14 sept 2025 | Unreal Engine developer task summary

Mauricio Eduardo Tarno (55maurit@gmail.com) - Link to Github Repo

How to play

The players start on the side of a small prototype level.

- Push with **Q** and slow down with **E**.
- They can turn with **WASD** or directional arrows.
- Jump with **Space**.

Obstacles where the player gets points for jumping on top of them and dies if hitting the actual barrier. The level can be **reset with Esc or K**.

Features

- The player controls the SkateCharacter, which can push to **speed up** and it's also able to **slow down**. Pushing is an impulse applied in **sync with the animation**.
- The SkateCharacter supports full movement and **velocity conservation**, with the speed diminishing due to the **ground friction**.
- The character works with inertia and its **speed will be redirected** while **turning with the skateboard**.
- The character will **slow down and slide on ramps** according to the direction of the ramp and the skateboard, so if the skate is perpendicular to the ramp, it will not slide.
- The **character cannot jump while pushing nor vice versa**. While an action is being performed, the skate will turn orange. When finished, it will be green again. The actions cannot be cancelled, unless the character falls off an edge while pushing.
- There are **jumping obstacles** which **award points** for jumping on top of them.
- If the character hits an obstacle, it gets its ragdoll enabled, simulating falling.
- There's a basic UI showing the controls and points awarded.

Implementation

All of the following is C++ unless stated otherwise.
I started with the ThirdPersonCharacter template. For the character, I knew the movement had to be kinematic, not a physical simulation per se. So I programmed the Pawn (**MauriSkateCharacter**.h) as a layer on top of the CharacterMovementComponent.
All the relevant variables are in the category "Skate". Those useful as a parameter are exposed as read/write, while those necessary for the animation are read only and the internal ones, private.
I proceeded disabling rotation functionality for the CharacterMovementComponent and poking inside of its source code to implement my own rotation and modifying its velocity. I made my own animation blueprint for the sake of simplicity, however, it doesn't support foot placement or skate rotation for ramps.
For the points system, I don't like to add functionality to the GameModeClass directly. I rather like components. So I made a **GamePointsComponent** meant to be added to the PlayerController. It has a dynamic delegate which broadcasts when new points are updated.
The obstacles are also Blueprints with an **ObstacleSettingsComponent** that finds a mesh for an obstacle and a box collider for a trigger. It subscribes to the delegates of those components and calls the GamePointsManager to grant them. I like to do functionality in components because they make adding functionally a drag and drop operation.

The UI is a simple widget that subscribes to the GamePointsComponent delegate from its graph in blueprints.