Quantic: Advanced Character Controller
Documentation

What is Quantic?

Quantic is an advanced character controller solution for any first-person game. It features multiple movement styles, such as traditional first-person movement, realistic zero gravity movement, and free-roam spectator style movement as well.

We provide you a really smooth, gravity and state based character controller system with dynamic footstep sounds, and head bobbing mechanics.

What does the documentation include?

This document will guide you through the setup process of Quantic, and it gives you information about the different systems and mechanics. You will also find information about how you can use the state system to extend the asset.

Setup Process

The setup process is pretty simple, you only have to assign a few things. Let's split this into two parts, one part for each script. In case you want to skip the setup process, you can use any of the prefabs!

Player Motor Behavior

1. Create an empty GameObject.
2. Add the PlayerMotorBehavior script to the GameObject. This will also add a bunch of other components that the asset will need.
3. Set the GameObject's layer to "Ignore Raycast".
4. Open the "Movement" tab and set the "Crouch Interact Layer" to everything except the "Ignore Raycast" layer.

Player Camera Behavior

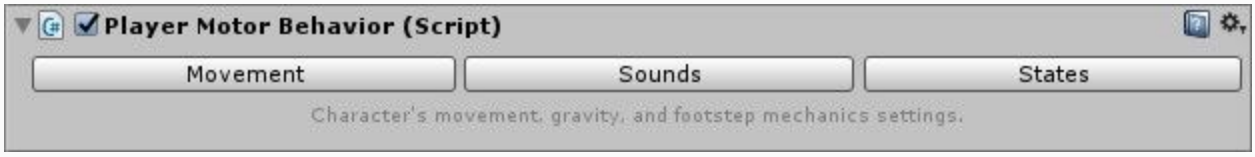
1. Create an empty GameObject as a child of the player's GameObject.
2. Open the "Head Motion" tab and attach the recently made empty GameObject to the "Head Transform" slot.
3. Create a Camera as a child of the head transform object.
4. Open the "Mouse Look" tab and attach the camera to the "Player Camera" slot.

If you are done with the steps mentioned before, you can press the play button and start moving around the scene. The setup literally took 1-2 minutes from scratch, but as I said you can just use any of the prefabs too!

Script Overview (1/3)

Let's take a look at the scripts which run the asset. Let's start with the main scripts.

Player Motor Behavior



This script is responsible for the character's movement, sounds, and states. The movement includes walking, running, jumping, crouching, and the gravity as well. The movement is seamless, so you can go into zero-gravity at any time!

The sounds includes variables for the footsteps, jumping and landing sounds. The states determine what the character can and can't do. More info on this inside "The State System" part of the document.

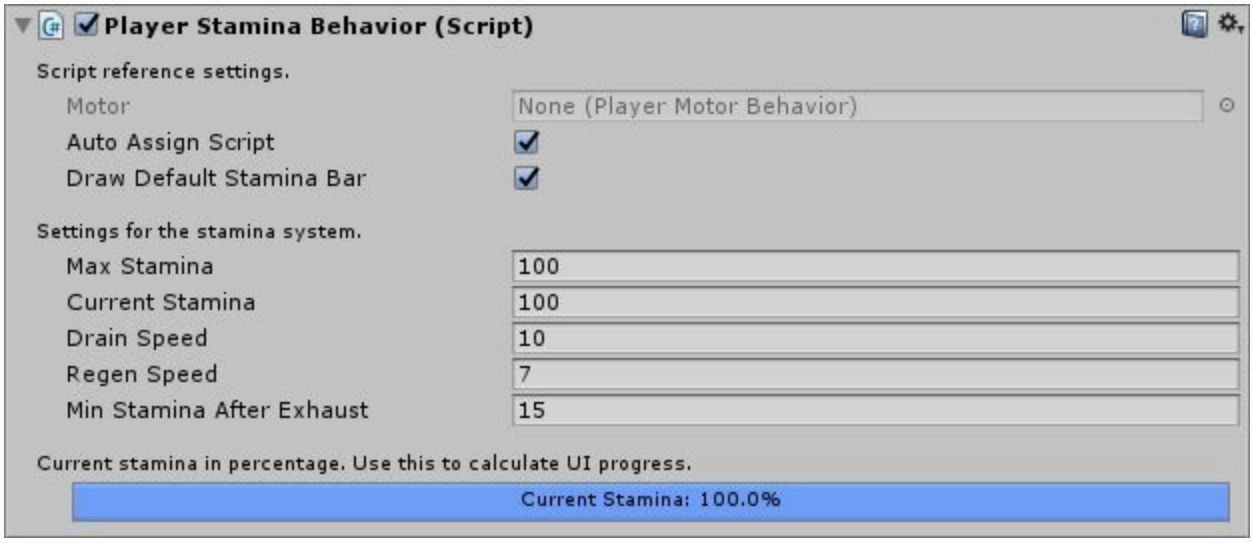
Player Camera Behavior

This script is responsible for the character's camera. It handles the typical mouse look mechanics, and the movement motions such as head bobbing and the landing motion.

Script Overview (2/3)

Let's continue with the addon scripts. There are two scripts currently available.

Player Stamina Behavior



This script has been added in the 1.7 version, and it serves like an addon to the motor script. Attach this script to a GameObject and set the reference to enable it.

Using this script will instantly add a stamina system to the motor script. You can set the maximum stamina, the drain and regeneration speed. If you completely run out of stamina, you will become exhausted.

There is an "isExhausted" passive state which you can use as well.

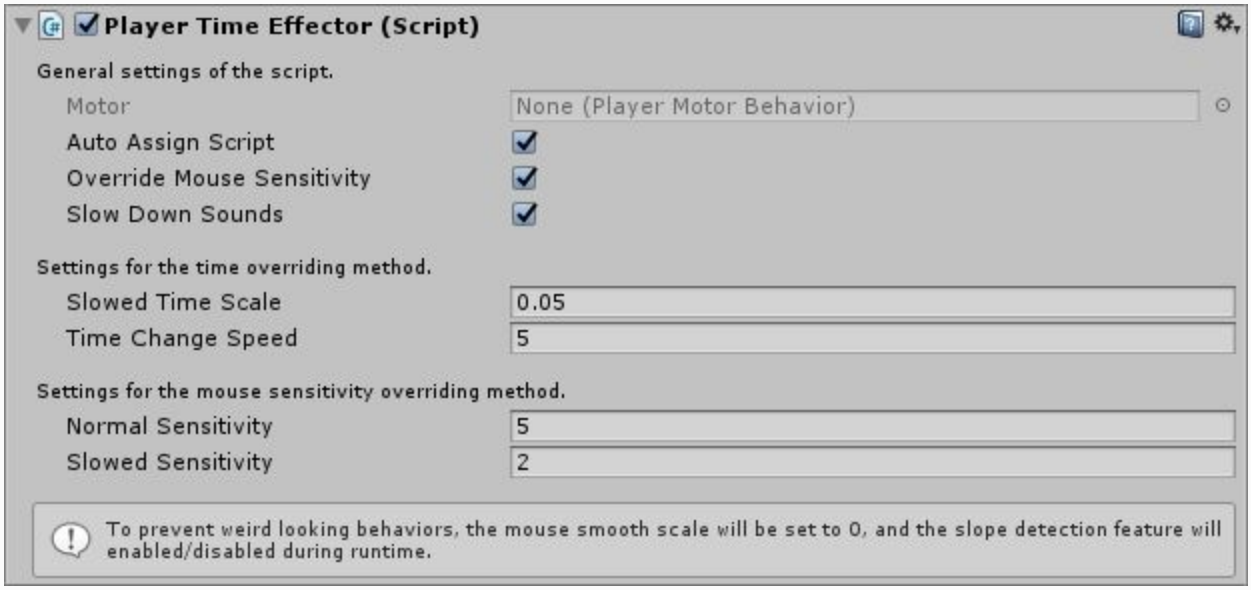
```
//Check if the player is exhausted (true/false).
PlayerStaminaBehavior.isExhausted;
```

For a quick start, you can use the default stamina bar generated by the system, but it's going to be useful only for testing purposes in most cases.

Script Overview (3/3)

Let's end the script overview with the recently added new addon script.

Player Time Effector



This script has been added recently in the 1.8 version. This script recreates the same slow-motion effect as seen in SUPERHOT.

Attach the script to a GameObject and set the reference to enable this addon. If the addon is enabled, it will slow down the time in the game based on the movement of the character.

To avoid weird looking behaviors, the mouse smooth scale will be set to 0 and slope detection feature in the motor script will be enabled/disabled during runtime. This will be invisible to see in the movement though.

The State System

You may be wondering what the hell is this state system which I mentioned before. Quantic features a unique state system which determines what the player can and can't do, and it provides a bunch of information about the character's movement.

There are two types of states: active and passive. Let's see what are the differences between these states.

Active States

These states determine what the character can and can't do. Also, it tells the system what features this character will use. Active states can be modified from the inspector or from scripts at any time. These will affect the character's movement seamlessly.

Passive States

These states are information providers. Passive states are updated by the system so they cannot be modified from the inspector nor from other scripts. These states provide you every information related to the character's movement.

Summary

Using and combining these two state types will give you complete freedom if you want to extend the asset. You can find more information about this in the "Extending The Asset" part of the document.

Extending The Asset

Because of the state system, Quantic can be extended pretty easily so you won't have nightmares about this I guarantee.

Extension Process

Example extension: Allow jumping only if the character is moving.

First, you need a reference to the script to access its variables. We can create a private variable and assign it in the start method.

```
private PlayerMotorBehavior motor;

private void Start()
{
     //Assign the reference to the motor script.
     motor = FindObjectOfType<PlayerMotorBehavior>();
}
```

We can use the update method to only allow jumping if the character is moving. We can take advantage of the state system here.

```
private void Update()
{
     //If we are moving, allow jumping.
     if(motor.isMoving) motor.canJump = true;

     //If we are not moving, disable jumping.
     else if(!motor.isMoving) motor.canJump = false;
}
```

Public Method

You can use this to change the gravity scale with a few other variables. The method can be called from the motor script with a simple reference.

```
UpdateGravity(float newGravityScale, float newMoveSmoothScale, float additionalGravityForce)
```