CS 175 Computer Graphics
Platform Game Writeup

Team members: Janet Liu, Yaxin Lei

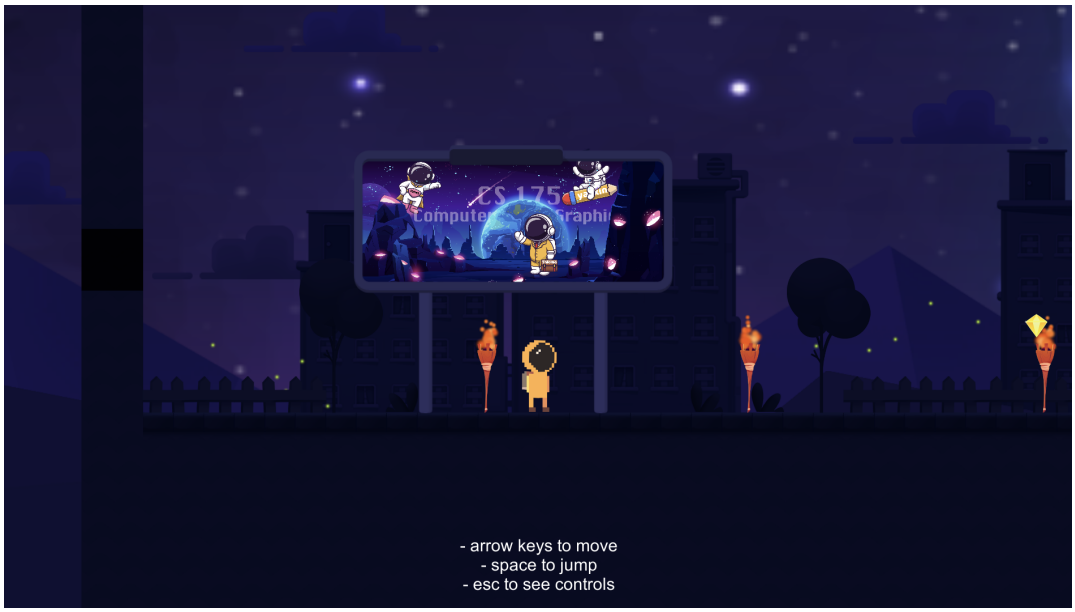
Introduction

Our CS175 final project built off of the basic 2D Platformer Microgame from the Unity tutorials. Neither team member had ever used Unity before, so we were starting from the very basics. After running through some tutorials to learn the ropes of Unity, we added our own mods to the game. These mods involved customizing the player sprite and movement, visual layering of the background, particle systems for fire and firefly simulation, aesthetics and more.

Our Github repo is: https://github.com/YAXINLEI/janet_yaxin_175final.git

Technical Details

Both team members used the most recent version of Unity available for Mac, 2020.3.7f1. The game sprite movement controlled by the arrow keys, and pressing space allows you to jump. The goal of the game is to gather the yellow beads, jump on top of enemies to kill them, and make it to the end of the platformer level!

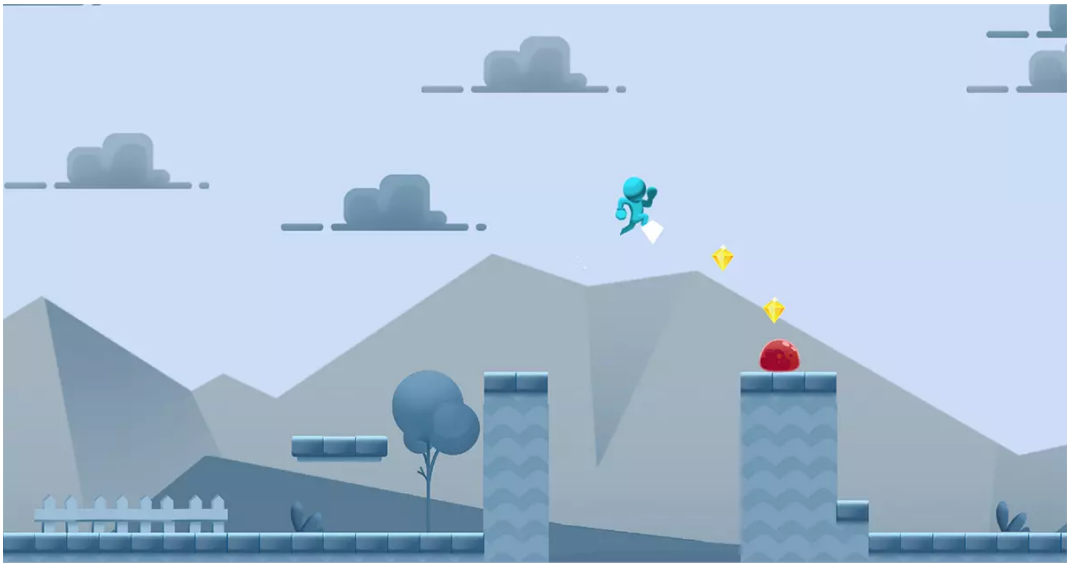


Technical Challenges

We had a really tough time collaborating on this project together, and using Git/Github with Unity. We followed several different tutorials about using Git with Unity (including the default Unity .gitignore file, using LFS, etc.). However, our assets and packages would often break or erase past work (this was also reflected by many user reviews for the Unity Particle Pack etc) and our scene would not load, regardless of the various changes we tried to follow online (reimporting the assets, redownloading packages, etc.). Because we had so many issues with this, we ended up just using Google Drive to upload compressed Unity project files. This meant we would work separately (Janet would make some updates to the code and upload her file, and Yaxin would download the project before making her updates).

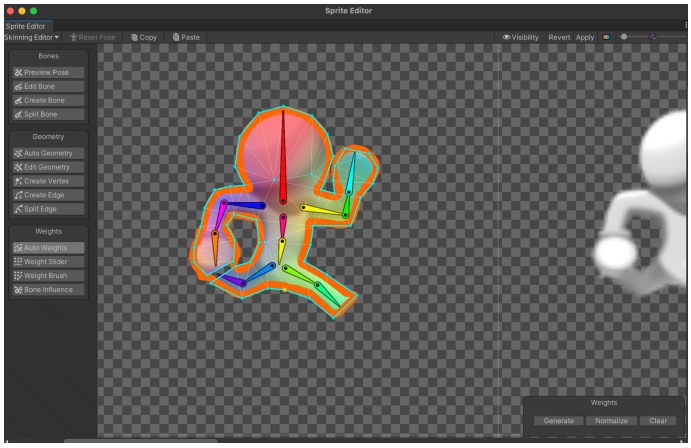
Customizing the Player

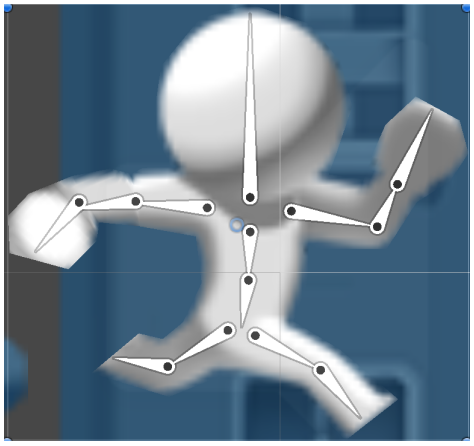
The original design of the game had a blue sprite player with a blue mountainous/block like background. There was a unique animation for running, jumping, landing, respawning, death, hurt, and victory. These animations were created by literal drawing assets that would draw, frame-by-frame, the various moves that went into each action.



Updating Sprite Animation

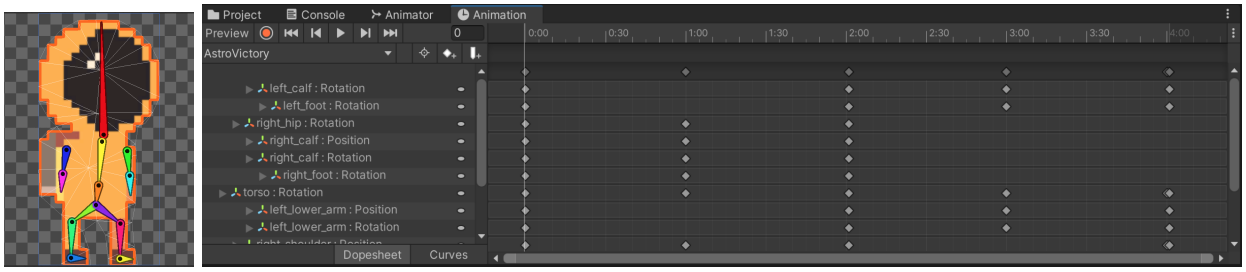
At first, we wanted to change the sprite animation a little to make it run faster. To do so, we used this Unity resource to learn how to rig the sprite. Rigging the sprite is very similar to what we learned about in class, where we created bones that attach to particular parts of the sprite character. To do so, we downloaded the 2D Animation Package from Unity, before opening up the sprite editor (on the left). In the sprite editor, we added in the 13 bones visible to the png file. After the bones, we auto-created geometry, before manually changing the geometry to finely fit the sprite character's body shape. This meant splitting existing edges, creating edges/vertices, etc. These changes are reflected in the scene, on the right hand side, where you can see the bones superlayed over the actual sprite skin. Because we were able to manually change our geometry, we could separate the left-side fist from the left-side leg, even though those were attached in the original png file (shown in the red box).



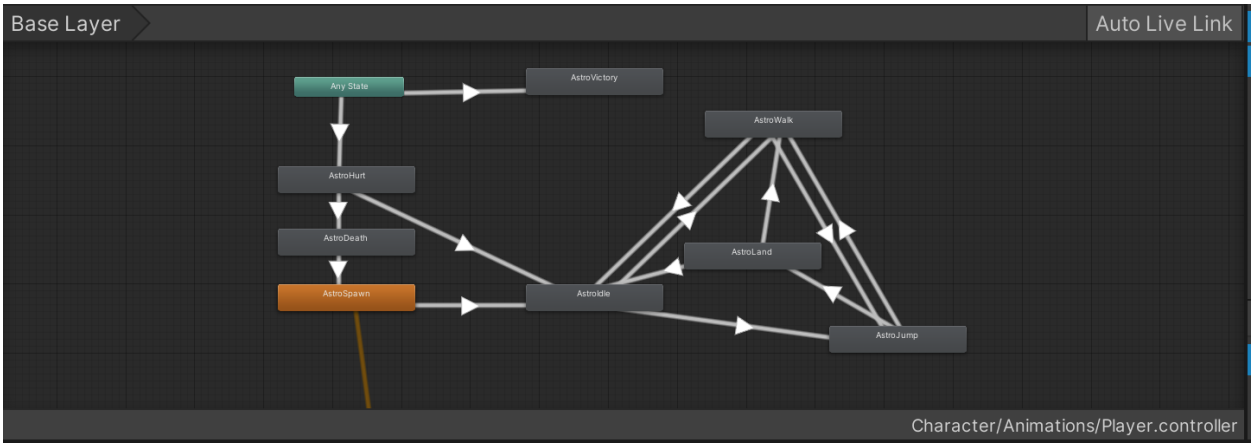


Changing the Sprite

Eventually, we decided on a space theme for our platform game, and decided to change the sprite. We surfed the Unity store to find some free 2D assets, eventually landing on this Pixel Space pack. This Astro pack came with a walk and jump animation, but we still had to create all the other animations that the original sprite had (land, spawn, idle, etc.). Again, we rigged the astronaut character. Then, we followed various animation tutorials to record different animation clips in Unity. This meant recording various key frames at half-second or second intervals. These key frames were created by adjusting the bones in our rigged sprite. Then, Unity animation would interpolate between these key frames to create a smooth animation. Using this technique, we created an idle, victory, land, and death animation.



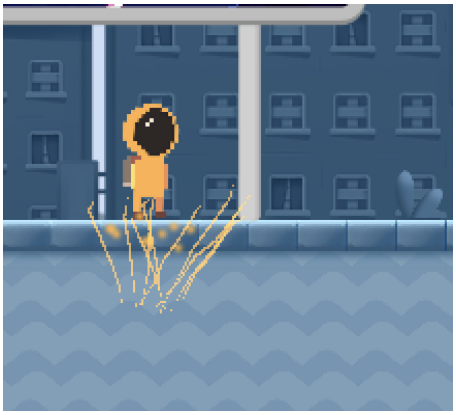
Once we had the various animations created, then we modified the player animation controller (picture below) that kept track of the various code states of our sprite. This controller also dealt with transitions, and linked the correct state to the correct animations. Lastly, our player had to have a RigidBody 2D and Box Collider attached to it. To understand how to use these components and modify them for our new sprite, we followed this documentation and tutorial.



One thing that we wish we had more time -- and the artistic skill to do -- was create our own assets. Some of the animation tutorials we saw had distinct png images for each of the skin components. For example, there would be a separate image for the right shoulder, left shoulder, head, chest, etc. Then, the rigging involved also putting the image components together to form a cohesive figure. Then, in animation, it created a very cohesive look because you could move the individual image components without transforming other image components. However, the assets that we found all were one cohesive png file. This meant that when we animated our sprite, moving the leg bone would deform the foot/lower body as

Unity tried to extrapolate how the animation should look. This meant our animations looked a lot less clean, and our sprite would get slightly deformed in the process (the death animation is a prime example of that).

Player Bouncing effect

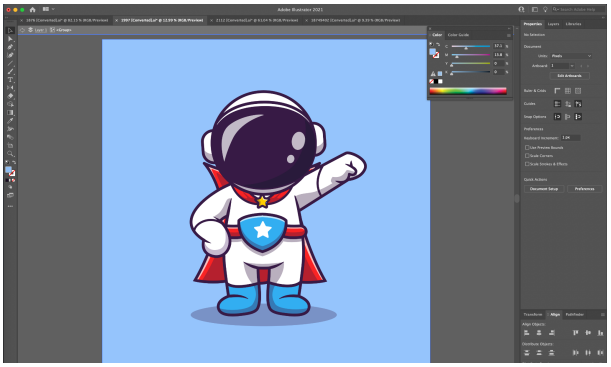


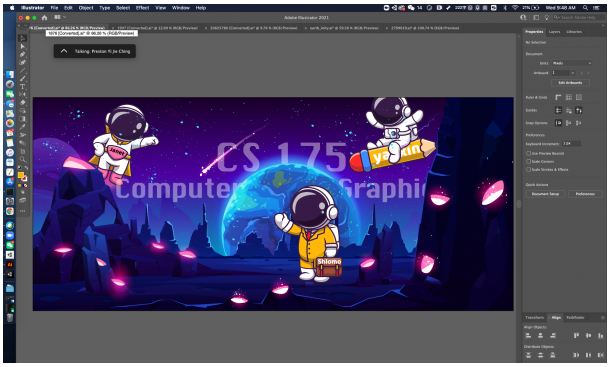
We wanted to add something to the player so it's more visually pleasing, so we added a bouncing effect where each time the player lands, it would have two overlaid effects simultaneously. This was done by overlaying a **lightning effect** asset (the orange lines starting from beneath the player's feet) and **fireworks particle systems** (the orange dots).

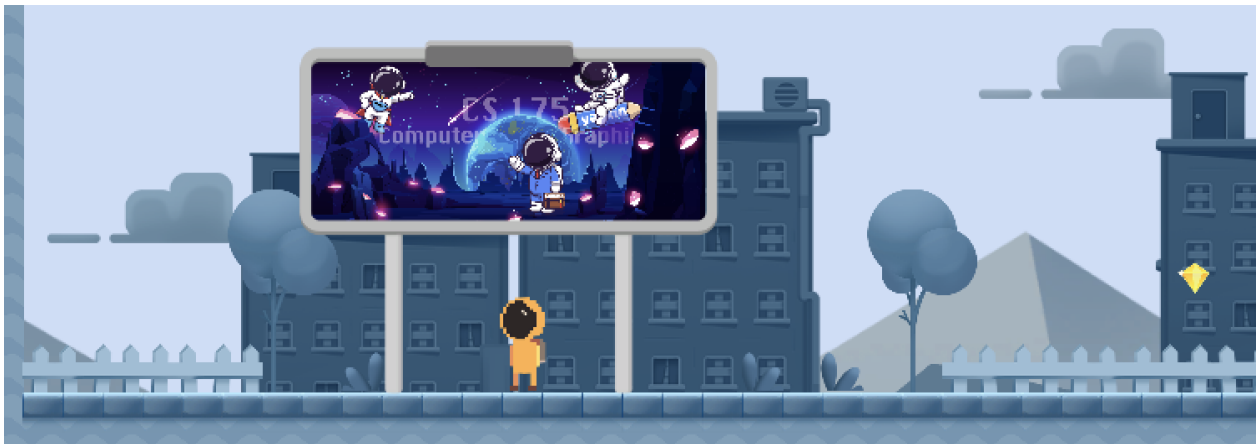
Customizing the World

Custom Billboard

The first thing we did for customizing the world was adding a billboard with customized figures for Professor Shlomo and team members Yaxin and Janet. These figures were created in Adobe Illustrator and uploaded into our project as new assets. We used a prebuilt billboard GameObject to actually add the photo to our game.



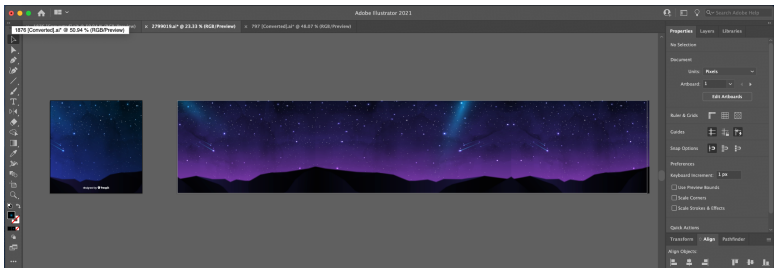




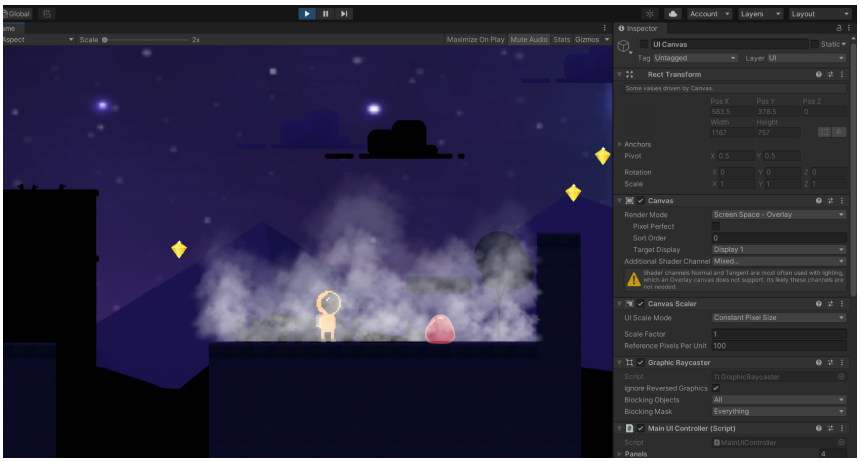
Aesthetics: Customize our world

To further customize our world, we then completed the following customizations which made the game have an overall blue-purplish space theme: **1. Color modifications, 2. Customized space background, 3. Adding Enemies, and 4. Adding fog.**

Apart from tweaking around asset colors of all existing buildings, levels, background. We added a customized background created in Adobe Illustrator:

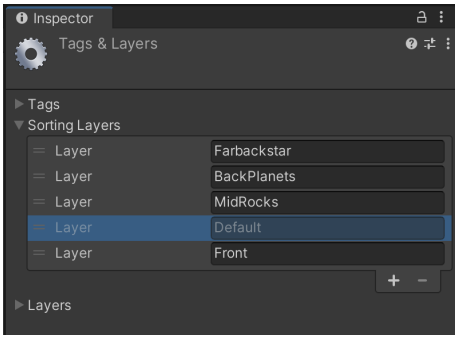


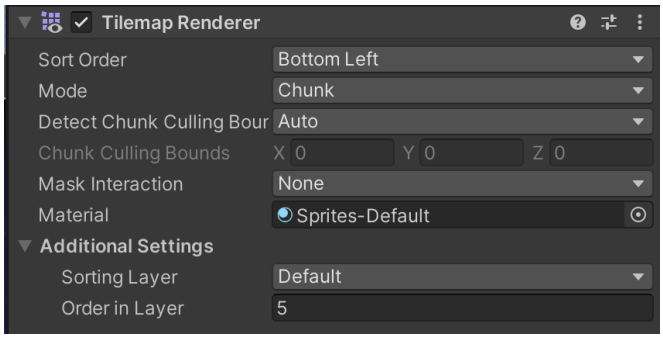
We also customized some enemies, as well as added some fog assets to make the game more challenging and aesthetically more pleasing. After doing the above efforts, our game now looks like the below:



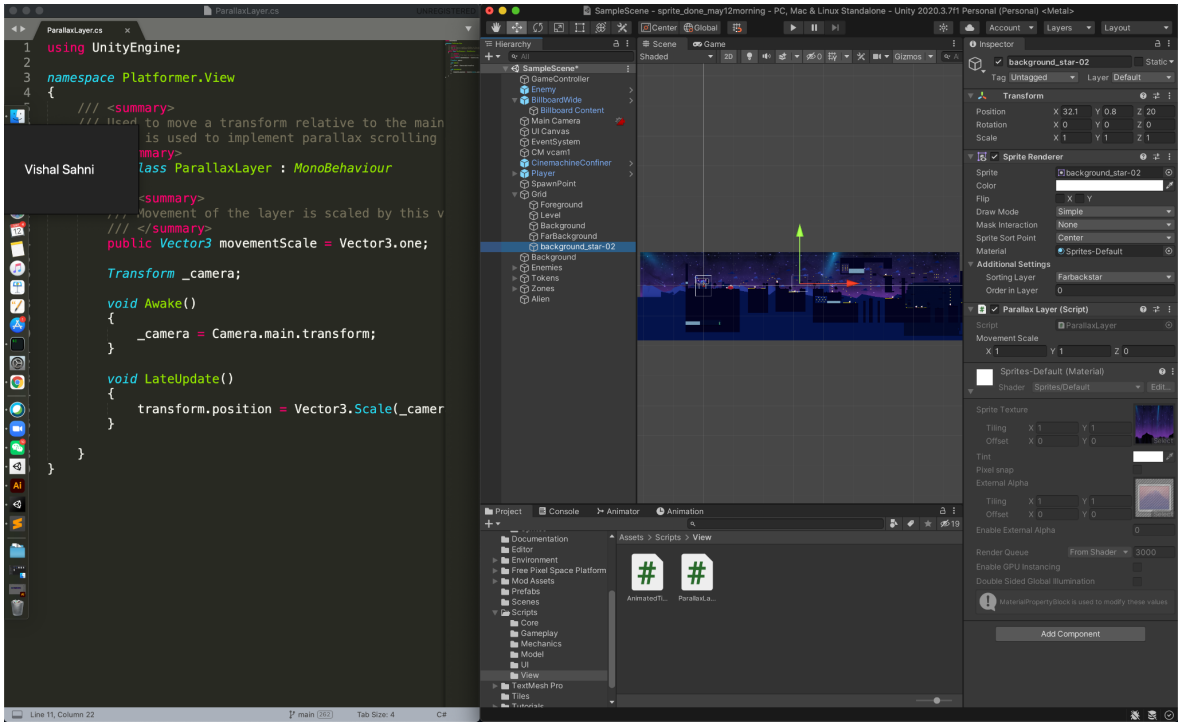
Creating depth: Multiple layers moving at different speed

Now the world has colors, but looks a bit flat. We want to create depth by adding different layers of background assets. Recall in class where we discussed how different objects were rendered such that the objects closer to the eye (camera) will always appear before the objects further to the eye (camera). Therefore, the first thing we did was to create a list of sorting layers to help the rendering process.
As shown below, we have five layers, from close to far, they are Front (for floating rocks and fog asset), Default (where the interactive game takes place), MidRocks (mountains), BackPlanets (floating plants), Farbackstar (our furthest background of the star sky). Within each layer, different assets can further be sorted using sorting by layer (below right screenshot).



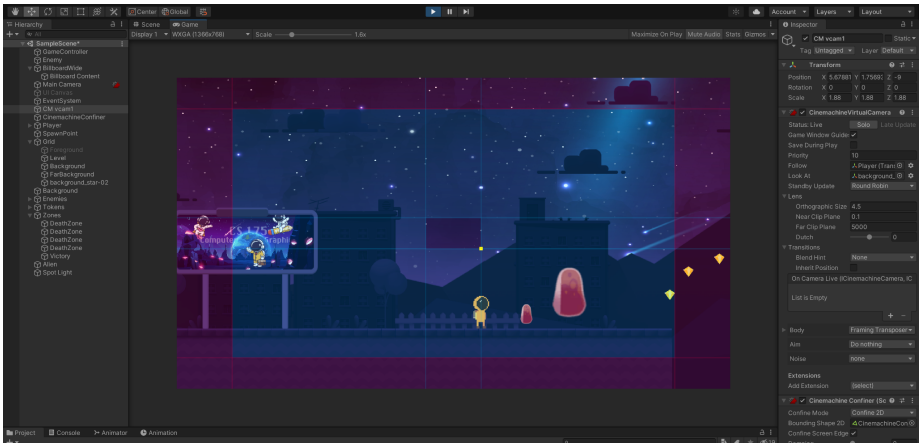


Even with the new layers added, the graphics still look flat, like a bunch of 2D images stacked together. To create depth, we've used the below script to scale the movement of different layers. As the player moves from left to right, to the camera view, the furthest assets and objects move extremely slowly (almost as if not moving), and the closest assets and objects move very fast. This creates an illusion as if different layers have different depth although in 2D.



Virtual Camera

We used a customized virtual camera to overwrite the main camera. There are three reasons: we want the virtual camera to do object tracking on the player so the player is always in the middle while always looking at the furthest background with stars. This will allow both player and background to always be centered, and we wouldn't go out of boundary with the background.



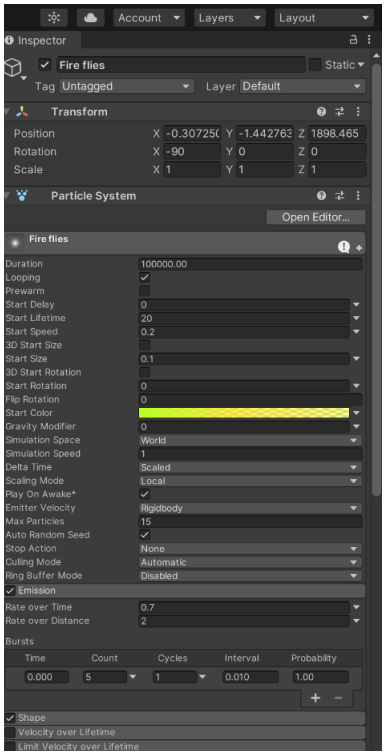
Particle System Simulations

The part where we discussed particle system simulations in lecture has always fascinated us. By editing color, emission, speed, texture, direction, lifetime, emission shape etc of a particle system, we could achieve so many cool effects from easy to extremely dynamic and realistic. We wanted to spice up things by adding some two types of particle systems in our game world.

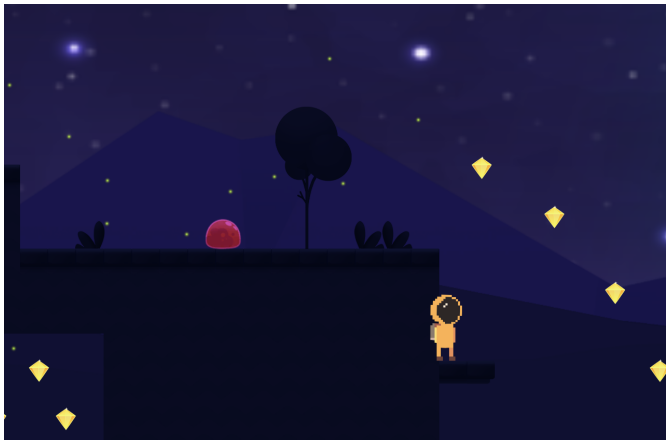
Particle System - fireflies:

Since the overall world was a night scene, we decided to add some fireflies using Unity default particle systems with manual setup. Compared to fire in the next section, the firefly is more straightforward, as fireflies maintain their particle shapes and colors almost static during the whole process. Like mentioned in class, we could edit the color, emission, speed, direction, lifetime, emission shape etc to generate the visual effects we want.

Specifically, we used a hemisphere shape and randomized movement directions to make fireflies diffuse into different directions randomly. Max particle was set to 15-20 depending on the specific position with an initial particle spurt. We used a bright green - orange - transparent yellow gradient to mimic the firefly's color. In general, we wanted to mimic the appearance and movement of the fireflies as much as possible. The below graphs shows part of the setup process.

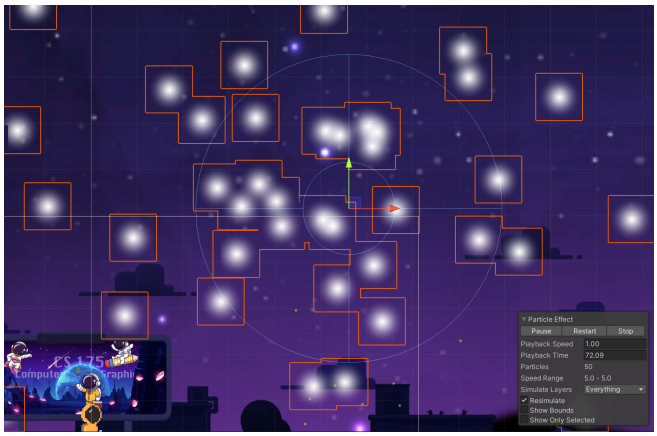


As shown below, together with the plants, the fireflies in the night scene generate a quiet and peaceful sensation. Multiple sets of firefly particle systems were placed at multiple locations in the game.



Particle System - Torch Fire:

With the success of adding fireflies, we decided to test the more challenging fire particle system simulation, which involves more manual setup and parameter tweaking to make it look realistic. When loading a default particle system, the below is what we see。

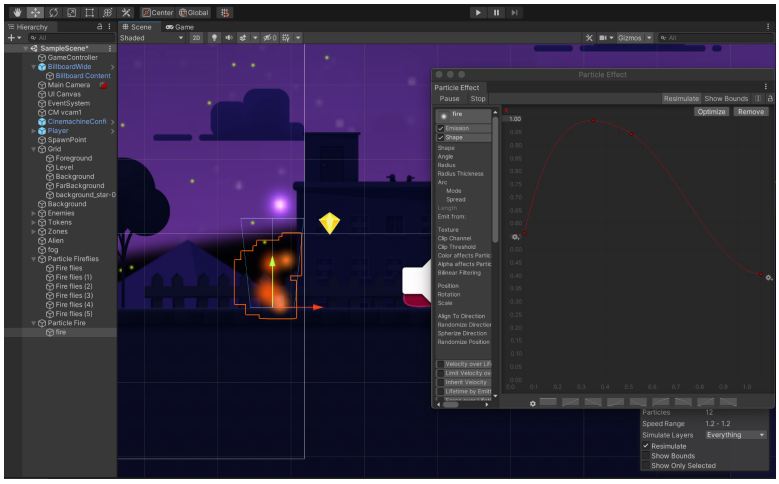


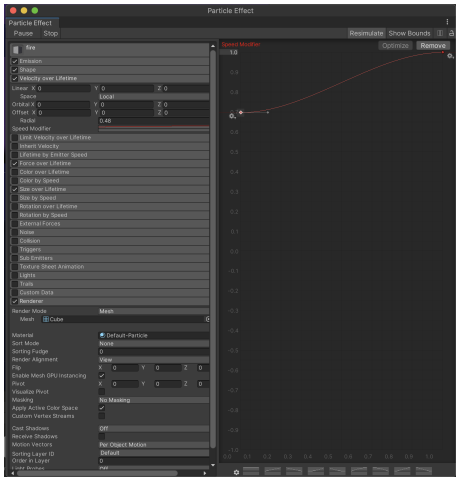
Note that there are a few important features of fire, each being mimicked with a different feature of real fire.

- Initial speed, initial size, emission: Normal fires looks much more dense than the default particle system. Therefore, we tweaked the lifetime, start speed, and emission to make the rough shape look good and particles more densely packed.
- Shape: We set our shape to be a cone with angle of 20, which gave the up-side-down cones shape of normal torch fires.
- Velocity/force/size over particle lifetime: To make the fire more realistic, we want particles to start at mid size at bottom, then get bigger and shrink to nothing at the tip of the fire. We also want the force field to push the particles more upward approaching to the end of the particle's lifetime. As a particle moves from bottom up, we also want the velocity to be increase as particles reach the tip of the fire.

- Shape-texture, renderer: After completing the above steps, our fire would still look like a white blob. Thus we included a customized texture that looks like the below also created using Adobe Illustrator. This gave the fires the right hue and variation in color.

The below shows some graphs of fire parameter tweaking and setup process. In general, this was not a one-pass task, we tweaked all parameters multiple times to achieve the most realistic results.





To better fit the scene, we imported another custom wooden torch asset generated from Adobe Illustrator as shown on the below left to match with the fire. When in motion, the two look pretty realistic as a torch.

Final Result

After all the efforts below, here are a few screenshots of our final game.
To play our game, please refer to the executables on github:
https://github.com/YAXINLEI/janet_yaxin_175final.git