Velocita Technical Design Document

Written By Zachary De Maria

# Introduction

Velocita is a racing game built in Unity featuring a physically based hoverboard character controller with controls similar to games such as Mario Kart, including a drift ability and arcade style boosting. Laps are tracked by using numbered gates, and the racers position in the standings is inferred by a combination of gate and lap number of each racer and relative distance from the previous lap gate in cases where the racers are at the same gate. Racers are warned via the HUD and automatically reset on the track if their velocity goes counter to the direction of the most recently passed through lap gate for longer than a configurable period of time and upon completion of the race players are presented with a results screen featuring lap times of all racers before being returned to the main menu.

# Coding Conventions and Folder Structure

The following folders are used throughout development:

## Characters:

Contains all scripts pertaining to characters, including player standings and visual controllers, but excluding HUD scripts etc.

## Track:

Contains all scripts that live on components placed on tracks, such as lap and rolling start gates, as well as the main player spawner and manager

## UI:

Contains all scripts pertaining to user interface, including HUD and menu scripts

# Scene Setup:

Each track requires a **PlayerSpawner** and several **LapGate**s to be placed, one of which must be a **StartGate**. Players are spawned from prefabs containing components including **PlayerController**, **HoverboardController** and **HoverboardVisualController. Each player will be spawned with a HUDController which manages the HUD canvas that each player spawns with. Menus are all controlled with their own classes simply handling selection and controller inputs.**

# **System Architecture Overview:**

The main classes in Velocita are detailed below:

## Controller:

An abstract class designed to expose control inputs for each hoverboard, is extended by PlayerController and AIController to allow for control inputs to be fed by a PS4 or Xbox controller or an AI behavior system respectively.

## HoverboardController:

The main meat of the gameplay code, the HoverboardController contains many tweakable variables that control a set of interacting offset forces and torques that allow the hoverboard to float, move and drift. Much of the class is varied AddForce and AddTorque classes, and a Rigidbody is required on the GameObject containing the HoverboardController. The board floats on a customizable about of “thrusters” each consisting of a raycast and an offset force applied at each of the thruster positions. A gizmo showing the location and desired height of each thruster is available to aid designers in tweaking these parameters. This class also exposes a lot of variables to the HoverboardVisualController class, and takes its inputs from a Controller class, which can either be a PlayerController or an AIController.

## HoverboardVisualController:

Controls all the visuals of the hoverboard, including the particles and trail effects, as well as camera FOV and helping to direct the animations of the character standing on the hoverboard. Needs to be attached to a hoverboard which also has a HoverboardController component on it.

## PlayerSpawner:

Handles spawning and assigning controllers to each player, as well as managing the players throughout the race, handling things like race completion, rubberbanding, and player standings. Contains vectors for spawn locations as well as references to each player, the player prefab, HUD prefab, etc.

## LapGate:

Tracks player progress through the level, its forward (local Z) vector determines the direction of the track, and a trigger is used to update players passing through the gate with the gate number, and direction, for tracking if the player is progressing the right way / has passed through gates in the correct sequence.

## HUDController:

Displays various data such as player speed and position, as well as current lap to each player, is added to each player and rendered on a separate camera on a per player basis to support splitscreen.

## CameraController:

Smoothly interpolates camera position according to player heading and velocity, keeping camera motioin smooth and pointing in a relevant direction even when the character controller itself moves rapidly, i.e. crashing, fast turning etc.