Andrés Arbona Interview

Introduction:

Good morning/afternoon/evening to whoever's reading this. My name is Andres, and for this test I built the requested simulator, centering it on Unreal's Gameplay Ability System (GAS) to implement the core gameplay loop.

Research:

The main resources I used were:

- https://github.com/tranek/GASDocumentation
- https://www.youtube.com/watch?v=tc542u36JR0&t=1094s
- https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-ability-system-for-unreal-engine
- Chat GPT-5: Deep research

How a Skateboard Works

Moving Forward

When riding, you're affected by the board's motion. Adding impulse (pushing) or going downhill builds momentum, which gradually decays due to ground friction.

- Reference video: https://www.youtube.com/watch?v=KTJnElipufg

Turning

A skateboard mainly turns based on how the rider shifts their body weight. Shift the weight forward to go left and back to go right. Sharper turns can be done by combining a center-of-gravity shift and a quick leg movement to pivot the board.

- Reference video: https://www.youtube.com/watch?v=7of2VqAiM2Q

Braking

There are three common ways to brake:

1. **Foot brake:** lower speed by dragging one foot on the ground.
2. **Tail drag:** shift weight back until the tail touches the ground.
3. **Emergency stop:** step off and press the tail to lift the wheels, bringing the board to a full stop.

- Reference video: www.youtube.com/watch?v=IdH-c2dbH10

Jumping

Jumping (an ollie) involves pushing off with your legs as in a normal jump, but crucially applying force on the tail first, then kicking the deck at an angle so it returns to a plane parallel to the ground—preventing instability on landing.

Video de referencia

- https://www.youtube.com/watch?v=trY6N5mhjUk&list=RDtrY6N5mhjUk&start_radio=1

The System:

I designed movement and gameplay around three pillars: Enhanced Input, GAS, and animation driven by Gameplay Cues.

I receive **W/A/D**, **Left Shift (Push)**, **Left Ctrl (Brake)**, and **Space (Jump)** in the `MainPlayerController`. While there's non-zero input, the Controller calls `ProcessMoveInput` on my character every frame. There I map `InputAxis.Y` to **throttle** (pushing along the actor's forward) and `InputAxis.X` to **rudder** (rotating the actor). Nothing moves in `Tick` without input, and the camera does not alter direction.

My character implements an **Ability System Component**. On begin play, I apply a startup **Gameplay Effect** that sets movement attributes (max speed, acceleration, friction, deceleration, air control, turn curve, and jump). A custom **SkateMovementComponent** reads those attributes every tick and copies them into `CharacterMovement` (without touching rotation), making the “feel” fully data-driven.

Actions are modeled as **GAS Abilities**:

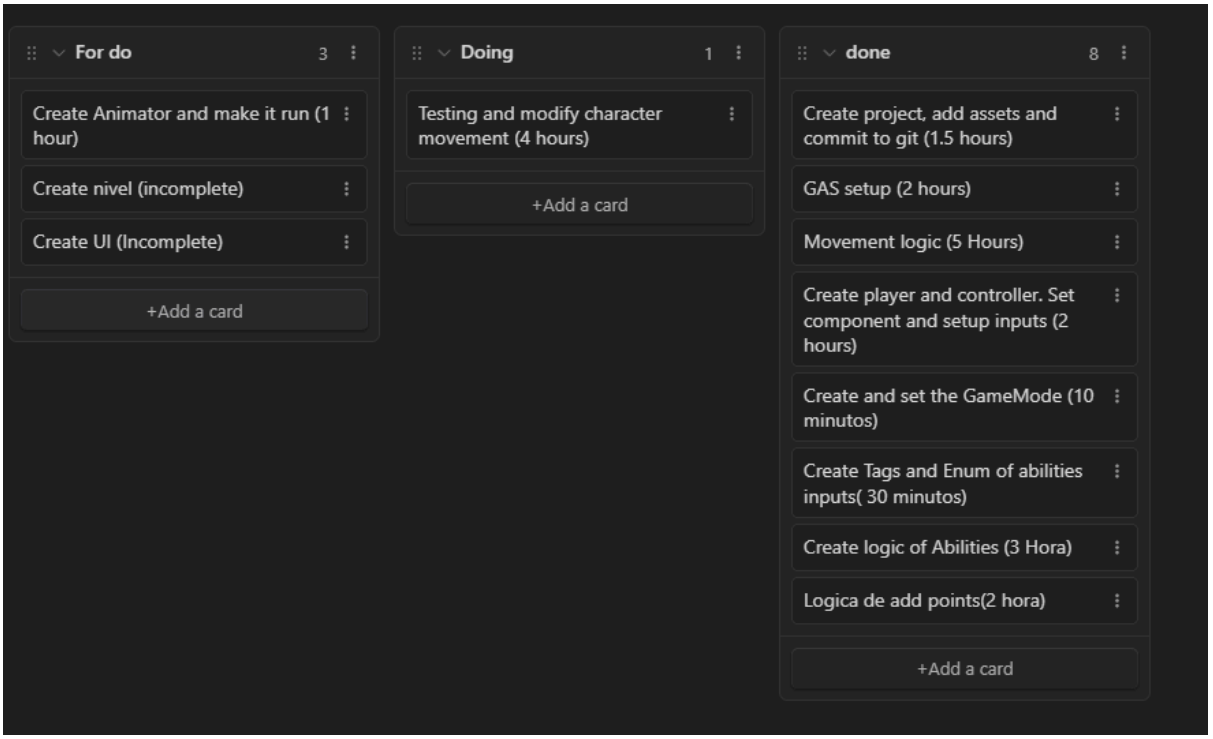
- **Push (Left Shift):** applies a brief GE that boosts speed/acceleration, then triggers a cooldown.
- **Brake (Left Ctrl):** holds a GE that raises friction and deceleration while the key is down.
- **Jump (Space):** checks that I’m grounded and at a minimal speed, then applies an impulse and can grant temporary extra air control.

Additionally, an **Obstacle** emits a Gameplay Event; my **AwardScore** ability responds by applying an instant GE with **SetByCaller** to add points—no manual writes to the `AttributeSet`.

Gameplay Tags (states and cooldowns) coordinate ability activation, animations, and HUD. The `AnimInstance` reads speed, grounded state, and tags to drive loops (Idle/Roll) and one-shots (Push/Jump/Land); **Gameplay Cues** fire SFX/VFX. The result is a modular, predictable system that’s easy to tune without recompiling.

Personal Note:

My thought process was straightforward. I first aimed for a basic system without GAS—just `CharacterMovement` tweaks for acceleration and some physics play. During the interview, I was asked about GAS, C++, and multiplayer, which led me to pivot toward a more “overkill” system given the limited time.



Across these 48 hours I put in 21 hours of work. I haven't slept much—it's 5 a.m. as I write this—but I genuinely enjoyed the challenge. I may not have reached a fully playable build or checked every box I wanted, yet I learned a lot and pushed hard to deliver.

I'm sincerely interested in working with you. I love coding more than Blueprints, I enjoy when systems click, and I'd be excited to keep learning and contribute my grain of sand to the team and the industry—if you'll have me.

Kind regards,
Andres Arbona