2D Platformer in Unity

Idea

The Idea of the game came when we decided to make a platformer game inspired by others like Celeste, as such we begun by making a normal 2D platformer where the character moves in a X, Y axis while keeping the Z axis locked. Key mechanics of this platformer are a Jump(normal jump, held jumps and short jumps) a Dash that moves the character to the direction inputted automatically in a quick motion that refreshes once you touch the ground (or an item) and Wall climbing/ Jumping. We added some personal touches after these basic mechanics, such as hard-coding certain timings and opportunities to retain air momentum through tactful usage of Dashes, as well as a dynamic camera following the player at an offset. The Goal of the game is simply to "beat the stage" by reaching the end platform of the stage, by moving to it, avoiding the obstacles and dangerous terrain in the way.

Story

The very basic Story we have for this version of the game, is that our player character is in a futuristic Virtual Reality Parkour Training Course to train for any upcoming challenges they might face, so they must traverse the simulated levels and reach the end in order to complete their training. Due to the fact that this game is done in Virtual Reality, we tried to make the Backgrounds as well as all of the Game Objects/Animations etc. to be more Virtual inspired, sort of like the matrix.

Our Process

Movement

We begun by following a small guide that followed a prototype version of our movement and Jumping scripts, as such we tested out the movements on a basic 3D Cube.

We wanted to allow for a wide range of control when it came to inputting jumps, and as such, we created two separate jump scripts that work in tandem in order to achieve desired jump arcs based on how long the player holds down the jump input. Any sort of basic jumping is managed in the movement script, but ontop of that, another script called BetterJumping was implemented, in which it reads how long a user holds the jump button. With quick taps of the jump button, the player can perform a "Short hop", and with long presses of the jump button, a player can increase the height of the jump just a tiny bit, which we've dubbed as a "Long jump"

After working on the prototype versions and adjusting them, we had a basic version of the movement system working.

Collision Detection

Before implementing a Rigidbody that would act as the playermodel, we knew before hand that we were going to need more than just a box collider in order to determine game logic. Not necessarily due to any sort of limitation, but rather to add detection of some events regarding

the rigidbody before the rigidbody would literally collide with terrain. Sort of like a buffer zone, we wanted to have additional hitboxes coming from both sides of the character, as well as bellow the character, in order to allow for more responsive controls.

As such, we created a Collision Script for the player to detect Walls and Ground as well as dangerous terrain later on. After which we found a model from mixamo.com and used it in our final product as well for our player character. Following that are the animations. We used an initial Animation Controller by Kevin Iglesias and tweaked it and added/adjusted it to fit our needs, by changing the connections, or removing unneeded animations such as Moving Backwards, Sideways etc. Following that we made the animations move according to the movement script, such as moving Right triggering the Forward animation, as well as rotating the character to face the direction the input was, for example if the character was facing Left and wanted to move right, the Script would turn around and Lock in on the direction the player wanted as well as triggering the forward animation. Other animations of note are Jump, Falling, Landing as well as the added animation of Wall Grab, which are all triggered accordingly from the Scripts. Details on the animation trees are included bellow as well.

Advanced mechanics

After the basic and crucial aspects of the game were covered, we moved on to more unneeded but unique mechanics to the movement, adjusting the movement script and adding several instances into the code, making it so that after certain actions momentum can be carried over to increase the overall speed of the character, seeing as this is a platformer game, where time is of the essence. We added the momentum carrying by adding an extra variable to the existing speed of the character, called airspeed, initialized the same as speed, however under some circumstances can become as high as 5 times more than speed, in cases where it is over the base speed, we made it gradually lose speed per game tick, simulating Deceleration. The instances where the momentum can be carried are all related to the Dash mechanic, such as if the dash is directed at the ground and collides with it on its animation before finishing, then a portion of the dash speed is converted into airspeed, however to keep a smooth gameplay after a small time period of landing on the ground, airspeed is reset to match speed, so if dashing onto the ground, there is a small time window to input a jump in order to retain a lot of momentum. Due to this change in the script, we also decided to scale the Dash according to the airspeed, as well as in circumstances where the dash ends on the ground again, to reset the dash Boolean as well as the jump for more freedom and speed.

Stage development and philosophy

After the movement was all but done, we felt confident in starting to develop our levels each on their own, since we had been collaborating on the workings of the code up until now, we had a good grasp on their functionalities, so we each begun making our own level and shared any extra Scripts or Game Objects we made, Scripts such as detecting Collision from dangerous objects like spikes, or scripts for functions like checkpoints etc. and Game Objects such as the UI such as the Time display or the Death Count display, as well as the general workings of the pause menu. After this we had a look at the stages and all of their mechanics and combined their settings and Script settings to work in similar ways to not differentiate the levels too much.

The two levels we created approached level design in ways that would accentuate the two sides of the game. For example, Stage 1, developed by Alexandros, has more of a focus on tight platforming segments. It sets you in a claustrophobic level, where your goal is incredibly clear, with tricky obstacles that require tight timing to overcome. This puts emphasis on the tight and snappy controls of the character. In contrast, Stage two, developed my Markos, is much bigger. Not necessarily as dense, but gives more movement freedom to the player. With this stage, the player is allowed to travers the stage using the advanced system mechanics more comfortably, whilst also showcasing how the advanced system mechanics can reward players who have mastered them by granting them access to routes that are otherwise not possible without HyperDashes.

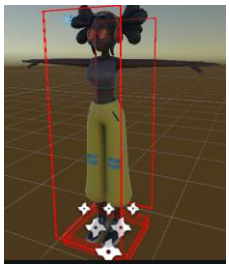
Additionally, each stage tries to implement their own type of “gimmick”. Instead of power ups in the traditional sense, each stage has a unique element that the player must use and/or navigate. For example, in Stage 1, there are these rotation cogs that the player must ride in order to progress in the level. And in Stage 2, there exist green bounce pads that the player can use to bounce high, with additional height being gained when the jump button is held.

From there we simply just ironed out some bugs or flaws that we may have had in logic. And once all the levels and scenes were made, we compiled them all together.

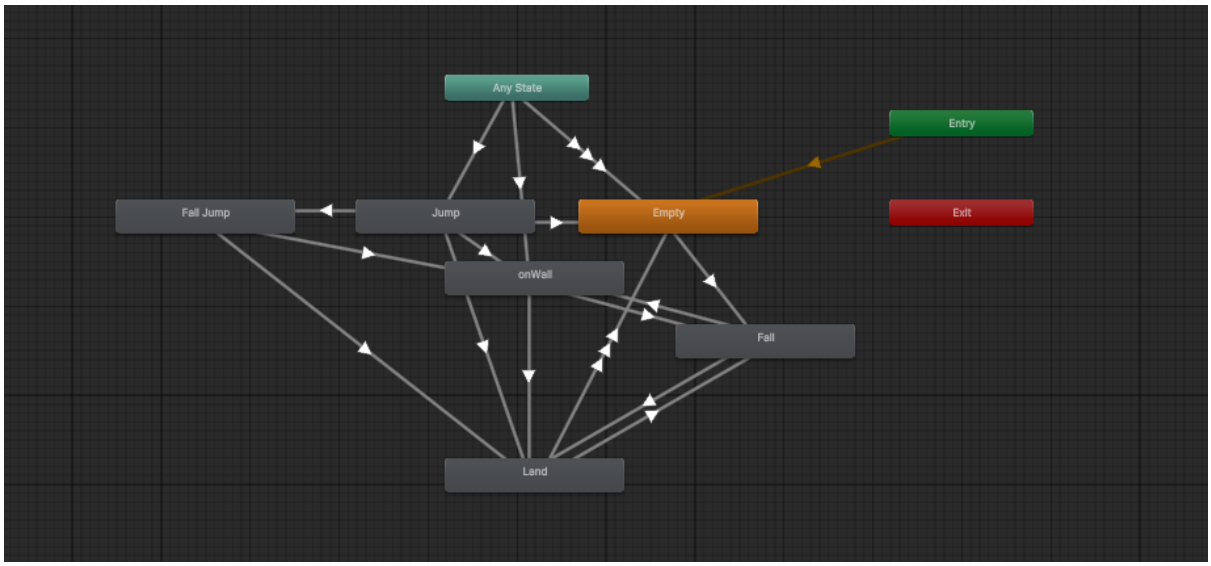
Implementation of requests

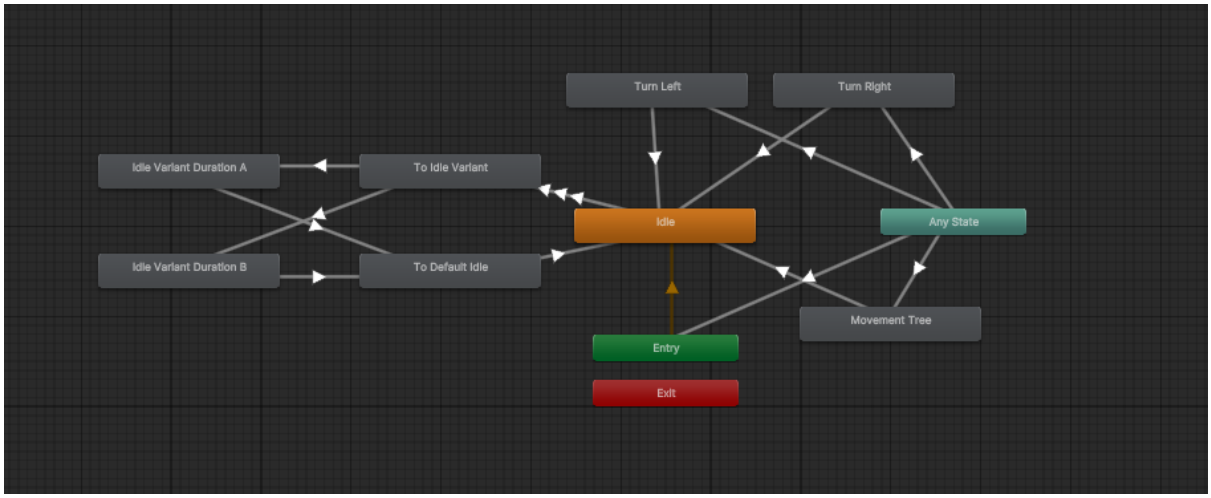
Χρήση μοντέλων:

Usage of Player Character being a Character Model imported from Mixamo



Χρήση Animation





We were able to implement the most basic of animations (i.e running, jumping, landing) through ready made assets that we found. However, in order to get them to work properly with our game, we had to develop two layers of animation trees in order to have the states be traversed based on the events occurring in both the movement script and the collision script.

A lot of checks are implemented in the Update routines of Movement that check for specific combinations of Booleans so that no animation gets skipped

One of the best examples of this tedious state checking is with the following:
Whilst the falling animation would obviously play after the player would input a jump, we had to take into account other ways that a player could enter a falling state without needing to jump. A couple of cases come to mind, primarily if a player would run off of a ledge. In this case, we need to make sure that the player was not dashing at that current tick of the game, and that they were not grounded, nor had they attempted to wall jump. This is one example of vigorous animation state testing.

Χειρισμός camera και μετακίνηση αυτής αναλόγως του παιχνιδιού:

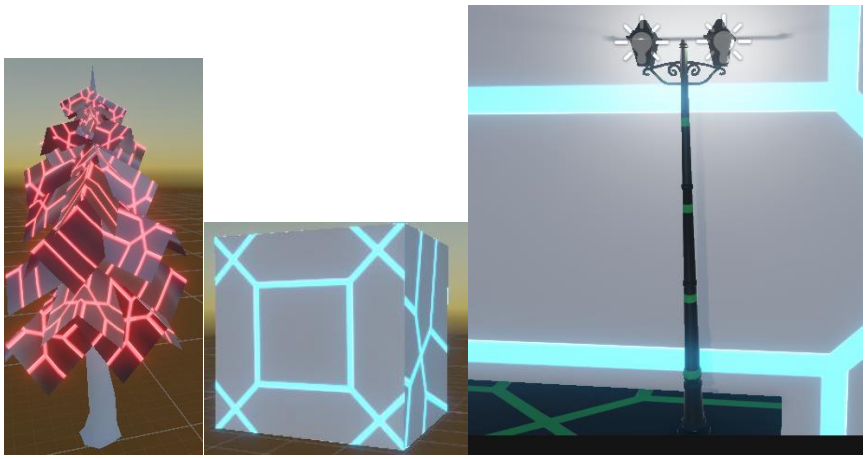
Because of the platformer element, the camera's sole purpose is to follow the player with the use of a Script following the Player character at a slight offset to view more of the stage, meaning the Camera is almost always in motion. On Character Death and Respawn, the Camera will Readjust to the new Respawn location.

Κατάλληλος φωτισμός:

Usage of Unity's Built in Light source making it have a slight Rotation, as well as using several Lamp posts with their Light Source being their Lamp, lastly the usage of Shaders with built in Light sources.

Χρήση υλικών στα αντικείμενά σας (Materials – Textures):

As well as Shaders:



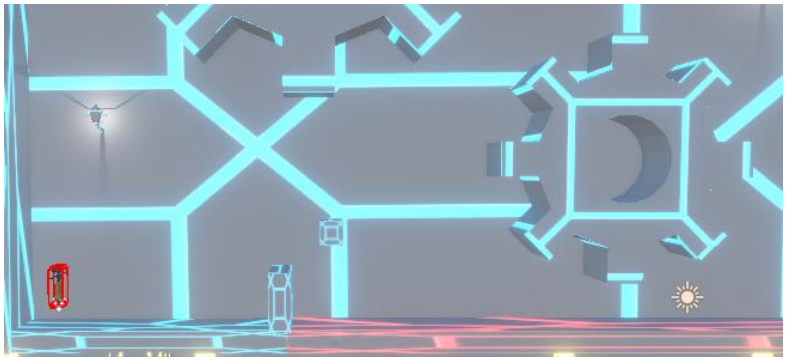
Usage of multiple Materials and Textures that include built in Shaders for some slight Glow, as well as Textures with Patters, simulating a Virtual environment with Virtual Trees, Lights and Cubes etc. The Effect was made possible by using a Shader that made the whole Object Glow a Bright Color, We used that Shader giving it a PNG of a shape we wanted, and then made the same Object Copied, slightly Smaller and made it a Solid Material, making the Transparent one on top to give it that stylistic glow.

Input από το keyboard ή/και από το mouse:

All of the Game's movement is done through the WASD Keys, where A and D are for movement, the Spacebar that simulates a Jump, as well as the Shift and CTRL keys, the Shift key is for Grabbing Walls and scaling them, and the CTRL key is the Dash button + any direction from WASD to trigger a dash in that direction. Lastly ESC is also used to Pause/Resume the game. These controls are also applicable to a Controller.

Navigation:

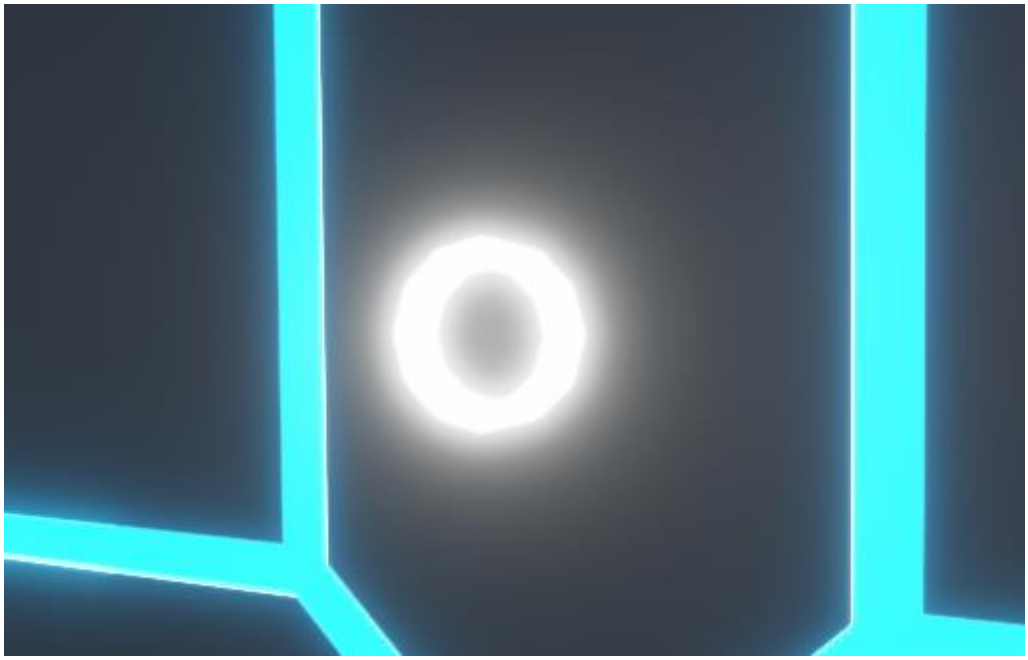
Navigation through the Stages by moving about and clearing the obstacles in the way, as this is a platformer game



There is also Navigation through the menus, where there is a main menu to select from 2 different Stages, and through the stages having the option to return back to the main menu by pausing.

“Power ups”:

We included one literal powerup, but in most cases, we preferred implementing variety through unique stage design. The only literal powerup in the game is the Dash Ring, that replenishes your dash if you’ve already used it mid air.

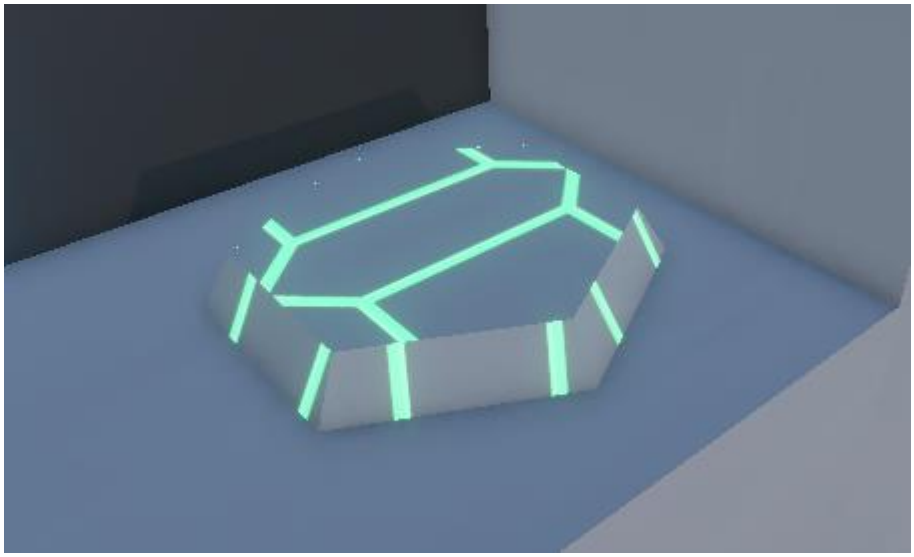


Stage Gimmicks:

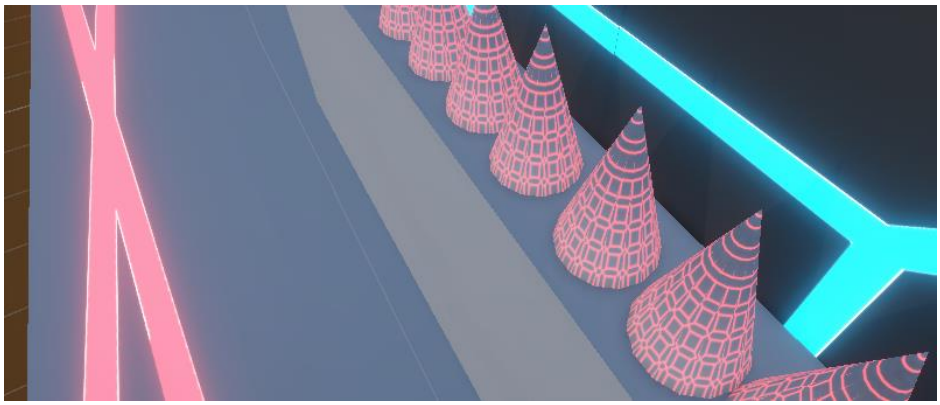
We included the following for the two stages we created

A) Moving platforms

B) Bounce pads



C) Hazardous terrain



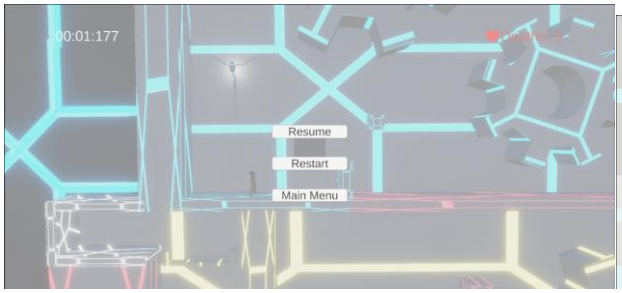
D) Moving cogs

Physics:

The use of Rigid Body on the Player Character as well as complicated Movement and momentum carriage, as well as Collision physics and Jump physics. There are multiple Scenarios from encountering other objects in the scene, firstly it checks their layer to see whether they are groundlayer or danger layer and act accordingly. For example the player character can climb a groundlayer wall but will die when it comes in contact with a danger object. The Objects apart from the Player Character do not strictly follow physics, such as rotating in place for the Gears, or moving up and down or left and right for a lot of Cubes and Platforms. The Player character is affected by gravity and their own movement, as well as any movement enhancing objects on the Scene (such as a Jump pad increasing Jump Height)

UI (HUD + pause menu):

The Game has a Menu System when launching to pick from Stage1, Stage2 or Quitting the game as part of the initial HUD. The game also features a pause menu from any time in the level. Upon Triggering the Pause, the UI HUD will appear with the features of Resume, Restart and Main Menu while time is frozen by scaling it to 0, meaning the Timer does not run and no physics are taking place. The Resume Button simply Un-pauses the game, the Restart Button returns the Player to their Spawn point and the Main Menu switches the Scene to the Main Menu Scene.



Audio:

The game uses Audio in plenty of ways and very frequently, there is a looping Background Song for each Scene for instance, as well as some sound effects for specific actions, such as Dashing, Jumping, Landing and Dying, Beating a level, as well as Pausing and Resuming the game.

Particles:

Particles are used sparingly in the game however they are there, for example Dashing Creates Orange smoke Particles following the player character until the animation is over. Dying has an Explosion of hollow cyan Triangle Particles in tandem with turning the character Cyan with a Shader, for a Simulation-like effect