

# Bachelor report



TITLE A framework for graphical and networked applications, an online 3D game, and tools			
CANDIDATE NUMBER 10003			
DATE 20.05.2019	SUBJECT CODE IE303612	SUBJECT Bachelor project	DOCUMENT ACCESS Open
STUDY Computer science		PAGES / ATTACHMENTS 95 / 35	BIBL. NUM. N/A
SUPERVISOR Saleh Abdel-Afou Alaliyat			

## SUMMARY:

We made a framework for graphical and networked applications in C++. The framework was developed with cross-platform support in mind, which means platform specific APIs were encapsulated. As for its features, it supports networking, audio, 2D and 3D graphics, skeletal animation, text rendering, and more.

Further, the framework was put to use in developing an online 3D multiplayer game. The game required content, and for that we developed some tools. We made a world editor, script editor, object editor, among others. The tools allow for editing the terrain's height map and textures, defining and placing objects, editing the scripts for dialogues and game events.

The core game code, such as renderers and scripts, were put in a separate library. The client, server, and tools use the common code, and were built on top of it.

---

*This assignment is an exam submission done by a student at NTNU in Ålesund.*

---

**Postal address**  
NTNU i Ålesund  
N-6025 Ålesund  
Norway

**Visitation address**  
Larsgårdsvegen 2  
**Internet**  
ntnu.no

**Phone**  
73 59 50 00  
**E-mail**  
postmottak@alesund.ntnu.no

**Bank account**  
7694 05 00636  
**Foretaksregisteret**  
NO 974 767 880

## Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1	Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	<input checked="" type="checkbox"/>
2	Jeg/vi erklærer videre at denne besvarelsen: <ul style="list-style-type: none"> <li>• ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands.</li> <li>• ikke refererer til andres arbeid uten at det er oppgitt.</li> <li>• ikke refererer til eget tidligere arbeid uten at det er oppgitt.</li> <li>• har alle referansene oppgitt i litteraturlisten.</li> <li>• ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.</li> </ul>	<input checked="" type="checkbox"/>
3	Jeg/vi er kjent med at brudd på ovennevnte er å <u>betrakte som fusk</u> og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. <a href="#">Universitets- og høgskoleloven</a> §§4-7 og 4-8 og <a href="#">Forskrift om eksamen</a> §§14 og 15.	<input checked="" type="checkbox"/>
4	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se <a href="#">Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver</a>	<input checked="" type="checkbox"/>
5	Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter <a href="#">høgskolens studieforskrift §31</a>	<input checked="" type="checkbox"/>
6	Jeg/vi har satt oss inn i regler og retningslinjer i bruk av <a href="#">kilder og referanser på biblioteket sine nettsider</a>	<input checked="" type="checkbox"/>

## Publiseringsavtale

Studiepoeng: 20

Veileider: Saleh Abdel-Afou Alaliyat

### Fullmakt til elektronisk publisering av oppgaven

Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten ([Åndsverkloven §2](#)).

Alle oppgaver som fyller kriteriene vil bli registrert og publisert i Brage HiM med forfatter(ne)s godkjenning.

Oppgaver som er unntatt offentlighet eller båndlagt vil ikke bli publisert.

Jeg/vi gir herved NTNU i Ålesund en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:

ja  nei

Er oppgaven båndlagt (konfidensiell)?

ja  nei

(Båndleggingsavtale må fylles ut)

- Hvis ja:

Kan oppgaven publiseres når båndleggingsperioden er over?

ja  nei

Er oppgaven unntatt offentlighet?

ja  nei

(inneholder taushetsbelagt informasjon. [Jfr. Offl. §13](#)/[Fvl. §13](#))

Dato: 20.05.19

## Preface

This report is written by Sebastian Søviknes Gundersen, a student at NTNU in Ålesund. It's the final assignment for the computer science study program, and grants 20 study points.

The author of this report has previous experience with games development in C++ using SDL and OpenGL. Primarily, the author has developed multiple offline 2D games during game jam events.

However, as opposed to using a third party library like SDL in this report, we will be using the operating system's APIs for windowing, network, and audio. The author has always had a desire to learn how things really work under the hood, and this project is a symptom of that. Additionally, we will learn about 3D computer graphics, and apply it to the project.

A special thanks to Saleh Abdel-Afou Alaliyat for guidance with the writing of this report.

Ålesund, 20.05.2019

*Sebastian S. Gundersen*

---

Sebastian Søviknes Gundersen

## Terminology

**A\*** A-star is an algorithm used for pathfinding

**anti-aliasing** Anti-aliasing is any method that smooth lines which would otherwise appear too sharp or jagged

**API** Application Programming Interface

**ARB** OpenGL extensions officially approved by the OpenGL Architecture Review Board

**C++** is a multi-paradigm programming language developed by Bjarne Stroustrup

**COLLADA** COLLABorative Design Activity is an XML-based 3D model file format

**constexpr** Constant expressions in C++ are expressions executed at compile time

**CRC** Cyclic redundancy check is an error detection method

**cross-platform** Cross-platform software can run on multiple platforms

**CRUD** Create / Read / Update / Delete

**FBX** Filmbox is a proprietary 3D model file format, owned by Autodesk

**flag** A flag is a particular bit in one or more bytes, which indicate a boolean value

**GL / OpenGL** is an API for hardware accelerated graphics rendering

**GLEW** OpenGL Extension Wrangler is a library which loads newer OpenGL functions

**GPU** Graphics Processing Unit

**HUD** Head-up display, contains important information for the player

**HTML** HyperText Markup Language

**IOCP** Input/Output Completion Port is an API for asynchronous I/O on Windows

**ImGui** *Dear ImGui* is a flexible and customizable platform independent C++ library for quickly implementing user interfaces

**lambda** An anonymous function

**library** Libraries implement an API, and contains reusable code, such as math functions or image loading

**member function** A function that is a member of a class

**method** Method is a synonym for member function

**MSAA** Multisample anti-aliasing

**multithreaded** Multiple pieces of code that run in parallel, each in a core on the processor

**mutex** Mutual exclusion

**NDC** Normalized device coordinates

**OBJ** Wavefront OBJ is a 3D model file format, only specifying the geometry of the model

**PAL** Platform Abstraction Layer

**PCM** Pulse Code Modulation

**PHP** PHP Hypertext Preprocessor

**PNG** Portable Network Graphics is an image file format with lossless compression

**preprocessor directive** In C and C++, a preprocessor directive is code that is executed before compilation

**RAII** Resource Acquisition Is Initialization

**RDBMS** Relational Database Management System

**SDL** Simple DirectMedia Layer is a framework for graphical applications in C and C++

**SFML** Simple and Fast Multimedia Library is a framework for graphical applications in C++

**SQL** Structured Query Language

**static** In C++ classes, the static keyword is used to mark a function or variable as part of the namespace of the class, instead of being a member of an instance

**TCP** Transmission Control Protocol is a networking protocol with a no packet loss guarantee

**TTF** TrueType Fonts

**UML** Unified Modeling Language

**Unity** is a game engine popular among independent developers by Unity Technologies

**Unreal Engine** is an industry standard open-source game engine by Epic Games

**vertex** A single point in a polygon. A triangle has three vertices.

**virtual** In C++, the virtual keyword marks a member function as polymorphic

**WASAPI** Windows Audio Sessions API for streaming audio in Windows

**WGL** Windows OpenGL is the API that connects OpenGL with the window system on Windows

**WinAPI** The Windows API provides everything needed to make Windows programs

**WSA / WinSock** Windows Sockets API for network communications in Windows

**XML** eXtensible Markup Language

# Contents

<b>Cover page</b>	<b>i</b>
<b>Preface</b>	<b>iv</b>
<b>Terminology</b>	<b>v</b>
<b>Contents</b>	<b>viii</b>
<b>List of Figures</b>	<b>xiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Objective . . . . .	1
1.3 Scope . . . . .	2
1.4 Structure . . . . .	3
1.4.1 Overview of framework modules . . . . .	3
1.4.2 Overview of game systems . . . . .	4
1.4.3 Overview of client . . . . .	4
1.4.4 Overview of server . . . . .	5
1.5 Outline . . . . .	5
<b>2 Theory</b>	<b>6</b>
2.1 C++ . . . . .	6
2.1.1 RAII . . . . .	6
2.1.2 Pre-processor directives . . . . .	7



2.1.3	Bitwise operations in C++	7
2.2	Platform abstraction	8
2.2.1	Platform abstraction layer	8
2.2.2	Run-time abstractions	8
2.2.3	Compile-time abstractions	8
2.2.4	Game loop	9
2.2.5	Fixed time steps	9
2.2.6	Variable time steps	9
2.2.7	Immediate mode user interfaces	10
2.2.8	Retained mode user interfaces	10
2.2.9	Multithreading	11
2.3	Network	13
2.3.1	TCP	13
2.3.2	Network sockets	13
2.3.3	Multi-threaded servers	14
2.3.4	Blocking sockets	14
2.3.5	Non-blocking sockets	14
2.3.6	Overlapped I/O sockets on Windows	14
2.4	Audio	15
2.4.1	Pulse code modulation	15
2.4.2	Buffering	15
2.5	Graphics	16
2.5.1	OpenGL	16
2.5.2	Immediate mode rendering	16
2.5.3	Vertex buffers	17
2.5.4	Vertex arrays	17
2.5.5	Index buffers	17
2.5.6	Shader program	18
2.5.7	Normals	19
2.5.8	Transform matrices	20

2.5.9	Texture interpolation	22
2.5.10	Mipmapping	22
2.5.11	Multisample anti-aliasing	23
2.5.12	FreeType	23
2.5.13	Unicode	23
2.5.14	Skeletal animation	24
2.6	Height maps	25
2.7	Picking	25
2.8	Relational databases	26
2.8.1	SQL	26
2.9	Web development	26
2.9.1	PHP	26
<b>3</b>	<b>Method</b>	<b>27</b>
3.1	Tools and libraries	27
3.1.1	Reasoning behind using C++	27
3.1.2	Using CMake to generate project files	27
3.1.3	Third party libraries	28
3.2	Miscellaneous framework features	28
3.2.1	Event listeners and queues	28
3.2.2	Debugging and metrics	29
3.3	Platform abstraction layer	30
3.3.1	Designing the interface	30
3.3.2	Abstracting the entry point of a program	30
3.3.3	Managing windows	31
3.3.4	Dealing with user input	32
3.3.5	Render context	32
3.3.6	Network sockets with WinSock	33
3.3.7	Synchronization of socket events	33
3.4	Graphics	34

3.4.1	Generic vertex layouts	34
3.4.2	Importing and exporting 3D models	35
3.4.3	Implementing skeletal animation	36
3.4.4	Optimizing skeletal animations	36
3.4.5	Separation of logic and rendering	36
3.4.6	Surfaces	36
3.4.7	Rendering text	37
3.4.8	Tile borders	37
3.5	Game features, and interacting with the world	37
3.5.1	Picking	37
3.5.2	Pathfinding	37
3.5.3	Fishing	38
3.5.4	Trading	39
3.5.5	Quest system	39
3.6	Node based scripting system	39
3.6.1	Script context	40
3.6.2	Game variables	40
3.6.3	Dialogues	40
3.6.4	Scripted object events	40
3.6.5	Item behavior scripts	40
3.7	Packet definitions	41
3.7.1	Defining new packets	41
3.7.2	Defining some macros	42
3.8	Editing the world's height map	43
3.8.1	Texture mapping the height map vertices	43
3.9	Client states	44
3.9.1	Auto-update	44
3.9.2	Lobby	45
3.10	Server behavior	46
3.10.1	Tracking player sessions	46

3.10.2 Validating player actions . . . . .	46
3.10.3 Synchronizing with database . . . . .	46
3.11 Database . . . . .	47
3.11.1 Structure . . . . .	47
3.11.2 Stored procedures . . . . .	48
3.11.3 Upserts . . . . .	48
3.12 Website . . . . .	49
3.13 Building the projects . . . . .	49
<b>4 Result</b>	<b>50</b>
4.1 Framework . . . . .	50
4.2 Game client . . . . .	51
4.2.1 User interfaces . . . . .	52
4.2.2 Head-up display . . . . .	59
4.2.3 Minimap . . . . .	59
4.2.4 Combat . . . . .	60
4.3 Tools developed to import assets and design the game . . . . .	61
4.3.1 World editor . . . . .	61
4.3.2 Editor for node based scripting . . . . .	62
4.3.3 Quest editor . . . . .	64
4.3.4 Model manager . . . . .	65
4.3.5 Object editor . . . . .	67
4.4 Performance profile . . . . .	68
<b>5 Discussion</b>	<b>69</b>
5.1 Completeness compared to the requirements specification . . . . .	69
5.2 Platform abstractions . . . . .	70
5.3 Models, skeletal animations, and attachments . . . . .	70
5.4 Technical game result . . . . .	70
5.5 Tools that were developed . . . . .	71
5.6 Move scripting functionality to the framework . . . . .	71

5.7 Proper asset management . . . . .	71
5.8 More graphical features . . . . .	71
<b>6 Conclusion</b>	<b>72</b>
<b>References</b>	<b>73</b>
<b>Appendices</b>	<b>80</b>
<b>A Preliminary report</b>	<b>81</b>
<b>B Progress reports</b>	<b>102</b>
<b>C Network API</b>	<b>109</b>
<b>D Graphics API</b>	<b>110</b>
<b>E Audio API</b>	<b>112</b>
<b>F Platform API</b>	<b>113</b>
<b>G Window API</b>	<b>114</b>

# List of Figures

1.1	Diagram showing how the framework modules are connected. . . . .	3
1.2	Diagram showing the game systems, and how they depend on the framework. . . .	4
1.3	Diagram showing how the client depends on the game systems and framework. . .	4
1.4	Diagram showing how the server depends on the game systems and framework. . .	5
2.1	Diagram of an 8-bit number being bit shifted 3 bits to the left. . . . .	7
2.2	Diagram demonstrating AND, OR, NOT, and XOR bitwise operators. . . . .	7
2.3	Illustration of a data race occurring. . . . .	11
2.4	Illustration of two atomic operations occurring in two separate threads. . . . .	12
2.5	4-bit PCM stream . . . . .	15
2.6	Example of fixed function pipeline in OpenGL. . . . .	16
2.7	Example of 3 vertices with XYZ position and RGB color stored in a vertex buffer . .	17
2.8	Illustration showing the shader pipeline from the vertex to fragment shader. . . .	18
2.9	Face normals . . . . .	19
2.10	Vertex normal . . . . .	19
2.11	Illustration comparing perspective and orthographic projections . . . . .	21
2.12	Comparison of nearest neighbour vs bilinear interpolated texture scaling. . . . .	22
2.13	Comparison of with and without mipmapping . . . . .	22
2.14	Illustration showing how a pixel with 4 samples is anti-aliased . . . . .	23
2.15	Image showing the metrics of a glyph . . . . .	23
2.16	Image showing the bones of the character's skeleton. . . . .	24
2.17	Screenshot of the world editor, showing the terrain height map with the grid. . . .	25

3.1	Some macros for logging. Note that they are simplified for this figure. . . . .	29
3.2	Example debug log output. . . . .	29
3.3	An example of what a custom vertex structure may look like. . . . .	34
3.4	Flowchart diagram showing the fishing process. . . . .	38
3.5	Flowchart diagram showing the trading process. . . . .	39
3.6	An example of a complete packet, defined in <code>to_client::game</code> . . . . .	42
3.7	Example of how a server may handle incoming packets with the help of a macro. . . . .	42
3.8	This function shows the bit shifting that is done to pack the corners into a key. . . . .	43
3.9	Flowchart diagram of the auto-update process. . . . .	44
3.10	Flowchart diagram for the authentication process. . . . .	45
3.11	Diagram showing the structure of the database schema. . . . .	47
3.12	The body of the stored procedure for updating a slot in an item container. . . . .	48
4.1	Screenshot of the game client, including interfaces. . . . .	51
4.2	Screenshot of the game client, without interfaces showing. . . . .	51
4.3	The user interface for the inventory tab. . . . .	52
4.4	The user interface for the equipments tab. . . . .	53
4.5	The user interface for the quests tab. . . . .	54
4.6	The user interface for the stats tab. . . . .	55
4.7	The trading interface, while the trade is still ongoing. . . . .	56
4.8	The trading interface when the other player has accepted. Accept button is hovered. . . . .	56
4.9	Going through the options in a dialogue. . . . .	57
4.10	Continuing with the dialogue after coming back with some salmon. . . . .	57
4.11	Script for a dialogue where a traveler wants to buy salmon from the player. . . . .	58
4.12	Screenshot of the minimap by a snowy mountain and a river. . . . .	59
4.13	Player with a spear fights a player with a sword and shield. . . . .	60
4.14	Spearman deals 13 damage to a swordsman. . . . .	60
4.15	Screenshot of the world editor. . . . .	61
4.16	Screenshot of the script editor. . . . .	62
4.17	Context menu for a node in the script editor. . . . .	62

4.18 Context menu for adding a new node in the script editor. . . . .	63
4.19 Context menu for the <i>give or take item</i> node in the script editor. . . . .	63
4.20 Screenshot of the quest editor. . . . .	64
4.21 Screenshot of the model conversion tool. . . . .	65
4.22 Screenshot of the model attachments editor. . . . .	66
4.23 Screenshot of the object editor. . . . .	67
4.24 Performance profiling results. . . . .	68



# Chapter 1

## Introduction

### 1.1 Background

Development of graphical applications today is mostly done by using existing frameworks and engines, such as Unreal Engine [21], Unity [52], SDL [37], and SFML [22]. SDL and SFML mainly focuses on providing a unified platform API, but they do include other features such as loading and using images, audio, and fonts. Unity and Unreal Engine on the other hand, contain features you didn't even know you would want. It's almost always beneficial to use any of these tools, over creating your own from scratch. They allow the programmer to not worry about learning the platform they are developing for, which is an enormous cost for a small company to take on.

### 1.2 Objective

Although the existing tools are great, it's often useful to learn how they work under the hood. This project attempts to show what is possible without them, and how you can make your own. We will also be creating an online 3D game, along with the tools necessary to produce content for it. The primary objectives are:

- Create portable network, audio, window, and graphics modules
- Support fonts, skeletal animation, custom shaders, and loading images
- Developing an online 3D multiplayer game alongside the framework, as well as a world editor, and other tools for making game content

## 1.3 Scope

The framework is put to use in developing an online 3D multiplayer game, with a set of tools to produce content. Finally, a simple website with user registration and leaderboards, to demonstrate the shared access of the database. The code is mostly C++, accompanied by some SQL for the database, and PHP for the website.

The framework is developed with cross-platform support in mind, which means an encapsulation of platform specific APIs. As for its features, it supports networking, audio, 2D and 3D graphics, skeletal animation, text rendering, and more. Multithreading is also utilized in some of these areas, to make as good use of the cores on the processor as possible.

The features are modularized, so they can be enabled or disabled depending on the use case. For instance, a server requires neither the capability of playing audio nor displaying graphics.

The game itself will be developed with minimal platform awareness, and will only be using what is given to it by the framework. A few interesting features for the game include the node based scripting system, autotiled terrain, and pathfinding with A\* algorithm. It's also worth noting how the coordination between the client and server is done. In short, the server may never trust the client.

The tools developed make it possible to quickly shape 3D terrain, and place objects on it. The objects can be linked to scripts, such as a dialogue script when it's interacted with, or other events. A node based editor must be used to create or modify the scripts. After all these dependent layers are put together, the final result is ready.

## 1.4 Structure

### 1.4.1 Overview of framework modules

The diagram in figure 1.1 shows how the framework is modularized. In orange, we see the public APIs that are available for users of the framework. The blue boxes represent implementations of the platform APIs. To support a new platform, all the blue boxes need to implement the respective platform APIs. The green box, labeled OpenGL, implements the renderer. A renderer is not necessarily tied to any platform, but we need a render context to draw on a window. Therefore, any platform that wishes to use OpenGL, must implement the link between the window API and OpenGL.

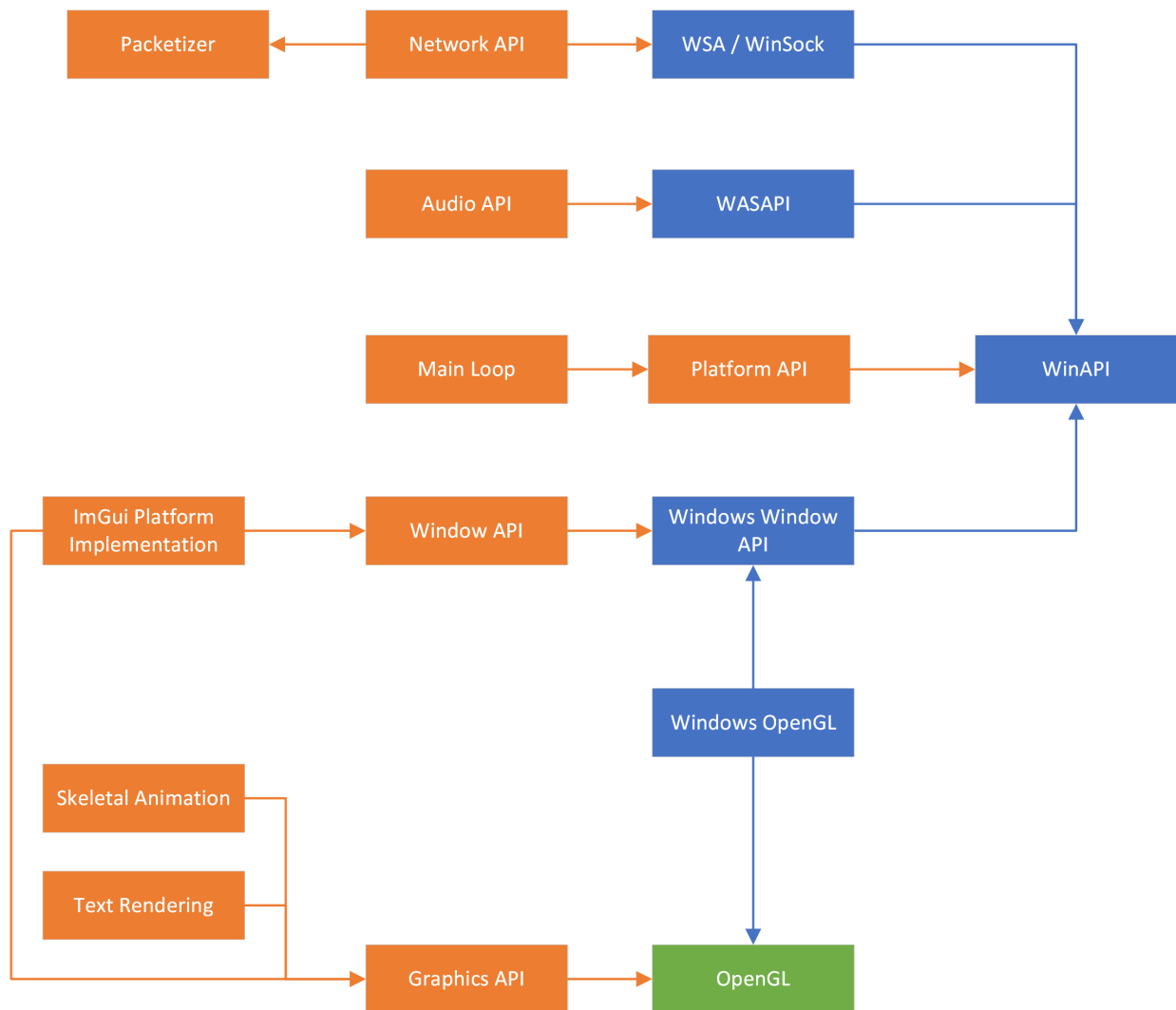


Figure 1.1: Diagram showing how the framework modules are connected.

### 1.4.2 Overview of game systems

The diagram in figure 1.2 shows how the game systems connect to the framework modules.

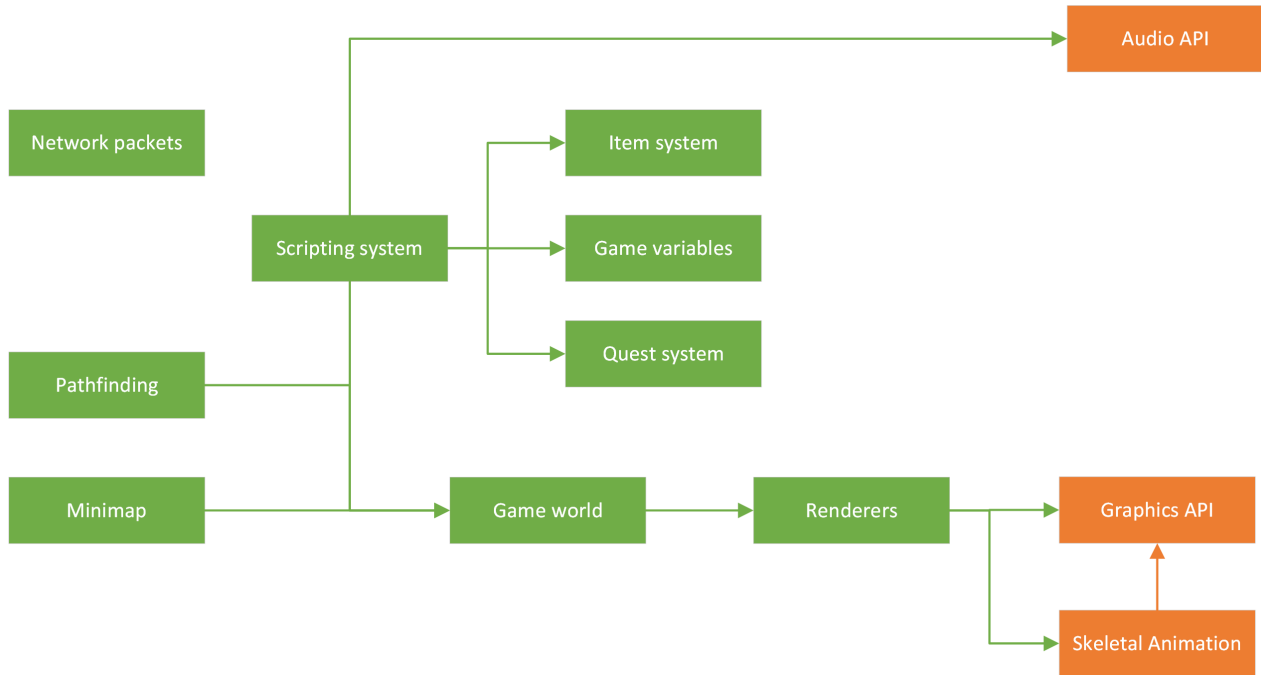


Figure 1.2: Diagram showing the game systems, and how they depend on the framework.

### 1.4.3 Overview of client

The diagram in figure 1.3 shows how the client uses the game systems and framework.

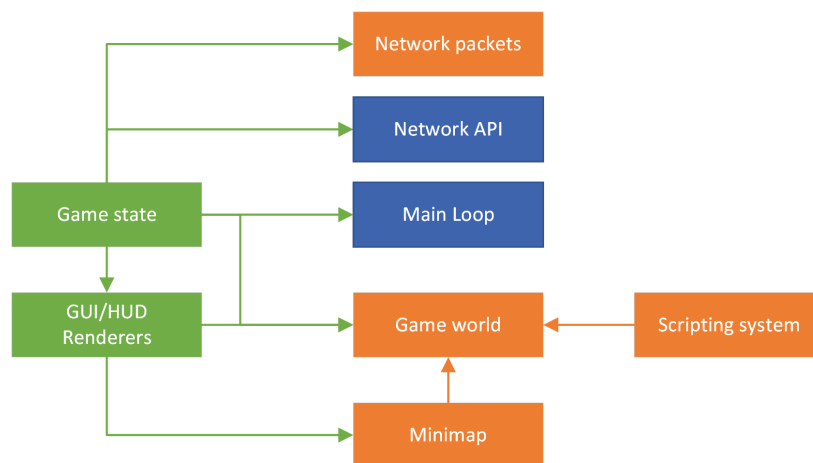


Figure 1.3: Diagram showing how the client depends on the game systems and framework.

### 1.4.4 Overview of server

The diagram in figure 1.4 shows how the server uses the game systems and framework.

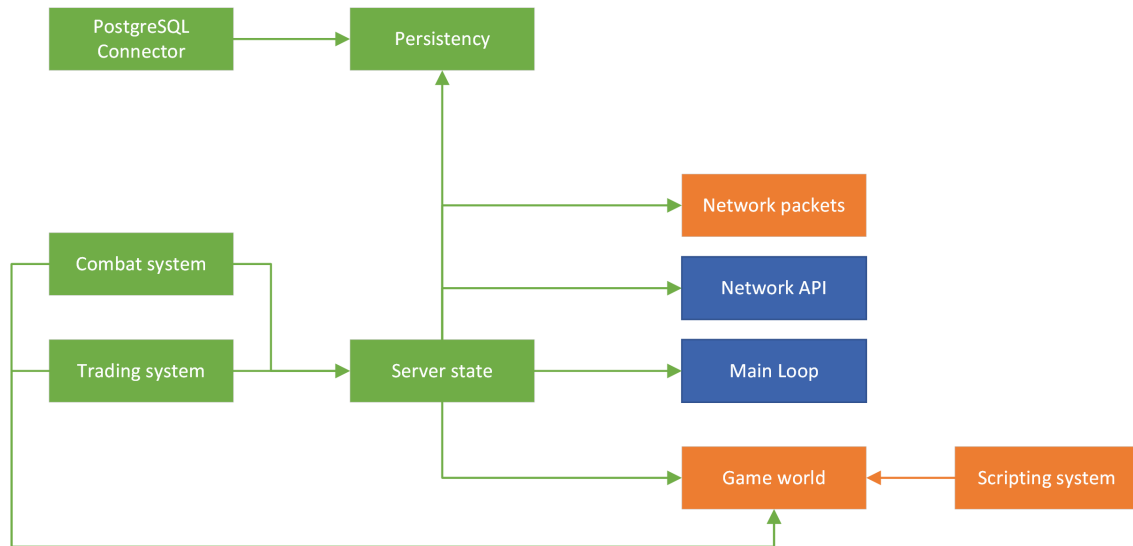


Figure 1.4: Diagram showing how the server depends on the game systems and framework.

## 1.5 Outline

1. **Introduction:** Presents the motivation behind the project, and goals for the end result. Shows the structure of the project in diagrams, giving an overview of how key parts connect.
2. **Theory:** Relevant areas within audio, network, graphics, and other topics, are explained. Illustrations are embedded to give a complementing visual view of the described theory. This information prepares the reader for the next chapter.
3. **Method:** The strategy and implementation of the major parts of the project is explained. Which methods were used for solving the problems, and how they worked out.
4. **Results:** The final results are described, and shown with screenshots.
5. **Discussion:** Completeness and quality of the objectives are discussed. We look at how the framework, game, and tools ended up in different aspects, and how they can be further improved.
6. **Conclusion:** Recounting the objectives, and summarizing the progress that was made.

# Chapter 2

## Theory

### 2.1 C++

C++ is a multi-paradigm programming language, initially developed in 1979 by Danish computer scientist Bjarne Stroustrup. It started out as a project to add classes to C, referred to as *C with Classes*. However, it was renamed to C++ in 1983 [51]. Both the language and the C++ community has recently been revitalized due to its new frequent release schedule. It has a long history of being used in most high performing games and 3D engines. Some examples being Unreal Engine [15], Unity [1], CRYENGINE [13], and OGRE [42].

#### 2.1.1 RAII

Resource Acquisition Is Initialization (RAII) is a key feature that distinguishes C++ from most other programming languages. With class constructors, you initialize an object as it is created, similar to other languages. However, it is also possible to create a destructor, which lets you clean up resources upon destruction. [11] This does create some pitfalls to watch out for, such as forgetting to declare a base class destructor as virtual, when it makes sense for derived classes to override it.

## 2.1.2 Pre-processor directives

Before the compiler actually compiles a source file, it scans for pre-processor directives. They all begin with a #, and are processed prior to compilation. The original source code is transformed by file inclusions, macros, and conditionals, and it's the result that is compiled. They can also be used to throw errors, for example as a result of a pre-processor conditional, or specify additional options to the compiler [9][10].

## 2.1.3 Bitwise operations in C++

Bit shifting allows us to shift all the bits of a value, either to the left or right. To shift in a direction, we need to specify the number of bits to shift. In figure 2.1, we see an 8-bit value being shifted 3 bits to the left, using the << operator. To shift to the right instead, the >> operator can be used.

We can also use other bitwise operators as shown in figure 2.2.

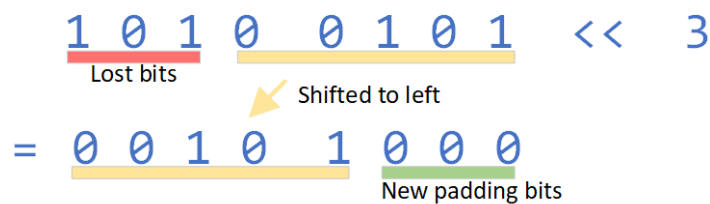


Figure 2.1: Diagram of an 8-bit number being bit shifted 3 bits to the left.

<p><b>AND</b></p> <pre> 1 0 1 0 0 1 1 1 &amp; 0 1 1 0 1 1 0 0 = 0 0 1 0 0 1 0 0 </pre>	<p><b>NOT</b></p> <pre> ~ 1 0 1 0 0 1 1 1 = 0 1 0 1 1 0 0 0 </pre>
<p><b>OR</b></p> <pre> 1 0 1 0 0 1 1 1   0 1 1 0 1 1 0 0 = 1 1 1 0 1 1 1 1 </pre>	<p><b>XOR</b></p> <pre> 1 0 1 0 0 1 1 1 ^ 0 1 1 0 1 1 0 0 = 1 1 0 0 1 0 1 1 </pre>

Figure 2.2: Diagram demonstrating AND, OR, NOT, and XOR bitwise operators.

## 2.2 Platform abstraction

### 2.2.1 Platform abstraction layer

All operating systems must provide their own APIs to be used for developing software. These interfaces allow for creating windows, communicate with devices such as network adapters and GPUs, manage files, and more. Each of them has their own data structures and functions. This requires a portable application to implement support for the ones it wants to be present on. [71]

Having an application use conditional pre-processor directives all over the code base would get messy quickly. Instead, it is more common to create one unified interface, known as a platform abstraction layer (PAL). This interface has absolutely no ties to the platform it abstracts, and exposes only platform agnostic code. If the abstraction is reasonable, it can be implemented on any platform. It's useful to only select features that are important, as it greatly simplifies the abstraction required.

### 2.2.2 Run-time abstractions

Some applications will allow the user to switch between OpenGL, Vulkan, or DirectX while they are running. Instead of selecting one of them during compilation, they are enabled or disabled in run-time code. This is possible because all those APIs are available on the operating system that the application is compiled for.

### 2.2.3 Compile-time abstractions

When abstracting operating system APIs, there is no way to access the Windows API on Linux. In other words, you can't run an application on Linux, and change it to running on Windows during run-time. Instead, we must use pre-processor directives, to conditionally compile what should be included or not.



### 2.2.4 Game loop

The game loop is responsible for continuously updating game logic, and rendering new frames to the window. [49] This can be done in a simplistic manner, without any consideration of frame times. The problem is that every computer performs differently. Newer devices may run twice as fast as the developers' test computers. This was a common problem in the early days of computer games, and sometimes even today. [65]

Solving this issue can be either easy or hard, depending on which method is chosen. For all methods, it's important to keep track of the frame time, and schedule update and draw calls appropriately. Frame time is the measured delta time between the beginning and end of an iteration in the loop. Both methods described below are equally valid, depending on the scenario. There are many more flavours to these, and there will never be one correct answer.

### 2.2.5 Fixed time steps

Also known as constant time steps, it's an easy method for scheduling the update and draw calls. Essentially, each update will assume the frame time remains the same for every call. [19][72] This takes away the responsibility of handling delta times, which can quickly end with critical bugs.

The obvious drawback is being limited to a fixed update rate for all users of the software. Users with high-end hardware might be restricted to a lower frame rate than their system can handle. Although a 120 Hz monitor can still be supplied with 120 frames per second without a problem, the logic updates will be limited to common rates such as 60 frames per second.

The opposite goes for low-end hardware. Their system will try to update 60 times per second, but the rendering fails to catch up. This results in choppy visuals, because for every draw call, two updates may have occurred. The system could potentially have had a smooth experience, if variable frame rates were used.

### 2.2.6 Variable time steps

This method does not care for scheduling update and draw calls. Instead, it performs as fast as it can, at all times. The role of synchronization is delegated to the application programmer. They must use the delta time as a factor when calculating movement of objects, animations, and so on. This makes for a very smooth experience for the end user. However, it's hard to implement correctly, and it may also demand a lot more power from the system.

### 2.2.7 Immediate mode user interfaces

Some user interface APIs don't need elements to be created beforehand. The state of an element is available directly after being updated. This is known as an immediate mode graphical user interface. [5] There is no state that must be managed by the application, as everything is handled internally in the library. Displaying an input box taking an integer with *Dear ImGui*, can be done as follows: `bool changed = ImGui::InputInt("Enter a value", &value);` [8]

Additionally, extra options can be passed. The above function accepts stepping speeds, and flags to specify the behavior of the input box. To create scopes, it is possible to push and pop identifiers, and have them wrap inner elements. The simple nature of an API like this makes it great for developing highly usable and extendable UIs very quickly.

### 2.2.8 Retained mode user interfaces

Retained mode is the opposite of immediate mode. This is where elements must be created, configured, stylized, and have event listeners registered, before using them. A lot more work must be done per element. Another detail is that due to the object-oriented nature of retained mode UIs, there is a larger focus on encapsulation.

## 2.2.9 Multithreading

### When and why to use multiple threads

Multitasking operating systems provide multithreading APIs, which allow for utilizing the cores on a processor as effectively as possible. A higher number of processing cores per processor is becoming more common, as singlethreaded processing speed has seen diminishing improvements over the past decade. Since only one thread can run at any one time in a core, it doesn't make sense for an application to create more threads than the number of cores on the processor. [3] It can even lead to worse performance, because of the overhead of context switches. [6]

### Data races

If two threads access the same resource simultaneously, race conditions may occur. [41] An example of a data race is seen in figure 2.3. Both threads A and B read the counter value of 0. Thread A increments it, and updates the memory. The counter is now 1. However, thread B still has the old value of 0 in memory. It increments the value, and updates the memory. The value is expected to be 2 now, but due to the race condition, it is only 1.

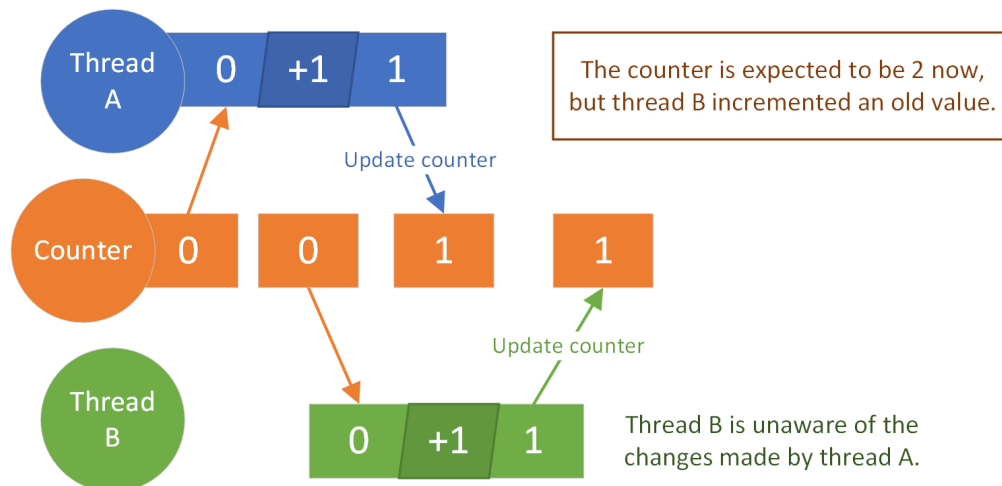


Figure 2.3: Illustration of a data race occurring.

### Mutual exclusion

To solve the issue of data races, we can use either a mutex, or an atomic variable. A mutex is an indicator, which may either be locked or unlocked. [34] Both thread A and B will attempt to lock the mutex first, and one of them will succeed. The winning thread now enters the critical section, which in this example consists of incrementing the counter. The losing thread must wait until the other thread has left the critical section, and subsequently unlocked the mutex. This behavior sometimes leads to a common pitfall when using mutexes. If the same thread ends up locking the mutex a second time, it will halt indefinitely. The mutex will now be forever locked, and the application may freeze as a result. This is known as a deadlock, and is important to look out for.

### Atomic variables

While mutexes are great for synchronization of bigger critical sections, they are often inferior to atomic variables when dealing with atomic operations. The example given in figure 2.3 shows each critical section only incrementing a counter. An atomic variable will load its value before having a comparison or assignment done to it. [68] While the atomic operation is ongoing, other processing cores may attempt to load the value, but the operation still requires mutual exclusion. The core must wait until the atomic operation is complete. If more complexity was involved, a mutex would likely have been the correct choice. In figure 2.4, we see how thread B cannot access the counter while thread A is performing an atomic operation.

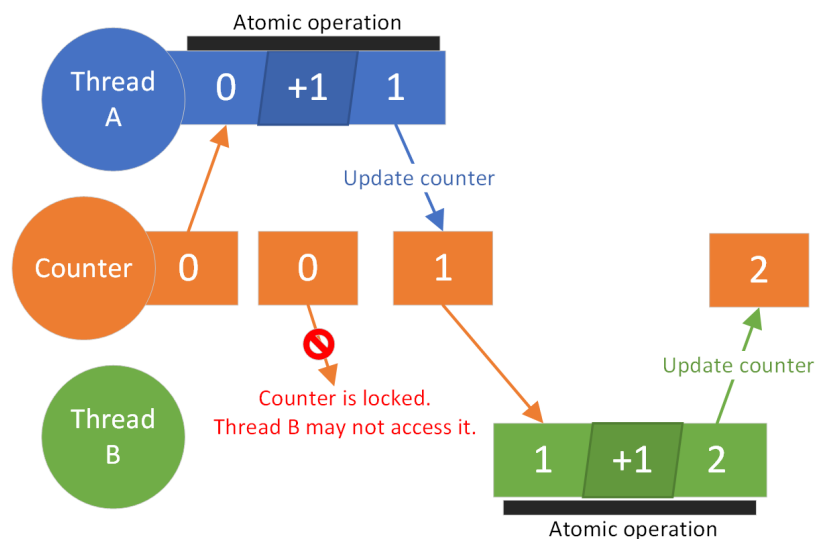


Figure 2.4: Illustration of two atomic operations occurring in two separate threads.

## 2.3 Network

### 2.3.1 TCP

TCP is a connection-oriented network protocol, first specified in 1974. [67] Making a TCP connection starts by two endpoints greeting each other with a handshake. Once both sockets are ready, they can send and receive data between each other. Both sockets perform error detection by checksums, but CRC is also often used in lower layers. If the packet contains errors, the recipient asks the sender to repeat the message. If a packet is lost on the way, it will be resent.

TCP also ensures that each packet is received in the same order as they were sent.

### 2.3.2 Network sockets

Network sockets are connection endpoints for sending and receiving data in a network. This is usually known as the Sockets API. Modern operating systems provide this API out of the box. Most socket interfaces are based on the Berkeley Sockets API, which later became part of POSIX as the POSIX Sockets API, with a few minor changes. [25][70] Berkeley sockets were designed to be used as file descriptors, to abstract interfaces for different data streams into one.

Most programming languages provide its own sockets API that wraps the operating system's sockets API. Java is one of those languages, and takes care of all the platform specific code under the hood. You simply need to use their abstracted interfaces, such as the `Socket` class. [43] While in C++, there is no standard sockets interface, so you must do all the platform checking yourself. However, there are plans to standardize network sockets in a future C++ version. [35][12]

Every language uses the operating system's interface for sockets. You can write two clients in Java and C++, a server in PHP, and the three programs will work together as expected. The same way these programs must use the sockets API, the operating system must use an API for the network adapter. Which is why those applications may also run on multiple operating systems, and still work as expected.

### 2.3.3 Multi-threaded servers

Multithreading is useful for writing servers, because we can handle multiple ongoing I/O operations simultaneously on multiple client sockets. There are many ways of going about making a multi-threaded server. [14] We will go through some below.

### 2.3.4 Blocking sockets

The simplest approach is to use blocking sockets. In this scenario, you would want up to two threads per socket. One for receiving, and one for sending data. It is also possible to have one thread per socket for receiving data, and one thread for sending data to all sockets. While this is the easiest way of making a multi-threaded server, it is also one of the least efficient ones, as it is not scalable. A server with hundreds of connections would use hundreds of threads, and that will make context switches a big problem. A processing core can only execute one thread at a time.

### 2.3.5 Non-blocking sockets

Non-blocking sockets are more flexible, as they are based on being polled rather than blocking the thread. Many methods exist for polling non-blocking sockets, and most are platform specific. Some are scalable, and some become slower depending on the number of connected sockets.

Some platforms allow for data to be received as the socket is accepted, such as in WinSock 2 with the `AcceptEx` extension. [40] This can be useful when dealing with thousands of connections on a single server, especially if they are short-lived connections.

### 2.3.6 Overlapped I/O sockets on Windows

While most operating systems strictly implement POSIX sockets, Windows has its own variation named WinSock (Windows Sockets), which deviates slightly from POSIX. [30] Since WinSock 2, it is possible to use overlapped I/O for socket operations. This is a very efficient method of asynchronous I/O on Windows. [69][31] These sockets have the benefits of both blocking and non-blocking sockets. When a send call is issued, it is guaranteed that the data will be transferred completely and asynchronously. This is assuming the network is in a working condition. When waiting for incoming data, a receive call is done to notify that we are ready to handle more data. Once more data is available, the worker thread pops the receive event off the stack, and processes it. After we are done, a new receive call is issued.

## 2.4 Audio

### 2.4.1 Pulse code modulation

Pulse code modulation (PCM) is a method of encoding uncompressed audio. PCM streams have a fixed sample rate per second and number of bits per sample. [50] Figure 2.5 shows a PCM stream with 4 bits per sample. We can see that the red line shows how the stream should be, in its most accurate state. The blue circles represent recorded samples, and an error margin is visible. With more bits per sample, they would be more precise, and as a result sound better.

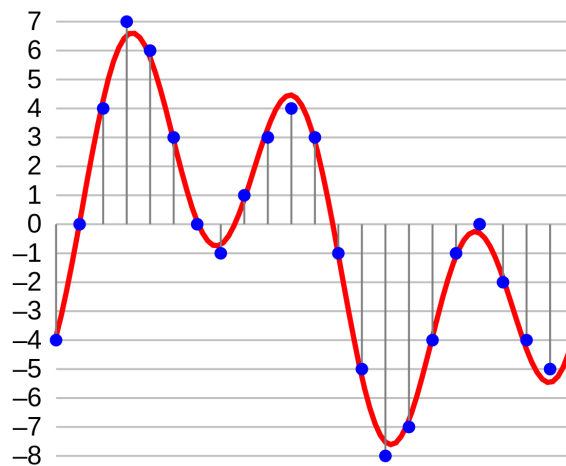


Figure 2.5: 4-bit PCM stream. Source: [upload.wikimedia.org/wikipedia/commons/thumb/2/21/4-bit-linear-PCM.svg/1280px-4-bit-linear-PCM.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/2/21/4-bit-linear-PCM.svg/1280px-4-bit-linear-PCM.svg.png)

### 2.4.2 Buffering

To guarantee continuous playback of audio, it must be streamed to the audio device at frequent intervals. If the audio buffer does not get filled up fast enough, there will be a noticeable delay in playback. To mitigate this, the audio device always needs a buffer of samples to consume. At least a few hundred milliseconds of samples should be available in the buffer, at any time. [32]

The main thread of a program might be unpredictable. If the frame rate slows down even for a little bit, the audio buffer might not be refilled in time. There can be blocking code, such as the open file dialog in Windows. A dedicated streaming thread is not always needed, but it's typically not a bad idea. Since the thread is not constantly writing to the buffer, it can generally sleep according to the frame rate. Another thing to consider is the usage of mutexes. If used, they must be controlled tightly to avoid long locks, leading to the buffer running empty. [38]

## 2.5 Graphics

### 2.5.1 OpenGL

OpenGL is a cross-platform graphics API used in many games and applications. It provides an interface to manage textures, buffers, shaders, and more. A lot of functionality has been deprecated in recent versions, in favor of flexibility and customisability. Older versions primarily used the fixed function pipeline, with immediate mode rendering. [53] The legacy features were far easier to use, and made it easy for anyone to quickly learn how to render complex visuals. While barrier to entry was low, it was difficult to push the performance of OpenGL beyond a certain point. This led to new extensions such as vertex and fragment shaders, and vertex arrays. [26]

### 2.5.2 Immediate mode rendering

Sending vertices to the GPU used to happen on-the-fly. You would chain a series of function calls, as seen in figure 2.6. It gives quick access to drawing vertices on the screen, but comes at the cost of having to send the vertices each time. Not only that, but quite a lot of CPU time is spent calling these functions, and that becomes a lot of overhead in total. This is rarely seen today, but it's worth keeping it in mind to understand why vertex buffers exist. [60]

---

```
glBegin(GL_TRIANGLES);  
glColor4f(1.0f, 0.0f, 1.0f, 1.0f);  
glVertex3f(-1.0f, 0.0f, -1.0f);  
glVertex3f(1.0f, 0.0f, -1.0f);  
glRotatef(135.0f, 0.0f, 1.0f, 0.0f);  
glVertex3f(0.0f, 0.0f, 1.0f);  
glEnd();
```

---

Figure 2.6: Example of fixed function pipeline in OpenGL.



### 2.5.3 Vertex buffers

A vertex buffer may contain either all the attributes of a vertex, or have them separated over multiple buffers. It can contain either a structure of arrays, or an array of structures, depending on the platform and use case. [55] Each attribute should also be aligned on a 4 byte boundary, as padding will still occur either way. If an attribute consists of 6 bytes, the last 2 bytes should be attempted to be made use of. [54]

### 2.5.4 Vertex arrays

Now that the vertices are in vertex buffers, we need to specify how the vertex attributes are stored. A vertex array object, also known as an input layout, maps certain offsets and sizes in the buffer to a particular vertex attribute. [59] Imagine a vertex buffer with the layout in figure 2.7. Both attributes have 3 floats, each with a size of 4 bytes. When specifying the attributes, we see that position starts at offset 0, and the next position is 24 bytes ahead. This means its stride is 24. The same goes for the color attribute. We see it starts at offset 12, and has the same stride as position.

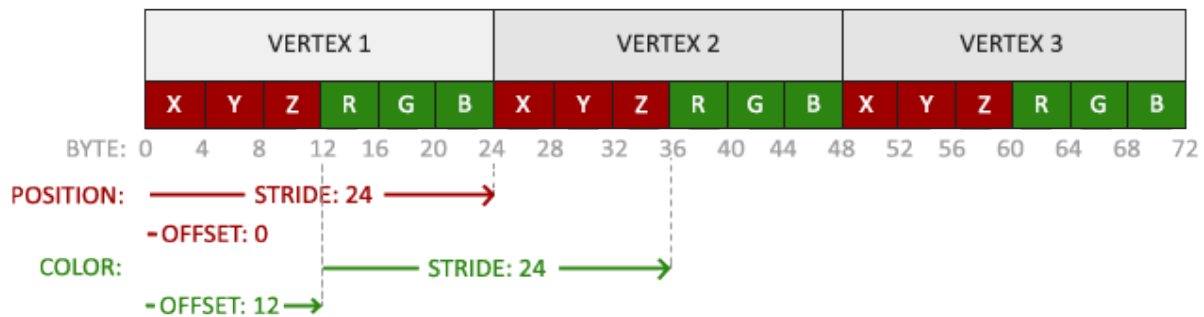


Figure 2.7: Example of 3 vertices with XYZ position and RGB color stored in a vertex buffer. Source: [learnopengl.com/Getting-started/Shader](http://learnopengl.com/Getting-started/Shader)s

### 2.5.5 Index buffers

A mesh has a lot of common vertices. It is almost always worthwhile to have an index buffer, also known as an element buffer, to reuse vertices. When drawing a quad or plane, we need two triangles, which means 6 vertices are required. Two of these are duplicates, and that could easily be a waste of 128 bytes, assuming they have a few other attributes such as colors and normals. If instead we introduce an index buffer, it will cost only 12-24 bytes to represent 6 vertices.

## 2.5.6 Shader program

A shader program is responsible for processing vertices passed to it, and calculating pixel values as output. They consist of multiple types of shaders, such as the vertex shader, and the fragment shader. They may contain variables that are modifiable by the application, as well as variables that are passed from one shader to another. Such as a vertex shader taking a texture coordinate, and passing that on to the fragment shader. A simplified pipeline can be seen in figure 2.8 below.

### Vertex shader

The vertex shader processes each vertex to be rendered. The output is the screen space position of a vertex, which is calculated by multiplying the world space position by the model view projection matrix. Other attributes such as color, texture coordinates, normals, and other vertex specific data are also passed to the vertex shader. These attributes may be mutated by the shader, such as converting the normal from world space to screen space. [58]

### Fragment shader

The fragment shader, also known as the pixel shader, calculates the color for each pixel that fills the vertices processed by the vertex shader. This sounds like a lot of work, but this is why GPUs are heavily parallelized. [56]

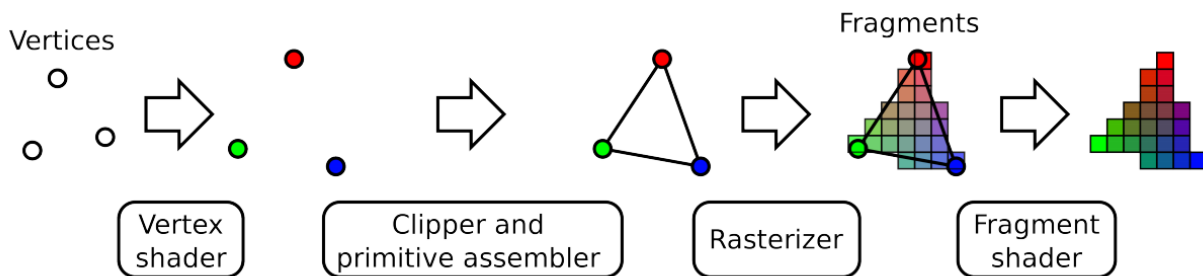


Figure 2.8: Illustration showing the shader pipeline from the vertex to fragment shader.

## 2.5.7 Normals

It's necessary to have normal vectors for the geometry that is rendered, to calculate the diffuse lighting. The normals used affect how the shading is calculated. In figure 2.9, we see a triangle have one normal vector for its entire face. This means the light will shine on it as if it is flat. This is often what you want for boxes, and other sharp edges. [16]

Face normals are great for flat shading, but usually smooth shading is preferred for most geometry. [33] In figure 2.10, we see that instead of the triangle face having one normal, each of its vertices have their own normal. The normal of these vertices are calculated by using their neighbouring vertices. This creates a smooth shading, as the light will seem to shine on the respective vertices instead of the face as a whole.

There is still room for improvement, and that is done by normal mapping. We will not use normal mapping in this report, but the concept will be briefly explained. You can think of this as a fragment normal. Imagine figure 2.10, but with each fragment having its own normal vector. This will make the face be shaded much more accurately, with bumps and edges. [28]

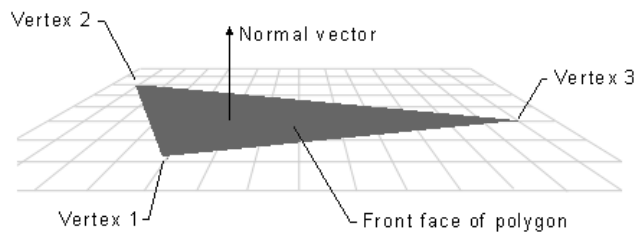


Figure 2.9: Illustration shows the face's normal vector. Source: [docs.microsoft.com/en-us/windows/uwp/graphics-concepts/images/nrmlvect.png](https://docs.microsoft.com/en-us/windows/uwp/graphics-concepts/images/nrmlvect.png)

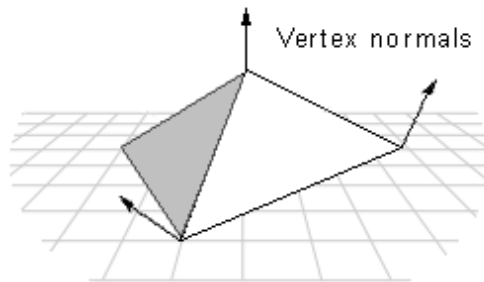


Figure 2.10: Illustration shows each of the vertices' normal vector. Source: [docs.microsoft.com/en-us/windows/uwp/graphics-concepts/images/vertnrml.png](https://docs.microsoft.com/en-us/windows/uwp/graphics-concepts/images/vertnrml.png)

## 2.5.8 Transform matrices

### Model matrix

When a model is loaded into vertex buffers, it has its vertices in a fixed local object space, usually around -1.0 to 1.0 in each axis. If we have a model of a tree, and want to draw several of them, they would all be drawn in the same location. It would be much better if they were spread out, and possibly even scaled or rotated differently. This is where the model matrix, also known as the world transform matrix, comes in.

In the vertex shader, we can transform the position attribute of each vertex as we wish. The shader can declare a matrix that can be set by the application, and it will be multiplied by the position. The vertex will then be transformed into world space.

### View matrix

In most applications, we want to have a camera that can be moved around, rotated, or able to zoom in and out. This can be thought of as a model matrix for the camera itself. We already have the vertex in world space, but now it must be transformed into view space. After multiplying the world space vertex with the view matrix, we have a view space vertex. [27][24]

### Projection matrix

The vertices need to be projected onto the screen, and this is done by transforming them into clip space. As you might imagine, this clips the vertices that are outside the screen.

### Model view projection

The final matrix we get is known as the model view projection:  $MVP = Projection \cdot View \cdot Model$ . To get the vertex position in clip space, we multiply the local space position with the model view projection matrix:  $Clip\vec{Position} = MVP \cdot Vertex\vec{Position}$ . The clip position is further transformed into normalized device coordinates (NDC), ranging from (0,0,0) or (-1,-1,-1) to (1,1,1), by doing perspective division:  $\vec{ndc} = \frac{(x, y, z)_{clip}}{w_{clip}}$ . This is automatically done by the vertex shader after it has finished execution. Finally, the NDC are mapped into the viewport space, and transformed into screen coordinates by the rasterizer. [57]

### Perspective projection

A perspective projection mimics how humans see the world. When an object is far away, it seems much smaller than it actually is. The opposite goes for nearby objects. This creates a line of sight effect where points are converging in the distance. The axes become narrower. The viewer perceives images projected with a perspective transformation as a realistic 3D world with depth. [36]

### Orthographic projection

An orthographic projection collapses the three dimensions into two, as seen in figure 2.11 below. The depth of each vertex becomes indistinguishable, but the camera's z-coordinate can still affect what is seen or not.

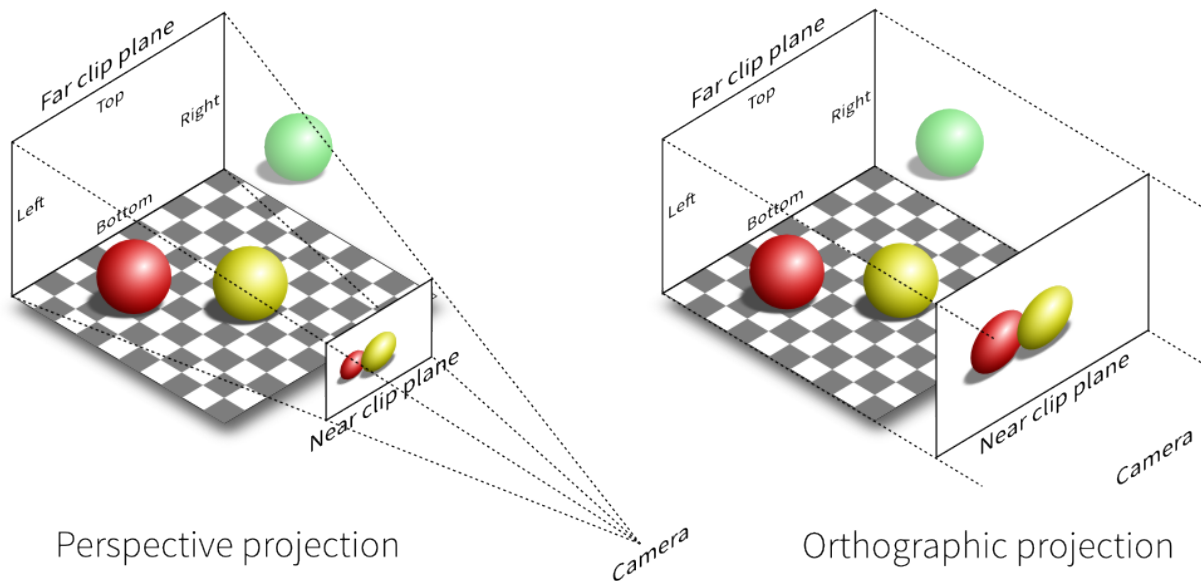


Figure 2.11: Illustration comparing perspective and orthographic projections. Source: [i.stack.imgur.com/q1SNB.png](https://i.stack.imgur.com/q1SNB.png)

### 2.5.9 Texture interpolation

A texture is often drawn minified or magnified, and it needs to be interpolated to do this. Two common interpolation techniques are known as nearest neighbour and bilinear interpolation. [61]

The differences can be seen in figure 2.12.

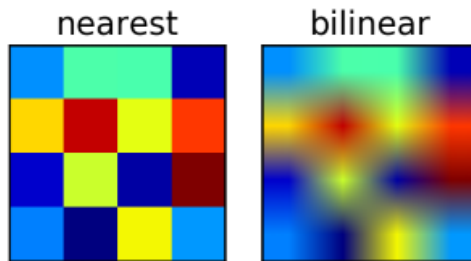


Figure 2.12: Comparison of nearest neighbour vs bilinear interpolated texture scaling.

### 2.5.10 Mipmapping

When looking at 3D objects from far away, the textures can seem noisy due to aliasing. This is visible on the left side in figure 2.13. The solution to this is generating mipmaps, which are copies with lower resolutions. It is common to repeatedly halve the resolution until  $1 \times 1$  is reached. An extra benefit is the increased render speed, but it comes at the cost of additional memory usage. [7][39]

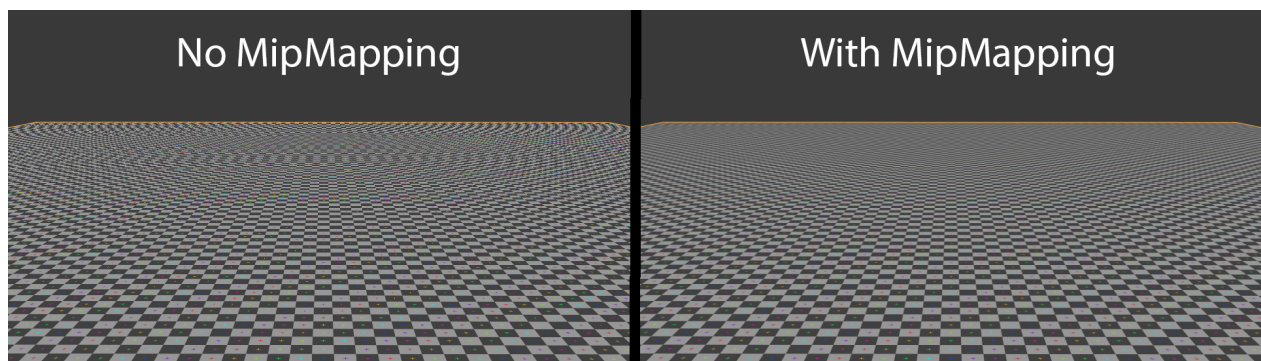


Figure 2.13: Comparison of with and without mipmapping, showcasing the aliasing effect. Source: [upload.wikimedia.org/wikipedia/commons/5/59/Mipmap\\_Aliasing\\_Comparison.png](https://upload.wikimedia.org/wikipedia/commons/5/59/Mipmap_Aliasing_Comparison.png)

### 2.5.11 Multisample anti-aliasing

Vertices must be rasterized when rendered to the screen. This results in scenarios such as to the left in figure 2.14. Two points compete to become the resulting pixel value. This can result in lines that look uneven. Multisample anti-aliasing is a way of reducing this conflict. [29]. Again looking at figure 2.14, on the right side, we see four samples are taken. They are averaged to give a blending color. The higher the number of samples taken, the more accurate the result will appear.

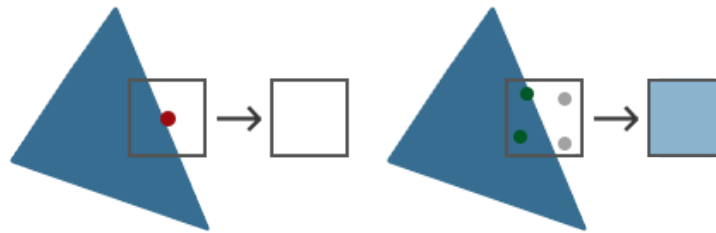


Figure 2.14: Illustration showing how a pixel with 4 samples is anti-aliased. Source: [learnopengl.com/Advanced-OpenGL/Anti-Aliasing](http://learnopengl.com/Advanced-OpenGL/Anti-Aliasing)

### 2.5.12 FreeType

The FreeType library is used to load TrueType fonts, and render glyphs to a bitmap. FreeType provides positioning information such as kerning, and other glyph metrics. [20] The horizontal metrics of a glyph can be seen in figure 2.15.

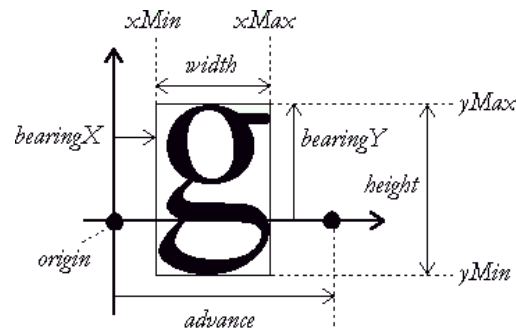


Figure 2.15: The metrics of a glyph. Source: [freetype.org/freetype2/docs/tutorial/metrics.png](http://freetype.org/freetype2/docs/tutorial/metrics.png)

### 2.5.13 Unicode

Unicode is a character encoding standard, intended to represent text in any writing system. It includes symbols in categories like objects and emoticons. [64] While Unicode defines the code points for each character, it is up to an implementer to decide how they are to be encoded in bytes.

### 2.5.14 Skeletal animation

Skeletal animations is a method of animating 3D models. It can also used for 2D sprites, but we will not use or cover that here. Instead of morphing the vertices from one frame to another, they are translated, rotated, and scaled in group by bones.

A bone is a transform matrix. They are stored hierarchically, such that parents affect their children. In figure 2.16, the `upperarmL` bone affects the `forearmL` bone. If a parent bone rotates a few degrees in some axis, the children bones inherit that rotation. [18]

Each vertex has a set of weights, which should all add up to 1. The higher a weight is, the more influence the associated bone has over the vertex. Vertices surrounding the `forearmL` bone, will likely have a very high weight associated with it. A vertex may be affected by many bones. Usually, programs will discard the lower weights if there are more than 4-6 in total. An extra weight means one additional matrix multiplication in the vertex shader. Each vertex also needs to store 6-8 more bytes per weight and bone, which can add up to a lot in detailed models. [62]

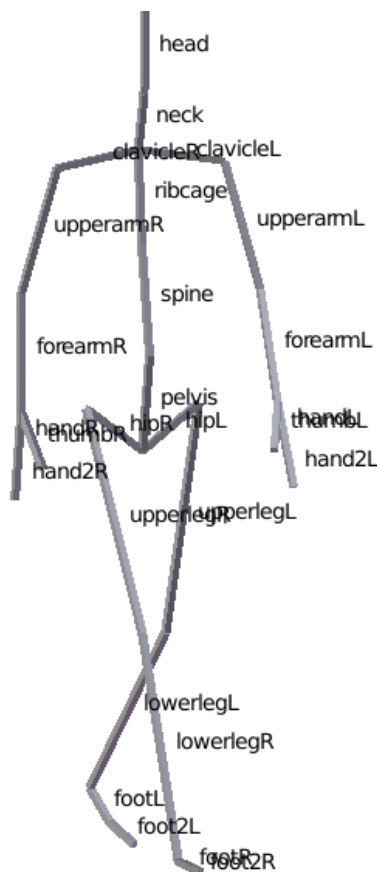


Figure 2.16: Image showing the bones of the character's skeleton.



## 2.6 Height maps

Height maps are two-dimensional arrays of numbers, each representing the height at a given horizontal point. They are often used to represent terrain in 3D worlds, as shown in figure 2.17.

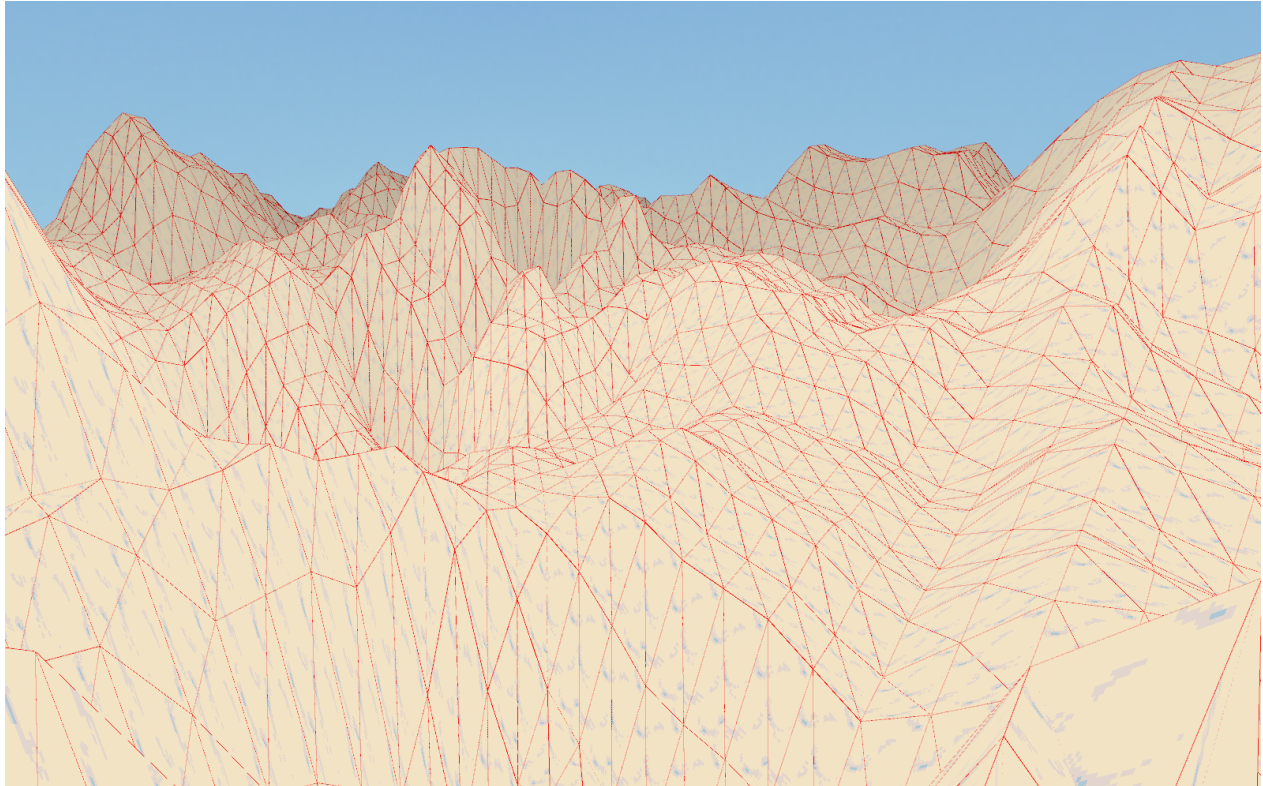


Figure 2.17: Screenshot of the world editor, showing the terrain height map with the grid.

## 2.7 Picking

In 3D applications, one often needs to translate the 2D position of the mouse to the 3D world position. This is known as 3D picking. There are two common approaches to this problem. The first approach involves unprojecting the mouse position, for both near and far  $z$ . This gives a ray, with an origin and direction, which can be used to perform a ray test against meshes. [2] The second approach is simpler, but sacrifices some performance. Render the scene a second time, with each mesh having a unique color, representing an identifier. Using a function such as `glReadPixels` or equivalent, the pixel at the location of the mouse cursor can be retrieved. [17]

## 2.8 Relational databases

A relational database is a collection of sets of data records, managed by an RDBMS (Relational database management system). [44]

### 2.8.1 SQL

Structured Query Language (SQL), originally developed in the early 1970s, is a programming language used to perform CRUD (Create Read Update Delete) operations on relational databases. There are many variations of this language, as most RDBMS use their own. Some examples are SQL/PSM, PL/pgSQL, and T-SQL. [66]

## 2.9 Web development

### 2.9.1 PHP

PHP Hypertext Preprocessor, which name is a recursive acronym, is a widely used scripting language for developing the backend of websites. First developed to add some simple front-end logic, to be used alongside existing languages, it quickly became the replacement of those other backend languages. This was to the original creator's dismay. Recent versions have aimed towards improving PHP specifically for heavy backend usage. [48][63]

# Chapter 3

## Method

### 3.1 Tools and libraries

#### 3.1.1 Reasoning behind using C++

A framework of this sort can be developed in any language, but it doesn't make sense to do so in most languages. In C++, there are no standard graphics, audio, or network APIs. For most areas we touch on, you are forced to use existing third party libraries, or directly use the platform's API. Since C++11, there is a standard concurrency library available, which we will take use of. It could be interesting to do this project in C instead, but there are many language features in C++ that make development smoother. Lambdas, auto, templates, and containers, are only a few of those.

#### 3.1.2 Using CMake to generate project files

CMake is a cross-platform utility for generating project files for many development environments. It uses a custom scripting language, which configures the project details. It's possible to specify include directories, source files, linker settings, targets, defines, and more. If using Visual Studio 2019 on Windows, the following command will generate the project: `cmake -G "Visual Studio 16 2019"`. If on Linux, running `cmake -G "Unix Makefiles"` will generate the makefile to build the project. This makes it easy to get quickly started when developing on a new computer.

### 3.1.3 Third party libraries

Although it would be wonderful not to, we do have to use some third party libraries. Here are some short summaries of why we use each of them:

- GLM helps us with various interpolations and matrix operations
- Sprites and textures are necessary, so we use libpng to load PNG images
- We use FreeType to load TrueType fonts, and render text with them
- Loading and playing audio is done using libogg and libvorbis
- We import COLLADA files with Assimp
- ImGui lets us quickly make user interfaces for tools
- The server must connect to the database. To do so, we use libpq, the official PostgreSQL connector
- We use GLEW (OpenGL Extension Wrangler) to load OpenGL extensions, since Windows does not support modern OpenGL versions out of the box. The library loads all the supported OpenGL functions on the system.

## 3.2 Miscellaneous framework features

### 3.2.1 Event listeners and queues

Events allow for other modules to extend the existing functionality. Any code with access to the event, can listen to it and attach their own handler. This aids in reducing coupling.

Signal events have no information carried with it, and only serves as a notification that something happened. Message events on the other hand, requires to be declared with a data type, such as an int or a struct. The event handlers are passed in as lambdas, and a listener identifier is returned. The identifier is used to later detach the handler, usually in the destructor. This is important to remember, because the event does not know if the lambda's capture is still valid.

An event queue is a container for event messages. Instead of emitting the event directly, it is moved into a queue. This is useful to simplify synchronization. Not only for multiple threads, but for tasks that need to be split up as well.

## 3.2.2 Debugging and metrics

### Debug logs

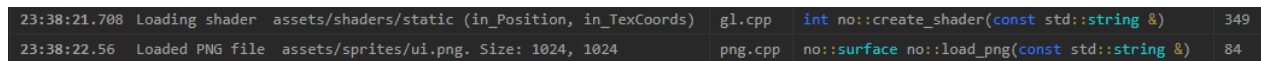
Logging is done using the global `no::debug::append` function. Several macros are defined, to make getting detailed output as short and easy as possible. Four types of log messages are available: Message, warning, critical, and info. The macros have a shorthand for each, and also have two extra variations for them, as seen in figure 3.1. The regular macros use the default log, 0. Figure 3.2 shows how output of these macros look like with the default style.

---

```
#define DEBUG(ID, TYPE, STR) append(ID, TYPE, __FILE__, __FUNCSIG__, __LINE__, STR)
#define MESSAGE(STR)          DEBUG(0, message_type::message, STR)
#define MESSAGE_LIMIT(STR, LIMIT) DEBUG_LIMIT(0, message_type::message, STR, LIMIT)
#define MESSAGE_X(ID, STR)    DEBUG(ID, message_type::message, STR)
```

---

Figure 3.1: Some macros for logging. Note that they are simplified for this figure.



```
23:38:21.708 Loading shader assets/shaders/static (in_Position, in_TexCoords) gl.cpp int no::create_shader(const std::string &) 349
23:38:22.56 Loaded PNG file assets/sprites/ui.png. Size: 1024, 1024 png.cpp no::surface no::load_png(const std::string &) 84
```

Figure 3.2: Example debug log output.

### Redundant bind metrics

Redundant texture, buffer, and shader binds are expensive. If the `MEASURE_REDUNDANT_BIND_CALLS` definition is true when compiling, the redundant binds counter will be enabled. It measures all bind calls to the graphics API.

## 3.3 Platform abstraction layer

### 3.3.1 Designing the interface

When looking at what similar platforms' APIs provide, there is often great overlap between them. It would be reasonable to assume that most general purpose operating systems let you manage windows, read user input from devices such as keyboards, and communicate with the network. Each platform has its own way of doing things, but these operations still have the same concept.

On Windows, one must register a window class before being able to create a window. It is also event driven, which means the caller must provide a callback function to process these events. In contrast, some platforms may be purely based on polling. Additionally, each platform has their own set of bugs. Some of these are not fixed due to legacy compatibility. The abstraction will encapsulate the platform's oddities, and provide one unified interface.

### 3.3.2 Abstracting the entry point of a program

On Windows, the `WinMain` function must be defined in order to be able to create windows. This is something that programs using the framework should not have to consider. The programs must be able to compile on any platform supported by the framework. This can be solved by having the programs define functions that the framework expects. These are then called appropriately. A couple of functions are expected for compilation to be successful.

```
configure()
```

When the framework has done minimal initialization, this function is called to allow the program to prepare for the subsequent full initialization. There is also an event which global objects can listen to, which is emitted directly after this call.

```
start()
```

At this point, the framework has initialized the audio and network systems. To create a window, the program must define a class which inherits from the abstract `program_state` class in the framework. Using the `create_state<T>` function, with `T=my_program_state`, a window will be created. The framework constructs a state object, and calls its `update` and `draw` methods when necessary. When it is time to shut down, a pre-exit event is emitted to allow for additional cleanup.

### 3.3.3 Managing windows

The PAL declares a platform specific window class depending on the compilation target. This class is defined in a source file, which is only compiled on that target platform. To avoid the mess of pre-processor conditions, the alias `platform_window` is used to refer to the correct class name. The window class constructs the correct platform window upon its construction, and the methods mostly call that object's methods.

#### Window classes

Not to be confused with the actual C++ window classes, a window class in WinAPI is a structure that defines how a window should be created. The registered window classes are unique to the running application, and will be unregistered automatically by Windows when the application terminates.

#### The window callback procedure

In the Windows implementation, a callback function known as `WindowProc` must be given when registering a new window class. It is possible to use `DefWindowProc` as the default handler, for when the events are not necessary. Since `WindowProc` is a contextless C function, it is not possible to bind a member function or create a lambda for it. This means there must be a way to pass state to the function, as it will potentially handle multiple windows. Thankfully, Windows does provide a way with the functions `SetWindowLongPtr` and `GetWindowLongPtr`. As one might expect, they associate the Windows window handle with a long pointer. The value is set to the pointer to the respective `window` object.

#### Updating the mouse cursor icon

When the mouse cursor moves anywhere in the window, a *set cursor* event is generated. If this event is delegated to `DefWindowProc`, the cursor icon will be changed according to what Windows sees fit. In graphical applications without Windows forms, this will just set it to the default arrow. However, if the event is discarded entirely, the cursor icon will not change at all. It is useful to handle the event to set a custom hardware accelerated mouse cursor, instead of simply hiding the real cursor and drawing another in its place. Keep in mind that hardware acceleration in this case means to use the graphics card's hardware cursor. This gives a cursor icon that is more responsive than the application, even when running at the maximum frame rate.

### 3.3.4 Dealing with user input

Each window has its own instance of a mouse, and a keyboard. This is intended to make it easy to emit input events only to listeners for that window. It is also possible to access the mouse and keyboard state directly, without the use of events. When retrieving the mouse position in Windows, it is not enough to use `GetCursorPos`. The position will be relative to the screen, so `ScreenToClient` is used subsequently. This is an example of how a platform may have a common feature, but with a unique take. The keyboard class has a great example, showcasing how events can be used to implement a missing feature. In the constructor, it listens to its own key press and release events, to mark whether a key is currently down or up.

### 3.3.5 Render context

To use hardware accelerated graphics, the window needs a render context. The abstraction is done similar to how `platform_window` was defined, except it is named `platform_render_context`.

#### Creating a context

With WGL (Windows OpenGL), it is fairly simple to create the context for a window. First, a pixel format descriptor must be configured and applied to the device context. The descriptor structure is passed to `ChoosePixelFormat`, which finds suitable formats that are available. If at least one format is available, it is enough to use `DescribePixelFormat` once, and get the first result written into the descriptor structure. All that is left is using `SetPixelFormat` to apply the format to the device context, then finally use `wglCreateContext` to create it and get its handle.

#### WGL ARB extension

This is when you realise it is not possible to use multisample anti-aliasing (MSAA) with this context. To take advantage of MSAA, the ARB extension `wglCreateContextAttribsARB` must be used in place of `wglCreateContext`. The irony of using this modern extension, is the fact that it is required to create a legacy context to create the ARB context. Not only does this mean it is necessary to create two contexts, of which one will be immediately deleted. Thanks to how Windows maintains windows, it is also not possible to change the pixel format descriptor after the fact. Essentially, this means the window itself must also be destroyed, before creating the new window. Although a weird characteristic, the user will generally not see the first window.



### 3.3.6 Network sockets with WinSock

The Berkeley sockets API is generally available on most platforms, but each platform implements its own quirks. Not only that, but Windows provides extensions that change the classic Berkeley flow entirely.

In order to take advantage of the benefits of these extensions, the network interface for the framework must allow for an implementation using either Berkeley sockets or WSA extensions. The final network API can be seen in appendix C.

Although it should not be difficult to add more features later, the new API is primarily designed around using TCP/IP sockets.

#### The AcceptEx extension

This function pointer is loaded with `WSAIoct1` using a listening socket. This is a far more powerful alternative to the regular `accept` function, as it is used with IOCP. Unlike `accept`, this extension needs an already opened socket that will be used for the accepted connection.

### 3.3.7 Synchronization of socket events

When an `accept`, `send`, or `receive` operation has completed asynchronously in an IOCP worker thread, the event is pushed to a queue. The application must call `synchronize_sockets` as often as it deems necessary per frame, usually once. It locks the mutex for a socket, then emits its queued events. This synchronization avoids sporadic mutex locks that could otherwise occur at any point.

## 3.4 Graphics

### 3.4.1 Generic vertex layouts

Applications must be able to write shaders with custom attribute layouts. Some common layouts are defined by the framework, but they cannot cover every need. The graphics API utilizes templates to allow for custom layouts. A vertex data structure must contain a `static constexpr` array of `vertex_attribute_specification`, named `attributes`, to be used as a template parameter. An example can be seen below, in figure 3.3.

---

```
struct textured_vertex {  
    static constexpr vertex_attribute_specification attributes[] = { 3, 2 };  
    vector3f position;  
    vector2f tex_coords;  
};
```

---

Figure 3.3: An example of what a custom vertex structure may look like.

It is also possible to configure each attribute more explicitly, using other constructor overloads. The default primitive data type for an attribute is float, but integer and byte may also be specified.

#### Structure padding

Note the importance of keeping the structure properly padded. If an 8-bit field were to be inserted in the middle, it would likely be padded to 32 bits. Small fields must either be declared at the bottom, or an attribute may be used to pack the structure. Using Visual C++, this is done by wrapping the structure with `#pragma push(pack, 1)` and `#pragma pack(pop)`, but this won't work elsewhere. For GCC, `__attribute__((packed))` must be specified at the end of the struct declaration. We mostly use floats, so there has been no need to use packed structures.

#### Automatically binding vertex attributes

It would become bothersome to keep manually associating attribute names with their location indices. The shader loader therefore parses the vertex shader attributes in order, and binds them to locations in the order they appear.

### 3.4.2 Importing and exporting 3D models

#### Reasons to convert from exchange formats to custom formats

Formats such as COLLADA, FBX, and OBJ, are designed to be as general purpose as possible. They are typically saved as XML or other text formats, and don't focus much on optimizations. Some models might store duplicate vertices, which is preventable by enumerating them in an index list. The faces of a mesh can consist of quads, and those would have to be triangulated before rendering them. Additional post-processing of the models can also be done, such as optimizing for cache locality by reordering the faces. Most importantly, it is far slower to parse an XML file than to quickly consume a binary blob. This is why it is important for an application to have its own format. The startup time is noticeably faster.

#### Importing models with Assimp

Assimp, short for Open Asset Import Library, is an open source library to import models in many exchange formats. It is one of the most popular open source model importer, and has many useful post-processing features. One of its downsides is the incomplete implementation of many of these importers. FBX support is a hit or miss situation with this library, and that is why models are exported to COLLADA instead.

Blender is likely the most used free 3D modeling software, but has poor exporting support. For example, the COLLADA exporter is known to only export the active animation. [4][23] This makes it a tedious task to update models with more than just a few animations, as each animation must be exported individually.

Merging the animations happen automatically in the toolkit's model manager. It starts by exporting each model to the custom format, then the resulting model uses the static data of the first model. All following models are then validated against it, to discard or warn about any mismatches. If successful, the animation is added to the resulting model.

### 3.4.3 Implementing skeletal animation

Skeletons consist of a hierarchy of nodes, associated with bones. Each node and bone has a transform matrix, which is animated with key frames. The matrix is local to the node, and needs to be multiplied by the animated transform matrix of the parent node. Animating a node's transform is done by finding its current position, rotation, and scale key frames, then multiplying them together in that order. To update the bone, its transform is set to the model's root transform, multiplied by the node transform, then by the identity bone transform. To make the animation scalable with a large number of key frames, the last key frame is kept track of.

### 3.4.4 Optimizing skeletal animations

Skeletal animations are one of the most expensive elements. Initially, the frame rate with many objects on screen was not feasibly high enough. Upon reviewing the code, it seemed that making the animations multithreaded is a possibility to improve the performance. The system was not designed with concurrency in mind, but that was fortunately not a problem. It's up to the game whether or not to create any threads, as control is delegated to the application. Synchronization is done with a function call, which locks a couple mutexes. Since the thread may sleep, and only update every 17 milliseconds, not much waiting for the write mutex is expected. The fact that animations were updated every frame earlier, was likely also an impact on the performance gains.

### 3.4.5 Separation of logic and rendering

It's useful to separate rendering from game logic. This allows the server to update everything, without having to render it. The opposite goes for the world editor, which should only draw the world. It doesn't make sense for the world to update when editing, as it would lead to a lot of unpredictable behavior.

### 3.4.6 Surfaces

A surface contains an array of pixels, and has some useful features. It lets you render other surfaces on it, lines, rectangles, and circles. You can also flip them horizontally and vertically, or flip  $x$  segments horizontally independently. The latter is useful for sprites.

Surfaces are non-copyable via the constructor and assignment operator. The reason is to avoid unwanted copies, as it involves a memory allocation.

### 3.4.7 Rendering text

The FreeType library is used to load TrueType fonts, and render glyphs to a bitmap. To render a string of text, some extra work is required. FreeType does provide positioning information such as kerning, and other glyph metrics. However, it's not made to render text out of the box. New lines are not handled directly either. The framework wraps FreeType, and offers a simple API which does that work for you. Rendered text is returned as a surface.

### 3.4.8 Tile borders

Due to how mipmapping works, there are artifacts caused by the lower precision. The tiles are textured by a texture atlas, where if the texture coordinates are even one pixel off, it will cause a visible difference. The grass tiles will suddenly have part of the dirt tile, and it will show brown lines along the edge of grass. To solve this, and still use mipmaps, a simple algorithm was made to add borders to each tile. The size of the border is configurable, and generally needs to be slightly over half the size of a tile. The reason being because the smallest mipmap is a  $1 \times 1$  texture.

## 3.5 Game features, and interacting with the world

### 3.5.1 Picking

In 2D applications, it's simple to translate the mouse position into a world position, by offsetting by the camera position. However, it's a bit more complicated with 3D projections. As explained previously, the mouse position must be unprojected for both near and far  $z$ . Then the resulting ray can be used to perform collision tests against meshes. Instead of this typically used method however, the pixel reading method is used. The reason being its simplicity.

### 3.5.2 Pathfinding

The A\* algorithm is used for pathfinding. The game is tile based, thus searching the terrain is straight forward.

### 3.5.3 Fishing

Fishing is a skill which the player can train, and is a source of food. Food can be used to rapidly heal during or after combat. To fish, a fishing rod can be casted up to 15 tiles away from the player. The farther away the bait lands, the higher the chance of catching a fish. Every couple of seconds, the server moves the bait closer to the player, and sends a packet to notify clients. In figure 3.4, we see a flowchart of how fishing works.

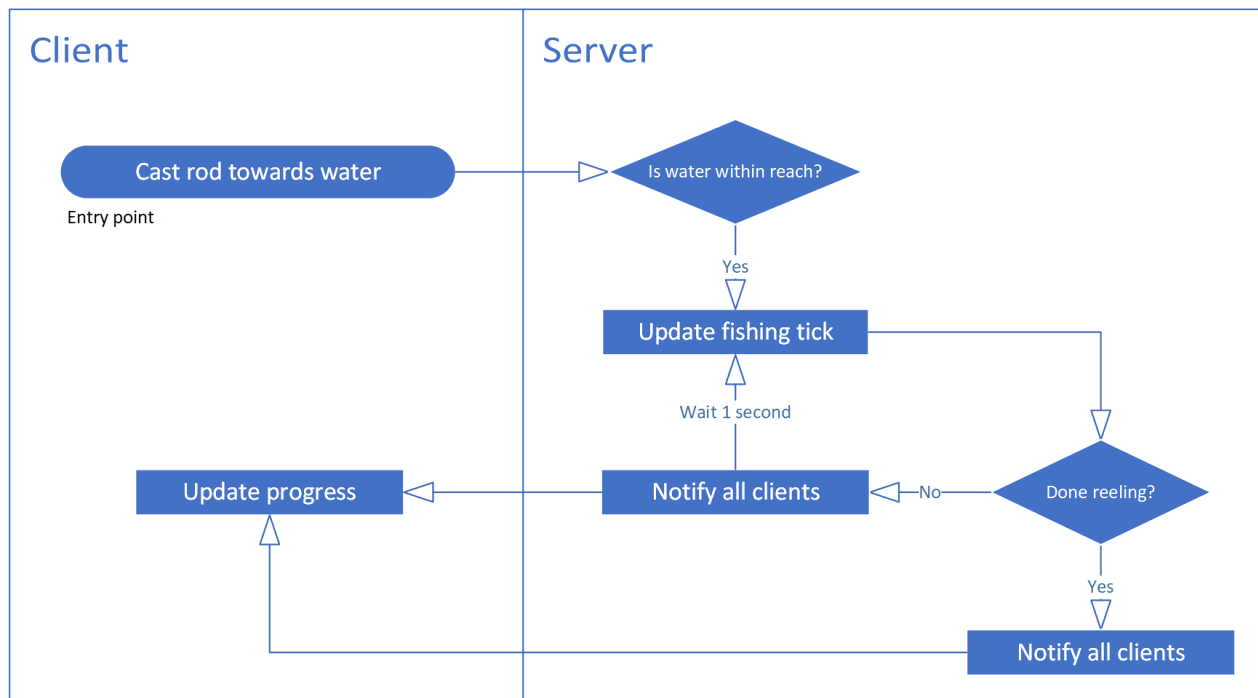


Figure 3.4: Flowchart diagram showing the fishing process.

### 3.5.4 Trading

Two players can trade with each other. Figure 3.5 shows the trading process with a flowchart. One of the players must send a trade request, and the other player has to accept the request for the trading to begin. At this point, both players may offer items from their inventory. When a player is satisfied with the trade, they can accept it. Once both players have accepted, the trade is successful. Otherwise, if one of them declines at any point, the trade immediately ends.

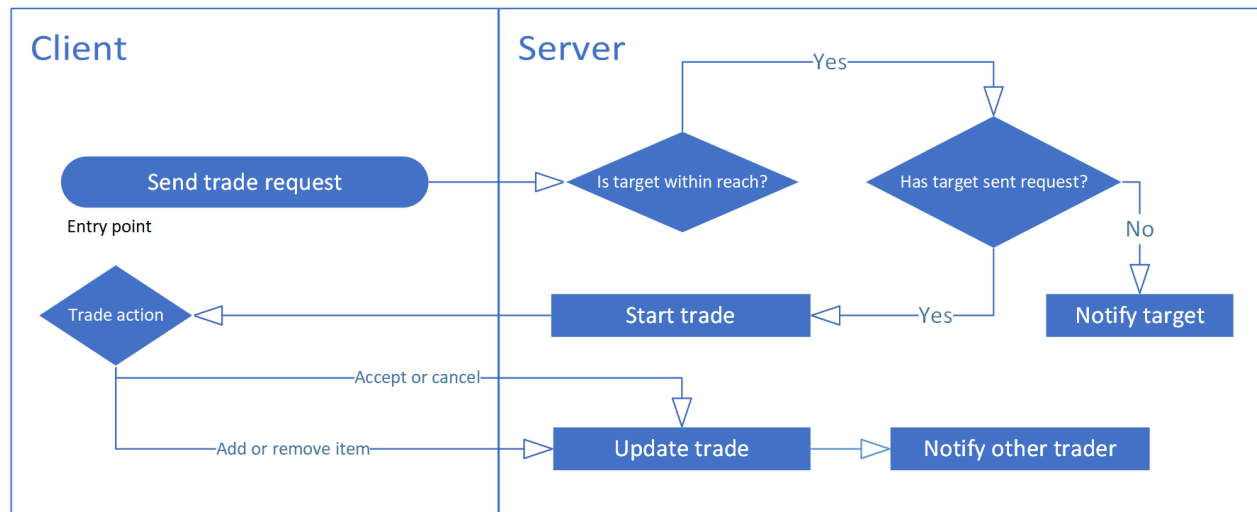


Figure 3.5: Flowchart diagram showing the trading process.

### 3.5.5 Quest system

Quests are made up of tasks and hints, as well as being used in scripts. The hints are unlocked by a certain task, while the tasks are completely independent from each other. Scripts can check whether or not a quest task is completed, or modify their completion status. This is slightly similar to game variables, as the same effects can be achieved. However, quests are listed in a user interface. The same cannot be done if variables were used independently.

## 3.6 Node based scripting system

A script consists of nodes connected to each other. Originally only meant for dialogues, it became evident as object events and such were added, that it's general enough to be used for those as well. Many nodes currently exist, and it is simple to add new ones. Each node has a different number of outputs, as configured by its output type.

### **3.6.1 Script context**

When scripts are run, they need some context. The context consists of the world state, and the player that is associated with the event. The player includes its object, stats, inventory, equipment, quest states, and variables. There are different types of scripts, such as for dialogues, object events, and item behavior. Some nodes are only available on their respective script type.

### **3.6.2 Game variables**

Variables are used to track the state of the world from a player's perspective. For instance, if the player kills a wild boar, the event script can increment a counter for boars slain. The counter can then be used in a dialogue script, to lead the dialogue in a different direction with conditionals. There are multiple variable types available: Integer, float, boolean, and string. There is also a distinction between local and global variables. A local variable has a scope, which is typically only associated with the script it's defined in. However, a global variable can, as the name implies, be accessed anywhere.

### **3.6.3 Dialogues**

Dialogues are scripts that are based around the message and choice nodes. Each message blocks the flow of the script, until a choice has been selected.

### **3.6.4 Scripted object events**

Every object definition can have scripted events associated with it. The script is created as normal in the script editor, just like dialogues. Each object definition can only link one script per event, which must encompass all the logic needed to handle it.

### **3.6.5 Item behavior scripts**

Item definitions also have events that are linked to scripts. In addition to the default events, they can have a dynamic number of custom events added to them as well. If an item is used on another, an event is emitted. The script can check which items were used on each other, and decide what to do based on that, as well as other conditions like usual.



## 3.7 Packet definitions

### 3.7.1 Defining new packets

The packets are defined by structs, encapsulated within namespaces. It's important to be able to distinguish which packets are meant for the client or for the server. To keep it simple, there are two main namespaces for packets: `to_client`, and `to_server`. Each namespace also contains three common inner namespaces: `updates`, `lobby`, and `game`.

All packets share some useful commonalities. Each has its own unique static identifier, relative to the identifier of the namespace it is declared in. There are also two member functions, which are self-explanatory: `read(io_stream&)` and `write(io_stream&)`. Finally, packets define a constructor that receives an I/O stream to be read. This is useful to send the packet to a function with list initialization. Example invocation: `on_chat_message(client, { stream })`. The function declaration looks like this: `void on_chat_message(int, const packet&)`.

While this is already useful, it can be made even simpler by defining a macro for each packet. The assumption then is the function and packet structure must share the same name, except prefixes.

### 3.7.2 Defining some macros

When dealing with long repeating patterns of code, it might be a good idea to write some macros to shorten it. Instead of declaring the structures and member functions for all the packets, they are reduced to a few keywords, as seen below in figures 3.6 and 3.7. Macros should be created sparingly, and it's often best to undefine them as soon as possible in the same file.

---

```
Packet(fishing_progress, 14)
    int32 instance_id = -1;
    vector2i new_bait_tile;
    bool finished = false;
Write(fishing_progress)
    stream.write(instance_id);
    stream.write(new_bait_tile);
    stream.write<uint8>(finished ? 1 : 0);
Read(fishing_progress)
    instance_id = stream.read<int32>();
    new_bait_tile = stream.read<vector2i>();
    finished = stream.read<uint8>() != 0;
End
```

---

Figure 3.6: An example of a complete packet, defined in `to_client::game`

---

```
#define OnPacket(N, P) case to_server::N::P::type: on_##P(client, {stream}); break
void server::on_receive_packet(int client, int16 type, no::io_stream& stream) {
    switch (type) {
        OnPacket(game, move_to_tile);
        OnPacket(game, start_dialogue);
        ...
    }
}
#undef OnPacket
```

---

Figure 3.7: Example of how a server may handle incoming packets with the help of a macro.

## 3.8 Editing the world's height map

There are two modes for editing the height map, where the affected tiles are the ones within the radius of the brush. The first mode will elevate the tiles' height by a fixed amount every tick the user holds the space key and left mouse button simultaneously. The elevation rate can be changed by the user at any time. The second mode is entered by ticking a checkbox, which locks the elevation of affected tiles to a custom value. The shift key can be used to negate the elevation rate in both modes. The former mode is useful for quickly shaping up hills and mountains, while the latter mode allows you to clear up areas for buildings and other objects.

### 3.8.1 Texture mapping the height map vertices

To be able to quickly modify the terrain, texture coordinates are automatically calculated based on the tile that is placed. Each tile has four corners, which are each stored as an 8-bit integer. Since there won't be 256 different tiles, we can use a bit mask to store other values too. The 6 lower bits represent the tile type, while the remaining 2 bits mark whether or not the tile is solid or water. To calculate the texture coordinates for the tile vertices, they are first mapped into a hash table. The keys are 32-bit integers, which pack the corners. Note that the flags are not part of these values. Figure 3.8 shows the bit shifting that is done to calculate the keys.

---

```
uint32 f(uint8 top_left, uint8 top_right, uint8 bottom_left, uint8 bottom_right) {  
    return (top_left << 24)  
        + (top_right << 16)  
        + (bottom_left << 8)  
        + bottom_right;  
}
```

---

Figure 3.8: This function shows the bit shifting that is done to pack the corners into a key.

## 3.9 Client states

### 3.9.1 Auto-update

The client must always be up to date to work with the server. Otherwise, the client might not understand the incoming packets, or send valid packets to the server. To ensure an updated client, it will query the server for the newest version. If it's already up to date, the client launches as normal. When a new version has been released, the server first transmits all the asset files for the game, as well as the executable.

Since some operating systems don't allow a running executable to be replaced in place, a bit of a workaround is needed. The solution is to rename the running executable, save the new one, then relaunch the client. When starting again, the old file is deleted by the updated client. A flowchart of the auto-update process can be seen in figure 3.9.

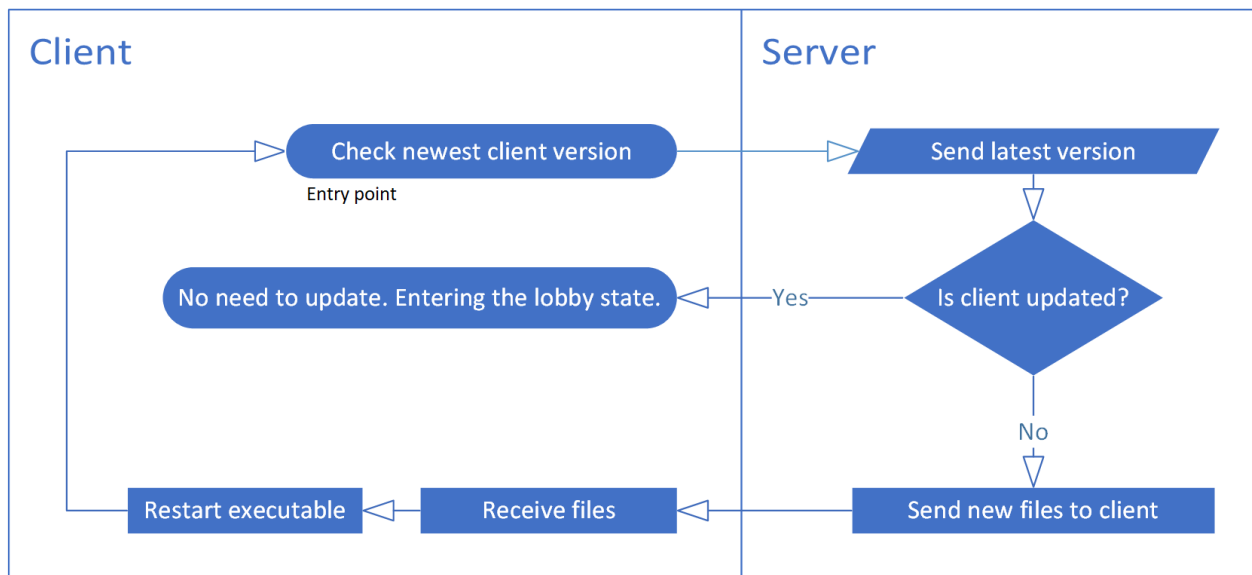


Figure 3.9: Flowchart diagram of the auto-update process.

### 3.9.2 Lobby

After the client is made sure to be updated, the user is presented with a login screen. This is where they write their username and password. A checkbox next to the input boxes allows the user to remember the username they type in. The value is saved to a local lobby configuration file, which is loaded the next time the lobby is entered.

Once the login button is pressed, a request is sent to the server to attempt authentication. If successful, the user can select which player or world they want to play on. Otherwise, the user is notified that the login attempt has failed. A flowchart of the login process is seen in figure 3.10.

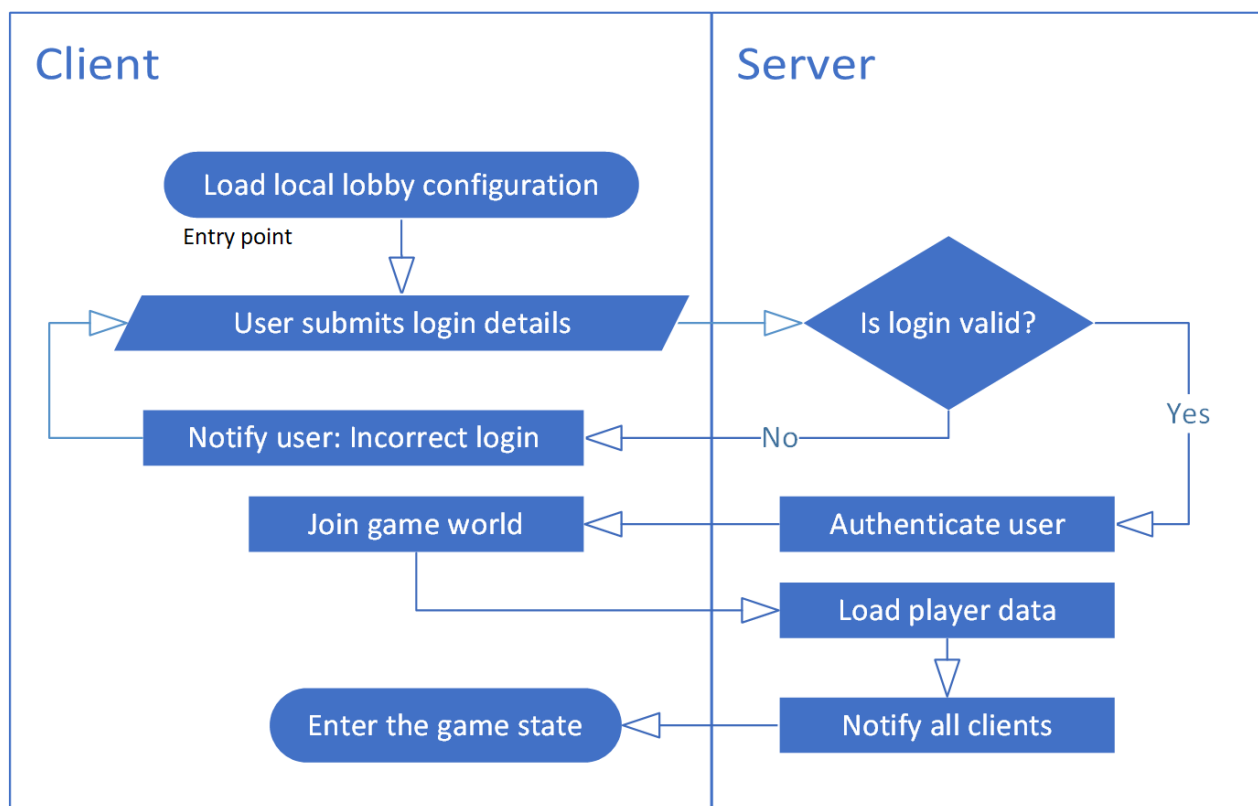


Figure 3.10: Flowchart diagram for the authentication process.

## **3.10 Server behavior**

### **3.10.1 Tracking player sessions**

When a client connects, its session begins. When the server receives a login request, with the player's username and password, it verifies the login. If authentication succeeded, the session is updated to reflect that. The player is not yet loaded into the game world at this point. The client must send a request, specifying which world to join. The idea is to allow the player to have multiple profiles in separate worlds. Once the server receives the request, the player data is loaded from the database. All of the loaded data is then sent to the client, while other players in the world is only notified about the details that is needed to draw the new player.

### **3.10.2 Validating player actions**

The server can never trust the client with anything, so each packet must be validated thoroughly. For instance, when a player wants to move to a new location, the client sends a packet with the target tile to the server. The server checks if the distance is within reach, and performs the pathfinding if reasonable. Each client is then notified about that player's new path.

### **3.10.3 Synchronizing with database**

When a player is loaded from the database, an interval timer is started. Every other minute from that point, the server writes the current player state to the database. The player is also saved if the client disconnects, or if the server is closing.

## 3.11 Database

### 3.11.1 Structure

The database has a simple design, and consists of 6 tables, as seen in figure 3.11. Most of the players' state is stored in the `variable` table. Its rows contain all the different flags and counters that are created when scripts run. The `item_ownership` table stores which items a player has in which container. The container field could be inventory, equipment, or warehouse.

It's possible for an account to be associated with multiple player profiles. There is no possibility of creating multiple profiles at the moment, nor is there a character selection screen. However, it's not difficult to add this feature later on.

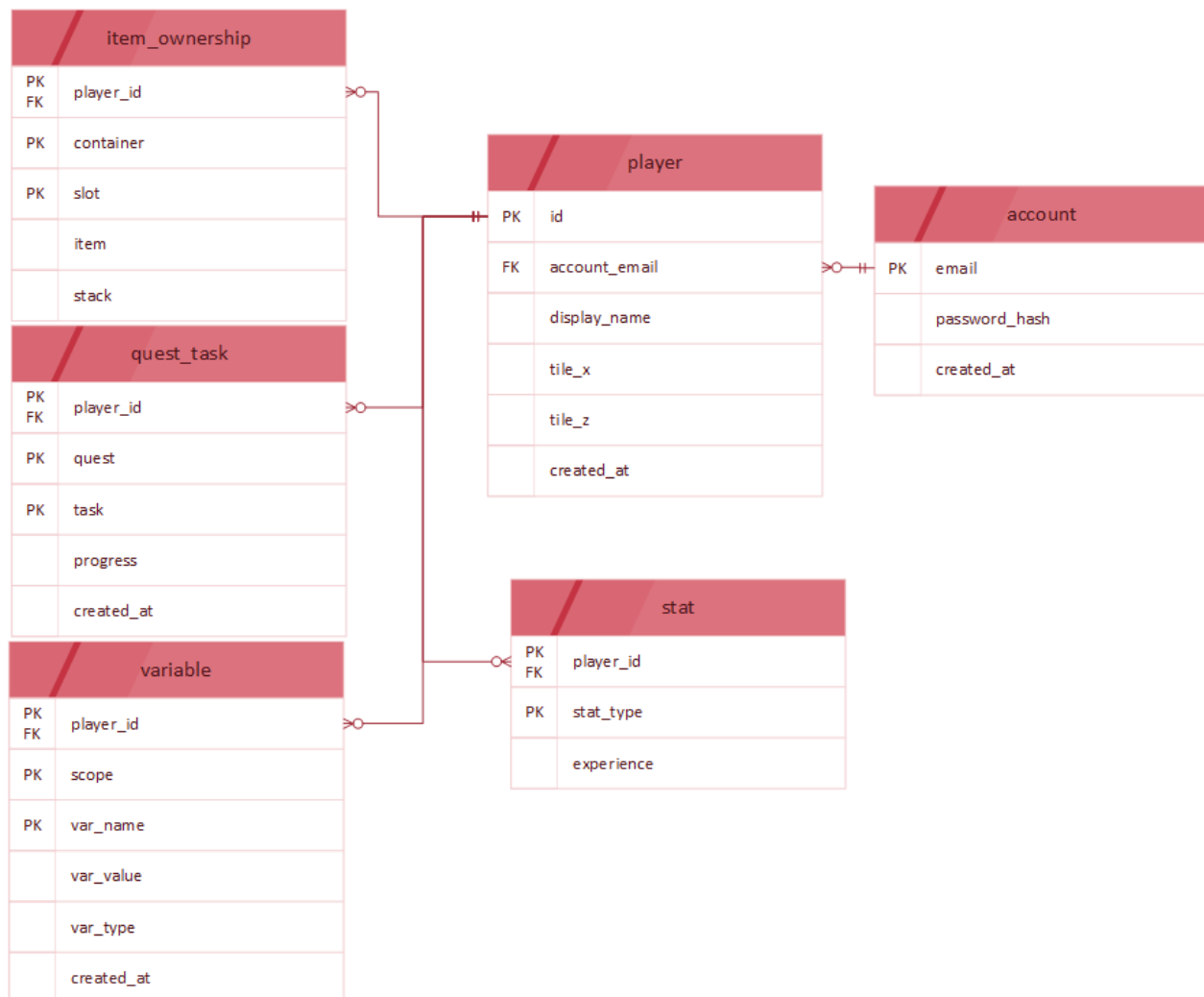


Figure 3.11: Diagram showing the structure of the database schema.

### 3.11.2 Stored procedures

Stored procedures are a feature in most RDBMS, and allows for saving queries to the database. [46] The queries may consist of multiple statements. There are some benefits to using stored procedures, over inline queries. When the query is saved, the RDBMS can optimize the query plan to a higher degree. The plan is calculated only during the creation of the procedure, unlike inline queries. Some caching is done either way, and it varies between systems how much performance is gained, but it generally never decreases performance.

Stored procedures also prevent any form for query injection attacks, on another level to that of prepared statements. Prepared statements sometimes don't protect against injections, when the driver is in the emulation mode, which is sometimes default. Additionally, it is a good idea to separate SQL queries from the applications. Especially when there are multiple applications connecting to the same database, performing the same operations.

There are still inline queries in the application code, but they are rather short, and exclusive to the application.

### 3.11.3 Upserts

Upserts are used in the stored procedures to create and update rows. It lets us have one query to perform both operations. [47] As we see in figure 3.12, a trigger is added to the insertion statement. If a conflict arises due to the primary key constraint, the update statement is executed. This saves us from having to create a lot of extra stored procedures.

---

```
insert into item_ownership (player_id, container, slot, item, stack)
    values (in_player_id, in_container, in_slot, in_item, in_stack)
on conflict
    on constraint pk_item_ownership
    do update
        set item = in_item,
            stack = in_stack;
```

---

Figure 3.12: The body of the stored procedure for updating a slot in an item container.



## 3.12 Website

The website backend is written in PHP, and handles user registration and leaderboards. It connects to the database, and calls the stored procedure for creating a new account to register users. It uses an inline query to find the top players for the leaderboard.

The passwords are encrypted using the `password_hash` function, with the default options as of PHP 7. [45]

The web directory has a `config.default.php` file, which must be renamed to `config.php`. The configuration file must be updated with the proper values.

## 3.13 Building the projects

It's only possible to build the project on Windows, as of writing this. The projects have only been tested with Visual C++, using the C++17 standard. To build the projects, CMake and Visual Studio 2017 or later should be installed. Each of the `VisualStudio15_x64.bat` files must be executed, which will generate the Visual Studio project files.

After generating the projects, copy each `default-config.hpp` file, and name them `config.hpp`. Change the configurations to what you want. At this point, you may build all the projects, and they should run successfully.

The framework also has a `UnixMake.sh` file for non-compilable experimental build on Linux.

# Chapter 4

## Result

### 4.1 Framework

The framework has met its minimum requirements, and is considered a success. However, we would have liked to test the abstractions by creating multiple implementations of some of the interfaces. It's unfortunate that, due to time constraints, this was not possible. The final interfaces can be seen in appendix [C](#) (network), [D](#) (graphics), [E](#) (audio), and [F](#) (platform). The implementations we managed to complete were as follows:

- Platform: Windows API
- Windowing: Windows' window API
- Render context: Windows OpenGL
- Renderer: OpenGL
- Network: Windows Sockets API
- Audio: Windows Audio Session API

## 4.2 Game client

Figures 4.1 and 4.2 show screenshots of the game client.



Figure 4.1: Screenshot of the game client, including interfaces.



Figure 4.2: Screenshot of the game client, without interfaces showing.

## 4.2.1 User interfaces

### Inventory interface

The inventory interface shows the items that are stored in your backpack, as seen in figure 4.3.

The items can be right-clicked, and options will appear in a context menu.



Figure 4.3: The user interface for the inventory tab.

### Equipments interface

The equipments interface shows the item equipped in each slot, as seen in figure 4.4. The slots can be right-clicked, and items may be unequipped.



Figure 4.4: The user interface for the equipments tab.

### Quests interface

The quests interface shows each quest on a separate row, as seen in figure 4.5. The idea is that you click on a quest, and the quest journal will appear. However, that was not implemented.



Figure 4.5: The user interface for the quests tab.

### Stats interface

The stats interface shows each stat on a separate row, as seen in figure 4.6. The yellow experience bar indicates the progress of reaching the next level of the respective stat. When hovering the experience bar, a tooltip will show the number of remaining experience that is needed.



Figure 4.6: The user interface for the stats tab.

### Trading interface

The trading interface, seen in figure 4.7, lets two players trade with each other. On the left, we see the other player's offers, while our offers are on the right. When one of the players press the accept button, the other player will be notified as shown in figure 4.8. The player who already accepted will see an alternative label, suggesting to wait for the other player to accept too.

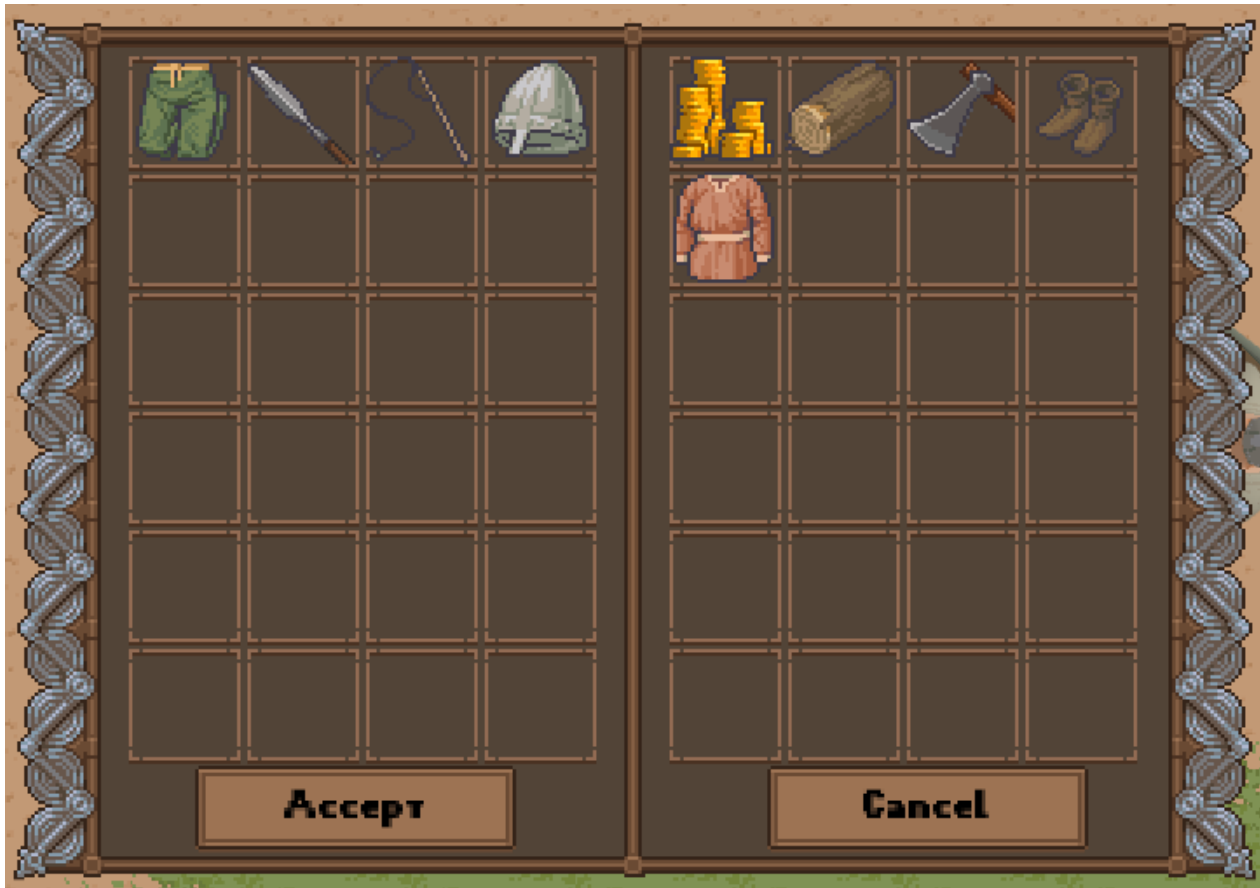


Figure 4.7: The trading interface, while the trade is still ongoing.



Figure 4.8: The trading interface when the other player has accepted. Accept button is hovered.



### Dialogue interface

The dialogue interface is shown in figure 4.9 and 4.10. The options are conditionally added, based on the nodes in the script. Figure 4.11 shows the script for the dialogue that is seen here.

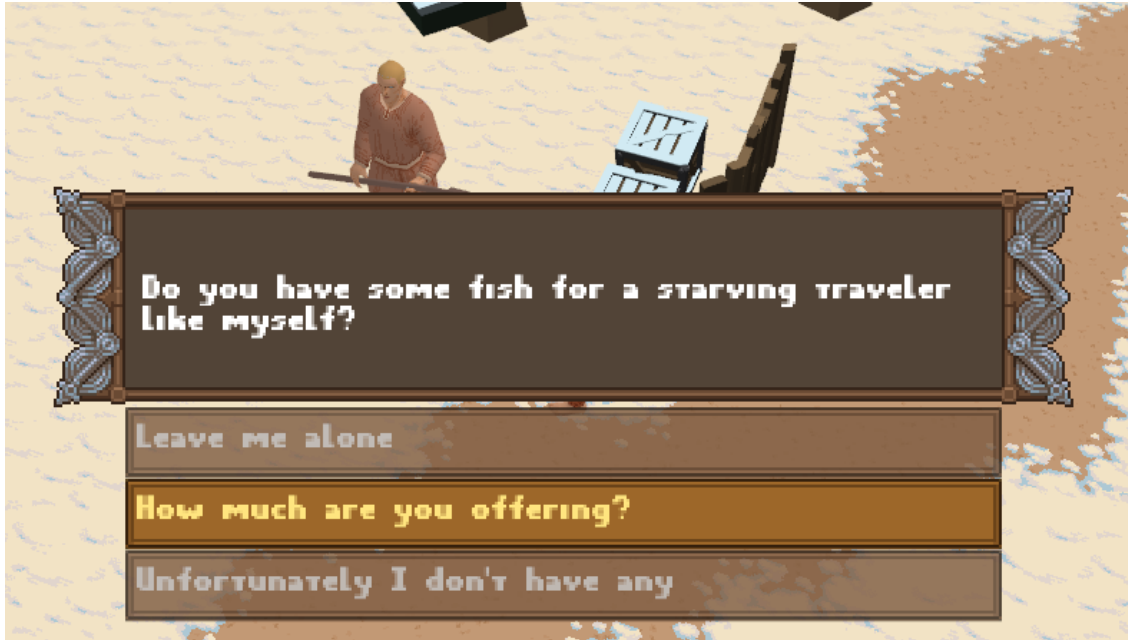


Figure 4.9: Going through the options in a dialogue.



Figure 4.10: Continuing with the dialogue after coming back with some salmon.

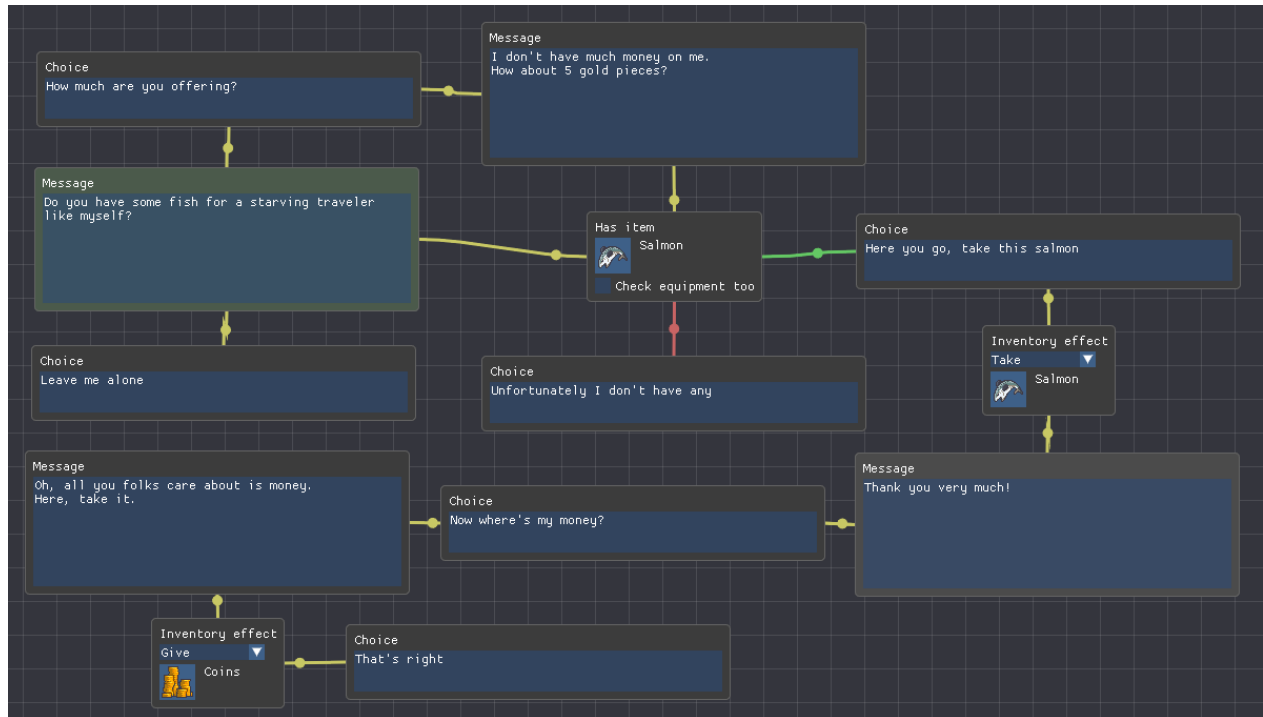


Figure 4.11: Script for a dialogue where a traveler wants to buy salmon from the player.

### 4.2.2 Head-up display

The HUD (Head-up display) shows useful information to the player. This can be how much health or stamina that the player has left, or having a minimap to see where you are in the world.

### 4.2.3 Minimap

The minimap shows a minified version of the world. To represent the terrain, we use green pixels for grass, blue for water, brown for dirt, and so on. For characters and other objects, colors such as white and red are used. This is updated every 200 milliseconds, or when the player moves a full tile, to avoid unnecessary rendering. The elevation of each tile is also indicated by the brightness of the respective pixel on the minimap. Figure 4.12 shows how the minimap looks like, over a snowy mountain and a river.



Figure 4.12: Screenshot of the minimap by a snowy mountain and a river.

#### 4.2.4 Combat

In figure 4.14 and 4.13, we see two players fighting. The combat is turn based, such that both opponents hit at the same rate. Damage dealt is calculated based on stats and equipment, but the formulas are not polished, and work poorly.



Figure 4.13: Player with a spear fights a player with a sword and shield.



Figure 4.14: Spearman deals 13 damage to a swordsman.

## 4.3 Tools developed to import assets and design the game

### 4.3.1 World editor

The world editor allows the user to modify the height map, place objects, and mark tiles as solid or water. In figure 4.15, we see a character object being selected. To the bottom left area, several properties can be edited, such as the transform, stats, and equipment.



Figure 4.15: Screenshot of the world editor.

### 4.3.2 Editor for node based scripting

The script editor defines a graphical interface for each script node type the game creates. On the left in figure 4.16, we can select which script to edit, see its identifier, and change the script name.

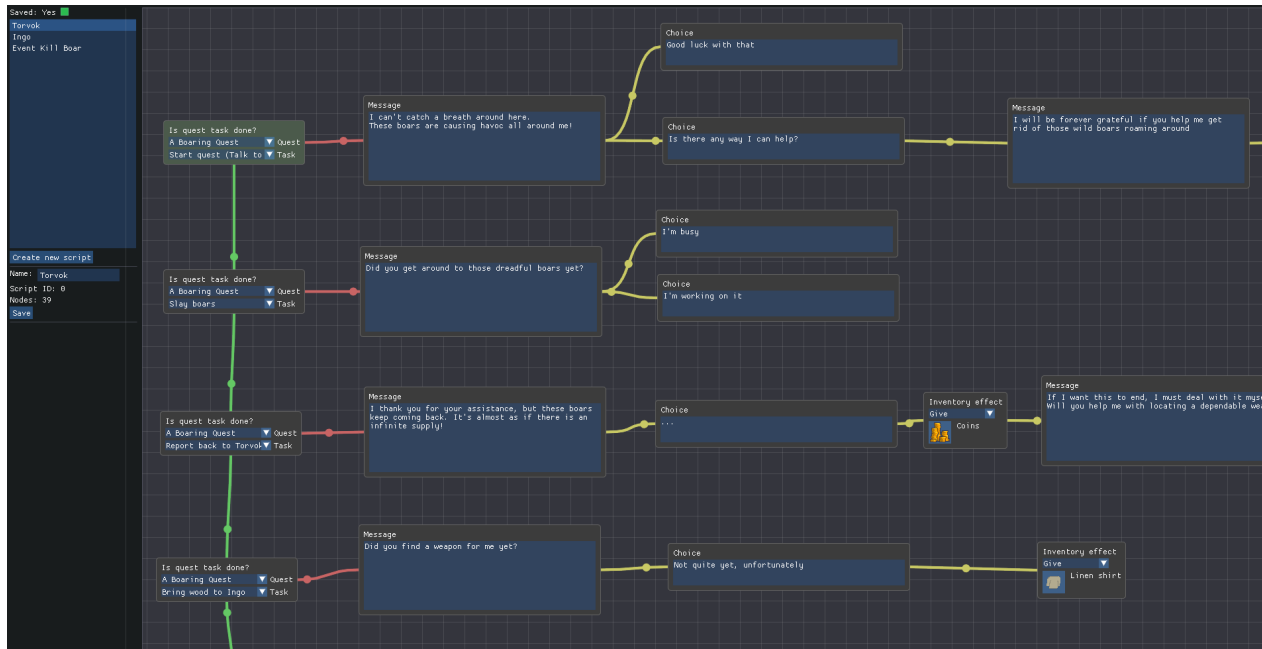


Figure 4.16: Screenshot of the script editor.

#### Entry point, and node links

The green node in figure 4.17 indicates that it is the entry point of the script. The nodes are connected by curved lines, where green and red symbolize true and false respectively. Each line has a moving circle on its path, which shows the direction the connection is flowing. A line is either an input or output.

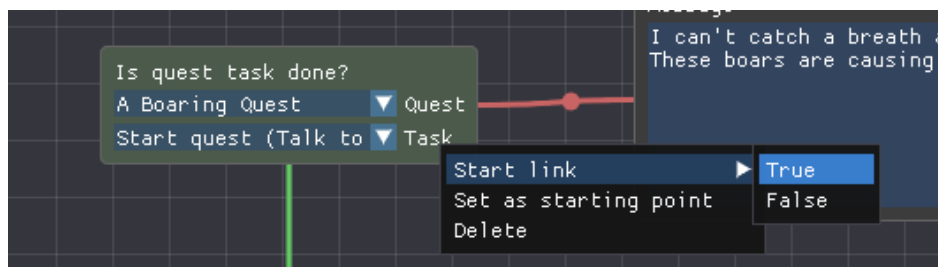


Figure 4.17: Context menu for a node in the script editor.

### Adding a new node to the script

When right clicking the background, a context menu appears like in 4.18. The nodes are categorized, and when an option is clicked, the node will be created at the position right clicked.

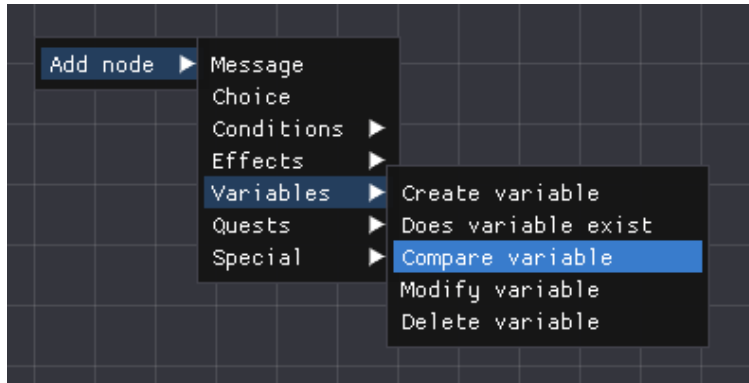


Figure 4.18: Context menu for adding a new node in the script editor.

### Interacting with a node

Each node display their own buttons and menus, as seen in figure 4.19. In this case, the item slot has been right clicked, and a context menu has opened. There is also a combo box, where you can select whether or not the node should give or take the item specified.

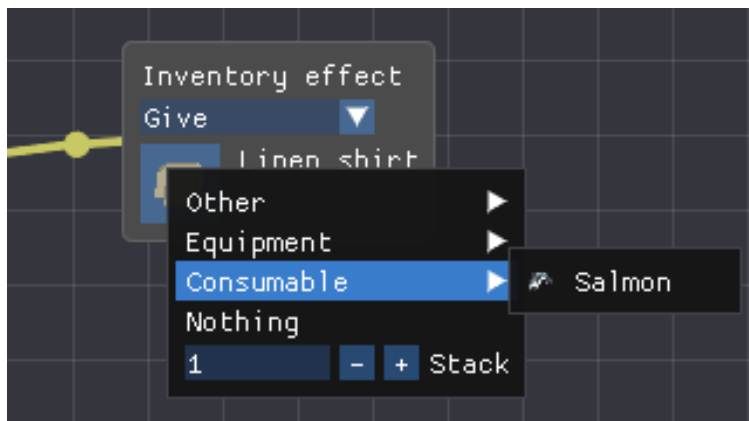


Figure 4.19: Context menu for the *give or take item* node in the script editor.

### 4.3.3 Quest editor

The quest editor, seen in figure 4.20, is a simple frontend for creating quest tasks and hints. The tasks can be given a name, marked as optional, and have a progress counter with a goal. The scripts can contain nodes that change the value of the progress counters, and whether or not a task is completed. On the right, hints for the quest can be written. The hints will read like a journal, and they are unlocked by a specified task.

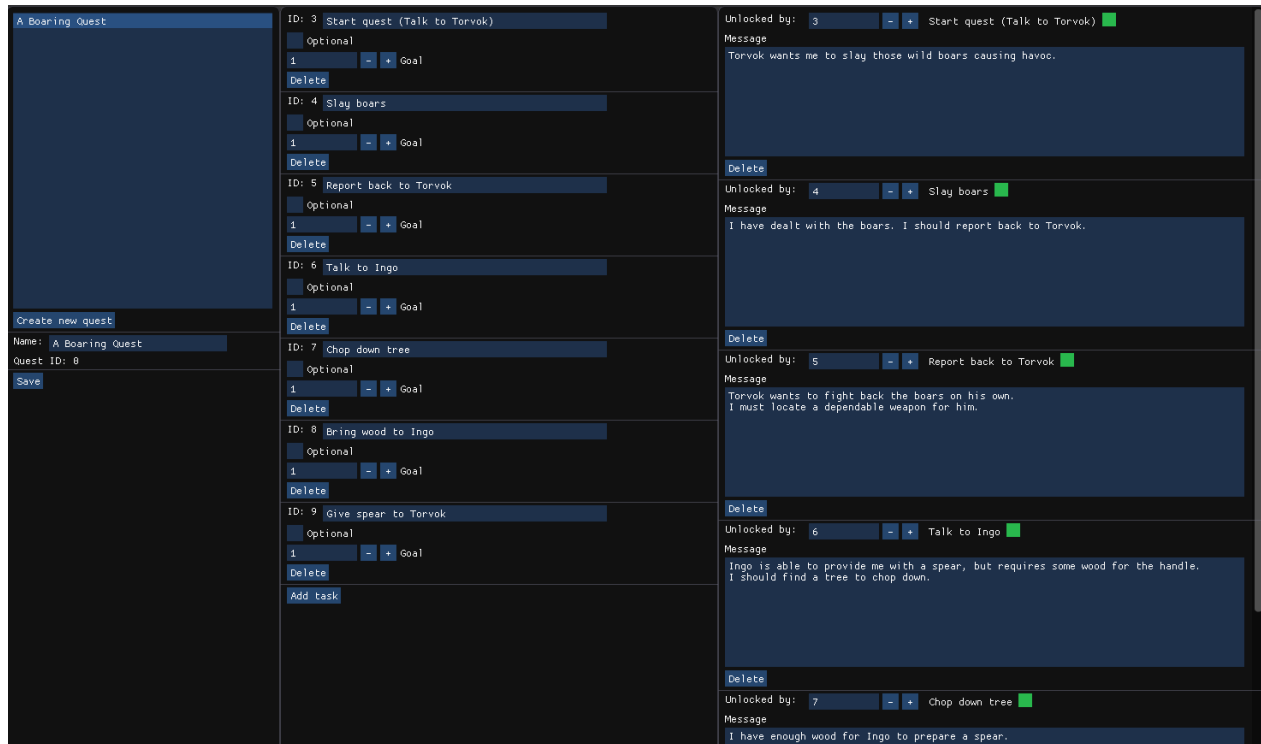


Figure 4.20: Screenshot of the quest editor.



### 4.3.4 Model manager

#### Conversion tool

The conversion tool, shown in figure 4.21, allows the user to convert models to the NOM format. The user can import COLLADA and OBJ files, with the specified target model type. The model types, *animated* and *static*, decide which vertex format is used.

For static models, the vertices may be scaled with the input field below the load button. This does not work for animated models, as only the vertices are modified.

The texture field lets the user bind a texture to the model. The value is relative to the assets' textures directory, and the PNG extension is implicit.

Further, we see a hierarchy of all the nodes in the model, and a list of all the animations.

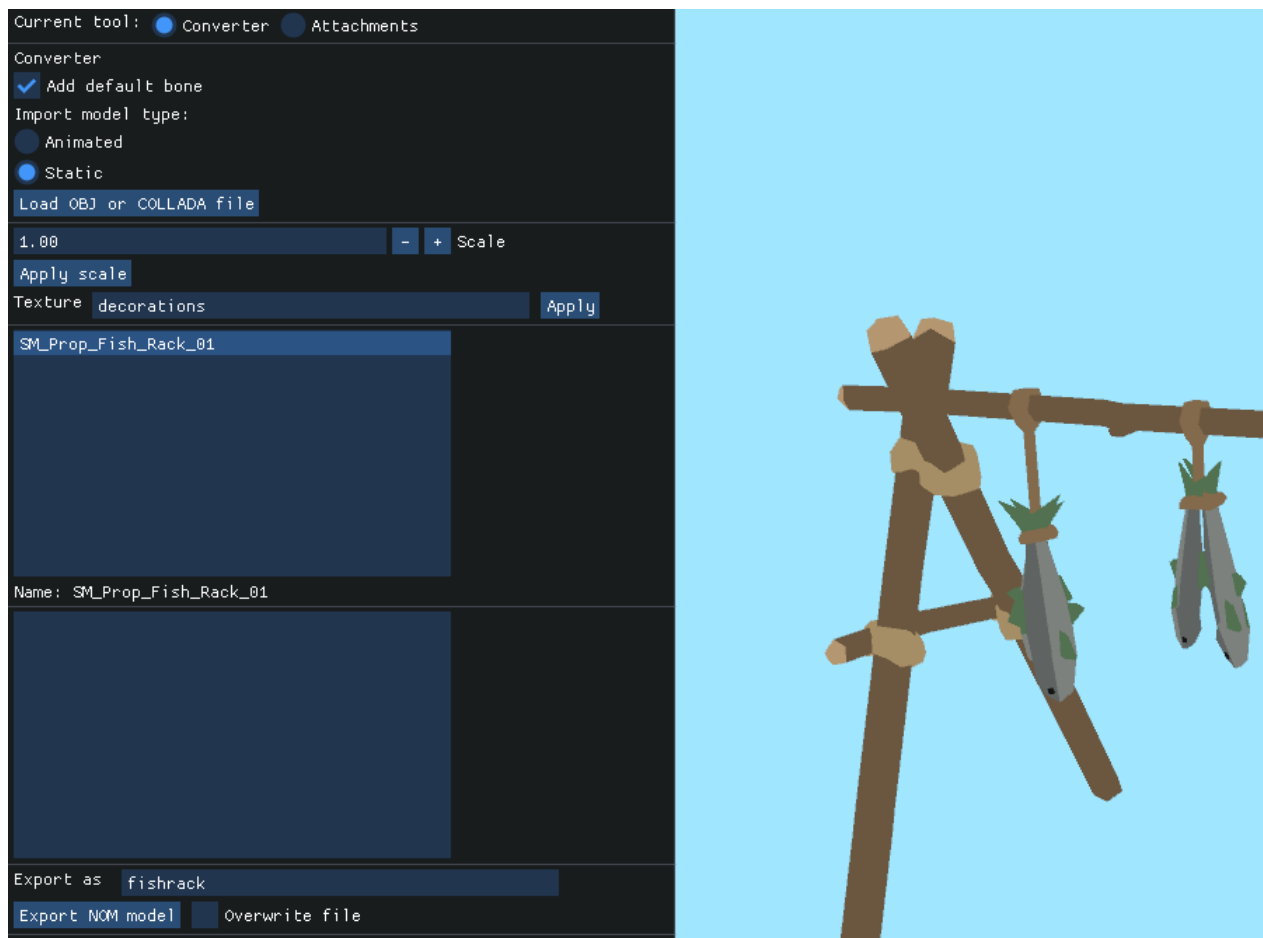


Figure 4.21: Screenshot of the model conversion tool.

### Attachment configuration tool

This tool is more of a workaround than a feature. Since there were some problems with the bones that Blender export, it was decided after some struggling that it's better to make a tool to edit the bone than to risk going behind schedule. As seen in figure 4.22, you can move and rotate the attachment bone to where it should be. All of the animations generally have the same mapping, but they may have a different one when necessary.

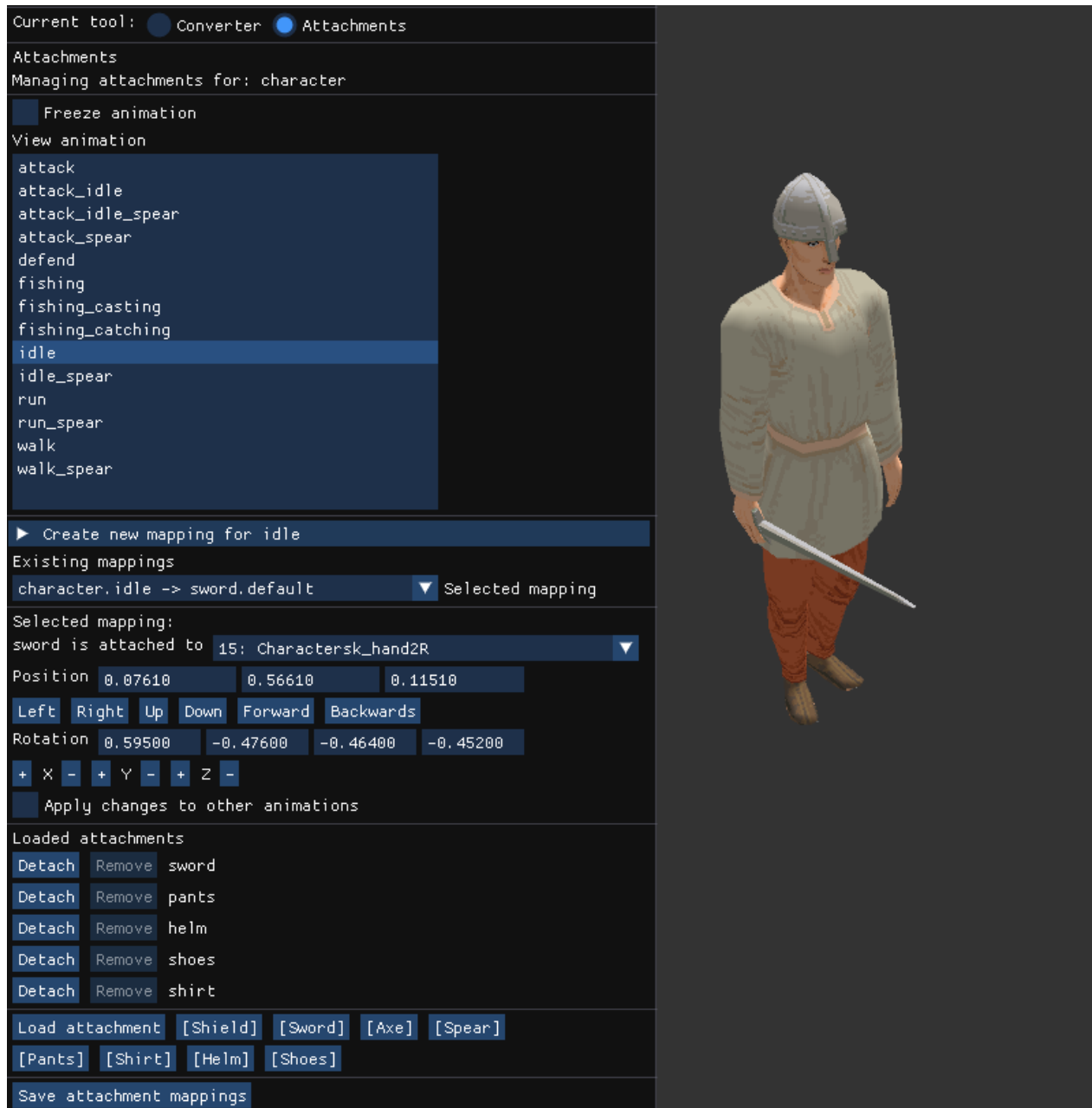


Figure 4.22: Screenshot of the model attachments editor.

### 4.3.5 Object editor

The object editor allows us to define new object definitions to be used in the game. As seen in figure 4.23, a list of values are configurable, based on the object type. The model specified in the model name field, is loaded and displayed to the right. There is also a bounding box adjuster, because you don't always want to use the minimum and maximum vertices to calculate bounding boxes. The script fields allow you to enter an identifier to a script, to be run during the event you link it to. The description contains the text that will appear when a player examines the object while playing.

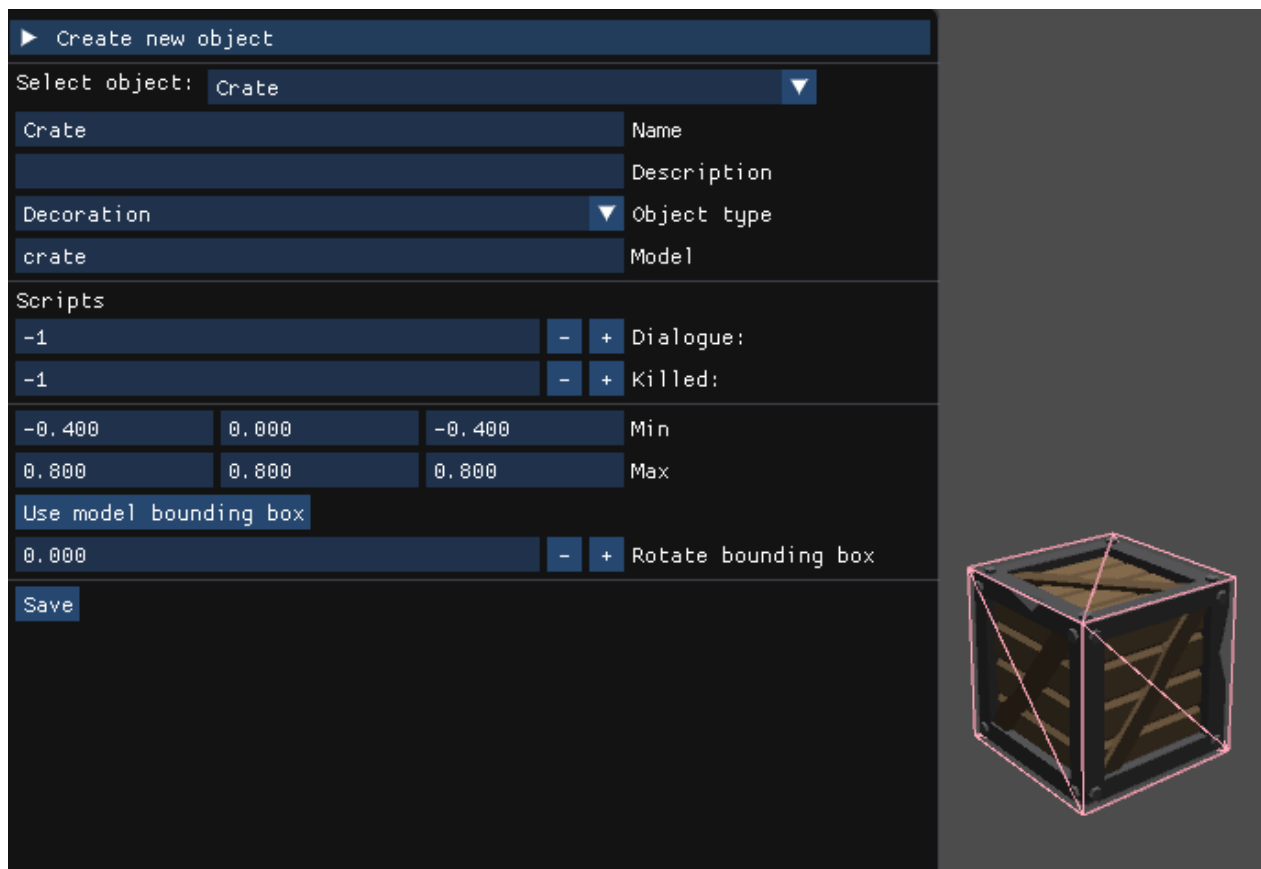


Figure 4.23: Screenshot of the object editor.

## 4.4 Performance profile

Figure 4.24 shows where the time spent in the main thread. As expected, the majority of the time is spent drawing to the screen. Further, we see that little time is spent rendering the user interfaces. However, a lot of time is spent drawing the world for picking. This is a consequence we discussed in chapter 2. If we finished the ray cast picking method, this would be an area where we could gain a lot of performance. For now, it is not a big issue.

To the right, we see the skeletal animations in the character renderer taking up most of the execution time.

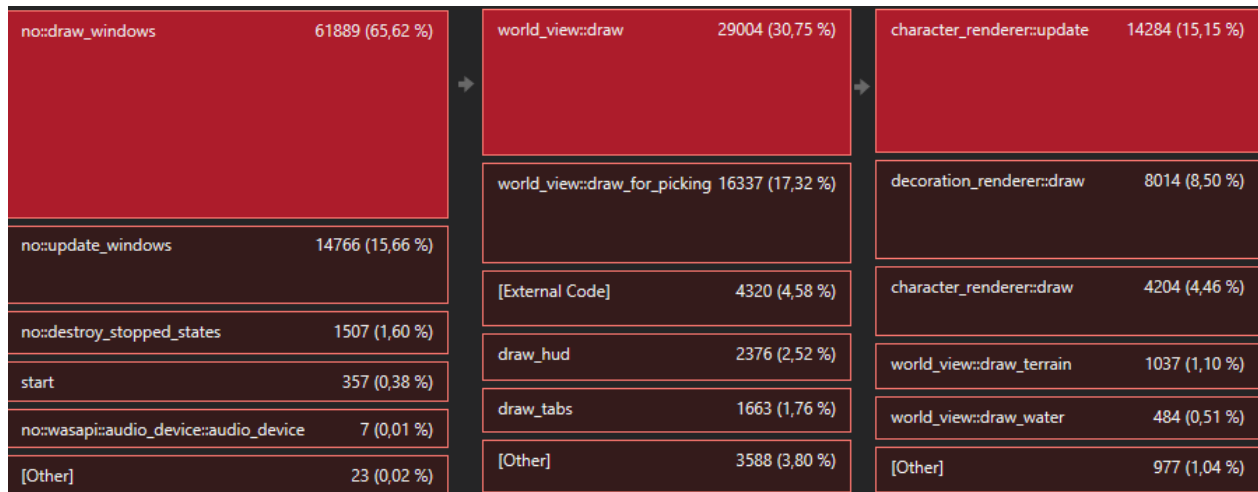


Figure 4.24: Performance profiling results.

# Chapter 5

## Discussion

### 5.1 Completeness compared to the requirements specification

The project has mostly been successful, and the primary objectives are satisfied. Lack of time resulted in some game features to not be implemented, such as ranged combat with bows. Content suffered a lot, and that was definitely the greatest deviation from the goals. While the content itself is not technical, it could have lead to the discovery of more bugs and missing features.

Since the framework is designed to be portable, it was desirable to have multiple implementations of at least one of the APIs. Unfortunately, each API only has one implementation as of the date of submission.

The code quality is not great in some places. Some features were rather rushed, and that affected the code quality. This happened mostly in the game projects, and especially in the toolkit code. On the other hand, the framework's network module is somewhat interesting. It minimizes memory allocations by using move semantics, has a simple interface, and uses function templates to accept data structures of any type that provides a `write` member function. It's possible to further decrease memory allocations, by having pre-allocated buffers.

The graphics module also has a few interesting parts, such as vertex attributes, and the non-convoluted wrapper of OpenGL.

## 5.2 Platform abstractions

Overall, the abstractions of platform APIs was done fairly well. The abstracted APIs aren't tested properly, as each only have one implementation. However, they are simplistic by design, and should be easily implementable anywhere. All of these APIs are optional as well, and a minimal build is possible by disabling all of them.

## 5.3 Models, skeletal animations, and attachments

Some issues arose due to a mix of Blender having poorly implemented model exporters, and Assimp having poorly implemented importers. The COLLADA exporter in Blender only exports the active animation, resulting in some irritating work that was not foreseen. There were also problems using the bone which Blender exported, to be used for attachments.

## 5.4 Technical game result

Almost all of the technical goals of the game were achieved. One minor change, a technical goal that was not achieved, is that the world was changed from non-chunked to chunked. The reason being it got too complex, and it became hard to come to a conclusion on the memory layout of the tile data. In hindsight, there was no good reason to attempt a non-chunked approach, but it sounded interesting at the time.

Other features like trading, chatting, minimap, quests, scripts, and so on were implemented. The scripting system is easily extendable, and can be used in any circumstance that makes sense. Security was also considered, and the client does not ever send anything besides input to the server.

## 5.5 Tools that were developed

Several tools were developed to aid with making content. As specified in the requirements, a model manager was created to import OBJ and COLLADA models, and allow the user to export them to the far superior, in terms of size and performance, NOM format. Inferior in interchangeability, of course. After exporting to NOM, the attachment tool can be used to specify how a model can be attached to another.

## 5.6 Move scripting functionality to the framework

The scripting feature was added as part of the game itself. However, the interface could very well have been integrated with the framework. This could then also include the game variable system, as it ties in well.

## 5.7 Proper asset management

The game did not have very many models, textures, or audio files to load. Therefore, there was no incentive to spend time making a proper asset management module. At the very least, the game does have shared assets, but they are always loaded in memory for the duration of the entire execution. In the future of the framework, the assets should be reference counted, so they may be unloaded when no longer used.

## 5.8 More graphical features

The requirements did not specify any graphical features, and intentionally so. Time had to be allocated for the more foundational features, and thus realistic rendering was not prioritised. Among others, features like shadow mapping and ambient occlusion are important for realistic visuals.

# Chapter 6

## Conclusion

The purpose of this project was to develop a cross-platform framework for graphical applications. To ensure the framework's usability, we also had to create some applications that use it. We chose to develop a game, consisting of a client, server, and a toolkit. Multiple areas of software development had to be delved into, within graphics, audio, network, and concurrency.

We have learned a lot about 3D graphics and skeletal animation. However, computer graphics is a vast field, and there is so much more that could have been part of this project. Normal mapping, shadow mapping, and ambient occlusion, are necessary features in most game these days, but were not part of the implementation.

For efficiency, we ran skeletal animations in parallel to the main thread. The number of matrix operations could likely have been reduced, but the performance is satisfactory. The attachments could have used a bit more refining, but we did get a working result.

Basic audio playback was not too complicated, but we lack 3D positioned audio, as noted in the requirements specification. The game uses a third person camera, so it was unnecessary to implement. It is, nevertheless, a feature many 3D games use.

To summarize, we have a framework capable of 3D graphics, TCP networking, and audio. The framework makes these features easily accessible to the end user, the application developers. To prove its usability, we developed an online 3D game, along with a set of tools to create the game world, manage models, edit scripts, and more.



# References

- [1] AngryAnt. Is Unity Engine written in Mono/C#? or C++. <https://answers.unity.com/questions/9675/is-unity-engine-written-in-monoc-or-c.html>, accessed 2019-03-20.
- [2] Anton Gerdelan. Mouse Picking with Ray Casting. <http://antongerdelan.net/opengl/raycasting.html>, accessed 2019-05-04.
- [3] Arch Robison. Why Too Many Threads Hurts Performance, and What to do About It. [https://www.codeguru.com/cpp/sample\\_chapter/article.php/c13533/Why-Too-Many-Threads-Hurts-Performance-and-What-to-do-About-It.htm](https://www.codeguru.com/cpp/sample_chapter/article.php/c13533/Why-Too-Many-Threads-Hurts-Performance-and-What-to-do-About-It.htm), accessed 2019-05-20.
- [4] Blender. Collada. <https://docs.blender.org/manual/en/latest/pipeline/collada.html>, accessed 2019-05-18.
- [5] Casey Muratori. Immediate-Mode Graphical User Interfaces. [https://caseymuratori.com/blog\\_0001](https://caseymuratori.com/blog_0001), accessed 2019-05-19.
- [6] Catalin Dan Udma. Context Switch. <https://www.sciencedirect.com/topics/engineering/context-switch>, accessed 2019-05-20.
- [7] Computer Graphics Laboratory. MipMap Texturing. [https://graphics.ethz.ch/teaching/former/vc\\_master\\_06/Downloads/Mipmaps\\_1.pdf](https://graphics.ethz.ch/teaching/former/vc_master_06/Downloads/Mipmaps_1.pdf), accessed 2019-03-20.
- [8] Omar Cornut. Dear ImGui. <https://github.com/ocornut/imgui>, accessed 2019-01-30.
- [9] cplusplus.com. Preprocessor directives. <http://www.cplusplus.com/doc/tutorial/preprocessor/>, accessed 2019-04-08.

- [10] cppreference.com. Preprocessor. <https://en.cppreference.com/w/cpp/preprocessor>, accessed 2019-04-08.
- [11] cppreference.com. RAII. <https://en.cppreference.com/w/cpp/language/raii>, accessed 2019-04-08.
- [12] cppreference.com. Experimental C++ Features. <https://en.cppreference.com/w/cpp/experimental>, accessed 2019-04-28.
- [13] CRYTEK. CRYENGINE. <https://github.com/CRYTEK/CRYENGINE>, accessed 2019-03-20.
- [14] Dan Kegel. The C10K problem. <http://www.kegel.com/c10k.html>, accessed 2019-05-19.
- [15] Epic Games. Unreal Engine FAQ. <https://www.unrealengine.com/en-US/faq>, accessed 2019-03-20.
- [16] Eric W. Weisstein. Normal Vector. <http://mathworld.wolfram.com/NormalVector.html>, accessed 2019-05-17.
- [17] Etay Meiri. 3D Picking. <http://ogldev.atspace.co.uk/www/tutorial29/tutorial29.html>, accessed 2019-05-04.
- [18] Etay Meiri. Skeletal Animation With Assimp. <http://ogldev.atspace.co.uk/www/tutorial38/tutorial38.html>, accessed 2019-05-18.
- [19] Glenn Fiedler. Fix Your Timestep! [https://gafferongames.com/post/fix\\_your\\_timestep](https://gafferongames.com/post/fix_your_timestep), June 10th, 2004 (accessed 2019-01-30).
- [20] FreeType. FreeType. <https://www.freetype.org/>, accessed 2019-01-30.
- [21] Epic Games. Unreal Engine. <https://www.unrealengine.com>, accessed 2019-01-28.
- [22] Laurent Gomila. SFML. <https://www.sfml-dev.org>, accessed 2019-01-28.
- [23] Gregery Barton. Blender collada export multiple animations. <https://stackoverflow.com/questions/42386188/blender-collada-export-multiple-animations>, accessed 2019-05-18.

- [24] Jeremiah van Oosten. Understanding the View Matrix. <https://www.3dgep.com/understanding-the-view-matrix/>, accessed 2019-05-03.
- [25] Jim Frost. BSD Sockets: A Quick And Dirty Primer. <https://cis.temple.edu/~giorgio/old/cis307s96/readings/docs/sockets.html>, accessed 2019-04-28.
- [26] Joey de Vries. Getting started: OpenGL. <https://learnopengl.com/Getting-started/OpenGL>, accessed 2019-04-28.
- [27] Joey de Vries. Coordinate Systems. <https://learnopengl.com/Getting-started/Coordinate-Systems>, accessed 2019-05-03.
- [28] Joey de Vries. Normal Mapping. <https://learnopengl.com/Advanced-Lighting/Normal-Mapping>, accessed 2019-05-17.
- [29] Joey de Vries. Anti Aliasing. <https://learnopengl.com/Advanced-OpenGL/Anti-Aliasing>, accessed 2019-05-18.
- [30] John Kennedy, Michael Satran. Porting Socket Applications to Winsock. <https://docs.microsoft.com/nb-no/windows/desktop/WinSock/porting-socket-applications-to-winsoc>, accessed 2019-04-28.
- [31] John Kennedy, Michael Satran. Overlapped I/O and Event Objects. <https://docs.microsoft.com/nb-no/windows/desktop/WinSock/overlapped-i-o-and-event-objects-2>, accessed 2019-04-28.
- [32] John Kennedy, Michael Satran. Rendering a Stream. <https://docs.microsoft.com/en-us/windows/desktop/coreaudio/rendering-a-stream>, accessed 2019-05-16.
- [33] John Kennedy, Michael Satran. Face and Vertex Normal Vectors. <https://docs.microsoft.com/en-us/windows/desktop/direct3d9/face-and-vertex-normal-vectors>, accessed 2019-05-17.
- [34] John Kennedy, Michael Satran. Using Mutex Objects. <https://docs.microsoft.com/en-us/windows/desktop/sync/using-mutex-objects>, accessed 2019-05-20.

- [35] Jonathan Wakely. Working Draft, C++ Extensions for Networking. <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4771.pdf>, accessed 2019-04-28.
- [36] Ken Power. Perspective Projection. [http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection/perspective\\_projection.html](http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection/perspective_projection.html), accessed 2019-03-25.
- [37] Sam Oscar Lantinga. SDL. <https://www.libsdl.org>, accessed 2019-01-28.
- [38] Michael Tyson. Four common mistakes in audio development. <http://atastypixel.com/blog/four-common-mistakes-in-audio-development/>, accessed 2019-05-16.
- [39] Microsoft. Texture Filtering with Mipmaps. [https://docs.microsoft.com/en-us/previous-versions/aa921432\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/aa921432(v=msdn.10)), accessed 2019-03-20.
- [40] Microsoft. AcceptEx function. <https://docs.microsoft.com/en-us/windows/desktop/api/mswsock/nf-mswsock-acceptex>, accessed 2019-04-28.
- [41] Microsoft. Description of race conditions and deadlocks. <https://support.microsoft.com/en-ca/help/317723/description-of-race-conditions-and-deadlocks>, accessed 2019-05-20.
- [42] OGRE. About OGRE. <https://www.ogre3d.org/about>, accessed 2019-03-20.
- [43] Oracle. Java Socket Class. <https://docs.oracle.com/javase/10/docs/api/java/net/Socket.html>, accessed 2019-04-28.
- [44] Oracle. What Is a Relational Database? <https://www.oracle.com/database/what-is-a-relational-database/>, accessed 2019-05-04.
- [45] PHP. password\_hash. <https://www.php.net/manual/en/function.password-hash.php>, accessed 2019-05-17.
- [46] PostgreSQL. CREATE PROCEDURE. <https://www.postgresql.org/docs/current/sql-createprocedure.html>, accessed 2019-05-17.

- [47] PostgreSQL Wiki. UPSERT. <https://wiki.postgresql.org/wiki/UPSERT>, accessed 2019-05-17.
- [48] Rasmus Lerdorf. PHP in 2018. <https://youtu.be/rKXFgWP-2xQ?t=269>, accessed 2019-04-05.
- [49] Robert Nystrom. Game Loop. <https://gameprogrammingpatterns.com/game-loop.html>, accessed 2019-05-19.
- [50] Sonoma. Pulse Code Modulation. [https://web.sonoma.edu/esee/courses/ee442/archives/sp2019/lectures/lecture09\\_pcm.pdf](https://web.sonoma.edu/esee/courses/ee442/archives/sp2019/lectures/lecture09_pcm.pdf), accessed 2019-05-16.
- [51] Bjarne Stroustrup. When was C++ invented? [http://www.stroustrup.com/bs\\_faq.html#invention](http://www.stroustrup.com/bs_faq.html#invention), accessed 2019-03-20.
- [52] Unity Technologies. Unity. <https://unity3d.com>, accessed 2019-01-28.
- [53] The Khronos Group. OpenGL API Documentation Overview. <https://www.khronos.org/documentation/>, accessed 2019-04-08.
- [54] The Khronos Group. Vertex Specification Best Practices. [https://www.khronos.org/opengl/wiki/Vertex\\_Specification\\_Best\\_Practices](https://www.khronos.org/opengl/wiki/Vertex_Specification_Best_Practices), accessed 2019-05-03.
- [55] The Khronos Group. Buffer Object. [https://www.khronos.org/opengl/wiki/Buffer\\_Object](https://www.khronos.org/opengl/wiki/Buffer_Object), accessed 2019-05-03.
- [56] The Khronos Group. Fragment Shader. [https://www.khronos.org/opengl/wiki/Fragment\\_Shader](https://www.khronos.org/opengl/wiki/Fragment_Shader), accessed 2019-05-03.
- [57] The Khronos Group. Vertex Post-Processing. [https://www.khronos.org/opengl/wiki/Vertex\\_Post-Processing](https://www.khronos.org/opengl/wiki/Vertex_Post-Processing), accessed 2019-05-03.
- [58] The Khronos Group. Vertex Shader. [https://www.khronos.org/opengl/wiki/Vertex\\_Shader](https://www.khronos.org/opengl/wiki/Vertex_Shader), accessed 2019-05-03.
- [59] The Khronos Group. Vertex Specification. [https://www.khronos.org/opengl/wiki/Vertex\\_Specification](https://www.khronos.org/opengl/wiki/Vertex_Specification), accessed 2019-05-03.

- [60] The Khronos Group. Legacy OpenGL. [https://www.khronos.org/opengl/wiki/Legacy\\_OpenGL](https://www.khronos.org/opengl/wiki/Legacy_OpenGL), accessed 2019-05-04.
- [61] The Khronos Group. glTexParameter. <https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glTexParameter.xhtml>, accessed 2019-05-17.
- [62] The Khronos Group. Skeletal Animation. [https://www.khronos.org/opengl/wiki/Skeletal\\_Animation](https://www.khronos.org/opengl/wiki/Skeletal_Animation), accessed 2019-05-18.
- [63] The PHP Group. What is PHP? <https://www.php.net/manual/en/intro-what-is.php>, accessed 2019-04-05.
- [64] The Unicode Consortium. Unicode® 12.1.0. <https://www.unicode.org/versions/Unicode12.1.0/>, accessed 2019-05-18.
- [65] TreyK, Journeyman Geek. Why do some old games run much too quickly on modern hardware? <https://superuser.com/questions/630769/why-do-some-old-games-run-much-too-quickly-on-modern-hardware/631131>, accessed 2019-05-19.
- [66] TutorialsPoint. SQL - Overview. <https://www.tutorialspoint.com/sql/sql-overview.htm>, accessed 2019-05-04.
- [67] University of Southern California. RFC: 675 Specification of Internet Transmission Control Program. <https://tools.ietf.org/html/rfc675>, accessed 2019-04-28.
- [68] University of Washington. Atomic Operations in Hardware. <https://courses.cs.washington.edu/courses/cse378/07au/lectures/L25-Atomic-Operations.pdf>, accessed 2019-05-20.
- [69] Warren Young. Which I/O Strategy Should I Use? <https://tangentsoft.net/wskfaq/articles/io-strategies.html>, accessed 2019-04-28.
- [70] Wikipedia. Berkeley sockets. [https://en.wikipedia.org/wiki/Berkeley\\_sockets](https://en.wikipedia.org/wiki/Berkeley_sockets), accessed 2019-04-28.
- [71] Wikipedia. Operating system abstraction layer. [https://en.wikipedia.org/wiki/Operating\\_system\\_abstraction\\_layer](https://en.wikipedia.org/wiki/Operating_system_abstraction_layer), accessed 2019-05-18.

- [72] Koen Witters. deWiTTERS Game Loop. <http://www.koonsolo.com/news/dewitters-gameloop>, July 13th, 2009 (accessed 2019-01-30).

# **Appendices**



# **Appendix A**

## **Preliminary report**

# Preliminary report



# NTNU

Kunnskap for en bedre verden

TITLE Einheri – building an online 3D game with a custom framework			
CANDIDATE NUMBER 997517			
DATE 31.01.2019	SUBJECT CODE IE303612	SUBJECT Bachelor project	DOCUMENT ACCESS Open
STUDY Computer science		PAGES / ATTACHMENTS 17 / 0	BIBL. NUM. Not used
ADVISOR Saleh Abdel-Afou Alaliyat			

## SUMMARY:

The title of this project, Einheri, is the name of a game that will be developed alongside a new framework. The framework will be made for making general purpose graphical and networked applications. Existing frameworks and engines such as SDL [17], SFML [11], Unreal Engine [8] and Unity [30] are already available, but this framework will be developed with a different approach. It's worth noting the objective is not to replace or reach the same feature set as the aforementioned tools. The goals for the framework are flexibility, low overall coupling of systems, and a small memory footprint.

To test the framework, an online 3D game consisting of a client, server, and toolkit, will be developed with it. To ensure playability and quality of the game, a group of testers will help by providing fortnightly feedback. Players will explore a world with stories and characters based on Norse mythology and the viking age. As a player, one can help certain in-game characters with tasks that range from easy to hard difficulty levels.

---

*This assignment is an exam submission done by a student at NTNU in Ålesund.*

---

**Postal address**  
NTNU i Ålesund  
N-6025 Ålesund  
Norway

**Visitation address**  
Larsgårdsvegen 2  
**Internet**  
ntnu.no

**Phone**  
73 59 50 00  
**E-mail**  
postmottak@alesund.ntnu.no

**Bank account**  
7694 05 00636  
**Foretaksregisteret**  
NO 974 767 880

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Terminology</b>	<b>2</b>
<b>3</b>	<b>Project organization</b>	<b>3</b>
3.1	Group . . . . .	3
3.2	Organizational tasks . . . . .	3
3.3	Management . . . . .	3
<b>4</b>	<b>Mindset</b>	<b>4</b>
<b>5</b>	<b>Project description</b>	<b>5</b>
5.1	Objective . . . . .	5
5.2	Requirements specification . . . . .	5
5.3	Methodology . . . . .	8
5.4	Information gathering . . . . .	8
5.5	Risk analyzis . . . . .	8
5.6	Primary activities in further work . . . . .	8
5.7	Progress plan . . . . .	8
5.8	Development tools . . . . .	10
5.9	Internal control and evaluation . . . . .	10
<b>6</b>	<b>Documentation</b>	<b>11</b>
6.1	Reports and technical documents . . . . .	11

<i>CONTENTS</i>	iii
<b>7 Planned meetings and reports</b>	<b>12</b>
7.1 Meetings . . . . .	12
7.2 Progress reports . . . . .	13
<b>8 Planned deviation management</b>	<b>14</b>
<b>References</b>	<b>15</b>

# Chapter 1

## Introduction

Einheri is the name of an online multiplayer 3D game that will be developed for this project. The game will be based on Norse mythology, and the viking age. Players will be able to interact with an open world, and explore the stories it has to tell. Developing content is a key part of game development. Shaping the world, editing the character dialogues, and defining objects and items. These jobs will be done using a toolkit developed specifically for this game. The toolkit will provide a node based scripting system for defining character dialogues and item behavior. All the components for the game will be built using a framework that will be developed for this project. The framework must implement systems for networking, skeletal animations, rendering, audio, input, and so forth. Typically, graphical software use existing frameworks or engines, but the goal for this project is to build something new.

A website will also be created, which will let new users sign up, download the client, and view leaderboards. Other features include a news section, and a world map.

# Chapter 2

## Terminology

**C++** is a multi-paradigm programming language developed by Bjarne Stroustrup

**Unreal Engine** is an industry standard open-source game engine by Epic Games

**Unity** is a game engine popular among independent developers by Unity Technologies

**ImGui** *Dear ImGui* is a flexible and customizable platform independent C++ library for quickly implementing user interfaces

**SDL** Simple DirectMedia Layer is a framework for graphical applications in C and C++

**SFML** Simple and Fast Multimedia Library is a framework for graphical applications in C++

**TCP** Transmission Control Protocol is a networking protocol with a no packet loss guarantee

**API** Application Programming Interface

**WASAPI** Windows Audio Sessions API for streaming audio in Windows

**A\*** A-star is a popular algorithm used for pathfinding

**TTF** TrueType Fonts

**UML** Unified Modeling Language

**MSDN** Microsoft Developer Network

# Chapter 3

## Project organization

### 3.1 Group

Student number	Name of student
997517	Sebastian Søviknes Gundersen

*Table 1: Students in the group*

### 3.2 Organizational tasks

Tasks must be completed according to plan. Progress reports need to be handed in every two weeks, and feedback must be compiled from testers. In the case of deviancy from the project plan, it must be measured. Actions must then be taken based on the planned deviation management.

### 3.3 Management

The management consists of Saleh Abdel-Afou Alaliyat at NTNU in Ålesund.

# Chapter 4

## Mindset

As a software developer, it is important to develop ethical software, and respect the user. Contemporary software is heavily focused on violating privacy [26] and monetizing personal information [18]. Every new project involving personal information, should also be based on the guidelines defined by Datatilsynet [3].

There is also the eery trend of *Software as a Service* (SaaS) that is being pushed by many companies, such as Microsoft [19]. It is unfortunate that people are no longer allowed to own the software they pay for [25]. In the future, even operating systems may have similar restrictions if things continue in this direction. There are alternatives such as GNU/Linux, but most consumers do not install a distribution of Linux when purchasing a computer. SaaS can be compared to arguably evil hardware manufacturers against the right to repair, such as Apple [15]. The free software community is one that more companies and software developers should look towards.



# Chapter 5

## Project description

### 5.1 Objective

There are two distinct goals for this project. One is to develop a framework with the required features specified in the next section. The other is to create an interesting and playable game based on that framework, along with a toolkit to produce content for the game.

### 5.2 Requirements specification

Both the framework and the game core will be compiled uniquely for the client, server, and toolkit. The client does not need model conversion and ImGui, while the server does not need any of the graphics or audio systems at all. With this in mind, all game logic must be completely separated from the rendering. The scope of the project is limited by the requirements specified below.

Requirements for the framework:

- Skeletal animation system with attachments support
- Multithreaded networking system with a packet protocol based on TCP
- Audio system based on WASAPI [21] – 3D sounds will not be implemented
- Logging system with detailed and readable output with easy usage
- Loading common asset files like PNG [27] and Ogg Vorbis [33][34]

- Import models using Assimp [16], then convert to a custom model format
- Flexible rendering system, easily allowing custom defined vertex layouts and shaders
- TTF [1][20] unicode text rendering with FreeType [7]
- ImGui [2] platform implementation to allow the toolkit to use ImGui
- Capable of being used without including third party libraries (besides GLM [10] and ImGui)
- Proper synchronization between frames, with a take on fixed time steps [5][32]
- Ortho and perspective cameras, with controllers
- I/O stream to easily write and read memory

### Requirements for the game core:

- Streamable game world without chunking
- Dialogue system with conditions, events, and variables
- Item system with similar behavior to dialogues
- Quest system, on top of the dialogue system
- Combat system with support for both melee and ranged attacks
- Inventory and equipment slots for the player
- Chat system to let players communicate
- Fishing to gather food for use in combat to heal lifepoints
- Stamina that affects combat and agility
- Pathfinding with A\* [24]
- Minimap
- Area music, and sound effects for actions
- Warehouses
- Player trading

### Requirements for content produced in toolkit:

- Minimum a dozen quests for the player to complete
- At least double as many important game characters as there are quests
- The world must be explorable, meaning it should take several minutes to cross it
- Various types of swords, spears, axes and bows must be available
- Unfriendly characters, animals or monsters to be killed in combat for rewards or progress in a quest

### Requirements for the toolkit:

- Import models in exchange formats [31] and convert them to a custom format
- Tool to easily view and configure attachments for models
- Edit the height map for the world, and place tile textures
- Place objects in the world, and edit their properties
- Configure items and objects that are used in the game
- Node based editor for the dialogue trees, providing intuitive visualization

### Requirements for the client:

- Update the game world according to packets received from the server
- Send player input to the server
- User interface showing inventory, equipment, skills and quests
- Must be able to automatically auto-update itself when there is a new update released

### Requirements for the server:

- Receive input from the client
- Validate the actions of players
- Update the game world based on validated actions, and broadcast to necessary players
- Synchronize player data with the database

### Requirements for the website:

- Visitors should be able to register a new account to be used in the game
- A highscore table indicating who ranks best in certain aspects of the game
- Ability to change the password of an account, or perform a password reset
- Link to download the game client
- Link to view a map of the game world
- List of feedback polls voted on by the testers
- A news section that summarizes the latest updates to the game

## 5.3 Methodology

Although this is an individual project, an agile approach will be attempted, with focus on iterative development. Start out with the minimum viable product, and decide further development from that. There is a plan to follow, but continuous feedback from the testers will be considered.

## 5.4 Information gathering

There are many websites with useful information, including articles, scattered throughout the web. MSDN will be important to develop for Windows. Reading the documentation for other libraries and APIs used will also be necessary. Sites such as Stack Overflow and GameDev.net have a lot of questions and answers in almost any topic.

## 5.5 Risk analyzis

The requirements are somewhat designed to minimize risks, but there is always a possibility of something going wrong. The most important thing early on, is to focus on the framework and game core. The content is very important for a playable game, but it also needs to have the necessary features implemented. Other things that can go wrong include the server not performing enough validation checks, and thus crashing. Some computers, such as low-end laptops, might not be able to run the game at a decent frame rate of 60.

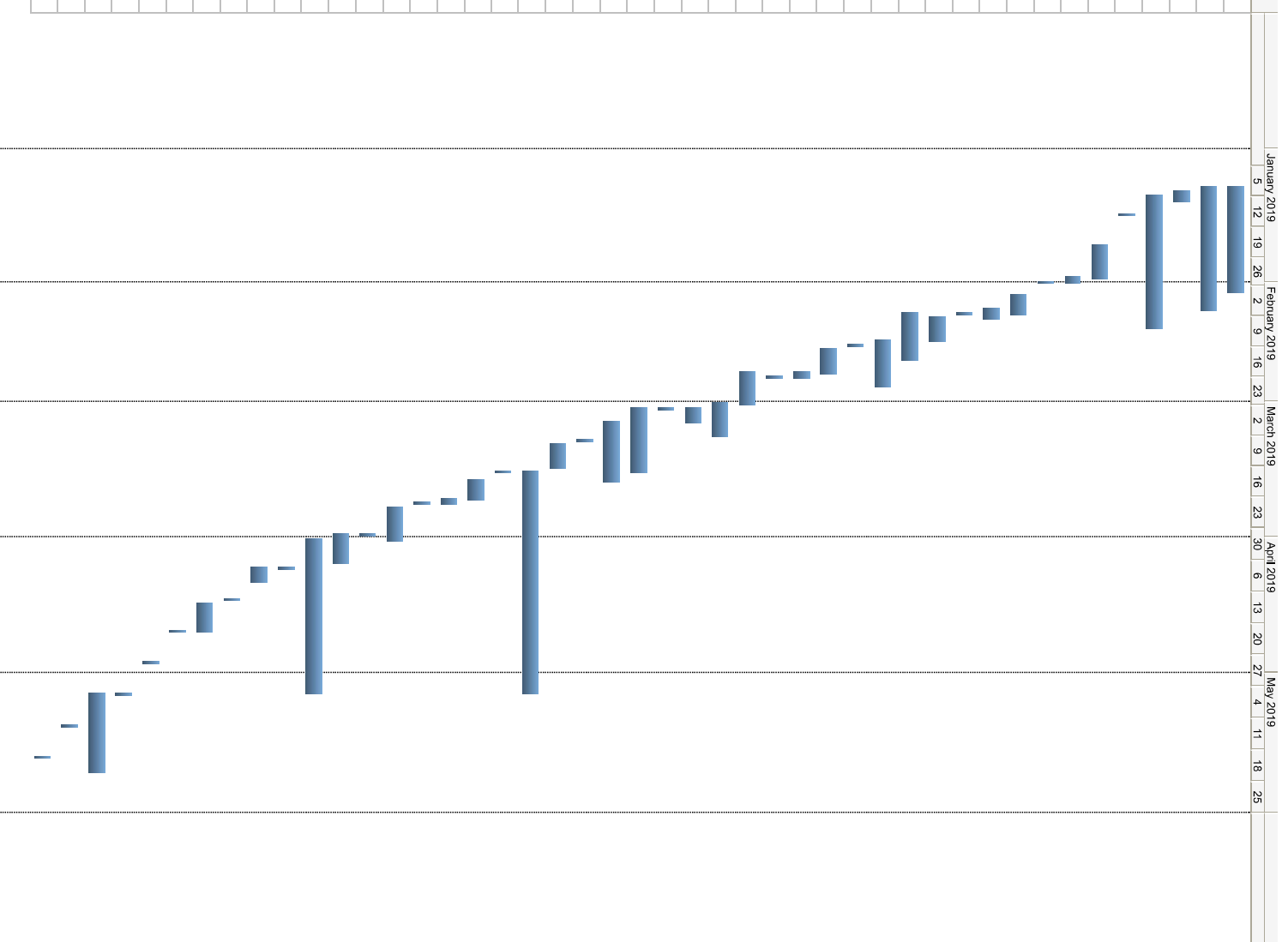
## 5.6 Primary activities in further work

The framework is in a functioning state, thus most of the work will be to continue on the game and toolkit. Additionally, it is important to add relevant content such as an interactive world with stories to tell.

## 5.7 Progress plan

See next page.

	Name	Duration	Start	Finish
1	Preliminary report	18days?	01/08/2019	01/13/2019
2	Framework: Implement all features according to requirements	20days?	01/08/2019	02/04/2019
3	Game: Players connecting to server and moving around	3days?	01/09/2019	01/11/2019
4	Toolkit: Create the world editor	22days?	01/10/2019	02/08/2019
5	Meeting #1	1day?	01/14/2019	01/14/2019
6	Toolkit: Create model manager for conversion and attachments	6days?	01/21/2019	01/28/2019
7	Toolkit: Create item and object configuration tools	2days?	01/28/2019	01/29/2019
8	Meeting #2	1day?	01/29/2019	01/29/2019
9	Website: Be able to register and download the game	3days?	02/01/2019	02/05/2019
10	Game: Inventory and equipment	3days?	02/04/2019	02/06/2019
11	Feedback poll #1	1day?	02/05/2019	02/05/2019
12	Client: Auto-update feature	4days?	02/06/2019	02/11/2019
13	Toolkit: Create the node based dialogue editor	9days?	02/05/2019	02/15/2019
14	Game: Dialogue interface	9days?	02/11/2019	02/21/2019
15	Meeting #3	1day?	02/12/2019	02/12/2019
16	Game: Chat system	4days?	02/13/2019	02/18/2019
17	Website: News section	2days?	02/18/2019	02/19/2019
18	Feedback poll #2	1day?	02/19/2019	02/19/2019
19	Server: Persistency	6days?	02/18/2019	02/25/2019
20	Game: Quest system	6days?	02/25/2019	03/04/2019
21	Game: Pathfinding	4days?	02/26/2019	03/01/2019
22	Meeting #4	1day?	02/26/2019	02/28/2019
23	Content: Draft of the world map	11days?	02/26/2019	03/12/2019
24	Game: Combat system	10days?	03/01/2019	03/14/2019
25	Feedback poll #3	1day?	03/05/2019	03/05/2019
26	Game: Minimap	4days?	03/06/2019	03/11/2019
27	Content: Quests	36days?	03/12/2019	04/30/2019
28	Meeting #5	1day?	03/12/2019	03/12/2019
29	Game: Players can trade eachother	3days?	03/14/2019	03/18/2019
30	Game and toolkit: Implement area background music, and sound effects	2days?	03/18/2019	03/19/2019
31	Feedback poll #4	1day?	03/19/2019	03/19/2019
32	Game: Fishing	6days?	03/20/2019	03/27/2019
33	Meeting #6	1day?	03/26/2019	03/26/2019
34	Game: Warehouses	5days?	03/26/2019	04/01/2019
35	Content: Make music and sound effects	25days?	03/27/2019	04/30/2019
36	Feedback poll #5	1day?	04/02/2019	04/02/2019
37	Website: View highscores and a map of the game world	4days?	04/02/2019	04/05/2019
38	Meeting #7	1day?	04/09/2019	04/09/2019
39	Content: Decorate world with more details	5days?	04/10/2019	04/16/2019
40	Feedback poll #6	1day?	04/16/2019	04/16/2019
41	Meeting #8	1day?	04/23/2019	04/23/2019
42	Feedback poll #7	1day?	04/30/2019	04/30/2019
43	Bugfixing and polishing	14days?	04/30/2019	05/17/2019
44	Meeting #9	1day?	05/07/2019	05/07/2019
45	Feedback poll #8	1day?	05/14/2019	05/14/2019



## 5.8 Development tools

- Visual Studio 2017 for writing and debugging C++ code [22]
- Visual Studio Code for writing shaders and the project report [23]
- CMake to generate project files [14]
- Blender for creating 3D models [6]
- GIMP for image editing [29]
- REAPER to record and edit audio [12]
- PhpStorm to develop the website [13]
- PuTTY for connecting to the server [28]
- Git for version control of the code and assets [9]
- GitHub for hosting the git repository

## 5.9 Internal control and evaluation

The goals are concretely defined, and it is therefore easy to decide whether or not they are complete. The group of testers will be responsible for providing constructive feedback and criticism to improve the game. If a system is implemented, but has minor bugs, it will be considered complete until the bugfixing period starts.

# Chapter 6

## Documentation

### 6.1 Reports and technical documents

Progress reports will be written for every two weeks. In addition, the feedback from testers, gathered every two weeks, will be compiled and turned into a form of documentation. Metrics for frame rates and other statistics, such as platform details, might be useful to gather. Writing documentation for the framework is also important. The contents will be a reference on the classes and functions, as well as general guidelines, and how to use it. This information might be partially generated by the usage of tools that parse source code, such as Doxygen [4]. The documentation will be made publically available online. Other documentation may include UML diagrams to show the relationship between all systems, and how they interact with eachother.

# Chapter 7

## Planned meetings and reports

### 7.1 Meetings

Meetings will be held with the advisor every second Tuesday to report on progress. The meetings are useful to get constructive feedback from the advisor. They also provide a way to prove responsibility, by showing progress according to plan.

Date	Time
14.01.2019	10:00
31.01.2019	12:30
12.02.2019	11:00
26.02.2019	11:00
12.03.2019	11:00
26.03.2019	11:00
09.04.2019	11:00
23.04.2019	11:00
07.05.2019	11:00

*Table 2: Planned meeting schedule*



## **7.2 Progress reports**

The progress reports will describe the planned activities, and the actual work that was done. The planned activities are decided by following the project plan, defined in the gantt diagram. If any deviations occurred, they must be explained in the progress reports. These reports will be sent to the advisor on the day before each meeting. This gives the advisor time to look it over for the meeting. The report will then be discussed during the meeting.

# Chapter 8

## Planned deviation management

There are many systems at play here. Each system must be treated differently. Since the framework mostly works as expected as of writing this report, major problems are not expected. If a game system is harder to implement than anticipated, it should consume time from content development rather than excluding the functionality. In scenarios where the system has a minor issue, that issue should instead be fixed in the final bugfixing and polishing stage of the project.

Risk Assessment		SEVERITY		
		LOW	MEDIUM	HIGH
P R O B A B I L I T Y	HIGH	Leftover bugs that do not interrupt gameplay		
	MEDIUM	Lack of game content, tech must be prioritized	Problems in the game core, resulting in players having a bad experience playing	Server not performing enough validation checks, resulting in crashes or cheating
	LOW		Sickness, halting plan progress	Bugs with the framework that cause major problems

Table 3: Risk assessment matrix

# References

- [1] Apple. TrueType reference. <https://developer.apple.com/fonts/TrueType-Reference-Manual/>, accessed 2019-01-30.
- [2] Omar Cornut. Dear ImGui. <https://github.com/ocornut/imgui>, accessed 2019-01-30.
- [3] Datatilsynet. Software development with Data Protection by Design and by Default. <https://www.datatilsynet.no/en/regulations-and-tools/guidelines/data-protection-by-design-and-by-default>, accessed 2019-01-31.
- [4] Doxygen. Doxygen. <http://www.doxygen.nl/>, accessed 2019-01-31.
- [5] Glenn Fiedler. Fix Your Timestep! [https://gafferongames.com/post/fix\\_your\\_timestep](https://gafferongames.com/post/fix_your_timestep), June 10th, 2004 (accessed 2019-01-30).
- [6] Blender Foundation. About Blender. <https://www.blender.org/about>, accessed 2019-01-31.
- [7] FreeType. FreeType. <https://www.freetype.org/>, accessed 2019-01-30.
- [8] Epic Games. Unreal Engine. <https://www.unrealengine.com>, accessed 2019-01-28.
- [9] Git. About Git. <https://git-scm.com/about>, accessed 2019-01-31.
- [10] GLM. OpenGL Mathematics. <https://glm.g-truc.net>, accessed 2019-01-30.
- [11] Laurent Gomila. SFML. <https://www.sfml-dev.org>, accessed 2019-01-28.
- [12] Cockos Incorporated. REAPER. <https://www.reaper.fm>, accessed 2019-01-31.
- [13] JetBrains. PhpStorm. <https://www.jetbrains.com/phpstorm>, accessed 2019-01-31.

- [14] Kitware. About CMake. <https://cmake.org/overview>, accessed 2019-01-31.
- [15] Jason Koebler. Apple Is Lobbying Against Your Right to Repair iPhones, New York State Records Confirm. [https://motherboard.vice.com/en\\_us/article/nz85y7/apple-is-lobbying-against-your-right-to-repair-iphones-new-york-state-records-confirm](https://motherboard.vice.com/en_us/article/nz85y7/apple-is-lobbying-against-your-right-to-repair-iphones-new-york-state-records-confirm), May 18th, 2017 (accessed 2019-01-31).
- [16] Kim Kulling. The Open-Asset-Importer-Lib. <http://www.assimp.org/>, accessed 2019-01-30.
- [17] Sam Oscar Lantinga. SDL. <https://www.libsdl.org>, accessed 2019-01-28.
- [18] Alexis C. Madrigal. Facebook Didn't Sell Your Data; It Gave It Away. <https://www.theatlantic.com/technology/archive/2018/12/facebooks-failures-and-also-its-problems-leaking-data/578599/>, Dec 19th, 2018 (accessed 2019-01-31).
- [19] Microsoft. What is SaaS? <https://azure.microsoft.com/en-in/overview/what-is-saas/>, accessed 2019-01-30.
- [20] Microsoft. TrueType overview. <https://docs.microsoft.com/nb-no/typography/truetype/>, accessed 2019-01-30.
- [21] Microsoft. WASAPI. <https://docs.microsoft.com/en-us/windows/desktop/coreaudio/wasapi>, accessed 2019-01-30.
- [22] Microsoft. Visual Studio IDE. <https://visualstudio.microsoft.com/vs>, accessed 2019-01-31.
- [23] Microsoft. Visual Studio Code. <https://code.visualstudio.com>, accessed 2019-01-31.
- [24] Amit Patel. Introduction to A\*. <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>, accessed 2019-01-30.
- [25] David Pogue. Adobe's Software Subscription Model Means You Can't Own Your Software. <https://www.scientificamerican.com/article/adobe-software-subscription-model-means-you-cant-own-your-software/>, October 2013 (accessed 2019-01-31).

- [26] Jon Porter. Google accused of GDPR privacy violations by seven countries. <https://www.theverge.com/2018/11/27/18114111/google-location-tracking-gdpr-challenge-european-deceptive>, Nov 27th, 2018 (accessed 2019-01-31).
- [27] Greg Roelofs. PNG. <http://www.libpng.org/pub/png>, accessed 2019-01-30.
- [28] Simon Tatham. PuTTY. <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>, accessed 2019-01-31.
- [29] The GIMP Team. About GIMP. <https://www.gimp.org/about>, accessed 2019-01-31.
- [30] Unity Technologies. Unity. <https://unity3d.com>, accessed 2019-01-28.
- [31] TurboSquid. About exchange formats. <https://www.turbosquid.com/3d-modeling/turbosquid/product-page/about-exchange-formats/>, accessed 2019-01-30.
- [32] Koen Witters. deWiTTERS Game Loop. <http://www.koonsolo.com/news/dewitters-gameloop>, July 13th, 2009 (accessed 2019-01-30).
- [33] Xiph.org. Ogg. <https://xiph.org/ogg>, accessed 2019-01-30.
- [34] Xiph.org. Vorbis. <https://xiph.org/vorbis>, accessed 2019-01-30.

# **Appendix B**

## **Progress reports**

# Progress report #1 - 12.02.18

## Planned activities

1. Add inventory and equipment
2. The client should download new updates from the server on launch
3. Finish world editor
4. Make a simple website where you can sign up and download the game
5. Conduct the first feedback poll

## Completed work

1. Added inventory and equipment interfaces and functionality
  - a. Includes adding right-click context menu
  - b. Item container has add/remove events, which UI renderer listens to
2. Added autotile and objects placement in world editor
  - a. Objects now saved to file, unlike before
3. Added auto-update feature
4. Made a simple website where you can sign up and download the game
  - a. Users stored in postgresql database
5. Conducted the first feedback poll
  - a. Some days late according to plan, to allow for some features to be added
6. Added tabs interface to change between inventory, equipment, etc...
7. Added sky color and fog
8. Finished item and object definition systems
9. The world object container now communicates with the renderer (add/remove)



## Progress report #2 - 26.02.18

### Planned activities

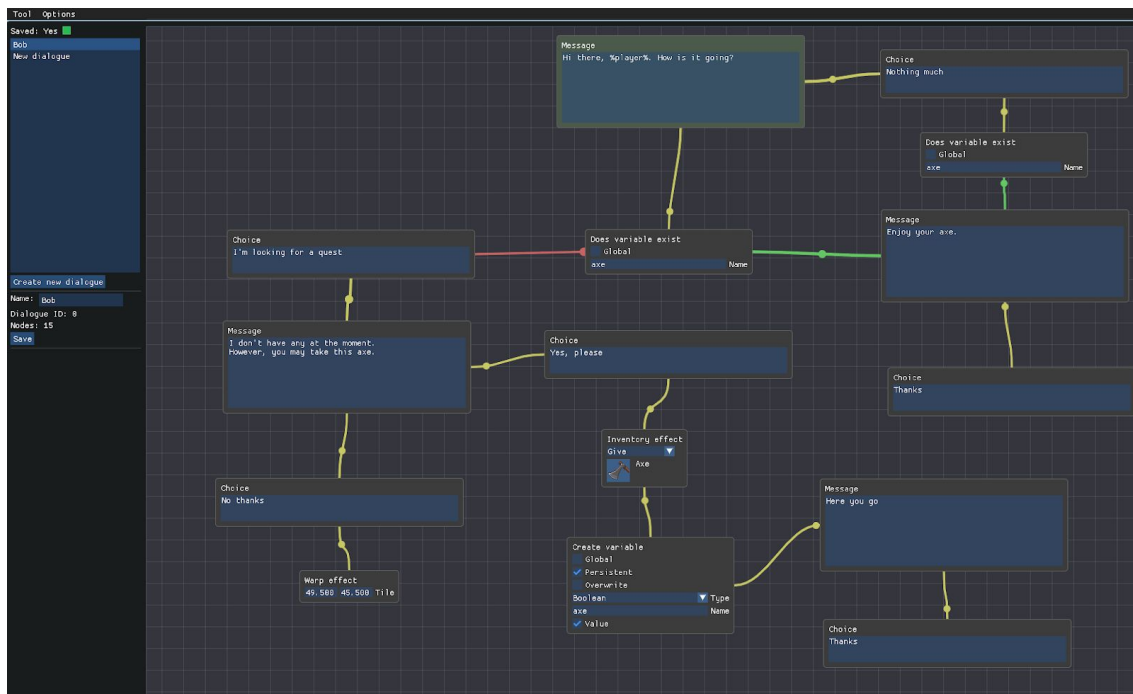
1. Create dialogue editor
2. Implement dialogue system and UI
3. Chat system
4. Persistency

### Not completed

1. Work was not yet started on persistency, due to some circumstances

### Completed work

1. Created dialogue editor
2. Implemented the dialouge system and UI
  - a. UI not entirely finished
3. Chat system was added
4. Finished auto update
  - a. Some critical bugfixes to networking code, which became apparent when sending large amounts of data at once



Screenshot of dialogue editor



## Progress report #3 - 12.03.18

### Planned activities

1. Persistency - write/read player data from database
2. Quest system
3. Pathfinding
4. Start on combat system

### Completed work

1. Refactored network code to make it simpler
2. Using libpq in server to read to and write from the database
3. Added quest system with editor for quests
4. Added pathfinding
  - a. Also added tile flags, such as "solid", which can be set in editor
5. Started on combat system
6. Refactored object system slightly

## Progress report #4 - 26.03.18

### Planned activities

1. Add a minimap to the interface
2. Make a quest
3. Add player trading
4. Define where area music plays in toolkit

### Not completed

1. Because of the refactoring done:
  - a. player trading was pushed back
  - b. did not do anything with area music

### Completed work

2. Implemented the minimap, although it needs some polishing later
3. Made the first quest, and started on the second
4. Refactored skeletal animation in framework, as it wasn't flexible enough
  - a. Improved the game's character renderer, for both flexibility and performance
5. Added equipment interface
6. Fixed a bug with cursor updates, and at the same time made it so pointer cursor appears when hovering certain interface elements
7. Refactor context menu slightly
8. Character objects now has Y coordinate set to the elevation of the tile
9. Added some random movement to characters

### Some videos:

- <http://einheri.xyz/files/client-1.mp4>
- <http://einheri.xyz/files/editor-1.mp4>
- <http://einheri.xyz/files/editor-2.mp4>
- <http://einheri.xyz/files/editor-3.mp4>

## Progress report #5 - 09.04.18

### Planned activities

1. Add player trading
2. Add the fishing skill
3. Add leaderboard to the website
4. Add a map of the game world to the website
5. Warehouses

### Not completed

1. Warehouses not completed
2. Did not add map to the website

### Completed work

3. Added player trading
4. Added the fishing skill
5. Added leaderboards to the website (but didn't add any sorting yet)
6. Objects outside the loaded terrain are no longer drawn
7. The character animations that were not put to use yet, are now working:
  - a. idle while in combat
  - b. spear variations of animations
  - c. fishing animations
8. Added some new items (coins, various pants, fish)

### Some videos:

- <http://einheri.xyz/files/client-2.mp4>
- <http://einheri.xyz/files/client-3.mp4>

## Progress report #6 - 23.04.18

### Planned activities

1. Bugfixing
2. Add minor features that are currently lacking (stats tab, skybox)
3. Improve how water looks

### Completed work

1. Minimap texture is now contained within the circular foreground
2. Added stats tab that shows progress in each stat
3. Players now get experience for combat and fishing
4. Added several items (shirts, pants, shoes, helm)
5. Fixed diffuse light and fog in the shaders
6. The terrain now has proper vertex normals calculated
7. Water no longer a tile, and is instead a plane drawn by a water shader
8. Items now have stats that affect combat
9. The UI is revamped
10. Skeletal animation is now multithreaded (improved performance by around 30-40%)
11. Character renderer now uses vertex groups to hide legs when wearing pants etc.
12. World now stored in chunks, since non-chunk based approach became cumbersome
13. Fixed issue where player moved back by half a tile if new target was set while moving
14. Added stone and snow tiles
15. Pathfinding now done only by server, to minimize sync issues
16. Added skybox
17. Rotation during movement more accurate
18. Combat positioning is better (but still a few minor issues)

### Some videos:

- <http://einheri.xyz/files/client-4.mp4>
- <http://einheri.xyz/files/client-5.mp4>

# Appendix C

## Network API

The framework's network API consists of 12 functions, and 1 data structure.

---

```
int open_socket();
int open_socket(const std::string& address, int port);
void close_socket(int id);
void synchronize_socket(int id);
void synchronize_sockets();
bool bind_socket(int id, const std::string& address, int port);
bool listen_socket(int id);
bool increment_socket_accepts(int id);
void socket_send(int id, io_stream&& stream);
void broadcast(io_stream&& stream);
void broadcast(io_stream&& stream, int except_id);
socket_events& socket_event(int id);

struct socket_events {
    message_event<io_stream> stream;
    message_event<io_stream> packet;
    message_event<socket_close_status> disconnect;
    message_event<int> accept;
};
```

---

# Appendix D

## Graphics API

The framework's graphics API consists of 30 functions, and 1 class.

---

```
int create_vertex_array(const vertex_specification& specification);
void bind_vertex_array(int id);
void set_vertex_array_vertices(int id, uint8_t* buffer, size_t size);
void set_vertex_array_indices(int id, uint8_t* buffer, size_t size);
void draw_vertex_array(int id);
void draw_vertex_array(int id, size_t offset, size_t count);
void delete_vertex_array(int id);

int create_texture();
int create_texture(const surface& surface, scale_option scaling, bool mipmap);
int create_texture(const surface& surface);
void bind_texture(int id);
void bind_texture(int id, int slot);
void load_texture(int id, const surface& s, scale_option scaling, bool mipmap);
void load_texture(int id, const surface& surface);
void load_texture_from_screen(int id, int bottom_y, int x, int y, int w, int h);
vector2i texture_size(int id);
void delete_texture(int id);
```

```
int create_shader(const std::string& path);
void bind_shader(int id);
shader_variable get_shader_variable(const std::string& name);
void set_shader_model(const glm::mat4& transform);
void set_shader_model(const transform2& transform);
void set_shader_model(const transform3& transform);
void set_shader_view_projection(const glm::mat4& view, const glm::mat4& proj);
void set_shader_view_projection(const ortho_camera& camera);
void set_shader_view_projection(const perspective_camera& camera);
void delete_shader(int id);

void set_polygon_render_mode(polygon_render_mode mode);
vector3i read_pixel_at(vector2i position);

class shader_variable {
public:
    int location = -1;
    shader_variable() = default;
    shader_variable(int program_id, const std::string& name);
    void change(T value) const;
    bool exists() const;
};
```

---

# Appendix E

## Audio API

The audio API consists of 3 classes. Virtual destructors omitted to avoid page break.

---

```
struct audio_source {
    virtual size_t size() const = 0;
    virtual audio_data_format format() const = 0;
    virtual const io_stream& stream() const = 0;
};

struct audio_player {
    virtual void play(const audio_stream& stream) = 0;
    virtual void play(audio_source* source) = 0;
    virtual void pause() = 0;
    virtual void resume() = 0;
    virtual void stop() = 0;
    virtual bool is_playing() const = 0;
    virtual bool is_paused() const = 0;
};

struct audio_endpoint {
    virtual audio_player* add_player() = 0;
    virtual void stop_all_players() = 0;
    virtual void clear_players() = 0;
};
```

---



# Appendix F

## Platform API

The platform API consists of 7 functions.

---

```
long long performance_frequency();  
long long performance_counter();  
void sleep(int ms);  
std::string environment_variable(const std::string& name);  
std::string open_file_browse_window();  
std::vector<std::string> command_line_arguments();  
void relaunch();
```

---

# Appendix G

## Window API

The window API consists of 1 class. Some synonymous member functions are omitted.

---

```
class window {
public:
    void poll();
    void clear();
    void swap();
    void maximize();
    void set_title(const std::string& title);
    void set_cursor(mouse::cursor icon);
    void set_size(const vector2i& size);
    void set_clear_color(const vector3f& color);
    std::string title() const;
    vector2i size() const;
    vector2i position() const;
    bool is_open() const;
    bool set_swap_interval(swap_interval interval);
    void set_viewport(int x, int y, int width, int height);
    void scissor(int x, int y, int width, int height);
    void reset_scissor();
};
```

---