Blue Gravity Interview Task

Documentation for the Skateboarding Simulator Prototype

By Douglas B. Gomes

Here, in this document, I will explain how the skateboarding system works and how to modify it.

Abstract
The presented system is a skateboarding simulator which features the following functions:

- Basic movement, such as: skating around, jumping, and falling from the skateboard.
- A score system.
- A skate trick system, containing four basic tricks: Ollie, Nose Grind, Quick Flip, and a "Gap," which are extra points given when the player jumps over a given gap.
- A combo system, which consists of summing points for each trick, also multiplying the points by the number of tricks you made in this specific combo.
- A basic playground level.
- Support for network replication.

Basic game controls
The game controls will be presented in this order:

Name of command: Keyboard keys/Xbox Gamepad.

So it goes:

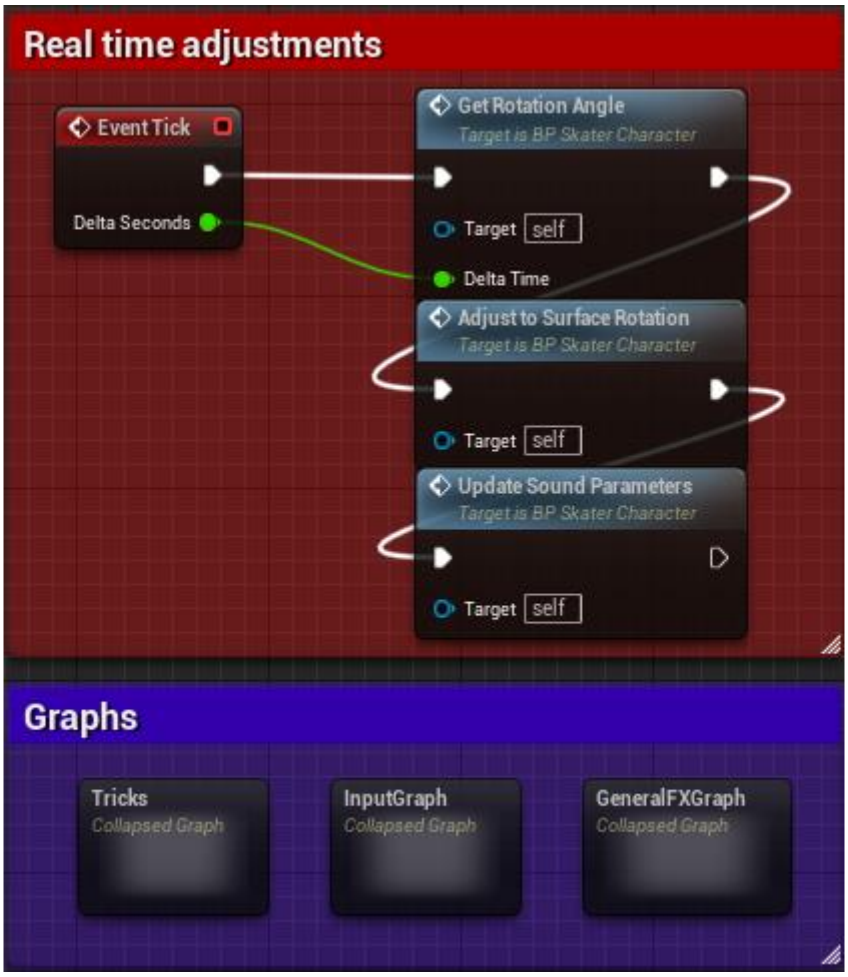
- **Turn left/right:** AD/ Left Thumb stick.
- **Move camera:** Mouse/ Right Thumb stick.
- **Slow down:** S/ Left Thumb stick down.
- **Jump (release); Start skating (when stopped); Run (hold):** Spacebar/ **A** button.
- **Quick flip:** E/ **X** button.
- **Grind on ledges:** Hold F or **Y** button when approaching a ledge.

How the system works

In this section I will explain how each of the modules of the system works.

- **Character Blueprint:**

First, I will present the main character blueprint. When you open the "BP_SkaterCharacter," you will find these three graphs, each one encapsulates the main functions of the character:



The only functions that are not inside it single graphs are those that runs on tick.

The trick graph contains the functions and events that makes the tricks possible, but the main one is the "Try to do a local trick," which is responsible for handling any trick that the player is trying to do. Further details can be found inside the BP comments.

As the name suggests, the Input Graph initialize and listens to any "Enhanced Input" event. Movement and trick events can be found inside this graph.

The "General FX" graph should listen and replicate any sound or visual FX called by the player. It only handles sound effects by now but is fairly easy to implement any other function of this kind using the same sound logic.

Main properties:

- **MaxNormalSpeed:** Controls how fast the character will go when it is not running.
- **MaxNormalAcceleration:** Controls how long the character will take to achieve the MaxNormalSpeed.
- **MaxRunSpeed:** Same as before, but now with the run speed.
- **MaxRunAcceleration:** Same as before, but now with the run speed.
- **FallDelay:** How long will the player stand on the ground when it falls from the skate.

- **Player controller:**

The player controller is only responsible for carrying the “AC_UIManager,” an Actor Component that lives inside it. I will further explain the features of this component in its own section.

- **Game state:**

The game state (“BP_SkaterGameState”) is responsible for listening to the “BPI_GameState” interface, which contains the following functions, in order:

- **AddScore (E_Tricks, PlayerState) –** Called by the player character when it does a trick.
- **StopTrick (PlayerState) –** Called by the player character when it finishes a trick.
- **CancelCombo (PlayerState)** - Called by the player character when it falls from the skate
  Note: This interface message can be called from wherever BP when you need to cancel a combo for whatever reason.
- **Finish combo (PlayerState) –** Called by the player character when he lands without falling from the skateboard.

As you may noticed at this point, all the interface calls need a reference to a player state, that is mainly because the game state at this point is acting like an “API” due to some replication reasons and the Unreal pipeline itself. Surely, I could call all those interfaces calls inside the player state itself, but that could lead to a hiccup if we need to replicate any information to other players in the game state, so this could be considered as a forward feature implementation.

Main property:

- **MaxComboTime:** How long without making any trick to finish the combo (only on air)

- **Player state:**

The player state is only responsible for carrying the “AC_ScoreManager,” an Actor Component that lives inside it. I will further explain the features of this component in its own section.

- **AC_UIManager:**

Inside this actor component you will find two functions:

- **Create Main HUD:** this event will create the main widget and will add it to the viewport.
- **Remove Main HUD:** this event will remove the main widget from the screen if it is valid.

The second one is not called anywhere in the project, but is there, just in case.

- **AC_ScoreManager:**

This is the component that is responsible for managing the score per trick and the combo system itself. It lives inside the player state.

Inside of it you will find the main following functions:

- **AddScoreToPlayer(S_TrickData):** It will read the information provided by the player character via a custom struct. If it is a loop, it will loop adding points until the “Stop Trick” function be called.
- **StopTrick:** Stops the current looping trick (not necessarily will finish the combo).
- **CancelCombo:** Will cancel the current combo and zero out the “ScoreToAdd” variable.
- **FinishCombo:** Will cancel the current combo but will add the points to the player’s score.

And that wraps up for the main blueprints in the project, the widgets only listen for events called by the previous ones, mainly by the “AC_ScoreManager”.

Now I will do some guides on how to set up new tricks for the system, how to set up a “griding zone” and also how to set up a “gap” zone.

Guides

- **How do I set up a new trick?**

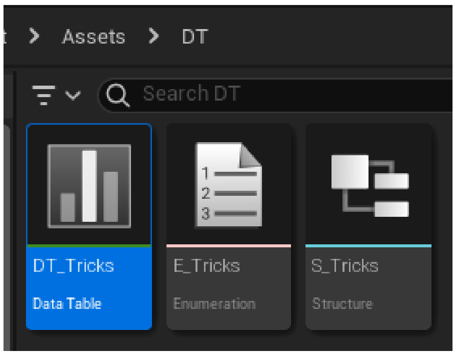
The trick system is data driven, which means that he uses a Data Table asset to store all the data needed to compose a trick.

- **Step one: create a new entry on the E_Trick enum.**

In the content browser, browse to Content/Assets/DT/. Inside of it, you will find the "E_Tricks" enumerator. Double click on it and add the name of the new trick you will going to add.

- **Step two: create a new row in the data table.**

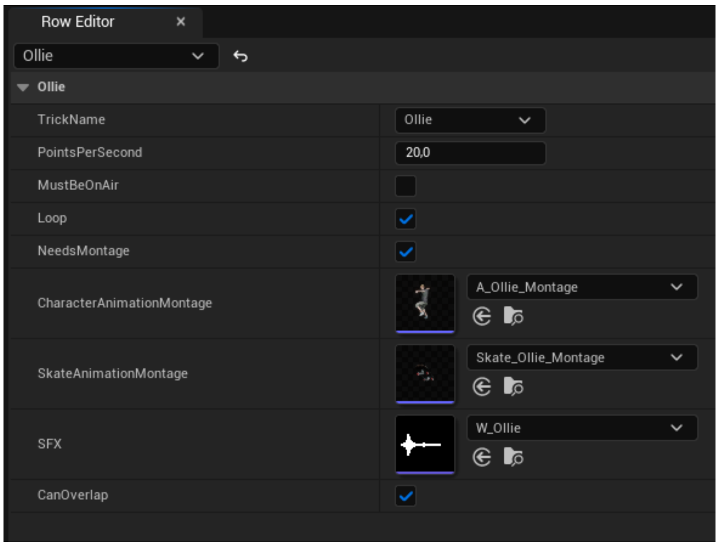
In the content browser, browse to Content/Assets/DT/. Iside of it, you will find the "DT_Tricks" asset, double click on it.



Now that you have the data table open, add a new row on it by clicking in the top bat button labeled "Add".

- **Step Three: configure your new trick.**

First, name the row exactly how you put the trick name on the Enum of the first step. Then, select the row and fill up the data.


Row Editor

Ollie

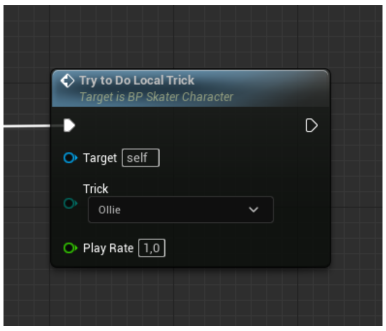
Ollie

TrickName	Ollie
PointsPerSecond	20,0
MustBeOnAir	☐
Loop	✓
NeedsMontage	✓
CharacterAnimationMontage	A_Ollie_Montage
SkateAnimationMontage	Skate_Ollie_Montage
SFX	W_Ollie
CanOverlap	✓

For instance, this is the data for the “Ollie” trick. Here is what those variables means:

- **Trick name**: The name of the trick, select the corresponding one.
- **Points per second**: As the name suggests, that is the number of points that this trick will give per second.
- **Must be on air**: The player must be on air to execute this trick?
- **Loop**: This trick will be giving points as it is executing or will be one shot?
- **NeedsMontage**: This trick will need a specific animation montage?
- **CharacterAnimationMontage**: The montage for the skater (if needed).
- **SkateAnimationMontage**: The montage for the skateboard (if needed).
- **SFX:** The sound effect to be played for this trick, leave blank if it is none.
- **CanOverlap:** Will be this trick overlap able? If yes, the player will be able to execute another trick while this one’s montage is not completed yet. Otherwise, the new trick will not be triggered.

Now that your trick is configurated, you can call the “TryToDoLocalTrick” function inside the player character, passing as an argument your new trick and the system will do everything else for you!


Try to Do Local Trick
Target is BP Skater Character

Target self

Trick Ollie

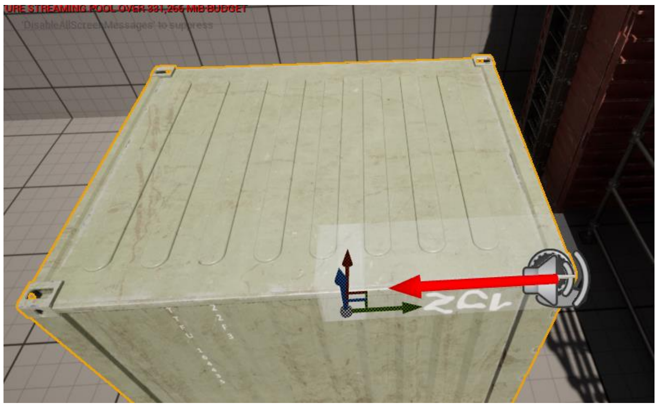
Play Rate 1,0

- **How do I set up a grinding zone?**

That is simple! Follow the steps:

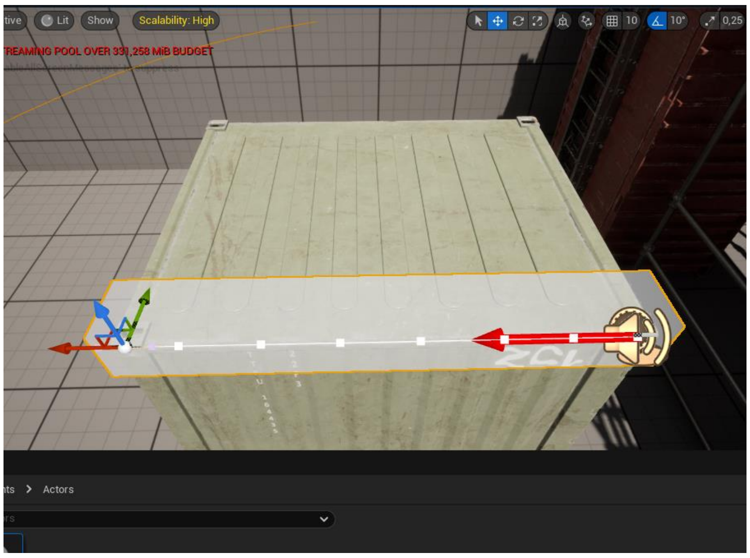
- **Step one: place the actor.**

In the content browser, navigate to “Content/Assets/Blueprints/Actors” to find the “BP_GrindActor”. Drop this actor in to your level, it should look like this:



- **Step two: draw the spline.**

Now that the actor is inside the level, you will only need to draw the spline in the shape of the object you are aiming for. In this example it will look like this:



The rectangular shape around the spline is a visual representation of the trigger that this actor will listen to.

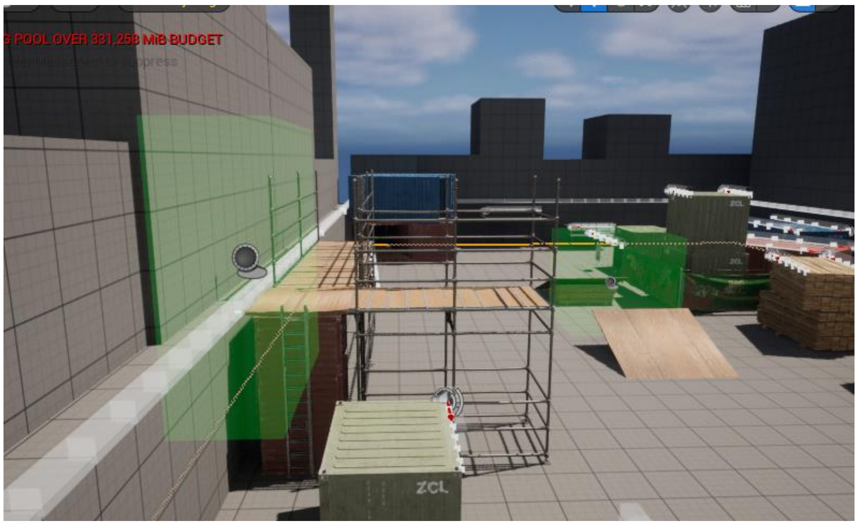
It is easy to add spline points by selecting one of them, hold the Alt key and drag the gizmo.

Note: The speed of the character along the spline is currently determined by the number of the spline points, it is not ideal, but I can fix this in a future update. If the character is too fast in this grind, just select some spline points and delete them.

And that is it, your new grind zone should be ready to go!

- **How do I set up a Gap zone?**

Gap zone is an actor that will trigger the “Gap” trick for the player if he passes through this zone while he is on air. Setting up the gap zone is as simple as putting it inside the level and adjusting its scale and location. The gap zones are represented by a green cube inside the level editor, see the example below:



The gap zone is located inside the “Content/Assets/Blueprints/Actors/” folder.

And that wraps up for the guides, I will be available in case of any doubts regarding the systems that I made.

Task notes:

- I am a 3D animator, so I just could not stick on using only the animations provided by Mixamo, so I made all the animations I considered necessary for this run such as Ollie, Grind. I think it really adds to the visual aspect of the project.
- Most of the sound effects came from Tony Hawk’s Pro Skater 1.