Improved 3rd Person Character Controller for 3D Platformer Game in Unity

Ashley Revlett
University of Louisville
anrevl01@louisville.edu

1. Introduction

Unity's default character controller component provides many but not all of the features a robust third-person controller requires. In particular, support for slope-sliding and detailed jump control is lacking. The new CharacterMotor script in this package adds these features. When combined with other scripts in this package, a fully-functional 3rd-person controller suitable for 3D platformer games is available. Animation blend trees and sound and particle effects are also included.

Because 3d platforming games often require unrealistic physics, such as sudden stops and unrealistic gravitational effects, rigid body physics are not used. Instead, the CharacterMotor script moves the character directly using the Character Controller's Move method, and the Character Controller's Capsule Collider is used to detect collisions.

2. Character Controller Component vs CharacterMotor Script

Features of Unity's default Character Controller Component:
- Collision detection using a Capsule Collider
- Prevents walking from flat surface onto a sloped surface greater than slopeLimit
- Stair-walking support
- Collision detection

Shortcomings of the Character Controller Component:
- Does not cause character to slide down a slope if they jump and land on it
- Does not cause character to slide down slope if they walk off ledge onto sloped surface

This leads to undesirable situations where the character is standing on a slope steeper than should be allowed, based on the slopeLimit:

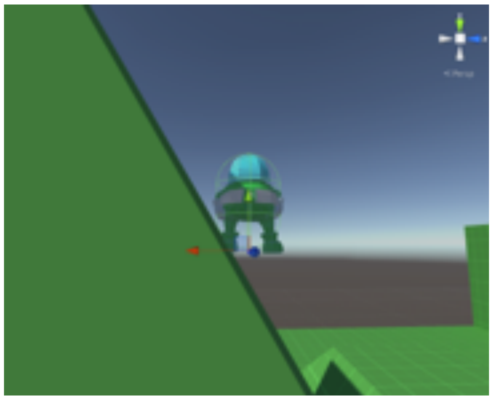


Figure 1. Undesirable standing position; slopeLimit = 45°

The new CharacterMotor script adds support for these missing features, as well as other improvements including:

- Adjustable Gravity
- Choice of Acceleration-based movement or "total control" movement
- Jump with optional variable height setting and double jump support
- Dust cloud particles during sudden acceleration
- Sound effects for footsteps, jumping, landing, and sliding, all synced with animations
- Variable amount of in-air control

Other scripts provide support for static and falling platforms, collectable objects, and non-player characters.

3. Character Motor Script

Slope Sliding Implementation:

When the character collides with a surface tagged “Terrain”, we check the normal vector of the terrain’s surface to determine the slope of the surface. If the slope is greater than the character’s slopeLimit, the character’s moveDirection is translated down the slope. Acceleration is applied based on how long the character has been sliding, up to a terminal velocity.

Calculating the surface slope:

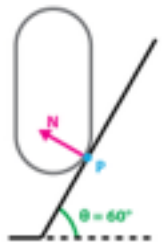


Figure 2. Character standing on steep slope

In Figure 2, Point P is the contact point between the capsule collider of the player and the surface of the terrain. N is the normal vector of the terrain’s surface, which is provided by Unity’s OnControllerColliderHit function. Since the surface normal is normalized, we can determine theta by taking the arc cosine of the normal’s y vector:

```
float slope = Mathf.Acos(hit.normal.y) * Mathf.Rad2Deg;
```

Determining the direction to slide:

The direction in which to slide is derived from the terrain surface’s normal vector. It is the normalized projection of the down vector into the plane defined by the normal. That value is then scaled according to the speed and acceleration of the slide:

```
Vector3 nonUnitAnswer = Vector3.down - Vector3.Dot (Vector3.down, hit.normal) * hit.normal;
slideDirection = Vector3.Normalize (nonUnitAnswer);
slideDirection *= Mathf.Min(terminalVelocity, (acceleration * timeSinceSlideStart * runSpeed));
```

Applying the slide:

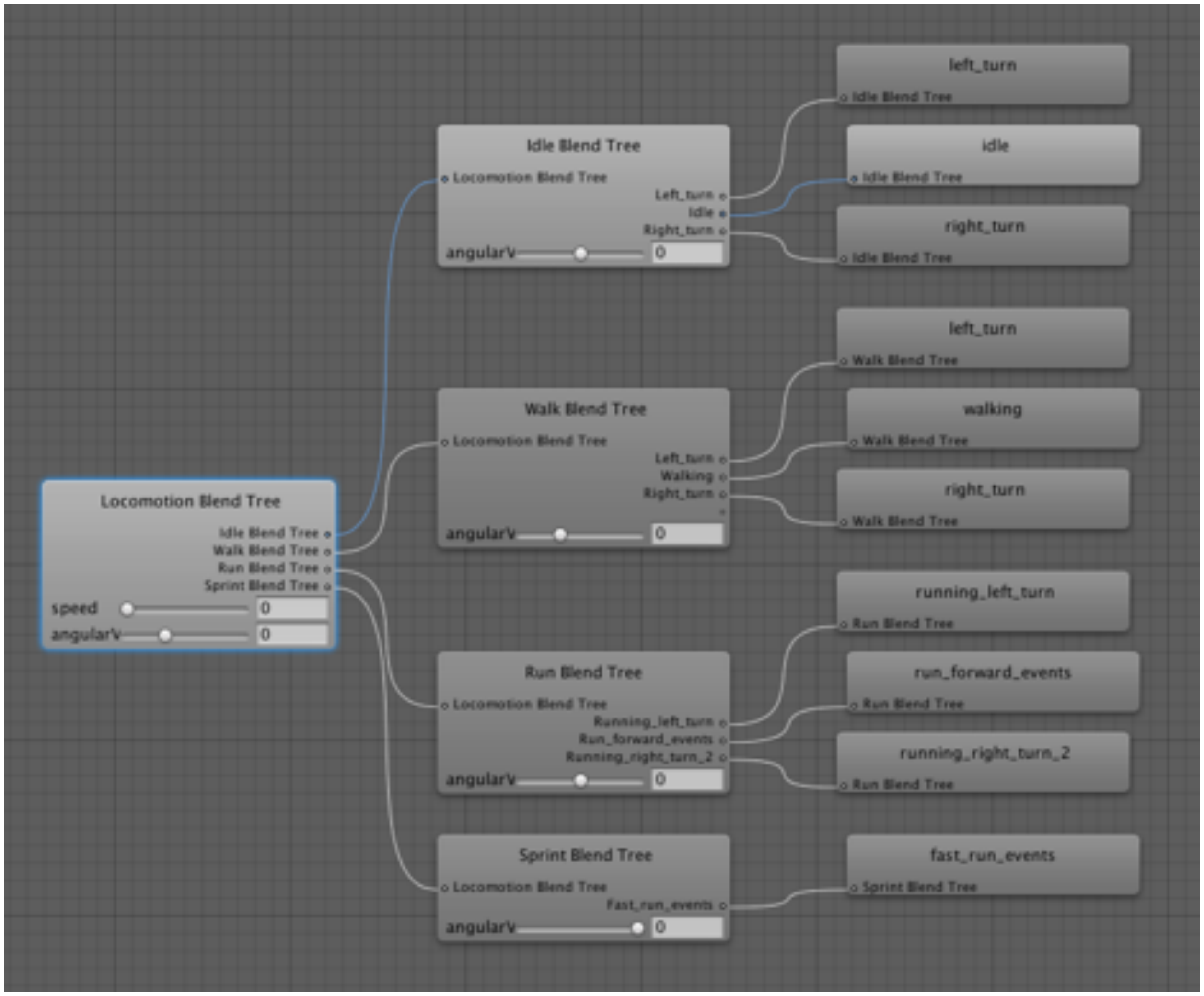
Collisions between the character and terrain are handled by the OnControllerColliderHit function. Because the collision is between a capsule collider and mesh collider and neither has a rigid body, collision information is not available through the character controller’s OnCollisionStay function. Instead, OnControllerColliderHit is triggered during the initial hit. Unfortunately, no “OnControllerColliderStay” function is available. Because of this, we have to set an “isSliding” flag and store the slideDirection during the OnControllerColliderHit function. In the Update loop we check the sliding flag and modify the moveDirection as needed. We also increment the timeSinceSlideStarted value for calculating the accelerated slide speed.

Stopping the slide:

The slide is stopped if the character collides with another terrain surface that has a slope less than slopeLimit.

3. Locomotion Animation

To achieve a realistic walk/run animation at a variety of speeds, a Blend Tree with 4 sub-trees was used to represent speeds from Idle to Sprinting. The sub-trees use a variable for angularVelocity to control whether the animation should be turning left, right, or not at all. The Sprint blend tree does not contain any turning animations because it looked more convincing without them.

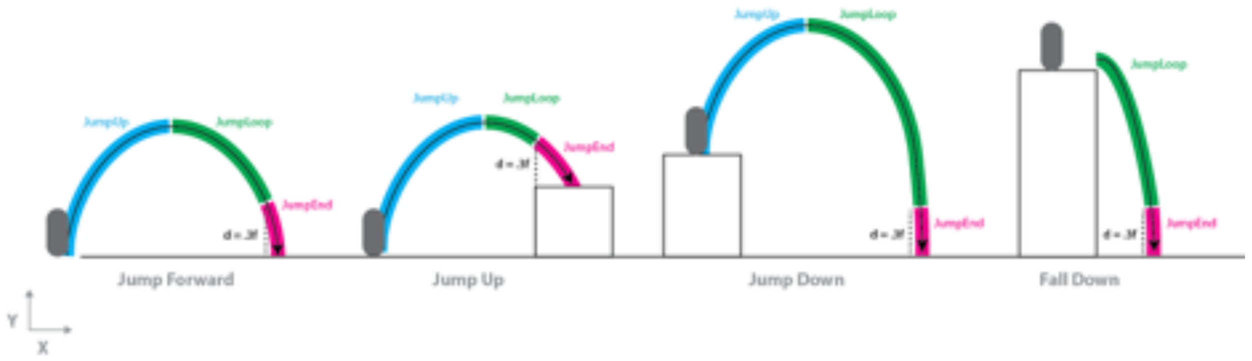


Syncing Footsteps Audio to the Animations

Animation Events were used to play a footstep sound each time a foot touches the ground during an animation. These events were added to the Sprint and Walk non-turning animations. Footstep events were not added to the other locomotion states because when the animations are blended, events from both animations will play. This causes double footstep sounds. By applying events to only one of the animations that will be blended, this is avoided. Despite not being pixel perfect, the end result is convincing enough.

4. Jumping and Falling:

Achieving a realistic-looking jump that can accommodate multiple heights as well as falling off ledges requires using 3 animations. The JumpUp animation includes the initial take-off and holds its ending pose until the jump's apex is reached. The JumpLoop animation is a short looped clip of the character in mid-air. JumpEnd contains the landing and occurs when the character is a certain distance (shown as d = .3 below) from the ground. The value for d is based on the character's height and animation speed and is unique to the model. The character's distance from the ground is determined by a CapsuleCast in the direction of movement. That result from the CapsuleCast is also used to detect if a character runs off a ledge without jumping. In that case, JumpUp is skipped and the JumpLoop is entered.

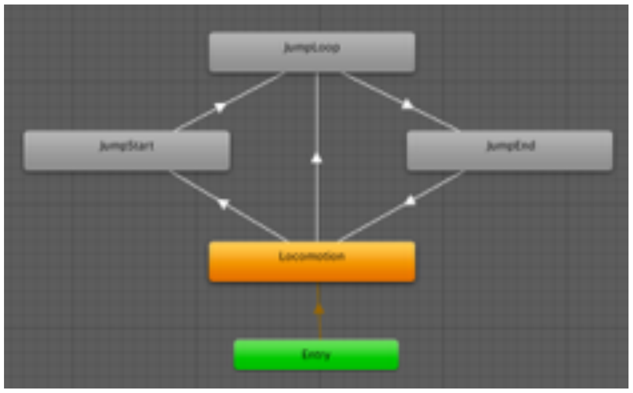


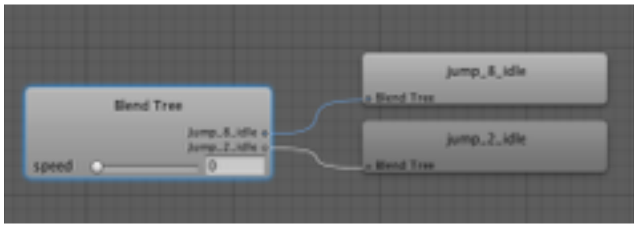
Gravity and Jump Force

Jumping and falling are determined by the verticalVelocity of the character. When the player is on the ground, the verticalVelocity is 0. When the character jumps, its verticalVelocity is set to the jumpVelocity, causing it to continuously move upward. Each update, the verticalVelocity is reduced by gravity, causing the jump height to decrease. Eventually, the jump reaches its apex when the verticalVelocity changes from positive to negative and the character begins to fall. It remains negative until it reaches the ground, when it's set to 0.

Animation State Machine

The animation state machine contains 4 Blend Trees – Locomotion (which contains the Idle state and walking/ running) and the 3 Jump animations. Each Jump Blend Tree contains two motions, a slow jump and a fast jump. The speed variable controls the blending of the animations.





5. Work In Progress

Controller Mechanics:
- Crouch & Crawl
- Double-jump

Camera:
- Avoid obstacles between player and camera
- Right joystick rotational control

Other Features:
- Enemies
- Power-up
- Jetpack Jump

Bugfixes

6. Controls

Gamepad / Keyboard
- Left joystick/WASD: player movement control
- Right joystick/Mouse movement: camera control
- Space/Button 1: jump
- Shift / Button 2: run
- . / Button 3: push / use item(?)
- , / Left joystick depress: crouch
- Esc / Start: pause