Technical Manual

Authors:
Jibran Jarwar 21353811
Ivan Shvydchenko 21317061
Supervisor: Prof. Renaat Verbruggen
Date: 01/05/2025

Introduction

This document lays out the technical guide to our project. It details our motivation, research, design, sample code, problems solved, testing and validation, results and future work.

Honeymoon is a 2D Game development tool provided to developers who wish to delve into the video game creation sphere but find tools such as Unity to be overwhelming and too complicated to start with. Our design is centered around simplicity with teaching as a secondary objective.

Our game engine enables users to create games using Lua scripting, while keeping our logic and development in C++.

Motivation

Our interest in technology and its capabilities stemmed from a love for video games both now and in the past. These games which were created from technologies that once seemed foreign and impossible to understand inspired us to learn the inner workings of them.This was furthered after using game engines such as Unity which require a lot of learning before even attempting to create anything. We wanted to create a more basic introduction to the world of game creation, one that allows the user to realize their ideas in a simpler environment while retaining the fundamentals of engines such as Unity.

Features

Key features of HoneyMoon Engine include:

- Native Scripting
- Dynamic GameObject creation
- Visual Editor
- File Manager System
- Data serialisation
- GameObject Collision
- Camera Functionality
- Game Preview Window

Research

We have learned a lot from working on this project, this has definitely helped our understanding of the fundamentals of OOP, System Architecture, Data Structures and Algorithms.In this section we will go through what tools we used, why we ended up using these tools and what we learned about the tools themselves. These include: C++, Lua, Sol library, AWS, Gitlab runners, SDL, Nlohmann Json library, ImGUI library.

C++

We chose C++ as the foundation of our 2D game engine due to performance requirements, architectural needs, and compatibility with libraries. Our engine handles real-time rendering, game object management, scripting integration, and UI interaction all within an update loop as such we having C++ low level control such as .reserve() allows us to keep consistency in frames by not reallocating memory. C++ provides compiled, native execution, which means we avoid the overhead of interpreted languages like Python. In our main loop for example we manage Gameobject instances, update Lua script states, and draw GUI elements with ImGui at a consistent FPS of around 70~80 frames per second.

Another reason we used C++ was its integration with SDL2, ImGui, and Sol2 (a C++ wrapper for Lua). SDL2 handles our rendering, window, and input subsystems, while ImGui provides an immediate-mode GUI for the editor interface. Since both are written in C/C++, we can use them directly without wrappers or foreign function interfaces allowing for minimum overhead and stability. Sol2 allows us to embed Lua scripts naturally within the engine, giving users the ability to define game behavior at runtime of the preview menu. This design with the core systems in C++, Lua as an extension is only efficient when the language can control memory, execution, and data structures.

C++ being an object oriented language is in conjunction with game engine design. GameObject, Camera, Window, and GameScreen are implemented as classes with encapsulated behavior and data. C++ allows us to control object lifetimes explicitly, use data containers like std::vector and std::unordered_map.

Lua

To provide native scripting capabilities within our game engine we considered Python and Lua because of them being high level programming languages which would make it much easier for new developers to get comfortable with programming, however we went with Lua because Lua is much easier to embed than Python since it has a Lua C++ API which was to our advantage since our game engine was coded in C++, Lua also consumes less memory than Python. Lua is a lightweight and fast scripting language which has runtime flexibility. Lua allows users to script game logic such as object behaviors, interactions, and custom events. This decouples game logic from engine architecture, enabling faster iteration and modularity. In our implementation, scripts are attached to GameObject instances and executed within the game loop function. Lua's simple syntax and low memory requirements make it well-suited for this task, especially in performance-critical environments like real-time rendering.

Sol

To integrate Lua with our C++ engine, we use Sol2, a modern C++17 wrapper library that bridges Lua and C++ with zero overhead. Sol allows us to expose entire C++ classes and functions directly to Lua without the hassle of dealing with Lua's C++ API which is much more difficult and time consuming. For instance, our RegisterGameObjectWithLua() function binds properties such as x, y, width, height, and methods like Copy(), OnCollision() from the C++ GameObject class to Lua. This enables Lua scripts to interact with engine components as if they were native Lua objects, but without sacrificing the performance and structure of C++. Through Sol, we also expose runtime data like the gameObjects table, support real-time deletion via DeleteGameObjects(), and define custom input logic using SDL bindings such as IsKeyPressed() and IsKeyHeld().

Sol provides type-safe and expressive bindings, automatically handling reference lifetimes, casting, and garbage collection integration. This eliminates common pitfalls like memory leaks or Lua state corruption. In addition, Sol's support for lambda expressions and C++ property bindings allows us to create elegant and reactive scripting interfaces. Using sol we can make sure that the Lua extension is only towards portions of our code such as gameobjects and not areas such as file manager which are system critical components that shouldn't be messed with by developers using our game engine. With this we ensure no malicious activity occurs within our game engine through the native scripting language.

AWS

Amazon Web Services is a comprehensive cloud computing platform used by Amazon that offers a wide range of services going from S3 buckets for storing information online to EC2 instances. For our project specifically we needed to create an EC2 instance that runs windows OS so we could have a gitlab runner working 24/7 for our pipeline. EC2 offers many options for running linux, Ubuntu, different window systems among other OS so we needed to figure out which type was most efficient and cost saving to use. After setting up the instance we needed to connect to the virtual machine and install all the necessary components.

Gitlab runner

Once the EC2 instance was set up we needed to install gitlab-runner so we could call our pipeline inside the virtual machine and call any tests and builds. We needed to figure out which type of runner to use since gitlab offers different options like instance, shell, docker etc. After much research we ended up with a running shell runner.

SDL

There are two popular graphic frameworks which are SDL and OpenGL, both provide rendering abilities however we went with SDL since our objective was to create a 2D game engine so while OpenGL does provide 3D but SDL doesn’t. OpenGL is also more complex to use while SDL is much easier and provides much more functionality like handling keyboard and mouse events, play sound, etc. SDL is also a much better tool to learn about game development in general and because of the limited time frame to create a functioning game engine SDL was the better choice for our case.

Nlohmann Json

This is a library to allow for easy use of json in C++, like it says on their github “Json for Modern C++”. This library is a single header which allows the use of json as easily as any other built-in C++ types. We needed to allow saving in our game engine which could have been done by saving either a binary file or a json formatted file, while binary is much faster it's harder to read and edit if any issues occur, in cases of debugging it would also be much harder with binary files. This is why we went with using Json data instead because of its easy use, the trade off is computation time however.

ImGUI

This is a library for creating immediate-mode graphical user interfaces (GUIs). This is used as our UI system to make the process of creation to navigating the file manager or moving gameObjects much easier by providing visual feedback to users that use our application, like the other libraries we used ImGUI is native to C++ so it was easier to implement into our project because of our choice of language being C++.

Design

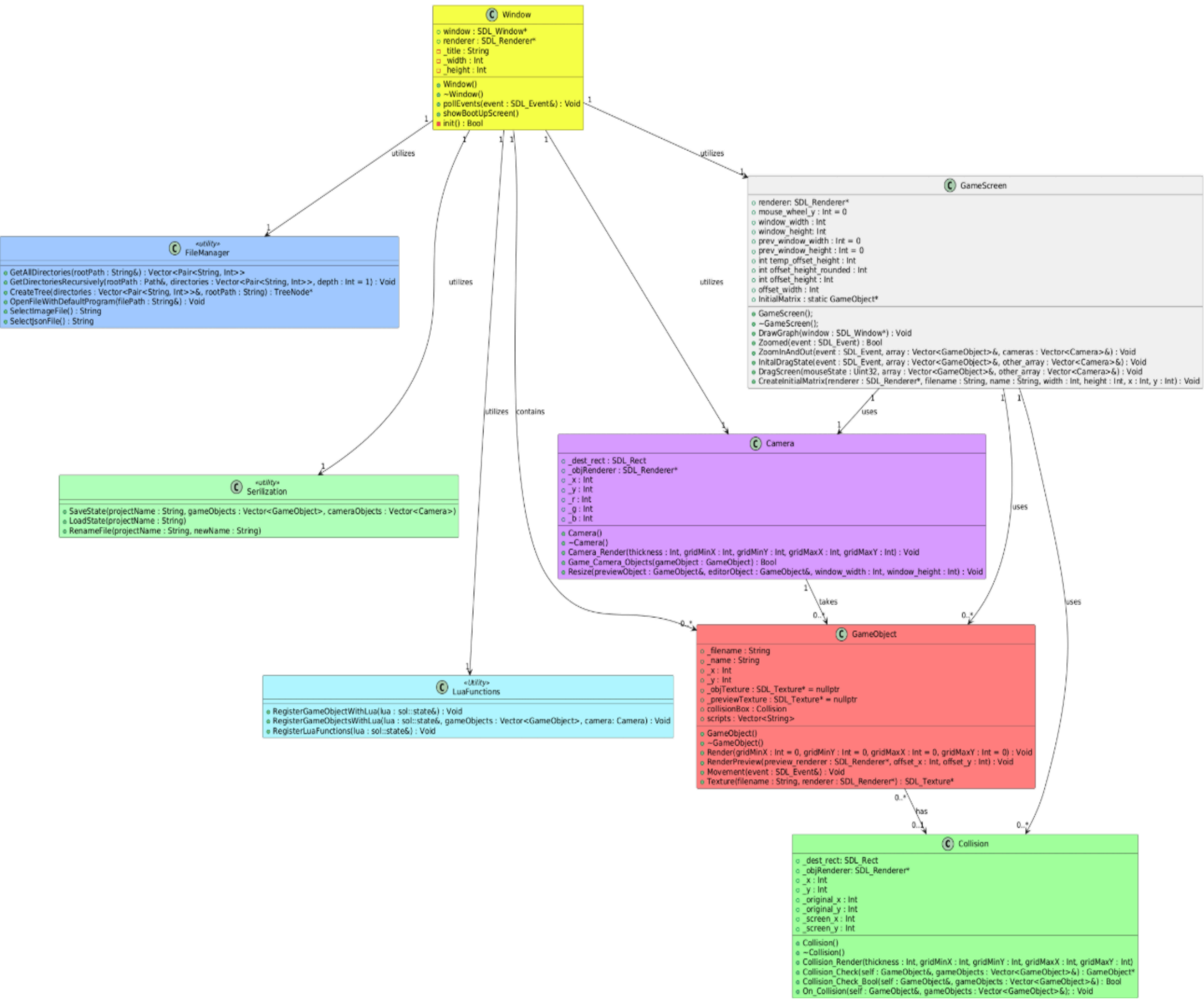

Window
window : SDL_Window*
renderer : SDL_Renderer*
_title : String
_width : Int
_height : Int
Window()
~Window()
pollEvents(event : SDL_Event&) : Void
showBootUpScreen()
init() : Bool
utilizes
utilizes
utilizes
contains
utilizes
utilizes
1
«utility»
FileManager
GetAllDirectories(rootPath : String&) : Vector<Pair<String, Int>>
GetDirectoriesRecursively(rootPath : Path&, directories : Vector<Pair<String, Int>>, depth : Int = 1) : Void
CreateTree(directories : Vector<Pair<String, Int>>&, rootPath : String) : TreeNode*
OpenFileWithDefaultProgram(filePath : String&) : Void
SelectImageFile() : String
SelectJsonFile() : String
GameScreen
renderer: SDL_Renderer*
mouse_wheel_y : Int = 0
window_width : Int
window_height: Int
prev_window_width : Int = 0
prev_window_height : Int = 0
int temp_offset_height : Int
int offset_height_rounded : Int
int offset_height : Int
offset_width : Int
InitialMatrix : static GameObject*
GameScreen();
~GameScreen();
DrawGraph(window : SDL_Window*) : Void
Zoomed(event : SDL_Event) : Bool
ZoomInAndOut(event : SDL_Event, array : Vector<GameObject>&, cameras : Vector<Camera>&) : Void
InitalDragState(event : SDL_Event, array : Vector<GameObject>&, other_array : Vector<Camera>&) : Void
DragScreen(mouseState : Uint32, array : Vector<GameObject>&, other_array : Vector<Camera>&) : Void
CreateInitialMatrix(renderer : SDL_Renderer*, filename : String, name : String, width : Int, height : Int, x : Int, y : Int) : Void
uses
uses
uses
«utility»
Serilization
SaveState(projectName : String, gameObjects : Vector<GameObject>, cameraObjects : Vector<Camera>)
LoadState(projectName : String)
RenameFile(projectName : String, newName : String)
Camera
_dest_rect : SDL_Rect
_objRenderer : SDL_Renderer*
_x : Int
_y : Int
_r : Int
_g : Int
_b : Int
Camera()
~Camera()
Camera_Render(thickness : Int, gridMinX : Int, gridMinY : Int, gridMaxX : Int, gridMaxY : Int) : Void
Game_Camera_Objects(gameObject : GameObject) : Bool
Resize(previewObject : GameObject&, editorObject : GameObject&, window_width : Int, window_height : Int) : Void
takes
0..*
«Utility»
LuaFunctions
RegisterGameObjectWithLua(lua : sol::state&) : Void
RegisterGameObjectsWithLua(lua : sol::state&, gameObjects : Vector<GameObject>, camera: Camera) : Void
RegisterLuaFunctions(lua : sol::state&) : Void
GameObject
_filename : String
_name : String
_x : Int
_y : Int
_objTexture : SDL_Texture* = nullptr
_previewTexture : SDL_Texture* = nullptr
collisionBox : Collision
scripts : Vector<String>
GameObject()
~GameObject()
Render(gridMinX : Int = 0, gridMinY : Int = 0, gridMaxX : Int = 0, gridMaxY : Int = 0) : Void
RenderPreview(preview_renderer : SDL_Renderer*, offset_x : Int, offset_y : Int) : Void
Movement(event : SDL_Event&) : Void
Texture(filename : String, renderer : SDL_Renderer*) : SDL_Texture*
has
0..1
Collision
_dest_rect: SDL_Rect
_objRenderer: SDL_Renderer*
_x : Int
_y : Int
_original_x : Int
_original_y : Int
_screen_x : Int
_screen_y : Int
Collision()
~Collision()
Collision_Render(thickness : Int, gridMinX : Int, gridMinY : Int, gridMaxX : Int, gridMaxY : Int)
Collision_Check(self : GameObject&, gameObjects : Vector<GameObject>&) : GameObject*
Collision_Check_Bool(self : GameObject&, gameObjects : Vector<GameObject>&) : Bool
On_Collision(self : GameObject&, gameObjects : Vector<GameObject>&); : Void

Figure 1: Full Class Diagram (Doesn't Contain all variables and methods because of size issue)

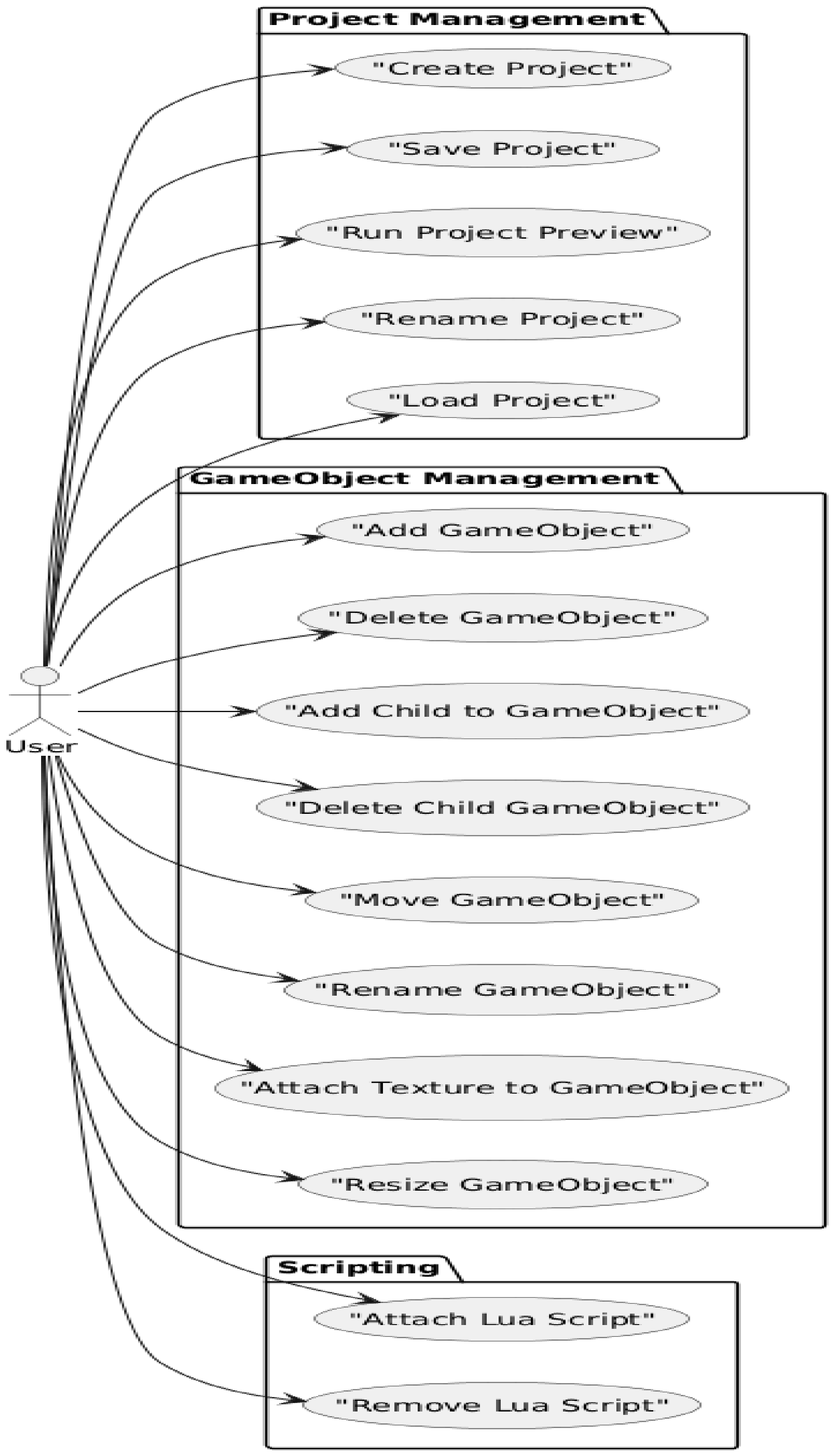


Figure 2: Use Case Diagram For Game Engine

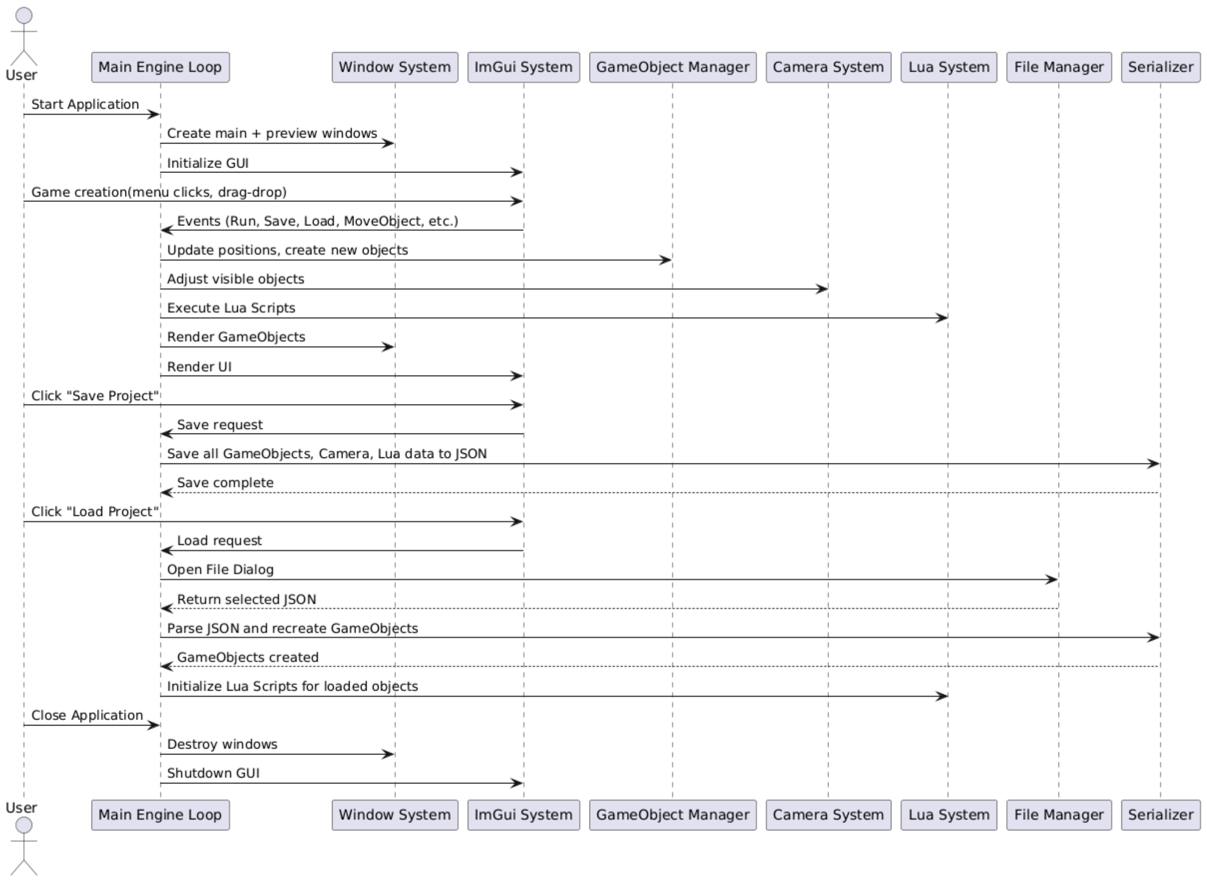


Figure 3: General Sequence Diagram for Game Engine

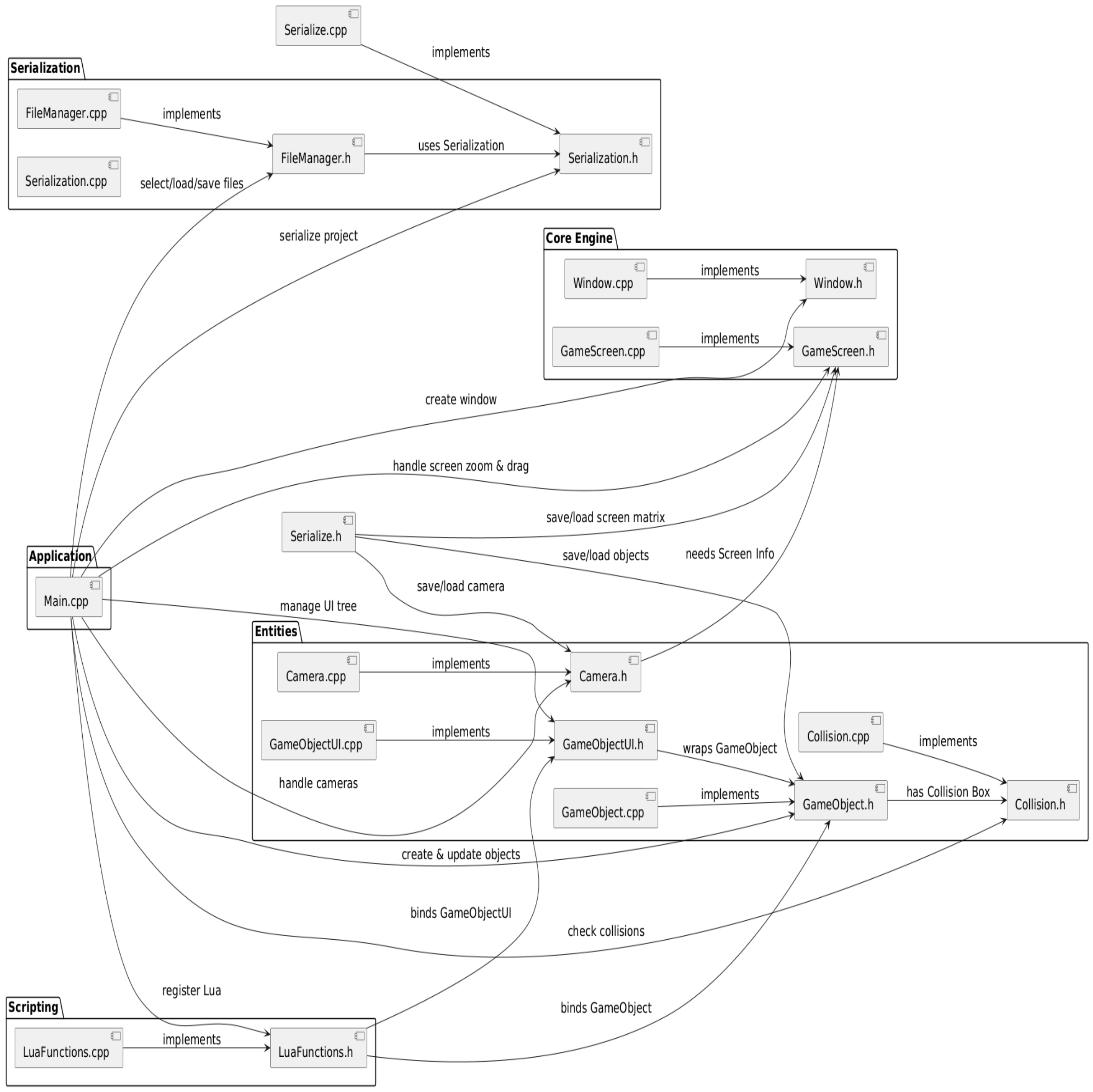
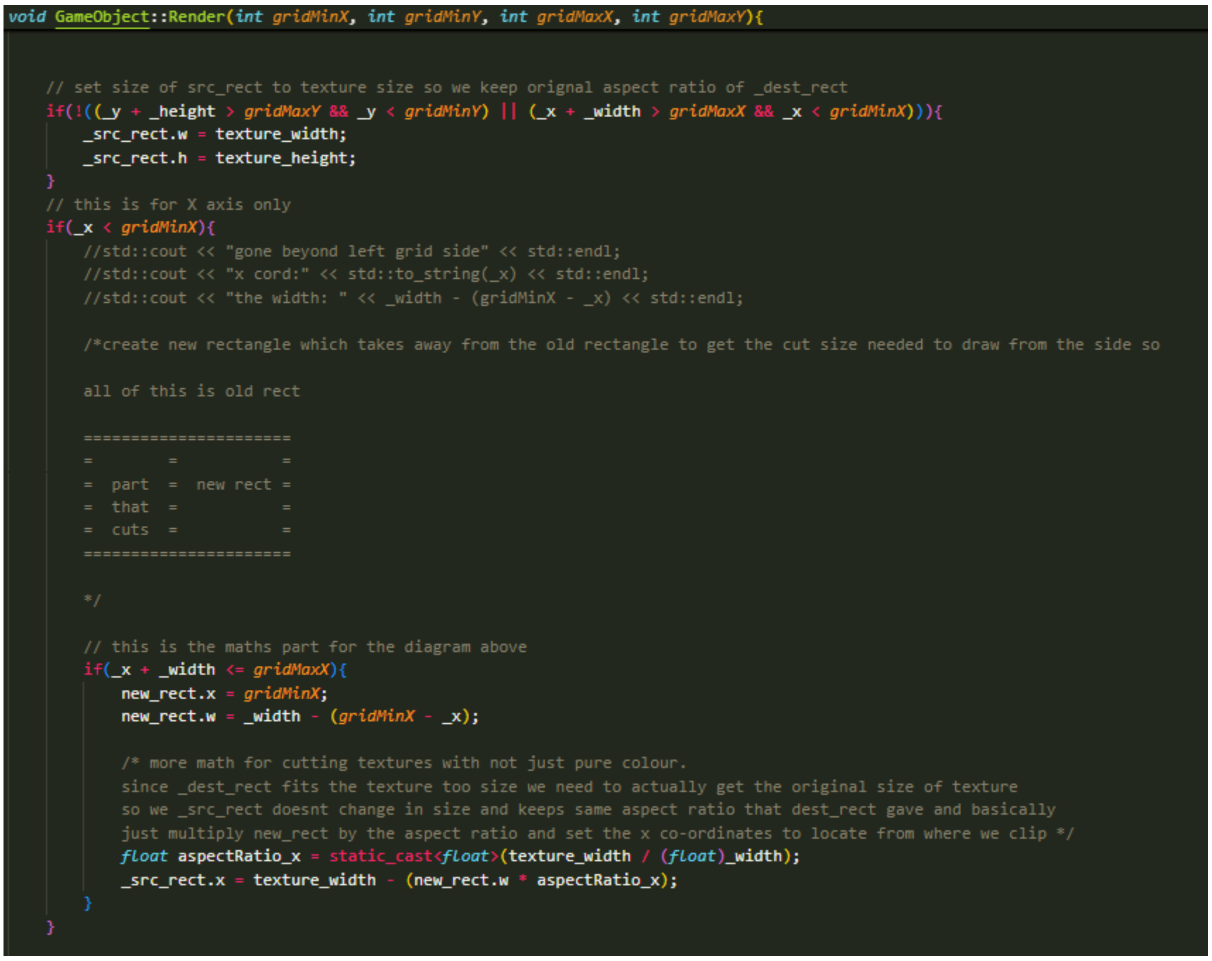


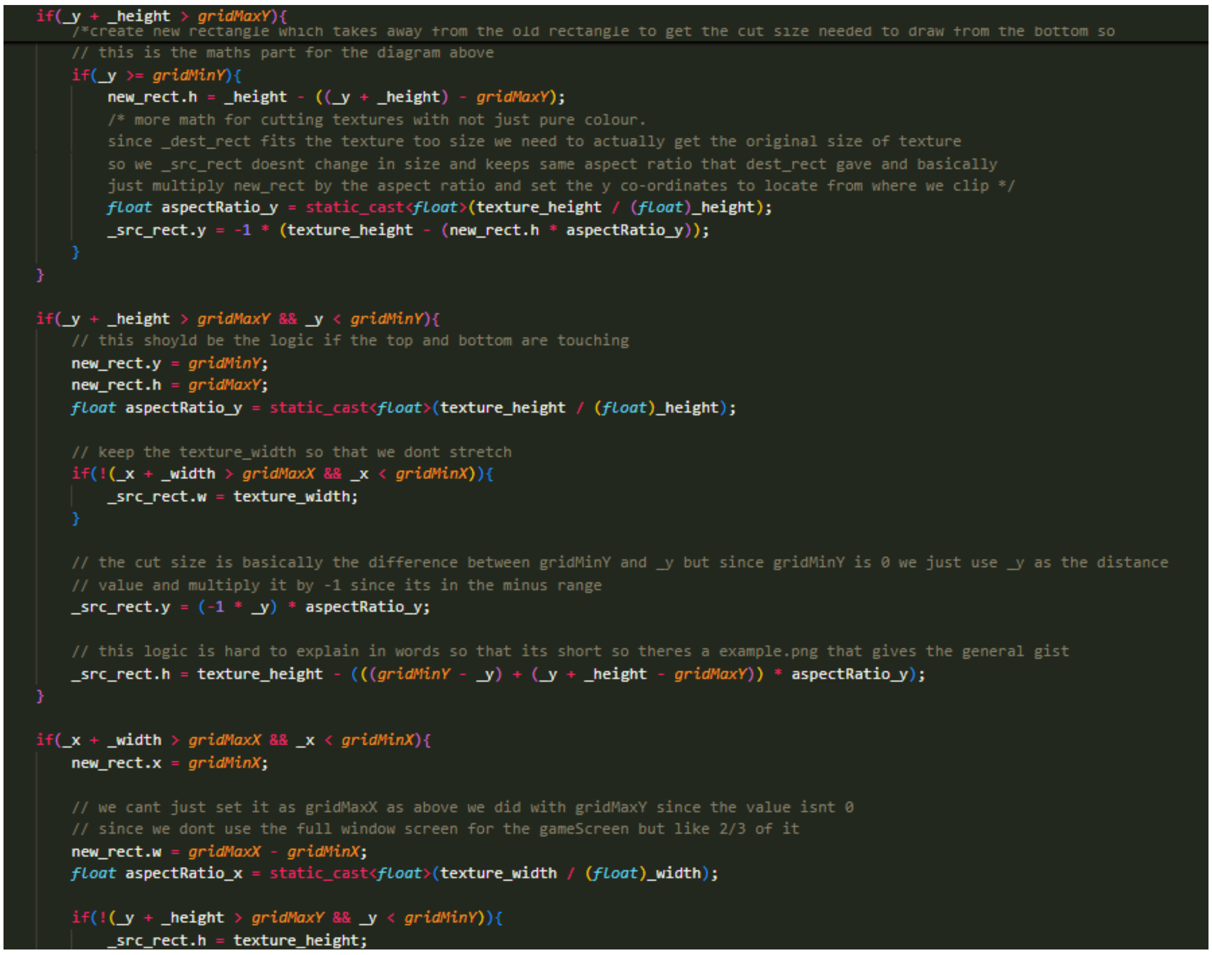
Figure 4: Architecture Diagram For Game Engine

Sample Code



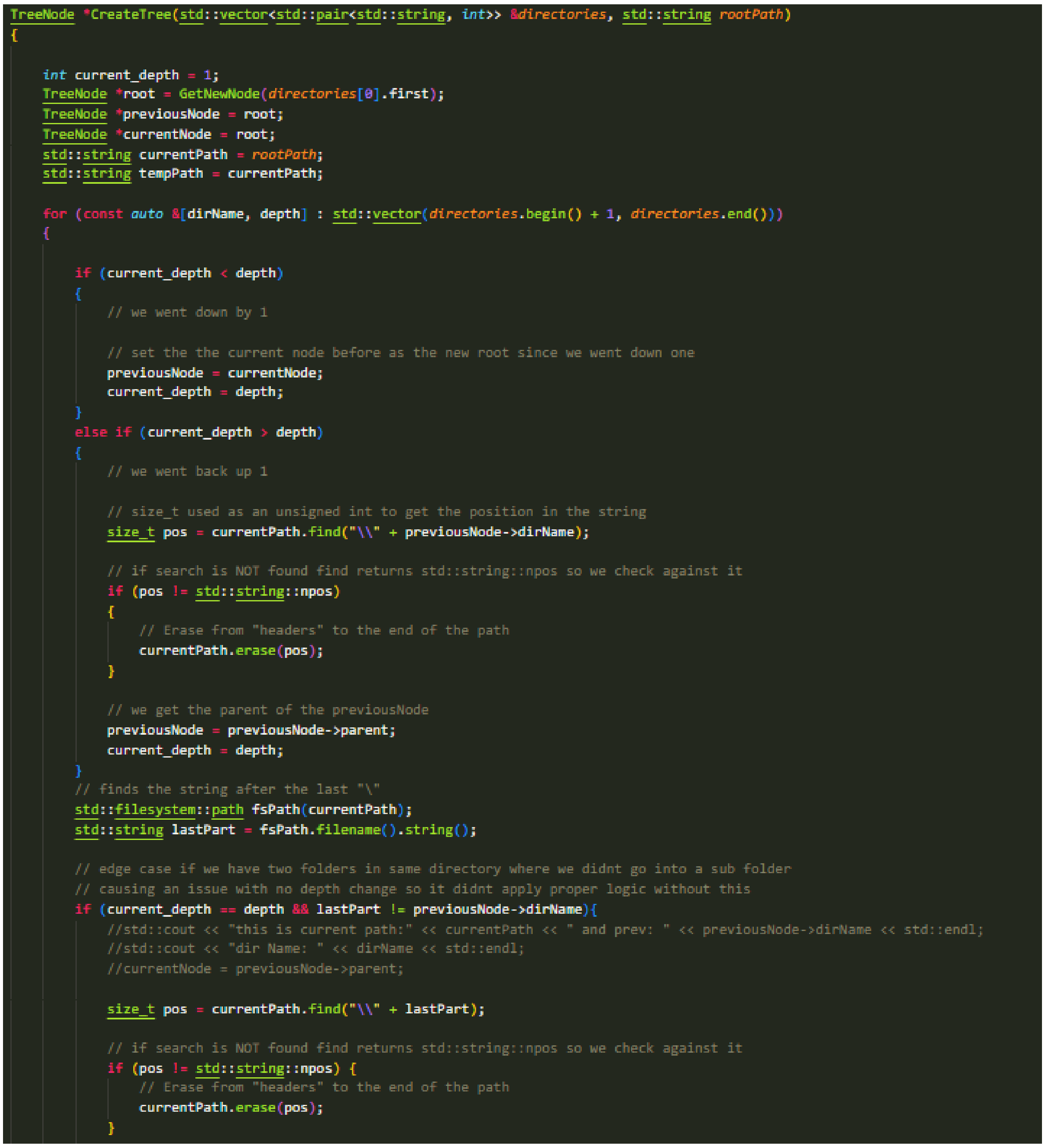
```cpp
void GameObject::Render(int gridMinX, int gridMinY, int gridMaxX, int gridMaxY){


    // set size of src_rect to texture size so we keep orignal aspect ratio of _dest_rect
    if(!((_y + _height > gridMaxY && _y < gridMinY) || (_x + _width > gridMaxX && _x < gridMinX))){
        _src_rect.w = texture_width;
        _src_rect.h = texture_height;
    }
    // this is for X axis only
    if(_x < gridMinX){
        //std::cout << "gone beyond left grid side" << std::endl;
        //std::cout << "x cord:" << std::to_string(_x) << std::endl;
        //std::cout << "the width: " << _width - (gridMinX - _x) << std::endl;

        /*create new rectangle which takes away from the old rectangle to get the cut size needed to draw from the side so

        all of this is old rect

        ======================
        =        =           =
        =  part  =  new rect =
        =  that  =           =
        =  cuts  =           =
        ======================

        */

        // this is the maths part for the diagram above
        if(_x + _width <= gridMaxX){
            new_rect.x = gridMinX;
            new_rect.w = _width - (gridMinX - _x);

            /* more math for cutting textures with not just pure colour.
            since _dest_rect fits the texture too size we need to actually get the original size of texture
            so we _src_rect doesnt change in size and keeps same aspect ratio that dest_rect gave and basically
            just multiply new_rect by the aspect ratio and set the x co-ordinates to locate from where we clip */
            float aspectRatio_x = static_cast<float>(texture_width / (float)_width);
            _src_rect.x = texture_width - (new_rect.w * aspectRatio_x);
        }
    }
```

Figure 5: Some Logic behind rendering of GameObject.



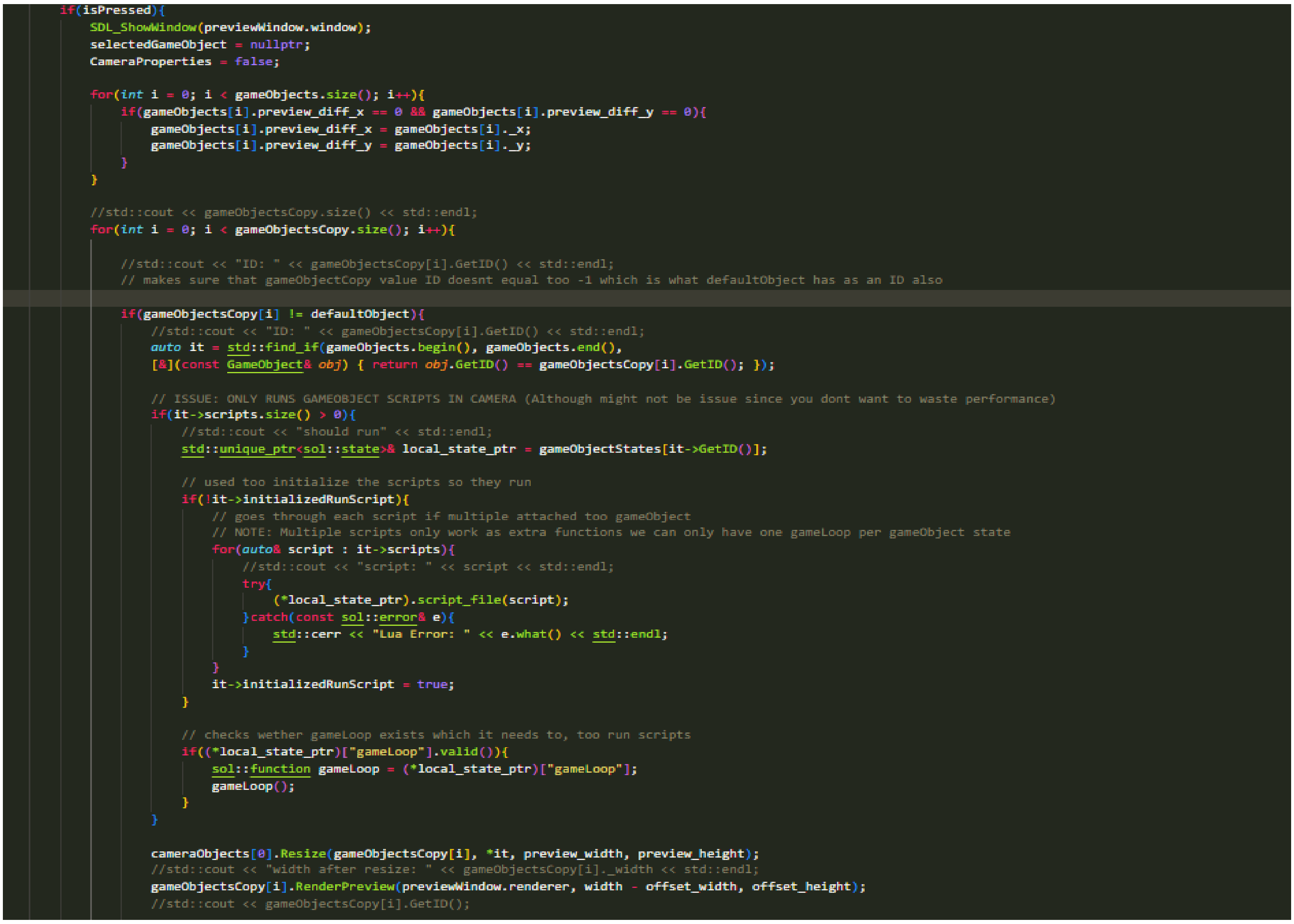
```
if(_y + _height > gridMaxY){
    /*create new rectangle which takes away from the old rectangle to get the cut size needed to draw from the bottom so
    // this is the maths part for the diagram above
    if(_y >= gridMinY){
        new_rect.h = _height - ((_y + _height) - gridMaxY);
        /* more math for cutting textures with not just pure colour.
        since _dest_rect fits the texture too size we need to actually get the original size of texture
        so we _src_rect doesnt change in size and keeps same aspect ratio that dest_rect gave and basically
        just multiply new_rect by the aspect ratio and set the y co-ordinates to locate from where we clip */
        float aspectRatio_y = static_cast<float>(texture_height / (float)_height);
        _src_rect.y = -1 * (texture_height - (new_rect.h * aspectRatio_y));
    }
}

if(_y + _height > gridMaxY && _y < gridMinY){
    // this shoyld be the logic if the top and bottom are touching
    new_rect.y = gridMinY;
    new_rect.h = gridMaxY;
    float aspectRatio_y = static_cast<float>(texture_height / (float)_height);

    // keep the texture_width so that we dont stretch
    if(!(_x + _width > gridMaxX && _x < gridMinX)){
        _src_rect.w = texture_width;
    }

    // the cut size is basically the difference between gridMinY and _y but since gridMinY is 0 we just use _y as the distance
    // value and multiply it by -1 since its in the minus range
    _src_rect.y = (-1 * _y) * aspectRatio_y;

    // this logic is hard to explain in words so that its short so theres a example.png that gives the general gist
    _src_rect.h = texture_height - (((gridMinY - _y) + (_y + _height - gridMaxY)) * aspectRatio_y);
}

if(_x + _width > gridMaxX && _x < gridMinX){
    new_rect.x = gridMinX;

    // we cant just set it as gridMaxX as above we did with gridMaxY since the value isnt 0
    // since we dont use the full window screen for the gameScreen but like 2/3 of it
    new_rect.w = gridMaxX - gridMinX;
    float aspectRatio_x = static_cast<float>(texture_width / (float)_width);

    if(!(_y + _height > gridMaxY && _y < gridMinY)){
        _src_rect.h = texture_height;
```

Figure 6: Further Logic Behind rendering of GameObject.



```cpp
TreeNode *CreateTree(std::vector<std::pair<std::string, int>> &directories, std::string rootPath)
{

    int current_depth = 1;
    TreeNode *root = GetNewNode(directories[0].first);
    TreeNode *previousNode = root;
    TreeNode *currentNode = root;
    std::string currentPath = rootPath;
    std::string tempPath = currentPath;

    for (const auto &[dirName, depth] : std::vector(directories.begin() + 1, directories.end()))
    {

        if (current_depth < depth)
        {
            // we went down by 1

            // set the the current node before as the new root since we went down one
            previousNode = currentNode;
            current_depth = depth;
        }
        else if (current_depth > depth)
        {
            // we went back up 1

            // size_t used as an unsigned int to get the position in the string
            size_t pos = currentPath.find("\\" + previousNode->dirName);

            // if search is NOT found find returns std::string::npos so we check against it
            if (pos != std::string::npos)
            {
                // Erase from "headers" to the end of the path
                currentPath.erase(pos);
            }

            // we get the parent of the previousNode
            previousNode = previousNode->parent;
            current_depth = depth;
        }
        // finds the string after the last "\"
        std::filesystem::path fsPath(currentPath);
        std::string lastPart = fsPath.filename().string();

        // edge case if we have two folders in same directory where we didnt go into a sub folder
        // causing an issue with no depth change so it didnt apply proper logic without this
        if (current_depth == depth && lastPart != previousNode->dirName){
            //std::cout << "this is current path:" << currentPath << " and prev: " << previousNode->dirName << std::endl;
            //std::cout << "dir Name: " << dirName << std::endl;
            //currentNode = previousNode->parent;

            size_t pos = currentPath.find("\\" + lastPart);

            // if search is NOT found find returns std::string::npos so we check against it
            if (pos != std::string::npos) {
                // Erase from "headers" to the end of the path
                currentPath.erase(pos);
            }
```

Figure 7: File Manager Using Data Tree Structure.



```cpp
    if(isPressed){
        SDL_ShowWindow(previewWindow.window);
        selectedGameObject = nullptr;
        CameraProperties = false;

        for(int i = 0; i < gameObjects.size(); i++){
            if(gameObjects[i].preview_diff_x == 0 && gameObjects[i].preview_diff_y == 0){
                gameObjects[i].preview_diff_x = gameObjects[i]._x;
                gameObjects[i].preview_diff_y = gameObjects[i]._y;
            }
        }

        //std::cout << gameObjectsCopy.size() << std::endl;
        for(int i = 0; i < gameObjectsCopy.size(); i++){

            //std::cout << "ID: " << gameObjectsCopy[i].GetID() << std::endl;
            // makes sure that gameObjectCopy value ID doesnt equal too -1 which is what defaultObject has as an ID also

            if(gameObjectsCopy[i] != defaultObject){
                //std::cout << "ID: " << gameObjectsCopy[i].GetID() << std::endl;
                auto it = std::find_if(gameObjects.begin(), gameObjects.end(),
                [&](const GameObject& obj) { return obj.GetID() == gameObjectsCopy[i].GetID(); });

                // ISSUE: ONLY RUNS GAMEOBJECT SCRIPTS IN CAMERA (Although might not be issue since you dont want to waste performance)
                if(it->scripts.size() > 0){
                    //std::cout << "should run" << std::endl;
                    std::unique_ptr<sol::state>& local_state_ptr = gameObjectStates[it->GetID()];

                    // used too initialize the scripts so they run
                    if(!it->initializedRunScript){
                        // goes through each script if multiple attached too gameObject
                        // NOTE: Multiple scripts only work as extra functions we can only have one gameLoop per gameObject state
                        for(auto& script : it->scripts){
                            //std::cout << "script: " << script << std::endl;
                            try{
                                (*local_state_ptr).script_file(script);
                            }catch(const sol::error& e){
                                std::cerr << "Lua Error: " << e.what() << std::endl;
                            }
                        }
                        it->initializedRunScript = true;
                    }

                    // checks wether gameLoop exists which it needs to, too run scripts
                    if((*local_state_ptr)["gameLoop"].valid()){
                        sol::function gameLoop = (*local_state_ptr)["gameLoop"];
                        gameLoop();
                    }
                }

                cameraObjects[0].Resize(gameObjectsCopy[i], *it, preview_width, preview_height);
                //std::cout << "width after resize: " << gameObjectsCopy[i]._width << std::endl;
                gameObjectsCopy[i].RenderPreview(previewWindow.renderer, width - offset_width, offset_height);
                //std::cout << gameObjectsCopy[i].GetID();
```

Figure 8: The Logic behind how we work the Preview/Game Window

Problems Solved

Rendering

Problem:

1. With Rendering, the Problem we had was how to cut the textures in a smart way so that they don't appear to stretch and properly decrease the size of the texture. It took a lot of time to find a way to make the texture disappear.

Solution

2. The way we achieved this is by smartly decreasing the size of the rectangle, this makes it appear as though the texture is vanishing but it's all just calculated geometry. The Final Issue with the Texture Was If a Texture touched two edges we needed to apply different logic since we need to cut from both sides, this was hard to imagine but eventually we got it down by having a mix of what we take from the texture and resizing the rectangle since SDL uses SDL_rect.

File Manager

Problem

1. To be able to include the functionality of a File Manager to be able to open and use the files we couldn't figure out how to dynamically store Files and Directories.
2. We later also discovered an edge case where if the depth didn't change by going into a sub directory the application crashed.

Solution

1. After much time of us trying many methods, we ended up trying to use the Tree data structure that we learned from our 2nd year of Computer Science in the module Algorithms and Data Structures. This turned out to work very well and we achieved the result we wanted with our file manager.

2. The way we solved the edge case problem was by removing the previous folder from the string and comparing the depth and previousNode because the logic made it so that the folder on the same level was counted as a sub directory which didn't exist before our new implementation and fix.

Game Screen

Problem

1. Giving the illusion that the screen had infinite space was hard to do since the SDL coordinates are different to our in game coordinates.

2. The issue that accrued later was accurately creating the gameObjects when the window size gets changed i.e fullscreen to window mode, when we stored a value there were some slight offsets which sometimes broke the system and the object was created in different places.

Solution

1. We solved the illusion problem by adding the drag option so you can move the objects and having separate static x and y coordinates when the user moves using

the GameObject properties tab which are the actual values we use for the developers coordinates.

2. to counter the issue with the offsets we decided to store an initial Matrix gameObject and track its co-ords and create objects based on where its located which ended up solving the issue

Scripting

Problem

1. Including Native Scripting was a difficult challenge where we initially were going to use the LuaCpp API that Lua provides but it was a lot more difficult and would require much more time to get acquainted with and to incorporate it.
2. We also needed to figure out how to incorporate multiple scripts dynamically.

Solution

1. To solve our first issue we decided to go with the SOL library to make the embedding easier and save development time.

2. At first we considered using threads but we were worried this would reflect performance especially if not handled properly since it uses multiple CPU cores so we ended up running the scripts sequentially but because of the quick runtimes we never had any issues with delays in scripts being performed simultaneously . And the way we have multiple scripts is when we attach a script to a game engine we create a sol::state which is a Lua VM. Using unique pointers to store these states in an unordered_map to fetch by ID.

Camera/Preview Screen

Problem

1. To have a good solid 2D game engine you need cameras, the issue with this is that depending on the size of the camera we need to then display the objects accurately depending on this camera.

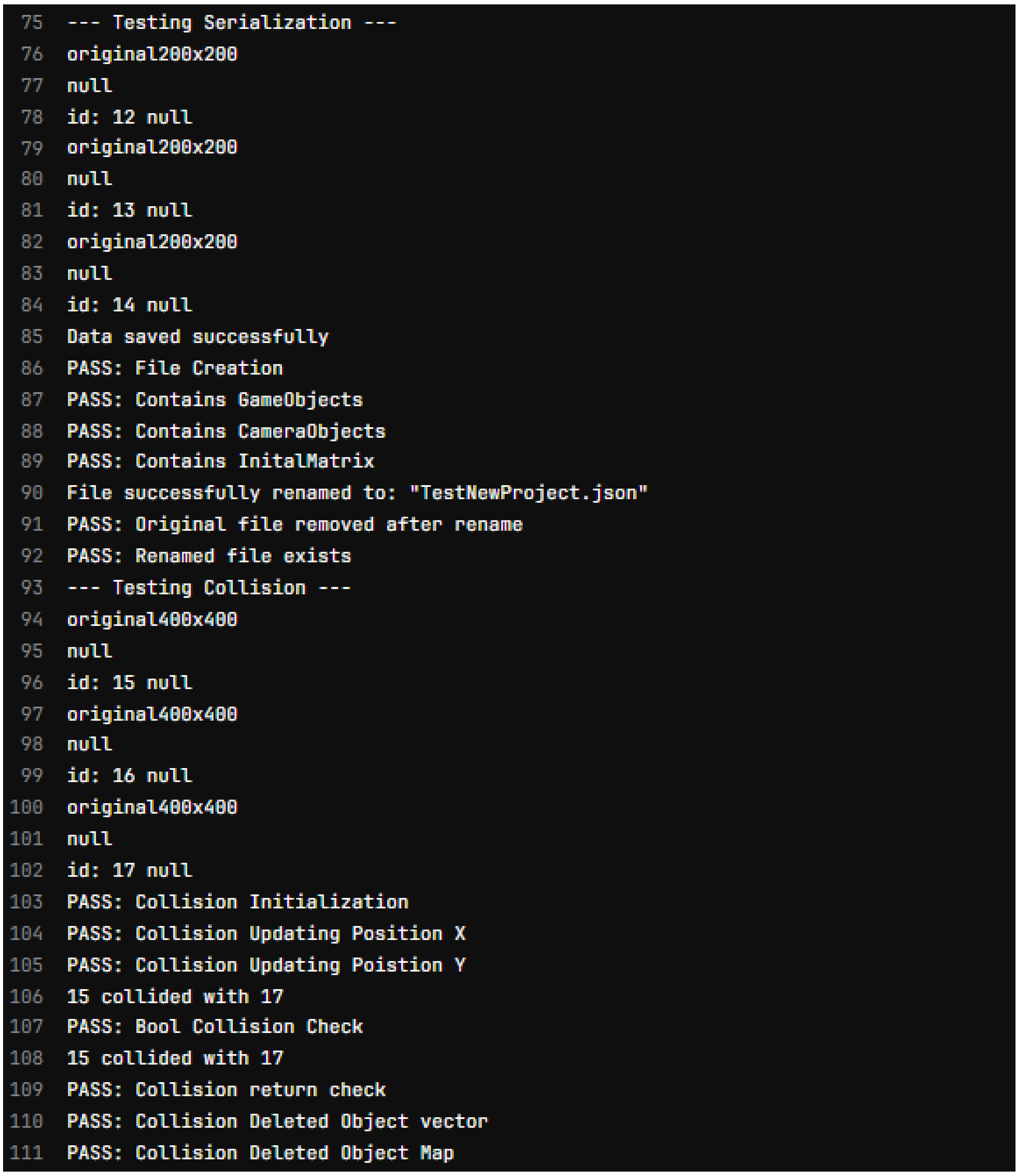
Solutions

1. we tried to change the value of the object themselves but this ended up expanding the object infinitely since every frame it would keep increasing without any damage control. The way we overcame this obstacle was by having two separate instances of Objects one for the editor and the other for the preview/game window. Once we run the game we get the editor Objects size and coordinates since it doesn't change and we made sure of this by disabling the drag feature and changing any values, and with some clever maths getting the distance from the centre of the Camera to each object we get the correct distances.

Additional
There was also the issue with when you change the preview window size to be different to the aspect of the camera the Objects may appear stretched but we decided that this was okay because this is how resolution works in general in other games.

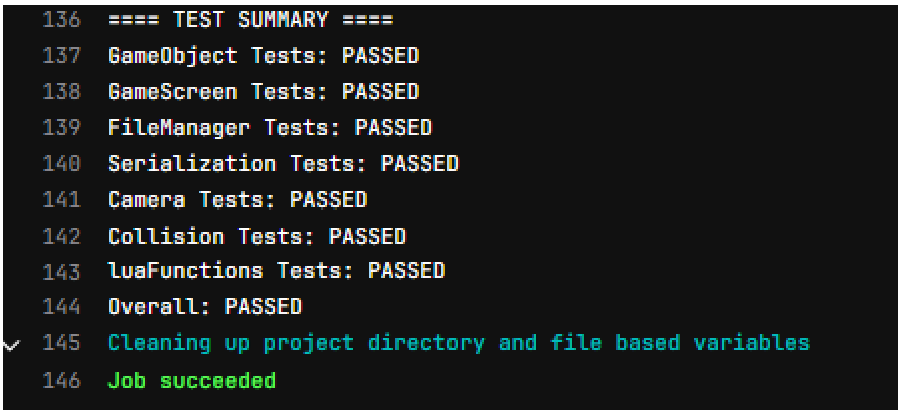
Testing and Validation

Unit Testing



```
--- Testing Serialization ---
original200x200
null
id: 12 null
original200x200
null
id: 13 null
original200x200
null
id: 14 null
Data saved successfully
PASS: File Creation
PASS: Contains GameObjects
PASS: Contains CameraObjects
PASS: Contains InitalMatrix
File successfully renamed to: "TestNewProject.json"
PASS: Original file removed after rename
PASS: Renamed file exists
--- Testing Collision ---
original400x400
null
id: 15 null
original400x400
null
id: 16 null
original400x400
null
id: 17 null
PASS: Collision Initialization
PASS: Collision Updating Position X
PASS: Collision Updating Poistion Y
15 collided with 17
PASS: Bool Collision Check
15 collided with 17
PASS: Collision return check
PASS: Collision Deleted Object vector
PASS: Collision Deleted Object Map
```

Figure 9: Different Unit tests on our classes used within our application.



```
==== TEST SUMMARY ====
GameObject Tests: PASSED
GameScreen Tests: PASSED
FileManager Tests: PASSED
Serialization Tests: PASSED
Camera Tests: PASSED
Collision Tests: PASSED
luaFunctions Tests: PASSED
Overall: PASSED
Cleaning up project directory and file based variables
Job succeeded
```

Figure 10: Overall Unit tests results on our classes (Many tests within the Class Tests themselves like in the previous Image).

User Testing

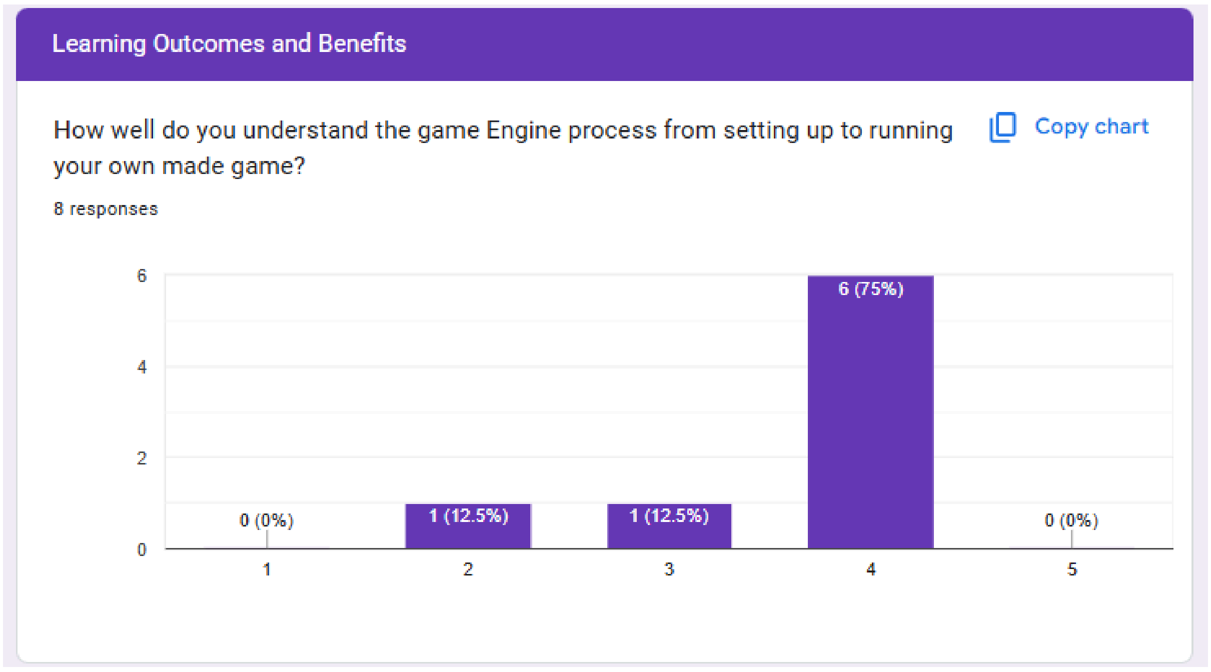
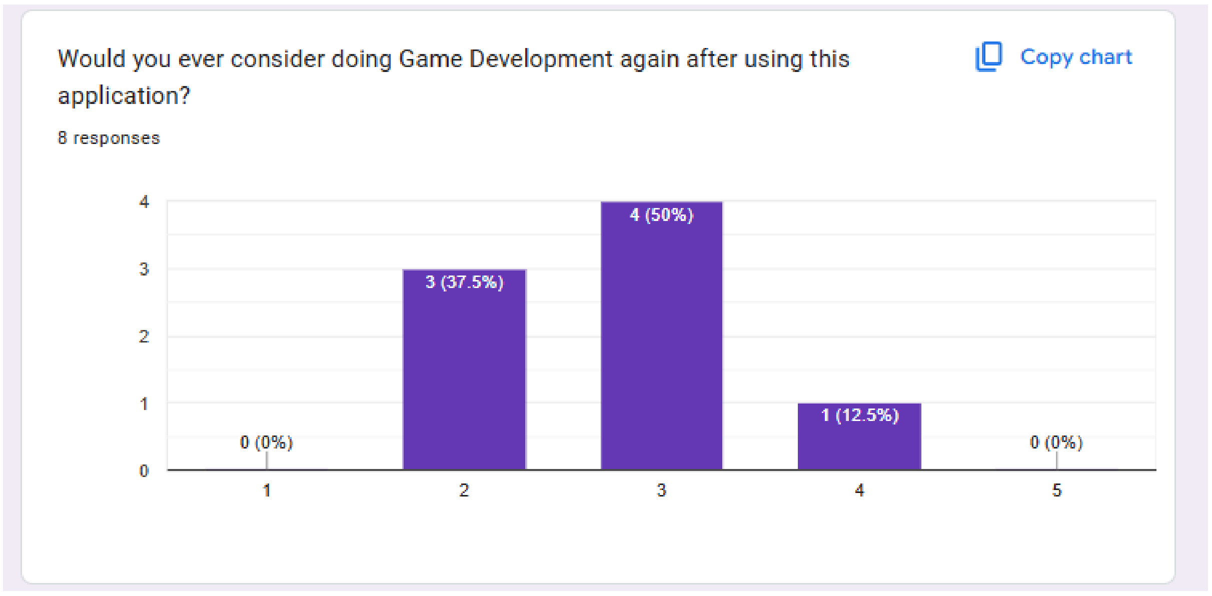


Figure 11: 1 being "don't understand at all" and 5 being "very understandable".


Would you ever consider doing Game Development again after using this application?

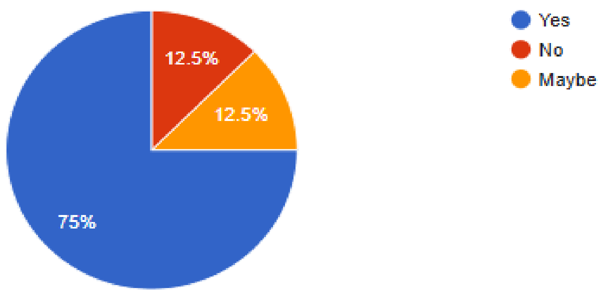
8 responses

Figure 12: 1 being "Definitely" and 5 being "Never".

Would you consider using this specific game Engine again

Copy chart

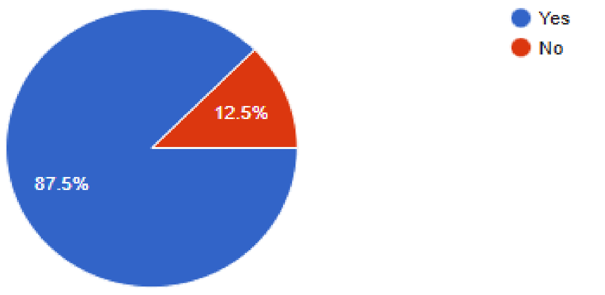
8 responses



Did the naming for all components make sense?

Copy chart

8 responses



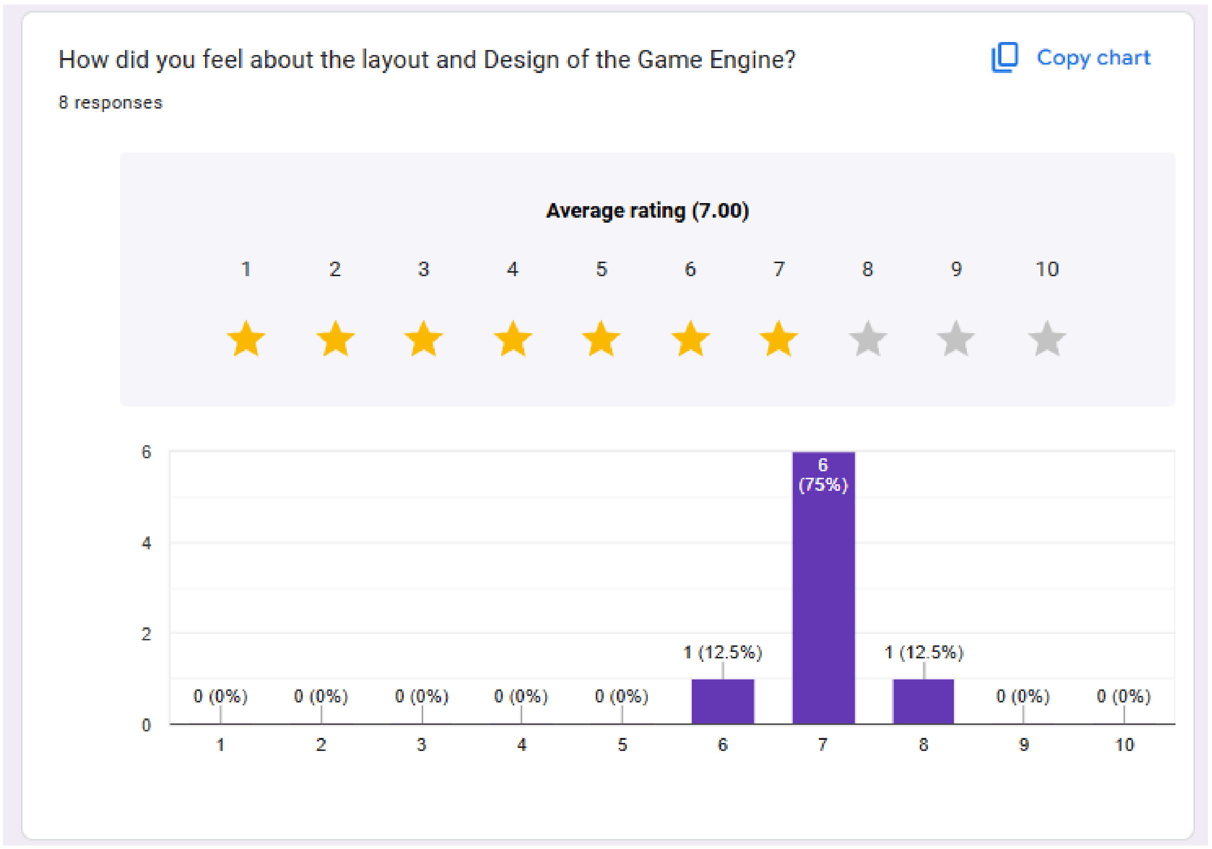
If answered No, what didn't make sense and would you recommend any name changes? Write a brief answer.

8 responses

NA

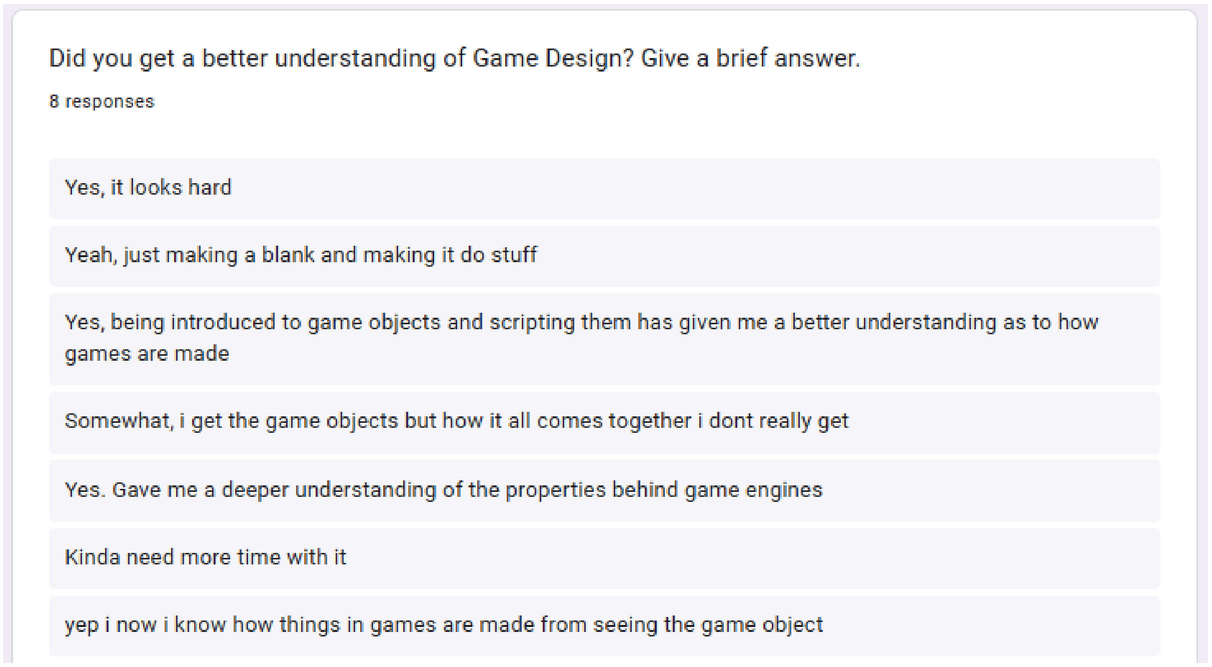
Maybe rename children to something else

Figure 13: User Testing Results on if our naming conventions made sense and would user testers like to use our application again.


How did you feel about the layout and Design of the Game Engine?

8 responses

Figure 14: Result about User Interface from User Testing.


Did you get a better understanding of Game Design? Give a brief answer.

8 responses

Yes, it looks hard

Yeah, just making a blank and making it do stuff

Yes, being introduced to game objects and scripting them has given me a better understanding as to how games are made

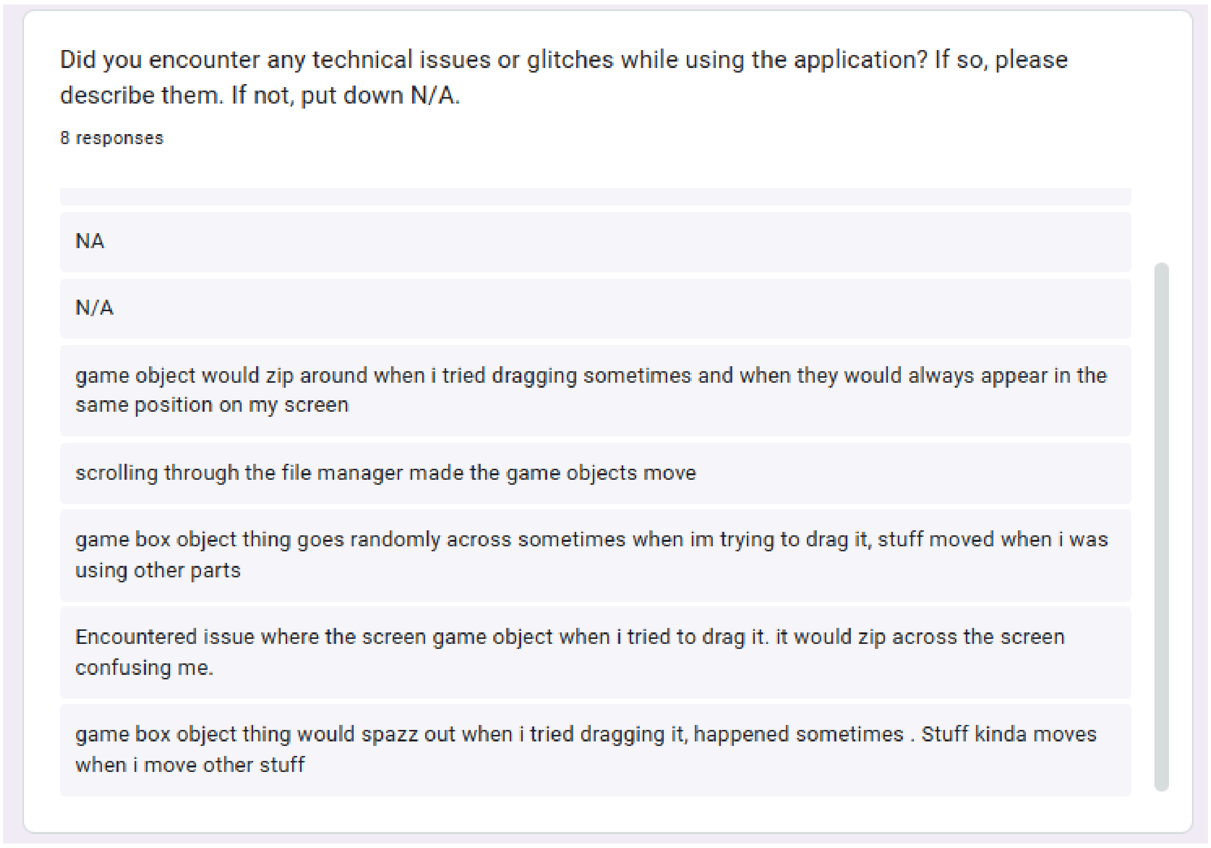
Somewhat, i get the game objects but how it all comes together i dont really get

Yes. Gave me a deeper understanding of the properties behind game engines

Kinda need more time with it

yep i now i know how things in games are made from seeing the game object

Figure 15: Answers from Users about the experience of Game Design given by our Game Engine.


Did you encounter any technical issues or glitches while using the application? If so, please describe them. If not, put down N/A.

8 responses

NA

N/A

game object would zip around when i tried dragging sometimes and when they would always appear in the same position on my screen

scrolling through the file manager made the game objects move

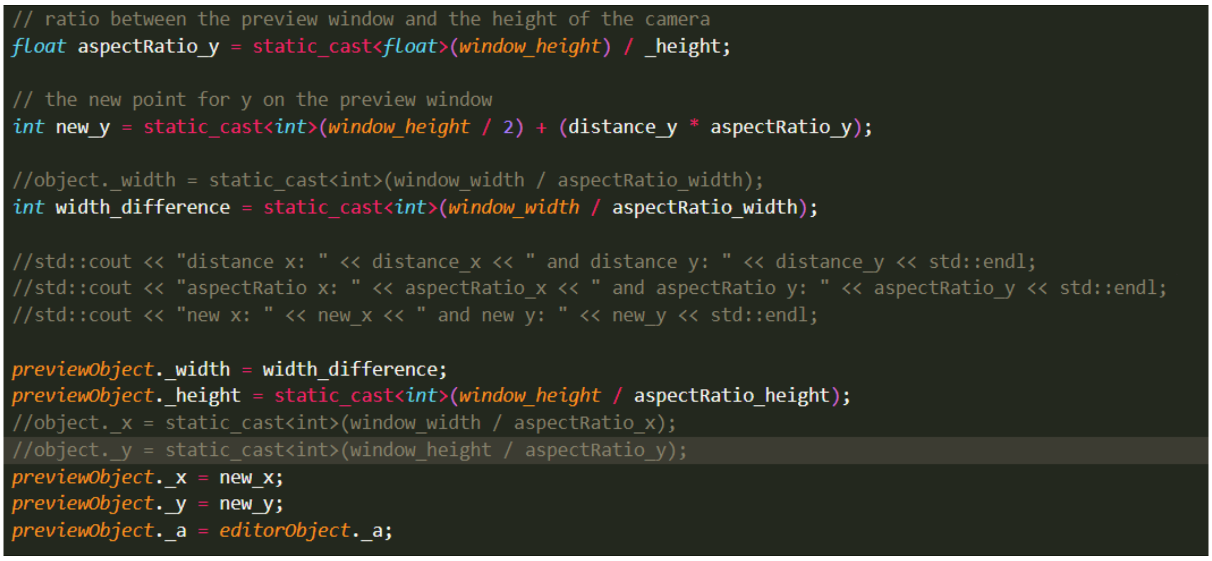
game box object thing goes randomly across sometimes when im trying to drag it, stuff moved when i was using other parts

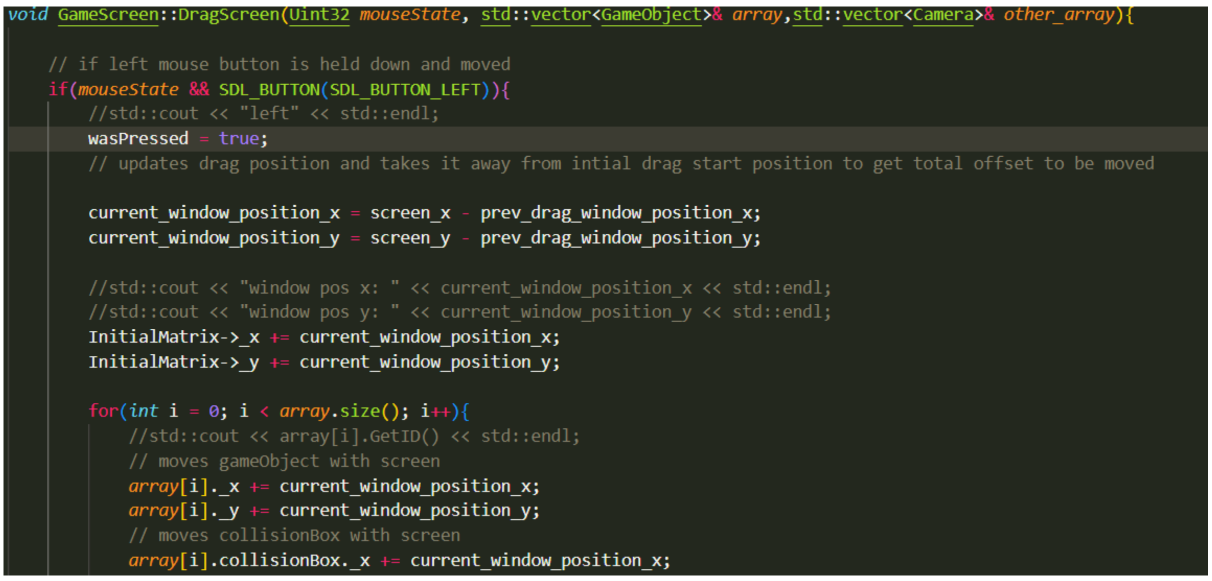
Encountered issue where the screen game object when i tried to drag it. it would zip across the screen confusing me.

game box object thing would spazz out when i tried dragging it, happened sometimes . Stuff kinda moves when i move other stuff

Figure 16: Some Technical Issues Users encountered during the use of the application.

Debugging


```
// ratio between the preview window and the height of the camera
float aspectRatio_y = static_cast<float>(window_height) / _height;

// the new point for y on the preview window
int new_y = static_cast<int>(window_height / 2) + (distance_y * aspectRatio_y);

//object._width = static_cast<int>(window_width / aspectRatio_width);
int width_difference = static_cast<int>(window_width / aspectRatio_width);

//std::cout << "distance x: " << distance_x << " and distance y: " << distance_y << std::endl;
//std::cout << "aspectRatio x: " << aspectRatio_x << " and aspectRatio y: " << aspectRatio_y << std::endl;
//std::cout << "new x: " << new_x << " and new y: " << new_y << std::endl;

previewObject._width = width_difference;
previewObject._height = static_cast<int>(window_height / aspectRatio_height);
//object._x = static_cast<int>(window_width / aspectRatio_x);
//object._y = static_cast<int>(window_height / aspectRatio_y);
previewObject._x = new_x;
previewObject._y = new_y;
previewObject._a = editorObject._a;
```
Figure 17: Debugging within application with terminal for verifying proper logic.



```cpp
void GameScreen::DragScreen(Uint32 mouseState, std::vector<GameObject>& array,std::vector<Camera>& other_array){

    // if left mouse button is held down and moved
    if(mouseState && SDL_BUTTON(SDL_BUTTON_LEFT)){
        //std::cout << "left" << std::endl;
        wasPressed = true;
        // updates drag position and takes it away from intial drag start position to get total offset to be moved

        current_window_position_x = screen_x - prev_drag_window_position_x;
        current_window_position_y = screen_y - prev_drag_window_position_y;

        //std::cout << "window pos x: " << current_window_position_x << std::endl;
        //std::cout << "window pos y: " << current_window_position_y << std::endl;
        InitialMatrix->_x += current_window_position_x;
        InitialMatrix->_y += current_window_position_y;

        for(int i = 0; i < array.size(); i++){
            //std::cout << array[i].GetID() << std::endl;
            // moves gameObject with screen
            array[i]._x += current_window_position_x;
            array[i]._y += current_window_position_y;
            // moves collisionBox with screen
            array[i].collisionBox._x += current_window_position_x;
```

Figure 18: More Debugging examples within application for verifying proper logic.

Results

Our Aim for this Project was to establish a bridge for new game developers for creating enjoyable games on the level of Game Engines like Unity while maintaining a simple and understandable interface which anyone who wants to learn or develop can pick up quickly.

With this in mind we achieved the result we were hoping for, we were able to embed Lua into our game Engine which was our main goal so that developers could start with a simpler programming language and don't have to go through the process of creating a window, rendering, memory management etc. We do the heavy lifting while the game developers can simply use our functions like DeleteGameObjects() which takes a table as a parameter. For implementing this behind the scenes in C++ we needed to find the gameObjects inside the gameObjects vector and manually delete them by searching for IDs then also reconstructing the table for Lua, this would be an example of making it easier for developers to use our game engine and programming in Lua.

We also were able to create a User Interface which is simple to understand and is able to change the gameObjects coordinates in a very clever way that creates the illusion of infinite space since we needed to convert from World space to Screen Space, the selection of Textures through windows OS and allow for multiple scripts under a single gameObject with some exceptions that only one of the scripts contain a gameLoop function, among many other features. Rendering was also a difficult task to overcome but we were able to make it possible and render in the specific way we wanted since we didn't use the whole screen for rendering purposes which in turn allowed us to save performance when needed.

All in All we were able to create a 2D game engine which has the capabilities to create Games with the power of native scripting. We consider this project a good starting base for a

2D game engine which can be even further improved upon with many different features which we will talk about in the next section for Future work.

Future Work

There is a lot that can be done with this project for future work since game engines take years to be refined with multiple features, because of this we will outline the most important functionalities that would be done firstly if we were to move on with this project so we can take our game engine to the next level.

Exporting Games

Currently in our game engine you can only play and preview the games you make within our application, we would like to allow for exporting of the games themselves and that they can be recompiled without the game engine interface and be its own .exe file that anyone can play and share with others.

Animation

Our current game engine has support for Images like PNG files and JPG files. Further improvements can be made on gameObjects by allowing for more life and adding animation. This would be done by storing multiple Textures for the different Images that would be played to make it look like it's animated. For this feature to be at its best we would also make the ability to reorder the Images within the engine so if you make a mistake with uploading you can fix it easily without having to re-do the process.

Physics Support

Physics is a very complex topic that we have never done before and we would need to learn much more about physics in general to provide users with more effective and realistic movements. We do include the math library for lua so developers can create their own physics but we would like to create our own lua functions for developers to use so they themselves don’t have to know the intricacies of Physics and can experiment with the lua functionalities instead by passing in parameter values only, which allows us to maintain our goal of simplicity for this game engine.

Sound/Music

Currently we have no sound or music that can be loaded in which makes the gameplay of games very stale which is why this feature would provide more creative freedom for developers without having to worry about the games being boring. This would also provide more immersion for users who play the games of the developers that use our application for game development.

Rotation

To implement Rotation we would need to refactor our rendering logic fully since currently our geometric logic only consider square/rectangular cases and doesn't work with other shapes i.e When A rectangle is rotated diagonally the shape that we would need to be cut would be a triangle (and different rotations would need different cases of getting the shapes to cut). When implemented this would still keep our illusion with the infinite space and objects disappearing when moved off the “gamescreen” but allow for more unique possibilities with gameObjects.

Layers

Currently the order you create the gameObjects is the layers, there is no way to change the order without deleting the gameObjects so for the future we would like the feature to specify what gameObject is on which layer and also re-order gameObjects in the editor if no specific layering is made by the user.