RPG Character Mecanim Animation Pack ReadMe

Last Updated **2017-10-25**

Hey, first off thanks for purchasing and using this pack! It is highly recommended to watch Unity's Animation Tutorial Videos before using this asset.

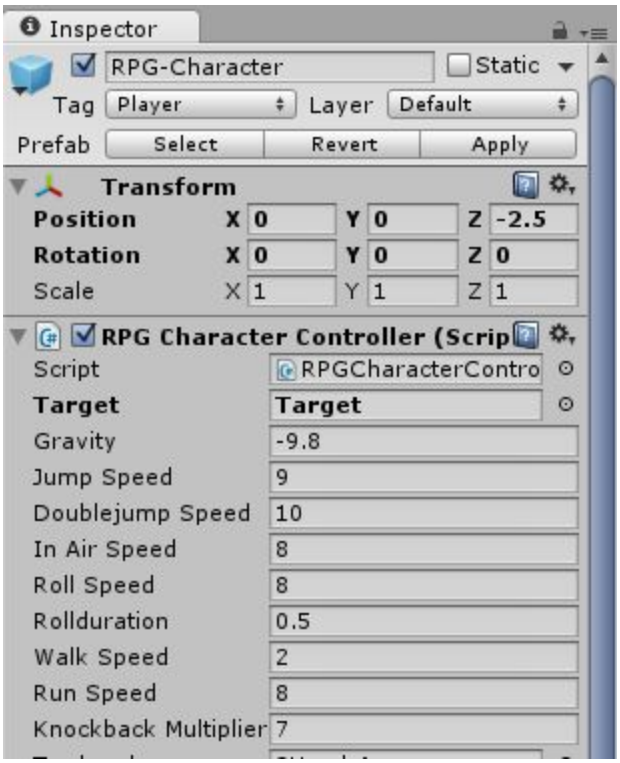
Installation

Before attempting to use the pack, you must first ensure that the tags and inputs are correctly defined. There is an included **InputManager.asset** included in the **InputManager.zip** file that contains all the settings. Copy these to your ProjectSettings folder.

Here's a video showing how to setup your own custom character using the RPG Character animations: https://www.youtube.com/watch?v=I8V6SL7Oy5M

If your character's proportions differ enough from the default character that the model's hands no longer place correctly on weapons properly, there's an included IKHands script that allows using IK to snap the hands to the correct positions. A video here shows its use: https://www.youtube.com/watch?v=o6B8KrfQg9s

The basic movement attributes of the character can be modified via values in the Inspector, as well as many other parameters such as jumping and rolling.



Physics are generally used to move the character, with some attack and special movement animations switching over to Root Motion to let the animations drive the translation so there's minimal feet slipping.

Knockback Multiplier is a force that is directional force that is applied to the character when the GetHit animations are played, and correspond to front, left/right, and back directions.

Control inputs are all handled in **Update()** or referenced methods. **FixedUpdate()** contains all the physics dependant functions. **LateUpdate()** handles all the calls to the Animator. Velocity X and Velocity Z parameters in the Animator are where the local movement values control the animator for directional movement animations.

UpdateMovement() is where all the movement processing of the character is handled, and it uses **CameraRelativeInput()** to make all directions based off the camera. Character faces the movement direction via **RotateTowardsMovementDir(),** which becomes disabled when characters are strafing.

All the jumping methods use **CheckForGrounded()** to determine if the character is on the ground to set whether it can jump, etc. There's a fallDelay variable you can use to prevent the character from transitioning into a fall state when walking over small bumps, or walking down declines.

```
if(canMove && !i
{
    MovementInput
}
Rolling();
Jumping();
Blocking();
if(Input.GetButt
```

If you wish to remove Jumping, Rolling, or Blocking from your character's actions, you can remove these code lines. AirControl can also be removed by deleting **AirControl().**

Rolling() allows freedom of movement when rolling, and then picks the closest corresponding rolling animation to play. The GUI commands for rolling are based off the direction the character is facing, and you can use these if you don't wish to allow rolling in any direction.

Coroutine **_LockMovementAndAttack()** is used by attacks and other action animations, which locks input and movement, while also enabling Root Motion on the Animator.

Coroutine **_SwitchWeapons()** handles all the weapon switching logic and is based off a weapon integer value, and also off leftWeapon,rightWeapon, and LeftRight variables in the Animator.

```
//0 = No side
//1 = Left
//2 = Right
//3 = Dual
//weaponNumber 0 = Unarmed
//weaponNumber 1 = 2H Sword
//weaponNumber 2 = 2H Spear
//weaponNumber 3 = 2H Axe
//weaponNumber 4 = 2H Bow
//weaponNumber 5 = 2H Crowwbow
//weaponNumber 6 = 2H Staff
//weaponNumber 7 = Shield
//weaponNumber 8 = L Sword
//weaponNumber 9 = R Sword
//weaponNumber 10 = L Mace
//weaponNumber 11 = R Mace
//weaponNumber 12 = L Dagger
//weaponNumber 13 = R Dagger
//weaponNumber 14 = L Item
//weaponNumber 15 = R Item
//weaponNumber 16 = L Pistol
//weaponNumber 17 = R Pistol
//weaponNumber 18 = Rifle
//weaponNumber 19 == Right Spear
//weaponNumber 20 == 2H Club
```

Directional Aiming is handled by the following variables which are updated in the Animator via the DirectionalAiming() method:

```
public float aimHorizontal;
public float aimVertical;
public float bowPull;
```

All the iputs are handled in Inputs() to make it easy to swap in your own control scheme:

```
	void Inputs(){
		//Input abstraction for easier asset updates using outside
control schemes
		inputJump = Input.GetButtonDown("Jump");
		inputLightHit = Input.GetButtonDown("LightHit");
	}
```

Note that there are animation events for all animations, and you'll need methods in a script attached to the same component as the Animator otherwise you'll get warnings.

```
	//Placeholder functions for Animation events
	public void Hit(){
	}
	public void Shoot(){
	}
	public void FootR(){
	}
	public void FootL(){
	}
	public void Land(){
	}
	public void WeaponSwitch(){
	}
```

Any questions about the Pack, please Email.

Thank you!