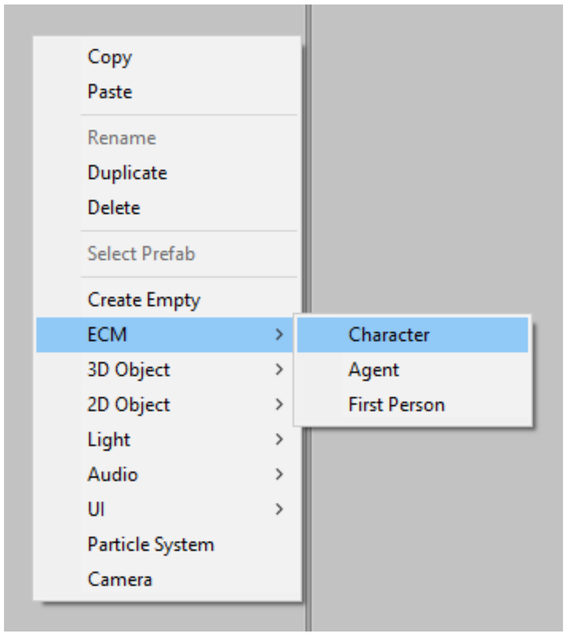
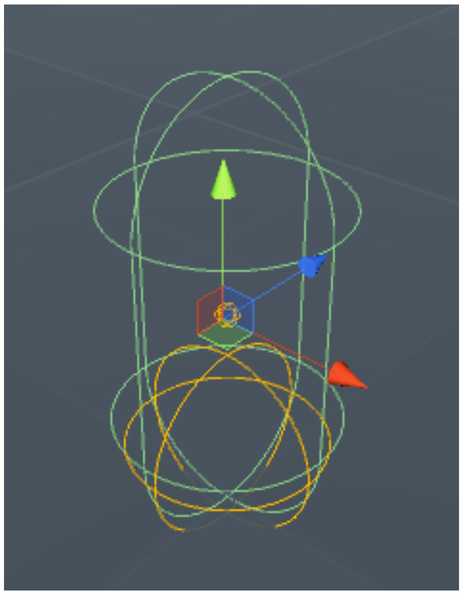
Quick-start guide

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

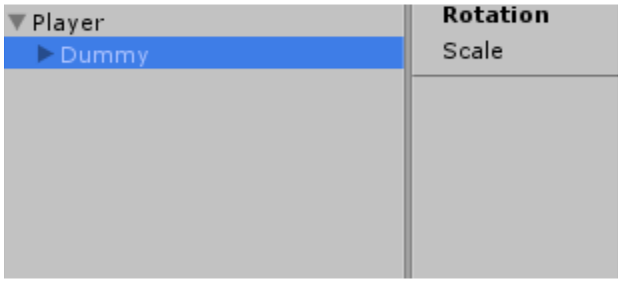
2. Right-click on the hierarchy window to open the creation dialog and select the ECM option, then choose the type of character you want to create.



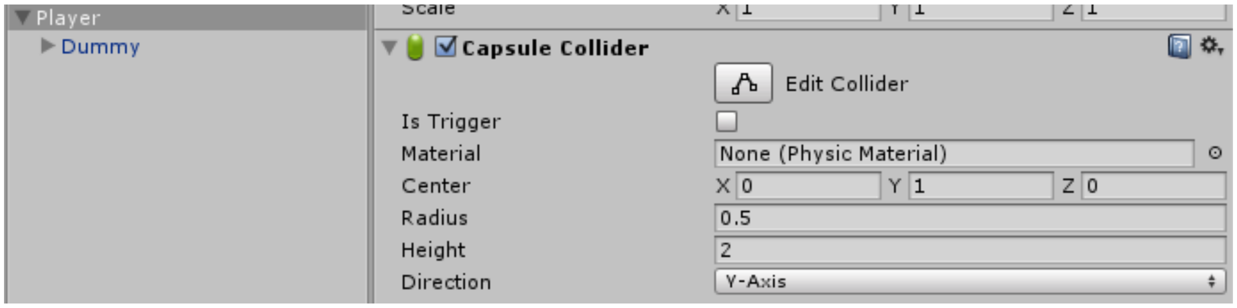
3. It will create an *empty* (no visual representation) character named **ECM_Character**, **ECM_Agent** or **ECM_FirstPerson** based on your selection. Please make sure its origin is at 0,0,0. This will save troubles when parenting.



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



5. Adjust the Capsule collider and SphereGroundDetection to match your model size.



Scale X 1 Y 1 Z 1
Is Trigger ☐
Material None (Physic Material)
Center X 0 Y 1 Z 0
Radius 0.5
Height 2
Direction Y-Axis

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.

Kind Regards,
Oscar

Overview

Easy Character Movement is a powerful yet incredible easy to use Rigidbody based character controller.

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

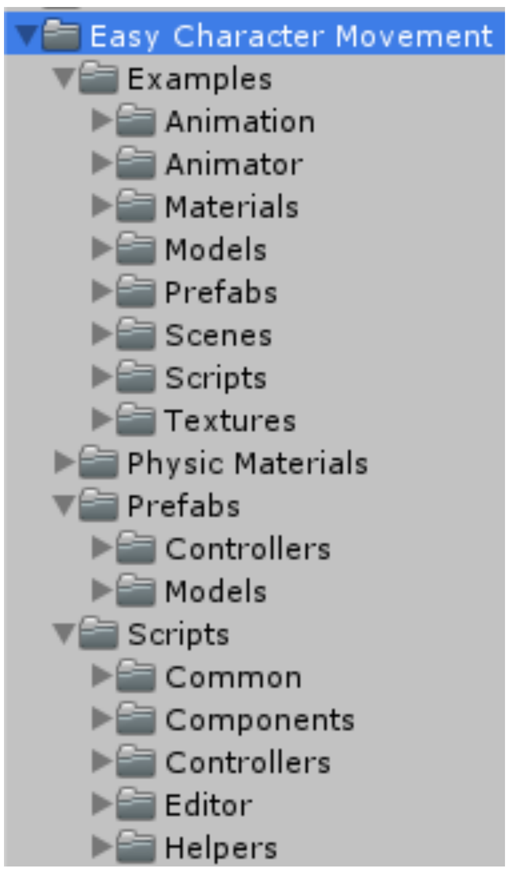
Easy Character Movement was designed from the ground up to be as efficient and flexible as possible with zero allocations after initialization. As a result, it runs great on all platforms including mobile.

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.
- Capsule and Box-based character colliders.
- Move and Rotate on dynamic platforms.
- Stay still on slopes.
- Keep same speed on straight segments and slopes.
- Slide on steep slopes (if desired).
- Base controller for Characters.
- Base controller for Agents (NavMeshAgent).
- Base controller for First-Person.
- Solid Root Motion support.
- Orient to ground slopes.
- 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 omitted when importing ECM into your projects.

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

Prefabs
This contains prefabs for the supplied base character controllers (**BaseCharacterController**, **BaseAgentController** and **BaseFirstPersonController**). This is optional and can safely be omitted when importing ECM into your projects.

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

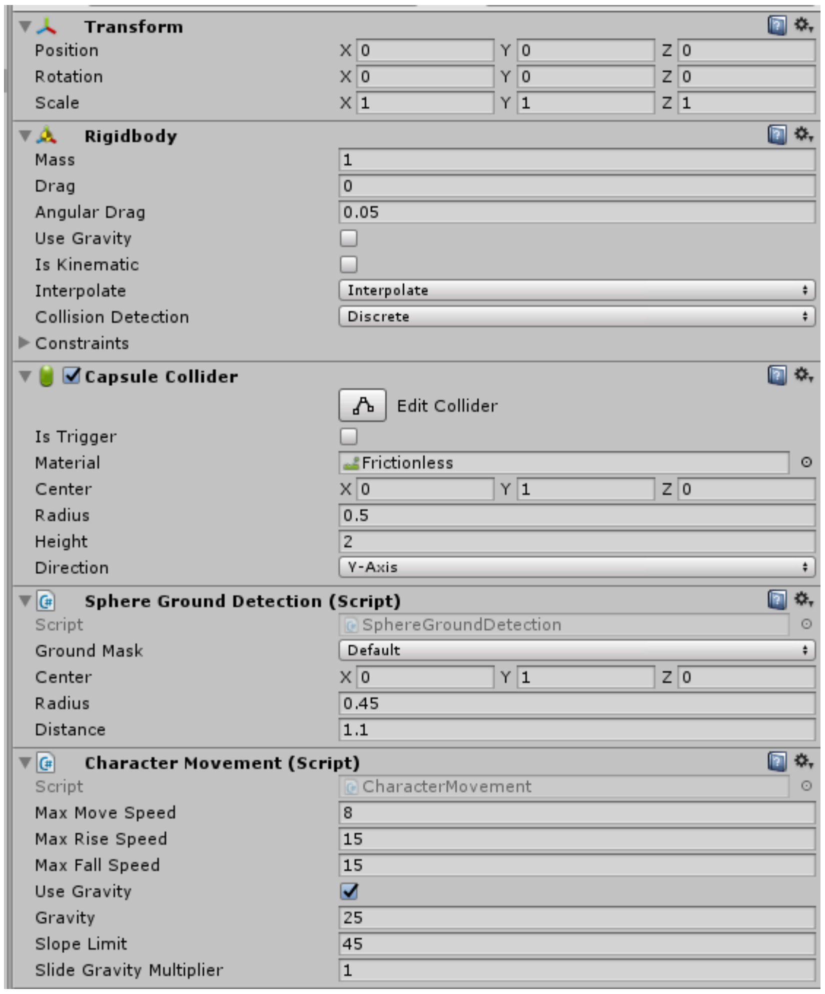
Description

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

The ***CharacterMovement*** component is the core of the ECM system and is responsible to perform all the heavy work to move a character (a.k.a. Character motor), such as apply forces, impulses, constraints, platforms interaction, etc. This is analogous to the Unity's character controller, but unlike the Unity's character controller, this make use of Rigidbody physics.

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

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



Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	☐
Is Kinematic	☐
Interpolate	Interpolate
Collision Detection	Discrete
Constraints	

Capsule Collider ✓

	Edit Collider
Is Trigger	☐
Material	Frictionless
Center	X 0 Y 1 Z 0
Radius	0.5
Height	2
Direction	Y-Axis

Sphere Ground Detection (Script)

Script	SphereGroundDetection
Ground Mask	Default
Center	X 0 Y 1 Z 0
Radius	0.45
Distance	1.1

Character Movement (Script)

Script	CharacterMovement
Max Move Speed	8
Max Rise Speed	15
Max Fall Speed	15
Use Gravity	✓
Gravity	25
Slope Limit	45
Slide Gravity Multiplier	1

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

Worth note that you can use any ***GroundDetection*** (Sphere, Box, Raycast 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 if required / desired.

I would like to point that ***SphereGroundDetection*** is the most robust of the included ***GroundDetection*** components and the only which offer **ledge detection**, and should be preferred in most of cases.

Above all, is the controller (eg: **BaseCharacterController**) which determines how the Character should be moved, such as in response from user input, AI, animation, etc., and feed this information to the ***CharacterMovement*** component, which perform the movement.

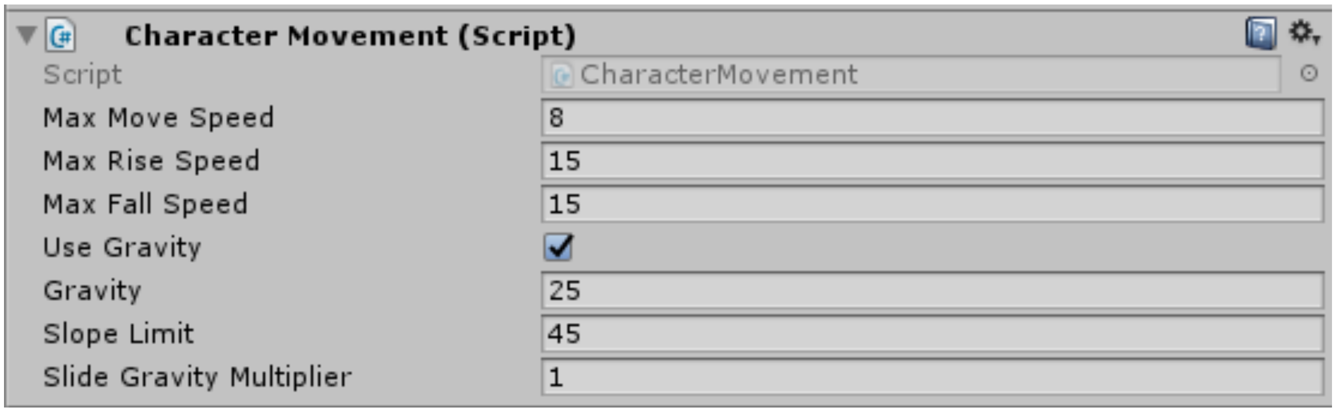
Worth note, that the ***CharacterMovement*** component by itself will do nothing, you must call its **Move** method in order to actually move the character.

Components

Character Movement

CharacterMovement is the core of the ECM system and is responsible to perform all the heavy work to move a character (a.k.a. Character motor), such as apply forces, impulses, constraints, platforms interaction, etc.

This is analogous to the Unity's character controller, but unlike the Unity's character controller, this make use of Rigidbody physics.



Script	CharacterMovement
Max Move Speed	8
Max Rise Speed	15
Max Fall Speed	15
Use Gravity	✓
Gravity	25
Slope Limit	45
Slide Gravity Multiplier	1

Max Move Speed
The maximum lateral (XZ plane) speed this character can move, including movement from external forces like sliding, collisions, etc.

Max Rise Speed
The maximum rising speed. Effective terminal velocity along Y+ axis.

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.

Ground Detection

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

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

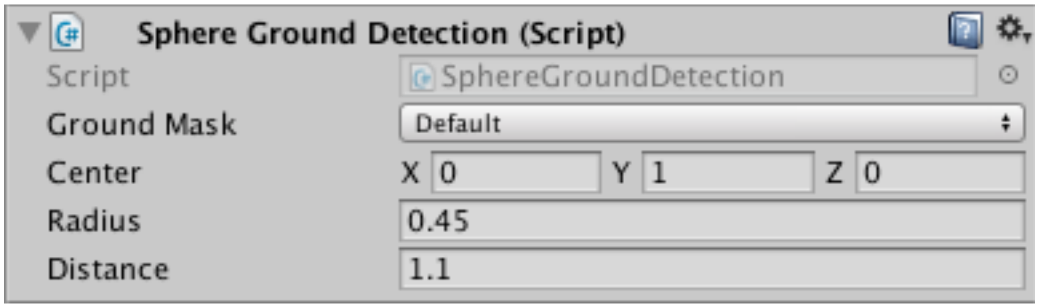
Included are three ground detection methods, one uses a Sphere to represent the character's feet, a second one which uses a Box to represent the character's feet, and a basic ray cast ground detection.

Worth note that ***SphereGroundDetection*** is the **most robust** of the included ***GroundDetection*** components and the only one which offer **ledge detection**, and should be preferred in most of cases.

Sphere Ground Detection

Extends the abstract ***GroundDetection*** class, and uses a Sphere to represent the character's "feet".

This is the more robust of all the included ***GroundDetection*** methods, and the **only** which offer **ledge detection**, and should be preferred in most of cases.


Sphere Ground Detection (Script)

Script	SphereGroundDetection
Ground Mask	Default
Center	X 0 Y 1 Z 0
Radius	0.45
Distance	1.1

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

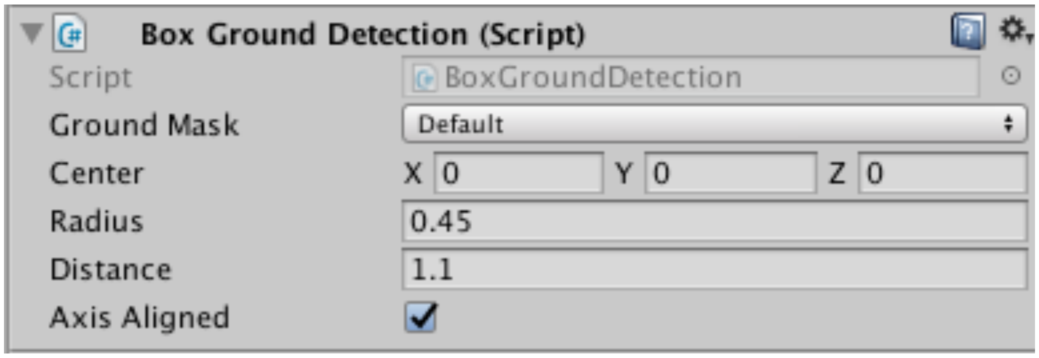
Center
The character's center position in object’s local space (relative to its pivot). This determines the cast starting point.

Radius
The bounding volume radius. This represent the radius of the 'feet' or bottom of the character. A typical value should be about 90-95% of your collider's radius.

Distance
Determines the cast distance. A typical value should be about ~10% more than your collider's half height.

Box Ground Detection

Extends the abstract ***GroundDetection*** class, and uses a Box to represent the character’s “feet”.


Box Ground Detection (Script)

Script	BoxGroundDetection
Ground Mask	Default
Center	X 0 Y 0 Z 0
Radius	0.45
Distance	1.1
Axis Aligned	✓

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

Center
The character's center position in object’s local space (relative to its pivot). This determines the cast starting point.

Radius
The bounding volume radius. This represent the radius of the 'feet' or bottom of the character. A typical value should be about 90-95% of your collider's radius.

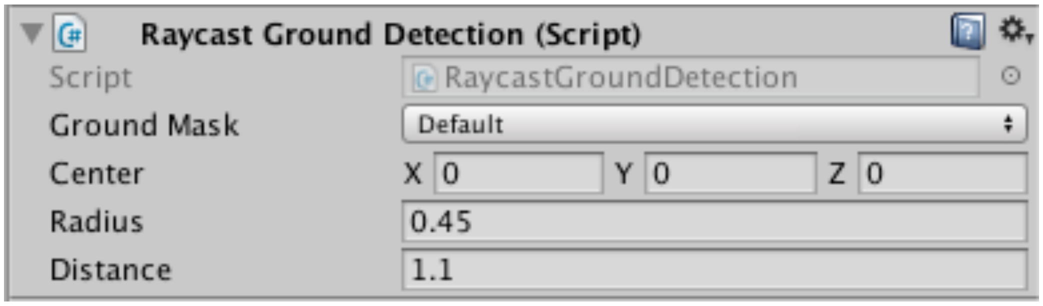
Distance
Determines the cast distance. A typical value should be about ~10% more than your collider's half height.

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

Raycast Ground Detection

Extends the abstract ***GroundDetection*** class, and uses a Sphere to represent the character's "feet".

This is a basic and lightweight ground detection method, and only should be used when the ground is mostly flat and no need the additional features that ***SphereGroundDetection*** offers.


Raycast Ground Detection (Script)

Script	RaycastGroundDetection
Ground Mask	Default
Center	X 0 Y 0 Z 0
Radius	0.45
Distance	1.1

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

Center
The character's center position in object's local space (relative to its pivot). This determines the cast starting point.

Radius
Not used.

Distance
Determines the cast distance. A typical value should be about ~10% more than your collider's half height.

Controllers

The responsibility of the controller, as stated above, is determine 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.

ECM offers 3 different Base (which can be extended) controllers, **BaseCharacterController,** a general purpose character controller, **BaseAgentController,** a base class for *NavMeshAgent* controlled characters, and **BaseFirstPersonController,** a base class for typical first person movement.

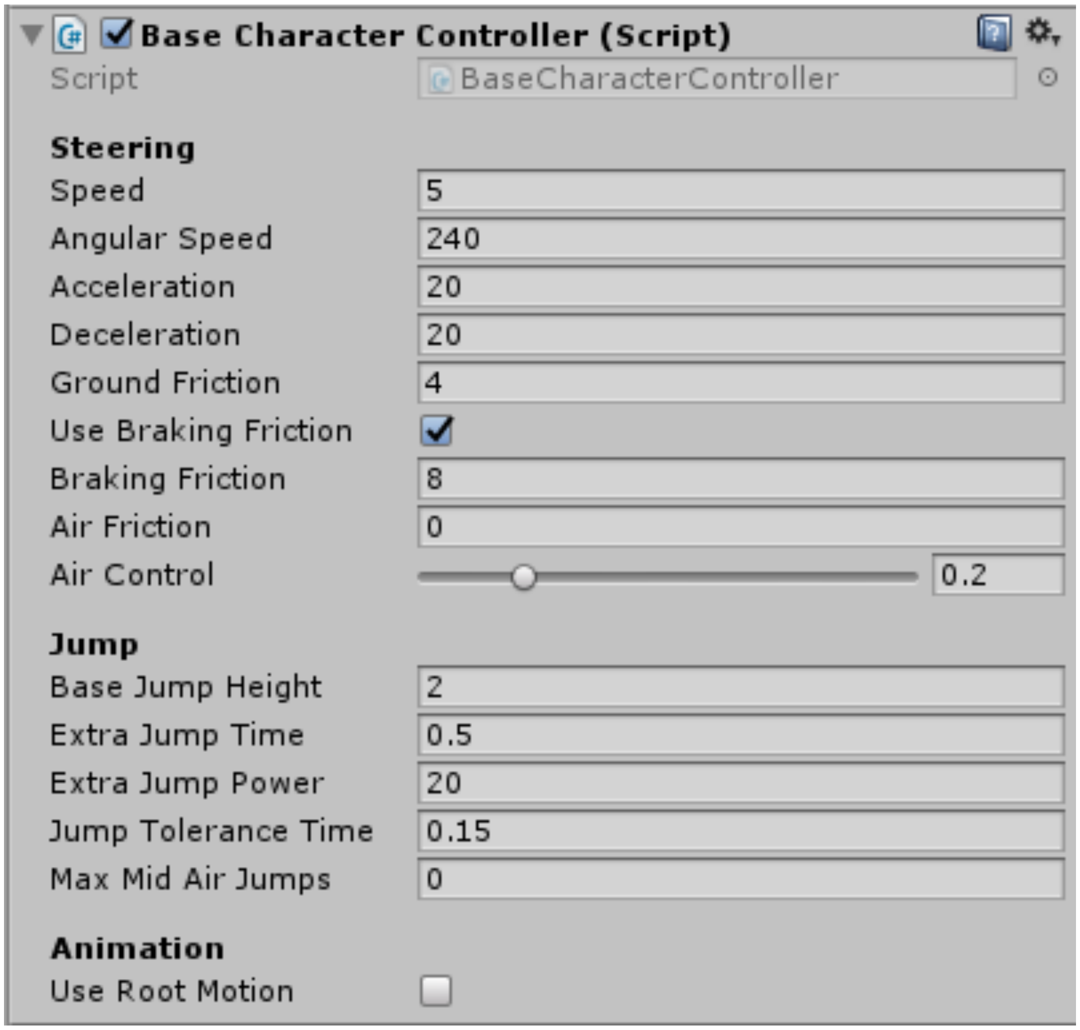
The suggested approach to work with ECM, is extend the supplied base controllers (please refer to the included examples) creating a custom character controller derived from base controllers (eg: **BaseCharacterController**), and add custom code to match your game needs, because ultimately no one knows your game better than you.

Worth note that the use of supplied base controllers is suggested but not mandatory, it is perfectly fine (if preferred) create your own character controller and relay on the ***GroundDetection*** and the ***CharacterMovement*** components to perform the character movement for you.

Base Character Controller

A general purpose character controller and base of other controllers. It handles keyboard input, and allows for a variable height jump, however this default behavior can easily be modified or completely replaced overriding this related methods in a derived class.

This is a robust class which among many features, it offers character *locomotion*, *variable height jump*, *unlimited mid-air jumps*, *root motion support*, etc., and should serve you as a strong foundation to develop your custom controllers on top.



Base Character Controller (Script)

Script	BaseCharacterController
Steering	
Speed	5
Angular Speed	240
Acceleration	20
Deceleration	20
Ground Friction	4
Use Braking Friction	✓
Braking Friction	8
Air Friction	0
Air Control	0.2
Jump	
Base Jump Height	2
Extra Jump Time	0.5
Extra Jump Power	20
Jump Tolerance Time	0.15
Max Mid Air Jumps	0
Animation	
Use Root Motion	☐

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.

Ground Friction
Setting that affects movement control. Higher values allow faster changes in direction. If *useBrakingFriction* is false, this also affects the ability to stop more quickly when braking.

Use Braking Friction
Should *brakingFriction* be used to slow the character? If false, *groundFriction* will be used.

Braking Friction
Friction coefficient applied when braking (when there is no input acceleration). Only used if *useBrakingFriction* is true, otherwise *groundFriction* is used.

Air Friction
Friction coefficient applied when 'not grounded'. This is analogous to *groundFriction*.

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.

Jump Tolerance Time
How early before hitting the ground you can press jump, and still perform the jump. Typical values goes from 0.05f to 0.5f.

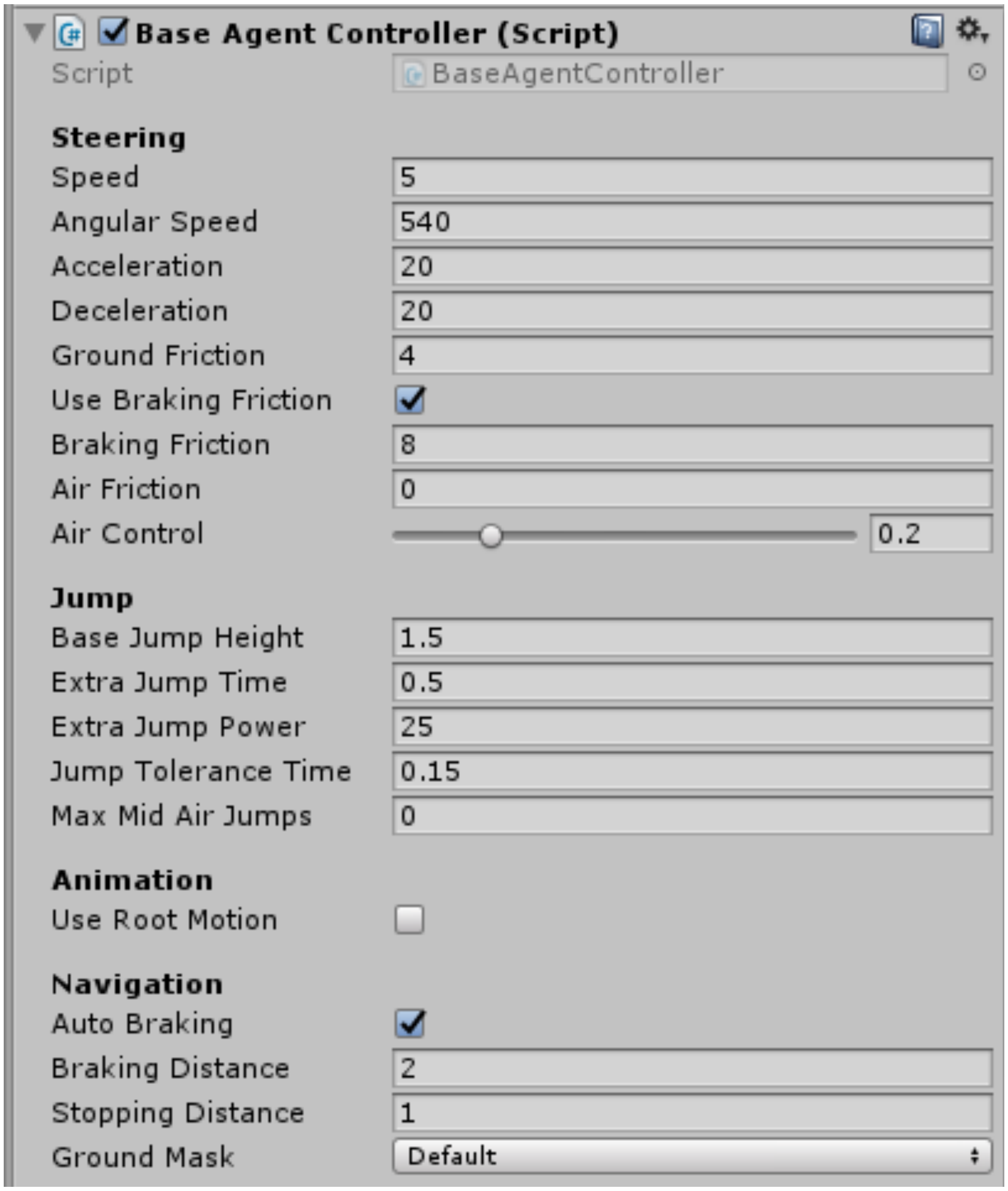
Max Mid-Air Jumps
Maximum mid-air jumps. 0 disables mid-air jump.

Use Root Motion
Should use root motion? If true, the animation velocity will overrides movement velocity. This requires a ***RootMotionController*** attached to the *Animator* game object.

Base Agent Controller

A base class for *NavMeshAgent* controlled characters. It inherits from **BaseCharacterController** and extends it to control a *NavMeshAgent* and intelligently move in response to mouse click (click to move).

Like the base character controller, this default behavior can easily be modified or completely replaced in a derived class.



Base Agent Controller (Script)

Script	BaseAgentController
Steering	
Speed	5
Angular Speed	540
Acceleration	20
Deceleration	20
Ground Friction	4
Use Braking Friction	✓
Braking Friction	8
Air Friction	0
Air Control	0.2
Jump	
Base Jump Height	1.5
Extra Jump Time	0.5
Extra Jump Power	25
Jump Tolerance Time	0.15
Max Mid Air Jumps	0
Animation	
Use Root Motion	☐
Navigation	
Auto Braking	✓
Braking Distance	2
Stopping Distance	1
Ground Mask	Default

Auto Braking
Should the agent brake automatically to avoid overshooting the destination point? If this property is set to true, the agent will brake automatically as it nears the destination.

Braking Distance
Distance from target position to start braking, eg: slowing area.

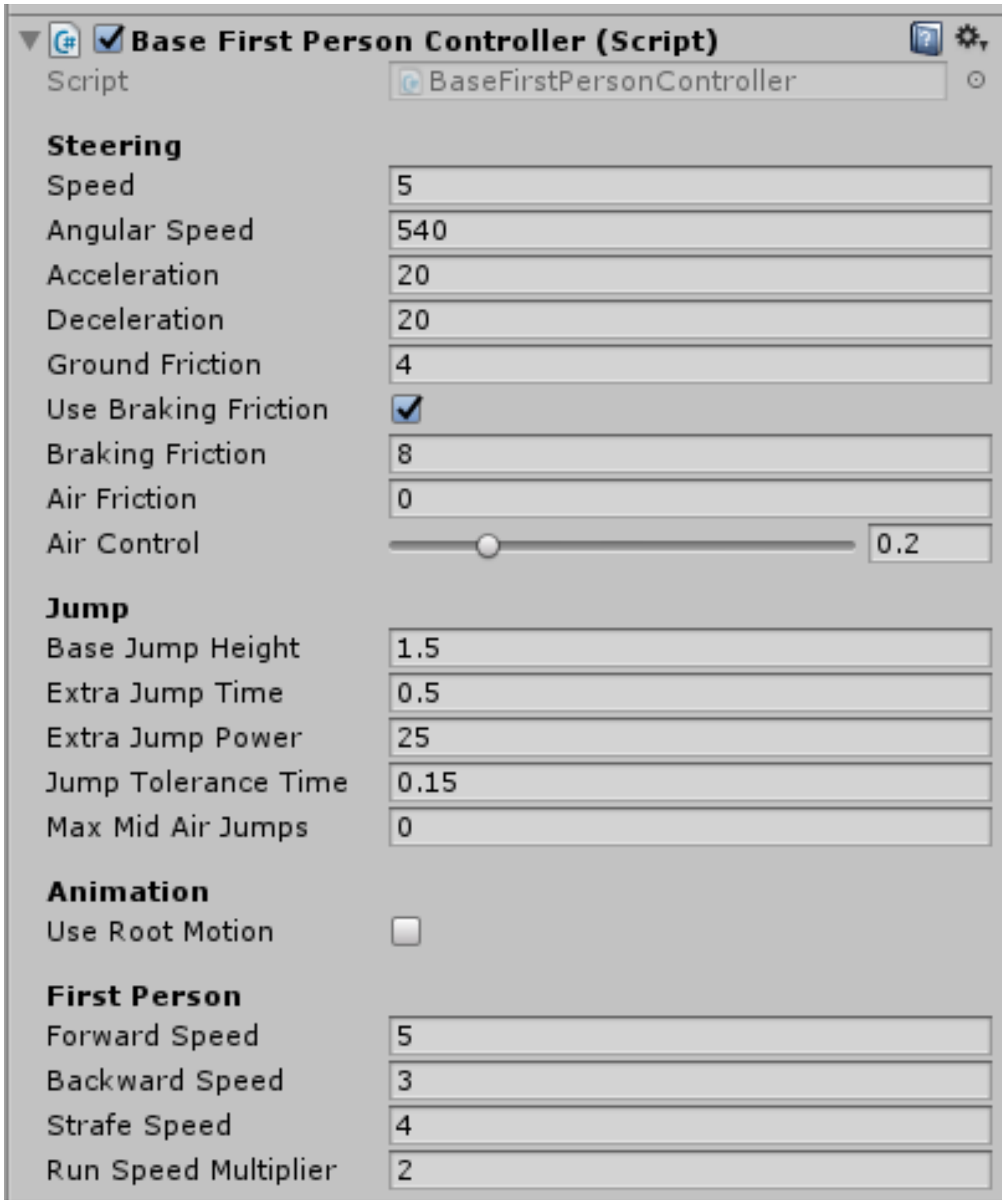
Stopping Distance
Stop within this distance from the target position.

Ground Mask
Layers to be considered as ground (walkable). Used by ground click detection.

Base First Person Controller

A base class for a first person controller. It inherits from **BaseCharacterController** and extends it to perform classic FPS movement.

Like the base character controller, this default behavior can easily be modified or completely replaced in a derived class.



Base First Person Controller (Script)

Script	BaseFirstPersonController
Steering	
Speed	5
Angular Speed	540
Acceleration	20
Deceleration	20
Ground Friction	4
Use Braking Friction	✓
Braking Friction	8
Air Friction	0
Air Control	0.2
Jump	
Base Jump Height	1.5
Extra Jump Time	0.5
Extra Jump Power	25
Jump Tolerance Time	0.15
Max Mid Air Jumps	0
Animation	
Use Root Motion	☐
First Person	
Forward Speed	5
Backward Speed	3
Strafe Speed	4
Run Speed Multiplier	2

Forward Speed
Speed when moving forward.

Backward Speed
Speed when moving backwards.

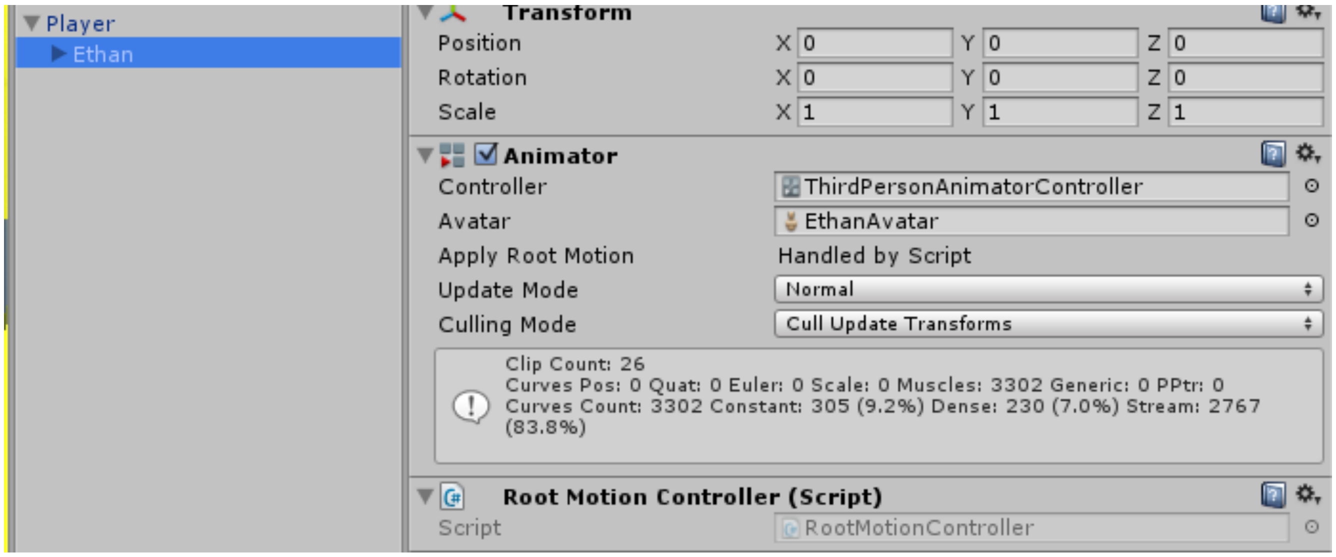
Strafe Speed
Speed when moving sideways.

Run Speed Multiplier
Speed multiplier while running.

Helpers

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.

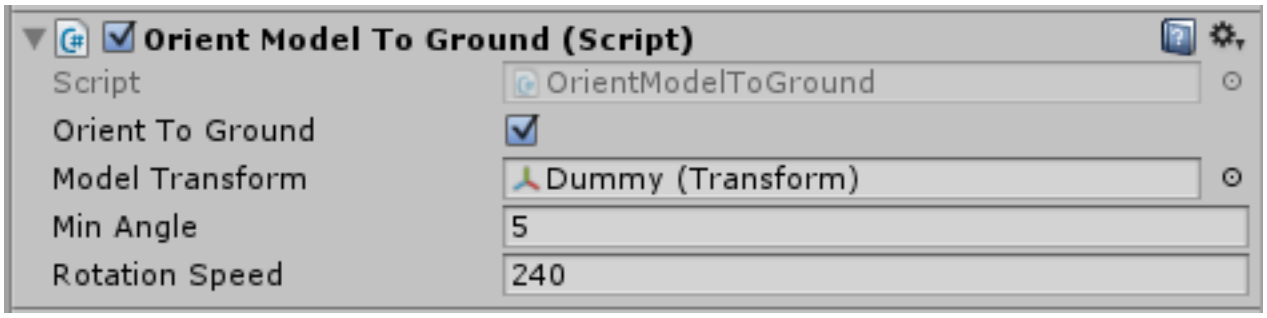


```
▼ Player
  ▶ Ethan

▼ Transform
Position        X 0    Y 0    Z 0
Rotation        X 0    Y 0    Z 0
Scale           X 1    Y 1    Z 1

▼ Animator
Controller          ThirdPersonAnimatorController
Avatar              EthanAvatar
Apply Root Motion   Handled by Script
Update Mode         Normal
Culling Mode        Cull Update Transforms
  Clip Count: 26
  Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 3302 Generic: 0 PPtr: 0
  Curves Count: 3302 Constant: 305 (9.2%) Dense: 230 (7.0%) Stream: 2767
  (83.8%)

▼ Root Motion Controller (Script)
Script              RootMotionController
```

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.

Please refer to prefabs (Examples/prefabs) if you are in doubt about the hierarchy your characters should follow.



Orient Model To Ground (Script)

Script	OrientModelToGround
Orient To Ground	✓
Model Transform	Dummy (Transform)
Min Angle	5
Rotation Speed	240

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).

Prefabs

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

The prefabs live in the Prefabs directory.

Code Reference

Please refer to the included source code to review the full code reference.