Introduction:

Princess Bunny has been trapped in a virtual ‘Get High’ world along with other players. They initially try to collaborate to escape, and as they get closer to leaving the virtual world, the game becomes evil and changes the rules of the world. The virtual world morphs itself into three distinct levels and implements the following rules:

*No 1 - “Survival of the Fittest”.*

*No 2 - Death isn’t an option.*

Princess Bunny and the other characters realise that it’s each person for themselves in this virtual world and turn hostile towards each other. They compete to clear all the levels, jumping over obstacles and trying to avoid treacherous pits. If they lose against a character, or make mistakes, they will be indefinitely stuck in that level. It’s a race to who conquers the entire map first.

Game Concept:

The game is meant to be a 3D platformer, where 2 players compete to reach the end of the level first. The levels comprise of both puzzle-solving elements as well as platforming challenges, clearing checkpoints along the way. Players will have unlimited lives, thus should they fall to their deaths, they will simply respawn to the last checkpoint. The game only ends when one of the two players reaches the end point.

Game Elements:

**Main Menu**

**Level Selection**

**Lobby**

**Gameplay**

**End-Game**

System Requirements:

**User Requirements**

* Ability for multiple users to control separate characters in real time
* Ability for in-game camera to follow players (3rd person perspective)

**Functional Requirements**

**Non-functional Requirements**

Use Cases & Use Case Diagrams

Design Process:

We started off with the idea of having a 2D platformer, where the players compete to be the first to be at the top. However, after an early prototype of the 2D platformer, we realized that we could do more with the Unity game engine and decided to attempt a 3D platformer instead. A large part of a game’s appeals is the visuals and we decided to craft a visually appealing world in which the user can immerse him/herself in.

Game/System Design:

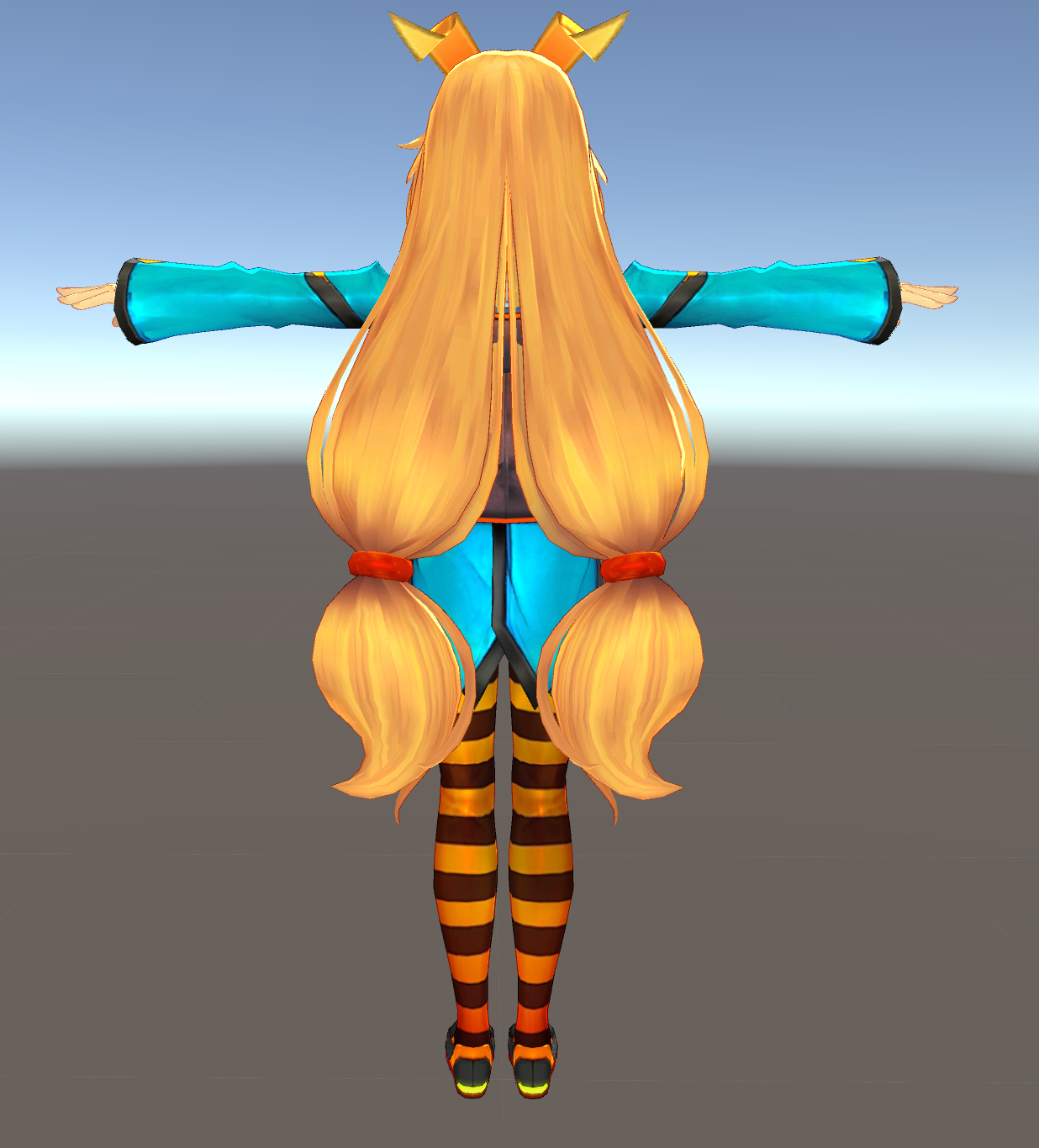
**Overall Architecture**

The application is developed with Unity (version 5), with their Mac OS development software. The server and online communications is implemented using UNET, Unity’s own networking multiplayer technology.

**Character Model Development**

After much discussion and research with the team, we decided to utilize open source character assets. This decision was made after careful consideration of time taken to create a character model on top of developing the game mechanics as well as online networking components.

Character Model:



**Character Physics**

Character physics are handled by “ThirdPersonCharacter” script, which defines the speed of movement of the character as well as the animations when it is moving, falling, jumping as well as crouching. An implementation of the Unity’s Animator is used to animate the character model as it moves about.

**Maps**

The in-game maps were implemented through the use of open sourced pre-fabs, which are in game models that can be tweaked to user specification. We utilized mesh renderers for the aesthetics of the in game models for walls, boxes, houses and other platforms for users. Box colliders were used to implement that physics aspect of the models, preventing the player model from walking through them and setting physical boundaries for the player.

Player instructions are also placed around the maps, to guide the player in the right direction. To do this, we used text meshes, with hints written for the player to follow.

We also implemented particle effects using Unity’s inbuilt Particle System, to create some dynamic visual cues for the player to follow.

Moving platforms are also an aspect of the map that we implemented, through the use of our own script, level 2 has moving platforms on which the player can jump onto and progress the level. A Lerp function was used in order to smooth the movement of the platforms, as in our initial testing simply moving the platform created stutters in the movement and using the Lerp function dramatically improved the smoothness of the platform movements.

Directional Lighting was used in order to light the maps in the game, simulating the illusion of real-time light that is not only natural but realistic. Soft shadows are used for the game, to improve game aesthetics.

**Movement Mechanics**

In-game character movement mechanics are handled via the “ThirdPersonUserControl” script, where a user’s input is handled via the “CrossPlatformInputManager” script, which supports inputs from multiple platforms such as mobile or PC, and translates the user’s input into physical movement of the character model in game.

**Concurrency Implementation**

In a nutshell, UNET provides developers with the following capabilities:

* High performance transport layer based on UDP to support all game types
* Low Level API (LLAPI) provides complete control through a socket like interface
* High Level API (HLAPI) provides simple and secure client/server network model
* Matchmaker Service provides basic functionality for creating rooms and helping players find others to play with
* Relay Server solves connectivity problems for players trying to connect to each other behind firewalls

Network Modules utilized:

**NetworkIdentity**

**NetworkTransform**

**NetworkAnimator**

For our game, we have utilized Unity’s high-level API to implement a client/server network model. One user hosts a game on his phone (the host) and the other user (the client) connects to the host user. Each player is assigned a NetworkIdentity as he is instantiated to differentiate the two players and send his status across the network. To synchronize the player’s movements across the shared map, we used NetworkTransform to send the movements across the server so that players movement will be reflected on each other’s screens. To allow each player to see the running animation of the character, we also implemented a NetworkAnimator to synchronize the animation of the player across the network as he/she runs about.

We also made use of Unity’s Matchmaker Service to allow users to create rooms and join previously created rooms to play in. This required us to import a LobbyManager library Unity’s Asset Store and utilize their open sourced lobby system and integrate it into our system. Our network architecture is Wifi-based, where two users need to connect to the same wifi connection and one player acts as the server whilst the other player acts as a client and connects to the server. Unity’s networking service implements UDP sockets to synchronize data across the network, to capitalize on the speed and performance gains it provides over TCP.

Unit Testing:

We adopted a modular approach to testing. There were several key components of the game that required testing. Due to the limitations of unit testing of the engine, we decided to go with manual user testing of the individual components.

Components that required testing:

1. Game Physics

* Gravity
  + Gravity had to be tweaked to an optimal level to allow the player to have a reasonable jump distance, to give the player an acceptable degree of control as well as challenge too.
* Friction
  + In our testing we discovered that the friction of the character model was buggy, as when another player collided into it, the character would just slide off the plane in an undesired direction not input by the user. This was rectified through repeated tweaking of friction values.
* Colldiers
  + In the testing, we have to ensure colliders are applied properly so that the players will not stuck between the colliders

2. Movement

* Turning
* Speed
  + Make sure speed is controlled by a cross-platform input joystick
  + Increment the velocity linearly to maximum if player is always targeting one direction
  + Decrement the velocity linearly to minimum if player is always targeting the opposite direction
* Jumping
  + Jumping has to be based on the real physics which applied an “in\_air” state on a player to make sure player cannot pivot themselves on landing reflecting a player is indeed in projectile motion.

3. Camera

* Structure
  + Multipurpose Rig (Applied With Scripts)
    - Pivot
      * Main Camera
* Sensitivity
  + The camera turn sensitivity had to be adjusted to give players a balance of responsiveness as well as viewability. Setting the sensitivity too high would be disorienting for the player and setting it too low would not give the player
* Following the character
  + Camera has to set a network behaviour to target a local player. Basically, the camera itself would identify its own local player and target on the player forever.
* Angle
* Avoiding Wall Collider
  + Scripts implemented Multipurpose Rig are supposed to identify the Mesh components behind the character and push the camera to zoom more based on the character position
* Rotation Speed
  + Rotation on camera should be smoothen which will allow a better ingame view experience

4. Network

* Latency
* Send/Receive Rate
* Animation Syncing
  + Double check Unity Network Animator is sending animation transform