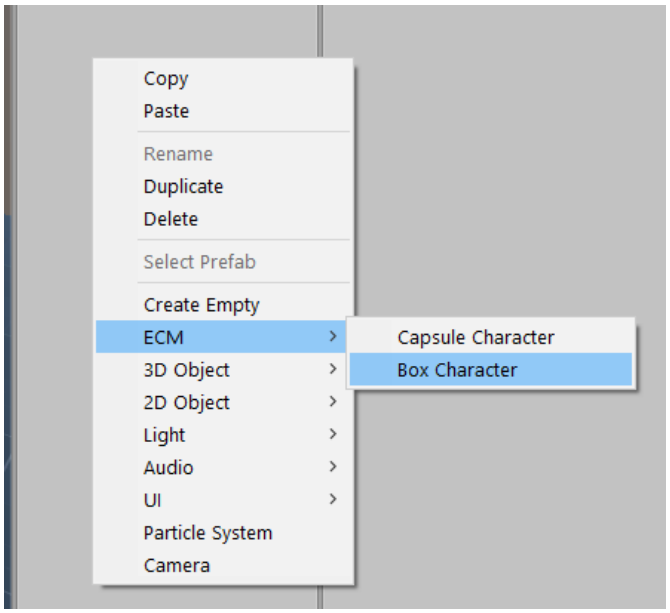
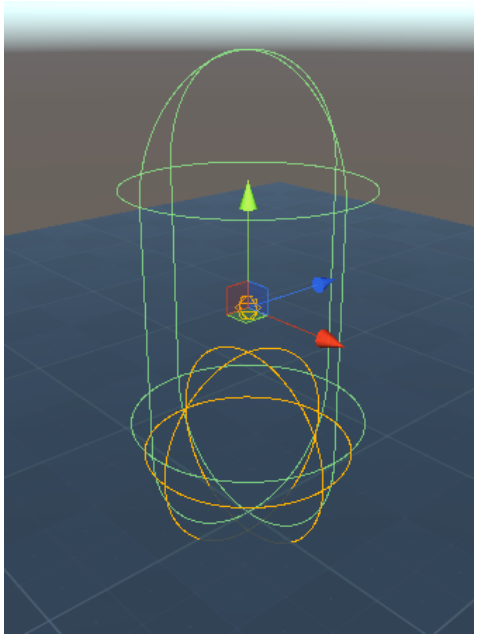
Quick-start guide

1. Import the Easy Character Movement package into your project.

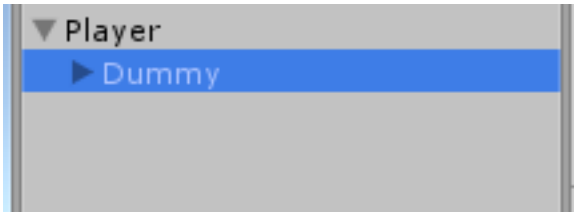
2. Right-click on the hierarchy window to open the creation dialog and select the ECM option, then choose the type of character you want to create.



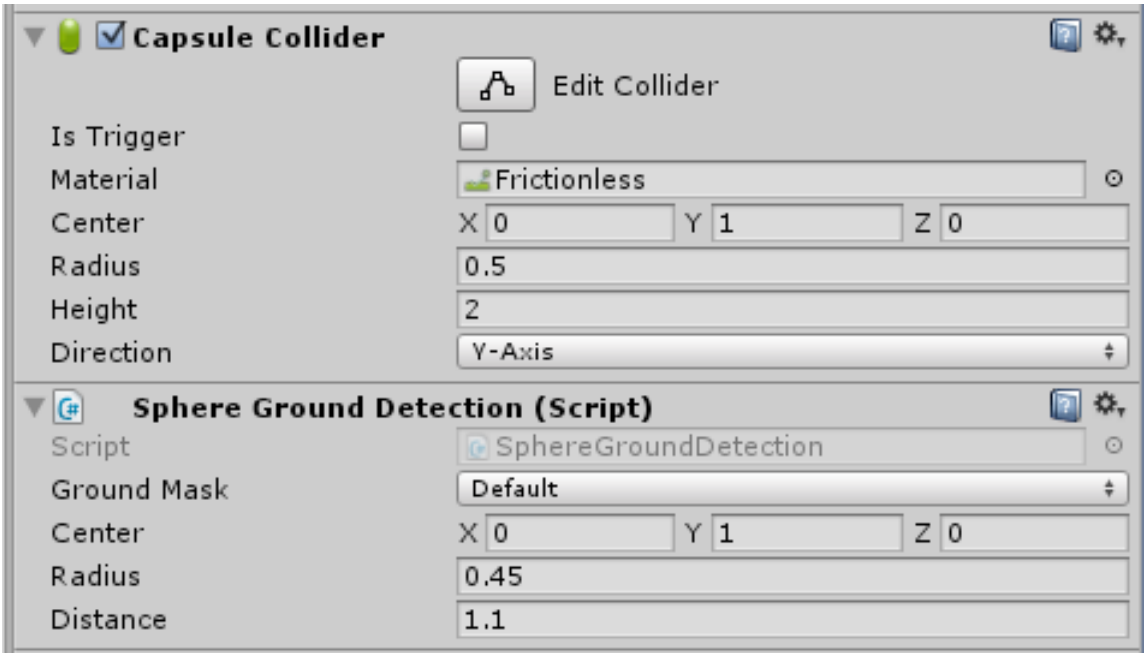
3. It will create an “empty” (no visual representation) character named “ECM_CapsuleCharacter” or “ECM_BoxCharacter” based on your selection. Make sure its origin is at 0,0,0. This will save troubles when parenting.



4. Parent your character model (character's visual representation) to this new game object.



5. Adjust the CapsuleCollider and SphereGroundDetection component to represent your character's model volume.



6. Done! You are ready to move around using the keyboard.

Foreword

Thank you for purchasing Easy Character Movement!

I've developed Easy Character Movement as part of my currently in development platformer game. In the end, I was so happy with the results that I decided to share it with the great unity community.

I sincerely hope this help you to make awesome games and have fun while doing it!

Support

I’m a solo developer and your feedback and support is greatly appreciated!

If you have any comments, need some support or have a feature you would like to see added, please don’t hesitate to email me at **ogracian@gmail.com** I’ll be happy to help you.

In order to get customer support, please include the invoice number for the purchase of the Asset from the Unity asset store.

Best Regards,
Oscar

Overview

Easy Character Movement is a powerful yet incredible easy to use Physics-based character controller designed for vertically-oriented characters.

It can be used for any character, from NPCs to Enemies, to Players, and for a wide range of games like, platformer, first person, third person, adventure, point and click, and more!

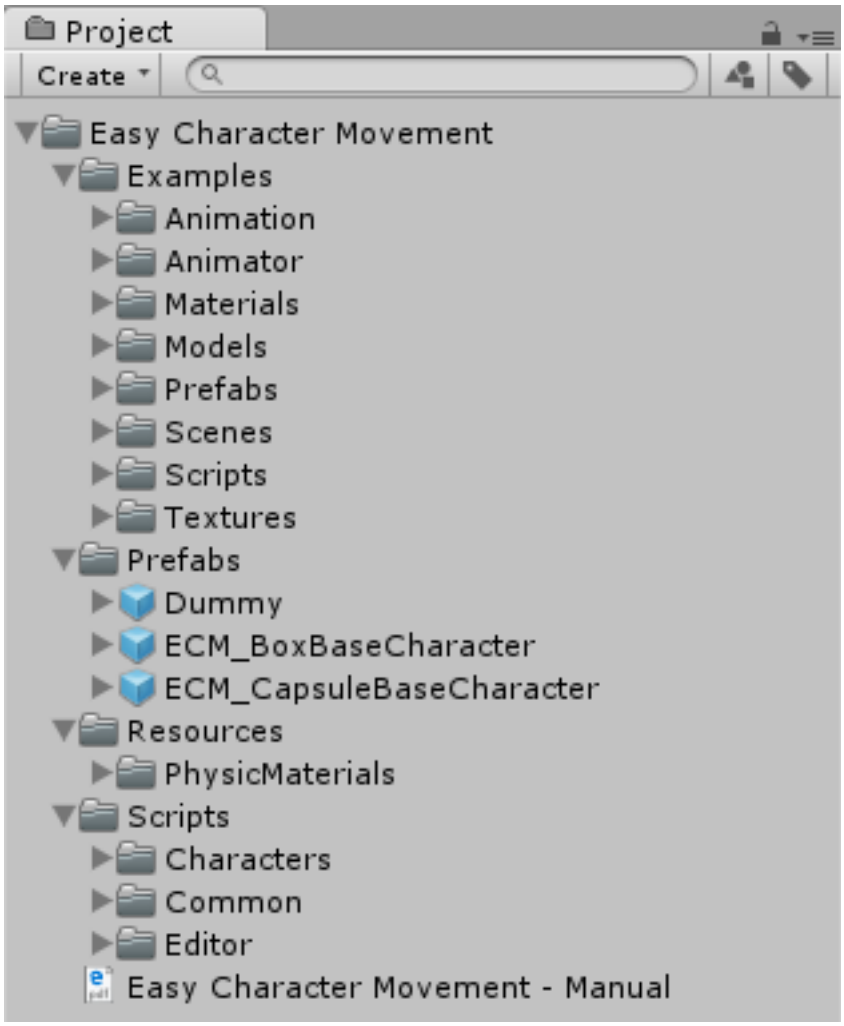
Easy Character Movement was designed from the ground up to be as efficient and flexible as possible with zero allocations after initialization. As a result, it runs great on all platforms including mobile.

If you looking for an easy, efficient and flexible character controller for your next project, please let Easy Character Movement be there for you.

Features

- Rigidbody-based character controller.
- Move and Rotate on dynamic platforms.
- Stay still on slopes, no more sliding down.
- Keep same speed on straight segments and slopes.
- Slide on steep slopes.
- Orient to ground slopes.
- Supports Nav Mesh Agents.
- Supports root-motion.
- Easy integration into existing projects.
- Fully commented C# source code. Clear, readable and easy to modify.
- Mobile friendly.
- Garbage-Collector friendly.
- and more!

Package Contents



Examples
This contains the assets required for the example / demo scenes. This is not required and should be removed when importing ECM into your projects.

Prefabs
This contains two supplied base character's prefabs, one for a Capsule based character and other for a Box based character and a dummy model to represent a character.

Resources
Contains the frictionless physical material required for the character's collider.

Scripts
Contains the C# source code of the Easy Character Movement package.

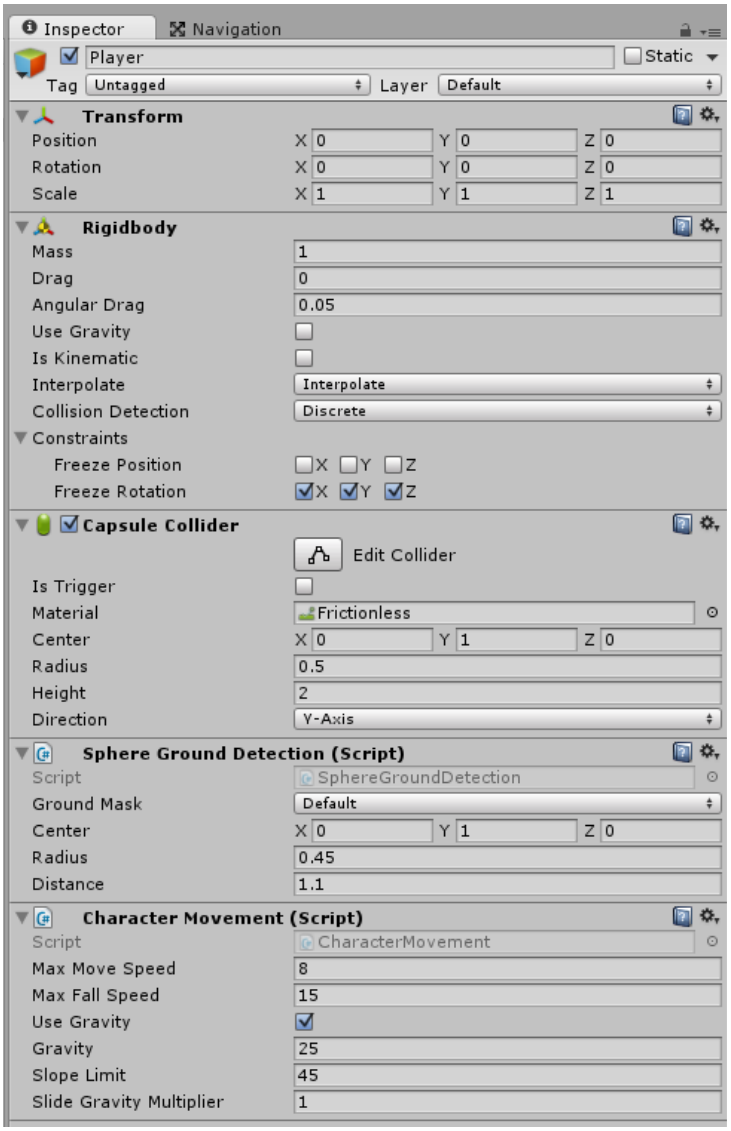
Description

Easy Character Movement follows the ***single responsibility principle*** where a given class should be, ideally, responsible for just one task. Thus, the Easy Character Movement system is composed of several classes with specific tasks, being the most important of these, the ***CharacterMovement*** component.

The main responsibility of ***CharacterMovement*** component as its name suggests, is to actually move and rotate the character, so it its responsible of apply forces, impulses, custom gravity and more, to fulfill its task.

ECM has been developed to be easy and flexible to use, however in order to operate the ***CharacterMovement*** requires a Rigidbody, and GroundDetection component (Box, Sphere or Custom) to be in the same GameObject where ***CharacterMovement*** is.

Here is a suggested set of components your characters should follow:



Worth note that you can use any GroundDetection (Box, Sphere or Custom) component no matter how you represent your character physical colliders, for example is perfectly fine to use CapsuleCollider for your character's physical representation, and a BoxGroundDetection component or a custom one is required / desired.

Along the ***CharacterMovement*** component, is the ***GroundDetection*** component (Sphere and Box included), which performs the ground detection and supply this information to the ***CharacterMovement*** component.

OrientModelToGround is an optional helper component, is responsible to orient a model to match the normal of the surface he is standing on. It will use the ground information supplied by ***GroundDetection***.

Above these components, is the controller (included several), which responsibility is to tell ***CharacterMovement*** to move the character in response to user input, AI, etc.

Worth note, that the ***CharacterMovement*** component by itself will do nothing, you must call its `Move` method in order to actually move the character.

Components

Ground Detection

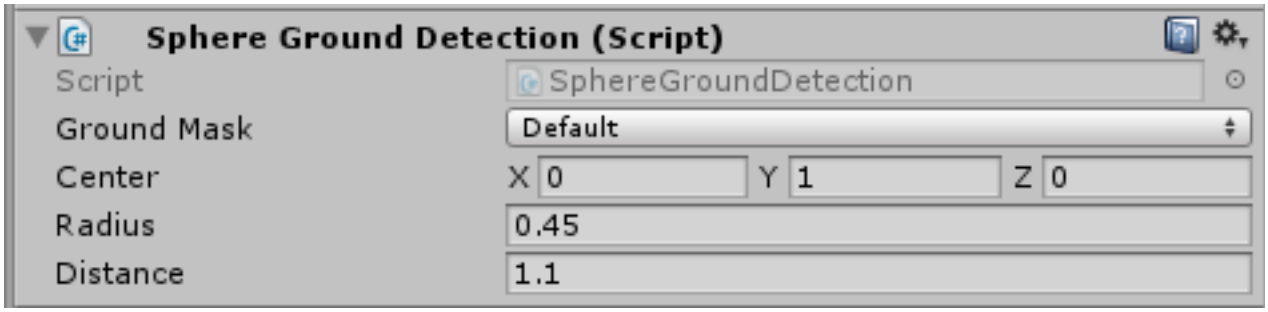
This is an abstract class, its the responsible of perform ground detection, and supply information about the "ground" such as the ground point (where characters touch the ground), the surface normal, and more.

This class can to be extended to perform your custom ground detection method if desired or required.

Included are two custom ground detection methods, one uses a Sphere to represent the character's feet, and a second one which uses a Box to represent the character's feet. This box can be axis-aligned, or follows character's rotation.

Sphere Ground Detection

Inherits from the abstract GroundDetection class and extend it to perform ground detection using a SphereCast.



Ground Mask
Layers to be considered as ground (walkable).

Center
The character's center position in object's local space (relative to its pivot).
E.g. If your character's height is 2 units, set this to 0.0f, 1.0f, 0.0f.

This determines the cast starting point.

Radius
The bounding volume radius. This value should be the radius of the "feet" or bottom of the actor, a typical value should be about 90-95% of your collider's radius or less.

Distance
Determines the cast distance. A typical value should be about ~10% more than your collider's half height. E.g. If your character's height is 2 units, you should set this to ~1.1 units.

Increase it if you loose ground often.

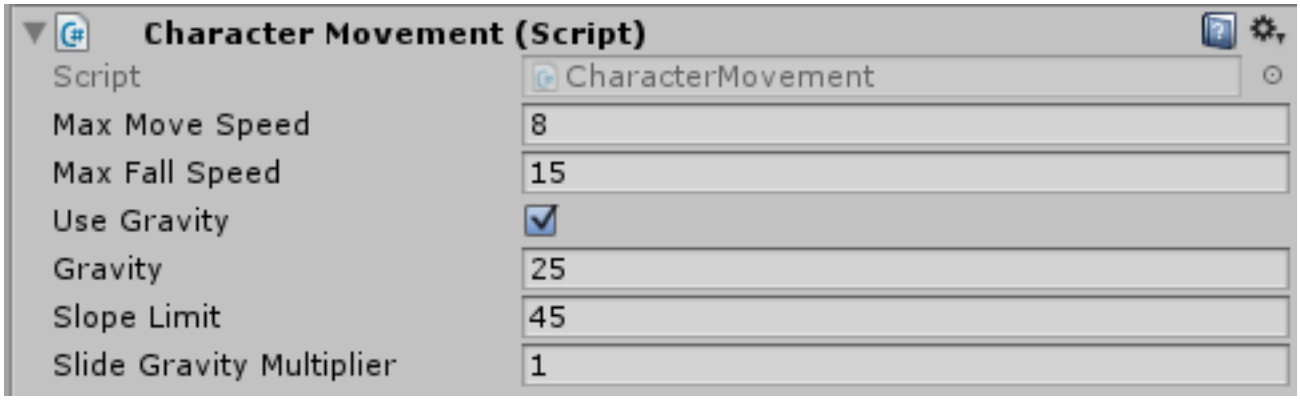
Box Ground Detection

Inherits from the abstract GroundDetection class and extend it to perform ground detection using a BoxCast.

Axis Aligned
Determines if the box is axis-aligned or should follows the character's rotation.

Character Movement

Perform full-body character movement and rotation, is the responsible of actually move and rotate the character based on the information passed by the controller.



Max Move Speed
The maximum speed this character can move, including movement from external forces like sliding, collisions, etc. Effective terminal velocity along XZ plane.

Max Fall Speed
The maximum falling speed. Effective terminal velocity along Y axis.

Use Gravity
Enable / disable Character's custom gravity. If enabled, the character will be affected by its custom gravity force.

Gravity
The amount of gravity to be applied to this character. We apply gravity manually for more tuning control.

Slope Limit
The maximum angle (in degrees) the slope (under the actor) needs to be before the character starts to slide.

Slide Gravity Multiplier
The percentage of gravity that will be applied to the slide.

Orient Model to Ground (Optional)

Helper component used to orient a model to ground. This must be attached to the game object with ***CharacterMovement*** component, and the model to orient must be a child of this.

Orient to Ground
Determines if the model transform will change its up vector to match the ground normal.

Model Transform
The transform to be aligned to ground. E.g. your child character model transform.

Min Angle
Minimum slope angle (in degrees) to cause an orientation change.

Rotation Speed
Maximum turning speed (in deg/s).

Controllers

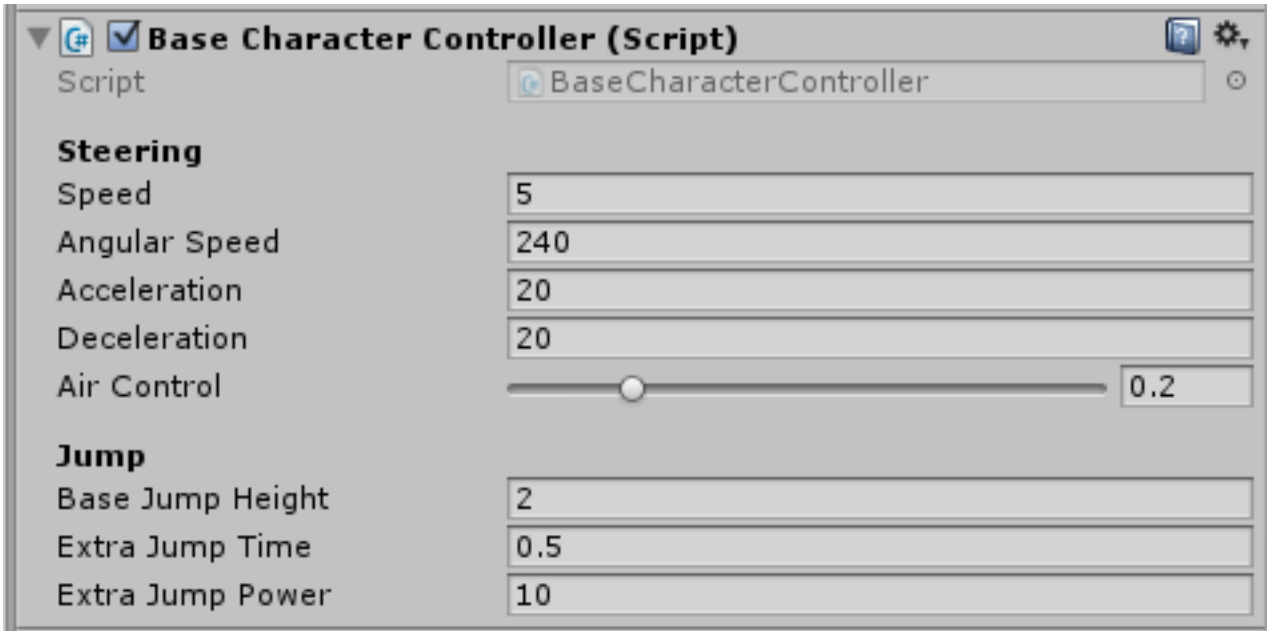
The responsibility of the controller, as stated above, is to determines how the Character should be moved, such as in response from user input, AI, animation, etc. and pass this information to the ***CharacterMovement*** component, which in response will perform the movement.

I've included several controllers to illustrate how easy and flexible is the ***CharacterMovement*** component to support different movements and how to interact with other Unity's components such as ***Animator***, ***NavMeshAgent***, etc.

Remember, ***these controllers are optional,*** but should serve you as a starting point to create your own Controllers to control your character however you want, because ultimately no one knows your game better than you.

Base Character Controller

This is the default character controller and base of the other controllers I've included. It handle keyboard input, and allows for a variable height jump. Its suggested to use this a base to crate your custom controllers, however it is not required.



Speed
Maximum movement speed (in m/s).

Angular Speed
Maximum turning speed (in deg/s).

Acceleration
The rate of change of velocity.

Deceleration
The rate at which the character's slows down.

Air Control
When not grounded, the amount of lateral movement control available to the character.
0 == no control, 1 == full control. Defaults to 0.2.

Base Jump Height
The initial jump height (in meters).

Extra Jump Time
The extra jump time (E.g. holding jump button) in seconds.

Extra Jump Power
Acceleration while jump button is held down, given in meters / sec^2.

Platformer Character Controller

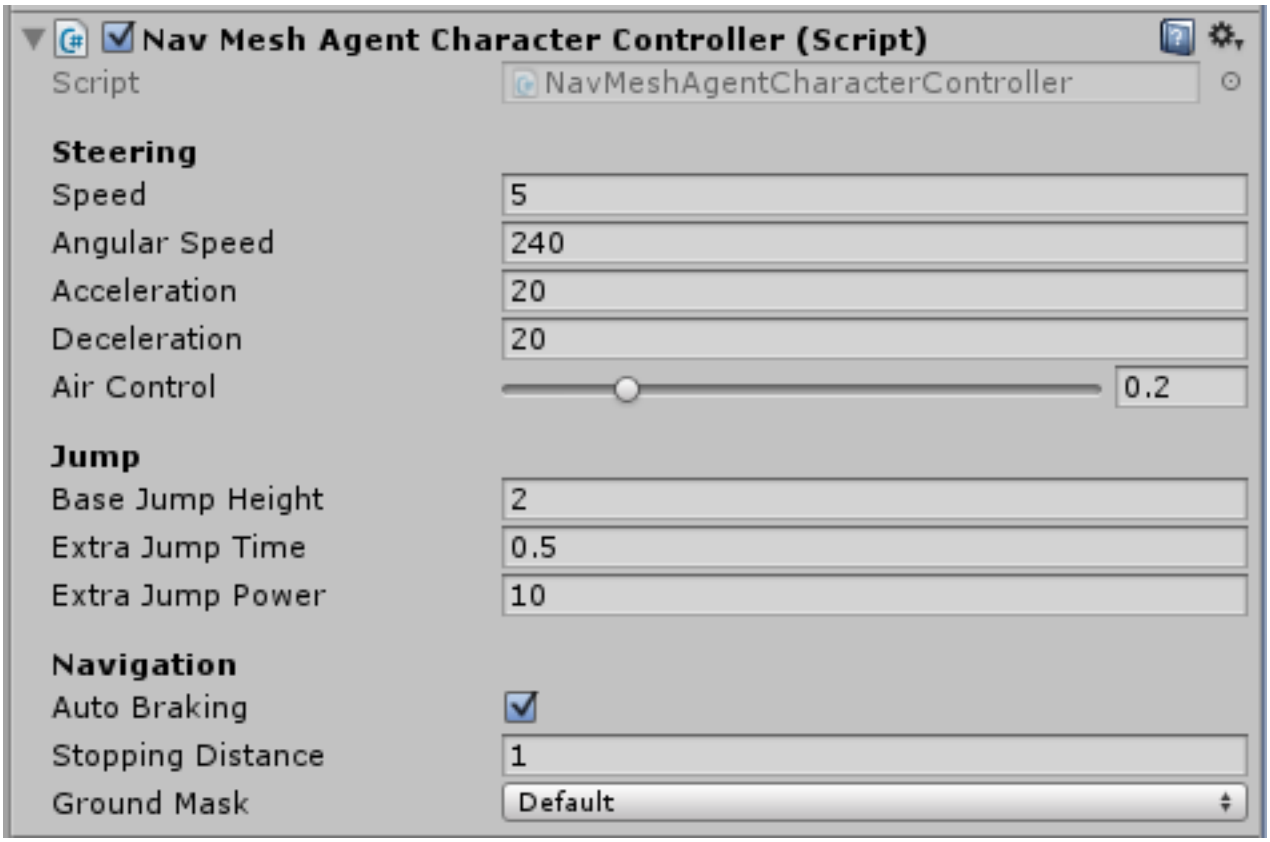
Inherits from Base Character Controller and extends it to allow a variable height double jump.

It have the same fields as Base Character Controller.

NavMeshAgent Character Controller

Inherits from BaseCharacterController and extends it to control a NavMeshAgent and intelligently move in response to mouse right click (Click to move).

Requires a NavMeshAgent attached to the game object with CharacterMovement component.



Auto Braking
Should the agent brake automatically to avoid overshooting the destination point?

Stopping Distance
Stop within this distance from the target position.

Ground Mask
Layers to be considered as ground (walkable).Used to find mouse click position.

Animator Character Controller

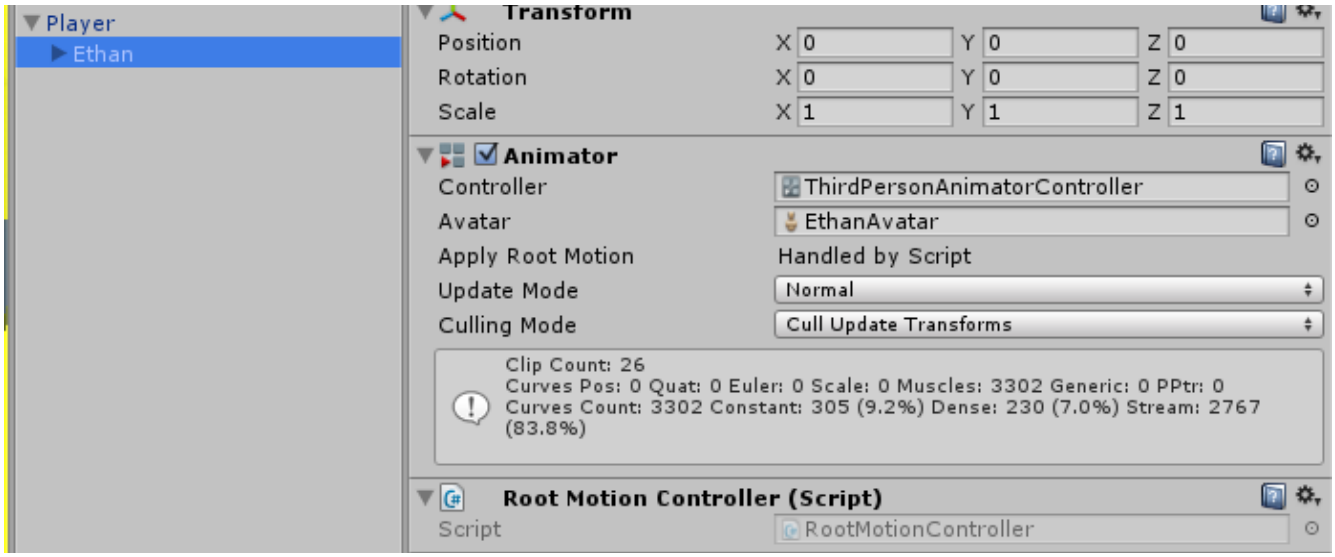
Inherits from Base Character Controller and extends it to animate a character and move with or without root motion.

In order to use root motion, You need to attach a RootMotionController to the Animator game object.

Use Root Motion
Should the character be moved using the animation velocity vector?

Root Motion Controller

Helper component used to supply ***Animator*** root-motion velocity vector (`animVelocity`). It must be attached to the game object with the ***Animator*** component.



FPS Character Controller

Inherits from BaseCharacterController and extends it allow a FPS control, where you moves relative to camera's view direction at different speeds (forward, backward, strafe) based on input and use mouse look around.

Prefabs

You can find a set of prefabs (one for each Controller) which you should use as a starting point or as help if you are in doubt about the hierarchy your characters should follow.

The prefabs live in the Examples/Prefabs directory.

Code Reference

This section will guide your through the public classes and members to use in your scripts.

Character Movement

Properties

```
public float maxMoveSpeed
```

The maximum speed this character can move, including movement from external forces like sliding, collisions, etc. Effective terminal velocity along XZ plane.

```
public float maxFallSpeed
```

The maximum falling speed (effective terminal velocity while falling).

```
public bool useGravity
```

Enable / disable character's custom gravity. If enabled the character will be affected by its custom gravity force.

```
public float gravity
```

The amount of gravity applied to this character when useGravity == true. We apply gravity manually for more tuning control.

```
public float slopeLimit
```

The maximum angle (in degrees) the slope (under the actor) needs to be before the character starts to slide.

```
public float slideGravityMultiplier
```

The percentage of gravity that will be applied to the slide.

```
public float slideGravity (Read Only)
```

The amount of gravity to apply when sliding.

```
public float slopeAngle (Read Only)
```

The slope angle (in degrees) the character is standing on.

```
public bool isValidSlope (Read Only)
```

Is a valid slope to walk without slide?

```
public Vector3 platformVelocity (Read Only)
```

The velocity of the platform the character is standing on, zero (Vector3.zero) if not on a platform. Worth note that moving and rotating platforms, these must be a kinematic rigidbody.

```
public Vector3 velocity
```

The velocity vector of the character.

```
public float forwardSpeed (Read Only)
```

The character forward speed (along its view direction).

```
public Quaternion rotation
```

The character's current rotation. Setting it comply with the Rigidbody's interpolation setting.

Methods

```
public void Rotate(Vector3 direction, float angularSpeed, bool onlyXZ = true)
```

Rotates the character to face the given direction.

`direction`
The target direction vector.

`angularSpeed`
Maximum turning speed in (deg/s).

`onlyXZ`
Should it be restricted to XZ only.

```
public void ApplyForce(Vector3 force, ForceMode forceMode = ForceMode.Force)
```

Apply a force to the character's rigidbody.

`force`
The force to be applied.

`forceMode`
Option for how to apply the force.

```
public void ApplyVerticalImpulse(float impulse)
```

Apply a vertical impulse (along world's up vector). E.g. use this to make character jump.

`impulse`
Option for how to apply the force.

```
public void ApplyDrag(float drag, bool onlyXZ = true)
```

Apply a drag to character, an opposing force that scales with current velocity. Drag reduces the effective maximum speed of the character.

`drag`
The amount of drag to be applied.

`onlyXZ`
Should it be restricted to XZ plane.

```
public void Move(Vector3 desiredVelocity, float acceleration, float deceleration, bool onlyXZ = true)
```

Perform character's movement. It will apply our custom gravity if useGravity == true. It must be called in FixedUpdate method.

`desiredVelocity`
Target velocity vector.

`acceleration`
The rate of change of velocity.

`deceleration`
The rate at which the character's slows down.

`onlyXZ`
Is the movement restricted to XZ plane.

GroundDetection

Properties

```
public Vector3 center
```

The character's center position in object’s local space (relative to its pivot).

```
public float radius
```

The bounding volume radius.

```
public float distance
```

Determines the cast distance. Increase it if you loose ground often.

```
public LayerMask groundMask
```

Layers to be considered as ground (walkable).

```
public bool isGrounded (Read Only)
```

Is this character standing on "ground".

```
public Vector3 groundPoint (Read Only)
```

The impact point in world space where the cast hit the collider. If the character is not on ground, it represent a point at character's base.

```
public Vector3 groundNormal (Read Only)
```

The normal of the surface the cast hit. If the character is not grounded, it will point up (Vector3.up).

```
public float groundDistance (Read Only)
```

The distance from the ray's origin to the impact point.

```
public Collider groundCollider (Read Only)
```

The Collider that was hit. This property is null if the cast hit nothing (not grounded) and not-null if it hit a Collider.

```
public LayerMask groundMask
```

Layers to be considered as ground (walkable).

Methods

```
public abstract bool DetectGround();
```

perform ground detection. Returns true if grounded, false if not.

Orient Model to Ground

Properties

```
public bool orientToGround (Read Only)
```

Determines if the character will change its up vector to match the ground’s normal.

```
public float minAngle
```

Minimum slope angle (in degrees) to cause an orientation change. If the slope angle is less than this, the character will keep its current orientation.

```
public float rotationSpeed
```

Maximum turning speed (in deg/s).

Root Motion Controller

```
public Vector3 animVelocity (Read Only)
```

The animation velocity vector (root-motion).