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)*[1] in that the enemies are 2D pixel-art sprites that always face you. We also took inspiration from *Paranautical Activity* for our world crafting throughout the game. Since our game was made using a modern engine for modern hardware, our visual style was an aesthetic choice. To best capture this, we had to implement code that made the enemy sprites always face the player. We also tried to use low-poly models in our world to further capture the aesthetic we wanted. 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

For most of our lives, we have played video games for entertainment; socializing; and overall, passing the time. We have spent all this time playing games, but we never stopped to think about the process of actually creating them. This was part of our motivation when it came to making Kronos. We wanted to try our hands at game creation to say that we tried and learn to appreciate the little things that developers had to go through when creating our favorite games. Along with this dedication to games, we have both had VR headsets for over two years now. In this time, we have not been able to put them to much use, so we decided to create a game where we could use them. Finally, we had to figure out what genre of game we wanted to develop. Since we didn't have much time, we began looking into roguelikes, a genre that would allow us to spend more time working on mechanics than the world crafting. On top of this, it would provide the ability for the player to replay the game without it feeling repetitive or stale too quick. For inspiration, we looked to a game we had played in the past called, *Paranautical Activity*. *Paranautical Activity* is a roguelike that embraces a voxel-based aesthetic, in which the objects in the game are created out of a combination of small cubes.

2.1 The Influence of DOOM and Paranautical Activity

While we wanted to capture the feel of *DOOM (1993)*, we felt that crafting the world in the same light that *DOOM* did would fall short. We believed this because we wanted to have props and other objects in the world to obstruct the player and provide cover. This could have been achieved the way *DOOM* did with simple objects like boxes, but we wanted a more complex environment. This is where we took some inspiration from *Paranautical Activity* and decided to create a more low-poly world. This is how *Paranautical Activity* and *DOOM* were the key factors in our decision to embrace a retro feel.

3. STORY

The story of the game is that a large corporation has broken the rules of time and has resulted in the timeline falling apart. The corporation behind the incident specializes in time travel, their business allows people to time travel without altering any events to prevent ripples in time. The company has secretly been selling artifacts from the past on the black-market, which led to this catastrophic event since their actions snowballed out of control thanks to the butterfly effect. In response to the timeline collapsing, the protagonist of the game, Kronos, must restore it to its original state. Kronos is an intergalactic space cop tasked with resolving this issue by any means necessary. The player can directly see the effects of time distortion in each floor of the game.

Figure 1. In-game representation of time distortion.

4. ART STYLE

When creating our game, we wanted to try making something that felt similar to *DOOM (1993)*, *Paranautical Activity*, and

[1] *DOOM*'s aesthetic was not necessarily due to a stylistic approach, but was instead necessitated due to a lack of hardware capabilites for computers at the time.

other retro feeling games. This led to it featuring a mashup of a 2D and 3D art style. The world itself is crafting using 3D objects, but the enemies and items are represented as hand-crafted 2D sprites that always face the player. This creates an atmosphere like none other than we have experienced in a VR game. While our sprites don't encompass what is seen in games like *DOOM*, where the enemy sprites actually turn away from the player based on their movement, we feel that it created a different style that feels like an homage rather than a copy.

Figure 2. 2D enemy and item sprites placed in the 3D world.

Creating Sprites

Because we were unable to find any front-facing sprites on itch.io [5] or anywhere else, we created five of the six enemy sprites found in the game. Tackling this issue was somewhat difficult since we are not artists or familiar with making pixel art. Our limited experience is evident when looking at the sprite and seeing choppy animation work. This is because the enemy sprites do not have many frames for each animation. To have a smoother animation would mean creating more sprites for the specific animation.

Figure 3. All of the enemy sprites that were handmade.

5. TECHNICAL SPECIFICATIONS

The project is built with Unreal Engine 5 [4] 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 [1] that places the rooms into a playable layout. To plan our roadmap we used Milanote [6]. The assets we have in our project come from Humble Bundle [2] and itch.io [5]. As for source control, we used Git LFS [3], which is used for large files on github, and the built-in source control for Unreal Engine.

6. OVERVIEW OF MECHANICS

From the start of our project's lifespan, we wanted procedural generation in our game because we wanted to construct a VR roguelike video game. There are several different ways of handling procedural generation and we chose the procedural generation of level layouts. Kronos offers two 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. The enemies' behavior (AI) was implemented using Unreal's Blueprints. Similar to most video games, each level is concluded by defeating a boss enemy that is stronger than the enemies you have been fighting on the current floor. There are multiple weapons, items that affect your player's stats, background audio, and a variety of enemies on each floor.

7. EXPERIENCES

Weapons

When we first decided to create a VR game, research was immediately started on how to import and use a low-poly weapon. It was overwhelming at first because we had never developed with any game engine, but it became easier as we read Epic's documentation on the subject. The most difficult part of getting a weapon to work was finding a decent muzzle fire animation. The focus of our project changed immediately to prioritize building the base mechanics first like movement to ensure we have a playable game.

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 us to learn a lot about the Blueprints scripting language. We were also able to learn about events, adding different elements to the viewport that aid with player location detection that are not visible to the player, and different custom collision descriptions. In the process of adding this, we 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.

Procedural Generation

In order to make a rogue-like game, Kronos needed 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, we decided to search if Unreal had a built-in system for this. Unreal did not have a free plugin available, but there is a free plugin on GitHub called Procedural Dungeon Plugin by BenPyton. This plugin

had exactly what we were looking for. The plugin allows 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. Because the plugin was built using a third person character, there was some trouble setting it up with Kronos since it is set in VR. For example, it was difficult trying to find out how to teleport the player to the generated level. After getting the player in the dungeon, we ran into the generated dungeon only showing the first two rooms, which led to several hours trying to troubleshoot on why that was occurring. Eventually we 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 we were able to fix the visibility of our dungeon by disabling a single setting called "Occlusion Culling" in the plugin's settings. The author had it enabled on default for performance purposes, but this was not needed for testing and debugging.

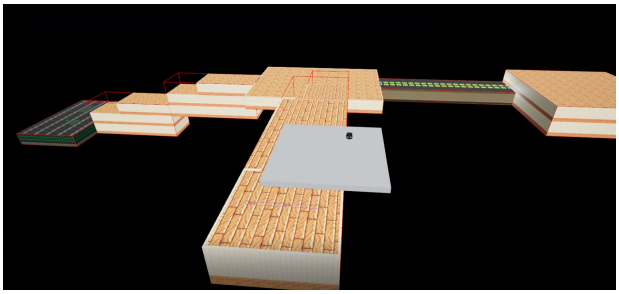

Figure 4. A fully generated dungeon after the procedural dungeon plugin is finished running.

Enemies

Because there were not free available sprites that fit our game's art style, we created sprites and flipbooks to make an animation for our 2D enemies. This also gave us 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, the behavior of the AI was created 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.

Artificial Intelligence

Implementing the AI was where the majority of our time was spent since there were many issues. There was so much to learn about AI and this resulted in us learning the ins and outs of Behavior Trees in Unreal Engine. In the beginning, the AI could move randomly in a specified radius. This little introduction allowed us to learn about the little pieces that are needed to make an AI in Unreal. Then moved onto Unreal's Blackboard, which is an object that holds all the values, in variables, that a single instance of an AI needs to function. The next step was AI Controllers, which is where you determine the Behavior Tree that needs to be run. This attaches 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 were 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.

Items

In order to create items, we considered everything that would be needed so that all the pieces 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 us to learn about how to spawn objects in the game through the Blueprints system. Not only that, but we were also able to learn about the randomness that Unreal can provide for random selection.

Weapons... Again

After creating the core mechanics of the game, the focus of the project shifted back to weapons. There were plenty of issues trying to allow the player to grab a weapon with both hands so we decided to only have one-handed weapons. Getting a two-handed weapon to work in VR would require a skeleton for the weapon and we were unable to create a skeleton with the weapon models already in the project. This was likely because our imported models did not have a skeleton. A skeleton would allow for bones to be created, meaning there are two bones for where each hand could be placed on the weapon. The decision to only have one-handed weapons 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 a 16-bit music pack from Humble Bundle [2]. This was used for the background music to our levels. We also added footstep and gunshot sounds that also were from Humble [2]. 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.

8. TRIALS AND TRIBULATIONS

Throughout the creation of Kronos, we ran into some troubles that slowed down progression.

Locomotion Movement

Initially, getting the locomotion movement to work without issue was troublesome. This issue arose with making objects able to be grabbed. Grabbing lead to a dive through collision to understand how Unreal processed what should collide with what and what to do upon colliding. The most common issue we experienced with grabbing was the case where the player would grab an object, which would then collide with the player's collision detection, moving the player in the opposite direction of the held object. Luckily, this was able to be fixed after learning more about collision presets.

Sprite Changing

Another problem we faced was sprite handling. As the enemies moved, attacked, took damage, and more, we wanted the sprite to update to represent this. Originally, we were creating our enemies through the Blueprint system alone. This meant that we were handling all sorts of events, and keeping up with what the enemy should be displaying. This wasn't a problem until

multiple events fired at the same time. The firing of multiple events led to sprites not being updated or getting updated to the wrong sprite for the action. This was undesirable, so some research needed to be done. Since this was a niche style of enemy, there was not much help available for us in this regard. During our research though, we learned about behavior trees for AI and how to implement them. This allowed for us to put more of the work and thinking on the built-in AI system in Unreal instead of creating our own and essentially recreating the wheel. Using Behavior Trees also just happened to solve our sprite problems, so it ended up being majorly beneficial.

Procedural Dungeon Plugin

There were several issues with using the procedural dungeon plugin, which is a third-party plugin. To begin with, the plugin was built using a first-person player and did not allow for a VRplayer pawn to be used as easy. This was troubling at first because our entire idea of Kronos was to create a VR rogue-like and not a first-person shooter.After sorting that issue, our progress came to a halt once we realized that the player and enemy were unable to move in the generated dungeon. This was due to the navigation mesh, which is what defines where a player and enemy can move to. The problem was that the navigation mesh did not rebuild itself when the dungeon was being generated. The resolution to this error was to enable a setting in the project's settings and allowed a dynamic navigation mesh.

9. CONCLUSIONS

At the end of the project, we are able to walk away with a lot of new knowledge. Over the course of development, we learned how to use Unreal Engine 5 and many of the features it offers. We learned more about visual scripting languages through Blueprints. We were able to experience world creation, AI behaviors, making sprites through pixel art, and much more. Overall, this has been a useful learning experience. Even if we don't make much use of these skills going forward in our careers, it will always be something can feel free to dabble with on our own time.

10. FUTURE WORK

As we move forward with the creation of Kronos, we have a couple of ideas in mind to flesh out. We would like to add a new floor, likely a modern era. With this, we would like to add more enemies and bosses to add variety to the enemies. Along with new enemies, we would want to refine our AI in order to create more intelligent behaviors to add depth to the gameplay. For example, implementing a cover system in which the AI looks for cover to hide behind while below 50% health. We would also like to create more items and possibly ways to earn items to further support the ability to continue to play new playthroughs without the experience getting stale. Finally, incorporating a save system is something we wanted to implement from the beginning but couldn't because of difficulties introduced by the procedural dungeon plugin we chose to use. Adding a save system would be difficult to implement because the dungeon is generated at runtime and does not save the player's stats, enemies, player and enemies' locations, and more. At first we tried getting a save system to work, but due to deadlines, we focused our attention on getting to a playable state first.

REFERENCES

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