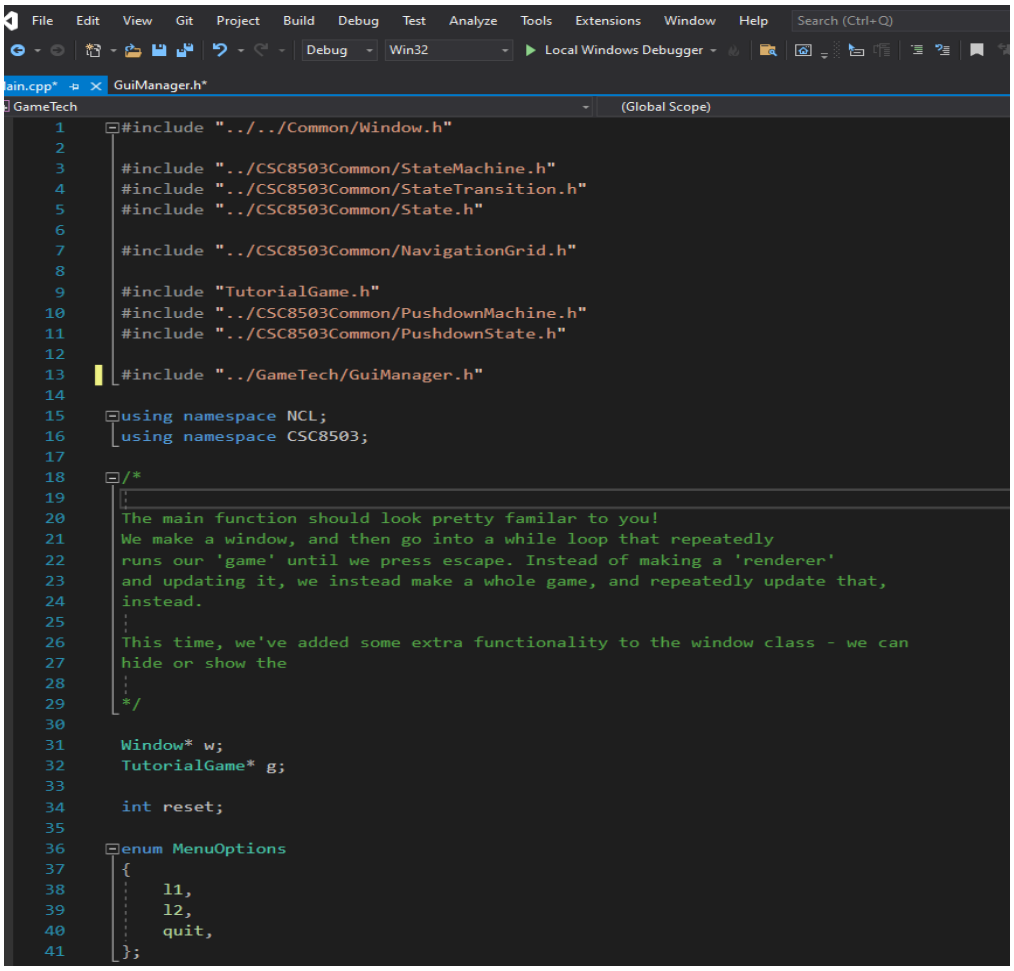
```cpp
#pragma once
#include "../CSC8503-master/backends/imgui_impl_glfw.h"
#include "../CSC8503-master/backends/imgui_impl_opengl3.h"
#include <stdio.h>
#if defined(IMGUI_IMPL_OPENGL_ES2)
#include <GLES2/gl2.h>
#endif
#include "../CSC8503-master/Imgui opengl example/libs/glfw/include/GLFW/glfw3.h"

#if defined(_MSC_VER) && (_MSC_VER >= 1900) && !defined(IMGUI_DISABLE_WIN32_FUNCTIONS)
#pragma comment(lib, "legacy_stdio_definitions")
#endif

static void glfw_error_callback(int error, const char* description)
{
    fprintf(stderr, "Glfw Error %d: %s\n", error, description);
}

#include<iostream>
#include "imgui.h"

class GuiManager
{
public:
    static int InitlGui() {

        //Setup window
        glfwSetErrorCallback(glfw_error_callback);
        if (!glfwInit())
            return 1;

        // GL 3.0 + GLSL 130
        const char* glsl_version = "#version 130";
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 0);
        //glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);  // 3.2+ only
        //glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);            // 3.0+ only

        GLFWwindow* window = glfwCreateWindow(1200, 800, "Dear Imgui tutorial", NULL, NULL);
        if (window == NULL)
            return 1;
```



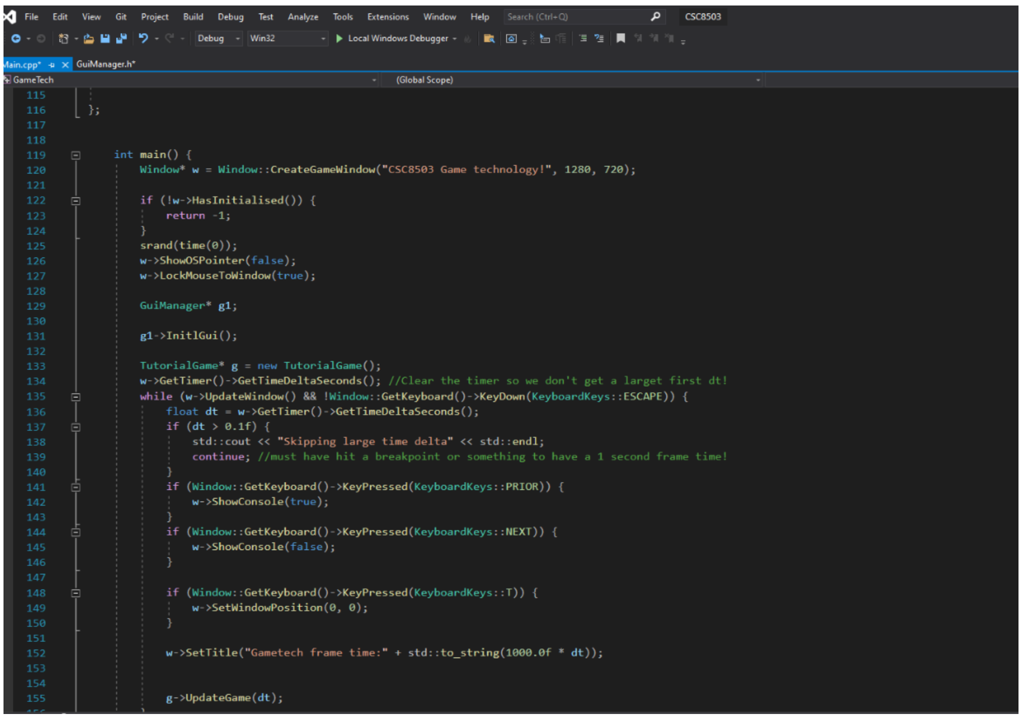
```cpp
        IMGUI_CHECKVERSION();
        ImGui::CreateContext();
        ImGuiIO& io = ImGui::GetIO(); (void)io;

        //Set Style for Dear ImGui
        ImGui::StyleColorsDark();
        //ImGui::StyleColorsClassic();

        //Setup platform and renderer
        ImGui_ImplGlfw_InitForOpenGL(window, true);
        ImGui_ImplOpenGL3_Init(glsl_version);

        //Out state
        bool show_demo_window = true;
        bool show_another_window = false;

        ImVec4 clear_color = ImVec4(0.45f, 0.55f, 0.60f, 1.00f);

        //Main loop
        while (!glfwWindowShouldClose(window))
        {
            glfwPollEvents();

            //Start Dear ImGui frame
            ImGui_ImplOpenGL3_NewFrame();
            ImGui_ImplGlfw_NewFrame();
            ImGui::NewFrame();

            if (show_demo_window)
                ImGui::ShowDemoWindow(&show_demo_window);
            {
                static float f = 0.0f;
                static int counter = 0;

                ImGui::Begin("Hello World");
                ImGui::Text("This is just a text");
                ImGui::Checkbox("Demo window", &show_demo_window);
                ImGui::Checkbox("Another widnow", &show_another_window);
                ImGui::SliderFloat("Float", &f, 0.0f, 1.0f);
                ImGui::ColorEdit3("clear color", (float*)&clear_color);
```



```
                //Show another window

                if (show_another_window)
                {
                    ImGui::Begin("Another Window", &show_another_window);
                    ImGui::Text("Hello from another window");
                    if (ImGui::Button("Close me"))
                    {
                        show_another_window = false;
                    }
                    ImGui::End();
                }
                //Rendering
                ImGui::Render();
                int display_w, display_h;
                glfwGetFramebufferSize(window, &display_w, &display_h);
               // glViewport(0, 0, display_w, display_h);
                //glClearColor(clear_color.x * clear_color.w, clear_color.y * clea
               // glClear(GL_COLOR_BUFFER_BIT);
                ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

                glfwSwapBuffers(window);
            }


            system("pause");
            // Cleanup
            ImGui_ImplOpenGL3_Shutdown();
            ImGui_ImplGlfw_Shutdown();
            ImGui::DestroyContext();


            glfwDestroyWindow(window);
            glfwTerminate();

            return 0;
        }

    };

```



```cpp
#include "../../Common/Window.h"

#include "../CSC8503Common/StateMachine.h"
#include "../CSC8503Common/StateTransition.h"
#include "../CSC8503Common/State.h"

#include "../CSC8503Common/NavigationGrid.h"

#include "TutorialGame.h"
#include "../CSC8503Common/PushdownMachine.h"
#include "../CSC8503Common/PushdownState.h"

#include "../GameTech/GuiManager.h"

using namespace NCL;
using namespace CSC8503;

/*

The main function should look pretty familar to you!
We make a window, and then go into a while loop that repeatedly
runs our 'game' until we press escape. Instead of making a 'renderer'
and updating it, we instead make a whole game, and repeatedly update that,
instead.

This time, we've added some extra functionality to the window class - we can
hide or show the

*/

Window* w;
TutorialGame* g;

int reset;

enum MenuOptions
{
    l1,
    l2,
    quit,
};
```



```cpp
    };


int main() {
    Window* w = Window::CreateGameWindow("CSC8503 Game technology!", 1280, 720);

    if (!w->HasInitialised()) {
        return -1;
    }
    srand(time(0));
    w->ShowOSPointer(false);
    w->LockMouseToWindow(true);

    GuiManager* g1;

    g1->InitlGui();

    TutorialGame* g = new TutorialGame();
    w->GetTimer()->GetTimeDeltaSeconds(); //Clear the timer so we don't get a larget first dt!
    while (w->UpdateWindow() && !Window::GetKeyboard()->KeyDown(KeyboardKeys::ESCAPE)) {
        float dt = w->GetTimer()->GetTimeDeltaSeconds();
        if (dt > 0.1f) {
            std::cout << "Skipping large time delta" << std::endl;
            continue; //must have hit a breakpoint or something to have a 1 second frame time!
        }
        if (Window::GetKeyboard()->KeyPressed(KeyboardKeys::PRIOR)) {
            w->ShowConsole(true);
        }
        if (Window::GetKeyboard()->KeyPressed(KeyboardKeys::NEXT)) {
            w->ShowConsole(false);
        }

        if (Window::GetKeyboard()->KeyPressed(KeyboardKeys::T)) {
            w->SetWindowPosition(0, 0);
        }

        w->SetTitle("Gametech frame time:" + std::to_string(1000.0f * dt));


        g->UpdateGame(dt);
```