1 Problem Statement and Initial Design Specification

There are currently few cross platform open world space games that allow for the building of complex systems and vessels, in a multiplayer scenario. Software such as Space Engineers have limited performance on platforms such as Linux or PS4, as they cannot run DirectX (only Xbox and Microsoft Windows can run DirectX). As well as having limited performance due to limitations of DirectX. Currently there are no massively open world space games that run on Vulkan (improved performance and compatibility in comparison to DirectX). While games such as From the Depths that allow for complex control systems, run into limitations with the physics engine they use, as they are limited to 32 bit floating point numbers by virtue of using the unity game engine. Using godot allows for the game to be supported into the future (Unity deprecating support for most of the features popular games use, forcing developers to port features or stay locked to an older version of the game engine).

Thus, novel solutions that can act as a jumping off point for games to be developed with an improved engine capable of multiplayer and large world support are needed. As currently all implementations such as the Frostbite engine (Keen Software House, used in Space Engineers) are proprietary and only support specific platforms and vendors.

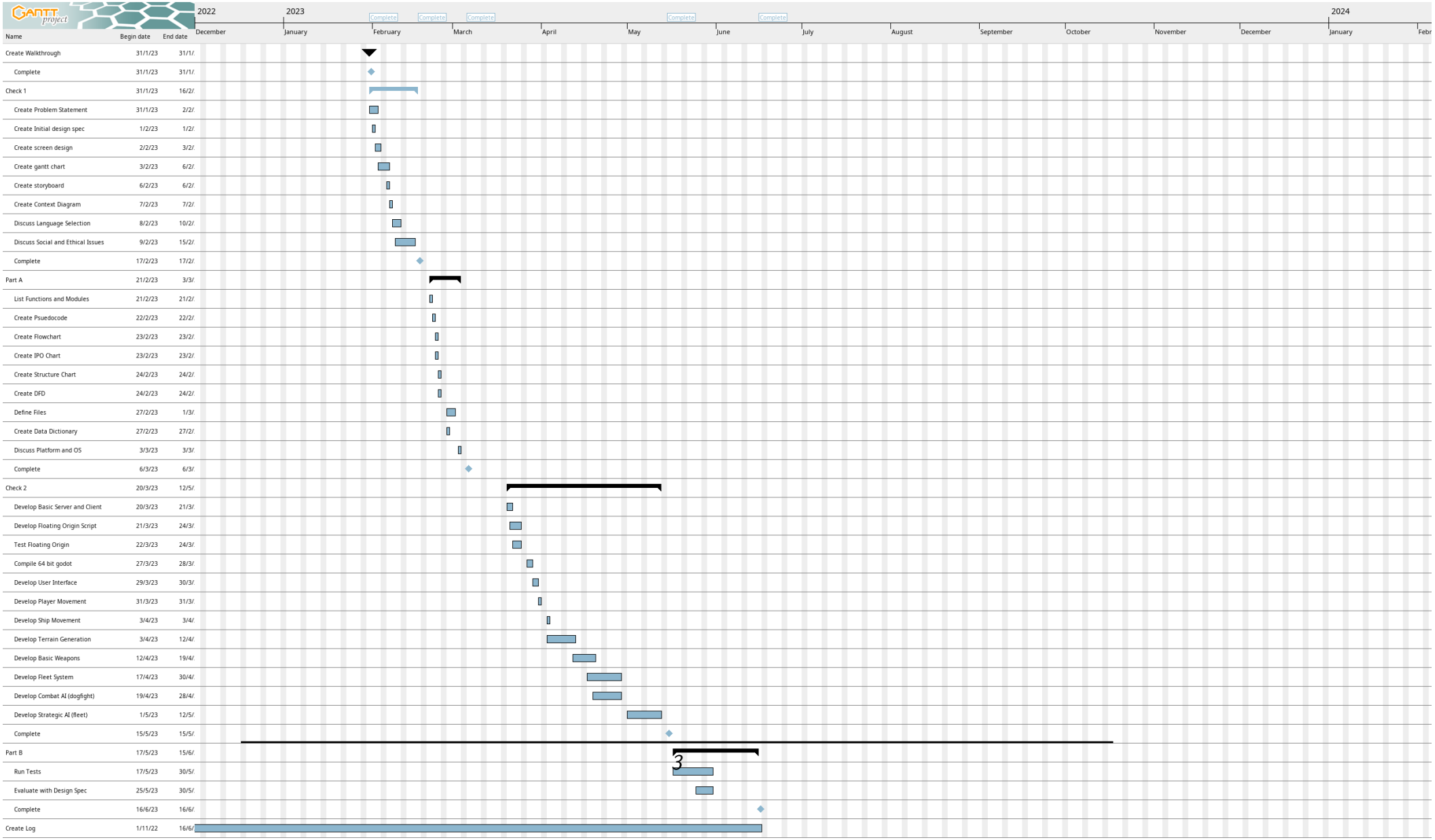
1.1 User Specifications

- Responsive controls and remapable control scheme
- User interface that allows for filtering
- Ingame tutorial for each major system in the game
- Personal details are used properly
- Network connections don't suddenly drop out
- Graphics look nice, are high quality
- Audio sounds nice and has volume controls

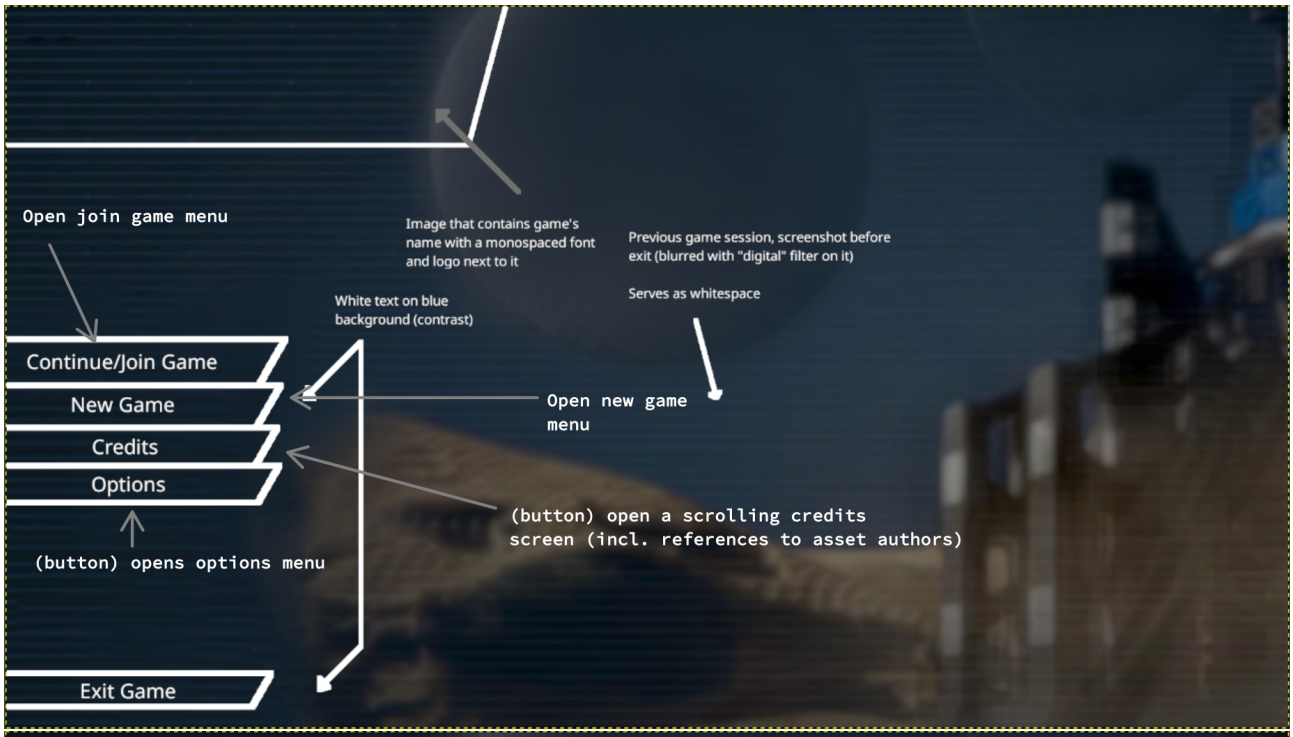
1.2 Developer Specifications

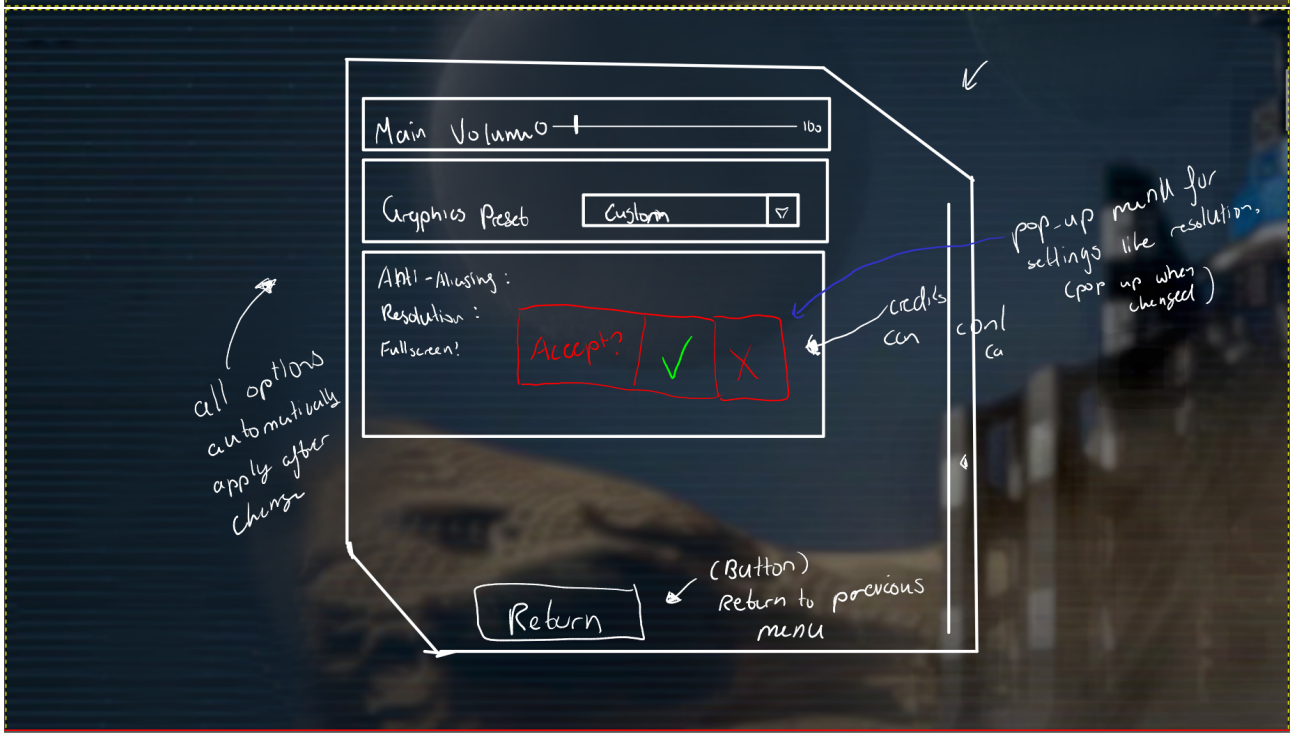
- Developed using an agile development approach
- Data structures can be efficiently serialised for sending over the network
- Server and Client FPS must not drop below 60 on an intel i7 1165G7 or equivalent
- API documentation for modding is exposed with a tool such as Doxygen
- Server and Client must be able to run under Vulkan or DirectX without bugs
- Server and Client must be able to run on both Windows and Linux without bugs

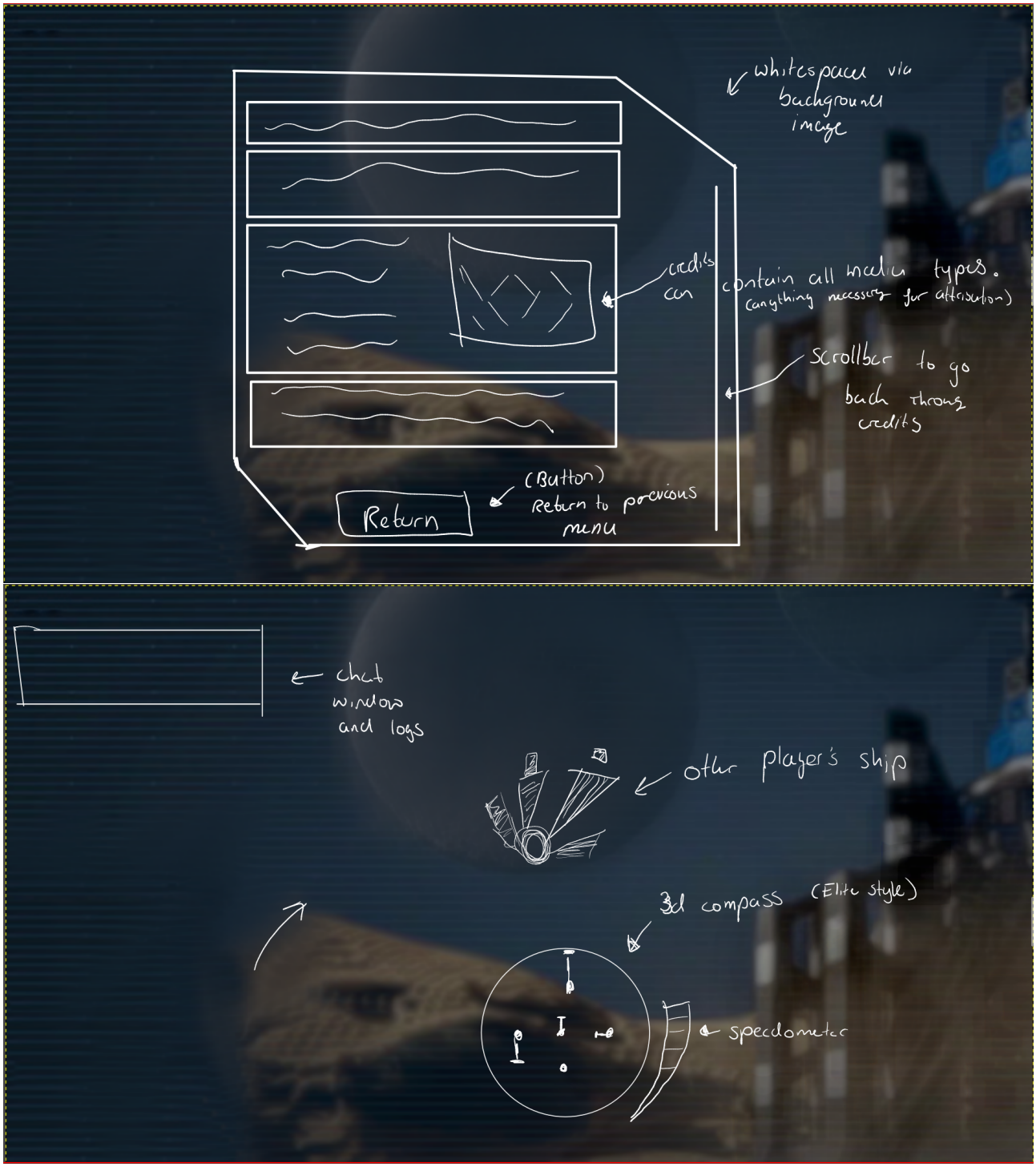
- Server to Client communications must be over IPC on the same machine, and Websockets or UDP on internet
- Server and Client communications must be tolerant to at least 50ms of latency and 10 p.c. packet loss
- Server must not use more than 16GB of RAM
- Client must not use more than 8GB of RAM

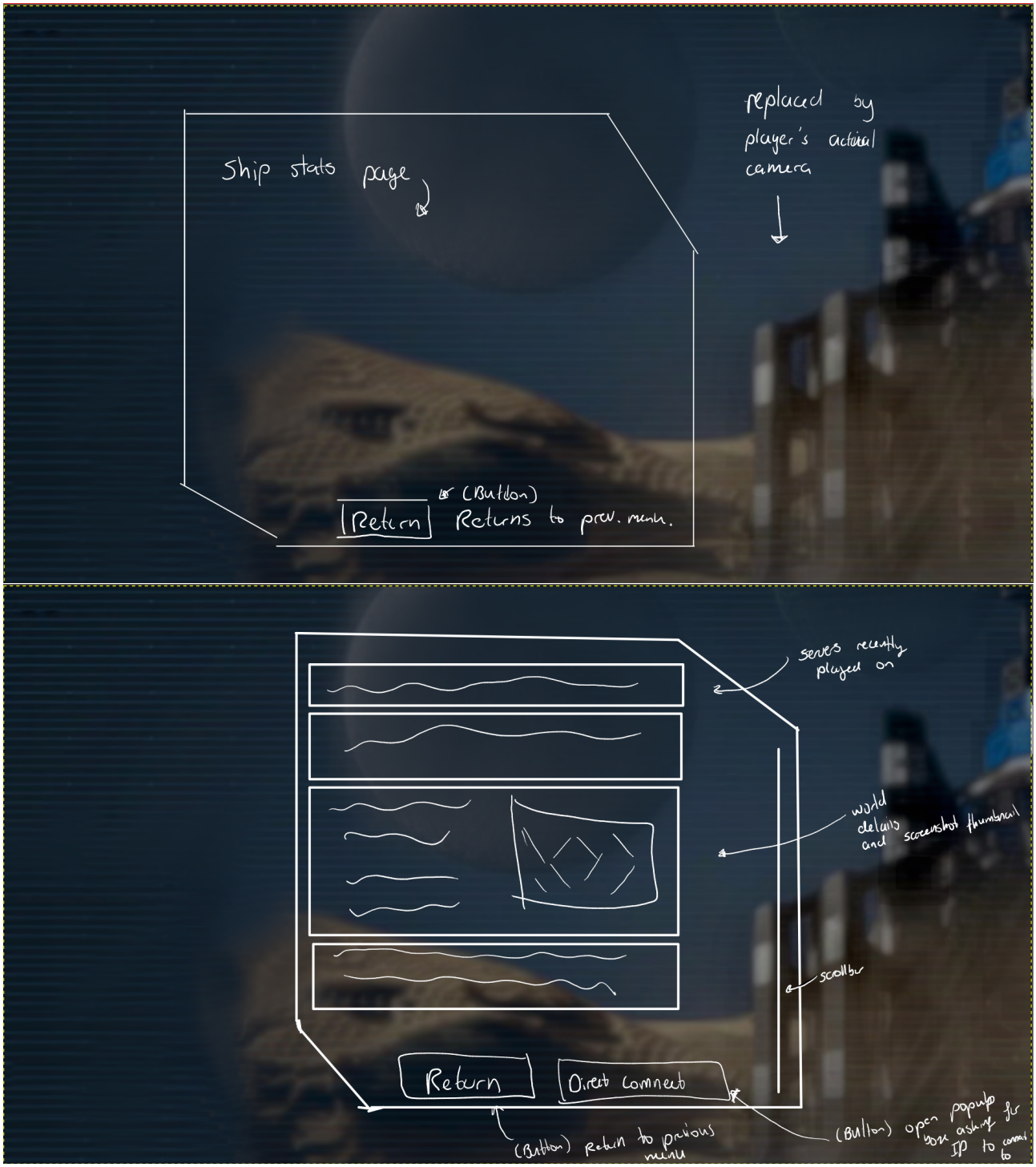

2022
2023
2024
December
January
February
March
April
May
June
July
August
September
October
November
December
January
Febr
Complete
Complete
Complete
Complete
Complete
Name
Begin date
End date
Create Walkthrough 31/1/23 31/1/
Complete 31/1/23 31/1/
Check 1 31/1/23 16/2/
Create Problem Statement 31/1/23 2/2/
Create Initial design spec 1/2/23 1/2/
Create screen design 2/2/23 3/2/
Create gantt chart 3/2/23 6/2/
Create storyboard 6/2/23 6/2/
Create Context Diagram 7/2/23 7/2/
Discuss Language Selection 8/2/23 10/2/
Discuss Social and Ethical Issues 9/2/23 15/2/
Complete 17/2/23 17/2/
Part A 21/2/23 3/3/
List Functions and Modules 21/2/23 21/2/
Create Psuedocode 22/2/23 22/2/
Create Flowchart 23/2/23 23/2/
Create IPO Chart 23/2/23 23/2/
Create Structure Chart 24/2/23 24/2/
Create DFD 24/2/23 24/2/
Define Files 27/2/23 1/3/
Create Data Dictionary 27/2/23 27/2/
Discuss Platform and OS 3/3/23 3/3/
Complete 6/3/23 6/3/
Check 2 20/3/23 12/5/
Develop Basic Server and Client 20/3/23 21/3/
Develop Floating Origin Script 21/3/23 24/3/
Test Floating Origin 22/3/23 24/3/
Compile 64 bit godot 27/3/23 28/3/
Develop User Interface 29/3/23 30/3/
Develop Player Movement 31/3/23 31/3/
Develop Ship Movement 3/4/23 3/4/
Develop Terrain Generation 3/4/23 12/4/
Develop Basic Weapons 12/4/23 19/4/
Develop Fleet System 17/4/23 30/4/
Develop Combat AI (dogfight) 19/4/23 28/4/
Develop Strategic AI (fleet) 1/5/23 12/5/
Complete 15/5/23 15/5/
Part B 17/5/23 15/6/
3
Run Tests 17/5/23 30/5/
Evaluate with Design Spec 25/5/23 30/5/
Complete 16/6/23 16/6/
Create Log 1/11/22 16/6/

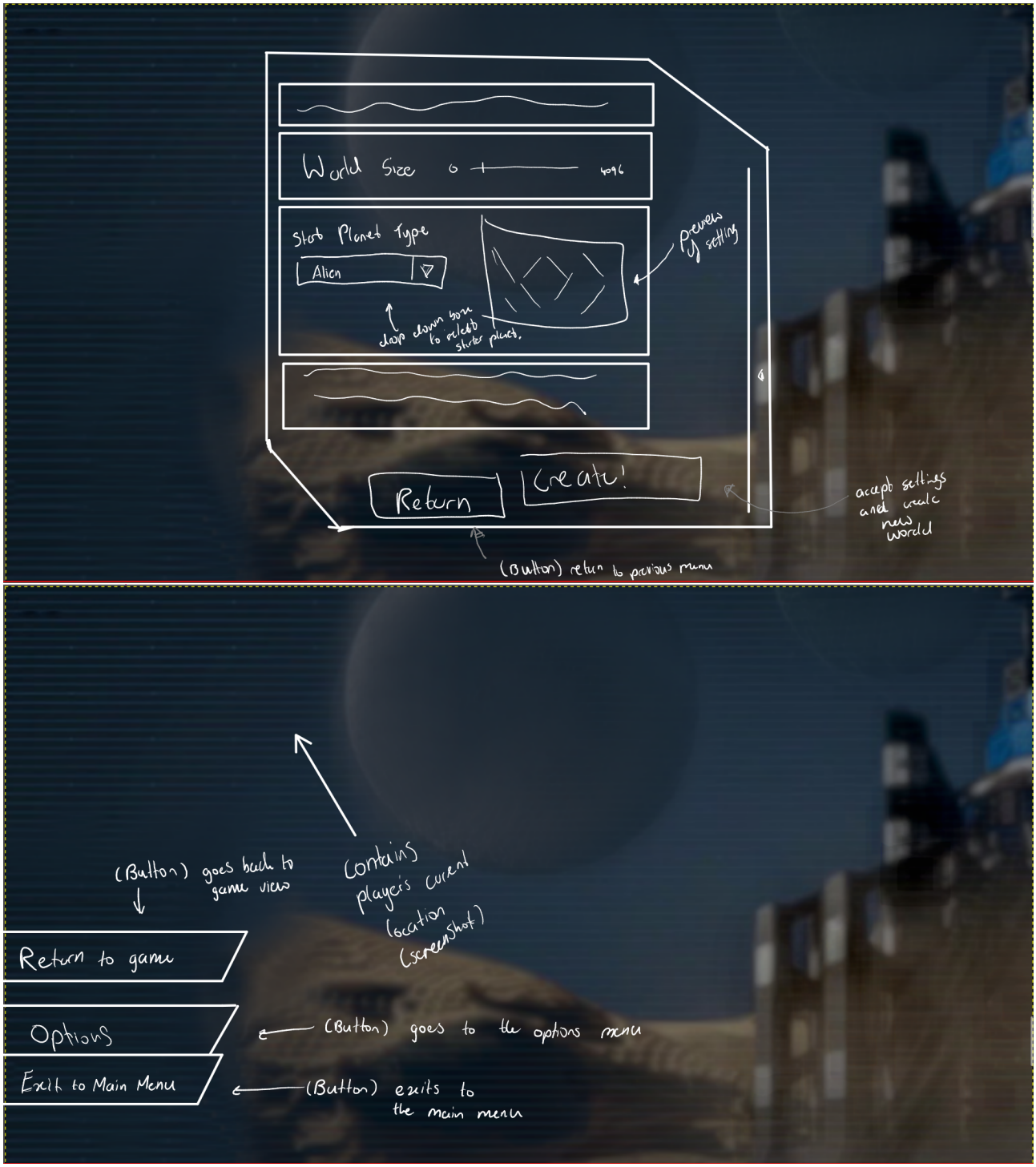
3 Screen Designs



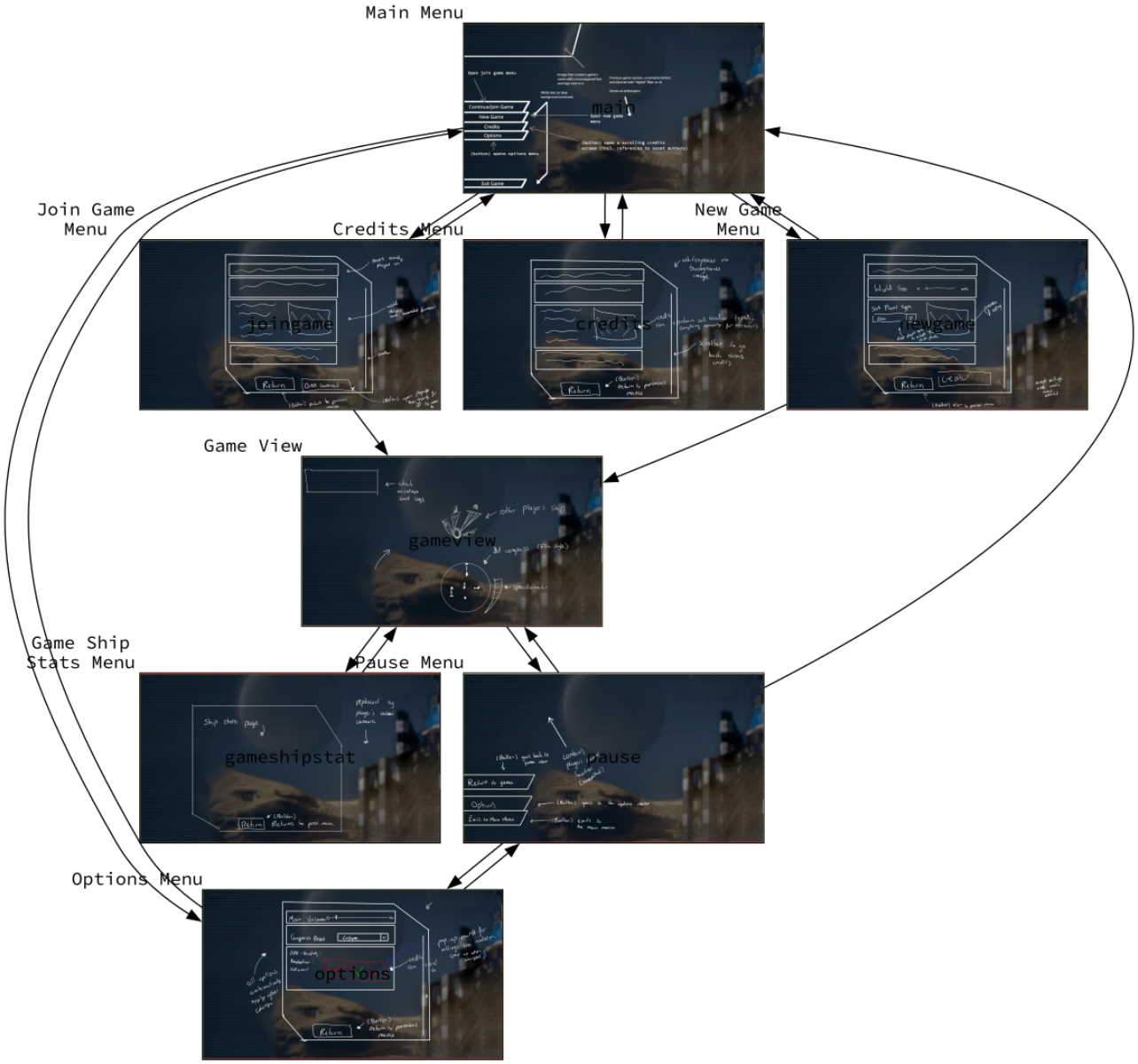



whitespaced via background image
credits can
contain all media types. (anything necessary for attribution)
Scrollbar to go back through credits
(Button) Return to previous menu
Return
chat window and logs
other player's ship
3d compass (Elite style)
speedometer

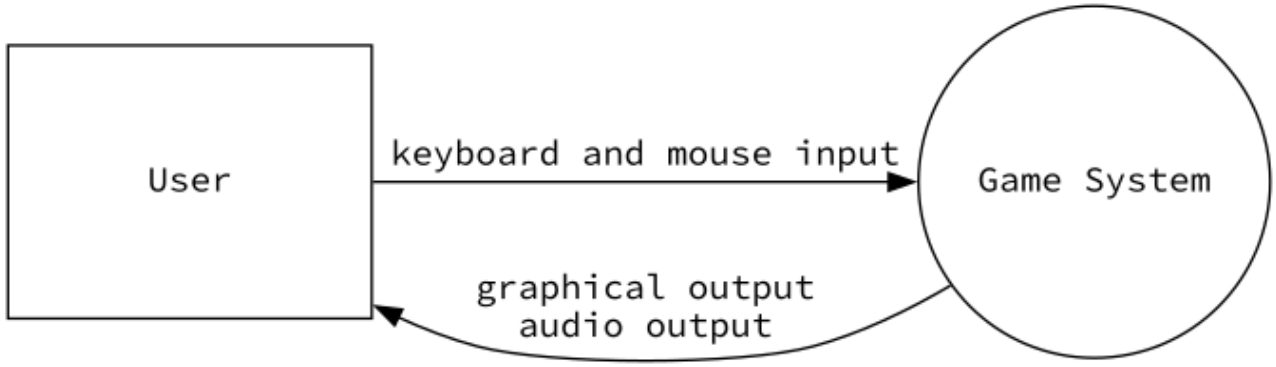

Ship stats page
replaced by player's actual camera
Return
(Button) Returns to prev. menu.
servers recently played on
world details and screenshot thumbnail
scrollbar
Return
Direct connect
(Button) Return to previous menu
(Button) open popup box asking for IP to connect to


World Size
0
4096
Start Planet Type
Alien
preview of setting
drop down box to select start planet.
Return
Create!
accept settings and create new world
(Button) return to previous menu
(Button) goes back to game view
Contains player's current location (screenshot)
Return to game
Options
(Button) goes to the options menu
Exit to Main Menu
(Button) exits to the main menu

4 Storyboard (initial)



5 Context Diagram



6 Discussion of Selected Programming Language

This project will utilise C# (Mono), GDScript (Godot Engine integrated language), and C++.

Both the server and client will be written using the languages mentioned above, as both will be developed using the godot game engine (as it is capable of producing builds optimised for servers with no graphics output). C# is intended to be the main language as it offers many advantages over writing the entire project in a traditionally compiled language. As this allows for rapid prototyping and debugging without having to wait for compile times that may be in the tens of minutes. This is due to C# being JIT compiled. It is also the primary language due to my familiarity with the language and use in previous projects, as well as the widespread use of C# in games and thus available documentation, tutorials, and pitfalls (anything that uses the Unity game engine, and many other indipendent games such as Aurora C#).

GDScript will be used for parts of the game that can be more efficiently written in the game engine native language. As it is created by and for the godot engine it has been shown to be faster than C# in certain tasks (execution time is lower). It is also an interpreted language removing the need for long compile times. Examples of places in which GDScript beats C# in performance include code which must deal with large numbers of tuples (generally 3D Vectors), as there are machine code optimisations that cannot be applied to C# available in GDScript (PackedVector3).

C++ Will be used sparingly, in performance critical sections of code. This is due to the long compile times of C++ projects (even with incremental compilers), the complexity of setting up the GDExtension system, and the balance of runtime speed VS development time (more bugs and more difficult to use as it is closer to hardware). The GDExtension system is a part of Godot 4.0 rewritten to allow the use of any language (especially Rust and C++) to be compiled as part of a godot game. C++ Will be used for segments such as the floating origin reorigin code, as it needs to complete many operations incredibly quickly. The use of C++ in these sections also allows for the use of software only accelerated quad floating precision numbers (128 bit floats, through GCC libquadmath on 32 bit or 64 bit hardware).

The godot engine for this project will be different on the server and client, as the client can use the release version of the Godot engine, but the server side necessitates the need for a custom build of the godot engine (with `float=64` passed) to allow for the use of 64 bit coordinates at the minimum.

7 Social and Ethical Considerations

7.1 Copyright

Assets such as code, textures, videos, and sounds must all be credited to their original author if under a commons license. Else proper attribution or non use in the game may be required to comply with licensing terms. This includes code uploaded to sites such as stackoverflow as the website EULA signifies any code posted is under their copyright.

This project also must be licensed with an open source, or source available license due to the code libraries planned to be in use (GNU and Godot). This will make security more difficult as the encryption code can be seen in public view easily, and reversed trivially to decrypt sensitive user information. This would also allow for cheating in the game as flaws could be exploited by cheaters. Though this is somewhat counteracted by an open source license that allows anyone (i.e. security researchers) to look at the code and fix the offending statements.

7.2 Accessibility

The accessibility of this project exacerbates the digital divide, as the minimum performance bar of a computer is quite high for this style of game. Due to the many simultaneous threads processing at any one time, only higher performance computers will be able to run this software. However efforts can be made to optimise slower parts of the game to lower the hardware lower entry limit.

Accessibility such as hearing impairment can also be accounted for with systems such as closed captions in the case of character speech, or systems that subtitle environmental sounds and the direction they originate relative to the player (e.g. Minecraft) as an aid. Visual impairments are difficult to accomodate in this genre as they mainly consist of environments that are difficult to portray on devices such as a braille console or aided through pure audio. This can be somewhat helped by the compatibility of this software with popular screen readers and custom output to allow for navigation in space.

7.3 Security

Details about connected clients such as IP addresses or other sensitive data can be used maliciously by other clients if the Server does not keep such details secure (e.g. GTA Online). Encryption can prevent this kind of attack, as well as warning the user when they are connecting to non official servers (operated by other users), as details can still be retrieved by server operators. Encryption of data in saved files can also combat this, to prevent third parties breaking into official servers and retrieving user details.

7.4 Gender Discrimination

Due to the nature of online games, gender discrimination of users is prevalent. Thus the exclusion of services such as VoIP or ingame chat may be desirable for some users. This can also be implemented in features such as the ability to block specific other users' communications (e.g. Valve online services) or the ability to apply specific filters to only allow certain messages by default (e.g. Discord communications platform).

8 Functions and Modules

Name	Description
Renderer	Contains code required to

9 Psuedocode and Flowchart

9.1 Psuedocode

```
BEGIN getClosestPlayer(x, y)
    Let distanceX = Players.x(1) - x
    Let distanceY = Players.y(1) - y
    Let minimumDistance = sqrt(distanceX ** 2 + distanceY ** 2)
    Let closestPlayer = Players(1)
    FOR i = 2 to length of Players STEP 1
    distanceX = Players.x(i) - x
    distanceY = Players.y(i) - y
    Let distance = sqrt(distanceX ** 2 + distanceY ** 2)
    IF distance < minimumDistance THEN
    minimumDistance = distance
    closestPlayer = Players(i)
    ENDIF
    NEXT i
    RETURN closestPlayer
END getClosestPlayer
```

9.2 System Flowchart

10 IPO Chart

Input	Process	Output
stuff	stuff	stuff

11 Structure Chart

sdf

12 Data Flow Diagram

ddd

13 Files and Data Structures

File Name	Type or Structure	Description
starmap.exr	Generated OpenEXR panorama sky image.	stuff

14 Data Dictionary

Data Item	Data Type	Format	Number of Bytes required for storage	Size for display	Description	Example	Validation
Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff
Online Username	stuff	stuff	stuff	stuff	stuff	stuff	stuff
Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff
Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff
Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff
Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff

Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff
Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff
Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff
Planet	stuff	stuff	stuff	stuff	stuff	stuff	stuff

15 Platform and OS Considerations

The primary requirements are an `x86_64` (also known as `AMD64`) CPU that is able to do accelerated math and must have fast primary memory (RAM) to be able to run the game server at high frame rates. Due to these feature older (32 bit) computers will not be able to run the **server**, but will be able to run the client, as a compromise has been made to allow for further compatibility.

Devices and platforms that do not expose a high-performance API that is available for open source integration will also be excluded. For the scope of this project that excludes macOS, iPadOS, and iOS from being able to run the game, as they do not provide an OpenGL, Vulkan or DirectX layer (as the Godot game engine only supports these API's). Due to this the platforms that will be able to run the client are: Windows 7 (or newer), Any Linux distribution using X11 or XWayland, such as ubuntu 18.04+, Debian 10+, and any BSD with X11/XWayland with support for Vulkan. Devices like Apple macbooks are not included as they do not support OpenGL or Vulkan, and require the use of the `Metal` graphics API. Godot does not support this API.

16 Logbook

Date	Description	Problems Encountered	What I need to do	Completion signoff

FIX	Figure out how to get godot to use higher precision floating point numbers, in order to combat FPPE more accurately	Godot cannot handle higher than 64 bit numbers even with libraries like libquadmath and require massive core rewrites to achieve any further. To enable 64 bit mode the game engine must be recompiled. Due to the project using C# the game engine and bindings must all be regenerated specifically for the server	Create a custom build script that can automatically download the engine source code and build my game alongside it. Consider looking into other solutions to use external libraries to achieve this.	in progress
FIX	FIX	FIX	FIX	FIX