

School of Computing, Electronics and Mathematics

PRCO304

Final Stage Computing Project

2017/2018

BSc (Hons) Computer Science

Dan Scott

10501358

Dan's World

Supervisor: Mr Martin Beck
Second Marker: Dr Serge Thill

Abstract

This report will outline the stages that were required to create a 2D Networked Online Roleplaying game, 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 operating system, be that windows, Linux or MacOS. To store the data, a MySQL Database server was used save character progression for the same reason as .NET Core's portability across platforms. Github was used for source control. Lastly MonoGame which is a games programming framework written for C# was used for the games client. MonoGame is a project that was inherited from the Microsoft XNA team.

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

Contents

Abstract.....	1
1. Introduction	4
1.1 <i>Dan's World</i>	4
1.2 <i>Project Goals</i>	4
2. Legal, Social and Ethical Issues.....	5
2.1 <i>Game Assets</i>	5
2.2 <i>Data Collection</i>	5
3. Project Management	5
3.1 <i>Method of Approach</i>	5
3.2 <i>Data and Time Management</i>	5
4. Deliverables	6
4.1 <i>Core Deliverables</i>	6
4.1.1 <i>Table of Core Deliverables</i>	6
4.1.2 <i>Clarification of Changed Deliverables</i>	7
4.1.3 <i>Clarification of Removed Deliverables</i>	7
4.2 <i>Desirable Features</i>	7
5. Quality Control	8
6. Project Plan	8
6.1 <i>Actual Project Timeline</i>	8
6.2 <i>Risk Plan</i>	9
7. <i>System Design</i>	9
7.1 <i>Network Protocol Design</i>	9
7.1.1 <i>Packet Design</i>	9
7.1.2 <i>Packet Byte Identifiers</i>	10
7.1.3 <i>Packet Request and Response Pairs</i>	11
7.1.4 <i>Packet design limitations</i>	15
7.2 <i>UML Diagrams</i>	16
7.3 <i>Database Design</i>	17
7.3.1 <i>Database Normalisation</i>	17
7.3.2 <i>Database ERD</i>	19
7.4 <i>Game Client Design</i>	19
7.4.1 <i>Controls</i>	20
7.4.2 <i>Scene switching</i>	21
7.4.3 <i>In-game Sprites and Animations</i>	22
7.4.5 <i>Unresolved issues</i>	23
7.5 <i>Server Design</i>	23
7.5.1 <i>Concurrency</i>	23
7.5.2 <i>Logging</i>	23
7.5.3 <i>Security</i>	24

8. Lessons Learned.....	24
9. Conclusion.....	25
Appendices.....	26

Table of Figures

Figure 1 - Rendering of the player and enemies. Shows player moving between enemies behind and in front of them.	4
Figure 2 - Initial UML for the server	16
Figure 3 - Initial UML of the client.....	17
Figure 4 – Initial Database ERD.....	19
Figure 5 - Control Interface enforcing behaviours	20
Figure 6 - Scene inheritance	21
Figure 7 - State pattern	22
Figure 8 - Base character sprite sheet	22
Figure 9 - Getting the frame rectangle.....	22
Figure 10 - Sprite Depth Formula	23

Word Count: 6,958

OneDirve source code link: https://liveplymouthac-my.sharepoint.com/:f/g/personal/daniel_r_scott_students_plymouth_ac_uk/ErcIO39foZJAtdn1FrcMZMYBpMx96uvmRRnu-Dxr7t9dRw?e=X8EJaQ

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 and art style influences from the Pokémon franchise.

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 that can be seen 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, new art assets would be created specifically for the purpose of Dan's World. 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. If Dan's World was to be taken into a commercial setting, a user terms of agreement would be drawn up to give permissions of personal data to be collected, for the use of the game experience only.

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 could function as planned 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 issues 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, these can be found in appendix D.

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 .
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.

¹ MySQL Database Size Limitations - <https://dev.mysql.com/doc/refman/8.0/en/innodb-restrictions.html>

5. Quality Control

The quality control of the project was primarily done on the basis of manual testing. When a system was completed it was tested and evaluated against the system requirements. If the system did not meet the whole of its requirements it was re-evaluated and a new system came as a result of the testing.

Unit testing was considered during development however, it was not seen as a help to test code segments against expected outputs as development was carried out with expected outputs in mind. As mentioned before, the systems were manually tested throughout to ensure they produced expected outputs, making use of Visual Studio's debugged tools and console output windows.

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	26 th Jan	2 nd Feb	Produced the project initiation document defining the features of the project
2. Phase 1	2 nd Feb	8 th 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	8 th Mar	12 th Mar	Created client and server network communication applications. Purchased VPS for testing Latency.
4. Phase 4	12 th Mar	13 th Mar	Normalised database and deployed MySQL server to remote VPS.
5. Phase 5	13 th Mar	14 th 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	14 th Apr	10 th 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	14 th Apr	10 th May	Various touch ups to the game server, added stability by making the file logging optional.

6.2 Risk Plan

The risk, control and communication plans can be found in appendix A.8. Even though the project fell behind schedule, risk plans were not produced as time was made up else-where in the project time line.

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 a high enough standard for submission. A full run through of the applications can be found in the user guide, in appendix E.

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 socket². 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 is a collection of data, usually a collection of bytes, that is sent between a source internet address and a destination address, with the data usually being referred to as a payload. 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

² MSDN Socket Programming C# Reference - [https://msdn.microsoft.com/en-us/library/w93yy28a\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/w93yy28a(v=vs.110).aspx)

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. A packet payload is built using a Packet Building class to ensure that the payload is immutable and will not get modified in any way, to ensure data integrity.

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.DELETE	
Type	Client-To-Server and Server-To-Client
Parameters	Id of the character to delete
Expected response(s)	None
Why the packet is sent	Character was deleted by the client and upon deletion the server replies to the

	client to confirm the deletion and is then removed from 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 the Microsoft Developer Network (MSDN)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 255 squared (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.

³ MSDN max value for a Byte - [https://msdn.microsoft.com/en-us/library/system.byte.maxvalue\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.byte.maxvalue(v=vs.110).aspx)

7.2 UML Diagrams

Uniform Modelling Language (UML) diagrams are a great starting point for paving the way for any code base. A couple of UML Diagrams were created, with many of the features not quite being realised 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.

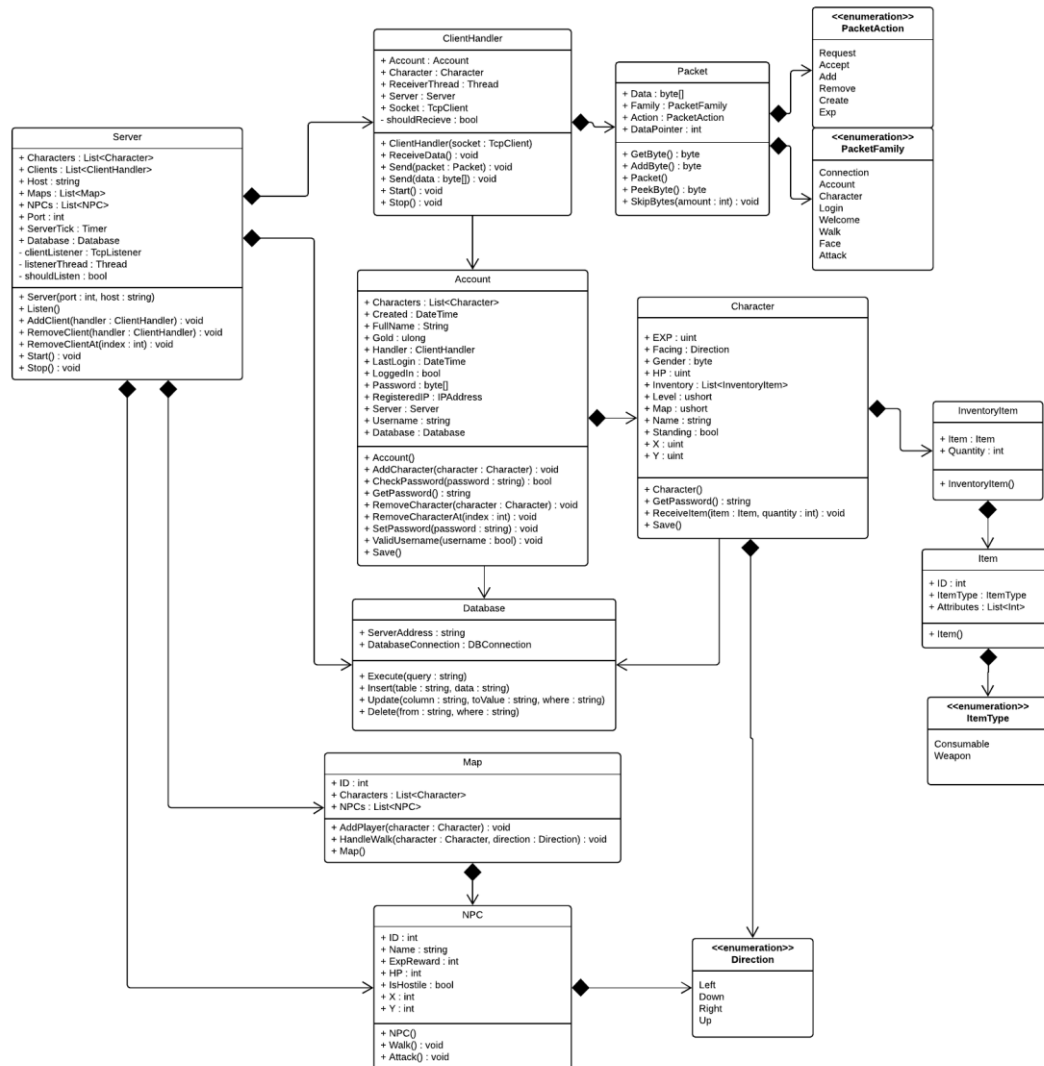


Figure 2 - Initial UML for the server

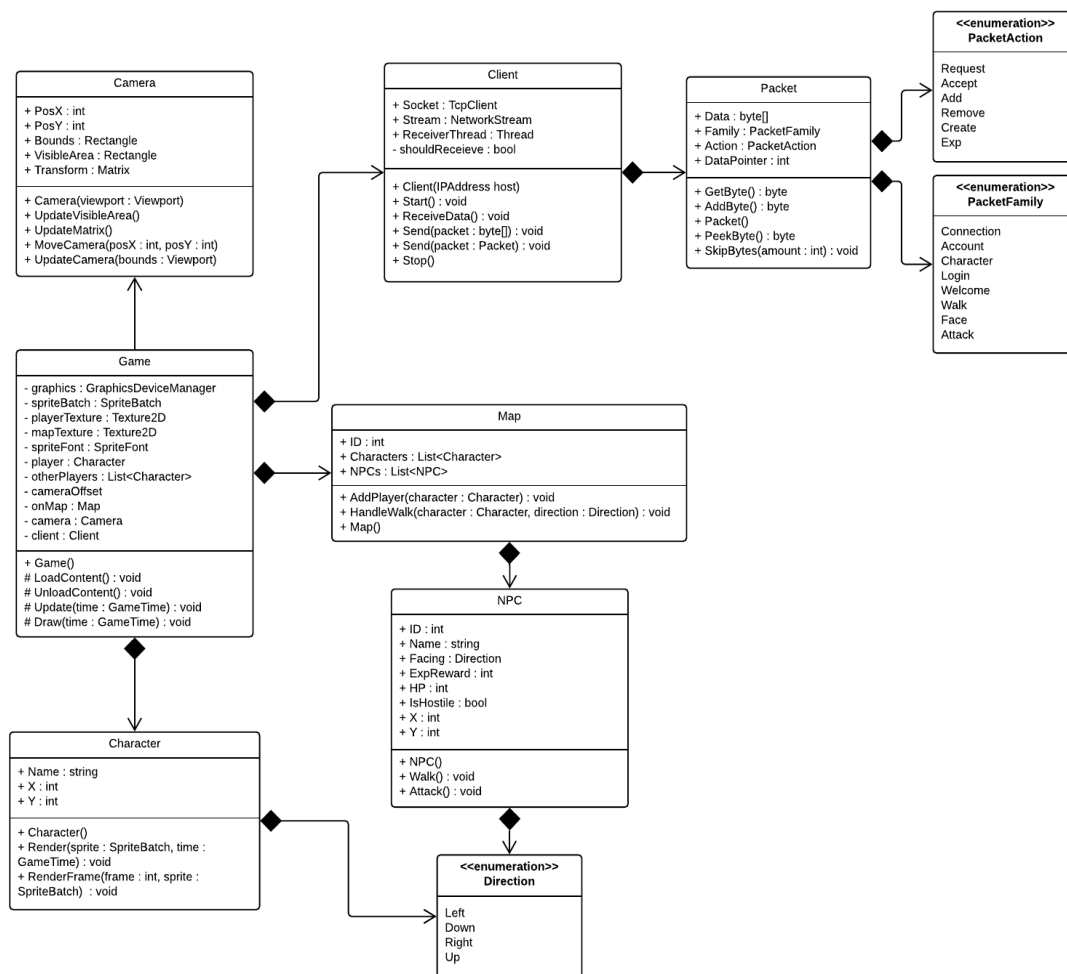


Figure 3 - Initial UML of the client

7.3 Database Design

From the outset it was very clear that not a vast amount 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 come contained in the game client package. 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, amongst other things. While most of the technology stack in this project is C#, MySQL was chosen to be the database technology primarily for its portability, which was another goal of this project. MySQL lends itself to being hosted on many different operating systems, with the specific one of interest being Linux for remote hosting.

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 rapidly increase 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.

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 Entity Relationship Diagram (ERD) below. The normalisation process detailing un-normalised form (UNF) all the way through to third normal form (3NF) can be found in the google sheet at this address:

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

The table presented in the link above will show primary keys underlined and foreign keys underlined with their name followed by an asterisk. Appropriate column headings will also show what stage of normalisation is expected in that column.

⁴ Agile Data on Normalisation - <http://agiledata.org/essays/dataNormalization.html>

7.3.2 Database ERD

The database ERD as mentioned before shows two separate relational entities. The first of which seen on the left-hand side are the relationships between the entities created by a player, and on the right, are the entities controlled by the server. The ERD shown in Figure 4 was created before database implementation and is almost identical to the final ERD shown in Appendix B. A data dictionary was also produced to describe in detail the data types of the properties of the given entities also shown in Appendix B.

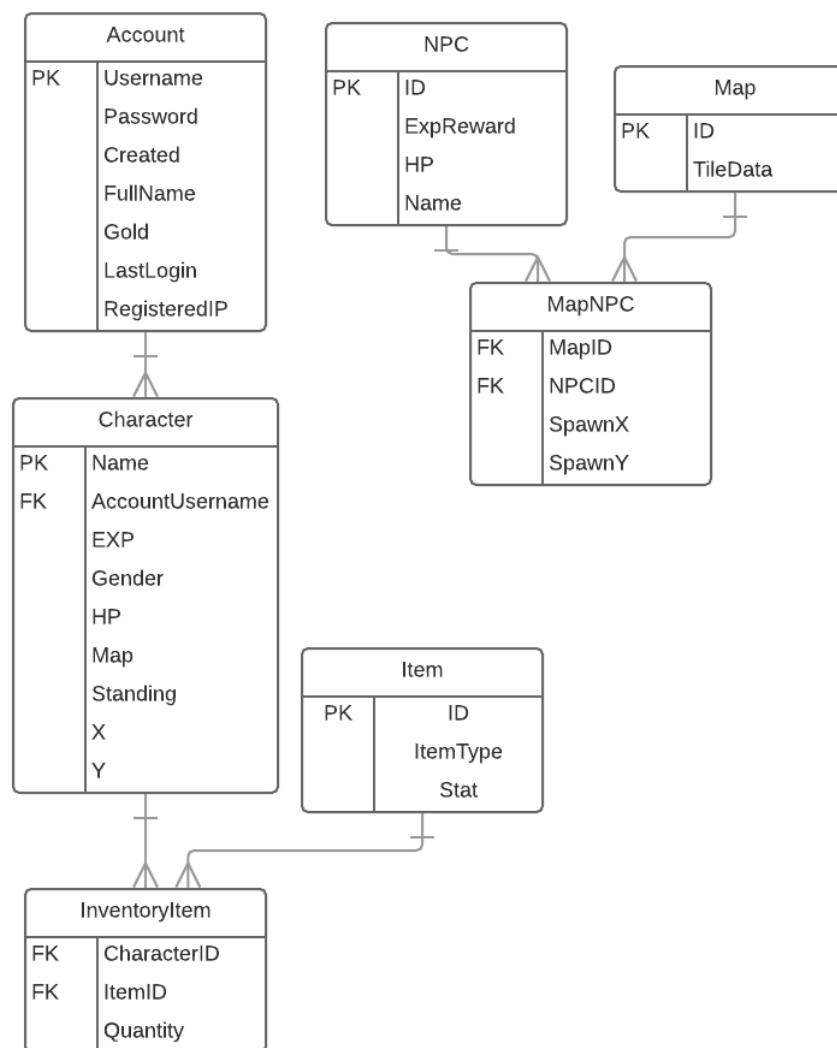


Figure 4 – Initial Database ERD

7.4 Game Client Design

According to an article from early 2013⁵, it was around the time of the article being published that Microsoft officially dropped support for their XNA project, which was

⁵ Gamesutra on XNA - https://www.gamasutra.com/view/news/185894/Its_official_XNA_is_dead.php

then swiftly picked up by the open source community to start the project known as MonoGame. MonoGame is a very well-established C# Games programming framework, which was a factor in selecting it as the technology for the game client as it was the same technology as the game server. It was later discovered that MonoGame is also capable of cross-platform compilations, which is just another added benefit.

The network side of the game client is very similar to that of the server as the protocols needed to match so it is logical to pull out those networking components and have them shared between both server and client. This factor alone is reason enough to have a third project aside from the server and client which houses these networking components among other things that are shared between the server and client such as the definition of a player and an enemy. For debugging purposes, the client also shows the latency between itself and the server (displayed in the top left of the in-game window), which is achieved by comparing timestamps, contained in the packet payloads of the ping and pong cycles between client and server.

A few of the planned classes in the game client changed substantially, one of those being characters. As the project developed, it was found that both Characters and Enemies shared a lot of the same properties that could be grouped under a Character base class and then encapsulated in a PlayerCharacter and an Enemy class, and the lack of user interface (UI) component classes.

7.4.1 Controls

An oversight prior to development on the project was that UI controls such as buttons and textboxes do not come as standard components of the MonoGame framework. This realisation came very early on in the project, therefore development started on controls that would speed up development such as labels, buttons and text boxes, which would later be combined to make other controls such as a game sprite. Controls make use of a series of rectangles and points to define where they would get rendered on the UI. Each control would come with its own Update and Draw loops with enforced behaviours through interfaces, to ensure they would not be missed in the update and draw loops shown in Figure 5.

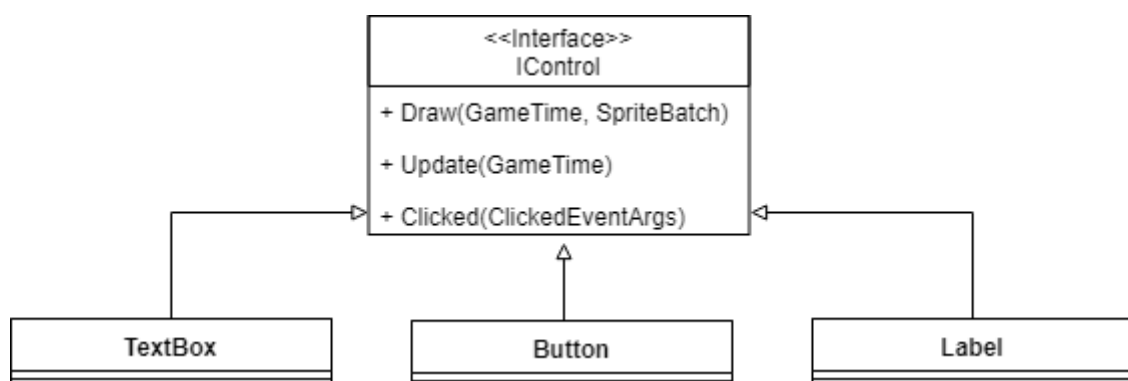


Figure 5 - Control Interface enforcing behaviours

From a maintainability aspect this would also help with clarity within the code base as only one loop would have to be called to update and draw every control that is present within the scene. This very same feature also allows for tabbing between text fields in

the order of when they were added to the array. The most complex of all the controls is the TextBox as it must handle user input which can be done via the Update loop provided by the MonoGame framework. Each individual key press needs to be evaluated as they are objects of KeyInfo (enumeration provided by MonoGame) as they hold no text character properties and then its text character equivalent is appended to the text box with a few varying properties from text field to text field, such as whether it should allow spaces or numbers.

The main factors that are considered when creating a text field are whether it is a password field, how many characters it should allow and what characters it will allow, such as the space character and special characters. Password fields also get their Draw method edited as the true password needs to be hidden behind asterisks or something to the same effect. Another factor that also was not thought about prior to developing the system is something as simple as an input cursor to show where the text is going to be inserted.

7.4.2 Scene switching

Once again, to help with code base clarity, controls would be collected in different scenes and then the scenes would become responsible of what is rendered based on their own collection of controls.

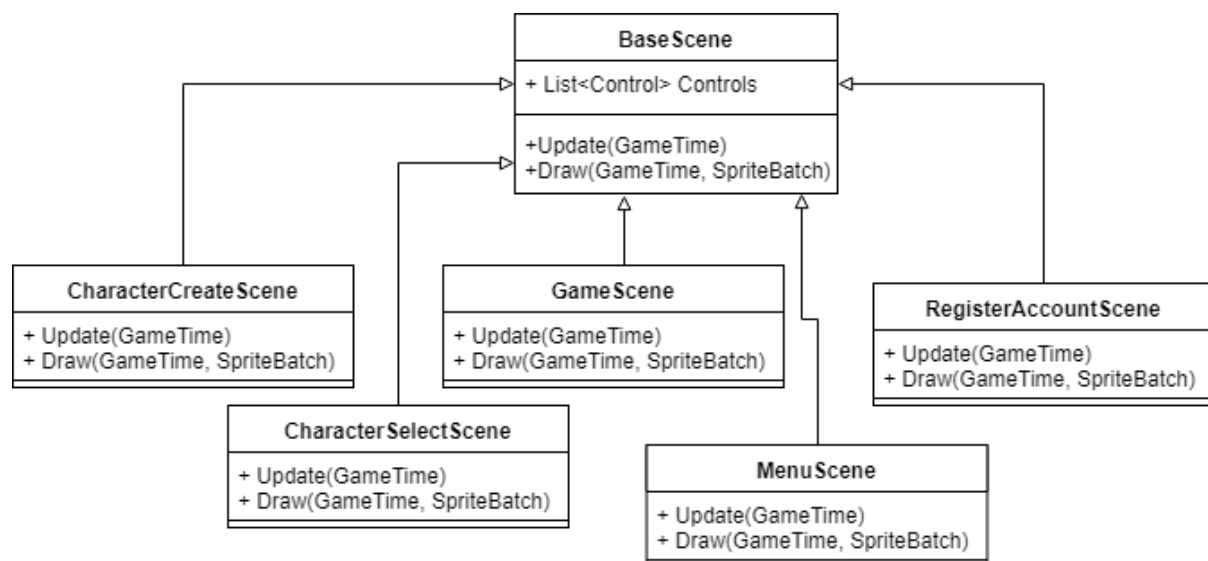


Figure 6 - Scene inheritance

Encapsulating the collection of controls in this way also allows the state pattern to come into play as seen in Figure 7. Storing the game state as the game goes through different scenes means the game client will only update and draw the controls that are relevant for the current state of the client.

```
protected override void Update(GameTime gameTime)
{
    switch(_gameState)
    {
        case GameState.MainMenu:
            Menu.Update(gameTime);
            break;
        case GameState.LoggedIn:
            CharacterSelect.Update(gameTime);
            break;
        case GameState.RegisterAccount:
            RegisterAccount.Update(gameTime);
            break;
        case GameState.CreateCharacter:
            CharacterCreate.Update(gameTime);
            break;
        case GameState.Playing:
            GameScence.Update(gameTime);
            break;
    }

    base.Update(gameTime);
}
```

Figure 7 - State pattern

7.4.3 In-game Sprites and Animations

To handle animations, sprite sheets were used with an algorithm to numerically assign, each sprite within the sheet, an ID. In Figure 8, there are two different character sprites with their respective moving frames. The first 3 frames are the male walking down, with the left movement underneath followed by the right movement and then finally the upwards movement.

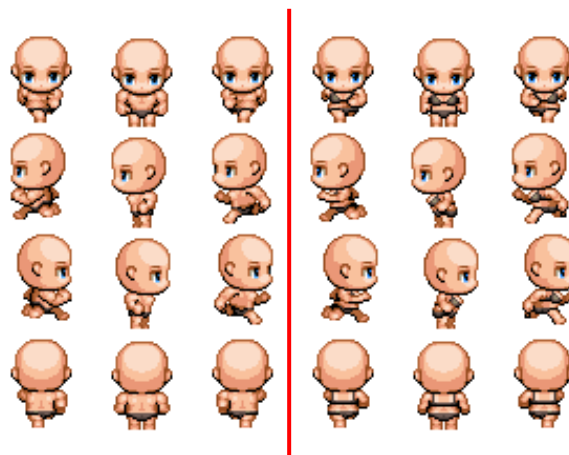


Figure 8 - Base character sprite sheet, male on the left, female on the right

The formula that would be used to obtain the correct frame can be seen in Figure 9. Using this formula, a rectangle can be created to crop a specific frame out of the sprite sheet or texture. This rectangle would have its location as a point with the values of FrameX and FrameY and its size as a point with the values of the frame width and frame height to obtain the correct frame from the sprite sheet. Giving the formula the ID of 7 will present the calling draw method with the frame of the male standing facing left.

ID = the ID of the requested frame (indexed from 0)
 FramesWide = Texture Width / Frame Width
 FrameX = ID % FramesWide
 FrameY = ID / FramesWide

Figure 9 - Getting the frame rectangle

To allow for animation, an id will be given to the sprite container with a value between 0 and 2, to cycle through the 3 different frames that are given for the sprite animation. There also be time delay to ensure that the frames can be distinguished between one another. The final design has this time delay to be 200 milliseconds.

7.4.5 Unresolved issues

The client in the final state did not allow for fluid attack animations as the source sprites did not contain any combat frames nor were any accompanying weapons so the combat has remained in a visibly unfinished state. A clean log out option was not implemented during development, meaning a hard exit is the only option. There is also an issue with sprite drawing order, whereby sometimes they would render in the correct order (the sprite with the highest Y value would be rendered last as though to appear “on top”) but other times it would seem random which was attempted to be rectified by applying sprite depth in the formula seen in Figure 10. Using the below formula, the client should be prioritising sprites with a value closer to 1 first and sprites with a value closer to 0 last. This means that with a sprite y value of 10 and a map maximum value of 10 the sprite will be at the bottom of the map and therefore rendered last as it is “on top”.

$$1 - (\text{Sprite Y} / \text{Map Maximum Y})$$

Figure 10 - Sprite Depth Formula

7.5 Server Design

The server was mostly kept to the planned UML, with a few exceptions. The data layer became much more fleshed out with the Account being responsible for creating its own database entries and updates and likewise for Characters, inserting themselves into the database through a reference to the data layer. The items of the characters and enemies were also omitted in the final design due to time constraints, so the depth of the server suffered in this area also.

7.5.1 Concurrency

The main focus of the design of the server was to ensure that any blocking functions, which as the name suggests, block execution of a thread, were put into their own worker threads, which allows the execution of the server to continue. The most notable of which are the bind and receive calls made by the listener object and the network stream respectively, as they need to wait for incoming network data. The blocking calls were therefore given their own worker threads in which to wait for that data. To ensure collections of data were not being modified at the same time, a lock call is made on the data, to make any other thread wanting access to the same (locked) data wait until the data is freed from its previous lock.

7.5.2 Logging

The initial design of the server would also allow the console output to be logged to a file in case of server crash. However, this was later changed as it seemed through testing, the amount of overhead generated by continuously opening, appending and closing a file stream every time an event occurred within the server, was increasingly becoming a noticeable problem. This was due frequency of the logging calls being

made. In the early development phases when raw packet data was also being logged to the console, was when file logging really became an issue, as there were hundreds of packets going in and out of the server every minute. Logging options were then added to the system to toggle on and off computationally intensive calls.

7.5.3 Security

In the event that the server database is compromised or the virtual machine that hosts the server is compromised, steps were taken to protect the users of the system by hashing and salting stored passwords. The plain text password is sent over the network socket, for the reason that C# code can be decompiled and any hashing or encrypting logic that would be kept in the client would be exposed to potential malicious intent. Passwords are hashed using a SHA256 hashing algorithm with their username and the phrase “PROCO” to ensure the hashing algorithm could not just be guessed by malicious users. “PROCO” came as a result of a typo of “PRCO”. Due to the nature of salting, if the phrase was to be corrected to “PRCO”, any existing users would no longer have a valid password as the resulting hash would not match the hash stored in the database.

8. Lessons Learned

The number of issues that arose around the project timeline were very apparent during the project. Many of the deliverables either took more time than was planned or significantly less due to the inaccurate estimation of the development time they required. The design and analysis phase of the project was unsuccessful in what was planned to have come from the phase.

The lack of documentation that goes along with the design of the systems is a testament to my own lack of ability to see their benefit, ahead of time before any fleshing out of the systems themselves. Once a sub system was in place, close to how they were designed, they were then re-evaluated if they did not fill the requirements. The project was developed by using the initial design documents as a starting point and as systems became more fleshed out, internal testing was carried out and systems were modified where appropriate to increase their modularity and maintainability.

Throughout the project, good object orientated (OO) practice was adopted and good consistency, with the exception of a few game client data structures. One example would be the use of Vector2's over Points and vice versa as those are data structures that are required by functions provided by the MonoGame Framework.

The aspect of the project that I am most happy with is the fluidity of the client experience from login to in game. Making good use of the strategy pattern makes it very easy to switch between scenes at a code level.

Finally, I believe that my project management leaves a lot to be desired which is what caused the suffering of the depth of the project. The foundations of the network protocol are solid and I believe it will scale well with a user base up to 100 concurrent players as 10 connections remain stable. Features such as an in game currency, dropping as rewards for killing enemies, and other itemisation would really help the

depth of the game along with a working stat system. The stat system could be implemented to the game in the future as a side project.

9. Conclusion

Overall, I feel like the deliverables that were met were a success. Most notably the network protocol which works flawlessly and packet payloads reach the server in good time even when hosted on a remote virtual private server (VPS). The gameplay is not exciting by the projects hand in date, however the core gameplay was not one of the objectives of the project. Despite this aspect, I am happy with the understanding of how the inner workings of the MonoGame framework to produce a prototype of a networked game client with a modular networking system.

The biggest setback for the client was the lack of visual assets that would thematically fit with the design of Dan's World. With those assets available I believe the gameplay would be deeper and more engaging, which therefore would exceed the initial requirements of the client.

The enemy AI is non-existent and they move randomly. AI Logic would need to be implemented into the server and enemy attacking animations would also need to be added client-side for the enemy expectations to be met.

Analysing the project goals that were outlined in the PID, the goals that were met I believe were implemented to a high standard. Until the server has a high user base, the stability of the server cannot be tested to say with any confidence. The client functions well and the user inputs are responsive. As a final note I believe there is huge potential for this project with the backing of professional graphic design.

Appendices

Table of Contents

Appendix Section	Description	Page Number
A	Project Initiation Document (PID)	26
B	Database Design Documents	34
C	Screenshots of Development	40
D	Highlight Reports	45
E	User Guide	46
F	Database SQL Dump	53

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 **Massively Multiplayer Online Roleplaying Game**. 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:

- a. A Massively Multiplayer Online game, handling hundreds if not thousands of concurrent connections to a live game server
- b. Character customisation and progression, such as gear looting and levelling up for the player
- c. Social interaction among the connected players, building an online community which is safe and moderated
- d. A database backend to store account data, character progression, hostile/friendly NPC (Non-Playable Character) data and world/map data.
- e. The player should be able to control and move their character using keybinds and activate interaction with the enter key to send messages
- f. 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:

- a. A persisting online world which enables players to register, connect, play (control their character using the game client) and disconnect.
- b. A basic combat system to fight hostile NPCs, gaining the player experience and the chance to level up to improve combat skills and presence.
- c. 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.
- d. Basic social interaction between other players and NPCs.
- e. A loot system granting a variety of bonus items to the player upon defeating hostile NPCs
- f. A basic questing system to make for a more immersive gaming experience
- g. Inventory system to keep track of what the player owns.
- h. 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
 - a. 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.
 - b. Developing a Game Client in a framework which is well documented for ease of development with also included functionality for network communications.
 - c. 2D environment with basic combat functionality.
2. The Game Server
 - a. 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.
 - b. The game server should back its state up to the database periodically in case of power failure or system crash.
 - c. 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.
 - d. 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
 - a. A database capable of large amounts of Game Data, holding a stable connection to the game server for periodic updates.
 - b. Expected to hold Account, Character, NPC, Item and Map Data.
 - c. Reasonably portable to enable local development and remote live copies of the database to be in use.
4. NPCs
 - a. Hostile NPCs should have some logic behind them to provide some challenge for the player during combat.
 - b. 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)
 - a. 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.

- b. Maybe having multiple races would be a good way to put certain players against others as they would then have motive.
- 2. Class System
 - a. 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
 - a. 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	22 nd January	2 nd February	PID Final Draft
2. Phase 1	2 nd February	16 th 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.
3. Phase 2	16 th February	23 rd 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.
4. Phase 3	23 rd February	9 th 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.
5. Phase 4	9 th March	16 th 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.
6. Phase 5	16 th March	2 nd April	Implementation of enemy logic in the server and communicating enemy locations and abilities to the connected game client(s).
7. Phase 6	2 nd April	5 th April	A few days dedicated to moderation functionality for in game moderators, developers and testers.
8. Phase 7	5 th April	9 th April	Final phase of development for touching up the game client and server for any additional features before the last testing phase.
9. Usability and testing	9 th April	12 th 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.
10. Phase 8	12 th April	20 th 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.
11. System and user acceptance testing	20 th April	27 th 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.
12. Final Report	27 th April	4 th 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.

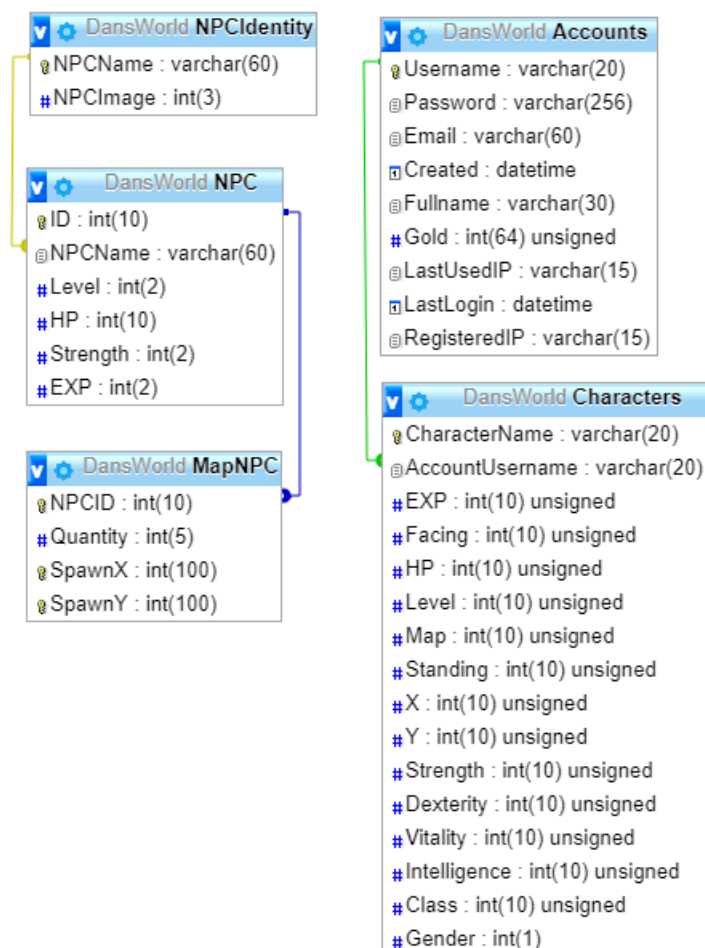
A.10.1 University of Plymouth Ethics Policy

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.

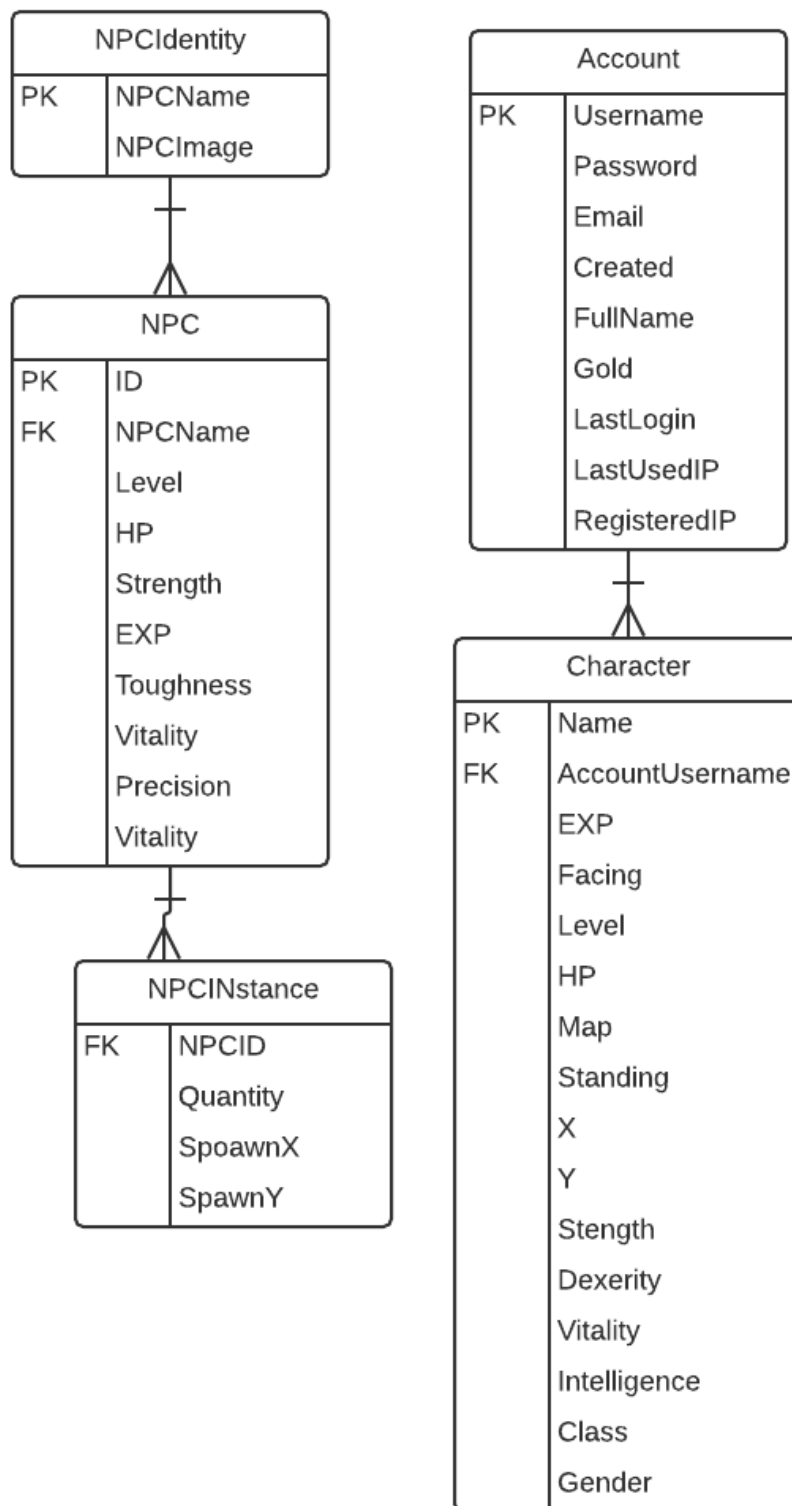
B Database Design Documents

B.1 Final ERD

B.1.1 PHPMysqlAdmin Generated ERD



B.1.2 Drawn ERD



B.2 Data Dictionary

Accounts

Field Name	Data Type	Data Format	Field Size	Description	Example
Username	Text	No spaces beginning with a letter, consists of only alphanumeric data	20	Unique username for all accounts	Testuser
Password	Text	alphanumeric	100	Password field designed to store a hashed and/or encrypted string	535d341b4ce31dec18a80c8b256bdc84
Created	DateTime	YYYY/MM/DD	10	The date of which the account was created	2018/01/01
FullName	Text	First name <space> Last name	40	The owner of the accounts name	Daniel Scott
Gold	Numeric	NNNNNNNNNNNNNN	64	The amount of gold the account has accrued across all characters	1000
LastLogin	DateTime	YYYY/MM/DD H:i:s	19	The last occasion that an account was logged into the game	2018/01/02 18:30:20
LastUsedIP	Text	NNN.NNN.NNN.NNN where N is not negative and there can be between 1 and 3 N's	15	The last IP that was used to log in to the account	192.168.0.1
RegisteredIP	Text	NNN.NNN.NNN.NNN where N is not negative and there can be between 1 and 3 N's	15	The IP that was used to register the account	192.168.0.1

Characters

Field Name	Data Type	Data Format	Field Size	Description	Example
Name	Text	Alphabetical characters and spaces only but no	20	Unique name for a character	Dan Scott

		consecutive spaces			
Account Username	Text	Alphanumeric data	20	Username of the account that owns this character	DanScott
EXP	Numeric	NNNNNNNNNNNNNN	64	The experience gained by this character	99999
Facing	Numeric	N	1	The direction of where the character is facing, 1-4	2
HP	Numeric	NNNNNNNNNNNNNN	32	Health count of the character	500
Level	Numeric	NNN	3	The level of the character, increments when reaching the experience threshold for a level up	10
Map	Numeric	NNNNNNNNNNNNNN	32	The numeric integer of the map the character is in	1
Standing	Boolean	N	1	Boolean flag to display whether the character is standing or not	0
X	Numeric	NNNNNNNNNNNNNN	32	X location of the character when it was logged out	50
Y	Numeric	NNNNNNNNNNNNNN	32	Y location of the character when it was logged out	50
Strength	Numeric	NNNNNNNNNNNNNN	32	The strength value of the character. Used to calculate the outgoing damage.	5000
Dexterity	Numeric	NNNNNNNNNNNNNN	32	The dexterity value of the player. Used to calculate the dodge chance of incoming attacks.	5000
Vitality	Numeric	NNNNNNNNNNNNNN	32	The vitality value of the character. Used to calculate the Health Points.	5000
Intelligence	Numeric	NNNNNNNNNNNNNN	32	The intelligence value of the character. Used to	5000

calculate the mana count and spell damage of a character.

Class	Numeric	N	1	The class identifier of the character. Matches against warrior/mage/archer	1
Gender	Numeric	N	1	Bool to specify male or female character	1

Inventory Item

Field Name	Data Type	Data Format	Field Size	Description	Example
CharacterName	Text	Alphabetical characters and spaces only but no consecutive spaces	20	Foreign field from Character	Dan Scott
ItemID	Numeric	NNNNNNNNNNNNNN	32	Foreign field from Item	10
Quantity	Numeric	NNNNNNNNNNNNNN	32	Quantity of the item belonging to the character	10

Item – not in final database

Field Name	Data Type	Data Format	Field Size	Description	Example
ID	Numeric	NNNNNNNNNNNN	32	The id of the item	10
ItemType	Numeric	NNN	3	Type identifier of the item, can	
Attributes	Text	%,%,%,%,%,%	500	A long string filled with item attributes separated by ','s	Name,Test Item,Power,10,Toughness,10
ImageID	Numeric	NNN	3	The ID of the image string identifier	2.png

NPC

Field Name	Data Type	Data Format	Field Size	Description	Example
ID	Numeric	NNN	32	ID of the NPC	10

Name	Text	Alphabetical with spaces	30	The name of the NPC taken from a foreign lookup table	Test NPC
Level	Numeric	NN	2	The level of the NPC (Recommended level of the player)	15
ExpReward	Numeric	NNNNN	5	The EXP reward given by the NPC	200
HP	Numeric	NNNNNNNNNNNNNN	32	Numeric value of the NPC's Health Pool	3000
Strength	Numeric	NNNNNNNNNNNNNN	32	The strength value of the NPC. Used to calculate the outgoing damage.	5000

Map – not in final database

Field Name	Data Type	Data Format	Field Size	Description	Example
ID	Numeric	NNNNNNNNNNNN	32	The ID of the map	1
Name	Text	Alphabetical with spaces		The name of the map	Tutorial Map
TileData	Text	X,Y,N,W,X,Y,N,W, X,Y,N,W where X is the X co-ordinate of the tile, Y is the Y co-ordinate of the tile and N is the ID of tile (graphic) and W is walkable (1/0)	3000	The tile data of the map	0,0,1,1,0,1,1,1,0,2,1,1,0,3,1,1

MapNPC

Field Name	Data Type	Data Format	Field Size	Description	Example
MapID – not in final database	Numeric	NNNNNNNNNNNN	32	ID of the map where the instance of this NPC exists	10
NPCID	Numeric	NNNNNNNNNNNN	32	The ID of the NPC to spawn	1

SpawnX	Numeric	NNNNNNNNNNNN	32	X Location of the NPC	40
SpawnY	Numeric	NNNNNNNNNNNN	32	Y Location of the NPC	40
Quantity	Numeric	NNNNNNNNNNNN	32	Quantity of NPC's to spawn	5

NPCIdentity

Field Name	Data Type	Data Format	Field Size	Description	Example
NPCName	Text	Alphabetical with spaces	30	The name an NPC which can be reused many times	Test NPC
NPCImage	Numeric	NNNNNNNNNN	3	The ID of the sprite found in the texture	100

C Screenshots of Development

C.1 10 Clients connected to the server

C.1.1 Server Console

Restricted output

```

C:\Program Files\dotnet\dotnet.exe
[03:27:19][INFO] Saving Character: test10 information
[03:27:19][INFO] Saving Character: test33 information
[03:27:19][INFO] Saving Character: test5 information
[03:27:19][INFO] Saving Character: test55 information
[03:27:19][INFO] Saving Character: test6 information
[03:27:19][INFO] Saving Character: test7 information
[03:27:19][INFO] Saving Character: test8 information
[03:27:27][INFO] Client connection accepted from 77.101.233.137:47219 Assigned ID: 1146
[03:27:27][INFO] Client handler thread spawned id:20
[03:27:27][INFO] Username: test9 Password: 175C5E54178C681C8DBFCAAAC1C6E3D7DFF25F7F4BAECA79EA8503FA7231894
[03:27:32][INFO] Username: test9 Password: CE8E0914C8BD3188249A54EBD74089B2B91CA7EE999C3B2C3BEDFE4A3C17BB52
[03:27:32][INFO] Login accepted from 77.101.233.137:47219 for account test9
[03:27:39][INFO] Character creation requested name test9 account test9
[03:27:39][INFO] Character Created. Name: test9
[03:27:49][INFO] Client connection accepted from 77.101.233.137:47244 Assigned ID: 7757
[03:27:49][INFO] Client handler thread spawned id:21
[03:27:49][INFO] Username: test10 Password: 519B152CC62C5ED9823FDE478D0FD3C8E9CA5879022B53228388270FEFE7385D
[03:27:49][INFO] Login accepted from 77.101.233.137:47244 for account test10
[03:28:02][INFO] Character creation requested name test11 account test10
[03:28:03][INFO] Character Created. Name: test11
[03:28:18][INFO] Saving Character: test information
[03:28:18][INFO] Saving Character: test1 information
[03:28:19][INFO] Saving Character: test10 information
[03:28:19][INFO] Saving Character: test33 information
[03:28:19][INFO] Saving Character: test5 information
[03:28:19][INFO] Saving Character: test55 information
[03:28:19][INFO] Saving Character: test6 information
[03:28:19][INFO] Saving Character: test7 information
[03:28:19][INFO] Saving Character: test8 information
[03:28:19][INFO] Saving Character: test9 information

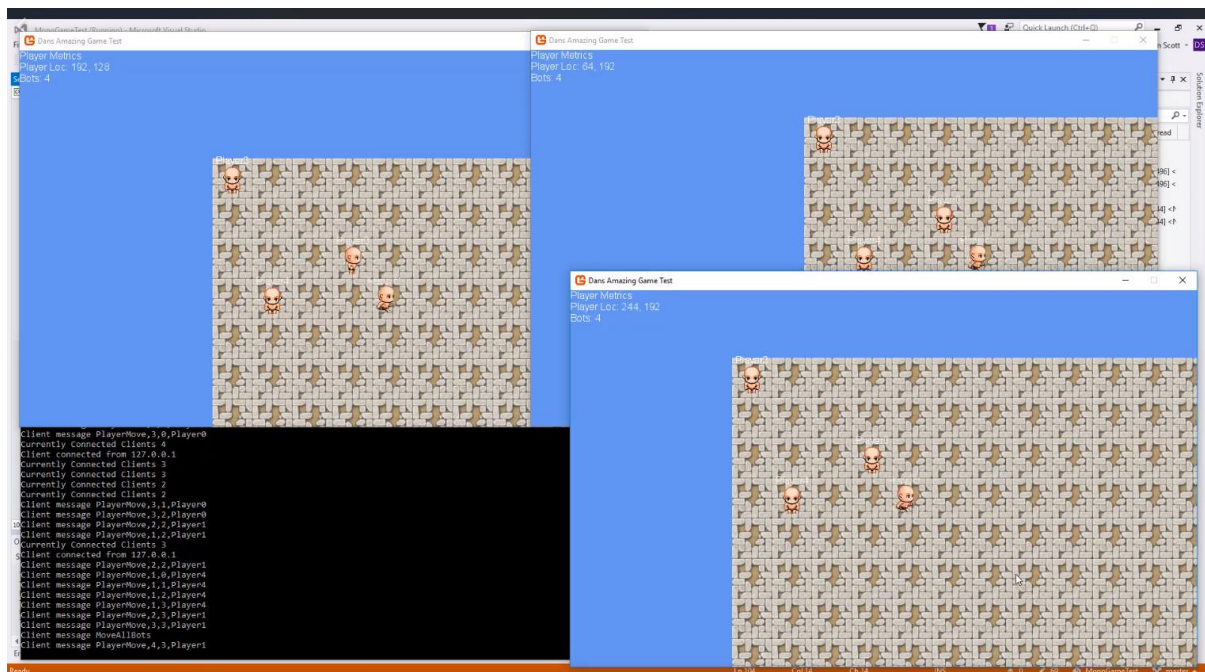
```

Console output of packet data. Flags 6 and 4 would give the packet identifier of ENEMY.MOVE.

```
C:\Program Files\dotnet\dotnet.exe
[18:04:52][INFO] Length: 15 Data: [6] [4] [8] [0] [0] [0] [242] [0] [0] [0] [23] [0] [0] [0] [2]
[18:04:52][INFO] Length: 15 Data: [6] [4] [13] [0] [0] [0] [221] [2] [0] [0] [159] [1] [0] [0] [3]
[18:04:52][INFO] Length: 15 Data: [6] [4] [14] [0] [0] [0] [247] [2] [0] [0] [71] [1] [0] [0] [0]
[18:04:52][INFO] Length: 15 Data: [6] [4] [16] [0] [0] [0] [200] [0] [0] [0] [52] [1] [0] [0] [1]
[18:04:52][INFO] Length: 15 Data: [6] [4] [17] [0] [0] [0] [145] [2] [0] [0] [103] [2] [0] [0] [0]
[18:04:52][INFO] Length: 15 Data: [6] [4] [18] [0] [0] [0] [113] [2] [0] [0] [128] [2] [0] [0] [2]
[18:04:52][INFO] Length: 15 Data: [6] [4] [19] [0] [0] [0] [47] [2] [0] [0] [203] [2] [0] [0] [2]
[18:04:52][INFO] Length: 15 Data: [6] [4] [20] [0] [0] [0] [30] [2] [0] [0] [136] [2] [0] [0] [3]
[18:04:52][INFO] Length: 15 Data: [6] [4] [21] [0] [0] [0] [191] [0] [0] [0] [32] [1] [0] [0] [0]
[18:04:52][INFO] Length: 15 Data: [6] [4] [22] [0] [0] [0] [106] [2] [0] [0] [254] [2] [0] [0] [1]
[18:04:52][INFO] Length: 15 Data: [6] [4] [23] [0] [0] [0] [185] [2] [0] [0] [127] [2] [0] [0] [2]
[18:04:52][INFO] Length: 15 Data: [6] [4] [0] [0] [0] [0] [47] [2] [0] [0] [231] [1] [0] [0] [3]
[18:04:52][INFO] Length: 15 Data: [6] [4] [3] [0] [0] [0] [20] [1] [0] [0] [240] [1] [0] [0] [3]
[18:04:52][INFO] Length: 15 Data: [6] [4] [4] [0] [0] [0] [180] [1] [0] [0] [61] [1] [0] [0] [1]
[18:04:52][INFO] Length: 15 Data: [6] [4] [5] [0] [0] [0] [188] [0] [0] [0] [23] [1] [0] [0] [0]
[18:04:52][INFO] Length: 15 Data: [6] [4] [6] [0] [0] [0] [163] [1] [0] [0] [9] [0] [0] [0] [3]
[18:04:52][INFO] Length: 15 Data: [6] [4] [7] [0] [0] [0] [52] [2] [0] [0] [98] [0] [0] [0] [0]
[18:04:52][INFO] Length: 15 Data: [6] [4] [8] [0] [0] [0] [243] [0] [0] [0] [23] [0] [0] [0] [2]
[18:04:52][INFO] Length: 15 Data: [6] [4] [11] [0] [0] [0] [6] [3] [0] [0] [230] [0] [0] [0] [3]
[18:04:52][INFO] Length: 15 Data: [6] [4] [13] [0] [0] [0] [221] [2] [0] [0] [158] [1] [0] [0] [3]
[18:04:52][INFO] Length: 15 Data: [6] [4] [14] [0] [0] [0] [246] [2] [0] [0] [71] [1] [0] [0] [0]
[18:04:52][INFO] Length: 15 Data: [6] [4] [16] [0] [0] [0] [200] [0] [0] [0] [53] [1] [0] [0] [1]
[18:04:52][INFO] Length: 15 Data: [6] [4] [17] [0] [0] [0] [144] [2] [0] [0] [103] [2] [0] [0] [0]
[18:04:52][INFO] Length: 15 Data: [6] [4] [18] [0] [0] [0] [114] [2] [0] [0] [128] [2] [0] [0] [2]
[18:04:52][INFO] Length: 15 Data: [6] [4] [19] [0] [0] [0] [48] [2] [0] [0] [203] [2] [0] [0] [2]
[18:04:52][INFO] Length: 15 Data: [6] [4] [20] [0] [0] [0] [30] [2] [0] [0] [135] [2] [0] [0] [3]
[18:04:52][INFO] Length: 15 Data: [6] [4] [21] [0] [0] [0] [190] [0] [0] [0] [32] [1] [0] [0] [0]
[18:04:52][INFO] Length: 15 Data: [6] [4] [22] [0] [0] [0] [106] [2] [0] [0] [255] [2] [0] [0] [1]
[18:04:52][INFO] Length: 15 Data: [6] [4] [23] [0] [0] [0] [186] [2] [0] [0] [127] [2] [0] [0] [2]
```

C.1.2 Game Client

Basic network communications using strings



Login Screen




DansWorld - Version 0.1.0.32314

DAN'S WORLD

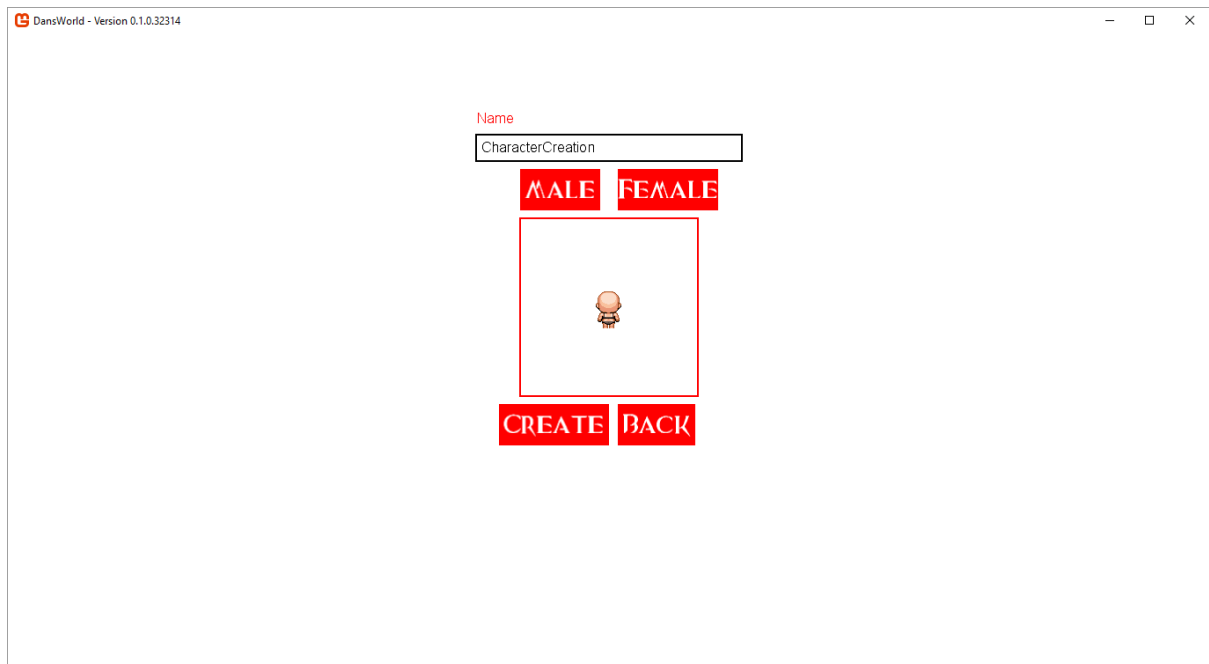
REGISTER **PLAY**

Character Selection Screen

DansWorld - Version 0.1.0.32314

<p>Dan Scott F</p>  <p>Lvl 100</p> <p>PLAY</p> <p>DELETE</p>	<p>DanScott</p>  <p>Lvl 80</p> <p>PLAY</p> <p>DELETE</p>	<p>test123</p>  <p>Lvl 0</p> <p>PLAY</p> <p>DELETE</p>
---	---	---

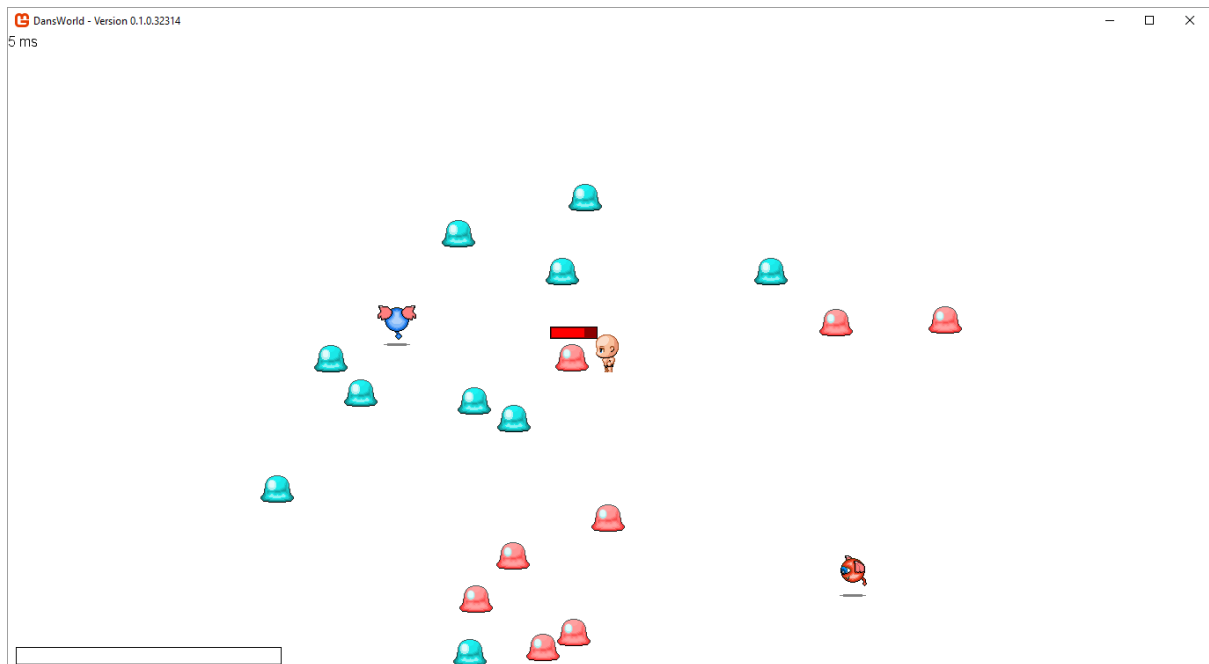
Character Creation



10 concurrent game clients connected



Testing more enemies in game, health visibility timers were also included.



Hardware usage of 10 running clients.

A screenshot of the Windows Task Manager Performance tab. The table shows the hardware usage for various processes. The columns are Name, CPU, Memory, Disk, Network, GPU, and GPU Engine. The data is as follows:

Name	32% CPU	64% Memory	0% Disk	1% Network	20% GPU	GPU Engine
Windows Audio Device Graph Is...	2.9%	33.6 MB	0 MB/s	0 Mbps	0%	
> Spotify (5)	0%	33.0 MB	0 MB/s	0 Mbps	0%	
> DansWorld (32 bit) (2)	2.3%	26.0 MB	0 MB/s	0.1 Mbps	1.5%	GPU 0 -
> DansWorld (32 bit) (2)	0.7%	25.9 MB	0 MB/s	0.1 Mbps	1.6%	GPU 0 -
> DansWorld (32 bit) (2)	1.0%	25.9 MB	0 MB/s	0.1 Mbps	1.6%	GPU 0 -
> DansWorld (32 bit) (2)	0.6%	25.9 MB	0 MB/s	0.1 Mbps	1.5%	GPU 0 -
> DansWorld (32 bit) (2)	2.2%	25.9 MB	0 MB/s	0.1 Mbps	1.6%	GPU 0 -
> DansWorld (32 bit) (2)	1.4%	25.9 MB	0 MB/s	0.1 Mbps	1.6%	GPU 0 -
> DansWorld (32 bit) (2)	0.9%	25.8 MB	0 MB/s	0.1 Mbps	1.5%	GPU 0 -
> DansWorld (32 bit) (2)	0.9%	25.7 MB	0 MB/s	0.1 Mbps	1.5%	GPU 0 -
> DansWorld (32 bit) (2)	1.5%	25.5 MB	0 MB/s	0.1 Mbps	1.6%	GPU 0 -
> DansWorld (32 bit) (2)	1.6%	25.5 MB	0 MB/s	0.1 Mbps	1.6%	GPU 0 -
> Task Manager	0.9%	24.3 MB	0 MB/s	0 Mbps	0%	
> NVIDIA Container (32 bit)	0%	21.0 MB	0 MB/s	0 Mbps	0%	

D Highlight Reports

PRCO304: Highlight Report
Name: Daniel Scott
Date: 09/02/18
Review of work undertaken Completed UML Class diagrams for both server and client networking layers, ERD for database. These documents were completed on the basis that they are subject to change at any stage during development of the project.
Plan of work for the next week Use case diagrams for all relevant areas of the project based on the initial planning documents.
Date(s) of supervisory meeting(s) since last Highlight: 30/01/18
Brief notes from supervisory meeting(s) since last Highlight Discussed draft PID, advised to expand on the background of the project and focus on the technical problem as opposed to the project as a whole. Once completed, submission of the PID was then approved.

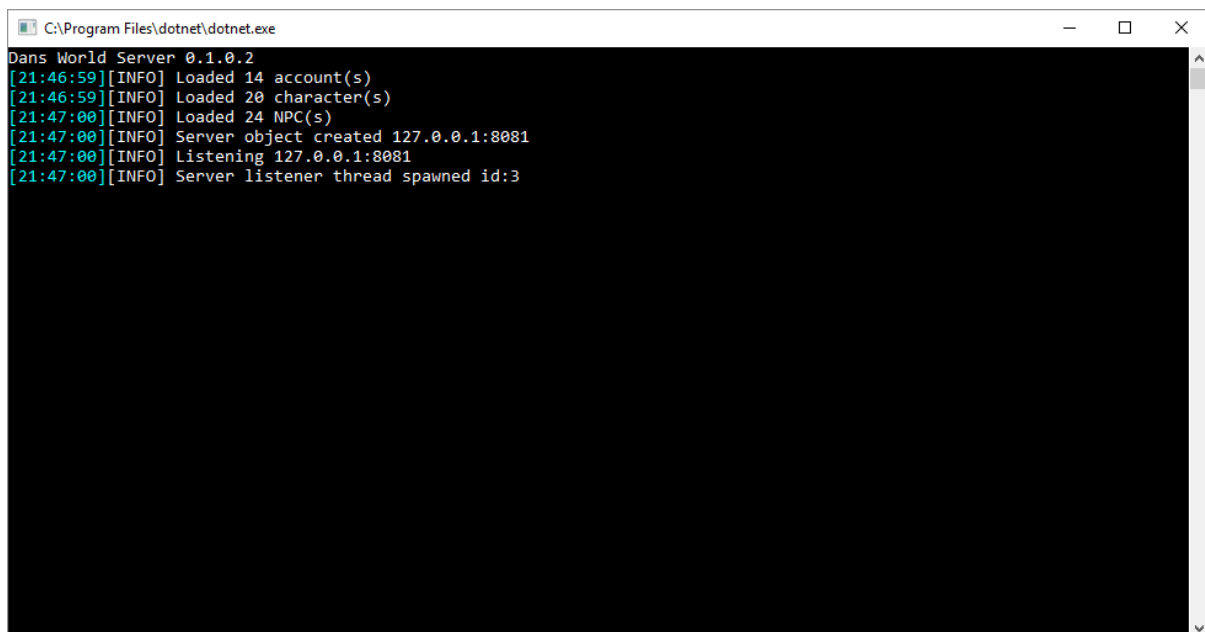
PRCO304: Highlight Report
Name: Daniel Scott
Date: 16/02/18
Review of work undertaken Completed Game Class Diagrams for the game client and ERD for database. These documents were completed on the basis that they are subject to change at any stage during development of the project.
Plan of work for the next week Touch up planning documents where needed and get ready for development next week.
Date(s) of supervisory meeting(s) since last Highlight: N/A
Brief notes from supervisory meeting(s) since last Highlight N/A

PRCO304: Highlight Report
Name: Daniel Scott
Date: 22/02/18
Review of work undertaken Completed Normalisation of database to support ERD design. Data dictionary produced for all tables within the proposed database.
Plan of work for the next week UML redesign
Date(s) of supervisory meeting(s) since last Highlight: 20/02/18
Brief notes from supervisory meeting(s) since last Highlight Encouragement for designing documents for the project. Discussion on how to proceed.

PRCO304: Highlight Report
Name: Daniel Scott
Date: 01/03/18
Review of work undertaken Database Created, API technology played with and server technology started
Plan of work for the next week UML
Date(s) of supervisory meeting(s) since last Highlight: N/A
Brief notes from supervisory meeting(s) since last Highlight N/A

E User Guide

To start up the server a database is required on localhost called `DansWorld` in order for the server to connect to it. It will expect the user "DansWorld" with the password of "uL2qDxZcInz7PU72". Once those things are in place you can go ahead and double click on the server executable after which you should be greeted with this console that shows the accounts, characters and NPCs that were loaded into the server, and also the listening address and port.



```

C:\Program Files\dotnet\dotnet.exe
Dans World Server 0.1.0.2
[21:46:59][INFO] Loaded 14 account(s)
[21:46:59][INFO] Loaded 20 character(s)
[21:47:00][INFO] Loaded 24 NPC(s)
[21:47:00][INFO] Server object created 127.0.0.1:8081
[21:47:00][INFO] Listening 127.0.0.1:8081
[21:47:00][INFO] Server listener thread spawned id:3

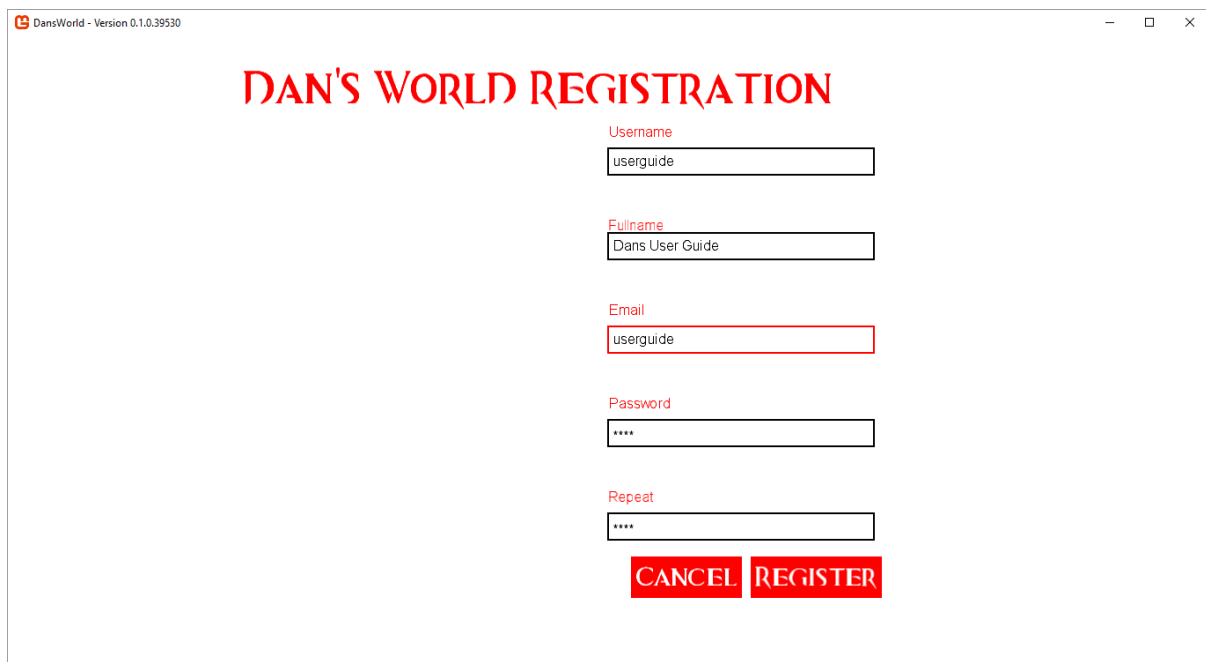
```

Once the server is running you can open the game client to connect to the server once again on localhost. To change the address that the game client will attempt to connect to, the client will have to be re-compiled with the matching address.

Upon opening the game client, you will be greeted with a window which resembles that of the one seen below.



To register a game account simply hit the register button and you will be greeted with the registration screen seen below



As you can see from the above screenshot the email has a red ring around the field. This means that the input given is currently not valid as it expects an email address to be input. Looking at the screenshot below you can see that the given input is accepted and we can now register.

DansWorld - Version 0.1.0.39530

DAN'S WORLD REGISTRATION

Username

Fullname

Email

Password

Repeat

CANCEL REGISTER

Once registration is complete you will be put back to the login page with a message to clearly tell you that the account has been created and is ready for use.

DAN'S WORLD

Username:

Password:

REGISTER PLAY

Account created! Username: userguide

Once logged in you will be greeted with the character selection screen shown in the screenshot below.

CREATE


CREATE

CREATE

From the selection screen you will see that you currently have no characters to select from. Clicking on any create button will take you to the creation screen shown in the screenshot below.

Name

MALE FEMALE




CREATE BACK

In the character creation screen, you will see 4 buttons. Two of which “male” and “female” change the gender of your character, which you will see rotating anti-clockwise to view it from all angles. From here you can either chose to create the character or go back. Upon creating a character with a name that already exists you will be prompted to know that is the case, shown in the screenshot below.

Name

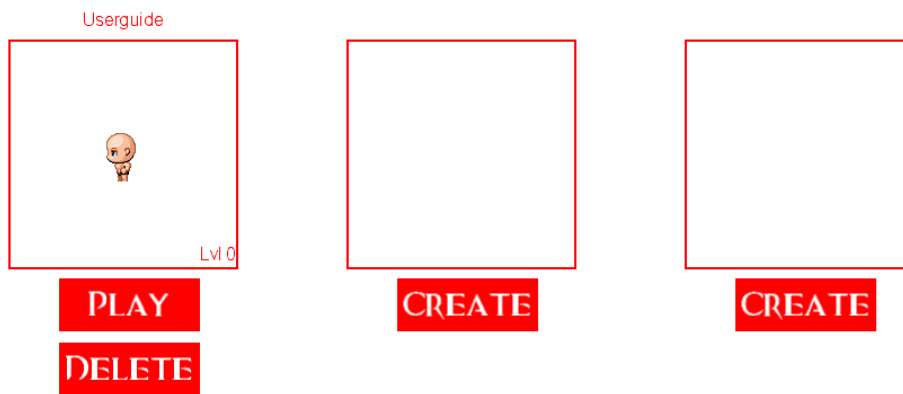
MALE FEMALE



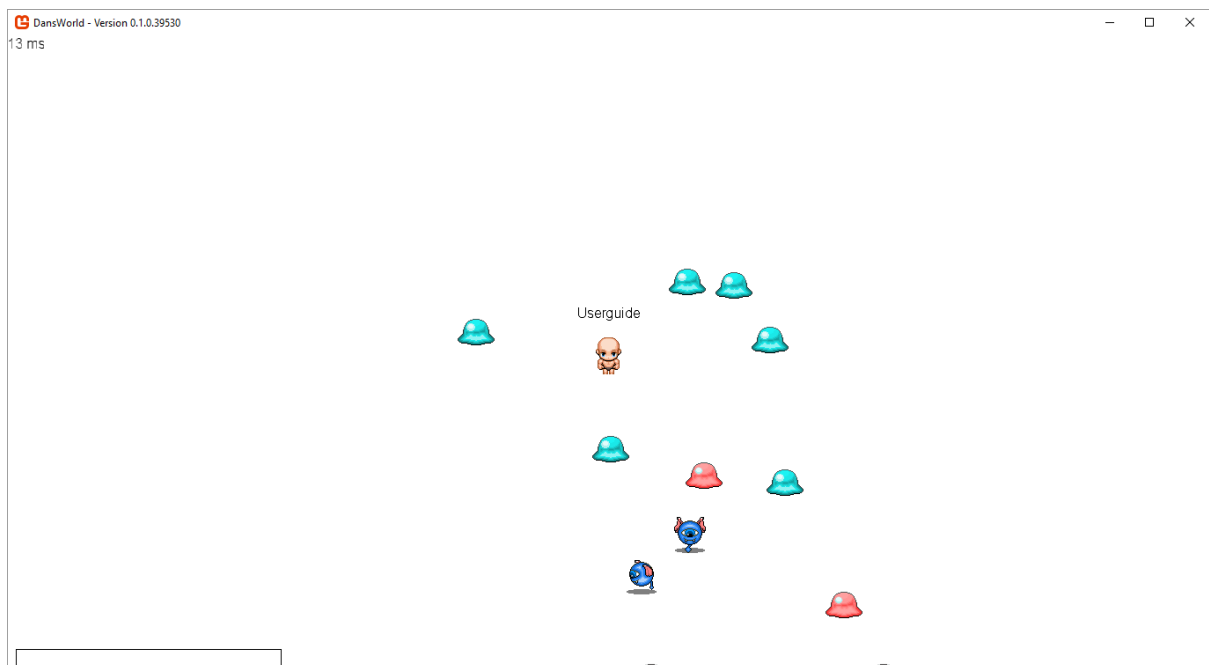
CREATE BACK

Character already exists

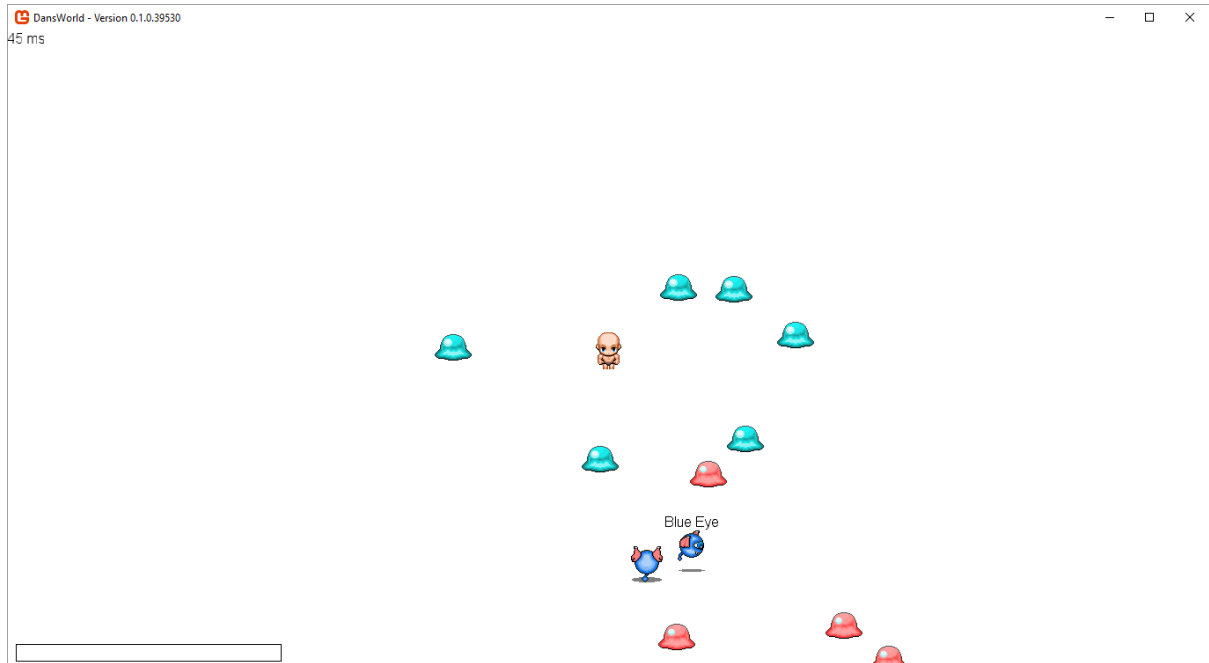
Once choosing a name that you desire to have, you will be taken back to the character selection screen where you will be able to either choose to enter the game or delete any of your existing characters.



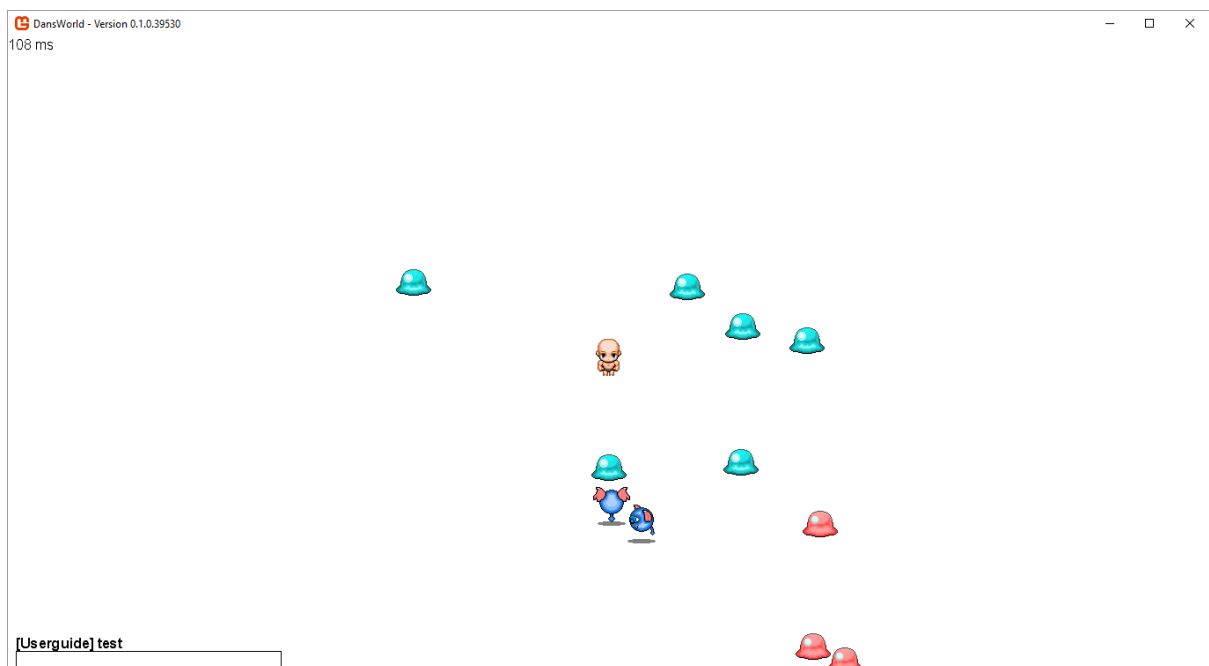
On your character select screen, once you have created a character, you will see their names above the boxes, and their levels in the lower right-hand corner of the character box.



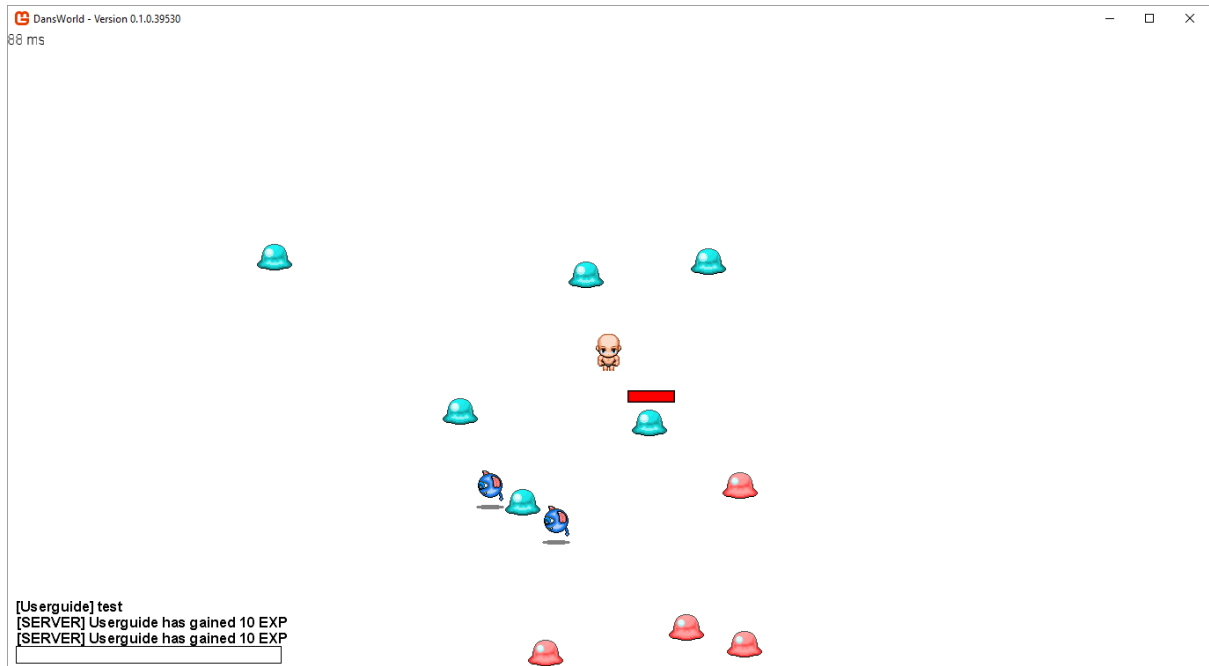
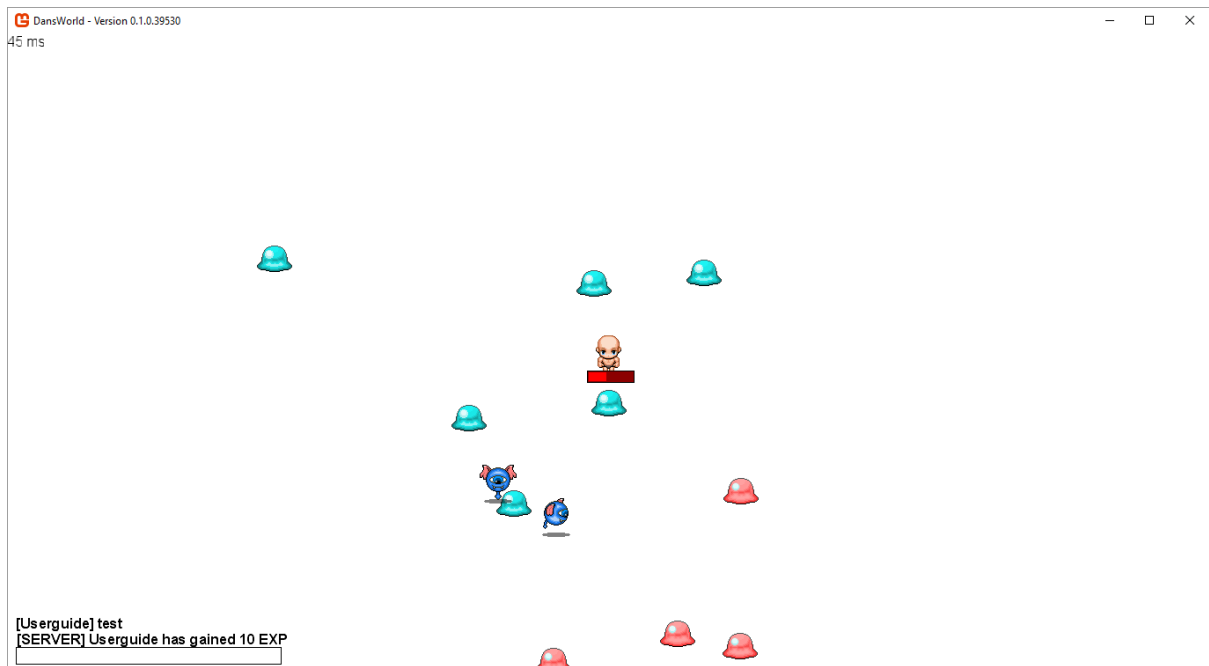
Selecting play will enter the character “Userguide” into the game world and you will be able to begin playing. To describe the user interface, in the top left-hand corner you will see a number followed by an “ms”, this is your latency, i.e. how much time it took for your ping packet to be received by the server, sometimes referred to as “ping”. Any sprites that do not resemble that of your character or the other gender variant are enemy npcs which are killable. Finally, the box in the lower left-hand side of the UI is the chat-box where you can send messages to all players of the game that are currently logged in. You can also hover over any in game agent, whether that is a player or enemy NPC, and it will reveal their name. Looking to the screenshot below you can see hovering over the blue eye reveals its name.



To activate the chat window, simply press enter to see the input cursor appear within the text box and type your message then hit enter again to send it off to the server to be relayed back to all players of the game. The messages will appear with the name of the person who sent the message inside of square brackets followed by their message. The screenshot below shows how received messages appear.



You may also attack enemy npcs in the direction your character is facing, to reveal their health bar by holding down left mouse button (left click). Once the health bar reaches 0 the NPC will respawn around its spawn location with its health restored to maximum and you can continue killing. The screenshots below show an NPC being killed, followed by a killed NPC and then the experience that is rewarded to the player for killing the NPC.



Once you have decided you have had enough, simply clicking the “x” in the top right of the game window will exit you out of the game and your character information will be saved for the next time you wish to log in and play.

F Database SQL Dump

```
CREATE TABLE `Accounts` (  
  `Username` varchar(20) NOT NULL,  
  `Password` varchar(256) NOT NULL,  
  `Email` varchar(60) NOT NULL,  
  `Created` datetime NOT NULL DEFAULT '1970-01-01 00:00:00',  
  `Fullname` varchar(30) NOT NULL,  
  `Gold` int(64) UNSIGNED NOT NULL DEFAULT '0',  
  `LastUsedIP` varchar(15) NOT NULL,  
  `LastLogin` datetime NOT NULL DEFAULT '1970-01-01 00:00:00',  
  `RegisteredIP` varchar(15) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `Characters` (  
  `CharacterName` varchar(20) NOT NULL,  
  `AccountUsername` varchar(20) NOT NULL,  
  `EXP` int(10) UNSIGNED NOT NULL,  
  `Facing` int(10) UNSIGNED NOT NULL,  
  `HP` int(10) UNSIGNED NOT NULL,  
  `Level` int(10) UNSIGNED NOT NULL,  
  `Map` int(10) UNSIGNED NOT NULL,  
  `Standing` int(10) UNSIGNED NOT NULL,  
  `X` int(10) UNSIGNED NOT NULL,  
  `Y` int(10) UNSIGNED NOT NULL,  
  `Strength` int(10) UNSIGNED NOT NULL,  
  `Dexterity` int(10) UNSIGNED NOT NULL,  
  `Vitality` int(10) UNSIGNED NOT NULL,  
  `Intelligence` int(10) UNSIGNED NOT NULL,  
  `Class` int(10) UNSIGNED NOT NULL,  
  `Gender` int(1) NOT NULL DEFAULT '0'  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `MapNPC` (  
  `NPCID` int(10) NOT NULL,  
  `Quantity` int(5) NOT NULL,  
  `SpawnX` int(100) NOT NULL,  
  `SpawnY` int(100) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO `MapNPC` (`NPCID`, `Quantity`, `SpawnX`, `SpawnY`) VALUES  
(1, 10, 300, 300),  
(2, 10, 500, 500),  
(3, 2, 400, 400),  
(4, 2, 700, 700);
```

```
CREATE TABLE `NPC` (  
  `ID` int(10) NOT NULL,  
  `NPCName` varchar(60) NOT NULL,  
  `Level` int(2) NOT NULL,  
  `HP` int(10) NOT NULL,
```

```
`Strength` int(2) NOT NULL,  
`EXP` int(2) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO `NPC` (`ID`, `NPCName`, `Level`, `HP`, `Strength`, `EXP`) VALUES  
(1, 'Blue Blobsie', 1, 50, 2, 10),  
(2, 'Red Blobsie', 1, 100, 10, 20),  
(3, 'Blue Eye', 1, 500, 1, 100),  
(4, 'Red Eye', 1, 1000, 1, 200);
```

```
CREATE TABLE `NPCIdentity` (  
  `NPCName` varchar(60) NOT NULL,  
  `NPCImage` int(3) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO `NPCIdentity` (`NPCName`, `NPCImage`) VALUES  
( 'Blue Blobsie', 96),  
( 'Blue Eye', 6),  
( 'Green Blobsie', 102),  
( 'Green Eye', 3),  
( 'Red Blobsie', 99),  
( 'Red Eye', 9),  
( 'Yellow Eye', 0);
```

```
ALTER TABLE `Accounts`  
  ADD PRIMARY KEY (`Username`);
```

```
ALTER TABLE `Characters`  
  ADD PRIMARY KEY (`CharacterName`) USING BTREE,  
  ADD KEY `AccountUsername` (`AccountUsername`) USING BTREE;
```

```
ALTER TABLE `MapNPC`  
  ADD PRIMARY KEY (`NPCID`, `SpawnX`, `SpawnY`);
```

```
ALTER TABLE `NPC`  
  ADD PRIMARY KEY (`ID`);
```

```
ALTER TABLE `NPCIdentity`  
  ADD PRIMARY KEY (`NPCName`);
```