“Picture Perfect” Emily’s Devlog

By Emily Donoghue (May 7th, 2024)

Overview

Throughout this whole bootcamp, we knew the capstone project was on the horizon and as we got further into the course I kept that in the back of my mind. I wanted to evaluate how much I personally was able to do in Unreal Engine 5 and determine which type of game was realistically feasible with my technical abilities. In my opinion, the only genre I thought I could pull off with a relatively complete game concept and vertical slice was the cozy game genre. The cozy game genre is not something I usually gravitate towards but I was willing to give it a try.

Once the capstone was finally upon us, Theo and I partnered up and we quickly began discussing genres. We had almost immediately agreed that the cozy genre was the best way to go. From a technical perspective, the stress-free gameplay that is a staple of cozy games seemed relatively easy to not only design but also to implement in the engine. From other projects shown in class, I saw that Theo enjoyed exploration games and knew exploration had to be included in whichever game we created.

Later on, Gavyn joined our team which we have named Cozy Camera Studios. We started brainstorming and began the first draft of our game design document. Initially, one of the first ideas that was brought up was similar to the final idea we chose. However, instead of the player looking for animals to photograph, the player would actually be the animal and their goal would be to avoid the hunters and photographers. That idea didn’t go very far once we realized that it would become less of a cozy game and more of a wilderness survival game. Long story short, we flipped it and that was basically how we decided on the final idea of our game. Early on, we also created a discord server with various channels to discuss art design, mechanics, level design, and to gather references and resources.

Through our research we found a few similar games that fell under the cozy exploration photography simulation umbrella like Lushfoil Photography Sim,

Pokemon Snap, Minecraft, Pupperazzi ,Toem, Alba: A Wildlife Adventure, and Martha is Dead. They were all inspiring and they helped us discover our overall intention and solidify the pillars of our game- exploration, photography simulation, and relaxation.

Picture Perfect is a calm, photography simulation cozy game rooted in exploration, creativity, and a love for the natural world. It should be stated that our final iteration of the game, or should we say the iteration we are submitting for the assignment, isn't the same game we initially came up with. It has been workshopped, modified and polished in order to create the optimal player experience in our time constraints. In the end we believe we have created a game that lovers of cozy games will enjoy. I should also mention that I was only responsible for blueprinting the mechanics. Theo and Gavyn handled the level design and Theo also took care of source control troubleshooting, so I can only assume they detailed their journey in their respective devlogs.

Camera

In terms of gameplay, our game is quite simple, but that doesn't mean that we don't have several mechanics and systems working together to create an engaging and immersive player experience. When we designed the mechanics we wanted to keep our pillars; photography simulation, exploration and relaxation in mind. Since our game is a photograph simulation, the most important gameplay mechanic involves the ability to take photos so I started blueprinting that mechanic first.

From a player experience perspective, the goal with the camera system was to ensure that it didn't feel like the player was taking a screenshot of the environment. We wanted our system to work like a real life camera so I designed a system that is both intuitive and simple. Having said that, from a technical perspective, my goals for the camera were the following; I wanted our camera to come up in front of the player's face, I wanted the player to zoom in and out with the scroll wheel, and I wanted to have the ability to toggle between various lens focal lengths by clicking the middle mouse button.

I scoured the internet for a "camera" tutorial but most of my findings were demonstrating how to create a "photo mode," similar to in-game screenshots. After extensive research, the only tutorial that was relevant to creating a photography mode was created in a previous version of the engine and required some input mapping modifications in order to get it working in UE5. Furthermore, our photo mode was still slightly different from what was demonstrated in the tutorial. In the tutorial, they had created a camera that could disconnect from the player character and fly around the map similar to a drone and that wasn't our goal. So after a lot of brainstorming, I realized that we wanted our in-game camera to work similarly to a sniper gun. Following that breakthrough realization, I overhauled the whole camera system to start creating an Aim Down Sight (ADS) system and finally, we were getting somewhere!

Next up was getting the camera to zoom in and out seamlessly using the scroll wheel, which was pretty simple to set up… or so I thought. Initially, I blueprinted the player camera's spring arm to grow and shrink in length, essentially moving the camera back and forth in physical 3D space thus creating a "seamless zoom.". Unfortunately, that caused the camera to clip through geometry and cause some extreme zoom scenarios. For example, say if the camera was already set to use the 50mm lens and the player used the scroll wheel. The camera would move forward in space in addition to having a narrower field of view therefore it felt like the player had zoomed in way more than intended. So the ADS system was working, until it wasn't and I had to rethink how I thought about the zoom function.

Unlike a sniper, I had to take into consideration that we were making a camera- with a lens. An in-game sniper weapon with a scope with an arbitrary zoom function works very differently than a camera that uses focal lengths, aperture and a variety of other things to capture a photo. I knew I wanted the player to zoom in two different ways. First of all, I wanted a zoom function where the player could use the scroll wheel to seamlessly zoom between the 14mm lens and the 50mm lens, somewhat imitating the way a digital camera works. Second of all, I wanted the player to have the ability to choose the exact lens size they desired by clicking on the middle mouse button and switching between our preset lens sizes.

In the end, I decided to go with five basic lenses, which offered five different levels of zoom; the 14mm, 20mm (default), 24mm, 35mm, and 50mm. As a team, in our initial conceptualization, we knew we wanted the player to have the ability to purchase telephoto lenses from the store, which is why we didn't include any lens larger than 50 mm. Interestingly enough, UE5's cameras measure their camera focal lengths using a field of view variable which is in degrees and not millimetres, so I also had to ensure I got the conversions right too.

Next on the list was creating the ability for players to actually capture a photo. It took some playing around with to get the scene capture component, the render target, the camera, and the widget to all communicate with each other, but in the end it was all pretty painless to get working. The only two problems I ran into were getting the scene capture's field of view and rotation to match that of the player's camera, aka our in-game camera. In retrospect, it was a simple fix. A few node additions in the blueprint editor and a change in hierarchy in the player character blueprint and it was all solved. At the time, I definitely over-thought it, and had turned a molehill into a mountain. It wasn't the first time and it wouldn't be the last.

To finish up the camera, the last essential part was to create a way for the player to view the photos they captured. I made a quick photo display widget that would show the most recent photo the player captured. It would appear briefly at the bottom of the player's screen after the photo was taken, and I also animated the widget to enhance the player's experience.

Phone

Once I felt confident that we had a workable camera, in an effort to procrastinate what I deemed the second biggest mechanic (the animal AI system), I worked on the third biggest mechanic- the in-game smartphone. Our goal with this mechanic was the most simple. We just wanted a phone that worked intuitively like a real life cell phone. I couldn't find a free 3D smartphone asset online that I liked, so luckily my education where I learned to become a 3D generalist came in handy. I modelled a phone in Maya, created the UVs and made some simple materials in Unreal.

Implementing the phone was an interesting process to say the least. I made a phone blueprint, "BP_Phone", added my model as a static mesh and added that blueprint as a child actor of our player character blueprint. In order, to get the player to interact with the phone, in the player blueprint I also added a widget interaction component.

After adding the interaction component to the player blueprint, I encountered a weird issue in the player's movement. Suddenly when trying to move forward, the player would move diagonally, and jitter weirdly. This one was interesting to figure out. I tried moving the child actor outside the player capsule, which would work for a while and then caused the same issue the next day. After sleeping on it for a few days, in the end, it was actually the collisions on the phone blueprint that was causing the issue. Having the phone to the right of the player and having the player attempt to move forwards, would cause the player to "collide" with the phone's mesh and would force the player to move more to the left, thus the diagonal movement. I don't remember how I figured this out but I'm happy I did.

Next up was creating the widgets that would essentially become the phone's UI. While implementing the widget interaction component, I encountered an issue that was really difficult to figure out and spanned across several days. As I was following a tutorial, I had everything working until I reached nearly the end of the video. Suddenly, when I playtested I wasn't able to interact with my widget. Long story short, after many hours of searching the internet for a solution, it turns out that the actual tutorial that misled me. At one point in the video, while in the blueprint, he switches out two nodes for ones that are extremely similar. It was easy to miss because the tutorial had no audio and the video edit essentially does a matchcut- it was a blink-and-you-miss-it change that drastically affected the final outcome.

Fortunately, the next few steps were pretty straightforward and relatively simple to implement. I began conceptualizing and further designing the freelance gig app, the store app, the exposure level, and the point system. I quickly created 2D assets for all the phone UI buttons and applications. For the store, I organised all of our store items and their prices. For the Freelance App, I had to design short narratives that would make sense given the animals and environment we had, and I

determined when each gig would be unlocked, and the player's rewards upon completion.

By the end of March, I was wrapping up the phone and camera. I had to make some UI adjustments, UI phone animations, the principle character animations (lifting the camera up/down to the players face, and bringing the phone up/down into the player's view), and I created blueprints that ensured the camera and the phone couldn't be used at the same time. In addition, when switching between regular camera mode, photo mode and phone mode I had to ensure that all the input modes worked accordingly. I also created a player HUD that displayed which freelance gigs were active, and would only be visible once the player accepted the job.

It should be noted that it was in the last few days of the month when the Email app was born. I wanted an engaging way for players to finish the freelance gig and receive their rewards, so sending an email off to their client seemed like a logical solution. Finally, I added a fade-to-black animation to the photo mode UI widget. Therefore, when the player lifts the camera to their face, once it gets to a certain distance from their face, the camera fades to black and opens back up in photo mode. That way, the player wouldn't be able to see through the camera mesh when they transitioned to photo mode. As many designers can attest to, the camera and phone system required rigorous playtesting to get it working the way we designed it.

AI Animals and Camera Sensing

With the first month of development behind us, April had begun and there was still much to do, specifically the second most important aspect of gameplay- the animals. Fortunately, I learned my lesson when making freelance gigs. Blueprinting something one time, testing it to make sure it works the way you want it to, and then duplicating it later is much easier that partially blueprinting something, duplicating it and then making changes which forces you to make those changes on all the duplications. In other words, don't duplicate your errors- finish the first thing before you move on to the next. Work smart, not hard.

That being said, luckily blueprinting the animal ai was relatively straightforward. I followed some tutorials online regarding NPCs and AI systems and applied it to the animals we had downloaded from an asset pack from the FAB marketplace. I started coding the crow first. I blueprinted the crow to fly, drain its energy, land to rest, and then take off to fly again once it regains its energy. In the end, I scraped the energy system for the crows, but it still applies for all the other animals.

Unlike the animal AI, it was a little trickier getting the camera to sense the animals. I knew I had to create some sort of line trace from the camera and that would also require each animal to have their own collision boxes around them. Therefore, when the collision box is hit the animal "takes damage" which just means that exposure points get added to the exposure meter. I started by doing extensive

research on the different forms of line tracing in UE5 and decided that a box trace would work the best for our use. Essentially, the way our camera works is that once a player takes a picture, a box trace gets created from where the player is, to a location about 2500 units in front of them. If the box trace collides with an animal, then the player receives points. The troubleshooting came in when I had to figure out how the box trace would know that it hits an animal and not a tree. To summarize, I had to create a custom object type for each animal and another custom object type for the freelance gigs. Then I changed the node from “box trace by channel” to “box trace by object” and set it up so that the trace would only register my custom object types.

The next task was to place the animals in the map to playtest them, which meant testing to see if they would actually roam within the bounds of Unreal’s navigation volume. Luckily, that worked almost immediately so I moved on to blueprinting the freelance gigs. Even more luckily, especially since we were nearing the end of the project, creating the blueprints for the gigs was pretty simple too. Essentially, I made a blueprint for each gig, and I made a blueprint spawner for each of those gig blueprints. Having done that allowed me to place the spawn in the map, which the player wouldn’t see in game. However, upon accepting the corresponding freelance gig, the collision box required to complete the gig would spawn so that the box trace could hit it when a photo was taken. Just like the freelance gig UI system, I duplicated that process for each animal and each freelance gig.

Photo locations

At one point, Theo had brought up the idea of having additional photo locations where players could capture picture perfect photos. Having finished the animal points system, this was an easy task so I enthusiastically agree. Our goal was to allow players to earn points in several different ways and it made sense that there should be a reward for beautiful landscape photography. They work the same way the animal and freelance system does but instead of an animal mesh, these locations are indicated by a white dot. Theo and Gavyn did such an incredible job creating a gorgeous environment that felt realistic in its natural beauty so how could we not include this mechanic.

Day/Night Cycle

By the end of April, Theo and Gavyn had finished grey boxing, set dressing, and overall designing the level game, which I’m sure they’ll explain in more detail in their respective devlogs. On my end, I also had a few things I wanted to add to polish off our vertical slice. I added a quick day to night cycle by adding animation to the main directional light. In game, the sun will quickly rise, stay in the sky for 20 minutes, and then slowly set. Since our game doesn’t allow our players to play at night, at the end of the cycle a widget pops up onscreen that prompts the players to start a new day. Upon clicking “start new day,” the screen fades to black and the

player respawns at the cabin where the day/night cycle starts over with the sun rising.

Tutorial

Before we packaged the game, we realized that we had time to create a tutorial. We had discussed that at the start of the game, the player would be prompted to take a picture of a specific crow. I called it “tutorial crow”, and taking a picture of it would award the player with 50 points- enough points for them to progress to the first exposure meter level. Theo and Gavyn brought up the fact that we should also show players what a photo location looks like- something that tells them to look out for them in their explorations. Fortunately, that was another relatively simple task and I finished it pretty quickly. Now, the tutorial crow and the first photo location are both worth 25 points each.

Finalization and Conclusion

By the beginning of May, we were in the thick of crunch time. We had to tie up any loose ends and Theo and I worked on optimizing the scene. I modelled a sign, similar to that of a national park sign, and created the UI for it. We thought a sign that one would find at a park entrance that explained the rules of the park was a fun way to explain the game’s controls. I added a few sound effects and ambient nature sounds, and Theo and Gavyn added some park direction signs. I also created a start screen and settings menu that is displayed when the game launches. It should also be noted that throughout this whole process Theo constantly worked on troubleshooting all of our source control issues and any errors that we ran into in the engine.

Lastly, we recorded gameplay and recorded our pitch presentation for pitch day. I edited the presentation video, while Theo packaged the final build of our prototype. As a group, we polished up the final draft of our game design document and any other documentation was wrapped up and ready for submission. We’re very proud of our game and our final vertical slice. I’m proud that we were able to take this project across the finish line while working as a team and still liking each other afterwards. The work was the right amount of challenging, and I’m confident that we didn’t let scope creep become our downfall. To say I learned a lot about blueprinting would be a gross understatement but I had a lot of fun, and I’m excited to see what we all create in the future!