Senior Project: 3d Card Game

For my senior project, I set out to create a 3D video game using the Unity game engine. My goal was to create a card game where two players take turns placing cards into the 3D world and attacking one another's cards. This project helped further strengthen what I learned in the Game Development course I took and expand upon it. In this paper, I will discuss why I chose Unity for my project, as well as some of the key challenges I encountered along the way.

When deciding on the engine to use for this project, I narrowed it down to three possible options. Unity, Unreal, or Godot. From my research and prior knowledge, these three are at the top of the list. While I ultimately settled on Unity as my game engine, all three have their advantages and disadvantages. Unreal Engine provides some of the best graphics and rendering capabilities, as well as the blueprint system. Godot is very lightweight, open source, and great for 2D projects.

The first and main reason I chose Unity was that I already had some experience with the engine. Prior to taking the game development course, I experimented with Unity. For the final project for the course, I created a sandbox building system, but not a full game. I had only worked with Unreal Engine's blueprint system and 2D Godot games, so this put Unity at the top of my list from the beginning.

Second, Unity has a massive amount of tutorials, forums, and documentation. In my opinion, there is so much to learn when it comes to game development. These options really help when it comes to learning new features Unity has, adding new features to your project, and troubleshooting. While most of the time tutorials don't work with your specific case, they do help you see the problem differently and how to work through it and tailor it to your project. Unity's documentation also provides reference guides for each component, class, and method.

The last reason I decided to choose Unity was that it excels at cross-platform compatibility. Developing the game for different platforms wasn't a requirement or goal for my project, but I was working on both Mac and Windows and wanted to easily be able to test and deploy on both platforms. The combination of experience, available resources, and compatibility ultimately led to Unity being an ideal choice for my senior project.

One of the first major challenges I faced was learning how to navigate Unity's UI. Even after some experience with the engine, I found myself taking a lot of time figuring out how to connect certain objects or the proper object to choose. Everything is presented to you very straightforwardly, but there is so much that goes into a game, and you almost always find something new to figure out.

The hierarchy window lists every game object in the scene. The large amount of different game objects to choose from, as well as managing what should be a child and what should be the parent, was very confusing initially and took some trial and error to figure out. Even after this project, I feel like there are different or better ways I could've organized the project.

The inspector panel was another component that took some time. This contains all the components that the game objects hold onto and their settings. This is where you can view, remove, or add mesh renderers, scripts, colliders, and plenty more. The panel is very nice for viewing what components each object has, but it can be tough to figure out what each setting does.

The last component of the UI that required some practice was the scene view. The scene view is the 3D space where the objects exist. Initially, it did not feel intuitive to navigate the scene and align objects in the space. This slowed down my workflow when it came to most of the 2D UI elements. However, almost all of my game objects on the screen are instantiated through code, so the scene view was helpful for debugging what the position or orientation of objects was off.

Understanding the UI was essential for everything else I did in this project. The more I worked on the project and figured out where everything goes and how they work together, I was able to work a little faster. There are still plenty of features and components that I haven't even touched.

Another major challenge, and one that probably occupied most of my time on this project, was learning the large number of classes, methods, and objects Unity has. It was very daunting at first, trying to figure out what to use and how to use it. Sometimes I would start with an idea in my head, but struggle to describe it well enough to find something I could use. This again is where tutorials were helpful, watching someone solve a problem allows you to see the different classes, methods, or objects you could use. Once you know the name of it, you can then find out how to use it, either from the tutorial itself or in more detail from Unity's documentation.

My project consists of two different types of C# scripts. Most scripts are monobehavior scripts, and a few are scriptable object scripts. Before this project, I had exclusively used monobehavior scripts, so I decided to see what I could use one of the other options for. I was able to use scriptable object scripts to easily create new cards in my game and store the data associated with them.

Monobehavior scripts have to be attached to a game object in the scene hierarchy. This allows you to reference the components attached to the game object in the code and do things like manipulate their position, rotation, and material, which was very useful in my project. You can also attach these scripts to an empty game object to manage different events because you get access to lifecycle events. The lifecycle events give you access to events like the start, when the game updates, when collisions happen, and more. In my case, I have empty game objects that handle spawning the grid for the map, handling the deck, turns, and the general game flow.

Scriptable object scripts exist independently in the project and are not attached to game objects. They are used for storing data or resources that can be accessed by monobehavior scripts. In my project, I used scriptable object scripts to easily create new cards and store their data, as well as creating a card library that holds onto all the cards so I can easily access a list of all the cards. Doing it this way allowed me to easily connect things in the editor rather than with code. This made it take a little more time to meet the lines of code requirement, but I wanted to focus on using these features to keep my project structure clean. I noticed really clean hierarchies in a large number of the tutorials I watched, and my previous projects in Unity seemed to always end up a mess. This made it very easy to find objects in my hierarchy and the components attached, rather than scrolling through what could end up being a very long list.

One of the most technical challenges I faced during the development of my project was raycasts. Raycasts are lines projected from one point in the scene view to another and are used to detect collisions with objects in a 3D or 2D space. My game relied heavily on raycasts for interacting with the tiles, cards, characters, and UI.

In my game, the player interacts with 3D objects in the world. When a player clicks on one of the 3D objects, the game needs to know where that click was in the 3D space compared to your 2D view. Without raycasts, I would have to do my own calculations to get this information.

With a ray cast, I can draw a line from where the user clicked out a set distance and detect what it collided with. The method requires a ray object, an out parameter for the raycast hit, distance, and optionally a layer mask. Because the camera exists in the scene view, I can draw a ray from the point on the screen that was clicked and send a ray out at the angle the camera is at. I set the tiles, characters, and cards to different layers so I can detect the collisions of those layers separately. Then, as long as I send the ray out far enough that the camera can't be too far away, I’m able to get what the user intended to interact with.

The last major challenge I faced was designing and implementing the general gameplay system. Compared to the others, this wasn't much of a challenge. I have previously created card games for Discord bots and even had an assignment to create a deck of cards and shuffle them using the Fisher-Yates shuffle. The main challenge was applying this knowledge to a 3D space.

Throughout development, I constantly ran into unexpected bugs, behaviors, or errors. Sometimes the cards weren't rotated correctly, randomly not being able to interact with objects anymore after adding a new feature, or random errors that didn't seem to make sense for what I was doing. I was able to overcome all of these through trial and error, tutorials, and the documentation. I encountered bugs that made me think something was wrong with my program, that I found out weren't important and could be ignored. Several times I had to resort to the classic shut it down and turn it back on again method.

Unity provides a few good tools that I discovered for debugging. Gizmos helped visualize the raycast in the scene view, which helped me determine how far to send out the ray. Different debugging messages and compatibility with Visual Studio also heavily assisted debugging. Sometimes I resorted to an old method of placing log statements before every line to see where it failed. In the end, I had enough resources to solve the challenges I faced.

While I'm frustrated that I'm not good at the graphics aspect of game development because my game is not visually pleasing, I'm happy to have completed a working game for the first time. Every time I've made a project in Unity, I've learned something new or a new way to do something. This time was no different. I feel like I could start this whole project over in a different way or structure things differently, and I'm sure I would be in the same boat again. On one hand, I feel like I've learned a lot, and on the other hand, I know I'm just scratching the surface.

Along with trying to keep my project clean, I seprated almost most things into their own scripts. I'm unsure if this was the best solution, as it's a lot of files to look through. Some of the relationships I made between the scripts, objects, and prefabs also became a little hard to keep track of. If I were to attempt this project again, I would research better or more professional-looking ways of separating and organizing the files. I would also focus on organizing the relationships between all the objects in a more efficient manner.

One of the stretch goals that I would have like to have implemented but didn't was networking the game. This is something I've attempted before and will probably try even after this. In prior attempts, I was able to get two instances of the game to connect in the editor, but when I tried it outside of the editor as a build, it would fail. One of the steps was connecting everything to a relay server. The one recommended to me most was Steam's relay servers. This is where I believe I ran into issues, as the logs seemed to point to an issue connecting. The method I tried was for peer-to-peer connections. Had I been able to get peer-to-peer to work, I would have liked to attempt to try it with a server. In the end, I will still most likely attempt to implement this either in this game or the next project.

I'm happy with how the project turned out and with my choice of game engine. It was very nice to have some practice with the editor previously. Unity uses C # for its scripts, which is a language I was comfortable using. It was nice to be able to find a tutorial or documentation on just about anything once I was able to get the words to describe it. Unity has received some backlash from the community because of recent changes made to revenue. For a while, it seemed like Unity might have dug itself into a deep hole. I have no plans to monetize a game currently, and plenty of developers still support Unity after the changes and some more clarification. I'm glad I chose Unity and furthered my skills with the engine.

Developing my senior project was challenging, time-consuming, but also rewarding. I was able to create my first full game in Unity and learned plenty of new things I can apply to projects I've been wanting to make, but needed more practice for. On one hand, I felt like I should have chosen something that would have been more challenging on the coding side of this project, but I enjoy making games and am glad I learned more about how Unity works and what it takes to make a game.

Corey Jandreau

108 Riverland Drive
Marquette, MI 49855
(906) 373-7257
Coreyjand@gmail.com

OBJECTIVE

To gain exposure to profession while persuing a college degree.

EXPERIENCE

Meijer, Marquette — deli

November 2020 - Present

- Serving food
- Cleaning
- Stocking

Eagle Mine, Humboldt — Mechanical Maintenance Intern

May 2022 - August 2022

- General Repair
- Welding
- Operating forklift
- Operating overhead crane

Eagle Mine, Big Bay — surface operations

May 2020 - August 2020

- Operating Sweeper truck
- Painting
- Cleaning

The Rock Shop, marquette — operations worker

September 2019 - April 2020

- Cutting stone with CNC and bridge saw
- Moving stone slabs by hand
- Moving stone slabs with forklift

Eagle Mine, Humboldt — *Warehouse summer student*

June 2019 - August 2019

- Grounds care (mowing, weed trimming, painting)
- Pallet unloading
- Package delivering
- Cleaning
- Furniture assembling

Great Lakes Fresh Market, Marquette — Stocker/Bagger

August 2018 - March 2019

- Stocking and facing shelves
- Bagging groceries
- Cleaning

EDUCATION

Northern Michigan University, Marquette — *Currently Enrolled*

August 2019 - Present

Marquette Senior High School, Marquette — *High School Diploma*

June 2019