Alpha Release Report

ChronoQueue

Team Team

Erinc Argimak, Achim Bunke, Rohan Fernandez, Fabian Nadegger

Progress Overview

Game Mechanics

As the Phantom behavior is one of our most important features and we focused more on it in this milestone. We created a solid structure with modular controllers that work for both Player and Phantoms. These controllers were changed to check if the Phantom has the Revenant status and can actually hit objects with its attack.

For our attacks we figured that having only 1 melee attack was not enough for diverse fights. So we added a third attack. For the 3 attacks we created animations and particles. Current the player has the attacks:

- **Headbutt attack:** Character instantly attacks with his head in front of him and does low amounts of damage.
- **Heavy Punch attack:** Character can hold the attack button down to charge up this attack. When the button is released the Character thrusts forwards a fist that damages all enemies hit. It does more damage and flies farther the longer it is charged.
- **Rocket attack:** Character can hold the attack button down to charge up this attack. When the button is released the Character fires a rocket forwards. It does more damage the longer it is charged.

Phantoms will mirror the exact inputs and yield the exact output actions as the player.

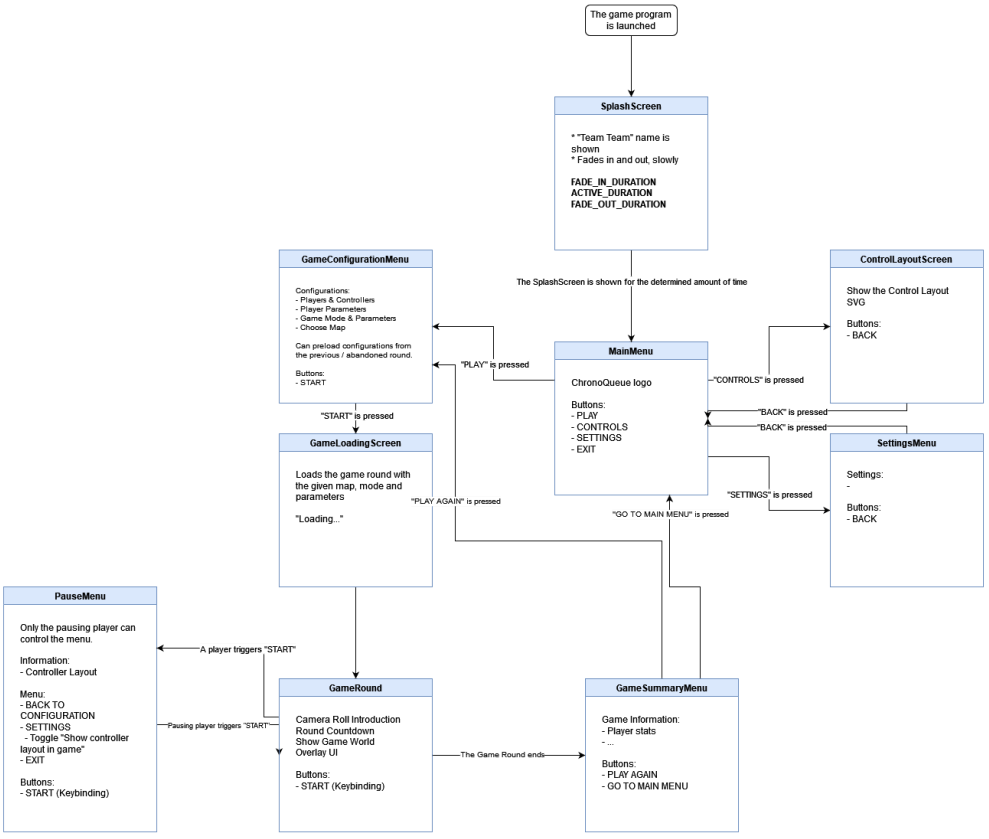
We added Chronogates for our Phantom to Revenant activation. Each player can currently hold a maximum of 3 Chronogates. He can pick up a Chronogate when he is close to it, refilling his inventory.
A Chronogate will always be placed on the floor. This leads to the problem that if the player jumps while placing it, every following Phantom would also not activate the placed Chronogate as it jumps over the activation area. Our solution is to simply extend the activation area to infinite height. Currently our arena does not have playing areas on top of each other so this does not lead to unexpected activations. If we decide to create an arena with much verticality we have to redesign the activation area.

Precise knockback calculation was added to characters.
Knockback makes the game more dynamic and more satisfying for the player to hit attacks. It also creates the possibility of advanced attack combinations with your Revenants.
Each attack now has multiple tweakable parameters to define its own knockback effect.

Game UI & Flow

As our game primarily focuses on a single and continuous game session, we did not have to come up with a complex UI flow that goes in and out of the game session. However, it was still reasonable to work on a design of how the game menu navigation should work, as such models allow the team members to be on the same page about what should be implemented and how. Moreover, it lifts the burden of making UX related decisions from the shoulders of developers, allowing for a clearer separation of responsibilities within the team.

Even though we were not able to release the new flow we designed for the game, we initiated the work on it by designing and starting with the implementation. The model image is attached below for reference. It tries to describe the game application as a state machine where different states are various screens that should come up during the game session experience.



Game flow diagram.

Rounds, Modes and the current level setting

The game allows for multiple modes and the system is made flexible to allow for that change, however at the moment we have a single game mode titled ‘Classic game mode’. A classic game mode can have multiple rounds and the 2 ways to end the game mode are by either the timer running out which is now set to 120 seconds or if there is only a single player left.

The players start off in the scene at different positions, they are initially disabled and can play after the countdown timer ends. After every 10 seconds during gameplay an instance of the character (phantom) would spawn and would re-enact the movements of the player, this would include attacks and can affect the opposition.

A UI during the game displays all the stats of the players, currently displays the player health along with the countdown timer. At any time during the gameplay the player can pause the game which would then show the pause panel to restart the game or exit to the main menu.

Players and Controllers

The game allows for 2 to 4 players. Each player is given a device (controller or keyboard). If there are 3 joysticks connected then the 4th player gets the keyboard, else if 4 joysticks are connected then all 4 players get the joystick. The joystick has a priority over the keyboard unless if there are less joysticks connected than the number of max players the usage of the keyboard will not come into play.

Game and State Management

The game includes a state machine to handle the overall condition/tasks of the game, this state machine progresses to different states calling a function for when the state starts and when the state exits as well as an event to notify the concerned listeners.
This prioritizes the managers on the point at which the user is in the game, ranging from the Main Menu UI, the Loading panel and the actual gameplay. For a simpler and an intuitive approach for the developers and the designers to design the game, the necessary functions are started in that state when it begins and ends when the state concludes. Since we now know the new and old state as well as provide that information in the event, each listener is then permitted to undertake its own class-specific actions with this information.
Each state can thus transition to any other, for example the Pause state can transition from and to the Gameplay state, and pausing the game when the pause state enters and unpauses the game on exit.

Gameplay Controller

To allow for the system to be as flexible and allow for expansion to different game modes, for example: In one game mode we need players to start off with 3 lives or another where the round time is different or another example is we would want something to spawn in the environment after every 20 seconds or 40 seconds in another game mode. With all these requirements in mind we have created a Gameplay Controller Base class which handles all related events and with the use of inheritance allow for each Controller to modify their own values. For now we provide a single controller to represent a classic mode and if we do wish to expand on the game or require any sort of modification it would then be easy to do so.

The Gameplay Controller is responsible to manage all gameplay conditions from the start of the countdown to when the game ends. It can set the round length and how many lives there would be in a game session. Also if there are rounds the gameplay controller would start a round and end/ go to the next round at the same time keep the data of the game session intact.
The end condition of the game is dependent upon if only a single player is left which means all the other players are dead or when the round time is over. As a flexible structure one can change this property by adding more game modes.

Camera

The camera during gameplay always strives to focus on the average position of the players that are at that point alive in the scene. It would be slightly more beneficial for the players for us to add a Field Of View modifier to provide some solution for edge cases, when 1 player is positioned extremely far off from the other

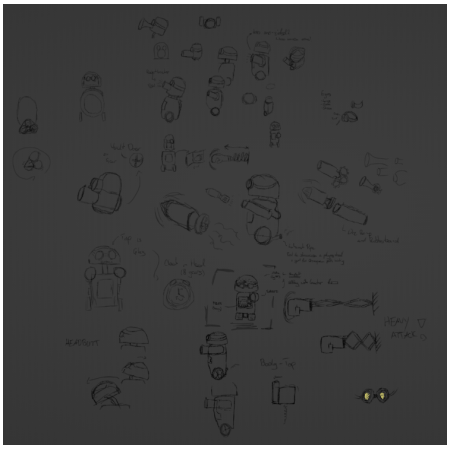
Assets

After the prototype assets were done, the task was to create the final assets we will use in the game.
As the visual style was in a more steampunk direction we started by looking at concept art and inspiration of similar games and movies with said visual style (e.g. Arcane or Dishonored).

Character models

First we created some concepts of what a playable character could look like. The character should incorporate one ranged attack, two types of melee attacks and the ability to jump and move.
We decided on a robot looking character which incorporates both styles - cute colorful style of arena battlers, as well as the grim metallic steampunk look. The robot's shape consists of 3 simple shapes (body, head and canon). Each of these has a different form so that the silhouette of the player is easy to understand in the battle.
With these basic shapes the character was further developed. As the heavy melee attack should have quite some reach we let the player have a little door open on its chest where a metallic fist can shoot out, punch the enemy and go back into its chamber. Also small details were added to make the characters more believable in the game world, such als gears connected to the wheels or an oven on the back which seems to power the players every movement. Because of the games visual style and the time related topic there had to be a mechanical clock incorporated into the players design, in our case the players head.

For realistic but also stylized texturing we used rather simple colors with only some noise as the base albedo for the models elements. With the same noise we control the roughness and normal map of the model. On top of these procedural materials we drew some stylized details as edges or smudges onto the model to create a more intriguing style.
As procedural materials have to be baked into textures to be used in games engines, we baked the roughness, normal and albedo map as well as ambient occlusion and, in the case of the oven, light.



Animations

For the amations we tried to combine simple animations to create good looking animations. For example the walking animation is not just simply the wheels turning but the player also bops a bit, the gears start turning and the not perfectly secured oven door wiggles a bit.
These animations which consist of multiple smaller animations have to be combined together back again in Godot.
In Godot the Animations of the models are placed back together via the AnimationBlendTree in which an AnimationTree controls all the players animations via parameters set in the player's movement controller.
Here, we were confronted with some trouble as Godot imported the models with empty animations on each mesh and duplicated animations on the animated meshes which decreased the overview. Additionally, there was confusion as Godot did not always save changes in the animations as believed as some of the duplicated animations were referencing one another.

Arena

As we did not yet decide on a final arena design, we built the assets for the arena in a modular manner, so that the arena can be built and redesigned in Godot itself.
So the arena assets for example consist of a uniform tileable floor, ramp and wall tiles, which allows to shape the arena's basic form.
To inherit the map we create some obstacles and decorative models such as windowed wall tiles or animated pillars.
The texturing process was very similar to the player models. But because the basic shapes of these models were very primitive we made some of the procedural textures in MaterialMaker which allows to simplify the baking process in Blender itself.
Until the final release more models will be added for a more visual immersive stage.

Challenges

Product and Project Management

Specifying the desired program condition including the supported features, required software quality measures and how exactly everything should feel and behave is a big challenge in itself that has pressured the developers and management alike, while also attracting many people in the field to come up with various solutions. When we add parallel work into this, the software and how it is architectured also comes into the play as concurrently working on the same project artifacts makes a team inefficient and can easily cause incorrect results. One more layer of complexity added on top of all of this is the external deadlines, which introduces the necessity of a plan for satisfying the intermediary deadlines while working towards longer term goals efficiently.

In our project, we have experienced most of these issues together. We are still on a road of constant development where we are iterating on the tools we use and how to use them in order to specify what versions of our game should be capable of. We started with a more complicated Kanban task screen where we would group tasks based on status, priority and the assignee. However, as we progressed, this approach had difficulties representing the actual development progress we were going through. It was not agile enough. Therefore, we simplified the model and returned to simple lists representing progress in different facets of development. This allowed us to see simpler lists that relate to the functionality of a version, but even then, various shorter and longer term ideas got intertwined and distracted us from our closest milestones. We had to refine the list multiple times in order to approach something that keeps us all on the same page and provide a clear vision about what we should expect from our game.

The fact that we had to work on the same project in parallel also forced us to adapt to a stable Git workflow that would make sure we don't make errors while comparing versions and always include the right changes in the next addition. Adapting to this workflow took some time but eventually put us onto a stable track of development.

One other issue that revealed itself during the process is the decision of when to focus on progressing with a functionality as much as possible versus refactoring the code in a way that will allow us to sustain our progress in the longer term without losing pace. Using different approaches to similar problems within the same project creates inconsistency and decreases reusability and program minimality. However, this technical debt requires work that takes time and energy; and it is usually the case that an external deadline pressures the developers into completing the intended features for this milestone. This issue is in the nature of software development and creates a big opportunity for solving this problem in balanced ways through assisting methodologies and products.

How much the work process should be documented and monitored versus how much the actual work should be done is the primary question in this field; we initialized and iterated on our own unique approach -- Which was a very educative experience.