BOTHELL
AUTUMN
2021

DYNAMICS & ADAPTATION IN VIDEO GAMES

CSS 496 APPLIED COMPUTING CAPSTONE

RYAN COLLINS

Table of Contents

Introduction & Background

Video games have been a huge part of my life as far back as I can remember. The original Game Boy handheld and the NES occupied a lot of my time in the mid-1990s, and I have continued to love video games into my 30s. I first became interested in video game development when I was 14 years old and taught myself enough BASIC to make simple games like Pong. The resources available online weren't quite what they are today so I relied heavily on published books that can't be asked questions. I tried my hand at developing in other engines such as RPG Maker 2000, RPG Maker 2003, etc. but never stuck with it enough to really produce anything substantive despite my interest in doing so.

Applied Computing, focusing on video game development, was not my intended course of study. The first two years of my education were heading toward computer engineering which derailed a bit when the pandemic sent everything online and course offerings became scarcer. This was tough at first, but since pivoting my major and focusing on game development, I have found a much more satisfying fulfillment in the work I've pursued, especially this project. I hadn't considered this field before, I think, because I didn't truly understand the variety of skills and knowledge that would feed into it, all of which I have really enjoyed studying. Advanced mathematics, physics, and audio engineering are all key parts of the video game development process, are all highly interesting to me, and are fields in which I feel I excel.

This project intends to address an issue I have had with the games that I play: character interactions are usually hollow or meaningless, static beings that regurgitate information and hold no influence over other characters or the world around them. Certainly, there are some games that go beyond this and create meaningful character relationships, but this is often achieved through games where the entire focus is the narrative and characters, the rest of the game tends to be unengaging in any meaningful way. Roleplaying games try to highlight character interactions and world changes but usually fall short of their inspiration: tabletop roleplaying games like Dungeons & Dragons or Pathfinder. The narrative and character interactions end up taking a backseat to combat systems or loot-chasing in dungeons. These are core components of the genre but are almost always front-and-center over the story and interactions.

Three Pillars of Dynamics & Adaptation

Many processes and constructs are organized into a three-pillar architecture, so I've adopted a similar thought process for tackling this project. I have identified three problems that I perceive in a plurality of video games and my approach to

address these problems: creating believable non-player characters, constructing a digital society, and affecting the world through player choices. The work I've laid out within these pillars is certainly out of scope of the time allotted for this project but tackling these problems will lay a foundation for future development to use these systems in roleplaying game project development.

The First Pillar: Creating Believable Non-Player Characters

In most games that I have played, non-player characters tend to feel static, as if their entire purpose is to be a billboard of information or to give the player tasks or act as a vending machine. They rarely feel as though they are part of the world or influenced by anything other than whether the player is nearby. Many games implement some sort of time-of-day feature to allow for more interesting visual lighting or to signify that more dangerous monsters are afoot, yet NPCs tend to remain in place at all hours, constantly available for the player to interact with.

Non-player characters should have lives, too, not just act as the player's information broker, and should react accordingly to things such as the time of day, the weather, or the presence of castle guards in their typically quiet town. I plan to do this simply, using finite state machine and node-based scheduling. This will allow for NPCs that can simulate going to a job on certain days, spending their weekend shopping at a market, or leaving town for the winter. They'll be unavailable in the night hours (unless their job utilizes a night shift) encouraging the player to rest themselves, or explore the town without others in their way, or sneak into characters' homes if they decide to take a more criminal route. Making characters unavailable to the player at certain times or due to certain triggers *can* be risky as this may cause the player to feel as though their game progress is being gated.

In addition to portraying NPCs as though they have lives of their own with jobs, tasks, and summer homes, NPCs should be influenced by the world around them as well as the other characters. Typically, in games, very little inter-character interaction occurs unless it's related directly to the player. This doesn't align with real-life even remotely, and causes game characters to feel flat and again, like a billboard. I will employ a data structure that allows actions by one NPC, or the player's actions with an NPC, to cascade through the rest of the NPCs including their family and faction standings. This will allow for consequential choices that are obvious in the world and in the future interactions with and regarding that NPC.

The Second Pillar: Constructing a Digital Society

Nearly every person is part of some sort of organized social structure be it family, nationality, company, etc. The relationship between two individuals can often

be influenced by their personal standing within these organizations as well the standing between the organizations. You can be born into a family with generations of rivalry with a neighbor family because your families are economic competitors. You may work for a company that produces rocket ships while another rocket ship producer is constantly trying to "one-up" your company. Organizations are just as capable of rivalry and friendship among its members as individuals are with others. These influences are rarely seen in any substantial way in video games other than a few lines of text expressing the rivalry or friendship.

To model a society, I begin with the individual. The individual's relationship within their family, the first organized social structure you can be a part of, is determined by factors such as the job they hold, their personality, or their level of wealth. The family can view these aspects relative to their history, such as a family of carpenters looking down on the members that took desk jobs, or a particular person within a family who just can't seem to be nice to anyone. These traits can be assigned to individuals and within the family, numeric values can be related to these traits, creating a family standing value. Families, in turn, have a standing within the faction they belong to. Families may have rivalries or friendships with families within the same faction and factions can do the same with other factions.

The Third Pillar: Affecting the World Through Player Choices

Generally, video games focus less on inter-character interactions and world influence in favor of player interaction and influence. This is not unreasonable; the player is the person who must be pleased by the game, not the digital characters. With so little focus on the other characters and world, however, it results in many games feeling more like a playground of billboards than a living, breathing world where relationships and interactions matter. Creating more immersive and believable characters and worlds may not work in every case to maximize player happiness, and may even be a niche, but there is no shortage of players who want to see NPC relationships influence the world in which they're playing.

To affect change within the game world, the relationship system is imperative as the things in the world that I want to influence can be tied to characters or the organizations they are members of. The player must be able to influence the relationship between characters and the organizations, but as an agent rather than the influencer. The player will interact with NPCs as an agent of an NPC, for example: the player can deliver critical goods to an NPC that were sent by a rival NPC from another faction whose relationship is tense with their faction. This action can not only influence how the receiver views the sender but can also be an effort to bring the rival factions toward unity.

The choices presented to the player must be consequential, if there are no stakes with choices, the options are little more than a façade. If the player decides to repair the relationship between Faction A and Faction B, this should result in cascading consequences through the world, such as Faction C now shunning both factions as the relationship between them all was rocky to begin with and Faction C sees the uniting of A and B as a possibility of teamed-up conflict. Many games allow you to choose one faction or another, losing access to the one you do not choose, but only in a few instances have I ever seen my choices reflected in a larger scale.

Considering Why Games Rarely Tackle These Pillars

Video games are a highly interactive medium. Typically, the goal is to provide an engaging experience to the player(s) through *doing*. When the 'doing' is reduced and the 'watching' increases, the experience begins creeping toward 'interactive movie' and this can be unappealing to a large portion of video gamers. A focus on the narrative over action can lead to players getting bored, so often narratives are only provided in quantity enough to encourage the player to continue 'doing'.

Alternatively, the visual novel genre encourages narrative exploration, character relationships, and branching paths. These games tend to be solely focused on the narrative, though, and offer little in the way of engaging game systems such as combat, inventory, item upgrades, and skill trees, to name a few. This genre often caters to more exclusive fans, where this is their preferred experience, and they aren't seeking depth other than in the narrative.

Possibly the most prominent issue with creating branching narratives, complex character relationships, and an evolvable world is technical complexity. There is a risk with creating many ways for players to interact with or influence a game that many of the explorable options will never be used, or that art assets such as building sets or entire regions will never be seen because the choices that lead the player there aren't interesting enough or worth the effort. This can lead to a large part of a game being developed essentially for no one to enjoy. With this, the size of the game and potential points of failure increases. These factors can lead to player dissatisfaction or put them off the game from the beginning.

Anticipating Pitfalls & Tackling Setbacks

There were a few foreseeable obstacles when considering pursuing this project. Simulating characters, their relationships, their lives, etc. is an advanced topic in which I have little formal knowledge. There was a high likelihood that advanced data structures and algorithms that I have not worked with will have needed to be utilized

to achieve the level of simulation I envision. The next likely obstacle was the ability to create a substantive amount of content that would showcase the systems' implementation and interactivity. Along with that, there was a possibility of the choices given within the content would be of varying quality and desirability, possibly resulting in effort expended on content that would never be utilized.

Of the problems I identified, it was only the first that I felt needed any focus. Seeing as this project was unlikely to become a fully realized game, the content shortage and desirability of the content was not something that warranted high prioritization. The main problem, and the one that needed focusing was unfamiliarity with more advanced data structures and programming patterns. This encouraged me to learn and understand patterns like singletons, structures such as B-Trees or Red/Black Trees, and others to design my data structures and object management. Ultimately, I did not use the tree structures specifically, but understanding them helped guide my design.

I experienced my first, and most significant, setback about two weeks into the project. I felt as though I could approach this project the same way that I had approached other small game projects in the past, and as such I ended up with a tangled mess of spaghetti code for systems to interact. As the difficulty of creating new systems or adjusting old ones increased, I realized that it would be necessary to take a new approach to the architecture. As I came to understand my problem, I began searching for a solution with a copy of Game Programming Patterns [6]. I perused each pattern while considering what I believed I would need to make my systems interact correctly and I landed on the Singleton pattern. I set out immediately and rewrote the entire project with a central manager in mind and other than adding in some event-driven elements, singletons are used to manage each system quite effectively.

The other difficulties that I experienced were related to my data structure, the associated algorithms, and having no prior experience working with any API. I decided to use a program called Ink that provides syntax-based text asset parsing to run my dialogue system as I quickly discovered that creating even a basic dialogue system that supports branching and inter-character interaction would take ages on its own, let alone parallel to the rest of the project. Having never worked with an API, I found some challenges in integrating the program with my project, especially some aspects of their interface that provide exactly what I needed but did not provide substantial documentation on the functionality.

Reflecting On the Process & What It Taught Me

This project was a solid reminder that I have a lot of room to grow as a developer, and a lot to learn about programming patterns, data structures, and event-driven systems. As I continue to develop games solutions to problems will become more obvious as I have more previous work to reflect on and draw from. I like to think that when I reach a point where I do not need to use references or look up how to do something that I will instead opt to tackle something unfamiliar as I believe that learning should be a constant process and should never stop. Consistently challenging myself appeals to me greatly and I have found that in the past when I felt as though I was no longer learning about something that I was no longer interested in doing it. Game development has provided such a consistent challenge to me that I know that I will never truly know it all and can look forward to a lifetime of learning.

Time on this project was spent near equally between actual development and learning through reading, tutorials, and trial-and-error. I approached many concepts that I had not before, and I feel as though I could further improve on my ideas here to make them more manageable, optimizable, and user-friendly. Ultimately, these systems would be provided to a designer who will have little knowledge of coding and will need to implement the concepts based on the interfaces and guidance I provide.

In past projects I usually approached them without prior planning. I know that problems arise more often when planning is not considered, but it was usually an effective approach for me, so I attempted the same here. Initially it worked. I planned a simple outline of the systems I intended to implement and set out to code them. As they began to come together, I couldn't help but think "Ha! I *can* get away with minimal planning!" Of course, two weeks into the project I was having a crisis of endless frustration with working with my slapped together codebase. I understood at that point that the quick iteration projects I had completed previously were well suited to the format, as they contained minimal inter-system interactions and were largely self-contained. I will never again approach a multi-system project in that manner. Even the amount of planning I engaged in for the rewrite may not have been sufficient for my true end goal for this project.

Along with the development aspects, I learned a great deal about game development such as tool usage like Blender for 3D modelling, ProBuilder inside Unity for quick level prototyping, working with shaders (even coding them), and a plethora of coding concepts including event-driven coding, which is immensely powerful for use in future projects and will allow further decoupling of classes.

Conclusion & Next Steps

On a scale of yes to no, I would rate this project's success as "mostly". I set out with great ambition to create a set of systems that I want to experience in a video game and while I do not have an epic forty-hour game to showcase after ten short weeks, I do have a project that proves that I can make great strides and learn quickly to create something substantive. To paraphrase a popular quotation: "the destination is nothing; the journey is everything." My project showcases exactly what could be expected of the short timeframe and exhibits the potential to expand from this point to provide players with a world that feels interactive and influenceable.

This project specifically will become a baseline from which to start developing a roleplaying game at Neon Gear Game Studio. It will require much expansion and optimization but has shown enough promise that it feels worthwhile to continue. To fully realize a roleplaying game from this project, we'll need to further expand the inventory system, add some method of conflict resolution whether it's combat or otherwise, and develop a story to tell that takes advantage of the systems created in this project. The other systems necessary to make this into a full game that could be released to the public will take substantial time and require further learning.

I completed this project completely on my own other than using purchased assets like 3D models and textures. I will not, however, be working alone at Neon Gear Game Studio, and I will be required to commit much of my time to the Navy meaning I need to "run" the studio well if I desire success or even completion of a project. While the work I did on this project was by myself, I learned an incredible amount about the game development process that will allow me to direct and delegate different aspects of the process to the other three members of the studio. Understanding better the entire scope, I now know where I best fit in terms of the development cycle and where I can best place the other members to utilize their specific strengths rather than our current process of everyone just working on whatever task is next. The idea of being a master in all aspects of development is certainly more foreign to me now than before, and I understand that for Neon Gear to do *good* work and to succeed as a studio we must become more focused in our process, beginning with planning.

Resources

[1] *3D Art for Games - Unity + Unreal*. (2021). Synty Store. Retrieved December 12, 2021, from https://syntystore.com/

[2] *Brackeys*. (2012). YouTube. Retrieved December 12, 2021, from https://www.youtube.com/c/Brackeys

[3] *Buckland, M. (2005). Programming Game AI by Example [E-book]. Wordware Pub.*

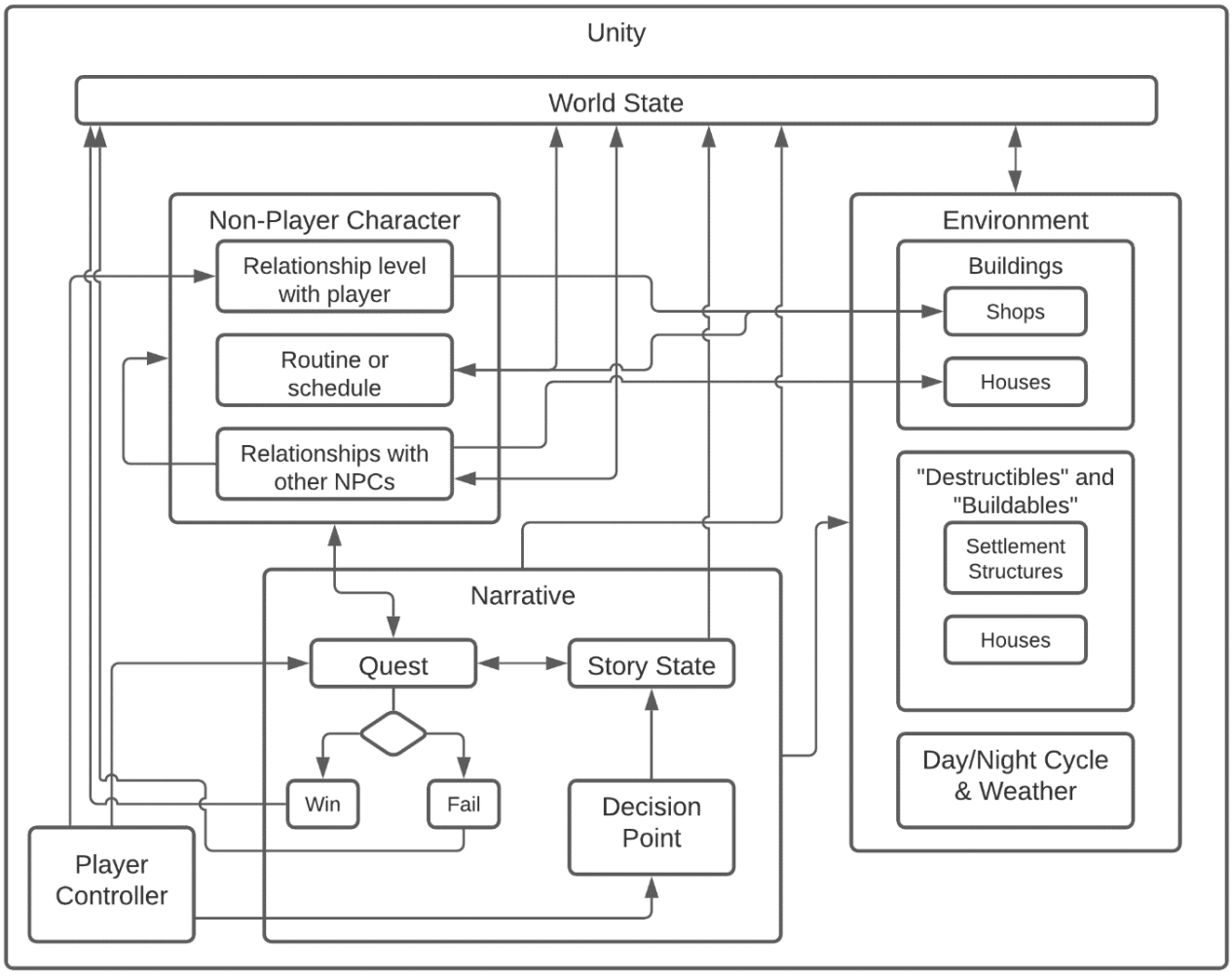
[4] *Code Monkey. (2018). YouTube. Retrieved December 12, 2021, from https://www.youtube.com/c/CodeMonkeyUnity*

[5] *Imphenzia. (2009). YouTube. Retrieved December 12, 2021, from https://www.youtube.com/c/Imphenzia*

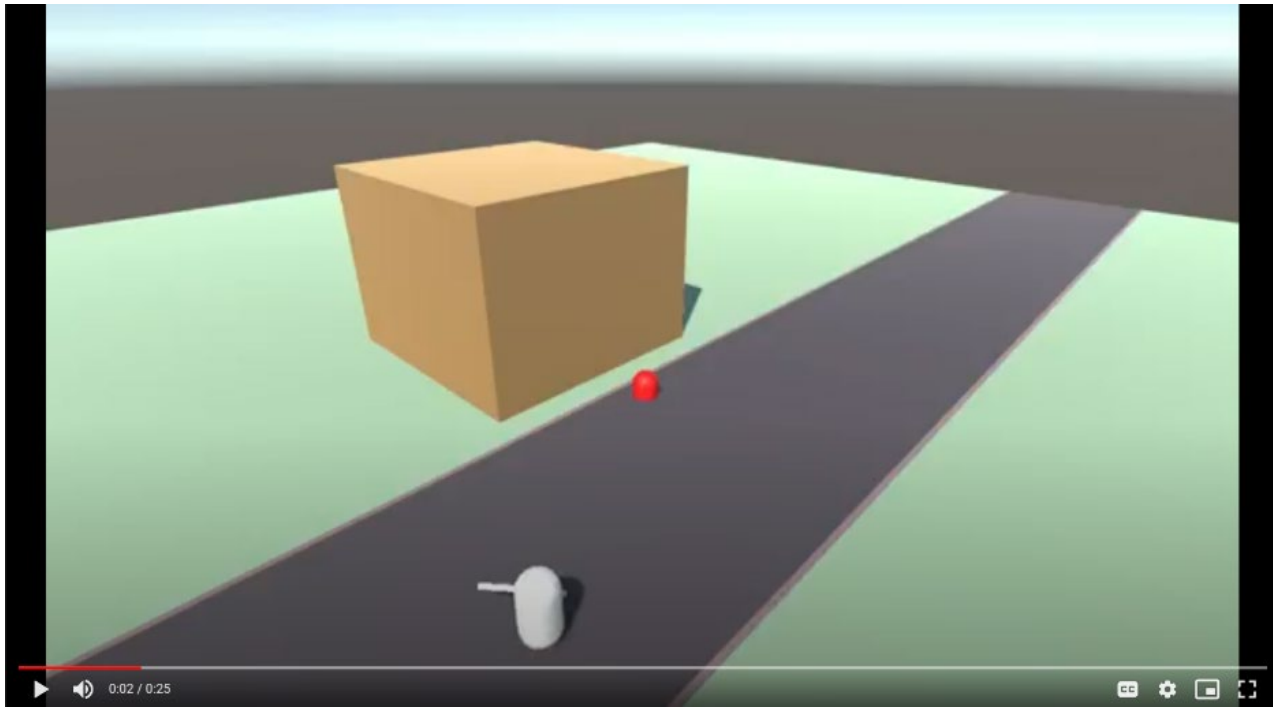
[6] *Nystrom, R. (2014). Game Programming Patterns (1st ed.) [E-book]. Genever Benning.*

[7] *Technologies, U. (2021). Unity - Manual: Unity User Manual 2020.3 (LTS). Unity Documentation. Retrieved December 12, 2021, from https://docs.unity3d.com/Manual/index.html*

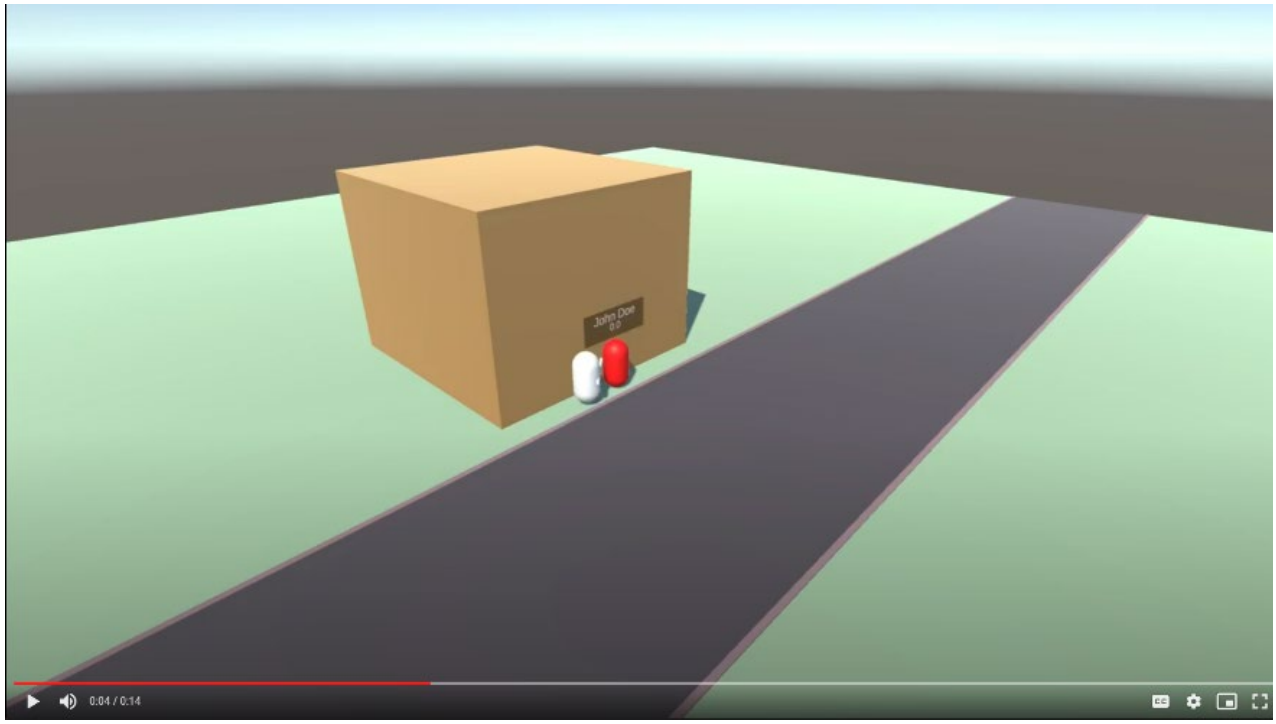
APPENDIX: Artifacts



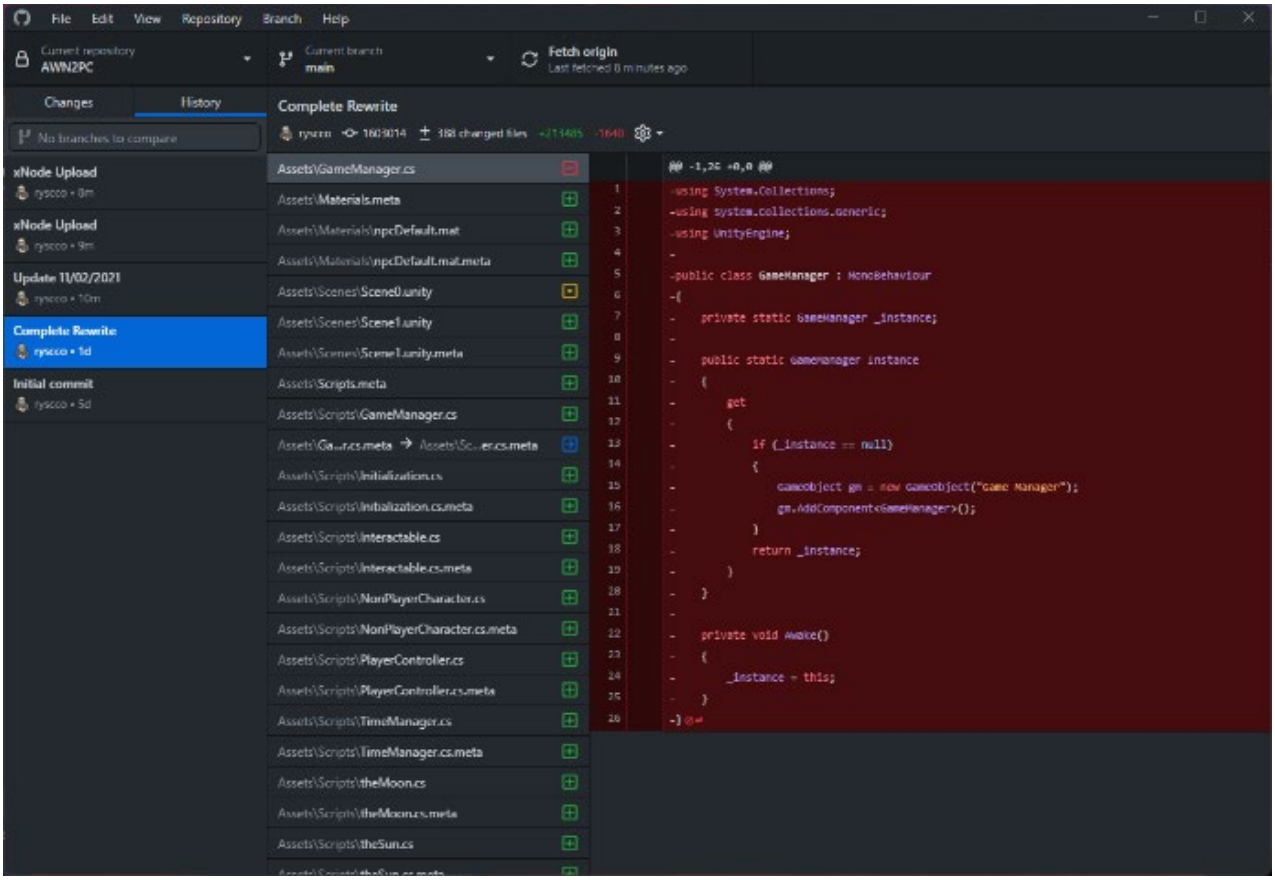
This is the initial conceptualization of how the different systems will interact and influence each other. This was before the first iteration of the project. It was vastly over-scoped.



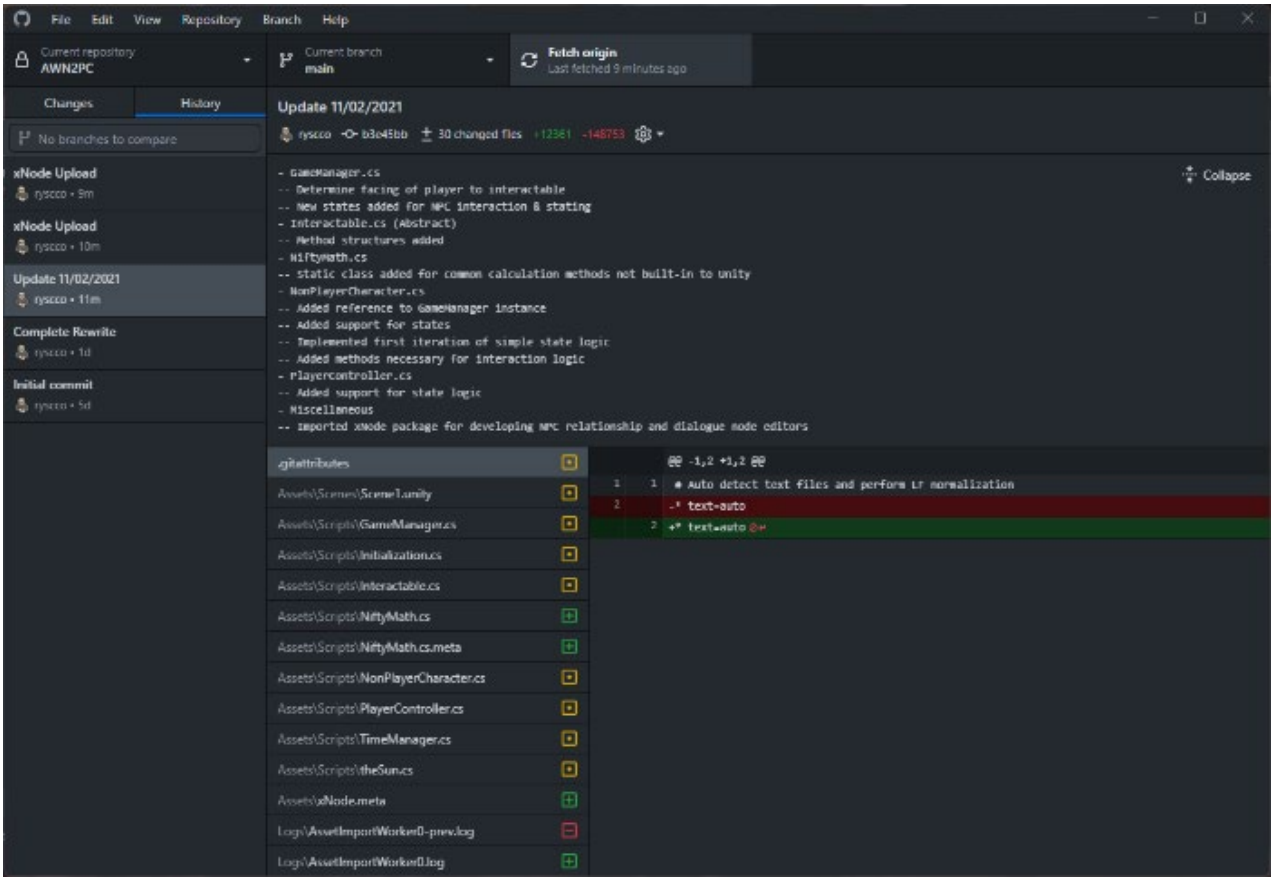
The initial project, player movement was implemented in a way that the player could move relative to the camera view rather than the world space. With a fixed camera angle, this is more intuitive than world-space movement when the camera transform is not parallel or perpendicular to the worlds-space X or Z axes.



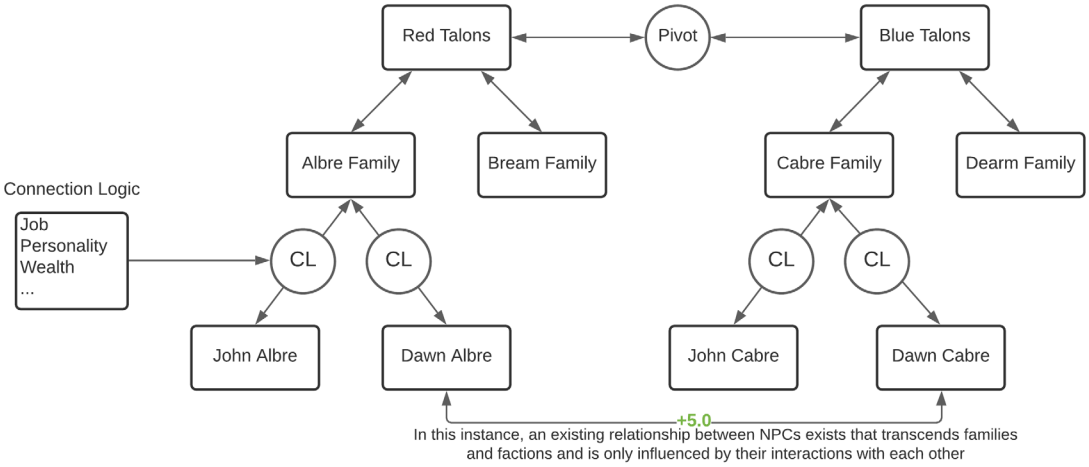
NPCs can sense the proximity of the player. This is integral to allowing the player to interact with NPCs as well as objects in the world.

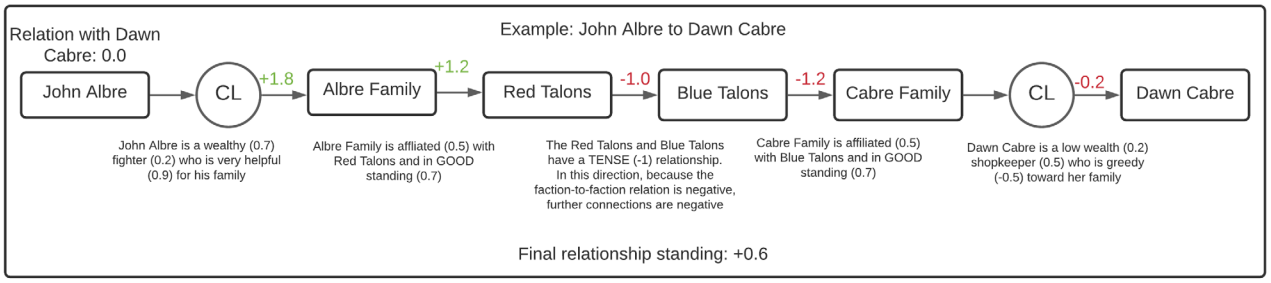


This is the initial commit after scrapping the original project and starting from scratch. The previous version of the project became very difficult to work within as there was no central pattern to the structure. In this iteration, the Singleton pattern is prominent and used to handle multiple world-spanning systems.



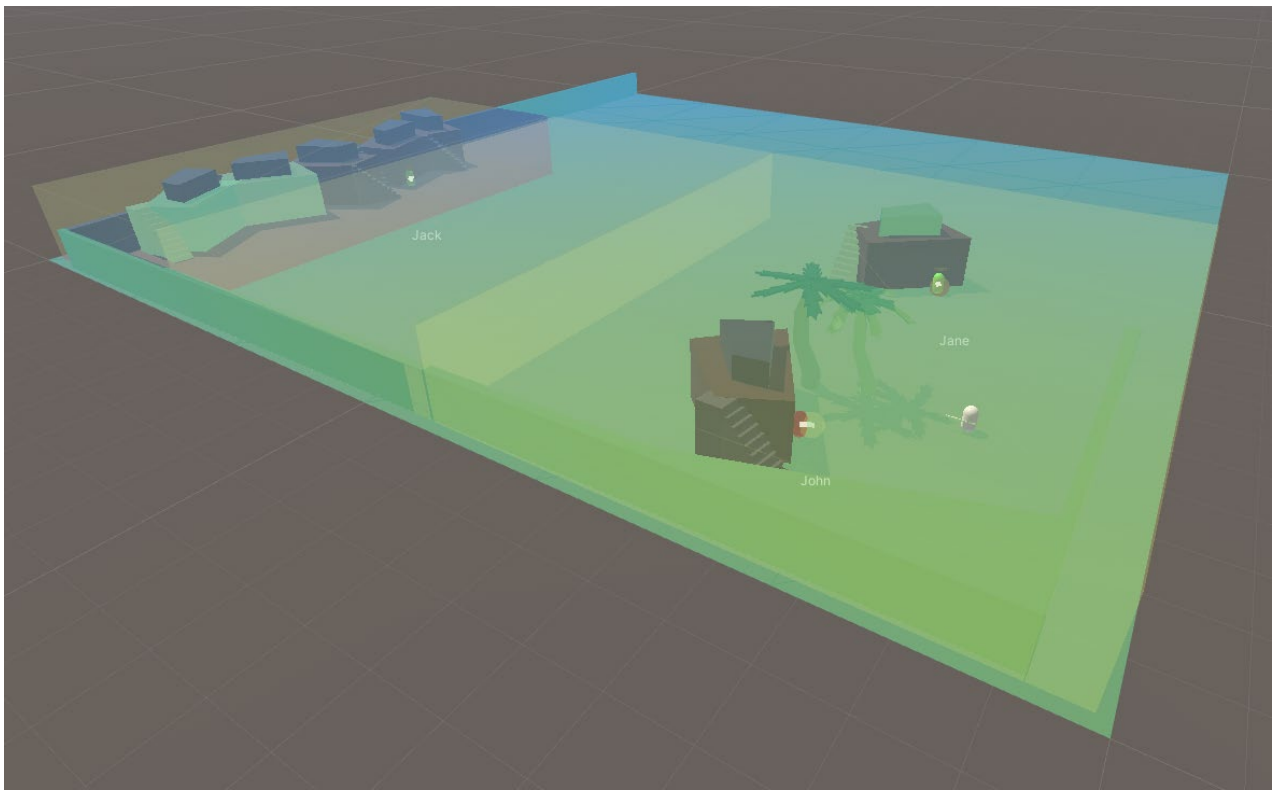
Significant progress on state logic, the player's movement is now locked when they are interacting with Interactables, and the NPC state logic is starting to take form that will allow them to follow a schedule.





The relationship between John Albre and Dawn Cabre is based only on their relationship *through* their families and factions since they have no pre-existing relationship. Interaction between the two will further shape their relationship.

This is the basic model I developed for NPC relationships, with an example. The structure is a tree where the connections have values. A character can then be traced to any other character to determine a baseline relationship. Each connection can also change based on interaction in the world, etc. which will then change the NPC-to-NPC base relationships as well. NPCs can also have direct connections outside the hierarchy (such as two people from feuding families that really value their relationship). Connection Logic is a set of factors that influence the character's standing in their family. The pivot is a point where values on the other side of the tree become positive or negative depending on the faction interactions.



This is a visual representation of the colliders I am using to set camera views. It's a little clunky to setup, and surely, I could build a tool in the future to make setting it up much easier.

The two images above are progressive work on the environment.

Above it is shown that this building changes when Jane's family changes. This is because that building being either a blacksmith or tavern is based on the relationship between Jane and John. Initially they are in different families that have a tense relationship. Then when bringing them into the same family, the building available changes.

```
public float GetRelationshipValue(GameObject a, GameObject b)
{
    NonPlayerCharacter npc1 = a.GetComponent<NonPlayerCharacter>();
    NonPlayerCharacter npc2 = b.GetComponent<NonPlayerCharacter>();
    float runningTallyA = 0f;
    float runningTallyB = 0f;
    float total = 0f;
    runningTallyA += GetFamilyValues(a);
    runningTallyB += GetFamilyValues(b);
    if (npc1.family == npc2.family)
    {
        //Debug.Log(runningTallyA + runningTallyB);
        total = runningTallyA + runningTallyB;
        return runningTallyA + runningTallyB;
    }
    runningTallyA += GetFactionValues(a);
    runningTallyB += GetFactionValues(b);
    if (npc1.family.inFaction == npc2.family.inFaction)
    {
        //Debug.Log(runningTallyA + runningTallyB);
        total = runningTallyA + runningTallyB;
        return runningTallyA + runningTallyB;
    }
    for (int i = 0; i < allFactions.Length; i++)
    {
        if (npc1.family.inFaction == allFactions[i])
        {
            for (int j = 0; j < npc1.family.inFaction.factionStandings.Length; j++)
            {
                if (npc1.family.inFaction.factionStandings[j].faction == npc2.family.inFaction)
                {
                    total = (runningTallyA + runningTallyB) *
                        npc1.family.inFaction.factionStandings[j].interFactionStanding;
                }
            }
        }
    }
    return total;
}
```

This code block is the core of the inter-character relationships. It traverses the data structure I've implemented that is currently three layers: character, family, and faction. It can be further expanded to include more layers or slightly modified/overloaded to operate on separate relationship tree structures.

```
#region SCHEDULE
1 reference
private Queue<ScheduleNode> makeSchedule()
{
    Queue<ScheduleNode> q = new Queue<ScheduleNode>();
    foreach (ScheduleNode s in scheduleNodes)
    {
        q.Enqueue(s);
    }
    return q;
}
1 reference
private void handleSchedule()
{
    if (_timeManager.TimeHour == schedule.Peek().hourToLeave)
    {
        advanceSchedule();
    }
}
2 references
private void advanceSchedule()
{
    ScheduleNode s = schedule.Dequeue();
    schedule.Enqueue(s);
    _agent.SetDestination(s.worldPosition);
    npcState = NPCState.MOVINGTONODE;
}
#endregion SCHEDULE
```

This code block is the NPC scheduling. NPCs can move between points in the world at specified times to make it appear that the NPCs are “living their life”.

```
private void SetStoryFile()
{
    foreach (NPC_Story n in story)
    {
        if (WantsItem && n.giftOrItemTriggerIndex > -1 && _haveWantedItem != null)
        {
            storyController.SetStoryFile(n.story);
            if (wanteds.Count > 0)
            {
                storyController.story.BindExternalFunction("receiveItem", (string i) =>
                {
                    ReceiveItem(i);
                    Debug.Log("ReceiveItem called");
                });
                return;
            }
            else return;
        }
        else if (n.giftOrItemTriggerIndex == -1)
        {
            storyController.SetStoryFile(n.story);
            return;
        }
    }
}
```

This code block is a triumph of a struggle to learn to work with the Ink API. This allows me to check that a character desires a certain item or trigger and have the story file trigger a C# function binding.