PRCO 304 Final Stage Computing Project

# Abstract

This report will outline the stages that were required to create a 2D Networked Online Roleplaying game, that’s called Dan’s World. These stages include the planning, design and implementation of all the components that were built for the project. The reason for choosing this project was to further personal understanding of specific areas of software engineering, those being network engineering and video games programming. The primary project objectives were to see how best to implement a network protocol that can handle communications to a server from multiple clients.

Object-Oriented programming was a key element within this project to promote modularity, encapsulation and maintainability within the project. These topics are discussed in depth within the report.

This project saw a lot of unforeseen challenges, multiple late deliverables but little changes in what was produced from the deliverables that were planned. Many of the initial requirements of the game itself suffered in favour of the network protocol getting to a stage where it is functional and stable.

The development process was incremental, meaning that requirements were added one at a time as a feature was written off as working. The main technologies used was C# .NET Core to allow the server to be hosted on any mainstream OS, be that windows, Linux or MacOS, MySQL Database server to save character progression for the same reason as .NET Core, Github for source control and lastly MonoGame which is a games programming framework written for C#. MonoGame is a project that was inherited from the Microsoft XNA team.

Finally, the project post mortem will look at the project management and project goals and whether they were a success or not. The conclusion will be interrogating the produced software along with personal thoughts and feelings of its success.

Contents

[Abstract 1](#_Toc514255169)

[1. Introduction 3](#_Toc514255170)

[1.1 Dan’s World 3](#_Toc514255171)

[1.2 Project Goals 4](#_Toc514255172)

[2. Legal, Social and Ethical Issues 4](#_Toc514255173)

[2.1 Game Assets 4](#_Toc514255174)

[2.2 Data Collection 5](#_Toc514255175)

[3. Project Management 5](#_Toc514255176)

[3.1 Method of Approach 5](#_Toc514255177)

[3.2 Data and Time Management 5](#_Toc514255178)

[3.3 Design Changes 5](#_Toc514255179)

[4. Deliverables 5](#_Toc514255180)

[4.1 Core Deliverables 6](#_Toc514255181)

[4.1.1 Table of Core Deliverables 6](#_Toc514255182)

[4.1.2 Clarification of Changed Deliverables 7](#_Toc514255183)

[4.1.3 Clarification of Removed Deliverables 7](#_Toc514255184)

[4.2 Desirable Features 7](#_Toc514255185)

[5. Quality Control 7](#_Toc514255186)

[6. Project Plan 7](#_Toc514255187)

[7. System Design 7](#_Toc514255188)

[8. Graphics Design 8](#_Toc514255189)

[9. Implementation 8](#_Toc514255190)

[9.1 Database Implementation 8](#_Toc514255191)

[9.2 Server Implementation 8](#_Toc514255192)

[9.3 Game Client Implementation 8](#_Toc514255193)

[10. Lessons Learned 8](#_Toc514255194)

[11. Conclusion 8](#_Toc514255195)

[Appendices 8](#_Toc514255196)

Word Count:

# 1. Introduction

## 1.1 Dan’s World

The video game project that will be covered in this report is called Dan’s World, a 2D Networked Online Roleplaying Game. As shown in Figure 1 you can see 2 players logged in to the game and the active chat window showing a message from the other player.

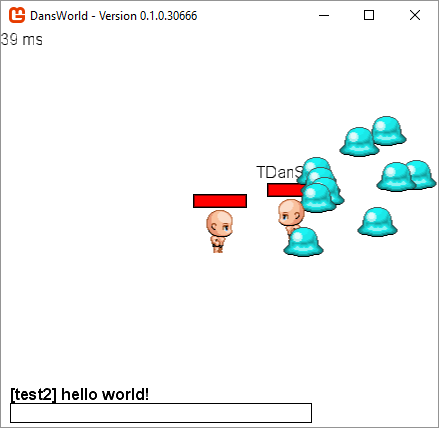


Figure 1 - Rendering of the player and enemies. Shows player moving between enemies behind and in front of them.

The idea of the game was to be able to register an account, create a character, and log in to the character starting at level 0. As the player eliminates enemies the character should gain experience, level up and be able to distribute skill points like many mainstream Online RPG’s, taking influences from games like World of Warcraft.

## 1.2 Project Goals

1. Experience sending data over a networked socket with a custom protocol
2. Experience with 2D games programming, making use of good software engineering practices to create a modular and maintainable code base and a popular Games Programming framework
3. Handling many concurrent network connections in a multithreaded environment

# 2. Legal, Social and Ethical Issues

## 2.1 Game Assets

In game character and NPC sprites were found on <https://opengameart.org/> with the enemy sprites requiring credit to a Strange Dragon who can be found at <https://strangedragonblog.wordpress.com/category/rpg-maker-mv-resources/page/2/>. The only legal issue I can see with using these assets is if they were to be used for revenue, however if this project was to be taken into a commercial setting, the art assets would be made completely bespoke. All buttons, labels and other UI components are programmatically constructed within the client code and the font used can be found here <https://fontmeme.com/fonts/gw-two-font/>.

## 2.2 Data Collection

Due to the nature of an online networked game some data must be collected about the person playing. That being an email to verify the integrity of a player’s account, a password to grant access to said account and their name in the case of account recovery. While data collection is possible, no data was obtained belonging to any other person, therefore there is no concern for legal issues surround data collection.

# 3. Project Management

## 3.1 Method of Approach

Initially the model of project management was decided to be PRINCE2 Management, however partway through the project, it was realised that not every aspect of the project can be ironed out from the offset. The initial design documents took much longer than anticipated and put the project behind schedule by a number of weeks, so development was hindered greatly by this. It is fair to say that some features were overlooked when choosing this as a project, therefore partway into the project, it was hoped to change the idea to something with a smaller scope. However, it was deemed too late into the process of the project to warrant a change of project idea.

Once some design documentation was complete as a starting point, development on the server and client architecture could begin. As time was limited, an MVP (Minimum Viable Product) approach was taken to get base functionality within the applications based on what was deemed to be most important.

## 3.2 Data and Time Management

Git was used on the social source control platform known as github, which allowed any technical disasters on the development machine to be avoided by obtaining the latest version of the code that was pushed to the github repository. The github repository can be found at: <https://github.com/iDanScott/DansWorld>. Another feature of github is that it tracks when commits come through so it’s possible to see when the most active development time on the project was.

A supervisor meeting was organised for the beginning of every week, of which most were attended. Some meetings were not attended as there was either no progress made on the project or there was no topics to be discussed at the meeting. A highlight report was written on the weeks where a meeting was planned to discuss with the supervisor the progress that was made.

## 3.3 Design Changes

* Comparing the resulting features to the features outlined in the initial PID

# 4. Deliverables

Appendix A Section 4.1 has detailed all of the initial planned deliverables that were laid out in the initial project plan. In the below sections, there will be details of what was completed, what was not and what was modified.

To visually explain these, a table with each feature will be provided with the following key:

|  |  |
| --- | --- |
| Key | Description |
| Original Plan | The deliverable was completed to the specification of the original deliverable with little or no modifications. |
| Functionality Modified | The deliverable was completed for the purpose that it was originally designed for, but has additional or less functionality than what was originally planned. |
| Removed / Uncompleted | The deliverable did not meet the requirements of the system that was unforeseen and was therefore removed in favour of an added feature, or was not completed. |

## 4.1 Core Deliverables

### 4.1.1 Table of Core Deliverables

|  |  |  |
| --- | --- | --- |
| ID | Description | Type |
| 1.a | A game client which is capable of drawing sprites and other visual elements for human interaction | User, System |
| 1.b | The game client should establish a network connection between the server and itself, to allow communication between multiple clients | System |
| 1.c | Basic combat functionality | System |
| 2.a | Game server should be able to handle hundreds of concurrent connections if not thousands, distributing the correct messages to the correct clients | System |
| 2.b | Game sever should back up the state of all connected characters periodically | System |
| 2.c | Game server should handle registration of game accounts and creation of characters, generating SQL queries to store the given information to the database in an encrypted fashion | System, User |
| 2.d | Moderation requests should be handled by the game server to punish players breaking the rules. Bans or mutes where appropriate | System, User |
| 3.a | A database capable of large amounts of data holding a stable connection to the game server for periodic updates | System |
| 3.b | The database should hold Account, Character, NPC, Item and Map Data | System |
| 3.c | Reasonably Portable to enable local development and remote live copies of the database to be in use | System |
| 4.a | Hostile NPCs should have some logic behind them to provide some challenge for the player during combat | System, User |
| 4.b | NPCs which are not hostile, typically vendors and quest givers should have access to dialogue systems to enable the player to interact with them | System, User |

### 4.1.2 Clarification of Changed Deliverables

|  |  |
| --- | --- |
| ID | How it was changed |
| 2.a | The issue with this one point which was not foreseen at the time of planning the project was how to exactly test that number of connections. The biggest factor in this is the hardware limitation with how many clients could be run concurrently. The highest number that was achieved concurrently though testing was 3 clients. |
| 3.a | This point was again not quite specific enough on the set out, as quantifying what constitutes to large amounts of data varies greatly depending on the context. According to the MySQL database documentation it’s largely dependent on the hardware that is hosting the database with up to 256TB per table[[1]](#footnote-1). |
| 3.b | This point was met almost fully, however the item dropping mechanic was never integrated into the game as much of the development was held back by the lack of visual assets and design documents. |
| 4.a | For the same reason as 3.b, the NPCs combat mechanics were never implemented, however they do move randomly separately from one another and have death/respawn mechanics. |

### 4.1.3 Clarification of Removed Deliverables

|  |  |
| --- | --- |
| ID | Why it was removed |
| 2.d | UI elements were not completed to allow the game client to show reporting options for players, and systems to point out accounts as moderator accounts was also not completed |
| 4.b | Systems to allow NPCs to not be hostile was not completed as it wasn’t seen to be a priority over the hostile NPCs. |

## 4.2 Desirable Features

The desirable features in appendix A.4.2 were not completed as they were predominantly features that fleshed out the gameplay. While the gameplay is important, it wasn’t the primary focus of the project which was the networking technology.

# 5. Quality Control

* Testing plans

# 6. Project Plan

The initial project plan which can be found in the PID in Appendix A.7 details 8 different phases of development. This was put together in week stints, with some a little more than two weeks, for different tasks that were planned for each phase. Some tasks were given more time as they either were thought to have taken more research or time to flesh out into their purpose. For the most part this plan was not kept to, as design work was held back because some features were not completely thought through from the outset, and art assets were much harder to come by than was previously anticipated. Having a bespoke graphics artist would have served the project much better than relying on open source assets.

## 6.1 Actual Project Timeline

In the table below, there are details of the chronological order of events that happened during the project timeline. The github commit log was used as a cross reference to produce the information below.

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Start Date | End Date | Deliverables Produced |
| 1. PID | 26th Jan | 2nd Feb | Produced the project initiation document defining the features of the project |
| 2. Phase 1 | 2nd Feb | 8th Mar | Drew up a few revisions of the Client and Server UML, ironing out all the core features of the system to enable network communication |
| 3. Phase 2 & Phase 3 | 8th Mar | 12th Mar | Created client and server network communication applications. Purchased VPS for testing Latency. |
| 4. Phase 4 | 12th Mar | 13th Mar | Normalised database and deployed MySQL server to remote VPS. |
| 5. Phase 5 | 13th Mar | 14th Apr | Main bulk of the game client development. Added enemies into the game, no real enemy logic was put in place as the combat mechanics were not implemented therefore they moved randomly. |
| 6. Phase 3 revisit | 14th Apr | 10th May | Continued building on the client, implementing the network code to the graphical game client and enabled registrations, character creations and logging in to the game. |
| 7. Phase 8 | 14th Apr | 10th May | Various touch ups to the game server, added stability by making the file logging optional. |

## 6.2 Risk Plan

* The risk plan found in appendix A section 8

# 7. System Design

This section will be covering all the processes that were involved in creating the core systems and sub systems in terms of the object-oriented classes included in the two solutions. On the outset many of these systems were rough guidelines for the project and had some changes and refinements to get them to the point where they are at for submission.

As a note, at any stage in the following section, when talking about characters, it is referring to the avatar of an in-game entity, or agent, that is controllable by the game client. When talking about players, it is referring to a character that is being controlled in the game world by the game client through human inputs.

## 7.1 Network Protocol Design

During the planning and design phase 1, it was found that the preferred socket communication method is by sending arrays of bytes to a connected socket[[2]](#footnote-2). To allow for a system where byte arrays can be categorised by the type of message that is sent in the lowest byte count. The way in which the network protocol was envisioned to work based on this idea, was through a series of request and response pairs. This would mean that for example the client would request a log in to the game server with 2 bytes identifying that array of bytes as a login request follow by; a username and a password as strings in the byte array payload. The server would then respond with either accepted or rejected byte flag, along with a byte identifier of the fact that it’s replying to a login request, based on the account information it evaluated.

### 7.1.1 Packet Design

A packet payload in the context of Dan’s World will be defined by firstly 1 byte which gives the length of the packet in a number of bytes. If the byte received gives the value of 100 then the byte array that can be expected will be 100 bytes long and the system will therefore keep attempting to read bytes until it has 100 bytes in an array to construct a packet with. Once the packet knows how big it has to be, it then expects 2 1-byte flags to begin its payload as defined in the sections below.

### 7.1.2 Packet Byte Identifiers

**Packet Family -** The packet family defines the category of the packet which helps to group the request/response pairs

|  |  |  |
| --- | --- | --- |
| Type | Value | Description |
| LOGIN | 1 | Login packet identifier. |
| REGISTER | 2 | Registration packet identifier. |
| PLAY | 3 | Packet identifier for a request to bring a character into the playing world |
| PLAYER | 4 | Packet identifier for packets that belong to any actions of a player. |
| CONNECTION | 5 | Ping/Pong cycle packet identifier. |
| ENEMY | 6 | Packet identifier for any packets that belong to the actions of an enemy defined by the server |
| CHARACTER | 7 | Packet identifier for packets that belong to any actions of a character or agent. |
| SERVER | 8 | Packet identifier for packets that originated from the server, primarily announcements |

**Packet Action -** The packet action is secondary to the family and brings further context to what the packet payload contains.

|  |  |  |
| --- | --- | --- |
| Type | Value | Description |
| REQUEST | 1 | Used to identify packets that are a request to the server for logging in and registration. |
| ACCEPT | 2 | A request has been accepted by the server. |
| REJECT | 3 | A request has been rejected by the server. |
| MOVE | 4 | A player or NPC moved within the game world |
| STOP | 5 | A player or NPC has stopped moving within the game world |
| WELCOME | 6 | A player has logged in to the game |
| LOGOUT | 7 | A player has logged out of the game |
| PING | 8 | A ping packet for the server to send to the client to test whether the connection is still live. |
| PONG | 9 | A pong packet for the client to send back to the server when a ping is received to clarify with the server that the connection is still live. |
| TALK | 10 | A player has sent a text message to the game server and therefore will be relayed to every connected client. |
| ATTACK | 11 | A player has sent an attack request for the server to determine if any enemies were hit |
| CREATE | 12 | The creation of in game entities such as characters will be encapsulated in a create action |
| DELETE | 13 | Requests to delete characters will come with the delete action |
| CREATED | 14 | After the creation of a character, the confirmation will come in a created packet |
| ALREADY\_EXISTS | 15 | An already exists action will come upon the request to make a character with a name that already exists |
| TAKE\_DAMAGE | 16 | Used to reduce the health bar of a character or enemy |

### 7.1.3 Packet Request and Response Pairs

In the below section there will be a list of the Packet Family and Packet Action pairs which identify the packet in the format of “PacketFamily.PacketAction” followed by a description of their payload and the packets that are expected to be received in return.

|  |  |
| --- | --- |
| **LOGIN.REQUEST** | |
| Type | Client-To-Server |
| Parameters | Username length  Username  Password length  Password |
| Expected response(s) | LOGIN.ACCEPT  LOGIN.REJECT |
| Why the packet is sent | Login requested by the client |

|  |  |
| --- | --- |
| **LOGIN.REJECT** | |
| Type | Server-To-Client |
| Parameters | Reason why the login was rejected |
| Expected response(s) | None |
| Why the packet is sent | Invalid password  Invalid username  Account is already logged in |

|  |  |
| --- | --- |
| **CHARACTER.CREATE** | |
| Type | Client-To-Server |
| Parameters | Length of the given character name  Character name  Gender |
| Expected response(s) | CHARACTER.ALREADY\_EXISTS  CHARACTER.CREATED |
| Why the packet is sent | Character was created on the client |

|  |  |
| --- | --- |
| **CHARACTER.CREATED** | |
| Type | Server-To-Client |
| Parameters | Length of the given character name  Character name  Gender |
| Expected response(s) | None |
| Why the packet is sent | Character name was available to use |

|  |  |
| --- | --- |
| **CHARACTER.ALREADY\_EXISTS** | |
| Type | Server-To-Client |
| Parameters | Character already exists message |
| Expected response(s) | None |
| Why the packet is sent | Character name was not available to use |

|  |  |
| --- | --- |
| **REGISTER.REQUEST** | |
| Type | Client-To-Server |
| Parameters | Username length  Username  Password length  Password  Email length  Email  Full name length  Full name |
| Expected response(s) | REGISTER.ACCEPT  REGISTER.REJECT |
| Why the packet is sent | Account registration request from the client |

|  |  |
| --- | --- |
| **REGISTER.ACCEPT** | |
| Type | Server-To-Client |
| Parameters | Username length  Username |
| Expected response(s) | None |
| Why the packet is sent | Account was created. Sent username back to the client so they are aware of the username post account creation. |

|  |  |
| --- | --- |
| **REGISTER.REJECT** | |
| Type | Server-To-Client |
| Parameters | Message as to why the account creation was rejected |
| Expected response(s) | None |
| Why the packet is sent | The username already exists |

|  |  |
| --- | --- |
| **PLAY.REQUEST** | |
| Type | Client-To-Server |
| Parameters | Character ID to play (1 – 3) |
| Expected response(s) | PLAY.ACCEPT  PLAYER.WELCOME  ENEMY.WELCOME |
| Why the packet is sent | Once logged in to the game the player can select between up to 3 characters to play |

|  |  |
| --- | --- |
| **PLAY.ACCEPT** | |
| Type | Server-To-Client |
| Parameters | Byte array of every character object logged in to the game including names, levels, gender, x, y and direction of facing. |
| Expected response(s) | None |
| Why the packet is sent | The player has logged in and needs to be able to see the other currently logged in players. |

|  |  |
| --- | --- |
| **PLAYER.WELCOME** | |
| Type | Server-To-Client |
| Parameters | Byte array of a character object including names, levels, gender, x, y and direction of facing. |
| Expected response(s) | None |
| Why the packet is sent | A player may have entered a map with other players on it, or a player may have logged in to the game and therefore the player needs details of the other players. |

|  |  |
| --- | --- |
| **ENEMY.WELCOME** | |
| Type | Server-To-Client |
| Parameters | Byte array of an enemy object including names, levels, x, y and direction of facing. |
| Expected response(s) | None |
| Why the packet is sent | A player may have entered a map with other players on it, or a player may have logged in to the game and therefore the player needs details of the enemies. |

|  |  |
| --- | --- |
| **PLAYER.MOVE** | |
| Type | Client-To-Server and Server-To-Client |
| Parameters | X location of the player  Y location of the player  Direction of the player  ID of the player |
| Expected response(s) | none |
| Why the packet is sent | A move request was made to the server by a client, therefore all other clients are notified of the move if the move was valid. |

|  |  |
| --- | --- |
| **PLAYER.STOP** | |
| Type | Client-To-Server and Server-To-Client |
| Parameters | X location of the player  Y location of the player  Direction of the player  ID of the player |
| Expected response(s) | none |
| Why the packet is sent | A player has stop moving and the server is notified, therefore all other clients are notified of the character no longer moving. Helps with animations. |

|  |  |
| --- | --- |
| **PLAYER.LOGOUT** | |
| Type | Client-To-Server and Server-To-Client |
| Parameters | ID of the player |
| Expected response(s) | none |
| Why the packet is sent | A player logged out of the game and therefore all clients are notified |

|  |  |
| --- | --- |
| **PLAYER.ATTACK** | |
| Type | Client-To-Server |
| Parameters | none |
| Expected response(s) | ENEMY.MOVE (if killed)  ENEMY.TAKE\_DAMAGE |
| Why the packet is sent | A player requested to attack at their current position and in the direction they are facing. |

|  |  |
| --- | --- |
| **ENEMY.MOVE** | |
| Type | Server-To-Client |
| Parameters | ID of the Enemy  X location of the Enemy  Y location of the Enemy  Direction of the Enemy |
| Expected response(s) | none |
| Why the packet is sent | An enemy was killed and position was reset  An enemy moved randomly |

|  |  |
| --- | --- |
| **ENEMY.TAKE\_DAMAGE** | |
| Type | Server-To-Client |
| Parameters | ID of the Enemy  Health of the Enemy |
| Expected response(s) | none |
| Why the packet is sent | An enemy took damage from a player in response to an attack request |

|  |  |
| --- | --- |
| **CONNECTION.PING** | |
| Type | Server-To-Client |
| Parameters | Current date/time stamp |
| Expected response(s) | CONNECTION.PONG |
| Why the packet is sent | In an attempt to not keep closed/dead clients in the game a ping/pong cycle is executed to test for alive network connections. |

|  |  |
| --- | --- |
| **CONNECTION.PONG** | |
| Type | Client-To-Server |
| Parameters | Current date/time stamp |
| Expected response(s) | none |
| Why the packet is sent | In an attempt to not keep closed/dead clients in the game a ping/pong cycle is executed to test for alive network connections. |

### 7.1.4 Packet design limitations

Dealing with network communications in this way comes with a few limitations. The first of which is the fact that a packet cannot be any larger than 255 bytes as noted by MSDN[[3]](#footnote-3) since the first thing we take is a byte count of the number of elements in the packet payload, and the value of said byte can be no larger than the data types max value (255). Another limitation of this system is the number of combinations for Packet Family and Packet action pairs. With the value of a byte being 255, we can have up to 255 unique identifiers for both Family and Action, and a potential 2552 (65,025) different combinations of the two, which is a large number but unlikely to have a single packet family with every potential action to match.

## 7.2 UML Diagrams

UML diagrams were a great starting point for paving the way for the code base. A couple UML Diagrams were created, with many of the features not quite being able to get ironed out at the start of the planning phase. The initial diagram of the server can be seen in Figure 2, and the client in Figure 3. Many of the features for the game client were not able to be realised at the time of planning since there was very little prior knowledge of games programming, so some sub-systems of the client are not included in the initial diagrams. All final UML diagrams can be found in Appendix B.

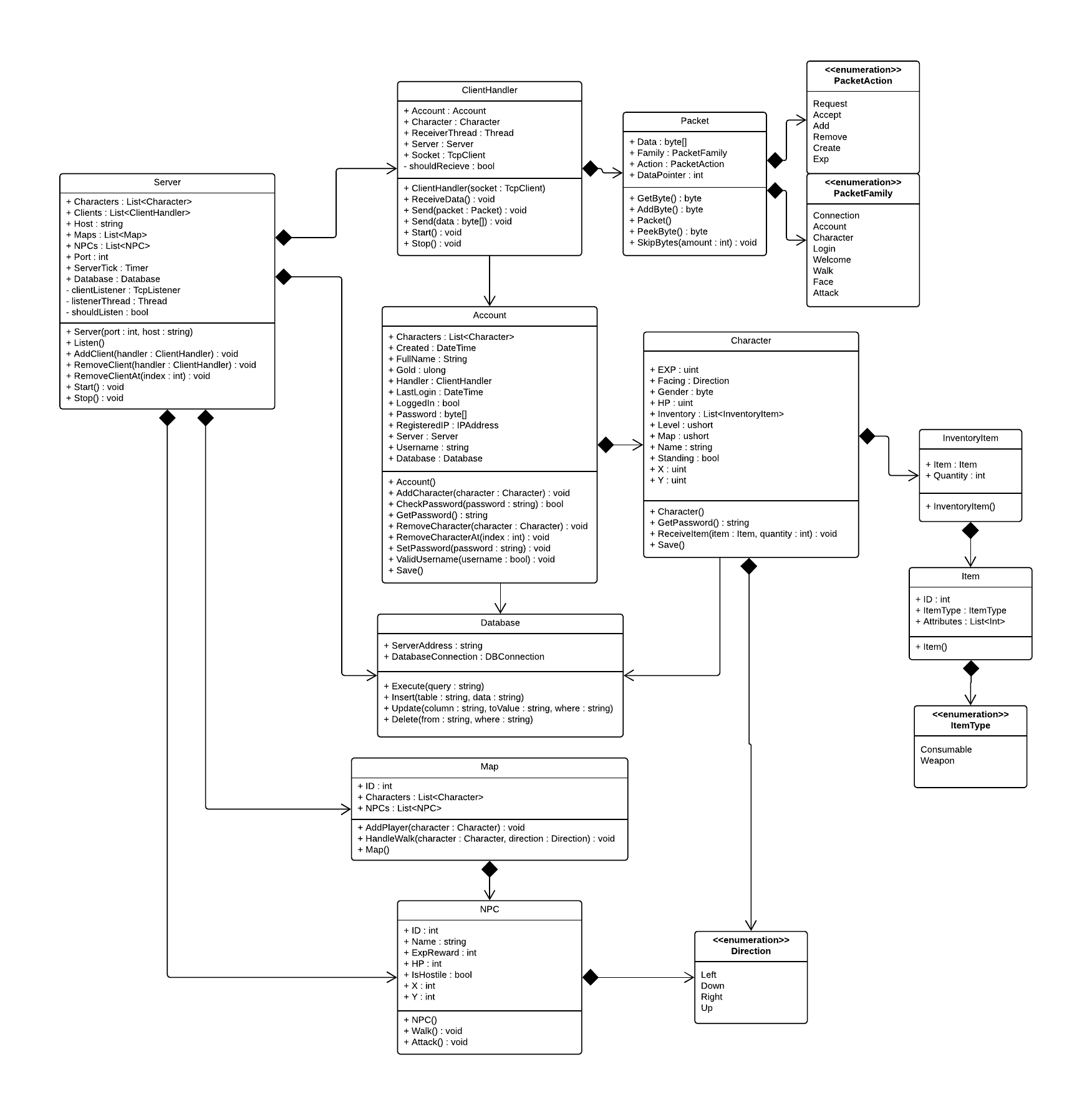


Figure 2 - Initial UML for the server

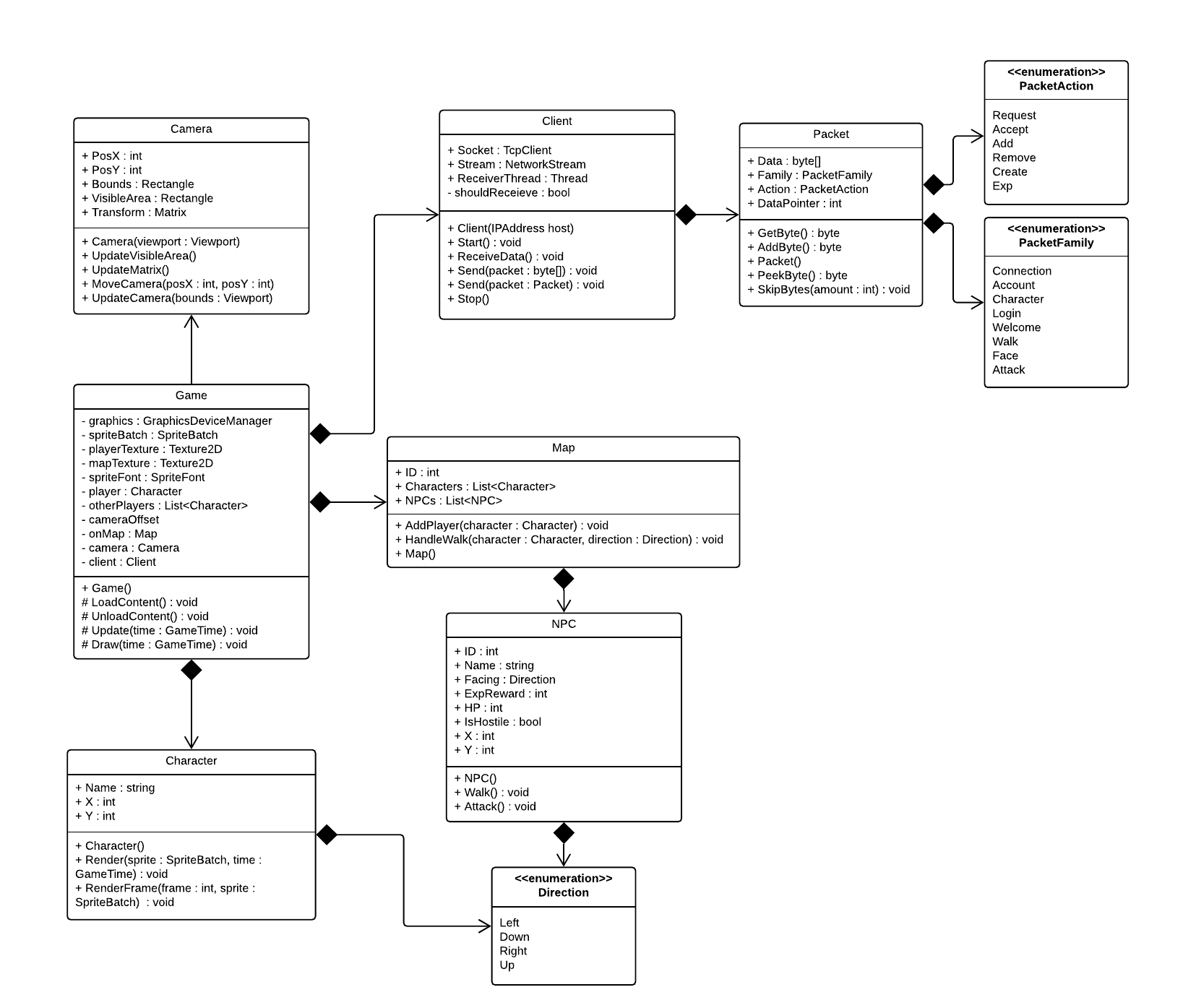


Figure 3 - Initial UML of the client

## 7.3 Database Design

From the outset it was very clear that not a whole lot of data would need to be saved to the database. In the context of multiplayer online games, the vast majority of the data is in the visual and audio game assets which ship packaged in the game client. In the context of Dan’s World, only player data needs to be saved such as the X and Y locations of the player when they logged out among other things.

### 7.3.1 Database Normalisation

Having a normalised relational database when embarking on making an online multiplayer game is incredibly important when bringing scalability into question. Should the player base skyrocket at any point, the database needs to be able to cope with the volume of data, and removing redundancies is a great way to ensure the maximum potential of any relational database. “the goal of data normalization is to reduce and even eliminate data redundancy”[[4]](#footnote-4).

In order to begin normalising, a list of data that would need to persist across play sessions and server downtime was put together. Some groups of data did not pose any relation to one another so two groups of normalisation was carried out to produce two relational entities that were not related to themselves as seen in the ERD below. The normalisation process detailing UNF all the way through to 3NF can be found in the google sheet at this address:

<https://docs.google.com/spreadsheets/d/1dsHPd8MgMSdynIMVNWvp47zQVugGjwg_Q0GGtPHgpho/edit?usp=sharing>

Upon

### 7.3.2 Database ERD

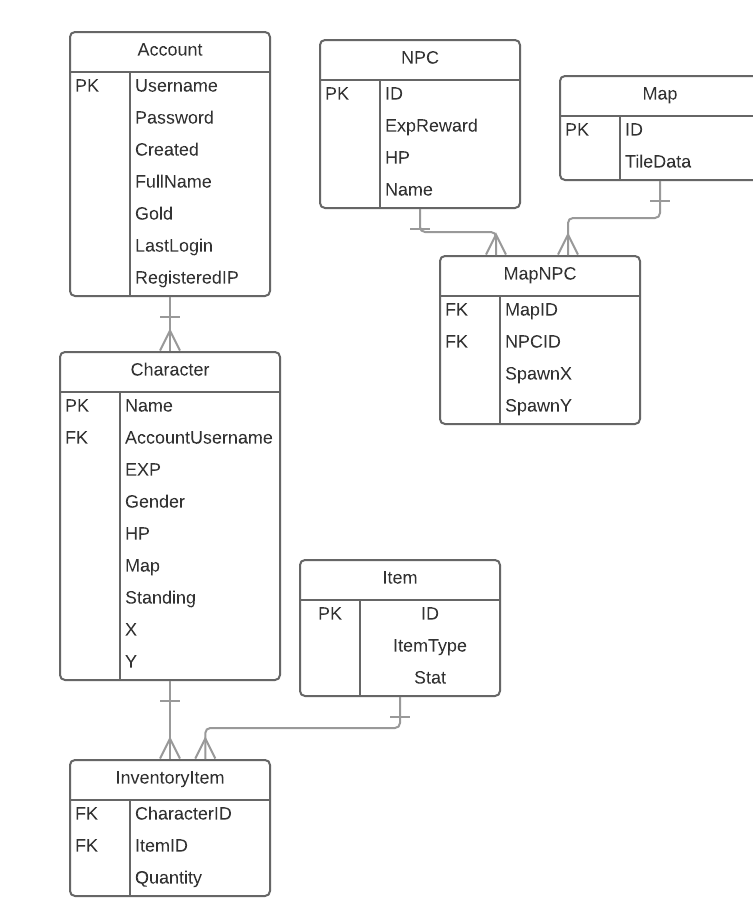


Figure 4 - Database ERD

# 8. Graphics Design

* The graphics used in the project came from internet sources as time constraints on the project and lack of a third-party artist did not allow for bespoke graphics to be created for the game. Continuing with this project past this project will mean that graphics for Dan’s World will have attention given to them.

# 9. Implementation

## 9.1 Database Implementation

* Database implementation, MySQL Server using the normalisation laid out in section 7

## 9.2 Server Implementation

* Server Implementation
  + handling of incoming connections and packets

## 9.3 Game Client Implementation

* Client implementation
  + handling of outgoing connections and packets
  + Construction of UI components such as TextBox, Label and Button, taking an approach very similar to that of WinForms whereby all these components inherit from a Control class which enforces Update and Draw functionality.
  + State Machines for ClientState and how they affect the running of the game

# 10. Lessons Learned

# 11. Conclusion

# Appendices

# A Project Initiation Document (PID)

# A.1 Introduction

For this project I will be acting as the client, as a video games enthusiast. The idea for this project is to create an MMORPG, which is the common abbreviation of a **M**assively **M**ultiplayer **O**nline **R**oleplaying **G**ame. The architectural requirements of such a project can be broken down into two relatively simple components. The first of which is a **Game Server** which handles the multiple connections (or Massively Multiplayer) aspect via the use of network protocols, and the second being the **Game Client** which responds to server messages and sends messages to the Game Server. The aim of this is to create an online environment in which video game enthusiasts can sit and play with one another to complete objectives and goals, either together or alone.

The motivation for wanting to develop such a software package comes from the source of my interest in programming. Ever since the age of 13, I have been playing around with code as a result of a video games company called Endless-Online discontinuing the development of their small-scale MMORPG, which caused the community to set up their own server emulation software to enable the franchise to continue. The game is in a “2.5d” artistic style which for this project I will be going away from and using a flat 2D style similar to that of the incredibly popular Pokémon games by Game Freak.

# A.2 Business Case

## A.2.1 Business Need

Currently on the market there is an exceedingly large collection of MMORPG’s that take on the same 2D aesthetic, some rocketing to popularity such as Habbo Hotel initially released in 2000 by Sulake as a browser based game, still going to this day. However, many of these games have enhanced playing experiences through microtransactions or paywalls, whereas for this project I would be able to develop a game to be played without any microtransactions or paywalls. Another aspect of these many free to play MMORPG’s is that they often have very cluttered user interfaces which massively over complicate the interaction within the game for the players.

My main motivation for picking this project, is that during my university career I have not been able to really explore what the ceiling of my programming ability is, when taking into consideration many different aspects of software, and indeed video game development, so this would be a great opportunity for me to explore some new technologies and a very different discipline. Network protocol engineering has always been fascinating to me seeing how computers from anywhere in the world can communicate with one another, which is another area that this project will allow me to explore.

The problem that the project hopes to resolve is the problem of having many hundreds if not thousands of concurrent connections to a server socket. This would have to be handled in a multithreaded synchronous environment, as having multiple connections to a single thread with a single socket would mean some messages would either be scrambled or be processed long after the message was sent.

## A.2.2 Business Objectives

To develop a Massively Multiplayer video game, playable on a windows desktop environment, achieving the following:

1. A Massively Multiplayer Online game, handing hundreds if not thousands of concurrent connections to a live game server
2. Character customisation and progression, such as gear looting and levelling up for the player
3. Social interaction among the connected players, building an online community which is safe and moderated
4. A database backend to store account data, character progression, hostile/friendly NPC (Non-Playable Character) data and world/map data.
5. The player should be able to control and move their character using keybinds and activate interaction with the enter key to send messages
6. There should also be roles within the game to assign to players, enabling moderation tools such as bans and mutes for abusive/exploitive players

# A.3 Project Objectives

To develop a Massively Multiplayer video game with the following functionality and properties:

1. A persisting online world which enables players to register, connect, play (control their character using the game client) and disconnect.
2. A basic combat system to fight hostile NPCs, gaining the player experience and the chance to level up to improve combat skills and presence.
3. A log in screen where the user can decide to either log into the game, create an account, choose a character to play or delete a character.
4. Basic social interaction between other players and NPCs.
5. A loot system granting a variety of bonus items to the player upon defeating hostile NPCs
6. A basic questing system to make for a more immersive gaming experience
7. Inventory system to keep track of what the player owns.
8. A database to store the progression of the characters and the state of the game upon player disconnect or server shutdown.

# A.4 Initial Scope

## A.4.1 Core Deliverables

1. The Network enabled Game Client
   1. Establishing a connection between server and client is the most fundamental part of the package, without it, it will no longer be a massively multiplayer game, removing much of the complexity.
   2. Developing a Game Client in a framework which is well documented for ease of development with also included functionality for network communications.
   3. 2D environment with basic combat functionality.
2. The Game Server
   1. The game server should be able to handle hundreds if not thousands of concurrent connections distributing the correct messages to the correct clients as and when its needed.
   2. The game server should back its state up to the database periodically in case of power failure or system crash.
   3. The game server should handle registration of game accounts and creation of characters, generating SQL queries to store the given information to the database in an encrypted fashion.
   4. Moderation requests should also be handled in the game server to punish players breaking the rules. This can be done with bans or mutes.
3. The Database
   1. A database capable of large amounts of Game Data, holding a stable connection to the game server for periodic updates.
   2. Expected to hold Account, Character, NPC, Item and Map Data.
   3. Reasonably portable to enable local development and remote live copies of the database to be in use.
4. NPCs
   1. Hostile NPCs should have some logic behind them to provide some challenge for the player during combat.
   2. NPCs which are not hostile, typically vendors and quest givers should have access to dialogue systems to enable the player to interact with the NPC.

## A.4.2 Desirable Features

Below I will detail the features and functionality of the project that would be great to have included with the project, but not necessarily have dedicated development time for. These will be features that will add to the depth of the game once the core foundation of the project is in place and it’s a mere case of “bolting on” features to make it a more complete package. Having this in mind will also enable me to make sure that I develop the package in a way which is modular and easy to work with at a high level, away from the core.

1. Player vs Player (PvP)
   1. It’s very common to find in most MMORPG’s on the market, some kind of Player vs Player environment where players can go against one another as a test of skill to see who is superior in their combat abilities.
   2. Maybe having multiple races would be a good way to put certain players against others as they would then have motive.
2. Class System
   1. Another common thing to find in other MMORPG’s is the ability to pick a class which defines what combat abilities you will have, and what role you will serve.
3. Bespoke Graphics
   1. Due to the nature of the project, everything the end user will see will be through the game client itself, which will be engineered in a way that means everything that will be seen, will be delivered through images. As I’m not very artistically inclined, it will take me quite some time to put very basic assets together, which is why I would go down the route of finding assets to use online.

# A.5 Project Resources and Dependencies

At this time, I do not plan to develop the assets for the game myself so I will be seeking free licenced assets which I can use for the development of my game client. As I believe there to already be a lot of resources that are free to use on the internet, I do not consider this to be an issue.

# A.6 Method of Approach

Software development will be carried out with the following steps

1. This stage will be devoted to assessing what elements of the project will be needed to get the core functionality working and playable. It will be used to flesh out the core features such as the model for connecting to a live server socket with a client socket, the model for the world, the enemies that persist within the world, how the world interacts with the server, the player of the game and finally the database for storing the player data. Design of the network protocols with requests and responses can also be expected in this stage. This stage will focus on the main areas of the project being: the game world within the server, the players and how they are structured within the server, the enemies controlled by the server and the core game logic both server side and game client side. Initial UML class diagrams and sequence diagrams will be produced where relevant in the previously specified areas. We expect these planning stage documents to change over the development of the project as they are just a starting point for development.
2. Construction of both the server network protocols and the client network protocols, ensuring that they are stable and capable of network communication across several different hosts. As this is the most fundamental part of the project, modularity and scalability is paramount and will be focused on and tested so that it is stable before continuing with development of the game client and any other features. The network architecture will be developed in the .NET framework with C# using as many of the latest .NET Core features as possible to allow for cross platform portability and hosting on an external VPS for optimal uptime.
3. Construction of the game client foundations to establish connection to the game server and initialise in game resources to view other players within a persisting world. During this development stage, the network protocol designed in stage 1 will be implemented and fleshed out until it is at a stage where the game can be played by multiple clients. Core gameplay of the client and player mechanics will also have their initial implementation at this phase. For the game client MonoGame will be the choice of framework as it means the technology can still be kept within C#. MonoGame is an open source initiative of framework which is what came from Microsoft’s XNA project.
4. Database development and storage of player and account data will be the focus of this phase, to enable visible progression of the player stored across separate play sessions. This will then integrate with the server to give the player access to his/her account data when returning to the game.
5. Implementation of enemies and their movement and combat logic so that they will provide players with adequate challenge and rewards with loot tables and experience for the progression of the characters.
6. Implementation of moderation functionality to remove players from the game for rule breaking and to test development features within the game such as content skipping.
7. Final phase of development focusing on the refinement of in game mechanics, enemy encounters and art assets if time allows.
8. Desirable features will be the last focus of the project after every other deliverable has been met and there is extra time left for polish as given in section 4.2.

# A.7 Project Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Stage | Expected Start Date | Expected Completion Date | Deliverables to be produced |
| 1. Initiation | 22nd January | 2nd February | PID Final Draft |
| 1. Phase 1 | 2nd February | 16th February | Initial plans for network protocol designs (requests / responses), initial database entity relationship diagrams to be drawn up, class diagrams for both server and client objects. |
| 1. Phase 2 | 16th February | 23rd February | Main development of the core functionality. Server and Client protocols to be implemented and tested. Deployment to a remote host would be beneficial to test the performance of the network architecture. |
| 1. Phase 3 | 23rd February | 9th March | Development of Game Client using initial designs for UI and sprites. Sprites are to be found using open licenced assets online. This time will also be used to investigate the MonoGame framework and produce a working networked prototype. |
| 1. Phase 4 | 9th March | 16th March | Database normalisation and implementation on an SQL server selected during the investigations in phase 1. Once complete, integration to the server and communications to and from the server are to be expected before the completion date of this phase. |
| 1. Phase 5 | 16th March | 2nd April | Implementation of enemy logic in the server and communicating enemy locations and abilities to the connected game client(s). |
| 1. Phase 6 | 2nd April | 5th April | A few days dedicated to moderation functionality for in game moderators, developers and testers. |
| 1. Phase 7 | 5th April | 9th April | Final phase of development for touching up the game client and server for any additional features before the last testing phase. |
| 1. Usability and testing | 9th April | 12th April | Testing and Evaluation. User testing of existing features and playability. This phase will be used to target features of the project, either existing features or the addition of new features detailed in desirables or features requested by user feedback. |
| 1. Phase 8 | 12th April | 20th April | Development of the project for the final touches and desirable deliverables. This will be largely using the features that were highlighted as needing to be reworked by the user feedback or requested features to enhance the user experience. |
| 1. System and user acceptance testing | 20th April | 27th April | Testing of the final product. This will be used to ensure that no horrible crashes occur and the player experience is smooth and seamless. |
| 1. Final Report | 27th April | 4th May | PROCO304 Draft Report |

## A.7.1 Phase Management

Phase management will be carried out by assessing what has been achieved by the end of the week during a phase, if all deliverables have been met by the stated deadline, then there is nothing to be done. If a phase is planned for less than a week or takes less than a week to complete, then the report is to be produced upon completion and plans to be re-shifted back to allocate more time for other phases which are likely to take more time than others. If a phase takes longer than intended then a re-prioritisation will take place to ensure the project is still on track for the final deadline, making sure that the project package is as complete as it can be. A report is to be produced of the stage detailing what was achieved and what had to be re-evaluated if anything.

## A.7.2 Control Plan

The following PRINCE2 control techniques will be employed:

1. Highlight reports as dictated by the PROCO304 module
2. End of phase reports
3. Review meetings with the project supervisor Mr. Martin Beck, as dictated by the PROCO 304 module; additional ad-hoc meetings as necessary
4. Risk management (see section 8); communication plan (see section 7.3); quality plan (see section 9); exception reports and plans as necessary

## A.7.3 Communication Plan

Review meetings will be held with the supervisor in line with the Control plan. Further ad-hoc communications will take place as needed.

# A.8 Initial Risk Assessment

|  |  |
| --- | --- |
| Risk | Management Strategy |
| Schedule Overrun | The schedule has been constructed in such a way that I have accounted for the possibility that it will be overrun, and therefore can be re-evaluated along with the scope of the overall project. An exception plan will be developed and approved by the project supervisor in the event of more than 1 weeks slippage. |
| Troubles learning the required technologies | A very simple system prototype will be developed during phase 1 to test the networking technologies and the MonoGame framework. |
| Failure to find required assets online | Should there be a struggle to find the assets online to create a basic, aesthetically cohesive interface, some basic placeholder assets will be developed until such a time that they are replicable. |
| Loss of database (Technology Failure) | The project will be zipped and backed up daily to github and dropbox as well as keeping local copies on a separate hard drive from the primary system drive. |
| Trouble connecting to remote server when usability testing | A remote windows machine will be running at all times during the usability demo outside of the university network to ensure that the university firewall is not the cause of the issue when testing the latency on the remote host. During the usability testing a local copy of the server will be deployed allowing the clients to connect to either a university machine or my own personal device. |
| Illness or family emergencies | In the event of illness or a family emergency, the project supervisor will be contacted immediately. If more than 1 week of development time is lost putting the deadlines at risk, an exception plan will be drawn up and agreed with the stated project supervisor and an application for extenuating circumstances will be produced. |
| Difficulties expanding the depth of the persistent world | Should the game itself lack depth and content for the player to enjoy, the project will simply have its testing focus to be shifted back to the networking stability, ensuring that even with a high number of clients the network protocol is stable. |
| Loss of online backup | Keeping the daily local backups should allow for any failures in either of the proposed online backup solutions. |

# A.9 Initial Quality Plan

|  |  |
| --- | --- |
| Quality Check | Strategy |
| Requirements | Requirements will be checked to ensure that they are correct and relevant. Requirements should also be traceable to their respective business objectives, complete, achievable and demonstrable. The requirements will also document required product quality criteria, most notably for usability. Prototypes will be built and tested by friends and colleagues. |
| Design validation | The design will be evaluated at every development phase of the game client where appropriate and checked against requirements compliance, HCI guideline compliance, UI design acceptance, database normalisation and software design principles. The design that is to be decided on will be tested with users as frequently as time allows, and any features that are requested as a result of the tests will have time allocated for them during the last phase of development at stage 10. |
| Sub-system Verification and validation | To be conducted at the end of each stage of the development timeline. This includes testing and quality control. Details of these will be outlined in the end of stage reports. |
| System and user acceptance | To be conducted within stage 10 |

# A.10 Legal, Ethical, Social and/or Professional Issues

The main concern with this project is ensuring that all art assets are to be taken from open licenced platforms, so that they do not infringe any copyright agreements. This also includes assets which are distributed under a non-profit or academic licence. Should any assets come into question, they will be removed from any section of the project where they are made use of.

## A.10.1 University of Plymouth Ethics Policy

The system evaluations that will be taking place at 2 different stages of development will involve human play-testers, however there are plans in place to set up test accounts for the prospective users and wipe any data gathered from the servers during the play testing session. This means that there is no need for any special documentation to be drawn up between either party of the play testing session.

# B UML Diagrams

1. MySQL Database Size Limitations - <https://dev.mysql.com/doc/refman/8.0/en/innodb-restrictions.html> [↑](#footnote-ref-1)
2. MSDN Socket Programming C# Reference - <https://msdn.microsoft.com/en-us/library/w93yy28a(v=vs.110).aspx> [↑](#footnote-ref-2)
3. MSDN max value for a Byte - <https://msdn.microsoft.com/en-us/library/system.byte.maxvalue(v=vs.110).aspx> [↑](#footnote-ref-3)
4. Agile Data on Normalisation - <http://agiledata.org/essays/dataNormalization.html> [↑](#footnote-ref-4)