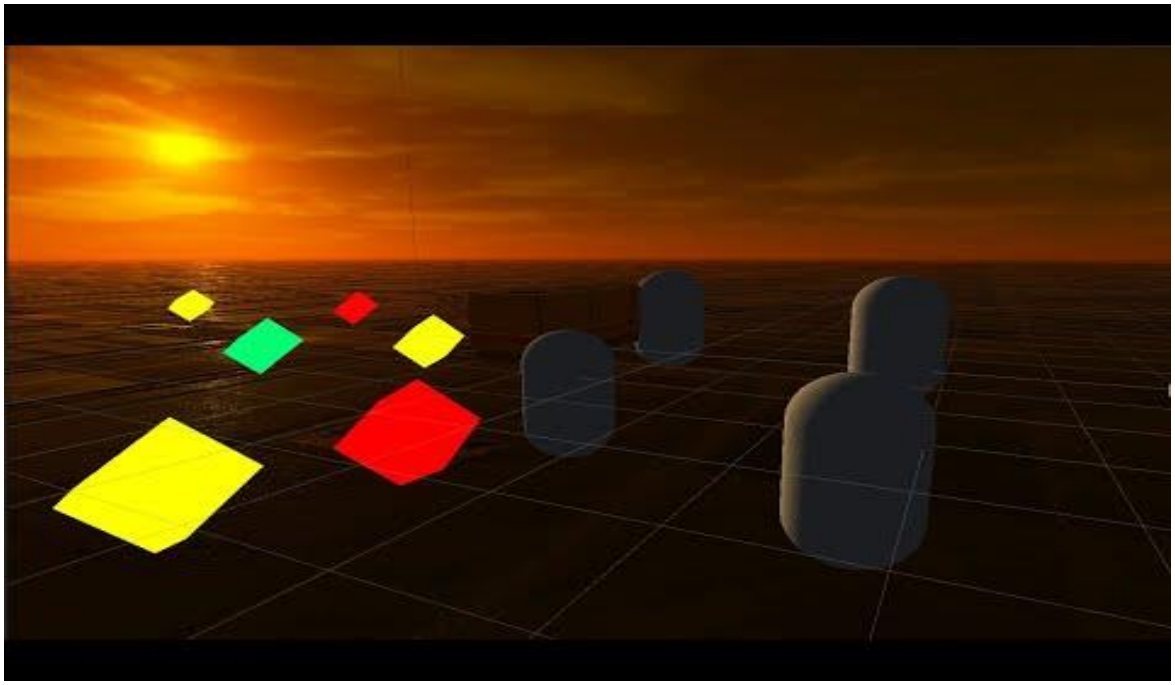
CSC8507 – Technology Implementation Appendices

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

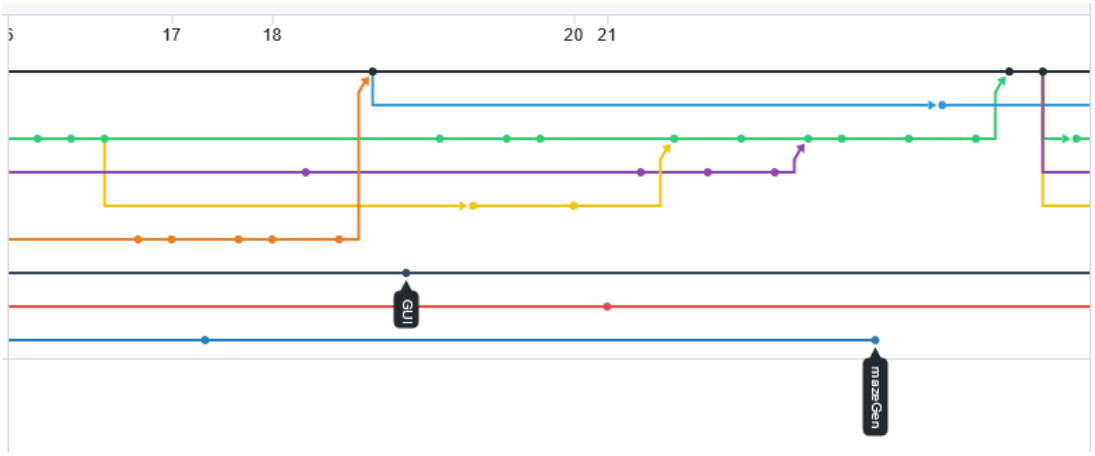
update
DavidWong85 committed 2 days ago — c624fc2

Merged SceneManager
hkjj293 committed 2 days ago — 0446959

fmod update
DavidWong85 committed 2 days ago — 1b69904

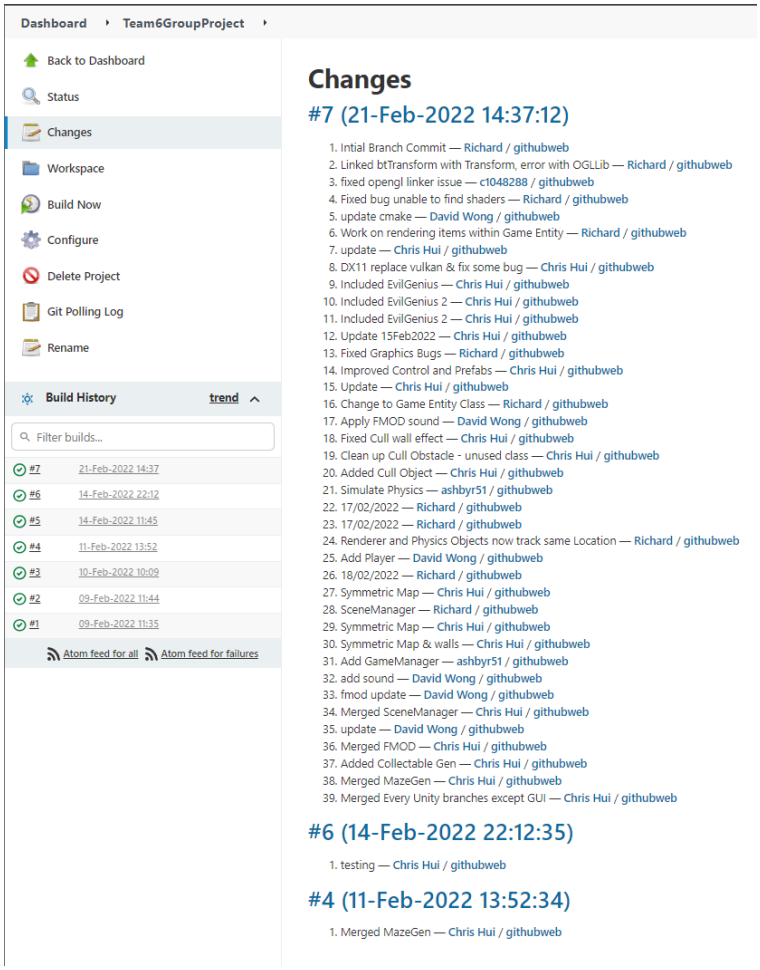
Merge branch 'SceneManager' into UnityGraphics
hkjj293 committed 2 days ago — 2dbea4d

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



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.

```h
#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();
}
```

```
//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());
```

```
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++
```

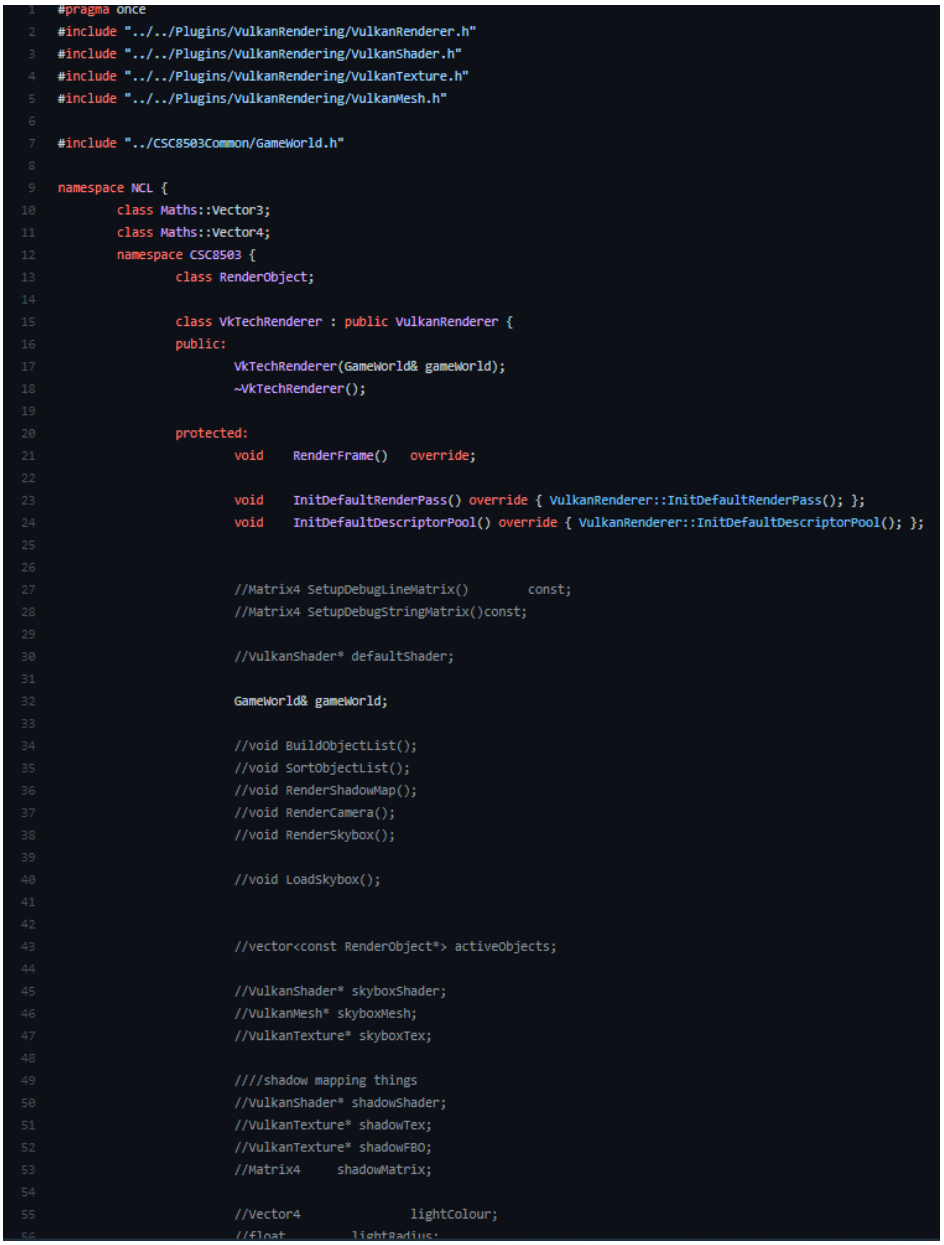
```
void AudioManager::InitSystem() {
    result = FMOD::System_Create(&system);
    ERRCHECK(result);

    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.

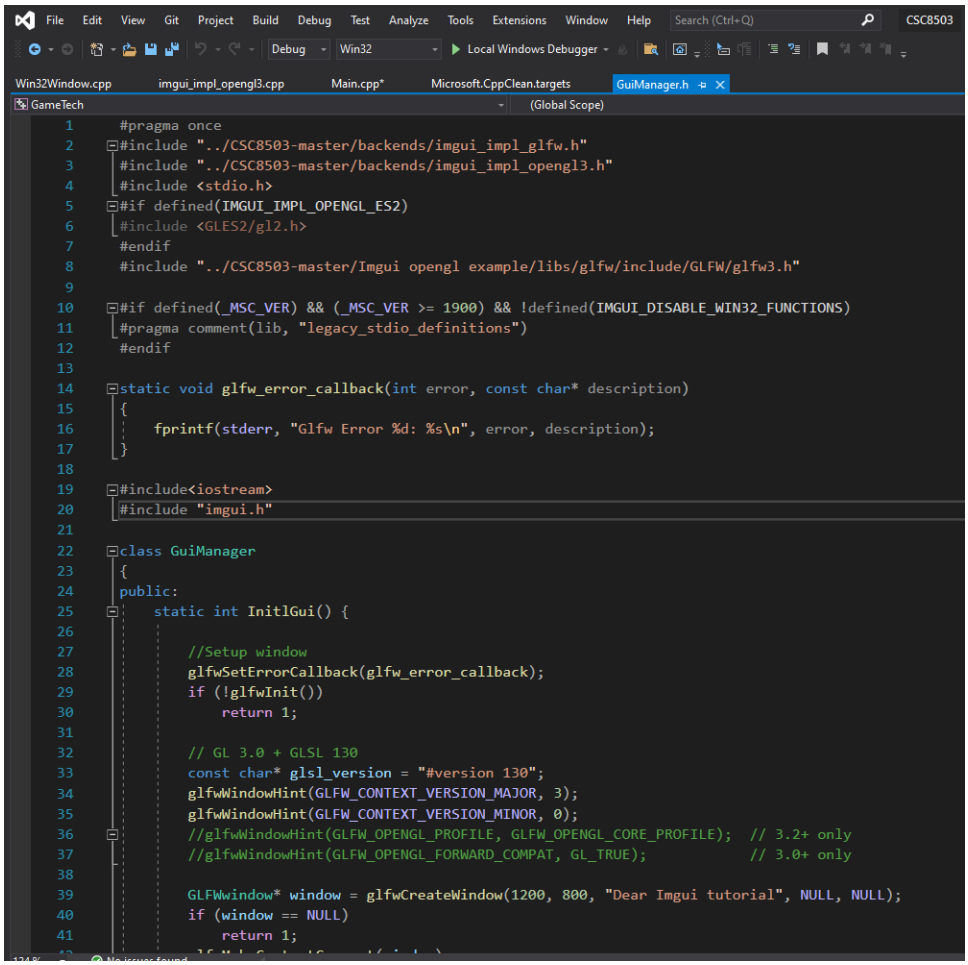
```cpp
#include "NetworkedGame.h"
#include "NetworkPlayer.h"
#include "../CSC8503Common/GameServer.h"
#include "../CSC8503Common/GameClient.h"

#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);
```

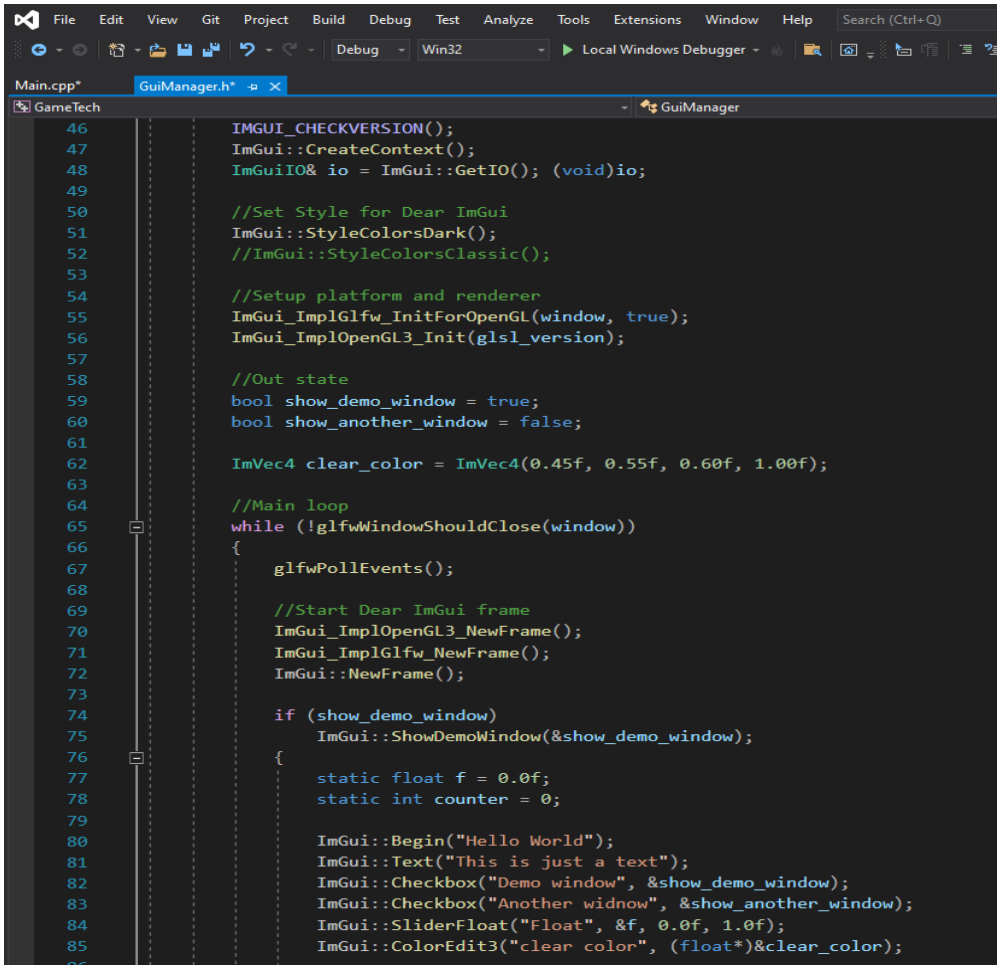
```cpp
#include "GameServer.h"
#include "GameWorld.h"
#include <iostream>

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;
```

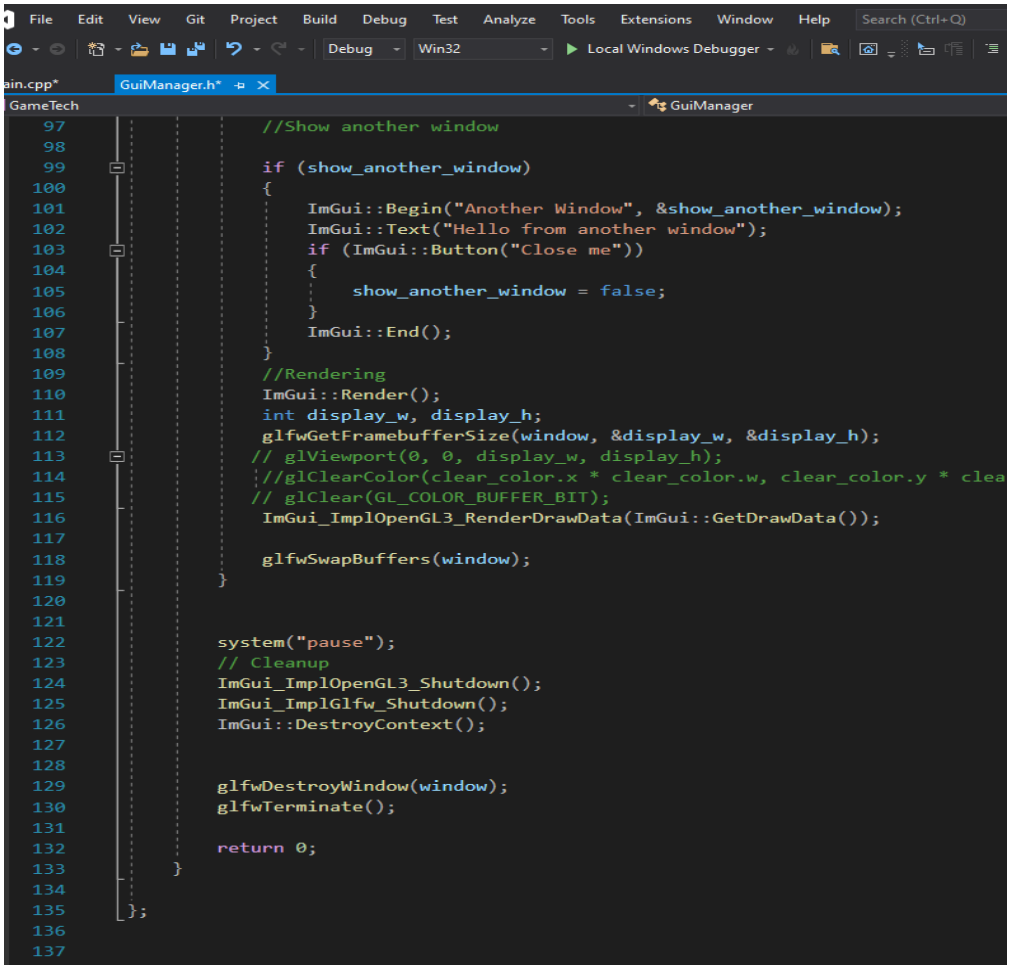
```cpp
#include "GameClient.h"
#include <iostream>
#include <string>

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)
    {
```

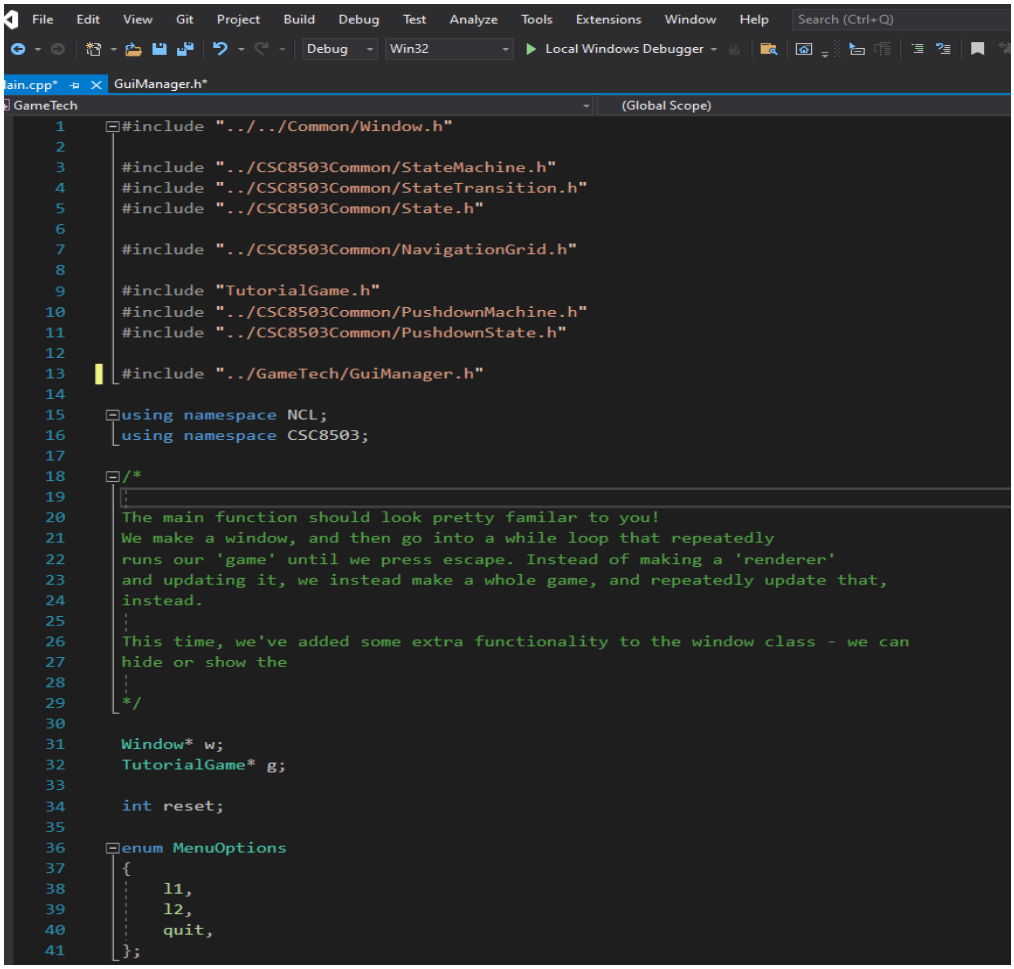
GUI– ImGui

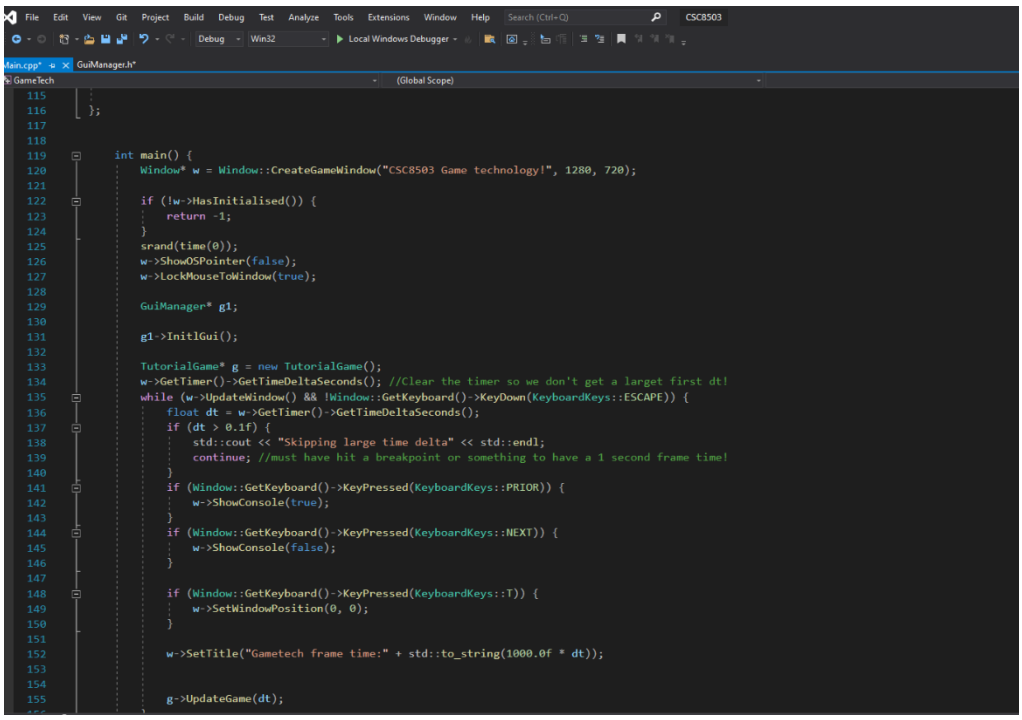
-GUI Manager class is created where imgui.h is defined to access the Imgui features
-Integrated a simple ImGui window using glfw and opengl3
-Imgui_impl_glfw and Imgui_impl_opengl3 invokes the backends used for this project
-Gui Manager here creates a function called InitlGui which is called in the main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




File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Debug
Win32
Local Windows Debugger
Main.cpp*
GuiManager.h*
GameTech
GuiManager
46 IMGUI_CHECKVERSION();
47 ImGui::CreateContext();
48 ImGuiIO& io = ImGui::GetIO(); (void)io;
49
50 //Set Style for Dear ImGui
51 ImGui::StyleColorsDark();
52 //ImGui::StyleColorsClassic();
53
54 //Setup platform and renderer
55 ImGui_ImplGlfw_InitForOpenGL(window, true);
56 ImGui_ImplOpenGL3_Init(glsl_version);
57
58 //Out state
59 bool show_demo_window = true;
60 bool show_another_window = false;
61
62 ImVec4 clear_color = ImVec4(0.45f, 0.55f, 0.60f, 1.00f);
63
64 //Main loop
65 while (!glfwWindowShouldClose(window))
66 {
67 glfwPollEvents();
68
69 //Start Dear ImGui frame
70 ImGui_ImplOpenGL3_NewFrame();
71 ImGui_ImplGlfw_NewFrame();
72 ImGui::NewFrame();
73
74 if (show_demo_window)
75 ImGui::ShowDemoWindow(&show_demo_window);
76 {
77 static float f = 0.0f;
78 static int counter = 0;
79
80 ImGui::Begin("Hello World");
81 ImGui::Text("This is just a text");
82 ImGui::Checkbox("Demo window", &show_demo_window);
83 ImGui::Checkbox("Another widnow", &show_another_window);
84 ImGui::SliderFloat("Float", &f, 0.0f, 1.0f);
85 ImGui::ColorEdit3("clear color", (float*)&clear_color);


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Debug
Win32
Local Windows Debugger
ain.cpp*
GuiManager.h*
GameTech
GuiManager
//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;
}
};


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Debug
Win32
Local Windows Debugger
ain.cpp*
GuiManager.h*
GameTech
(Global Scope)
#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,
};


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
CSC8503
Debug
Win32
Local Windows Debugger
Main.cpp*
GuiManager.h*
GameTech
(Global Scope)
};
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);