QUICK CHARACTER

By Kadir Lofca

Welcome to the Quick Character documentation!

Here, you will learn how the system functions and how you can build your characters. Read through the whole documentation to have the best experience using Quick Character!

Contents

1 - File Hierarchy

```
📂Quick Character
├📂Resources
├📂Core
|  ├QuickCharacter.cs
|  ├QuickLibrary.cs
├📂Examples
|  ├📂Animations
|  ├📂Scripts
|  |  ├AdvancedCharacter.cs
|  |  ├SimpleCharacter.cs
|  |  ├SimpleController.cs
|  ├📂Player Input
|  |  ├QuickInputActions.cs
|  |  ├QuickInputActions.inputactions
|  ├AdvancedCharacter.prefab
|  ├SimpleCharacter.prefab
```

📂**Core:** Scripts that are vital for the system.
📂**Examples:** Samples that demonstrate the system.

2 - Style & Conventions

All scripts try to follow the same conventions.

Naming Convention

- camelCase variables.
- SNAKE_CASE constants.
- PascalCase functions.

Commenting

Comments have been added appropriately when they are needed, you will not find a comment for each line.

Function Order

Functions are ordered by how they are referenced and their access specifiers. For example, if functions (private) A and (public) B are called inside function (protected) C, the order of which the functions are written in that class will be: B, A, C.

Code Organization

Quick Character is designed to minimize the amount of scripts needed for character movement. You will find that functions are grouped by `#region`s. All region names are self explanatory.

3 - Overview

Namespaces

Quick Character scripts are in a namespace called QUICK. There is another namespace called QUICK.EXAMPLE which contains demonstrations..

QuickCharacter.cs

Base class for all your character movement scripts. Handles transitioning between movement modes and has various functions which you can use to move the character in different ways. Uses a Rigidbody and CapsuleComponent.

QuickLibrary.cs

Contains types that are used across multiple scripts. This includes types that are critical to QuickCharacter.cs, and other, generic, types.

AdvancedCharacter.cs

A more advanced, but not complicated, demonstration of Quick Character. Derives from QuickCharacter.cs. The character can run, double jump, and wall climb. The character also has camera animations. (inspired by a cyber-ninja in a popular video game.)

SimpleCharacter.cs
A bare-bones demonstration of Quick Character that can walk and jump.

SimpleController.cs
Responsible for taking player input and sending them to a QuickCharacter.cs derived script. Also handles camera controls.
Although its name is similar to SimpleCharacter.cs, this is a generic script that works with both demonstrations (simple and advanced).

QuickInputActions.inputactions
Defines actions the player can input. This is an asset from the new input system you can find in the unity registry from the package manager.
There is a script that uses the same name (QuickInputActions.cs), this script is generated when you save the input actions asset. It is referenced inside SimpleController.cs in order to read player input.

Prefabs
The SimpleCharacter and AdvancedCharacter prefabs are complete characters that you can drop into your scene and start playing. You must import the new input system from the package manager for it to work properly.

4 - How It Works

In its core, Quick Character is a single script that acts as a framework for you to write your character in. In this section, we will learn how the `QuickCharacter` class works.

4.1 - Regions and functions

The class has its functions grouped by `#region`s. Let's see what these regions have in them.

4.1.1 - Character controls

This region contains public functions that are intended to be called by the player to control the character.

4.1.2 - Character physics

This region provides functions for you to call inside your character class (derived from `QuickCharacter`). These functions apply continuous forces to the attached Rigidbody, and so are intended to be called every frame.

4.1.3 - Framework

This region contains functions that run the entire system. The system is built to give you control over the exact physics of the character. You will see many private functions that will run under the hood, and some protected virtual functions that you should override in your class.

5 - Getting Started

In this section, we will create a character using Quick Character.

5.1 - Create character script

Start by creating a new script. At the top of the script, include the `QUICK` namespace. Derive your class from `QuickCharacter` instead of `MonoBehaviour`. You may delete `Update()` and `Start()`.

5.1.1 - Important note

`QuickCharacter` relies on the `FixedUpdate()` and `Awake()` functions. You will not need these functions for your script, however, if you do need to use them for any reason, you must use the `override` keyword, like this:

```
protected override void FixedUpdate()
{
    base.FixedUpdate();
}
```

Tip: when you type `override`, Visual Studio will list all functions that you can override in your script.

5.2 - Override PhysicsUpdate()

Same way as in section 5.1.1, `override` the `PhysicsUpdate()` function. We will apply our character physics here and solve the next movement mode. The result should look like this:

```
protected override MoveMedium PhysicsUpdate()
{
    return base.PhysicsUpdate();
}
```

By default, `PhysicsUpdate()` returns `MoveMedium.ground` or `MoveMedium.air`. If you want to check for additional movement mediums, you may delete the `base.PhysicsUpdate()` call, and replace it with your own code that solves and returns the next movement medium.

5.3 - Applying forces to the Rigidbody

Inside `PhysicsUpdate()`, apply forces to the Rigidbody appropriate for the value of `medium`. You can use if statements, or a switch statement to apply forces per movement medium, it is up to you how you want to apply forces.

5.3.1 - Plug and play physics

You may use the built in "Character Physics" functions inside QuickCharacter.cs to apply forces to the Rigidbody. The built in functions require some parameters, declare them in your class and pass them to the functions.

5.4 - Adding new movement mediums

Add new movement mediums to `enum MoveMedium` inside QuickLibrary.cs. By default, `MoveMedium` has values for ground, air, and wall movement.

5.5 - Player input

Although there is an example in the asset, Quick Character does nothing special relating to player input. You can check out SimpleController.cs to see how I handled it, but ultimately, you must write your own solution for any additional player input.

Your player input controller/manager can communicate with your character using the “Character Control” functions inside QuickCharacter.cs.

5.6 - Done

Your character script is finished! Add some animations and sounds to enhance the experience!

6 - Support

This document was the first line of support, thank you for going through it! If you didn’t find what you are looking for in this document, please DM me on Discord to let me know of your issue. I will do my best to help you out!

My Discord: haftaici#0346