CHARACTER CONTROLLER AND CUSTOM INPUT MANAGER

BY AHMETOVIĆ EMIR

Character Controller and Custom Input Manager package is a commercial release and is not free.

Version: 2.0

FEATURES

This is the perfect Character Controller with a Custom Input Manager for your project. The system has various features.

Camera system features:

- Occlusion detection
- Collision detection
- First Person and Third Person
- Automatically switches from Third Person to First Person if the camera is too close to the player
- Player fade in/out in regards to camera distance (Players materials doesn't have to be transparent shader materials)
- Multi-camera system (The system consist out of 3 cameras: HUD Camera, Near Camera and Far Camera for faster performance and better details)
- Adjustable sensitivity and smooth
- Adjustable calculation level for better performance on various devices

Character Controller features:

- Idle
- Walking
- Running
- Jumping
- Sliding
- Using
- Falling
- Landing
- Animator script is ready for use, you just have to import animations for the player
- Adjustable running and walking speed, jump height, rotation speed and slide angle

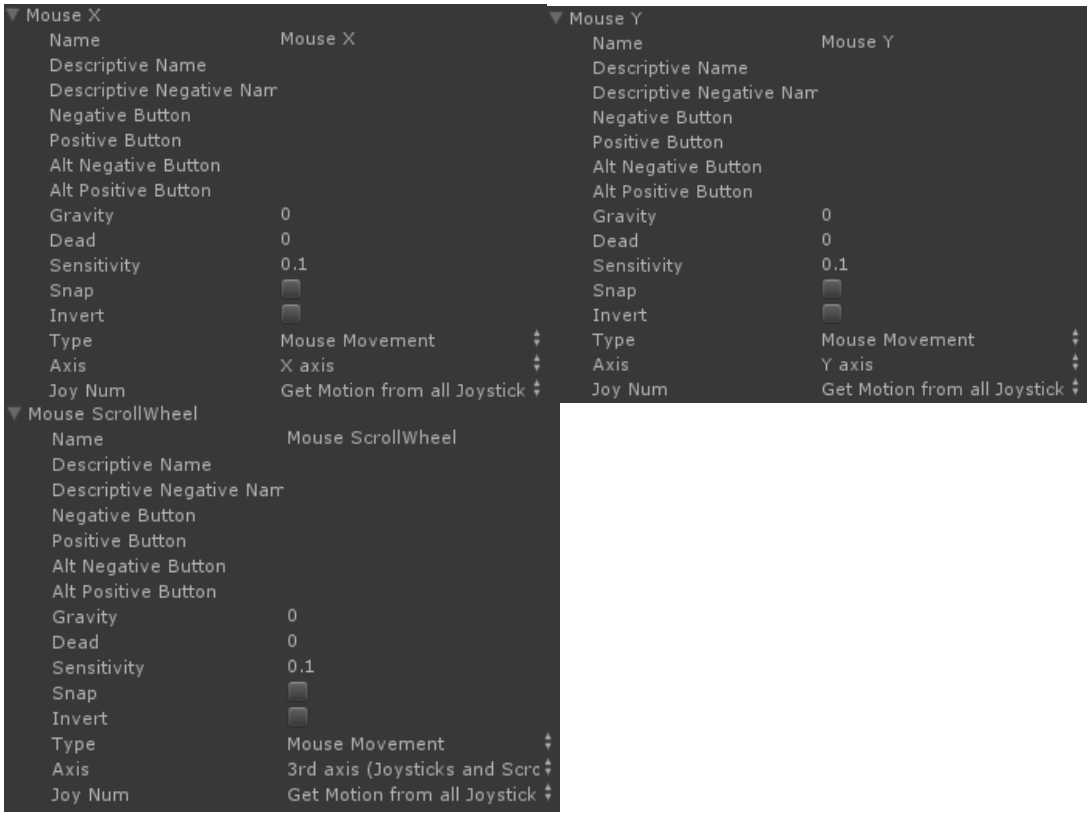
Custom Input Manager features:

- In-game key bind changing
- Key binding supports key modifiers (e.g. LeftShift + W)
- Supports every key from the keyboard
- Can be used for other projects

SETUP GUIDE

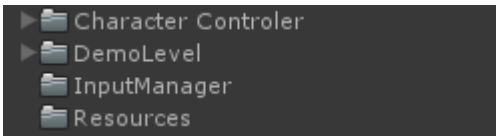
There are just few steps to get your character ready:

1. Before you start, open the standard unity input manager (Edit > Project Settings > Input) and check if you have these three inputs:



   You can delete all other inputs from the default input manager to avoid conflicts with the custom input manager provided within this project.

2. Import the asset from the Asset Store
3. After the import is finished you will have a folder structure in your root folder like this:



   As you can see, the package includes a DemoLevel just for demonstration. You can delete this folder after you have finished with the setup of your character.
4. To import the character to the scene, open the folder with the name „Character Controler“.
5. In this folder you will see a prefab named „Character“. Drag this prefab into your scene. Your character is now ready for use.

IMPORTANT: You do not need a main camera in your scene for the character, so delete all cameras. If you want to add components to your camera do it via scripting through the script TP_Camera. If your character falls through the terrain check the position of the character and try to raise it above the ground.

GETTING STARTED

This package includes a demo level to show how to setup this controller and how to use its functionalities. If you are already familiar to it you can skip this section. Let's load this demo scene to see the controller in action. The scene file is located in DemoLevel > DemoScene. After the scene is loaded you can hit play and test the character.

If you want to enter the Settings menu simply press the Esc key. From this menu you can edit the key bindings, camera and player settings. Try to change some key bindings by clicking on one of them and pressing the desired key to bind. You can notice that you can bind keys with modifiers (CTRL, ALT and SHIFT). These are only the basic setting you can edit. When you build your game, there are much more settings in the inspector panel to fine tune the character controller.

CHARACTER CONTROLLER

When you click on the 'Character' object in the Hierarchy panel you will notice few scripts attached to the character. These scripts provide fields to customize the character.

The TP_Motor script has the following properties:

- **Run speed**
- **Walk speed**
- **Jump speed** – used to set the force of the jump.
- **Rotation speed** – the speed of the rotation when the player turns left or right.
- **Gravity** – used to set the force of the gravity
- **Terminal velocity** – the maximum falling speed.
- **Slide threshold** – the y-component of the normal vector from the ground. If the ground is flat, the y-component is 1. If the slope is greater the y-component is lower. This field defines the slope level at which the character starts sliding.
- **Maximum controllable slide** – below this slope level the character can't control the slide, it just slides down.

CAMERA SYSTEM

This package includes a complete camera system which can be used outside this package too. The system supports occlusion and collision detection whose calculation levels can be adjusted for better performance on various devices. The camera system can be set to first person and third person or to combine both.

MULTI CAMERA SETUP

The system is made up from three cameras: HUD Camera, Near Camera and Far Camera. You can use the HUD Camera to show some specific objects like hands of the player. Objects that are shown in the HUD Camera are rendered last, so they will never go through other object. They will always stay on the top, no matter how close other objects are to the HUD Camera. The Near Camera can be used to show detail objects around the character witch can't be seen from the distance, so the system won't render them in the distance. Objects with lower details should be rendered through the Far Camera. To achieve maximum performance you can make different

layers for object and assign these layers to the cameras culling masks (etc. HUDLayer, DetailLayer, NormalLayer). This should be done in the TP_Camera script in the method AttachCamera() because these three cameras are created dynamically from the code in this method.

FADE PLAYER

Another feature of this camera system is that it fades in/out the object that it is looking at. It has various setting where this feature can be fine-tuned. It's known that Unity supports materials whose alpha can be edited. The shader of these materials must be 'Transparent'. This camera system can fade out objects that are not made out of materials that have the 'Transparent' shader. It uses a custom made mechanism to accomplish this feature.

SCRIPT PROPERTIES

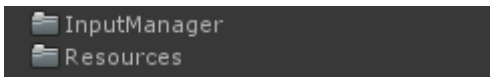
- **Target Look At** – defines the GameObject which will be looked at.
- **Distance** – the starting camera distance.
- **Distance Min** – the minimum distance of the camera from the character.
- **Distance Max** – the maximum distance of the camera from the character.
- **Distance Smooth** – the level of smoothing when the camera changes the distance.
- **Distance Resume Smooth** – the level of smooth when the camera goes back to the original position after it is being occluded and moved to another position.
- **Mouse Sensitivity** – the sensitivity of the mouse movement.
- **Mouse Wheel Sensitivity** – the sensitivity when the camera changes the distance.
- **X_Smooth** – the level of smooth of the mouse movement X-axis.
- **Y_Smooth** – the level of smooth of the mouse movement Y-axis.
- **Y_MinLimit** – the minimum angle the camera can reach on its Y-axis.
- **Y_MaxLimit** - the maximum angle the camera can reach on its Y-axis.
- **Occlusion Distance Step** – when the camera is occluded it is moved forward until it is not occluded. This property defines how much the camera goes forward. For smoother transitions use smaller step. For better performance use greater step.
- **Max Occlusion Check** – the number of occlusion checks in one frame. For smoother transitions use a bigger number of checks. For better performance use a smaller number of checks.
- **Occlusion Check Grid Size** – defines the level of calculations made in one frame. For better performance use a smaller grid size. For better occlusion detection user greater grid size.
- **Near Camera Far Clip Plane** – defines the distance of the Clip Plane between the Near and Far Camera. Used for shrink or expand the region of the Near and Far Camera.
- **Fade Distance** – defines the distance at which the character begins to fade out.
- **Fade Offset** – defines the starting alpha offset, e.g. if you set this parameter to 0.3 the characters starting fade alpha will be 0.7.
- **Target Look At Offset** – if the target look at GameObject is at an undesirable position along the Y-axis you can set the desired offset without changing the GameObject's position by setting this property.

To edit these properties edit them in the TP_Camera Script not in the inspector because this script is created dynamically by the character.

CUSTOM INPUT MANAGER

If you have used the default input manager that unity offers you could notice that it doesn't offer much flexibility. The first drawback is that you can't change key binds during gameplay. That is unacceptable for a wide range of games. The second drawback is that you can't make inputs that are combined with modifiers (CTRL, ALT or SHIFT) unless you hard-code it into your scripts. Consider that every game where you play with a character has the running state bound to LeftShift + W. This Custom Input Manager is developed to resolve these major drawbacks.

After you import the package, two folders from the root folder will be related to the Custom Input Manager:



In the folder *InputManager* are two scripts which do the entire job behind the scene. Don't edit the code in these scripts unless you want to change the behavior of the Custom Input Manager. When you open the *Resources* folder you will notice a prefab called *cInputManager*. If you click on that prefab a script will show up in the inspector panel which has a similar UI to the Unity's default input manager. This script has the same functionality like the default input manager. You don't have to instantiate this prefab into you scene to use the virtual inputs. Just edit everything in the inspector panel and the scripts will do all the other work for you.

ADD NEW VIRTUAL INPUTS

You can add new virtual inputs like you did in the default input manager. To add a new input you have to increase the Size property in the inspector panel. A new virtual input will show on the bottom of the list which you can now edit to fit your needs.

INPUT PROPERTIES

When you edit an input you will notice the following properties for every input:

- **Name** – Every virtual input must have its unique name.
- **Type** – The virtual input can be an Axis or a Button. An Axis virtual input has the exact same behavior like an axis in the default input manager. It has two keys a positive one and a negative one which increases or decreases the value of the input. This type of input can have a value in the range of [-1, 1]. A Button virtual input represents a single key. If the virtual input is set to this type the negative button will not affect the value of the input. Set only the positive button and the alternative positive button for this type of inputs.
- **Positive and Negative Button Name** – This names can be used when you show the key binds to the player (e.g. in the setting menu).
- **Positive and Alt Positive Button** – These buttons are used to increase the value of the virtual input.
- **Negative and Alt Negative Button** – These buttons are used to decrease the value of the virtual input. If the virtual input type is set to Button, these buttons will have no impact to the value of the virtual input.
- **Gravity** – This property defines how fast the value will decrease to zero when you release the virtual input.

- **Sensitivity** – This property defines how fast the value will increase when you press the positive button of the virtual input (or decrease if you press the negative button).
- **Lock** – If you have defined some virtual inputs that shouldn't be changed any more then check this property. Virtual inputs that have this property checked won't show in the setting menu and the player won't be able to change them. For example, if you define a virtual input that opens the Game Menu if pressed and bind the Esc key to it, the player shouldn't be able to change that key. To permit this, just check the Lock property of that virtual input.

USING THE VIRTUAL INPUTS

The Custom Input Manager provides methods that are similar to the default Unity's Input class. If you want to get the value of an Axis use the following method:

```
float verticalAxis = cInput.GetAxis("Vertical");
```

You can notice that the method behaves like the standard *Input.GetAxis()* method. It has one parameter: the Name of the Axis. If you want to check if a virtual input of the type Button is being pressed use the following method:

```
if (cInput.GetButton("Jump"))
{
    Jump();
}
```

Or

```
if (cInput.GetButtonDown("Jump"))
{
    Jump();
}
```

Or to check if it is being released:

```
if (cInput.GetButtonUp("Jump"))
{
    Jump();
}
```

These methods have also one parameter: the Name of the Button. Now comes the interesting part. If you want to change a key bind for a specific virtual input, just use this method:

```
cInput.SetKey("Jump", cInput.ButtonType.AltPositiveButton);
```

After you call the method with these parameters, the next pressed key from your keyboard will be assigned to the Alternative Positive Button of the Jump virtual input. You can call this method whenever you want, in gameplay too. Let's show one more example, if we want to change the Negative Button of the Vertical virtual axis call the method like this:

```
cInput.SetKey("Vertical", cInput.ButtonType.NegativeButton);
```

The first parameter is the Name of the Virtual Key and the second parameter is which button should be affected (positive, negative, alt positive or alt negative). Remember that you can bind keys with modifiers. If you want to bind the LeftShift modifier and the W key into a virtual input

button, call the method and first press and hold the modifier key LeftShift and then press the key W.

If you want to manually add a new key bind use this method:

```
cInput.SetKey("Vertical", "LeftShift+W", cInput.ButtonType.PositiveButton);
```

After you call this method, it will assign a new key bind to the selected virtual input without waiting for a key from the keyboard.

IMPORTANT: When you edit the inputs from the inspector panel you should follow some rules:

- You can't bind two keys together into one virtual button (e.g. Positive Button = A+W)
- You can bind a key with a modifier but the modifier must be on the first place (correct: RightControl+F, incorrect: F+RightControl)
- Don't use spaces in the key bind definitions (correct: RightAlt+T, incorrect: RightAlt + T)
- Don't use a same key binds for more than one virtual key

Script Reference

GetAxis

```
GetAxis (string axisName)
```

Parameters

axisName The name of the axis.

Description

Returns the value of the virtual axis identified by **axisName**. The value will be in the range [-1, 1].

Example

```
float verticalAxis = cInput.GetAxis("Vertical");
```

GetButton

```
GetButton (string buttonName)
```

Parameters

buttonName The name of the button.

Description

Use this to determine if a button is being held down. GetButton returns true repeatedly while the user holds down the key, and returns false if the key is not being pressed. The use of GetButtonDown or GetButtonUp is recommended if you want to trigger an event only once per keypress, e.g., for jumping.

Example

```
if (cInput.GetButton("Jump"))
{
    Jump();
}
```

GetButtonUp

```
GetButtonUp(string buttonName)
```

Parameters

buttonName The name of the button.

Description

Use this to determine if a key has been released. GetButtonUp returns true only once when the key is first released. The use of GetKey is recommended if you want to trigger an event repeatedly while the key is being held down, e.g., for continuous movement.

Example

```
if (cInput.GetButtonUp("Jump"))
{
    Jump();
}
```

GetButtonDown

```
GetButtonDown(string buttonName)
```

Parameters

buttonName The name of the button.

Description

Use this to determine if a key has been pressed. GetButtonDown returns true only once when the key is first pressed down. The use of GetKey is recommended if you want to trigger an event repeatedly while the key is being held down, e.g., for continuous movement.

Example

```
if (cInput.GetButtonDown("Jump"))
{
    Jump();
}
```

SetKey

```
SetKey(string name, string button, ButtonType type)
```

Parameters

name The name of the virtual input.

button The new key bind for the selected virtual input

type The affected button of the selected virtual input. It can have one of the following values: `cInput.ButtonType.PositiveButton`, `cInput.ButtonType.NegativeButton`, `cInput.ButtonType.AltPositiveButton`, `cInput.ButtonType.AltNegativeButton`.

Description

Use this to change a key bind af a virtual key. After you call this method, it will assign a new key bind to the selected virtual input without waiting for a key from the keyboard. Also optionally assigns modifier keys.

Example

```
cInput.SetKey("Vertical", "LeftShift+W", cInput.ButtonType.PositiveButton);
```

SetKey

```
SetKey(string name, ButtonType type)
```

Parameters

name The name of the virtual input.

type The affected button of the selected virtual input. It can have one of the following values: `cInput.ButtonType.PositiveButton`, `cInput.ButtonType.NegativeButton`, `cInput.ButtonType.AltPositiveButton`, `cInput.ButtonType.AltNegativeButton`.

Description

Use this to change a key bind af a virtual key. After you call this method, the next pressed key from your keyboard will be assigned to the selected virtual input. Also optionally assigns modifier keys.

Example

```
cInput.SetKey("Jump", cInput.ButtonType.AltPositiveButton);
```

VALID INPUTS

KEYBOARD INPUTS

Alpha0	F3	KeypadPeriod	Slash
Alpha1	F4	KeypadPlus	Space
Alpha2	F5	LeftAlt	SysReq
Alpha3	F6	LeftApple	Tab
Alpha4	F7	LeftArrow	Underscore
Alpha5	F8	LeftBracket	UpArrow
Alpha6	F9	LeftControl	A
Alpha7	F10	LeftParen	B
Alpha8	F11	LeftShift	C
Alpha9	F12	LeftWindows	D
AltGr	F13	Less	E
Ampersand	F14	Menu	F
Asterisk	F15	Minus	G
At	Greater	Numlock	H
BackQuote	Hash	PageDown	I
Backslash	Help	PageUp	J
Backspace	Home	Pause	K
Break	Insert	Period	L
CapsLock	Keypad0	Plus	M
Caret	Keypad1	Print	N
Clear	Keypad2	Question	O
Colon	Keypad3	Quote	P
Comma	Keypad4	Return	Q
Delete	Keypad5	RightAlt	R
Dollar	Keypad6	RightApple	S
DoubleQuote	Keypad7	RightArrow	T
DownArrow	Keypad8	RightBracket	U
End	Keypad9	RightControl	V
Equals	KeypadDivide	RightParen	W
Escape	KeypadEnter	RightShift	X
Exclaim	KeypadEquals	RightWindows	Y
F1	KeypadMinus	ScrollLock	Z
F2	KeypadMultiply	Semicolon	

MOUSE INPUTS

Mouse0	Mouse4
Mouse1	Mouse5
Mouse2	Mouse6
Mouse3	

You can use these input keys to assign key binds directly from the inspector panel. Remember to follow the rules listed in the Custom Input Manager section.

REVISION HISTORY

Version	Details	Date
1.0	Initial functionalities: - Smooth camera system - Basic movement - Animator script with basic animation methods	2013-10-31
1.1.1	New functionalities: - Turn left and right - Multi camera setup	2013-11-8
2.0	- Added Custom Input Manager New functionalities: - Running state - Settings menu	2014-2-22

CONTACT INFO

If you need any help or have any ideas to upgrade this package or find some bugs feel free to contact me.

E-mail: **a.emir.91@gmail.com**