Interim Report ChronoQueue

Team Team

Erinc Argimak, Achim Bunke, Rohan Fernandez, Fabian Nadegger

Current Progress

Player movement

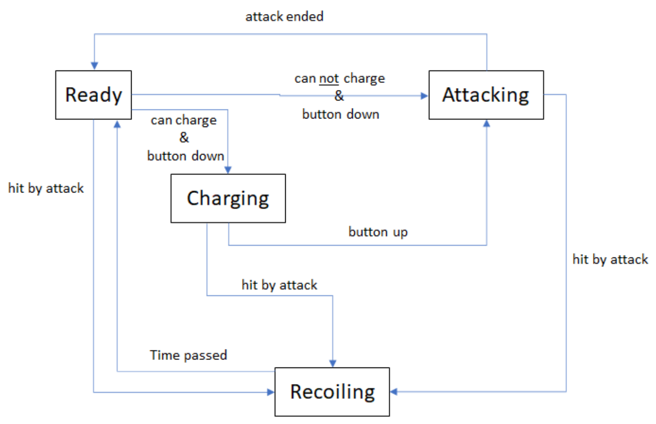
We have implemented basic movement for up to 4 controllers simultaneously. Additionally one single character can be controlled by keyboard and mouse. Keyboard support is not our main target and is only supported for debugging purposes.
Currently moving consists of 2 dimensional moves as well as the ability to jump.
Movement variables are accessible by other components to modify a player's behavior while doing other actions, an example is explained in the battle mechanics chapter.

Our game is supposed to have simple movement as the combat and the ChronoQueue mechanic are the main focus and should not be avoided or have its importance decreased by making movement abilities too strong. To achieve our desirable target balancing of the movement parameters is still required as it feels a bit lacking currently.

Battle mechanics

For battle mechanics we focused on creating an efficient structure.
It was designed in a way that allows arbitrary attacks to be easily constructed in the future.
These attacks must also be able to be used by Revenants that receive input by reading the saved state information of the player's timeline.
Currently there are two functional types of attack that were already created.
Hitbox attacks use colliders that are switched on and monitored when executed. They will collide with the hurtbox colliders of characters.
Projectile attacks create and fire projectiles when executed. Projectiles then behave the same as hitbox attacks.
After a hit on a character was recognized the attack checks multiple parameters before damaging the opposing character. These parameters include:

- Has this attack already hit that character
- Character invulnerability
- Is character friendly



Attacking results in the attacking character as well as the target character to switch between multiple states impacting many other components of that character.
While recoiling a character cannot perform any actions.
Additionally to recoiling a character also can get invulnerability time after getting hit by an attack. Invulnerability prevents a character from getting damaged while still being able to move.
Every state can affect the movement abilities of the character in different ways like movement speed, turning speed, jumping ability, etc…

For testing purposes we created a simple punch attack as well as a projectile attack that just shoots a bullet in the characters looking direction. While these attacks do not have any fancy animations or models they confirm how easy the construction of attacks is with our structure. This will allow us to easily create different characters in our high target goal.

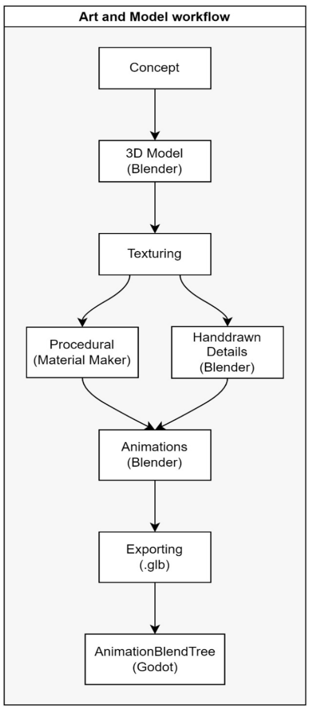
We regard our battle mechanics as finished for our minimum target and after creating 2-3 more polished attacks for the character as finished for our desirable target.

Game states

A simple state machine is created for game management. This allows us to divide the stages of our game into distinct parts, for example: Main menu, gameplay, game ended. For easy game management, each state gets an event invoked when its state is entered and exited. We also get the state ID’s from the newly entered or exited state to configure the code to behave differently in different state changes.
We plan to create the following states, Main Menu, Loading, Gameplay, Pause, Score, End game. Transitions are possible from set state for example: From End to Main menu or Pause to Main Menu.

Art

We created simple prototype models to visualize our character and stage to start working on the PlayerController and Camera scripts.
For the more complex models which will follow we created a workflow from start to finish.
After conceptualizing how a model should look like it will be 3d modeled in Blender. The models will be base colored by procedural materials we will create in Material maker - a node based procedural material maker which was also made in Godot. Additionally we will hand paint some details in color- and bump-maps if needed. Then in the last step the models will be animated if they have any as e.g. the players.
Every model has to be exported as an .glb file for Godot, which will also include materials and Animations. The latter has to be arranged (and maybe put together again) in an AnimationBlendTree.



Project Plan

Our initial plan was to have the Phantoms and Revenant feature in the game at this point in time. But due to unexpected delays we were not able to get this feature ready. This will definitely shift our tasks for the next milestone so that more people are assigned to this feature than planned. For the other features we are progressing as planned.

We expected to have support for more than 2 players only in our high target but after figuring out the controls it was very easy to scale for more players. That's why we currently already support up to 4 players simultaneously.

In respect to our battle mechanics we found a problem with the number of attacks and attack types. Our original plan was to have two different attacks where one is melee and one is ranged. We figured that only having one melee attack is not enough for our gameplay. In close combat a player would only have one button to press as ranged attacks should not be viable at that distance. This heavily reduces the strategic choices a player has. For that reason we decided to add an additional melee attack that's viable for close ranged battle.
While discussing this feature we also added attack variations to our high target. These variations depend on the motion state of a character (upwards/downwards/forwards motion). These require many new attacks that are not necessary to create the core game but definitely can make it more fun and diverse.

Reflection

The choice of Godot as our game engine made us question how easy it is for us to adapt because all of us are in general more knowledgeable in Unity.

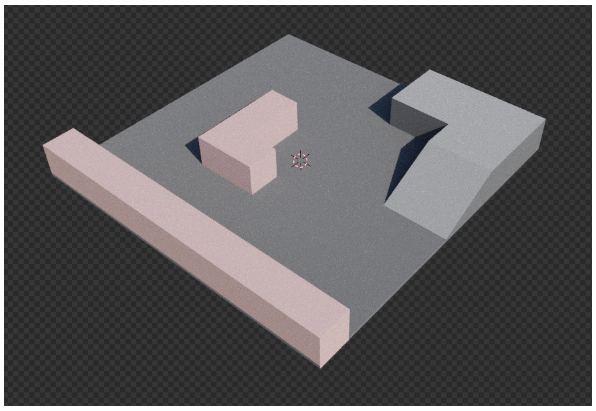
While we encountered some unexpected changeups in the art import workflow the vast majority of Godot was very easy to adapt to compared to Unity.

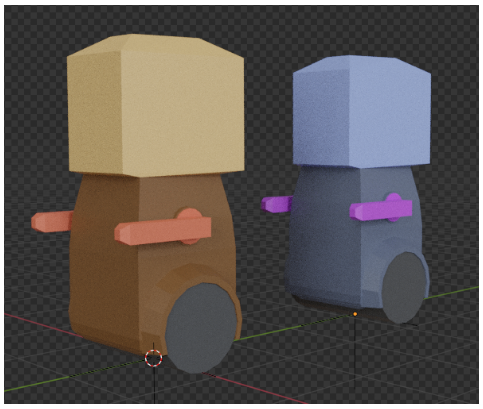
Getting the input from multiple controllers was more difficult than expected. In Godot you can define Input maps for each button on any device but a method to correlate the button to the device is not offered by Godot. The easiest solution and the one we chose was instead of creating Input Actions that listen for each button on any device we created a different Action for each button for each controller. This requires more work when changing controls because you would have to do it 4 times (1 for each controller) but this solves the issue of figuring out who controls which character.

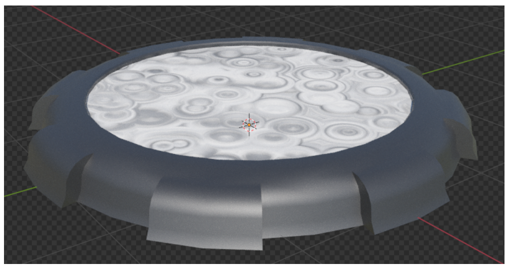
To import 3d models with multiple animations on different meshes as a single animation turned out to not be as easy as expected. It was not possible to simply recombine the animations or to play them simultaneously but we had to create a whole AnimationBlendTree. We would have anyway but as we have to recreate/reblend certain animations back to one the AnimationBlendTree gets bigger than first anticipated.

For our Git workflow, we decided to create a main branch and a development branch as our major branches.
The main branch will have the major working versions of our game and the development branch will be a more continuous branch that will receive more frequent updates of our development process.
From the development branch out, we will branch off into different sub branches divided into their respective fields (e.g features or art) which will be further divided into the specific task we implement (e.g feature/player_controls).
After the task of one of these subbranches is completed, we will request to merge the branch into the development branch. Here we will also communicate and review the merge requests to minimize the risks of any major merge conflicts.
After a task of a branch is done, the branch will not be used any further and will be deleted to keep the branching as clean as possible.

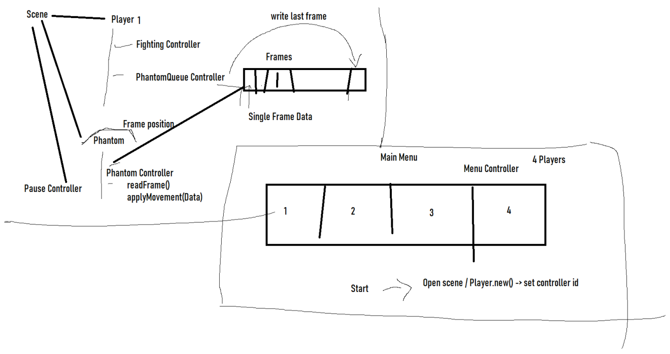
Prototype Models


Prototype stage


Prototype Players


Chronogate

Chronoqueue structure


Scene
Player 1
write last frame
Fighting Controller
Frames
PhantomQueue Controller
Single Frame Data
Frame position
Phantom
Phantom Controller
readFrame()
applyMovement(Data)
Pause Controller
Main Menu
Menu Controller
4 Players
1
2
3
4
Start
Open scene / Player.new() -> set controller id