Kronos

A Virtual Reality Roguelike

Reily Stanford
Union City, Tennessee
rstanfo1@utm.edu

Enrique Tejeda
Martin, Tennessee
enrgteje@ut.utm.edu

ABSTRACT

Kronos is a virtual reality video game with roguelike elements in which time is falling apart at the seams. It falls on the protagonist, Kronos, to restore the timeline to its chronological order. Along the way, the hero is met with obstacles to overcome to achieve this goal. Kronos has pixel-art 2D sprites and textures in a 3D world that is inspired by games like DOOM (1993) and Paranautical Activity. The game is a first-person shooter where you are traveling between a mismatch of time periods in order to restore the original order of time. Kronos is built using Unreal Engine 5 and a Procedural Dungeon Plugin created by BenPyton. The behaviors of the enemies within the game, the item effects, and the weapon mechanics are created using Unreal Engine's Blueprint system, which is a visual scripting language. Each floor is set in a different time period with various elements about each period feeling out of place. This can be seen in the enemies and objects within the level.

1. INTRODUCTION

Kronos is a virtual reality game that is set in the first person. The art style is similar to DOOM (1993) in that the enemies are 2D pixel-art sprites that always face you. The game is a shooter where you are tasked to bring the timeline to its original state. Someone has disrupted the chronological order of events leading Kronos, the protagonist, to fix the timeline by any means possible. There are two levels that each correspond to a time period where you can easily tell that time has been altered. This is apparent when you see the environment that does not match the current time period the level is set in.

2. MOTIVATION

3. STORY

4. ART STYLE

5. MECHANICS

A primary mechanic that the game offers is procedural generation of level layouts. The game offers three methods of movement: head-based locomotion and hand-based locomotion. The locomotion moves like a standard video game in which you hold the movement button to move in your desired direction. Head-based locomotion uses the direction of where the player is facing in order to determine the direction of movement, while hand-based locomotion uses the direction of the left hand. Each level is concluded by defeating a boss enemy that is stronger than the enemies you have been fighting on the current floor. There will be multiple weapons, items that affect your player's stats, and a variety of enemies on each floor.

6. TECHNICAL SPECIFICATIONS

The project is built with Unreal Engine 5 [5] and we used Unreal's Blueprints, a visual programming language, to create the logic of the game. We use a plugin called Procedural Dungeon Plugin [2] that places the rooms into a playable layout. To plan our roadmap we used Milanote [7]. The assets we have in our project come from Humble Bundle [3] and itch.io [6]. As for source control, we used Git LFS [4], which is used for large files on github, and the built-in source control for Unreal Engine.

7. PROJECT DEVELOPMENT

Throughout the development of Kronos, we have split responsibilities and tasks so that we were not both working on the same feature to maximize efficiency. This has led to the both of us having different experiences when creating our game.

Reily's Experiences

One of the first things that I started messing with in Unreal Engine was adding models and proper collision to those models in the built-in example world in the engine. I quickly learned the basics for adding props to the world. I then decided to tackle my first real addition to our game, locomotion movement.

Locomotion Movement

Unreal Engine's built-in VRPawn object is created with a teleportation-style movement instead of the smooth movement you would expect from a standard 3D video game. Adding smooth locomotion to the game allowed me to learn a lot about the Blueprints scripting language. I was also able to learn about events, adding different elements to the viewport that aid with player location detection that aren't visible to the player, and different custom collision descriptions. In the process of adding this, I ran into one major problem, collision. The default collision provided by Unreal led to the player being moved while holding an item due to the addition of smooth locomotion. Once the collision had been sorted, it was time to move to the next feature, enemies.

Enemies

Adding enemies allowed for me work with sprites and flipbooks to make an animation for our 2D enemies. It also gave me a chance to experiment with behavior trees to allow for the AI to function on its own. Prior to the decision to use behavior trees, I was creating the behavior entirely through Blueprints. This caused a lot of issues, especially when it came to handling animations. Occasionally, the enemy would change their sprite while another event would occur, stopping the enemy's sprite from reflecting their current action.

AI

Working on the AI was where a majority of my time was spent working on this project. I had to learn about the ins and outs of Behavior Trees in Unreal Engine. I started out by making an AI that could move randomly in a specified radius. This little introduction allowed me to learn about the little pieces that are needed to make an AI in Unreal. I had to learn about the Blackboard, which is an object that holds all the values, in variables, that a single instance of an AI needs to function. I also had to learn about AI Controllers,

which is where you determine the Behavior Tree that needs to be run. It allowed me to attach senses to the enemies, so they could visually analyze their surroundings to look for the player. AI Controllers can also talk to the Blackboard to change values based on these senses that may be helpful for AI behaviors. Finally, we have the Behavior Trees. These look just like what you would see with regular trees in computer science. When working with the actual Behavior Trees, there are three main elements that I used: selectors, sequences, and tasks. Selectors run the first child node that does not fail to run properly. Sequences run each child node from left to right until one of them fails or all children have been executed, which stops the sequence and returns to its parent node. Tasks are the actual functionality of the AI. This includes things like attacking, finding the player location, moving, and more. All these pieces combine to make an AI that can make informed decisions based on its senses. Alongside enemies and AI, I worked on the creation of items.

Items

In order to create items, I had to think about all the pieces to make a working item. This included the objects that the player can collect, the spawning mechanism for the items, and the 2D sprites for the items. This allowed for me to learn about spawning objects in the game through the blueprints system. Not only that, but I was also able to learn about the randomness that Unreal can provide for random selection.

Enrique's Experiences

When we first decided to create a VR game, I immediately began researching on how to import and use a low-poly weapon. It was overwhelming at first because I had never developed with any game engine, but it got easier as I read Epic's documentation on the subject. The hardest part of getting a weapon to work was finding a decent muzzle fire animation. I stopped creating weapons as soon as I built one so that I could focus on world design.

Procedural Generation

In order to make a rogue-like game, we needed Kronos to have procedural generation. We both wanted the generation of levels to be similar to how The Binding of Isaac's system works. This type of generation places levels together randomly until a valid dungeon is created. A valid dungeon is when there is a start level, an end level, and a number of levels that connect the start and end level together. Instead of creating a system that places the levels for us, I decided to search if Unreal had a built-in system for this. Unreal did not have a free plugin for what we needed, but I did find a free plugin on YouTube called Procedural Dungeon Plugin by BenPyton. This plugin had exactly what we were looking for. The plugin allowed you to set the minimum and maximum amount of levels between the start and finish since you would want to have a cap on the amount of rooms on a level, otherwise the plugin would be adding levels forever. Because this plugin is not provided by Unreal, it was difficult to get it working with our project since the plugin was built using a third person character, while Kronos uses a VR character. This was problematic since it was difficult trying to find out how to teleport the player to the generated level. In addition, the generated level were only showing the first two rooms so I spent hours trying to troubleshoot what was going wrong on my end. I checked the documentation, the Blueprint I modified, and the example game the plugin provides to compare Blueprints to see why I could only see the first two rooms. A day goes by and I decide to check the plugin's issues tab on GitHub and found a ticket that had a similar problem. The author of the plugin resolved their issue and I was able to fix mine by disabling a single setting called "Occlusion Culling" in the plugin's settings.

Weapons... Again

After creating two levels, I decided it was time to change my focus back to weapons. I had a lot of issues trying to allow the player to grab a weapon with both hands so we decided to only have one-handed weapons. This decision was made for the sake of time and trying to get the game to a playable state first.

Audio

Importing and adding music to our Unreal project was surprisingly easy. We decided that we wanted a retro tone to our game so we used 16-bit music pack from Humble Bundle [3]. This was used for the background music to our levels. We also added footstep and gunshot sounds that also were from Humble [3]. We were unable to find an audio clip that could be used for our player taking damage so we did not incorporate it into our game.

Creating Sprites

Because we were unable to find any front-facing sprites on itch.io [6] or anywhere else, I created five of the six enemy sprites found in the game.

8. TRIALS AND TRIBULATIONS

9. FUTURE WORK

REFERENCES

1. Adobe. Mixamo. https://www.mixamo.com/, 2022.
2. BenPyton. Procedural dungeon plugin. https://github.com/BenPyton/ProceduralDungeon, 2019.
3. Humble Bundle. Sfx and music for your games. https://www.humblebundle.com/software/sfx-and-music-for-your-games-software, 2022.
4. GitHub. Git large file storage. https://git-lfs.github.com/, 2014.
5. Epic Games Inc. The Unreal Engine. https://www.unrealengine.com/en-US, 2022.
6. itch.io. itch.io. https://itch.io/, 2022.
7. Milanote. Milanote. https://milanote.com, 2019.