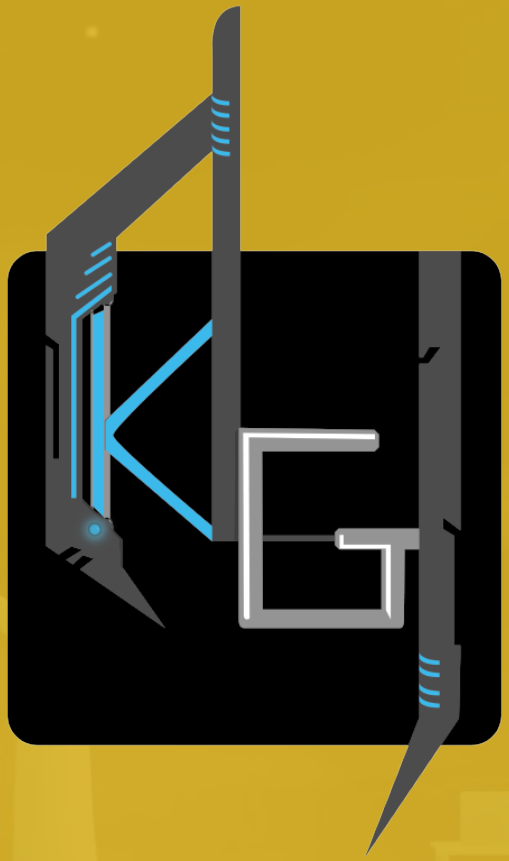


Kevin H Gordon

Process Book

Project Title : Homebound

Table of Contents

Bookmarks

Bookmark Recommendation

Section 1 - pg. 4

Section 2 - pg. 10

Section 3 - pg. 17

Section 4 - pg. 25

The Project

What is Homebound?

Homebound was a project I created with a team for Global Game Jam 2019. I'd been a part of several game jams up to that point as an artist, but this time I wanted to challenge myself by dedicating my time only towards technical and system design. For this project, I served the role of lead programmer in charge of developing system mechanics. The project for this Global Game Jam had to be completed in 36 hours, so there was a lot of ground to cover.

The piece served as a purely technical Unreal Engine 4 project, as such I refrained from 3D modeling any assets. Main goal was making sure the game was functional, and that all game mechanics worked as expected. In a deconstruction, one would find that this piece is composed of three core functionalities. They are as follows:

- Player Follow System
- Rope Snap System
- Rope Pull System

Though there were many hardships when it came to programming this piece, at the end of the day the project was a success. All of the functionality worked as expected and the game was effectively playable.

In this book, I will break down the journey of this project, and show what went into it at each stage.

Game Jam Theme

What home means to you?

This question was the general theme of 2019's Global Game Jam. When we first got this theme, we knew we had to narrow the theme further. As such, the team and I first brainstorm intensively about what home meant for us personally. While there were many ideas through around during that time, the one idea that resonated the most with us was clear. Home can be anywhere, so long as you're with the one's that you love.

From that line, a clear theme and goal was formed within us. We knew that no matter what this game would feature 2 characters working together to find home. With that established, the next step was ironing out character details. After debating on different character ideas, an idea was brought up about having a dog be a playable character. Dogs love their human companion unconditionally, and we'd all seen instances of dogs sticking by their partner no matter the circumstance. We all resonated with this idea well and the characters were solidified.

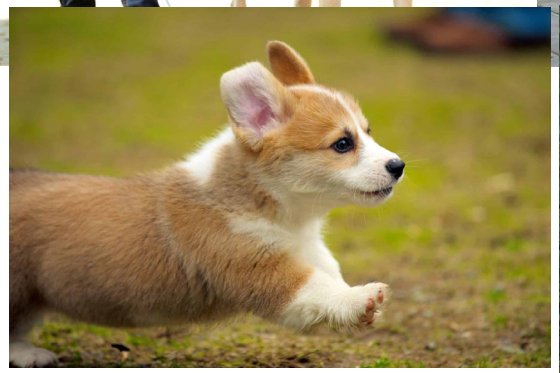
Gameplay Elements

With a theme solidified and characters established, the only thing left was to iron out the visuals and game play style. As our team had a wide range of artistic skills, from 2D illustrators to 3D artists, we wanted to make a product that perfectly blended the skill set of all team members. We brainstormed a couple of different solutions internally, but ultimately settled on creating a 2.5D game. That decision was based on the desire to utilize the 2D character illustration skills of our members, while relying on 3D art for background elements.

The game play style was a relatively easier process to narrow down. As we were creating a 2.5D game, we looked to past 2D platformer games for inspiration. Since the theme was connected to “home” we thought it would be appropriate to have the player always working to guide their friend towards home. We decided to make the game a puzzle platformer, where the dog player has to guide his human home.

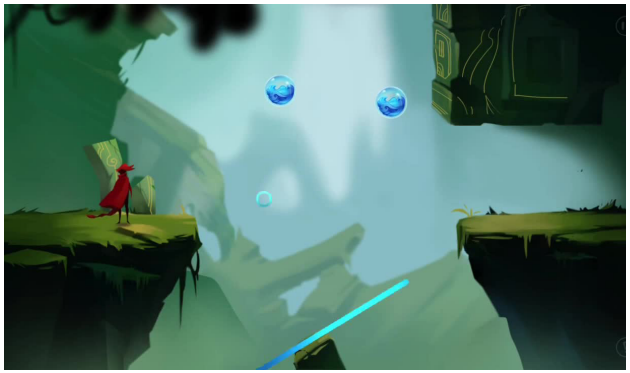
Once all of the initial project elements were established, it was settled. The final product would be a 2.5D puzzle game that’s focus was a dob bringing it’s friend home. The last thing was to find more inspirational images to support our piece.

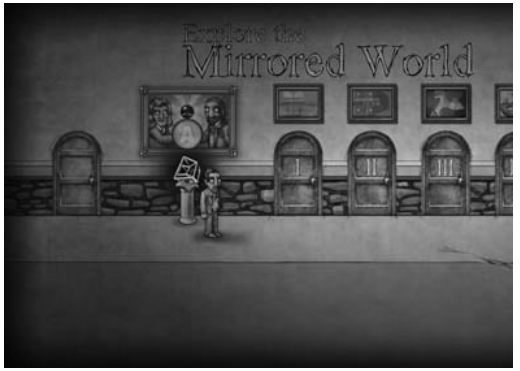
Man’s Best Friend

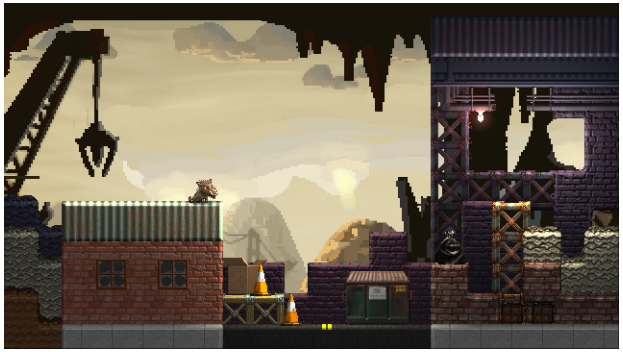


2.5D Platformer Reference

2.5D Platformer Reference







Movement Systems

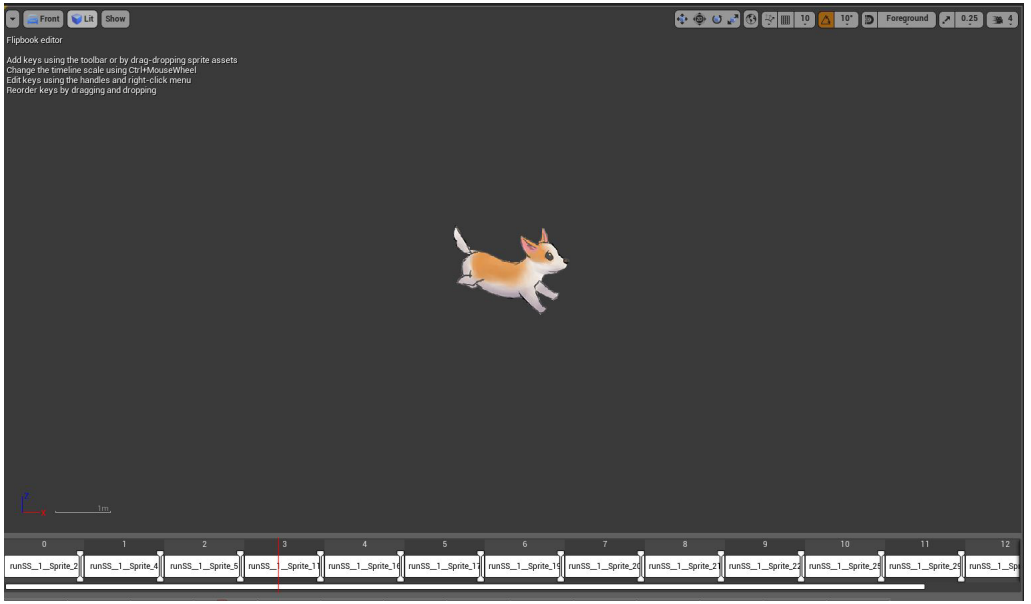
Implementing 2D in Unreal

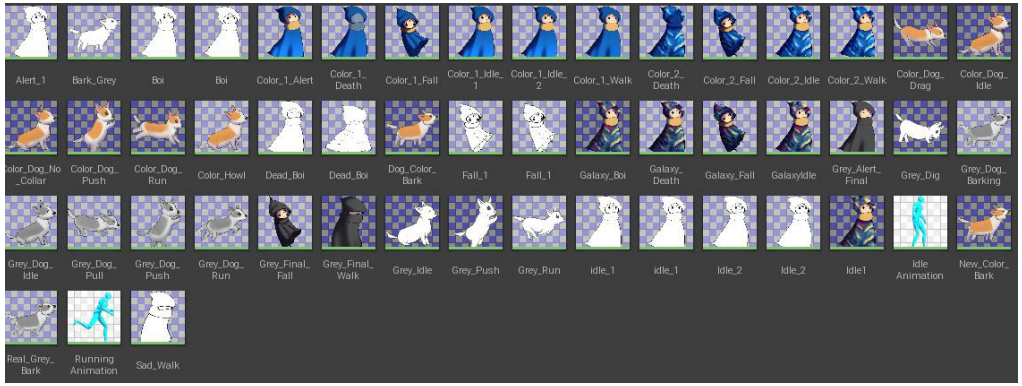
As I was charged with developing the core systems for the game, my first hurdle to overcome was getting 2D characters to work within Unreal Engine. Now Unreal Engine is a robust game engine that is known for creating the cutting edge 3D graphics featured in many AAA games. What it is not known for, is it's 2D capabilities. As such, many of the documentation and help community resources that I'd come to rely on when scripting systems was arguably limited.

Thankfully after a bit of research I was able to find Unreal Engine's documentation on their 2D flipbook system. I'd utilize this documentation extensively for the duration of this project.

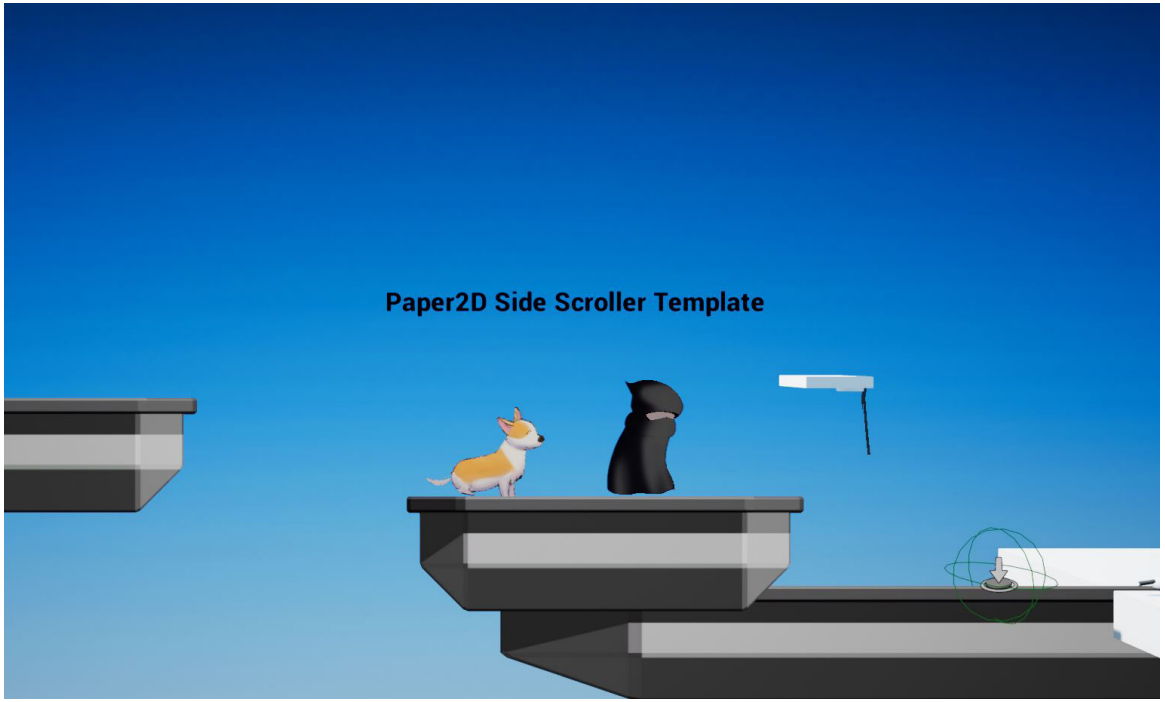
2.5D Character Scripting

The first step in creating this game was to get the character working inside of the engine. As alluded to above, this was done using the Flipbook system in Unreal Engine. The process was relatively straightforward, and consistent with how 2D characters are created in other engines. Once I received the sprite sheet from the character illustrator, those sheets were imported into the engine. From there, I used the sprite cutter tool inside of Unreal to splice the images into sections. Flipbook's timeline was used to set the rate of each animation. Images of this process, and the final result in-engine featured.





UE4 Flipbook Scripting



Once the base Flipbook was created for each character, I imported them into a base playground scene (featured to the left). This scene is where all systems testing took place.

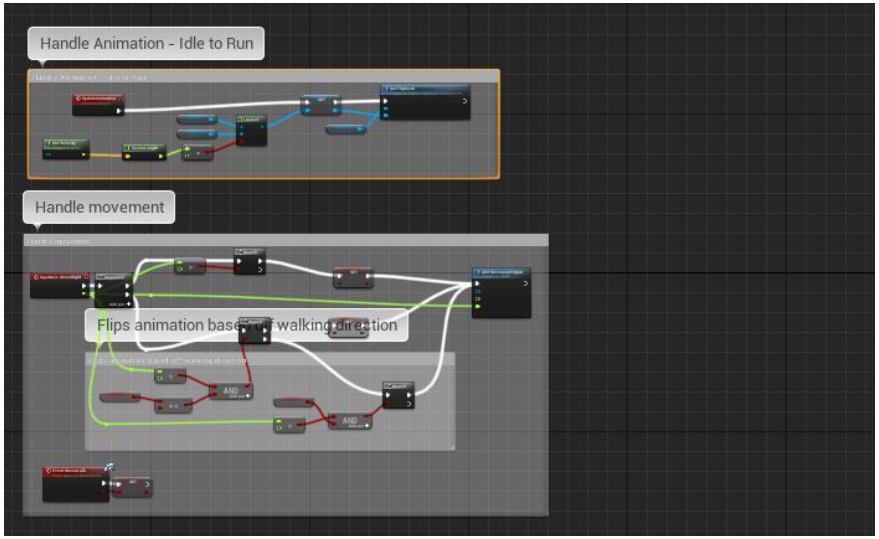
With the two characters imported, the next step in this Jam process was in implementing the character movement system.

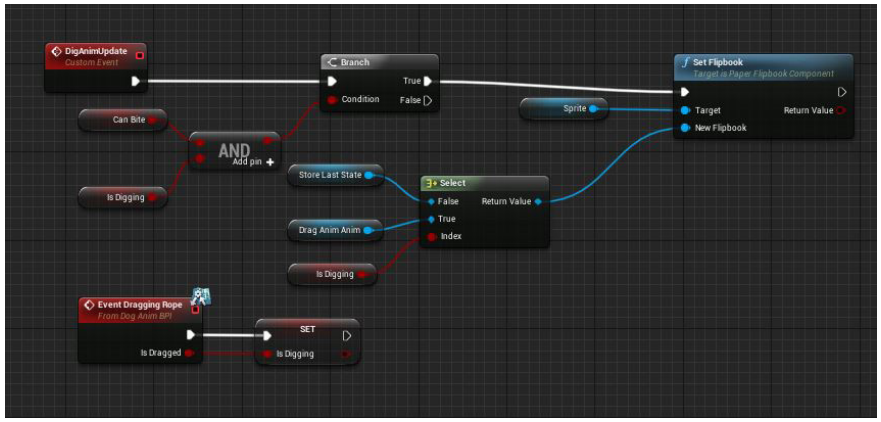
Dog Player Movement

As there's two character in this game, there needed to be two unique systems of movement for each character. To start this process, I got began working on the base run/walk movement of the dog character. Getting the character to move to the right was simple, as I was able to reuse code from UE4's flipbook blueprint. The trick came when getting the character to move to the left.

Given the original animation has the character facing the left, there needed to be a way to flip the animation when moving in different directions. Additionally the need to save animation states was a must, as each character has several different animation choices. To solve this, a script was first created to flip the character's animation when changing directions. Once the movement script was handled, several event functions were created the store the different animation states of the player. This all culminated in a fairly robust and modular system capable of animating and moving multiple character actions.

Snapshots of the system code to the right



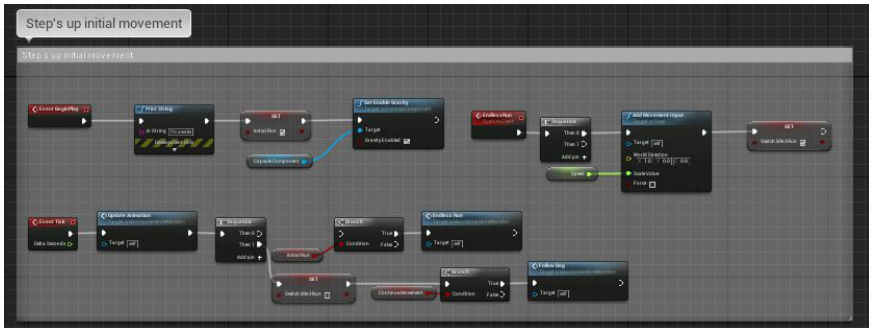


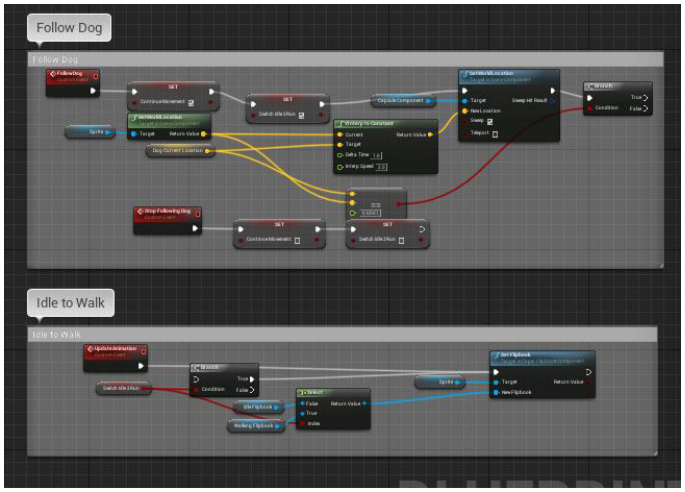
Human AI Movement

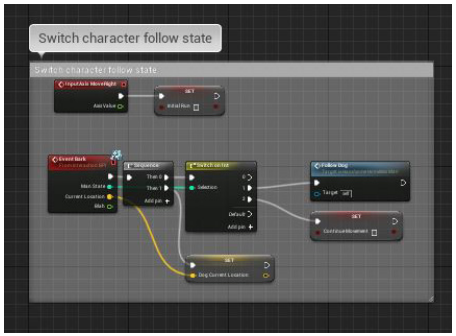
With the player movement successful created, I worked on tackling the script of the human AI. The script was rather multifaceted, and required several different layers in order to get it working. The good thing was many of the same programming methods utilized in the player movement script could be reused here.

When the game first starts, we wanted the AI to continuously move to the right. It's only when the dog calls the Human character over, that the character stops the continuous movement and follows the player. As such, we step up the script to tackle just that. The image featured on top purely handle setting up the initial run script. Whereas the middle image shows the function required to get the AI to follow the dog.

Since the two character are co-dependent on each other, an interface node was created to tie the two movement scripts together. This way, when the dog player barks, the human AI knows to go to the dog's current position.

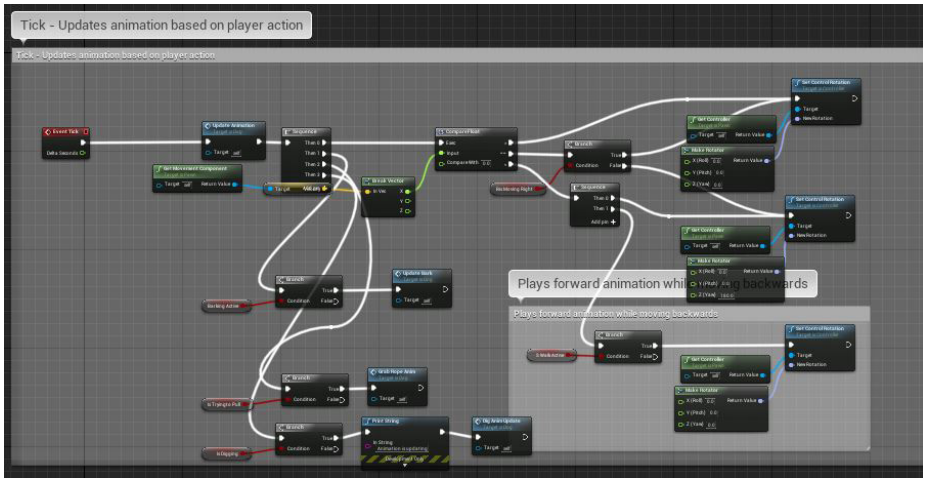






Blending Movement

With both the player and character movement scripts completed, all that's left was making the two scripts talk to each other. As discussed in the AI script, a interface node was set up so the player and AI scripts can talk to each other. In addition, the animation for both characters is set to change depending on what action the player was currently performing. All of this culminates in a fairly robust player movement system for players to utilize.



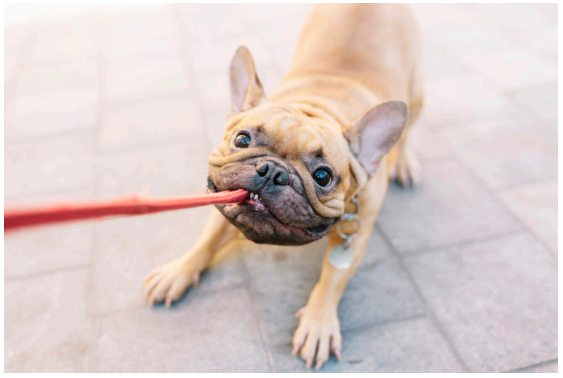
Core Gameplay Systems

Intro to the Rope System

Once the player and AI movement scripts were created, the next task was to work on the first major game play system. That system was decidedly the Rope Snap system. First, a bit of pretense before getting into the system itself. As we knew this to be a puzzle game based around the theme “home”, our plan was to have the player guide the human AI home.

We knew there needed to be obstacles standing in the way of the player, but the question was what those obstacles could be. As a result of our dog player’s small stature, we knew that moving traditional large objects was out of the question. From brainstorming, we can up with the idea of the player pulling rope as a core mechanic. As dogs often drag and pull objects in real life, we felt this idea was a safe bet to build systems around. As such, the rope system was formed.

The idea is that the player has to pull and drag objects in order to progress through the level.

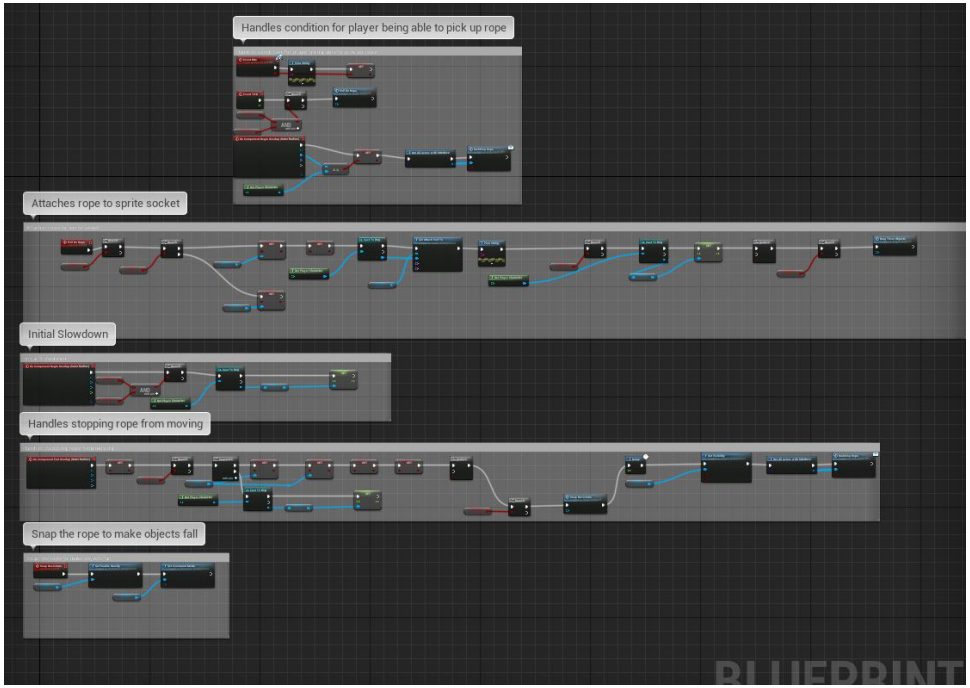


Rope Pull System

As the core system mechanic was established, it was time to begin working on the first system, pulling the rope. Getting this system working was no easy task, and was arguably the most complex aspect of the entire project. As I had no idea how to begin this system, I first started this project by mapping out a flowchart for how this script should work. From that mapping, I determined that for this script to function, 3 components were required.

- A rope, or wire, that could be simulated in engine
- A collidable sphere at the beginning of the rope for the player to grab
- An outer sphere for determining when the player has exceeded the rope's length. (enabling the next action to occur)

Only thing left to figure out was how to make a rope in Unreal. Luckily Unreal has a cable component that fit the bill. With the core components of the script established, I was able to generate a clear path forward.



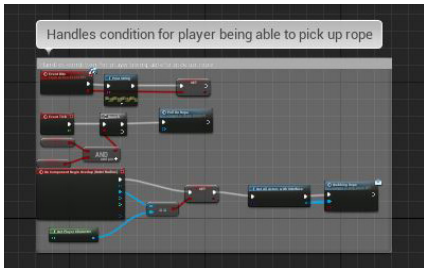
Scripting the Pull System

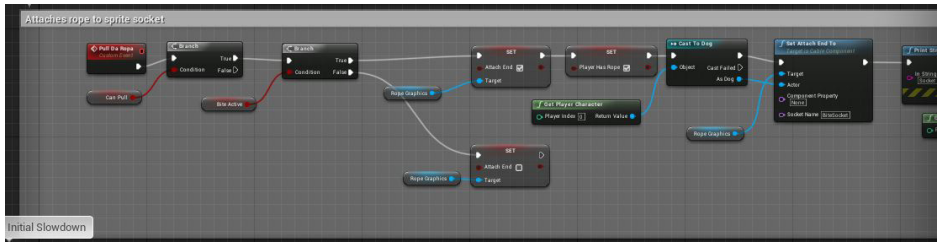
With all of the initial planning done, it was time to script this system in the game. I started out by creating the core components, and included the colliders for the object as well as the physics rope. There is a setting within the cable component that allows part of the cable to be rigid. I set the end of the rope to alway start rigid, while the beginning of the rope was affected by gravity.

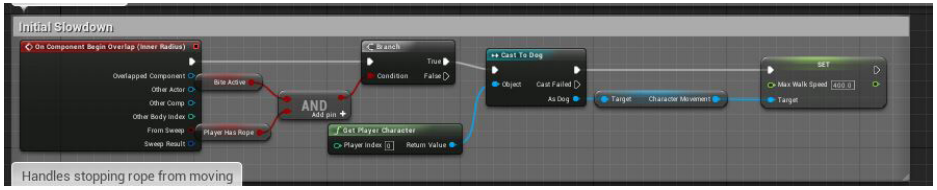
When scripting the rope mechanics, I set the inner collider to check whether the player was grabbing the rope. If the player was, the rope would then attached to the player and change it's speed to move slower. As the player moved further out and into the outer collider, the speed would change again to move even slower. This was done to simulate the real-world increased resistance that's felt when pulling objects.

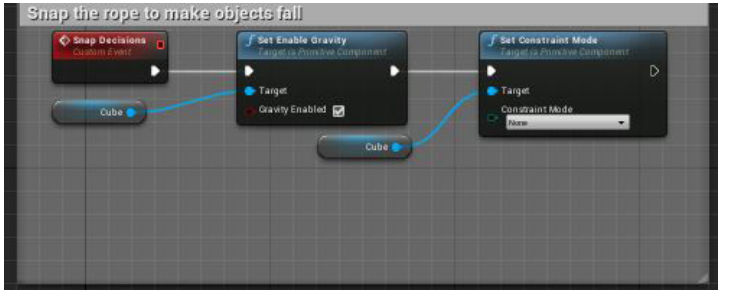
The finale of this system come when the player finally exits both colliders. The end of the rope that was initially rigid becomes dynamic, moving whatever object it was attached to.

Snippets of this process featured to the right.









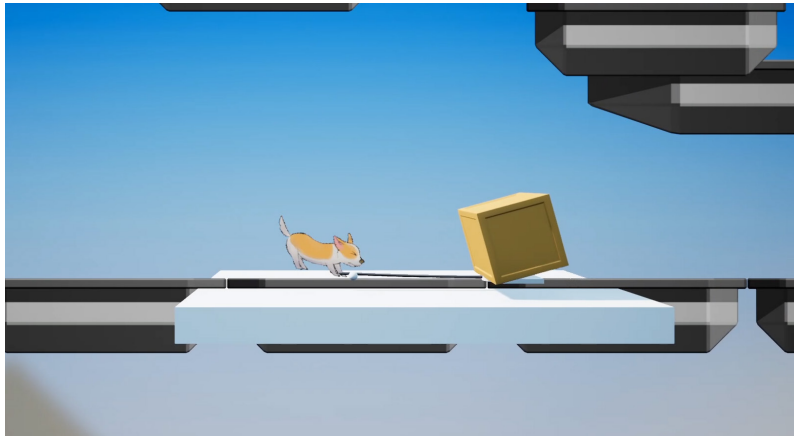
Gameplay Examples

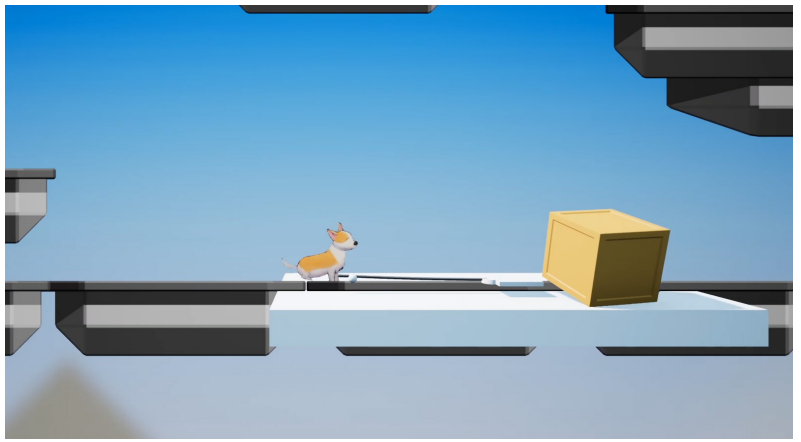
Rope Drag System

Since the pull system had been created and fully implemented, it was time to work on the final system. This system was getting the player to drag object using a rope, or simply the Rope Drag system.

As the first rope system was designed so robustly, I was able to trickle down the scripts in that system to this new one. Most of the code for the rope drag system is identical to the pulley system. The only different is in how the cable attaches to the end object. In the first system, the rope simply tore off after the player got too far. This time around, the end of the rope was to remain rigid. The “draggable” platform was then attached as a constraint to the end of the rope. This allowed the platform to be dragged with the rope.

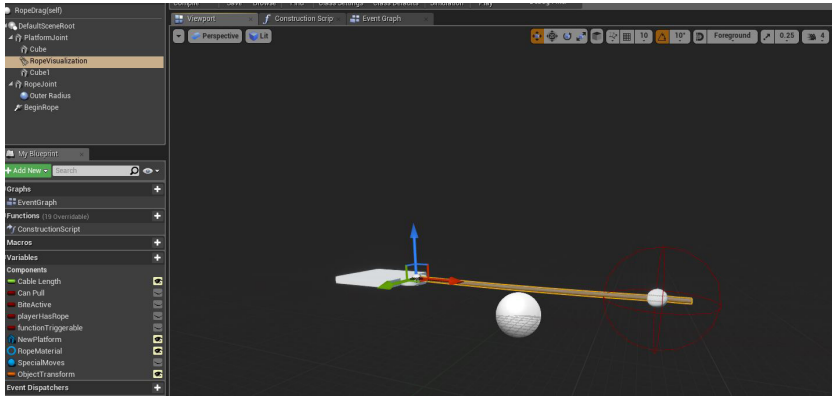
Images of this in action featured to the right.

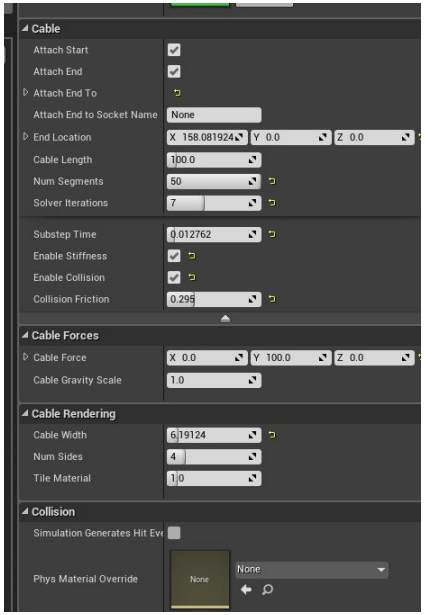




Rope Drag System

Most of the base remained the same for this piece of functionality. One major item that was added however was the inclusion of a platform object. As stated before, this platform is attached to the rope at all times. Which makes it ideal for dragging object. All other scripts remained largely the same. With this system complete, all of the major game play functionality was finished.





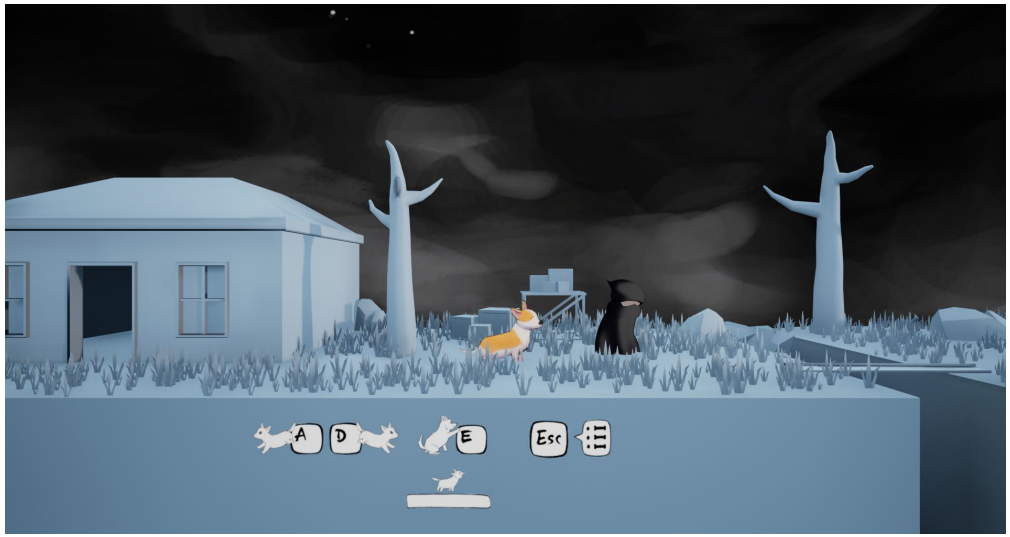
Final Thoughts

This was definitively a challenging project to create. To create a full game in only 36 hours is no easy feat, without the help of the team this could've never been completed. This project not only pushed my game development in Unreal Engine, but also made me a more polished programmer as a result. Never knew that 2.5D game development was even a possibility in Unreal Engine until this project came around.

Though there are many technical systems that I listed in this documentation, that only scratches the surface of the programming that went into this game. Some systems, such as the respawn system, were added to the game but not showcased, as to devote more time to the core aspects of the game.

In-game screenshots featured to the right -->

Final Renders



Special Thanks

Special thanks to all of my friends, family and professors who’ve supported me throughout the years. Without them, none of the work featured in this book would be possible!


HOMEBOUND