Cross platform kart racing game in virtual reality

Michael Kidd

John Mannion

Raymond Mannion

Kevin Moran

B.Sc.(Hons) in Software Development

26th April 2019

Final Year Project

Advised by: Dr Patrick Mannion

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

Abstract We have created a cross platform multiplayer video game that can be played by using a mobile device, a traditional desktop or laptop and can be played using a virtual reality headset. The object of the project is to create a heterogeneous system that demonstrates an ability to communicate between multiple languages, operating systems and databases. This will be achieved through the use of data serialisation and marshalling and unmarshalling of data. The project will involve four programmers, using four different programming languages and 3 different databases. We also intended to deploy the system back-end within an AWS server and then deploy these separate systems within docker.

Authors The authors of this Document are Michael Kidd, John Mannion, Raymond Mannion and Kevin Moran, The authors are 4th year students in Galway Mayo Institute of Technology.

Github Link `https://github.com/Michael-Kidd/4th-Year---Main-Project`

Chapter 1

Introduction

The idea for our project came from a video that showed a version of Mario Kart in virtual reality. It depicted the fan favourite Mario Kart racing game that Nintendo originally released august 27 1992, but this time with a player sitting into a specially designed kart, placing a head mounted display unit on their heads and proceeding to play the game. There are currently 8 Mario Kart games, not including the VR versions or arcade versions. The unfortunate thing with this version of the game was that it was not open to the general public, it is simply an arcade version of the game and can only be played at designated venues.

The project that we decided on was to create a similar style game to Mario Kart in virtual reality, however we also wanted that the game could be played cross-platform with a computer and also with mobile devices, while maintaining the ability for each of these versions to be able to play within the same instance. This would mean that a person playing the game on a mobile device and a person playing on a virtual reality headset could play together. From the very start we decided to make different versions of the game that would still allow the games to be played within the same instances. By separating the versions of the game we ensure that each version does not require nor depend on the packages and API's that the other versions depend on, for example the mobile and desktop versions would not depend on the Oculus API to function and therefore should not contain the files at all.

There are also other kart games that already exist within virtual reality, however they have not been very popular as they often fall short in their implementation, such things as have no hand movement with the game. Forcing the player to use a steering wheel, while not being completely immersion breaking, is still not perfect. The games often are single player, removing the need to program for a multiplayer environment. Another issue with many virtual reality games is that the player base in virtual reality is simply too small, as the equipment needed to play is expensive, the setup of which takes up a large amount of space which can leave many people having to spend most of their time, assembling and dismantling their virtual reality rigs. Not to mention the expensive gaming PC that a player must already own to play the games, that are also sold separately. All in all, using virtual reality for gaming is expensive and the owner must have a desire to look past its limitations and requirements to want this. Another issue with virtual reality gaming is that players with virtual reality headset often can not play games with their friends who don't own the same technology. This is why when designing this game we decided that it would be important to have a cross platform game that didn't have any limitations on who could play with each other. For us, it was important that we show how a person moving their body and hands in virtual reality could be seen moving in real time.

We decided to create the main game within the Unity engine and using the C# programming language, this will contain different scenes within the program that will serve different functions, the first being a login service. It will allow a user to create an account, then when a user has an account of their own it will allow the user to enter these details in order to login to the game. Since the game will be cross platform, a user can login to any of the different platforms and use the same credentials to gain access to the game. The login service would be programmed using the Java programming language and it will then use a MongoDB non relational database to be able to add new users when someone creates a new account. It will allow a user to access the game when they have entered the correct details. When the player enters incorrect details, they are refused entry to the game. After a successful login, the user will be presented with a screen where the user can host a game or find a game that is already being hosted by another user.

The match finding service will be programmed using the Go programming language and will connect to a Redis Database. The purpose of this system will be to keep a record of all games that are being hosted, it will keep a record of the username of the person hosting the game and that persons IP address. This will allow other players to see a list of hosted games and a list of player names but will not need the persons IP address in order to access the game. Once the user selects a game that they wish to join or host, they will be sent to a lobby screen, where they will be able to see the list of other players that are playing the same game. When all players have clicked a button to signal they are ready, the game will start.

The score keeping service will be programmed using the Python programming language and will connect to a MariaDB database. When the game has ended the players positions in the race will determine the score that they are given and their overall scores will be added to a database and sorted in order of how many points the players have accumulated. The player with the most points will be the first on the list and the player with the least points will be last in the list. When the players have completed the game they will see the position each player finished, then each players global position in the leader-board.

With the back-end of the system for such features as user login, finding an active game and for global score keeping we intended to create 3 different programs that would be running within an Amazon web services server. Within the server we will run one instance of docker with 3 different containers. Each container will house one of the programs and its corresponding database, either the Java login service with the MongoDB instance, the Go match finding service with the Redis database, or the Python scoring service with the MariaDB database.

Part of the problems that will be faced when intentionally creating a heterogeneous system, is that there will be issues with data serialisation and with marshalling and un-marshalling of the data from one language to another. The C# program will be communicating with 3 different programs that will act as services to accomplish specific tasks. The connections between the client side C# program and each of the services will be made using an IP address and port number (socket) combination. Then once the connections are made the objects that will be added to the database or that may need to be manipulated by the service programs, will be passed through these sockets as serialised objects using either JSON or XML. The purpose of these formats is that they can be used by many different languages and frameworks as they are platform independent. This does not mean that it is guaranteed to work, along the way there is bound to be issues that we encounter and that we have yet to account for in our planning. As this is a college project, the principle of which is to develop us as programmers and to take us out of the areas that we have grown comfortable, we believe this project will present us with enough of a challenge to help us develop and also be within a realistic target so that we can have a product at the end that is at least a minimal viable product that works as a game.

We don’t expect that we will have a completely polished product at the end, and are aware of our own limitations as programmers and that we have not reached the level of experience that can be gained through working in the real world. That we don’t yet understand all the possible bugs a real finished product would face or the vulnerabilities such a product would contain. Therefore we could not anticipate completely how our system could be interfered with or manipulated. In a worse case scenario, if it would be possible for all the players data to be intercepted. If such an attempt would be possible then the usernames, password and IP addresses of users could be compromised. During the programming phase of the project we simply made the game and its features function, we did not make a conscious attempt at protecting the data or transfers within the services or within the game itself. We are aware of the security issues and if the project was being released to the public, we would would have made security a priority. But, as the project is for educational purposes, we have decided to leave this feature out.

Chapter 2

Context

The goal of our project is to create a system that allows users to play a video game using a mobile device, a computer or a virtual reality headset without distinction between the devices. A system that allows a user to pick up one of these devices and join a game that would contain multiple players using any combination of the previously mentioned devices.

2.1 Objectives

- Create a kart style racing game in virtual reality.
- Create a kart racing game on a desktop gaming system.
- Port the desktop version of the game and modify it to be played on a mobile device.
- Create a heterogeneous back-end system that allows players to login, find active games to join, and keep a scoring system as a leader-board and method of player progression. These systems must act independently of the game, this would in theory, allow another game or system to be quickly retro fitted to the system and still function.

Chapter 3

Methodology

For the project we used the Waterfall methodology. We started by mapping out the requirements for the project. Then we designed how the UI would be implemented, how the user should be able to add an account on the system. We used GitHub for source control of the program, we also used Overleaf for collaboration with the dissertation.

We had regular meetings as a group and also with the group supervisor to discuss issues we were facing with the project and to make any changes that needed to be made.

3.1 Requirements

We mapped out the requirements of our project:

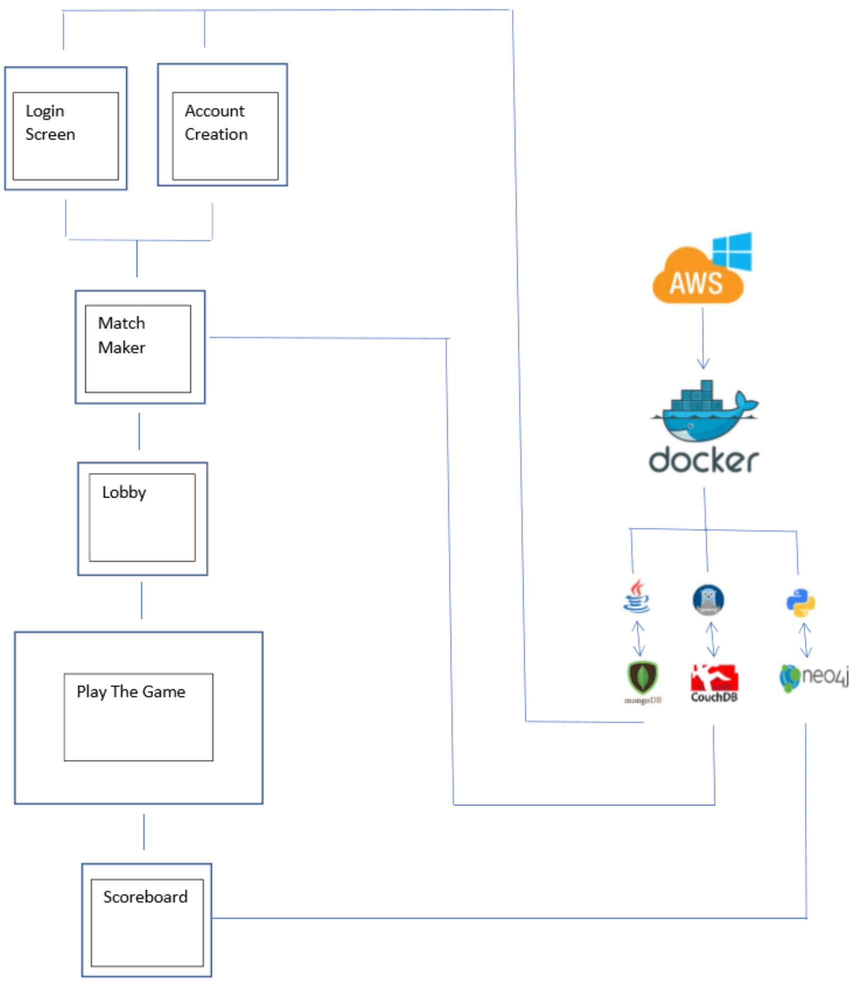
- We would need a server or some form of central point that connects the users to any back-end that we would require to run the game.
- We decided the system required a login system that would allow a user to create an account or login to an existing one.
- We then would need a system that allowed players to connect to other users.
- We would need a lobby systems where players can wait for a game to start or wait for other players to join the game.
- Once the game has ended we would need a system that could keep track of the players games and achievements such as a scoreboard system.

- The separate systems speak to the client and do not need to send data to each other, this will allow the system to be interchangeable.
- Elements of the system should be capable of being replaced by another system, without replacing other parts of the system. For example, if we decided to change the database or language for the scoreboard system, we should be able to do so without having to change any part of the login system, the match making system and minimal changes to the game itself, if any.

The game Itself:

- Must have a procedural track to allow for non repetitive play.
- Different versions of the game should connect to any instance and should not need to specify whether they can play with desktop or virtual reality headset users.
- Desktop platforms must be able to see head and hand movement from the head mount display players.
- Players must start in specified positions and must not have to enter any input for this to occur.
- The Races must have a start and end point.

3.2 Design Stage



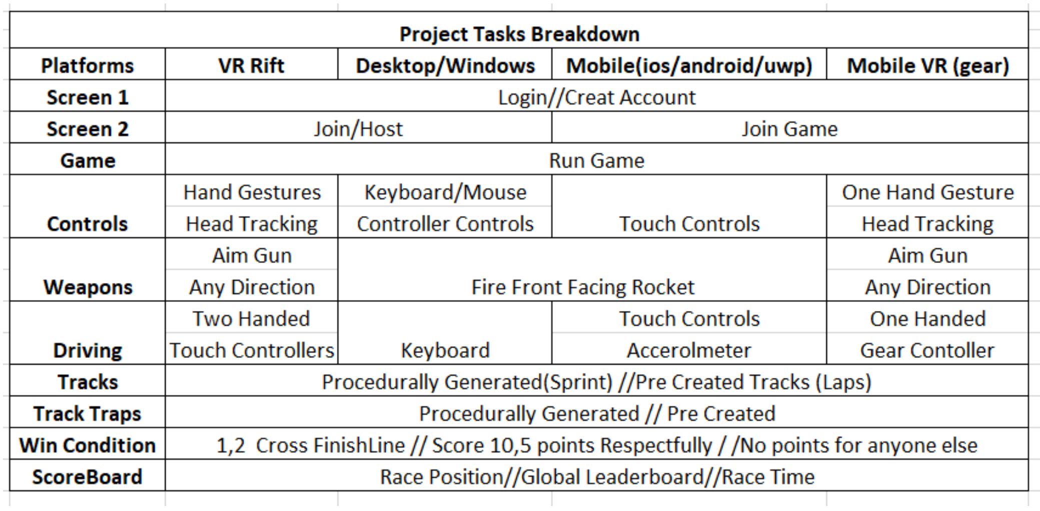
3.2.1 Overview

The design stage is a critical and vital part of a software development project. Without first setting out goals and deliverables, a project can quickly lose it way. Design and modelling have been used in various projects dating back to the Ancient Egyptians and Romans and has been used many civilization's since. This process is widely used in science and engineering to give a high-level view of the system.

The first stage is to create a design document and a model of the intended project. This document is a detailed layout of the system used to give an overall guide to the architecture of the project. It should include attributes and relationships between data and their structures along with architectural, interface and procedural designs.

The next stage is to design a model of the project. This model is then used to obtain a better understanding of the system and is referenced to help understand the flow of data and the various steps that take place throughout the project. [4]

The next stage of design is the analysis stage. This part of the process is about analyzing the performance of the various stages of the software and its requirements and limitations. This is a vital part of the process and extremely relevant to our project as there was many different technologies and programming languages used. With these factors researched and discussed, we began the task of designing our project.



Project Tasks Breakdown				
Platforms	**VR Rift**	**Desktop/Windows**	**Mobile(ios/android/uwp)**	**Mobile VR (gear)**
Screen 1	Login//Creat Account			
Screen 2	Join/Host		Join Game	
Game	Run Game			
	Hand Gestures	Keyboard/Mouse		One Hand Gesture
Controls	Head Tracking	Controller Controls	Touch Controls	Head Tracking
	Aim Gun			Aim Gun
Weapons	Any Direction	Fire Front Facing Rocket		Any Direction
	Two Handed		Touch Controls	One Handed
Driving	Touch Controllers	Keyboard	Accerolmeter	Gear Contoller
Tracks	Procedurally Generated(Sprint) //Pre Created Tracks (Laps)			
Track Traps	Procedurally Generated // Pre Created			
Win Condition	1,2 Cross FinishLine // Score 10,5 points Respectfully / /No points for anyone else			
ScoreBoard	Race Position//Global Leaderboard//Race Time			

3.2.2 The Game

During the design stage, we had to map out the system and decide on which elements were needed and how they could be implemented. From the very start we knew that the game itself must be implemented in either Unity with C# or using the Unreal 4 engine with C++. Our knowledge and experience of the Unreal engine and with with the C++, whereas we already have a working knowledge of Unity and C#. So we went with Unity. Using any form of third gaming engine would not be practical as the libraries for the virtual reality devices have been focused on these two game engines.

3.2.3 System Back-end

We then needed to consider the back-end of the system and which platforms were available. Firstly we could have used a physical computer that we personally owned and set it up as a running server. This would not have been a bad idea as we already own personal computers that could have effectively done this. Other options included using cloud based servers such as Amazon Web Services(AWS) or Google cloud. We had setup some instances of Google cloud, we even ran them and began launching some of the programs on the system. From prior experiences, we had encountered situations were we had been charged by Google cloud for use of such cloud instances and had found AWS to be a much cheaper rate. We also had some prior experience with AWS and opening firewall rules for this service, as we are using sockets, this was certainly going to be needed. For this reason, we went with AWS.

3.2.4 Login system

Development of the login system started with creating two Java programs that used sockets to make a connection to one another. A socket is one endpoint of a two-way communication between two programs running on a network with a corresponding socket on the other end. Each socket is bound to a port number that the TCP layer can identify that allows data to be sent back and forth between the two programs.

Before development went any further a decision was made to use MongoDB as our database for the user login. The installation of MongoDB involved logging into the AWS virtual machine we are using for our project. For this project we decided to use the free version of MongoDB known as the community version, using the current stable release (4.0.3) (MSI).

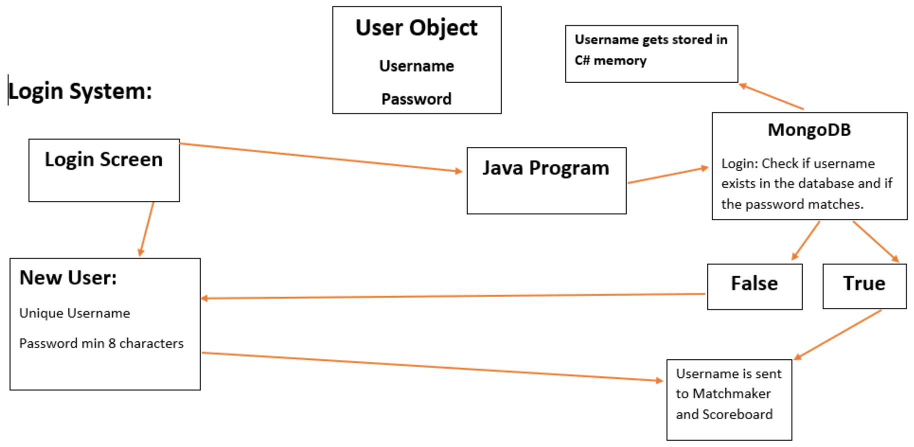
Once the two Java programs were up and running a number of basic tests were performed to verify the programs were working correctly by sending and receiving basic information across the connection. The next step was to take the existing program and use it to receive a connection from the Unity game scene in the user login details page. This required making major modifications to the existing java programs. For this project our database is called usersdb and our collection is called users.

The modifications involved creating several new java programs to receive a connection from the server and connect to the database. Then we created a user object in the java program that consisted of a username and password. We created an interface called database connection that contains the methods to add users, find users and verify passwords.

This involved making a connection using a C# login script in visual studio, through sockets, on port numbers 5000 and 5001. The login details page in the unity project required a username and password. The username and password are represented using strings in C#. The username and password are game objects in unity that allow a user to type the details into input fields. In unity every object in a game is represented as a game object. Game objects vary from game characters to collectable items or text fields that exist in a game scene in Unity.

Once the C# script was set up to make a connection through the port numbers and connected to the java application on the virtual machine it was time to connect to the database. A connection to the MongoDB is made using the java application. This allows users to create a new account for login purposes using a valid username, must be unique and not already exist in the database, and create a password that must be eight characters or more. The users information is then stored in memory in C#.

This users information that has been sent to the MongoDB where it either creates a new valid user or finds an existing user in the database is validated and the username is then sent to the matchmaker scene.
[2]


Login System:
User Object
Username
Password
Username gets stored in C# memory
Login Screen
Java Program
MongoDB
Login: Check if username exists in the database and if the password matches.
New User:
Unique Username
Password min 8 characters
False
True
Username is sent to Matchmaker and Scoreboard

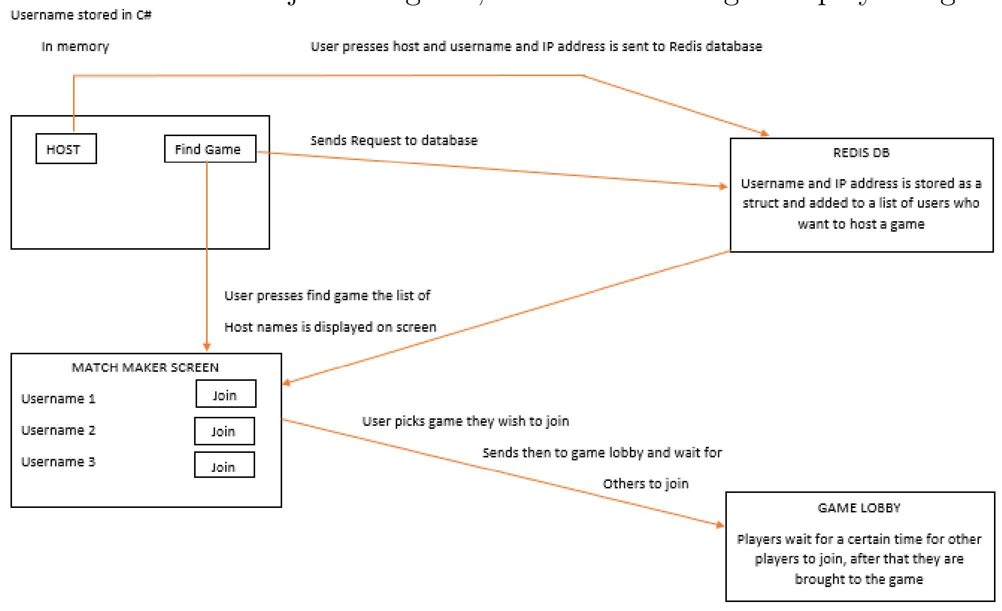
Overview of Log-in System

3.2.5 Match Maker

The match making system allows users the option to either host a game or find a game to join. Hosting a game means the user is hosting a game on his own network/machine. Joining a game simply allows you to join a game that is being hosted by someone else.

The user name that has been stored in memory in C# is sent to the match maker. If the user chooses to host a game his username and IP address is sent to the redis database. The username and IP address is then stored in a struct and added to a list of users who want to host a game. The username is then sent back to the match maker screen and a join button appears next to his name. Once other users decide to join the game, the users are redirected to the game lobby screen. They wait here for other players to join, and are then brought to play the game.

If the user decides he does not want to host a game, they clicks on the find a game button which sends a request to the redis database that sends back a list of users who are hosting a game. The user then clicks the join button next to the hosts name. The user then gets redirected to the game lobby where he waits for others to join the game, and then is brought to play the game.


Username stored in C#
In memory
User presses host and username and IP address is sent to Redis database
HOST
Find Game
Sends Request to database
REDIS DB
Username and IP address is stored as a struct and added to a list of users who want to host a game
User presses find game the list of
Host names is displayed on screen
MATCH MAKER SCREEN
Username 1
Username 2
Username 3
Join
Join
Join
User picks game they wish to join
Sends then to game lobby and wait for
Others to join
GAME LOBBY
Players wait for a certain time for other players to join, after that they are brought to the game

Overview of Match Maker System

3.2.6 Score Board

The scoreboard will be the last window the user will interact with. Once the race has ended, the game sends a list of the player objects containing the player username and the points the player received for their position in the race. Once the list is received, a loop runs through the objects and the player name and points received are printed to screen using GUI text field in Unity.

Next the program will serialize the list as XML for the purpose of sending to the MariaDB database running on the Virtual machine. Before this is done the object must be tagged with the Serializable attribute. It then creates an XML document and declaration and a loop sends the object to the XML file. (The file is not actual created; the XML is saved to a variable). This is done using the XMLSerializer class. [6]

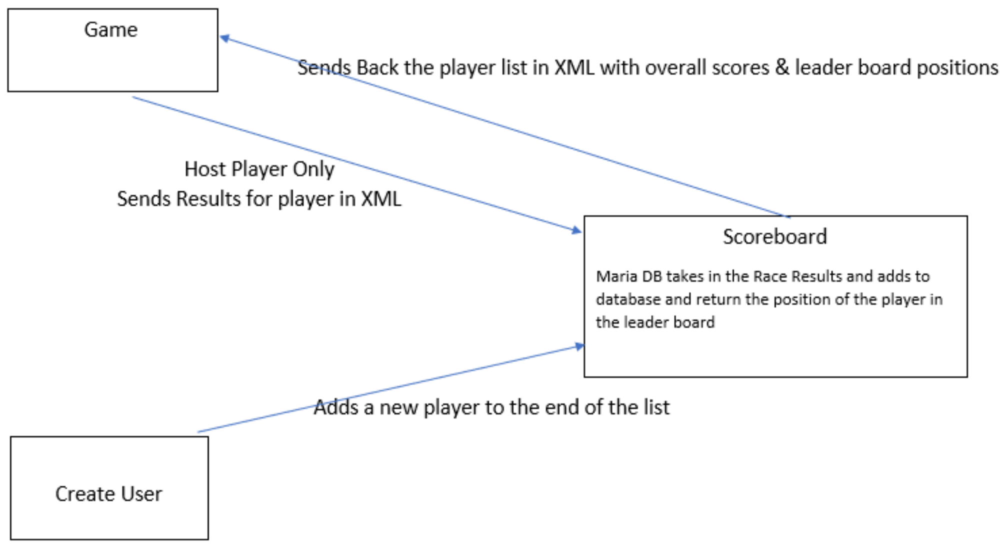
The program then makes use of the TCP Client class in C#. The program makes a connection to a socket listening on the virtual machine. Once the connection is made the variable containing the XML is passed to a byte array which is sent using the Network Stream class. This is done by reading the data in and using the write method in the Network Stream class. [5]

The python program on the virtual machine then receives the XML as a byte stream it decodes the bytes, then it converts them to string, and using ElementTree, converts them back into XML. It then passes the XML into a second python file called ScoreboardDB. There it takes the XML passed from the ClientConnection file and passes UserName, Score into two lists, one called playerNames and the other called playerScore.

The program then connects to the MariaDB and runs a query to either add the player, if they don't already exist or to update there score if they do exist. This is wrapped in a try/catch to bring up an error if for any reason the data can't be added to the database or if a connection to the database can't be made.

After the database has been updated with the latest results, the rankDB definition is called and it assigns the players rank to a list called playerRank. There rank is based on there total score from all there races. The writeToXML definition then adds the results to an XML file, which is not an actual file but a BytesIO stream. This is used to add the XML header which can't be added in python without creating a file or using a binary file stream as used here. This is then passed back to the ClientConnection file. Then on the same connection that the XML was received the updated XML is returned to the C# program.

The next step in the program is to receive the updated XML using the StreamReader class, making use of the TCP getSteam method available in C#. This data is then read in and saved as XML file in order to deserialize back to an object. This is done by again using the XMLSerializer class, but this time using the deserialise method. The file is opened using the FileStream class and a loop runs over the XML and deserialises the file back into a list of objects. This new and updated list is then rendered to screen using GUI text field in Unity.


Game
Sends Back the player list in XML with overall scores & leader board positions
Host Player Only
Sends Results for player in XML
Scoreboard
Maria DB takes in the Race Results and adds to database and return the position of the player in the leader board
Adds a new player to the end of the list
Create User

Overview of Score Board System

3.3 Implementation

3.3.1 Game Engine

Our first decision was to choose a game engine to design the game with. As we had some previous experience with Unity through other modules of our course, and we all had the software installed on our personal computers, it was an easy decision. As Unity works with the C# language, it was much better for us as we all had the necessary software and editors already installed.

3.3.2 Virtual Machine

The process of choosing a cloud-based platform involved researching what was available to us for free. We explored using a Google cloud platform machine but we choose to use AWS. We choose to deploy an AWS virtual machine using a windows 10 operating system. We all had previous experience working with an AWS virtual machine through other modules in our course and setting up the machine was quite painless. We also had previous experience working with sockets to make connections on an AWS machine from a previous module. For ease of access we uploaded the AWS.rdp file to our shared github repository. This allowed us to connect to the machine while working together and individually.

Working with AWS can sometimes be difficult as it only allows one user to access the machine at any given time. This resulted in the creation of a messaging group between all group members which was used to collaborate, and delegate time spent on the virtual machine. [1]

Installed on the Virtual Machine

- MongoDB, RedisDB, MariaDB
- Eclipse Oxygen IDE
- Visual Studio Code IDE
- Docker
- Heidi
- Go Lang compiler
- Python compiler

3.3.3 Databases

Login Database

The choice and implementation of the databases took a bit longer to finish. We decided to use several database in order to display the heterogeneous aspect of the project. As the project made use of several databases, we had to make sure we had the right one for each section of the project. For the log-in system we had first decided to use a MongoDB Atlas database. This is a cloud-based version of MongoDB. It had all the same capabilities of the original but is hosted on a cloud-based service of your choice.

This was not a good choice for us as we would have been charged for the service, and as we would be been testing and sending data so often, we could see ourselves breaking the free threshold. With this in mind, we decided to go with MongoDB community edition as it was an open source release. We would also have full control of the database and be able to turn it off and make changes whenever we had to. We also had some experience using MongoDB from a previous module.

To install MongoDB:

- Click on the file just downloaded which opens the installer.
- Click Next.
- Accept Terms and Conditions and click Next.
- Choose Complete Setup.
- Click Install.
- Click Finish.

To setup MongoDB:

- Open the Windows cmd prompt and navigate to the following folder:
- C: Program Files MongoDB Server 4.0.3 bin
- Start the mongo daemon as follows:
- Mongod
- In a new Windows cmd prompt navigate to the bin folder:

- C: Program Files MongoDB Server 4.0.3 bin
- Start mongoDB by entering this command:
- mongo

Match Maker Database

For our match making system we needed a database that would work well with the Go programming language. We first looked at using a CouchDB. This proved a bit difficult to implement with Go. We had the database set up but were unable to get the program to send data to it. Also, with CouchDB working with a file-based system we did not think it was a good fit.

We then tested using a RedisDB for our purposes. This database was a better fit for us, as it was an open sourced and made use of in memory data storage. It was also very easy to implement using Go.

To install RedisDB:

- Download the .tar file installation package
- CD to the .tar file location
- Open a cmd prompt window and run the command
- tar vxf ¡ tarfile name ¿
- Then run the command
- sudo ./install.sh -s /var/run/redislabs

ScoreBoard Database

For the scoreboard database, we originally looked at using a NEO4J database. The reason for this was because we were using Python to connect to the database, and we had heard that NEO4J worked well with it. After installing and testing it, we found it was not a viable option for us. NEO4J is a graph database management system and was not appropriate for our needs. We next looked at using a MariaDB for the scoreboard. This is an SQL relation database and was a much better option for the scoreboard.

As we were using MariaDB, we decided to install HeidiSQl on the virtual machine. This is a tool that allows a user to create and query a database using a user interface, rather than write the SQL commands from scratch. As this would be a time saving advantage, we installed it on the virtual machine.

To install MariaDB:

- Click on the downloaded file.
- The MariaDB setup wizard will launch, click next.
- Accept the License Agreement and click next.
- Here you can set your Root password and select to allow to access the database from remote machines.
- Click next on the default instance properties
- Click install

To install HeidiSQL:

- Run HeidiSQL
- Click the new button in session manager
- Select Network type MySql (TCP/IP)
- Set Hostname as 127.0.0.1
- Set User as root from MariaDB setup
- Set Password as password from MariaDB setup
- Set Port to 5004 (or any available port)
- Then click open

3.3.4 XML/JSON

For the project, we could have easily used one of these for the sending of information throughout the system. Since this was a learning experience, we decided to use both of these mediums in order to gain a better understanding and working knowledge of them.

3.4 Integration and System Testing

We were going to use Loadview system to test the scalability of the project. But with the version of AWS that we were using we could not perform testing on it without going over the free tier limit, as on occasion we were already going over the free tier limit.

We have tested on multiple devices that we can send and receive data from each of the databases as needed throughout the game.

3.4.1 Integration Testing

Username Validation

When the user enters their username to create a new user, if they enter a name that has already being used, they are prompted to try again. They should not be allowed to use a username that is not unique.
Result: Pass

Password Validation

The password must be at least eight characters long or the user is prompted to renter there password and is not allowed to proceed until the password is entered as required.
Result: Pass

Login Validation

When the user attempts to login with their username and password, it is sent to the MongoDB as a user object. The database is then searched to confirm that the user exists and that they entered the correct password for that username. If they enter incorrect information they are shown an error that says ”invalid log in details”. They then have the option to click okay

and they will be brought back to the login screen and can enter the correct information or create a new user account.
Result: Pass

Host Validation

When a user decides to host a game. There name and IP address are sent to the RedisDB and added to a list of hosts. Then there username is displayed whenever a user wants to look for a game to join. When a user tries to join a game, they should receive an up to date list of hosts that have opted to host a game.
Result: Pass

User can Respawn

When racing, if the user goes off the track, they can press the R key (Desktop). This will respawn the player back onto the track at a position close to where they left the track.
Result: Pass

User can Jump

At any point in the race the user can press space bar in desktop and A in VR and there vehicle will make a slight jump.
Result: Pass

Controls testing desktop

Users will be able to control the cart using their keyboard keys W, A, S D. W key will control going forward. A key will control turning leaf. D key will cover turning right. S key will cover going in reverse.
Result: Partial Pass: Going forward, turning left and turning right all worked as expected. Reversing was unresponsive.

Controls testing VR

A button to jump, B button to respawn. The two hat analog sticks pressed at the same time will re-center the camera of the players camera.
Result: Pass:

Scoreboard Race End

After the race is completed the results are sent from the C# program to the MariaDB on the AWS. The database is updated with the results from the race and the updated rankings are sent back to the host. The host then displays the results for the rest of the users to see.
Result: Fail Scoreboard needs to be implemented. The scoreboard works when fed test data.

Scoreboard New User

After the race is completed the results are sent from the C# program to the MariaDB on the AWS. If there is a new user, their record is created and their score is set to their results from that race. The updated rankings are then returned to the host to display.
Result: Fail Scoreboard needs to be implemented. Scoreboard can add new user. Has being tested with test data.

Scoreboard displays only the users who raced

After the race is completed the results are sent from the C# program to the MariaDB on the AWS. The scoreboard should return a list of results with only the users who raced.
Result: Fail Scoreboard needs to be implemented. Scoreboard will display only the user who have raced, when fed test data.

Scoreboard displays updated world rankings

After the race is completed the results are sent from the C program to the MariaDB on the AWS. The results that are returned should display updated world rankings.
Result: Fail Scoreboard needs to be implemented. The results are updated when the scoreboard is fed test data.

Race start, players don't spawn in same location

At the start of the race the players should be spawned in four separate locations, before the start line.
Result: Pass

Track generation

At the start of each race a track should be generated for the players to race on. The track should have a start and finish line.
Result: Pass

Procedural Track generation

At the start of each race a track should be generated for the players to race on. The track should be a different procedurally generated track each time.
Result: Pass

3.4.2 System Testing

Full Game Test

User logs in to there account. Joins a game hosted by another user. Races the length of the track. Is able to collide and be rammed by other players, can respawn if exits track or gets turned around. Finishes the race and see there results.

Result: Fail User is able to sign in and race a full race with other players. Collision's don't cause any issues and the players can respawn when needed. Issue at end of race and the displaying the scoreboard. The scoreboard is not implemented.

Chapter 4

Technology Review

4.1 Unity 3D

Unity 3D is a a game development environment created by David Helgason who setup Unity Technologies. 59% of all virtual reality games are made using the Unity 3D engine. Unity works on 30 platforms, including Windows, iOS, Android, Nintendo Switch, Playstation 4, Oculus Rift, etc. Unity competes with Epic's Unreal Engine, the game engine behind Fortnite and many games on the PS4 and Xbox One. 50% of all games are made using Unity 3D. As we are making a virtual reality game, the two engines we could use are the Unity 3D engine or the Unreal 4 engine. Though the Unreal engine creates very beautiful games and is very well optimised, the programming language is C++. We have never used C++ and starting a new language would make the process more complex. The Unity engine used the C# language, which we have used previously.

4.2 Unity UNET

UNet is a feature in Unity that allows the programmer to create multiplayer or networked games over a network. It accomplishes this by allowing one player or service to act as a host or dedicated host. A standard host is also a client, a dedicated host is just a server and does not have a client connection. Each client sends data to the host and receives all data from the host instead of communicating with the other clients. UNet as of 2018 is deprecated and is planned to be replaced, although there is no information yet on its replacement. Part of the reason for UNet being replaced is due to data being sent from client to server, then to all other clients and has a latency issue as expected due to the path the data uses, as latency is usually the amount of time it takes for data to travel from client to server and back. This can be seen even in our program, when a player jumps their character, the players character can be seen stuttering but doesn't have such a noticeable effect when the character is not jumping.

4.3 Oculus Rift

The Virtual Reality headset we used for our project is an Oculus Rift, this is one of the top head mounted displays on the market. Along with the Rift, Oculus has also released two development kits before releasing the Rift. They have released a mobile VR kit known as the Oculus Go. In spring 2019 Oculus will also release two more units, one known as the Rift S that will be similar to the Rift and one other unit known as the Oculus Quest, which will be a stand alone unit that will not require a computer to operate.

The Rift connects to a gaming graphics card with a HDMI port and a USB 3.0 connection for the head mounted display. The headset comes with two touch controllers and two external sensors that also each require a USB 3.0 connection. Due to the problems that occur when using only two sensors, a third sensor is recommended for a much smoother experience. The position of the controllers and the headset in 3D space is captured using the external sensors.

4.4 Java

Java is a high-level programming language developed by Sun Microsystems. It was originally designed for developing programs for set-top boxes and handheld devices, but later became a popular choice for creating web applications. The syntax is similar to that of C++ but it is strictly an object orientated programming language. Java programs contain classes and methods. Classes are used to define objects and methods are assigned to individual classes.

Java programs are not run directly by the operating system. Instead, Java programs are interpreted by the JVM, Java Virtual Machine, which can be run on multiple platforms. This means all Java programs can be run on different platforms such as Mac, Windows and Unix computers. In order for Java applications or applets to run at all the JVM must be installed.

4.5 C#

C# is an object orientated programming language used for networking, web and game development. Dutch developer Anders Hejlsberg formed a team in 1999 during the development of the .NET framework to create a new C like Object Orientated Language (COOL).

C# is used to develop robust applications through the .NET framework. C# is a very similar language to java and C++ and generally developers who are familiar with java can adapt to C# very quickly. C# is the programming language used for game development in Unity.

4.6 Go Lang

Go Lang was conceived by Robert Griesemer, Rob Pike and Ken Thompson when they started sketching the goals for a new language on a white board on September 21, 2007. By January 2008, Ken Thompson had started working on a compiler with which to explore ideas. The compiler generated C code as its output. By mid 2008 the language had become a full-time project and had settled enough to attempt a production compiler. Go became a public open source project on November 10, 2009. There are now millions of Go programmers, or gophers, around the world. Go's success has far exceeded all expectations.

At the time of Go's inception, only a decade ago, the programming world was very different from today. Production software was usually written in C++ or Java, GitHub did not even exist and most computers were not yet capable to run complex algorithms, and IDEs such as Visual Studio and Eclipse were the only real option. Throughout its design, Google have endeavoured to reduce clutter and complexity. There are no forward declarations and no header files as everything is declared exactly once. Go Lang is a relative new, yet popular language and seems to be on the rise in terms of application development.

4.7 Python

The Python programming language was created by dutch programmer Guido Van Rossum who started developing the new script in the late 1980s and finally introduced the first version of that programming language in 1991. Today, it is developed by the Python Software Foundation. Python is a multi-paradigm language, Python programmers can accomplish their tasks using different styles of programming: object oriented, imperative, functional or reflective.

Here are some of the principles that python is based on:

- Beautiful is better than ugly
- Simple is better than complex
- Complex is better than complicated
- Readability counts
- In the face of ambiguity, refuse the temptation to guess
- There should be one, and preferably only one, obvious way to do it
- If the implementation is hard to explain, it's a bad idea

Python has seen somewhat of a resurgence in recent years as an effective programming language for machine learning and the development of neural networks as we recently discovered through our Emerging Technology module.

4.8 MongoDB

Mongodb is an open source non relational database management system. MongoDB was created by Dwight Merriman and Eliot Horowitz, who had encountered development and scalability issues with traditional relational database approaches while building web applications at DoubleClick, an online advertising company that is now owned by Google Inc. The name of the database was derived from the word humongous to represent the idea of supporting large amounts of data.

MongoDB uses collections and documents unlike relational databases that use tables and rows. A record in MongoDB is a document that contains a data structure made up of field and value pairs. MongoDB documents use a varient similar to JavaScript Object Notation (JSON) called Binary JSON known as BSON that allows for more data types. The fields in the documents are similar to columns in a relational databse and they can hold a large variety of data including other documents to arrays of documents.

Documents in MongoDB must use a primary key as a unique identifier. The (id field) is added by MongoDB to uniquely identify the document in the collection i.e.(id, Name, Address) id is the object id/primary key. Collections contain sets of documents and act as the equivalent of tables in a relational database.

MongoDB does not require predefined schemas and it stores any type of data. Tihs provides scalability and felxibility compared to relational databases which makes it a useful database for companies running big data applications. One of the advantages of using documents is that these objects map to native data types in a number of programming languages. Also, having embedded documents reduces the need for database joins.

MongoDB is available in community and commercial versions through vendor MongoDB Inc. MongoDB Community Edition is the open source release, while MongoDB Enterprise Server brings added security features. [7]

4.9 Redis DB

Redis (REmote DIctionary Server) is an open source (BSD licensed), in-memory data structure store, used as a database. It supports a large variety of different types of data that can be stored ranging from strings and lists to hashes and sorted sets. Redis allows you to run atomic operations on these types, like appending a string, increment the value in a hash, pushing an element to a list, computing set intersection, union and difference or getting the member with highest ranking in a sorted set.

Key features of redis are:
High-Level Data Structures: Provides five possible data types for values: strings, lists, sets, hashes, and sorted sets. Operations that are unique to those data types are provided and come with well documented time-

complexity (Big O notation).

High Performance: Due to its in-memory nature, the project maintainer's commitment to keeping complexity at a minimum, and an event-based programming model, Redis boasts exceptional performance for read and write operations.

Lightweight With No Dependencies: Written in ANSI C, and has no external dependencies. Works well in all POSIX environments. Windows is not officially supported, but an experimental build is provided by Microsoft.

4.10 MariaDB

MariaDB is an open source relational database management system (DBMS). MariaDB is the compatible drop-in replacement for the widely used MySQL database technology. MariaDB intends to have high compatibility with MySQL and exact matching with MySQL APIs and commands. MariaDB's API and protocol are compatible with those used by MySQL, and also some other features to support local non-blocking operations and progress reporting.

MariaDB is developing continuously and any new updates are transmitted to end users very fast with updated features like bug tracking that can be viewed in detail. They also offer a cluster database intended for commercial use that enables multi-master replication.

MARIADB VS MYSQL

- MariaDB has superior query performance.
- MariaDB has a more open source attitude.
- Making the switch to MariaDB is very easy.
- Galera implementation is better in MariaDB.
- MariaDB is available as an option with some hosting environments, like RackSpace Cloud.

4.11 AWS

Amazon Web Services (AWS), the pioneering cloud computing platform provided by Amazon.com, emerged from separate internal initiatives at Amazon over 16 years ago to both aid developers and also improve the efficiency of the company's own infrastructure.

Publicly launched on March 19, 2006, AWS offered Simple Storage Service (S3) and Elastic Compute Cloud (EC2). By 2009, S3 and EC2 were launched in Europe, the Elastic Block Store (EBS) was made public, and a powerful content delivery network (CDN), Amazon CloudFront, all became formal parts of AWS offering. These developer-friendly services attracted cloud-ready customers and set the table for formalized partnerships with data-hungry enterprises such as Dropbox, Netflix, and Reddit, all before 2010.

4.12 Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host.

Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines.

4.13 JSON

JSON stands for Java Script Object Notation. JSON is a lightweight format for the storing of and transporting of data. JSON is a text format language that is completely language independent. JSON is used mainly to transfer data between a server and web or mobile applications. [3]

4.14 XML

XML stands for eXtensible Markup Language and is used for storing information almost everywhere around the world. XML is independent of propriety software applications which means when you write in XML it can be passed around and used in a variety of computer systems. This allows for cross platform communication and data transference.

XML is structured in a tree format beginning with a (Parent) root element that branches to a (Child) element that can contain an attribute that branches to a (Sub-Child) element that can contain different information.

Chapter 5

System Design

5.1 Overview

The purpose of this project was to create a multiplayer racing game, with the capability to be played on both desktop and VR headset and mobile. The game will have a log in system and will be handled by a C# and Java with usernames and passwords saved on a Mongo database on a virtual machine.

Players will have the ability to either host or join a game in progress. This match making system will be handled by a mix of both C# and Go programs. It will involve saving a list of players willing to host on a Redis database on a virtual machine and sending back the list to the game for player to join them.

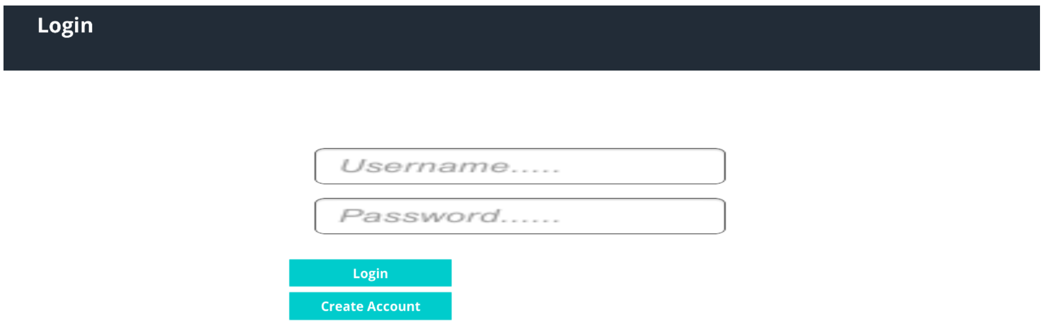
The game will be a kart racing game, with desktop players using a kart while the VR players will be using a motorbike. Player on VR will be able to see the desktop player, and players on desktop will be able to see the VR players. Desktop player will also be able to see the VR players hand movements and gestures.

The game will have a procedural track created as the game starts. This track is created on the hosts machine and is shared with the players who have joined. Controls for the desktop will be: W for forward, A for left, D for right, S for reverse and spacebar to jump and R to respawn. VR controls will be A to jump and B to respawn.

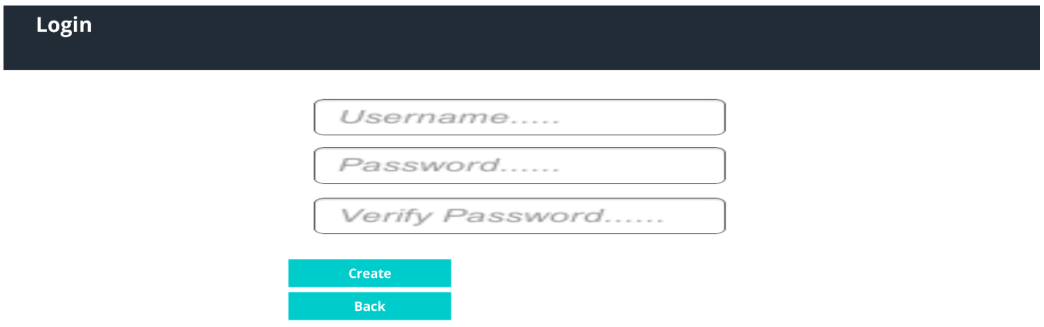
Once the race is over, the player names and scores are sent to a scoreboard. These results are then sent to a Maria database running on the virtual machine and is handled by C# and Python programs. The database will update the new results and a leader board is sent back to the host and displayed on screen.

5.2 Log-in System

When the user starts the program, they are presented with a login page. The user is prompted to enter a username and password to proceed. The username must be unique, having multiple users with the same username would cause confusion, and the password must be a minimum of eight characters long. If the user already has an account they simply enter their details and click login to proceed.



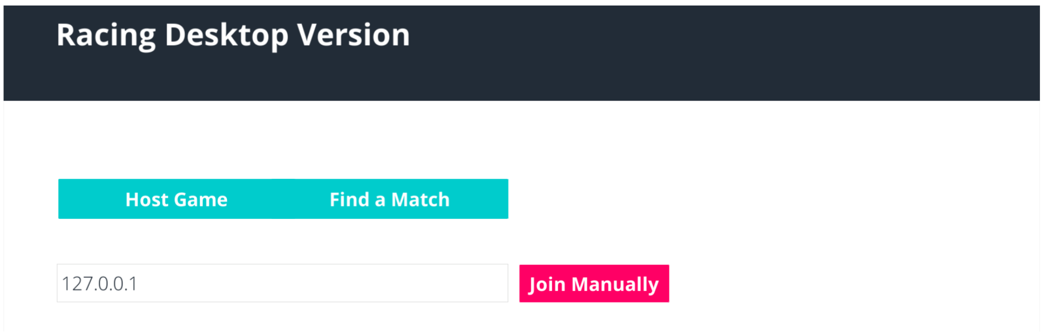
If the user is new to the game and does not have an account set up they must click create account and register as a user. Once the user is registered and logged in they get redirected to the match maker page to choose what they want to do next, host or join a game.



5.3 Hosting a Game

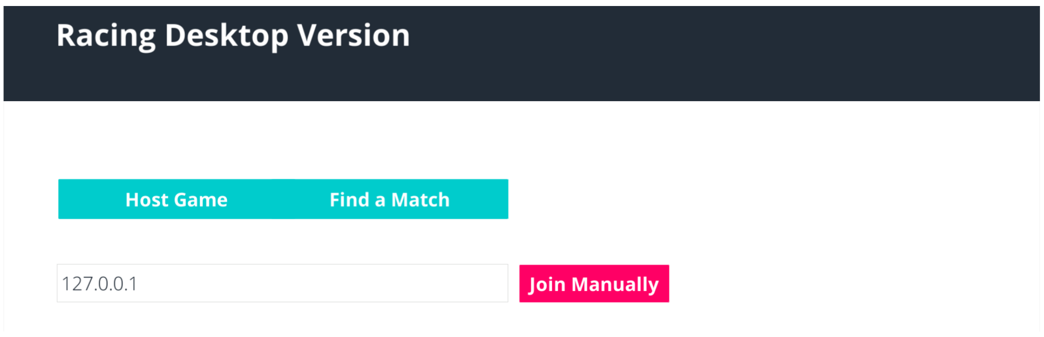
Once the user has logged in or created a new account, they are presented with the option to host or join an already hosted game. If the user wishes to host a game, they can press the host button. This will then take the users player name and IP Address and store them as a player object.

This object is then sent to the virtual machine and is received by a Go program, the object is then pushed into a struct. This struct is then sent to the Redis data base and stored until someone wants to join a game. The host is then sent to the game lobby to wait for other players to join.



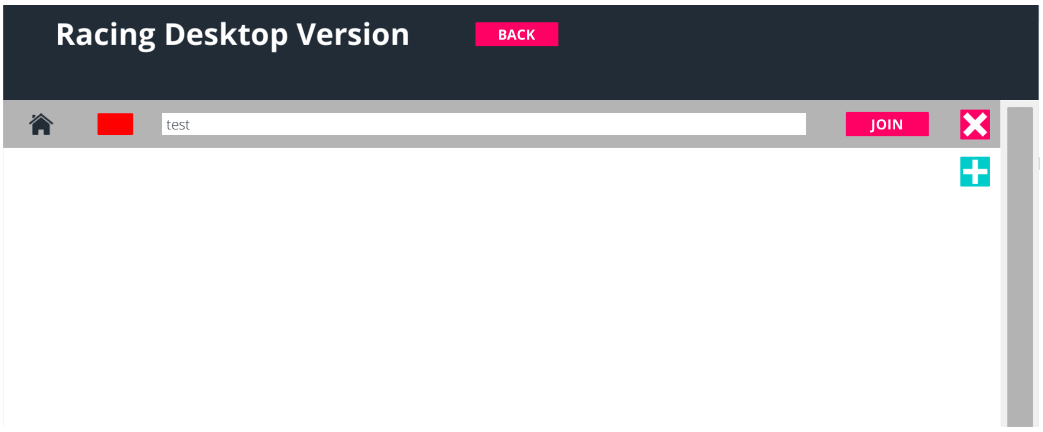
5.4 Joining a game

When the user clicks find a game. The program sends a request to the Redis database and gets back a list of games being hosted. The player can then select to join one the games that are displayed. When the user has joined the game they are brought to the lobby page.



5.5 In Game Lobby

The host player is sent to the lobby immediately and then they wait for other players to join. As other players join the lobby there name is displayed on the screen. At anytime while in the lobby they can click join. When all players in the lobby click join they are presented with a counter that counts down from three. They are then sent to the race start.

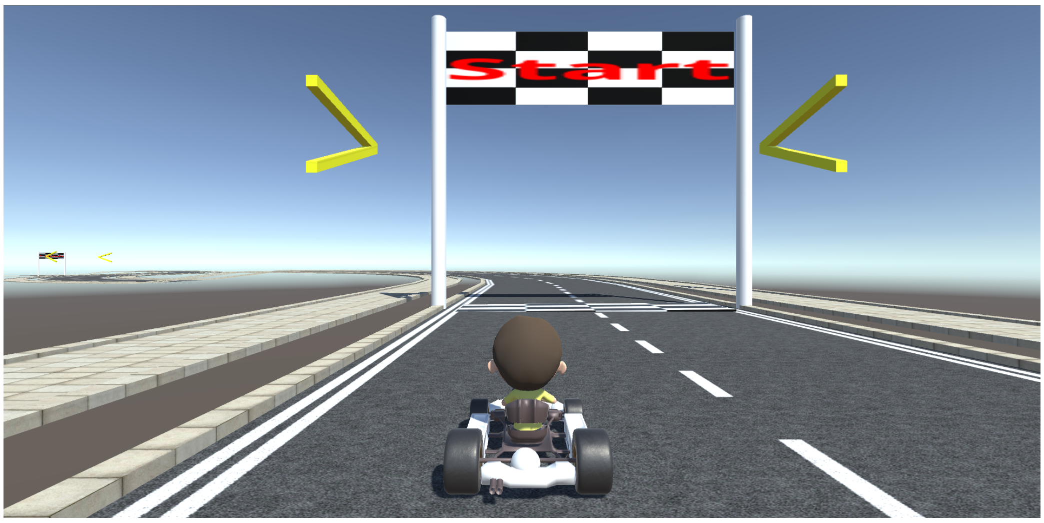


5.6 The Game on Desktop

The game on desktop consists of the player controlling a kart with the game character Morty (desktop only). The player lines up alongside his competitors to wait for the race to begin. Once the race starts the players race each other to the finish line on a procedural generated track that varies for each race.

Desktop Player Controls:

- W moves the player forward.
- A moves the player to the left.
- S moves the player in reverse.
- D moves the player to the right.
- R re-spawns the player.
- Space-bar allows the player to jump.



5.7 The Game in Virtual Reality

The virtual reality section of the game is implemented differently from the desktop version of the game. In place of using a traditional monitor, the virtual reality devices use a head mounted display unit that utilises the full motion and direction of the users head, as if the person is actually in the game. With some of the virtual reality devices, players can use controllers that track the positions of the players hands. The players can use the buttons on the controllers to operate some functions of the game, such as jumping or grabbing an object.

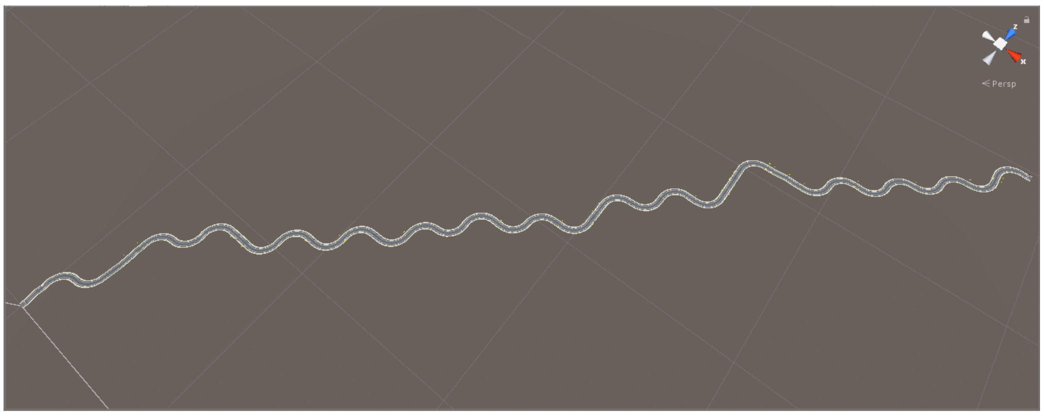
For our implementation of the game, we used the A button to jump, the B button to respawn. The two hat analog sticks pressed at the same time will re-center the camera of the players camera. We then imported some hands that are tracked by 3 sensors that are placed around the parameter of the players play space. the players hands move in real time with the player and can act just like real hands, given the right animations.

We were able to implement these features using the Oculus SDK. Some of the issues that had presented during the project were usually to do with the hardware rather than software. For instance the left earphone on the headset was damaged and had to be returned to Oculus which took two months for it to be returned.

5.8 Procedural Track

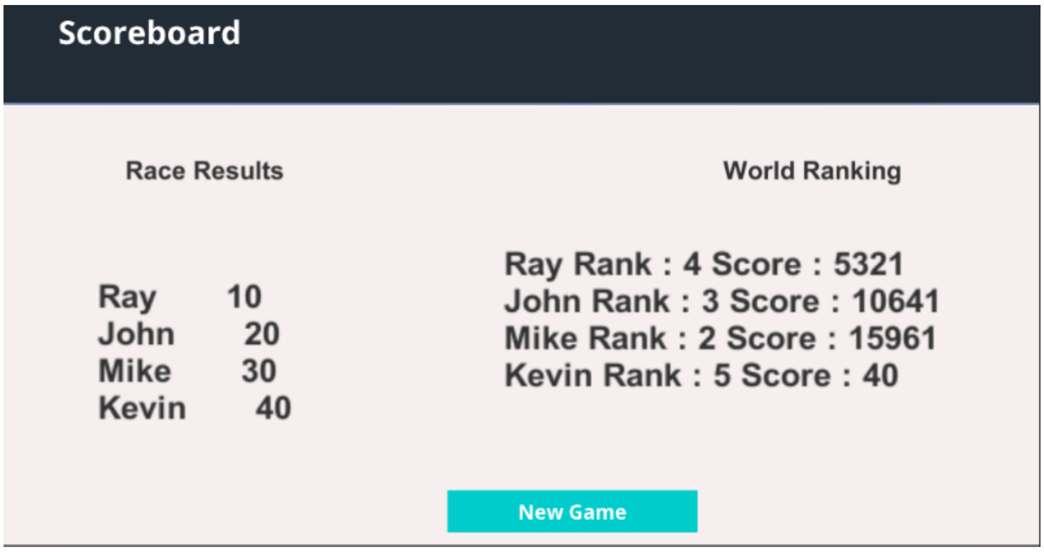
A decision was made for the host to generate the race tracks in a procedural manner to keep consistency throughout the different platforms of the game. This would guarantee each player who joins a game would be racing on an identical track to everyone else who is playing either on desktop or VR. This also reduced the need to create multiple tracks, as the tracks that are generated are different for each race. This was a good decision on our behalf as the original plan was to create the race track on desktop and then recreate it in VR. This would have been a very time consuming and troublesome process.

Originally we had mathematically generated this track, however a mistake was made that sometimes caused a bug where the track would generate a piece out of line on the track. So the approach was changed. Instead of mathematically calculating where each piece should appear, we created a set of track pieces, for example, one straight road piece and a left and right turn. Then some pieces that allow the track to descend or ascend. Each prefab for the track has two markers, one marker at the starting position. Then another marker at an ending position, which is the point where the next piece should join on to the current piece. So when the program creates the track, it slots the road pieces together like lego. Generate one piece at point 0.0, then randomly pick the next track piece and place its starting marker at the ending marker of the previous piece.



5.9 Scoreboard Screen

Once the race is over, the host sends a list of player objects that includes their username and result to the scoreboard scripts, which then serializes them. It then passes them to the MariaDB as an XML list. The database then updates their records or creates a record if this is the players first game. This is then returned to the scoreboard and displayed to all the players.



Ray	10
John	20
Mike	30
Kevin	40

```
Ray Rank : 4 Score : 5321
John Rank : 3 Score : 10641
Mike Rank : 2 Score : 15961
Kevin Rank : 5 Score : 40
```

Chapter 6

System Evaluation

6.1 Testing

Throughout the development process the program was tested more and more as each new piece of the backend system was added. The game performed very well at each stage and there were no catastrophic failures encountered just minor tweaks here and there regarding scaling and audio.

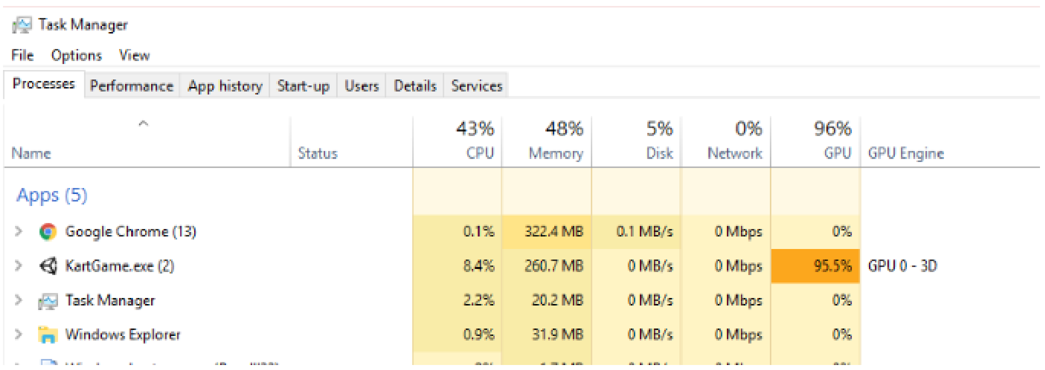
In theory the system should be able to be retro fitted to be used to facilitate a number of different types of games. We did not have time to conduct a test of this nature but it is definitely something that will be explored during future development.

6.2 Drawbacks and Limitations

Testing proved difficult on a standard PC. Unity 3d is a large and powerful piece of software that requires a lot of resources to run. This resulted in slow performance when testing for some members of the team. When tested on a standard laptop we found the game was extremely taxing on the GPU, 86 - 95% output at times.

As a result the game would work fine from the login process to the procedural track generation but would sometimes result in the game character not begin able to move forward. Alternatively when run on custom built gaming PCs, which two members of the group have, they experienced no problems at all.

We encountered same issue with the game character while filming the video demonstration for our project. While recording our video on a non gaming PC, while running Unity and screen o matic, the game character became unresponsive to any forward movement. Having the two pieces of software running simultaneously caused a huge drain on the resources of the standard graphics card on the PC. Once the recording software stopped the game character began to behave as normal.



Name	Status	43% CPU	48% Memory	5% Disk	0% Network	96% GPU	GPU Engine
Apps (5)							
Google Chrome (13)		0.1%	322.4 MB	0.1 MB/s	0 Mbps	0%	
KartGame.exe (2)		8.4%	260.7 MB	0 MB/s	0 Mbps	95.5%	GPU 0 - 3D
Task Manager		2.2%	20.2 MB	0 MB/s	0 Mbps	0%	
Windows Explorer		0.9%	31.9 MB	0 MB/s	0 Mbps	0%	

Another slight drawback was working with MongoDB as it was sometimes troublesome due to a lot of the java syntax for mongo being deprecated. A maven project approach might have been a better idea as there are more resources available with regards to working with mongo using java.

The AWS virtual machine we used served its purpose as a free option but the initial set up, in terms of disc storage and RAM, was basic. We found out, throughout the development process, that we needed to restrict the amount of software installations on the VM. This was brought to our attention when the VMs performance had become extremely sluggish and sometimes unresponsive.

As a result we decided to discuss any future software installations unless it was vital to the project. The removal of the eclipse IDE was a necessary step once the login process was up and running. This helped free up some space and memory for other resources that needed to be constantly active and running. In future, the implementation of a dedicated server to run the databases might be a better option.

We originally intended to deploy the game on a mobile platform but unfortunately we did not have the time to achieve our goal. The games backend systems, match making and scoreboard system, were much larger tasks than originally thought so we decided to focus on developing for VR and desktop only.

6.3 Outcomes

The project goals, as laid out in the introduction, was a success to a point. The processes involved in allowing a player to login works very well. The database and program running this process has been running since the start of the project and has never given us any difficulty since being implemented.

The hosting and joining system also works very well. The data that is required when sending and receiving to and from the databases does so very efficiently. And the list of hosts is updated and shown on screen.

The game works quite well. The procedural track is built very quickly and there is a different one built each time. All the players spawn at a different places on the start line, which was a problem at the beginning. The racers are all able to move left and right and the acceleration button works. The problem here was that on a non gaming PC the player sometimes can not drive as the game is very taxing on some GPU's. The VR game also works well, both the VR and desktop players can see each other on the screen and can race against each other. We encountered a problem near the end though, the oculus was updated and the players hands were visible at a different position to the original placement.

Independently, the scoreboard system works very well. A list of player objects is serialized into XML and is sent to the scores database. The database is updated and the players global position is updated and the new list is sent back. Results and leader board are then put on screen.

Unfortunately, we encountered a problem with the race results. The player object gets destroyed as they cross the finish line. The issue was there was not enough time to figure out how to structure the order in which they cross the line and send that information to the scoreboard. This was a big disappointment to us as we feel as though we have the ingredients for a really fun, exciting and fully functioning game.

Chapter 7

Conclusion

With the project at the stage it is at this moment, it can be considered complete as there is no time remaining to continue the project. Many aspects of the project had worked as we had expected and other aspects of the project were harder or almost impossible, from the design stage to the implementation.

When we started the project, it was the biggest project that we have ever undertaken as programmers and because of this, we had no idea really what to expect from a group project of this type. We designed and implemented the project with the level of experience and understanding that we had at that time. If we were to start this project today, we would definitely approach the project differently.

One aspect we would change was the methodology, we used a waterfall method. This was taken due to the unpredictable lifestyle of a student, we didn't know exactly what issues we would face each week. At the start of the project, during semester one, we didn't know the timetable that we would have during the second semester. We would not know the volume of projects that we would undertake during the year or the level of complexity each project would entail. For these reasons, the waterfall methodology seemed like a good fit. However, on reflection the agile methodology would have been ideal for a system that was as heterogeneous at this project. It would have been ideal to break the project down to its smaller components, and program these parts individually as completely detached programs from the system as a whole.

A lot of portions of the game have been left unfinished, if we had designed a less heterogeneous system, we could have focused on one type of database and accompanying language, this would have allowed more time to finish the inner workings of the game. This is not something that we would have changed as we believe that learning to create these multiple systems that interconnect is more valuable than having a finished game. Finishing the game would at this point, only be a grouping of smaller fixes in bugs and implementation of small enhancements in the game.

An example of this would be the implementation of the scoreboard. The code for the scoreboard is written and it is fully tested. But in our rush to complete other aspects of the project we did not add a win condition to pass to the C# program, so we could update the scoreboard database at the end of each race and then display it to the users.

Also we were reluctant to change which technologies we would use. At the start of the project we felt a need to stick with the plan above all else. This led us going down some dead ends with the databases we used. Both CouchDb and Neo4J ate up more time than they ever should have. When we first hit issues with them we should have looked at other alternatives. This was a valuable lesson in coding. Not to try to force a square peg into a round hole.

Originally, we had wanted to package the game for mobile platform. Unfortunately, due to time constraints and problems in other parts of the project, we had to make the decision to lower the priority of the mobile version and focus on the rest of the project. This was a tough decision, but we have discussed looking into creating the mobile version in the future.

During the planning stage of the project we believed we had broken the project down to its smaller components, but if we were to start a similar project today, We would approach the project differently. We would separate the project into much smaller projects with timed tasks within that project. This would not only help with the time consuming tasks but also ensure that there are fewer bugs in the system. We also had to update the software and frameworks that we used during the project, in some cases simply to gain access to elements that we preferred. Such as the road pieces we used for the track, for this we updated Unity. This caused an issue where the objects that we used in the virtual reality section of the project were remapped by Unity. This was a mistake and was unnoticed for a time during the project. This was also partially due to the discomfort of trying to program a game while wearing a heavy head mounted display. The OLED display and lens didn't help this matter as long term use causes blurred vision.

The limitations of the virtual machine were also an issue during the project. The free tier version that we choose to use was fine at the beginning, but as we added more software and databases to it, we noticed it became more sluggish and error prone. If we were to undertake a similar project in the future, we would investigate the option of using a dedicated server for the purpose, as this would be more powerful and more efficient.

Overall we are reasonably satisfied with the project outcome. There are definitely some elements that would be approached differently. In the future an approach to creating a project like this would be to complete a fully functioning game first and fix any bugs and issues it might have. Then once the game was fully complete the focus would be on creating the back-end of the project.

Bibliography

[1] Using a virtual machine environment for developing, testing, and training for the um-ukca composition-climate model, using unified model version 10.9 and above. *Geoscientific Model Development*, 11(9):3647 – 3657, 2018.

[2] Mohammadreza Hazhirpasand Barkadehi, Mehrbaksh Nilashi, Othman Ibrahim, Ali Zakeri Fardi, and Sarminah Samad. Authentication systems: A literature review and classification. *Telematics and Informatics*, 35(5):1491 – 1511, 2018.

[3] Antonella Galizia, Gabriele Zereik, Luca Roverelli, Emanuele Danovaro, Andrea Clematis, and Daniele D'Agostino. Json-gui—a module for the dynamic generation of form-based web interfaces. *SoftwareX*, 9:28 – 34, 2019.

[4] Hassan Gomaa. *Software Modeling and Design : UML, Use Cases, Patterns, and Software Architectures.* Cambridge University Press, 2011.

[5] Wei-Meng Lee. *C 2008 Programmer's Reference.* Wrox Programmer's References. Wrox, 2009.

[6] Rick Miller. *C for Artists : The Art, Philosophy, and Science of Object-oriented Programming.*, volume Second edition. Pulp Free Press, 2015.

[7] Jongseong Yoon, Doowon Jeong, Chul-hoon Kang, and Sangjin Lee. Forensic investigation framework for the document store nosql dbms: Mongodb as a case study. *Digital Investigation*, 17:53 – 65, 2016.