50.003 Game Project Report

Group 5: Aditya Manikashetti, Jonathan Bei, Navnidh Bhalla & Zhou Xuexuan

Storyline:

The game is based in a virtual world called 'Get High', in which Princess Bunny, along with other players, has been trapped. The players initially collaborate to escape the virtual world, but as they get closer to doing so, the game becomes evil and morphs the virtual world into three distinct and formidable levels to keep the players trapped. The new world has two crucial rules: Firstly, only one player can escape the world – survival of the fittest. Secondly, death is not an option, meaning that if you fail to escape, you will be trapped for all eternity. Princess Bunny and the others realise that its every person for himself/herself and begin to turn hostile towards one another. Instead of collaborating, they are forced to compete to clear all the levels, jumping over obstacles and trying to avoid treacherous pits. Only the best will escape the 'Get High' world and re-enter reality. Who will it be?

Game Concept:

Initially, we prototyped a 2D vertical platformer multiplayer game, where players jumped on platforms and used power ups in a race to the finish. However, after user testing the game, we realised that the game was too easy, unrealistic and people were getting bored quickly. As such, we started to brainstorm ideas on how to improve, and eventually decided to refine the game into a 3D version to engage and challenge the user more effectively. Additionally, we introduced 3 unique and exciting maps of varying difficulty, a pixel-like design to the entire game, in-game sound effects and dynamic features such as moving obstacles and teleportation. These changes were well received during our second round of user testing, reaffirming that our efforts had indeed made the game more immersive and enjoyable. Here are screenshots taken from our final game:

Main Menu

Figure 1: Our minimalist main menu screen allows players to easily navigate to the game lobby or view the developer team.

Level Selection

Figure 2: Players can opt to play in levels of varying difficulty, with level 1 being the easiest and level 3 being the hardest.

Multiplayer Lobby

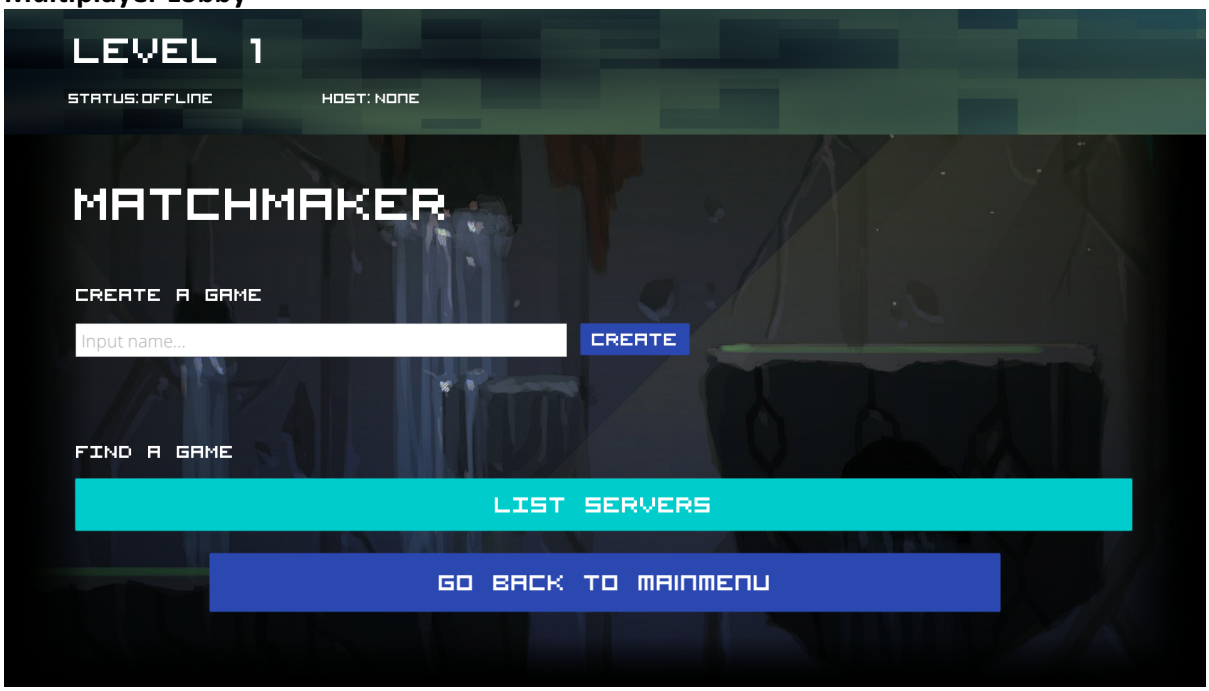

Figure 3: The lobby allows players to either host or join a game. It also shows their online status and the level selected.

Level 1 Gameplay

Figure 4: The first level is set in a medieval market and is meant to help players get used to the game controls and features.

Level 2 Gameplay

Figure 5: The second level is set in an enchanted palace with dynamic platforms and a demanding path to reach the top.

Level 3 Gameplay

Figure 6: The final and most unforgiving level, with its deep pits and eerie vibes, puts the players' skills to the absolute test.

End-Game

Figure 7: Upon completing a level, a win/loss screen is displayed along with the option to quit and start another session.

System Requirements:

User Requirements

- *Multiplayer-Enabled:* Allows multiple users to connect and play together in real time
- *Multi-Sensory:* Allows players to enjoy the in-game experience visually and audibly.
- *User-Friendly:* Allows players to conveniently navigate through the game application
- *Engaging:* Allows players to feel immersed and entertained as they play the game.
- *Progressiveness:* Allows players to constantly challenge themselves and improve.

Functional Requirements

- *Fast Network:* Allows users to play in real time without experiencing any lag/delay.
- *Viewable Area:* Game camera should follow the character's movements accurately.
- *Realistic:* Game physics of characters needs to be in tune with player expectations.
- *Assets:* For a stimulating user interface, assets need to follow a theme and be clear.
- *Testing:* The game should not have major bugs/errors which harm user experience.
- *Security:* The game should not open doors to new threats/leak user data.
- *Modularity:* Allows for easier testing, debugging and implementation.

Non-functional Requirements

- *Wide Audience:* The game should appeal to as many people as possible.
- *Economical:* Game development should be low cost

To better illustrate the various system requirements, here are the use cases of 4 major components of our game: The main menu, the game lobby, the in-game component and the game over component:

MAIN MENU USE CASE

ID	**Get High Main Menu**
Name	In Game Menu Screen
Objective	To display different gameplay options and information regarding the developer team
Pre-conditions	1. Game has been successfully installed on the players' device 2. No bugs/issues exist when the game is launched

Post-conditions	Success:
	The following scenarios occur for each option:
	• Network Game
	  ○ Player is taken to the game lobby, where he/she can host a new game or join an existing one
	• Staff Info
	  ○ Player is taken to a screen outlining the names of the development team
	Failure:
	• The above scenarios do not occur for any one of the options

Actors	• Primary 1. Player • Secondary

	1. Unity Game Engine
Trigger	GameLaunched() state activated
Normal flow	1. Game successfully launches 2. In game sound starts playing 3. Player is brought to the main menu screen 4. Player chooses one of the above post-condition options 5. Unity Game Engine handles player input accordingly
Alternative flow	Scenario 1 (Game is unsuccessfully installed) 1. Player is unable to install the Android Package Kit (APK) 2. Player retries to install the APK 3. If successful, the normal flow continues 4. If unsuccessful after repeated events, player reports issue to the developers 5. Developers provide support/fixes to rectify the issues Scenario 2 (Game crashes upon launch): 1. Game quits unexpectedly upon launch 2. Error message identifying issue is shown on screen 3. If issue persists, player reports it to the developers 4. Developers provide support/fixes to rectify the issues
Interacts with	Lobby, Staff Info
Open issues	N.A.

GAME LOBBY USE CASE

ID	**Get High Game Lobby**
Name	Get High Game Lobby
Objective	To create an online multiplayer game lobby for players to host/join games
Pre-conditions	1. Player is connected to the Internet 2. Game server is online 3. Player has chosen a level to play 4. More than one player is online wanting to play the same level

Post-conditions	• Success
	  ○ Host has created a game lobby accepting players
	  ○ Player can join a game lobby if he/she is not hosting
	• Failure
	  ○ Player's internet connectivity drops/is too weak
	  ○ Game server has crashed

Actors	• Primary
	  1. Player (includes host and non-hosts)
	• Secondary
	  1. Unity Network (UNET)

	2. Unity Game Engine
Trigger	1. Player has selected a level he/she wants to play 2. Player has found a lobby with other players
Normal flow	1. Host has created a joinable game lobby 2. Other players can find the lobby and join in 3. All players click the 'Ready' button 4. Game commences
Alternative flow	1. Existing lobbies are full 2. Player cannot find an empty lobby 3. Player either hosts his own game, or waits for a lobby to open 4. Normal flow continues
Interacts with	Various Game Levels, UNET
Open issues	Latency caused by the host player influencing the whole lobby status

IN GAME USE CASE

ID	**Get High Main Game**
Name	Main Game Mechanics
Objective	Update game movement mechanics and animations across the various players and allow them to play smoothly in real time
Pre-conditions	1. Players successfully joined a game lobby 2. Chosen map/characters have been loaded 3. Game lobby has transitioned to the chosen level successfully

Post-conditions	• Success
	  ○ Players have a strong network connection and are able to play the game in real time
	  ○ Maps/characters have been successfully rendered
	• Failure
	  ○ Players are experiencing lag or connection drops
	  ○ Maps/characters have been unsuccessfully rendered

Actors	• Primary
	  1. Host
	  2. Other players
	• Secondary
	  1. Unity Network (UNET)
	  2. Unity Game Engine

Trigger	Player uses joystick to move around and the jump button to jump over obstacles. Map interacts with player actions accordingly
Normal flow	1. Players have been placed into a properly rendered game environment and have strong network connection. 2. The game commences and player actions/movements are initiated

	and registered into the Unity Game Engine 3. UNET servers handle the user inputs from the engine and sync them across the players 4. Players are able to see their own and other players' actions in real time.
Alternative flow	Scenario 1 (Game Disconnected): 1. A player drops their network connection 2. If it's the host, the game server is shut down and players are returned to the game lobby 3. If it's another player, he/she is disconnected while the rest proceed with the game Scenario 2 (Game Assets Improperly Rendered): 1. Players are placed in a map with improperly rendered assets. 2. Players can choose to restart the game or continue with the faulty assets for the existing game session
Interacts with	Lobby, Maps, Characters, Game Over, UNET
Open issues	N.A.

GAME OVER USE CASE

ID	**Get High Game Over**
Name	Game Over Screen
Objective	Launch game over screen and display results (You Win/You Lose) along with the option to quit and restart
Pre-conditions	Game successfully completed with host not losing connection midway

Post-conditions	• Success
	  ○ Players are shown the game over screen
	• Failure
	  ○ Players are returned to the game lobby due to connection error

Actors	• Primary
	  1. Player (includes host and non-hosts)
	• Secondary
	  1. Unity Network (UNET)
	  2. Unity Game Engine

Trigger	GameEnd() state reached.
Normal flow	1. Game successfully ends 2. Unity Game Engine changes screen to the game over screen 3. Display the win/loss status along with the option to quit and restart

Alternative flow	1. Host drops their network connection 2. Game server is shut down and players are returned to the game lobby
Interacts with	Lobby, Maps, Characters, UNET
Open issues	N.A.

System Design:

Overall Architecture

To design the entire game system, we adopted the iterative and incremental software development methodology, with every iteration time-boxed at two weeks. We gathered user input after every iteration, and incorporated it into our subsequent releases, helping us to continually add functionality and improve our game throughout the development process.

We used Unity (version 5) as the core development platform for our game. This is because Unity presents numerous benefits, such as having very well documented learning materials online which we could refer to as we developed the game, having an extensive asset store and being very user friendly with its intuitive interface and helpful 'Scene View', to name a few. We created our own assets, own scripts and own tests throughout the game development process. We also utilised some of Unity's technologies, particularly UNET which was an ideal framework to facilitate online server-client communications.

Asset Development

Our two primary assets were Princess Bunny, our in-game character, and the 3 maps, which acted as the different gameplay levels. Since our game was in 3D, creating all the assets seemed like a daunting task, given that we only had a few weeks to develop the entire game and needed to work on the game itself and the online networking components. Hence, we decided to use open source prefabs in some instances. However, we still had to ensure that these prefabs fit in with our game storyline and game setting, resulting in us customising these assets accordingly. Here are more details regarding character and map development:

CHARACTER DEVELOPMENT

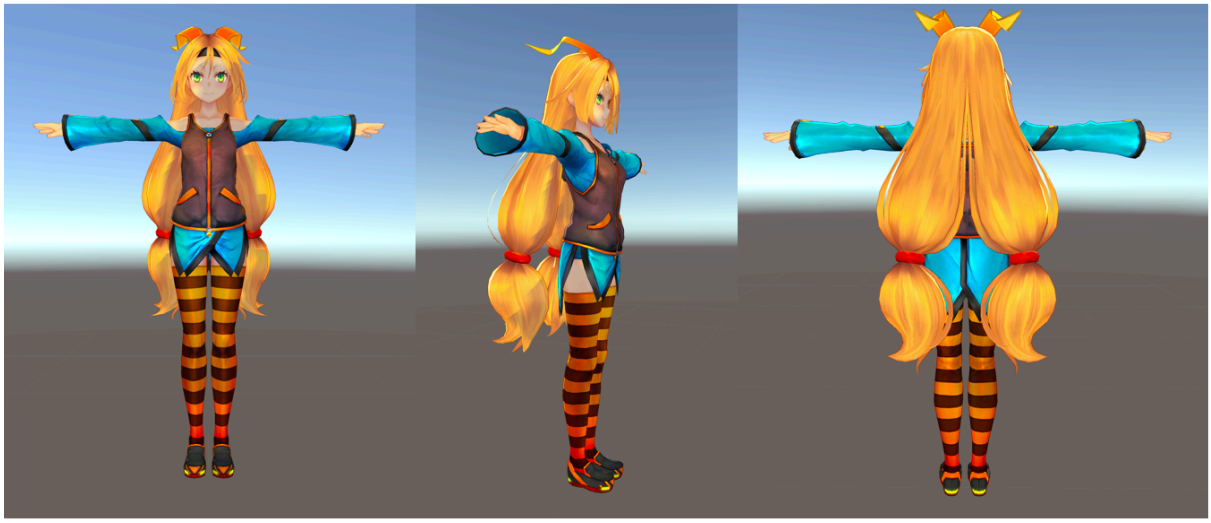

Figure 8: 3D Model of our main character, Princess Bunny, from a 360-degree camera angle to show her various aspects.

As seen from Figure 8, Princess Bunny was inspired by manga art. The bright orange, yellow and blue colours of her outfit helped in allowing the character to stand out against our maps. To make Princess Bunny's actions as realistic as possible, we payed close attention to the game physics. Our 'ThirdPersonCharacter' script manages character physics by defining attributes such as speed and actions like moving, jumping, falling and crouching. The user inputs for these actions are handled via our 'ThirdPersonUserControl' script which, with the help of our 'CrossPlatformInputManager' script, translates the user's input from multiple platforms (PC/Mobile/Tablet) into a tangible character action. We also payed close attention to character animations. To do so, we used Unity's Animator and an animator state machine. The state machine determines which animation to deploy based off the characters' actions, movements and location. For instance, if the character is airborne, it will trigger the in-air animation and if the character is on the ground, it will trigger the on-ground animation. Hence, with the character physics and animations being in sync, we were able to create a realistic and smooth gameplay for the character.

MAP DEVELOPMENT

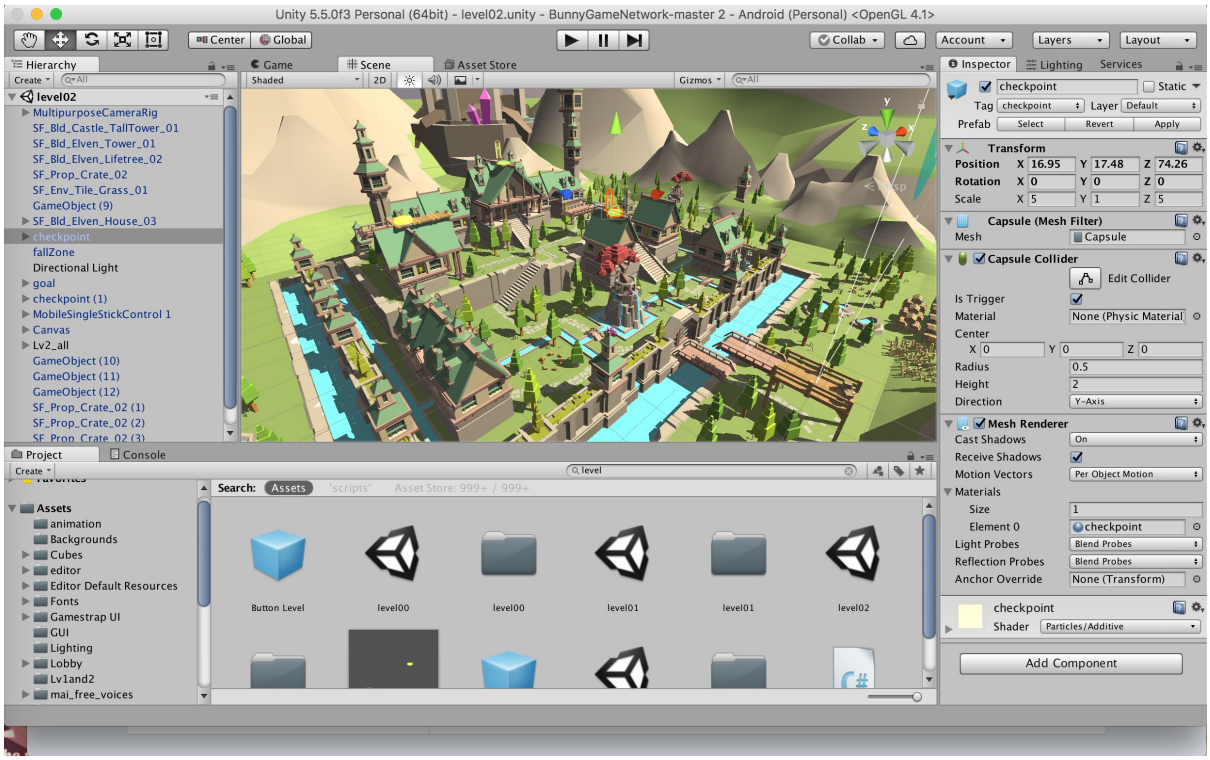

Figure 9: A screenshot depicting the many assets used in developing our Level 2 Map using the 'Scene View' feature of Unity.

To aesthetically enhance the map prefabs that we had obtained, we used mesh renderers for in-game objects such as walls, boxes and obstacles. We also used instructional text meshes throughout the maps to guide the player towards the level endpoint. In addition, we implemented particle effects using Unity's inbuilt Particle System, to create some dynamic visual cues for the player to follow. Directional lighting was also used to illuminate the maps, creating the illusion of natural light and shadows which helped to make the game more realistic. With the visuals out of the way, we then focused on the map physics. Some of the physics elements of our map include moving platforms, friction between the character and the map object and various colliders which allow for the character to interact with the map (checkpoints, physical boundaries to prevent the player from walking through objects/walls, 'death zones' to trigger a player respawn once he/she has fallen into a treacherous pit, et

cetera). The 'MovingPlatform' script defines the attributes of the dynamic parts of the maps, particularly the platform itself. "Position1" and "Position2" scripts identify the start and end points for the platforms and they are repeatedly run to allow the platforms to oscillate between their start and end positions throughout the game session. A Lerp function was also used to smoothen the movement of the platforms, since they were initially jerky and unnatural. Our 'CharacterHolder' script ensures that a character using the dynamic platform moves with the same velocity as the platform itself so he/she does not fall off. The aesthetics, physics and improved animations ensured that our maps were exciting, fun and lifelike.

Concurrency Network Implementation

As mentioned before, we used UNET as a framework to facilitate online server-client communications. We chose UNET over other possible frameworks such as Photon Unity due to the following:

- UNET is a high-performance transport layer based on UDP to support all game types
- Its Low-Level API (LLAPI) provides complete control through a socket like interface
- Its High-Level API (HLAPI) provides a simple and secure client/server network model
- Its Matchmaker Service provides the basic functionality for hosting/joining games
- Its Relay Server solves connectivity problems for players trying to connect with each other behind firewalls

As such, UNET was the preferred choice. To complement UNET, we also imported and integrated the 'LobbyManager' library (an extension of the 'NetworkLobbyManager' class of the Unity High-Level API) into our game due to the following:

- The library limits the number of players joining a game
- It prevents players from joining a game that is in-progress
- It allows players to re-join the lobby when the game is finished
- It presents a simple user interface for interacting with the lobby
- It has a per-player ready state, so that the game starts only if all players are ready

Here is how the entire network infrastructure works for our game:

1. Upon entering the game lobby, players can choose to either host or join game rooms. This action is handled by the Matchmaker Service.

2. The host takes on two roles – that of a server (since we have no dedicated server) and that of a local client. The 'LobbyManager' script will call SetupServer() and the SetupLocalClient() method and create the game room.

3. Players joining a game room only act as remote network clients. The 'LobbyManager' script causes Lobby Player Game Objects to be created on the server for these players.

4. Each player is assigned a 'NetworkIdentity' which allows the server to differentiate the various clients and send their status across the network.

5. The 'NetworkTransform' module synchronises all players' movements across the shared map, allowing for smooth and real time updates of characters across the players' devices.

6. Similarly, the 'NetworkAnimator' synchronises player animations across the network.

7. Players can now enjoy a real time multiplayer game over WiFi.

To summarise, here is a diagram overviewing the UNET architecture used in our game:

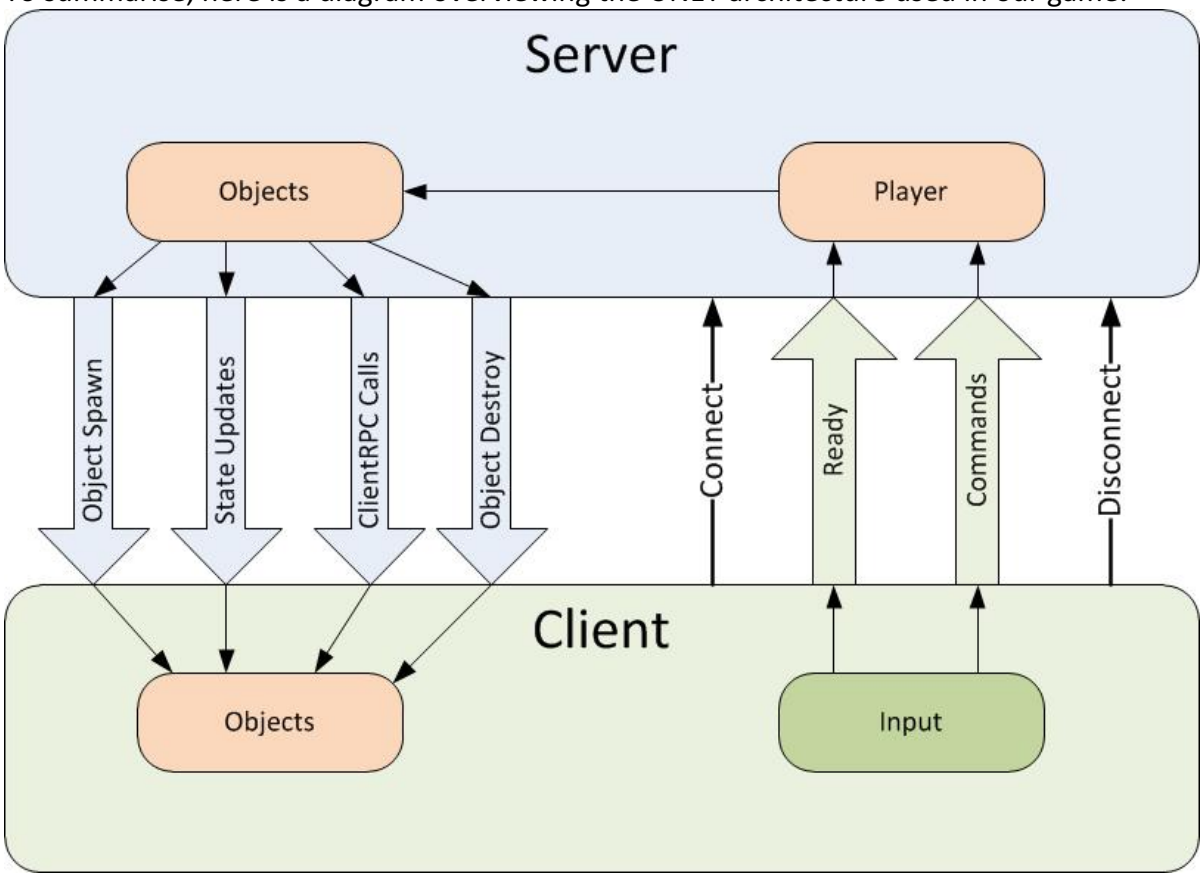

Figure 10: Diagram depicting how UNET manages client-server communications amongst various players in a game session.

System Testing

Throughout the game development process, we used several techniques to test our game. On the user side, we did usability testing to evaluate the user friendliness of the game as well as beta testing to gather feedback and make improvements. On the developer side, since we had various iterations of our game, we did regression testing to ensure that the changes we made to our game did not damage existing modules or cause unexpected results. In addition, after every iteration we manually tested each of the modules (the Unity Game Engine has limited options for effective unit testing), before conducting integration testing to ensure everything works collectively. Whilst performing these various tests, we focused on several key components of our game:

1. *Game Physics*

- Gravity
    - We had to test the gravity and ensure that it suitable for the player. Too much gravity would make it too difficult, and too little gravity would make it too easy. Hence, we needed to test for the optimal value that would make jumping among obstacles doable, but at the same time challenging.
- Friction

    - Through testing, we discovered that the friction of the character model was buggy. When another player collided into it, the character would just slide off the plane in a direction not specified by the user. This was rectified through repeated tweaking of friction values.
- Colliders
    - In the testing, we discovered a glitch regarding colliders. Characters who passed through narrow colliders just froze midway. We solved this by testing the suitable distance value between colliders which would not cause the glitch to occur, and implemented it amongst the various assets in our map.

2. Movement

- Turning
    - Since we used the joystick interface in our game to control the movement of characters, we needed to ensure responsiveness was at an intuitive level. Too high joystick sensitivity would cause the player to turn very quickly, while too low sensitivity would result in very slow turning. We tested these values until we found the optimal sensitivity that was intuitive for the user.
- Speed
    - The extent to which the joystick was moved from its centre resting position needed to be proportionate to the speed of the character. The further it is, the higher the speed. As such, we needed to calibrate the minimum and maximum speed of the character.
    - We also had to ensure that the velocity is increasing linearly to its maximum when the player is moving only in one direction
    - We also had to ensure that the velocity is decreasing linearly to its minimum when the player is moving in the opposite direction
- Jumping
    - We ensured that the jumping action was based on real physics, so that when the player was airborne, he/she followed projectile motion.

3. Camera

- Sensitivity
    - The camera turn sensitivity had to be adjusted to give players a balance of responsiveness as well as view ability. Setting the sensitivity too high would be disorienting for the player and setting it too low would cause the game to appear as if it is taking place in slow motion. We tested these values until we found the optimal sensitivity that allowed for real time movement.
- Following the character
    - A camera had to be associated with each character so that the player could see the screen from his/her perspective. To do so, we implemented a network behaviour which allowed the camera to identify the character it is following and stick with that character throughout the game.
- Angle
    - The camera angle had to be set properly as well. A low angle would mean that the player is seeing the map from the ground level, while a high angle would mean that the player seeing the map from a bird's eye view. We needed to

calibrate the angle such that both the character and what lies ahead in the map was visible to the player.

- Object Interaction
    - The camera also needed to avoid crashing with walls and seeing through them. To ensure this, we wrote scripts which implemented a Multipurpose Rig to identify the Mesh components. Our script also adjusted the zoom of the camera based on the character position with respect to game objects.

4. Network

- Latency
    - To ensure that we have minimal lag time, we tested latency by using different values of data send rate.
    - We also tested by performance testing the system by allowing multiple clients to connect to the server. We found that restricting the game lobby to two players minimised latency issues and allowed for smooth networking.
- Animation Syncing
    - Whilst playing the game, we had to ensure that the Unity Network Animator was sending animation transform.

5. Thread Safety

- Unity only runs the game on a single thread (the main thread). As such, thread safety is ensured by default.