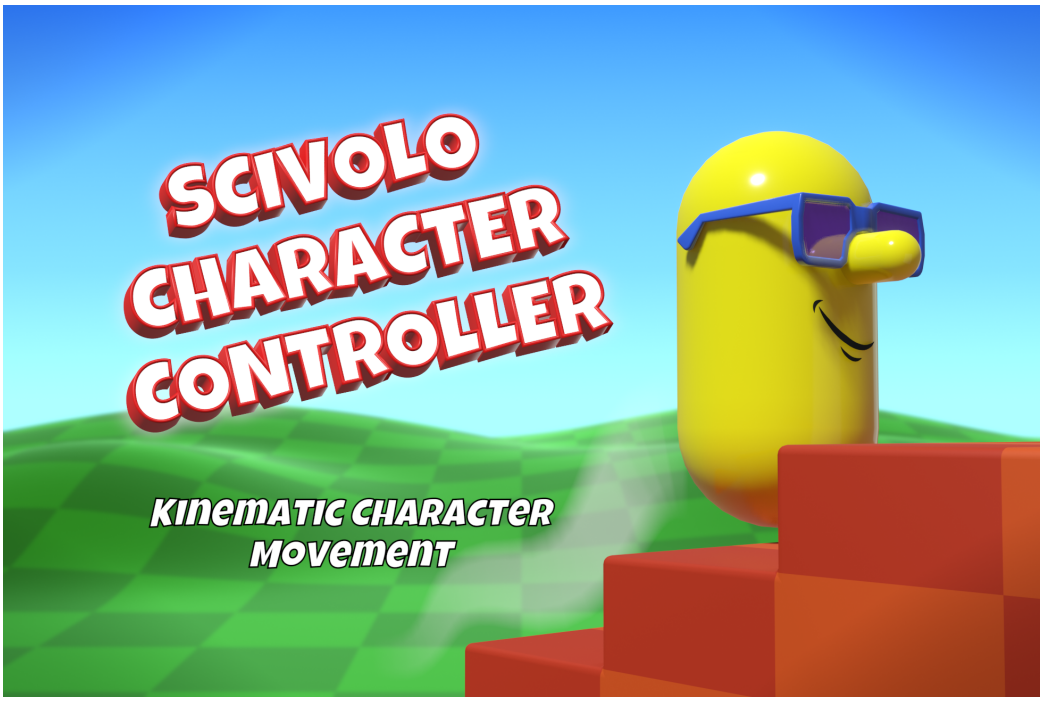
Scivolo Character Controller

Version 2.1

User Manual



Introduction

Scivolo Character Controller is a set of components that help in the creation of a custom character controller. It essentially provides the basic building blocks with which one can build the movement mechanics of his own character. Its purpose is to propose an alternative to the Unity's built in `CharacterController` component as well as expand its functionality for example allowing to freely rotate around any axis.

It has been made with kinematic movement in mind so for situation where the character is not meant to behave or move like an actual physical body.

Each component doesn't hold any movement logic to allow a great flexibility in many different scenarios, still trying to fulfill the common tasks that every character controller should accomplish.

Main features:

- Free capsule rotation
- Slope limit
- Step climbing
- Horizontal movement conservation
- Slide down on slopes
- Ground detection

Quick Setup

Create an empty game object which will represent your character and drag and drop your character model to it as a child. Make sure to remove any non-trigger collider or move them to a layer which doesn't interact with the layer of the character game object. Also make sure that the character game object is not scaled, that is its `Transform` component has a scale of `[1, 1, 1]`.

Select the character game object and attach to it the `GroundDetector` component which should also automatically attach the `CharacterCapsule` and `CharacterMover` components if not already attached.

The `GroundDetector` component is optional so a custom component can be used instead for detecting ground. In that case only the `CharacterMover`

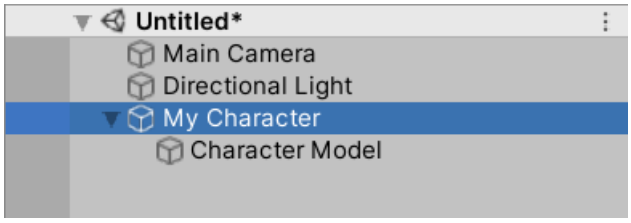


Figure 1: Character hierarchy

component need to be attached so that also the CharacterCapsule component is attached.

Adjust through the inspector the CharacterCapsule settings so that the capsule fits the shape of your character model.

Finally create your custom character controller script, attach it to the game object and start writing your code.

In order to access the components from the script, it should be used the MenteBacata.ScivoloCharacterController namespace.

The script should start like this:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using MenteBacata.ScivoloCharacterController;

public class MyCharacterController : MonoBehaviour
{
    // Your character code...
}
```

Furthermore, if the script is not inside the default assembly definition, a reference to the ScivoloCharacterController assembly definition must be manually added to the assembly definition file of the script in order to access the controller scripts.

Package Overview

The controller is made up of three separate MonoBehaviour components:

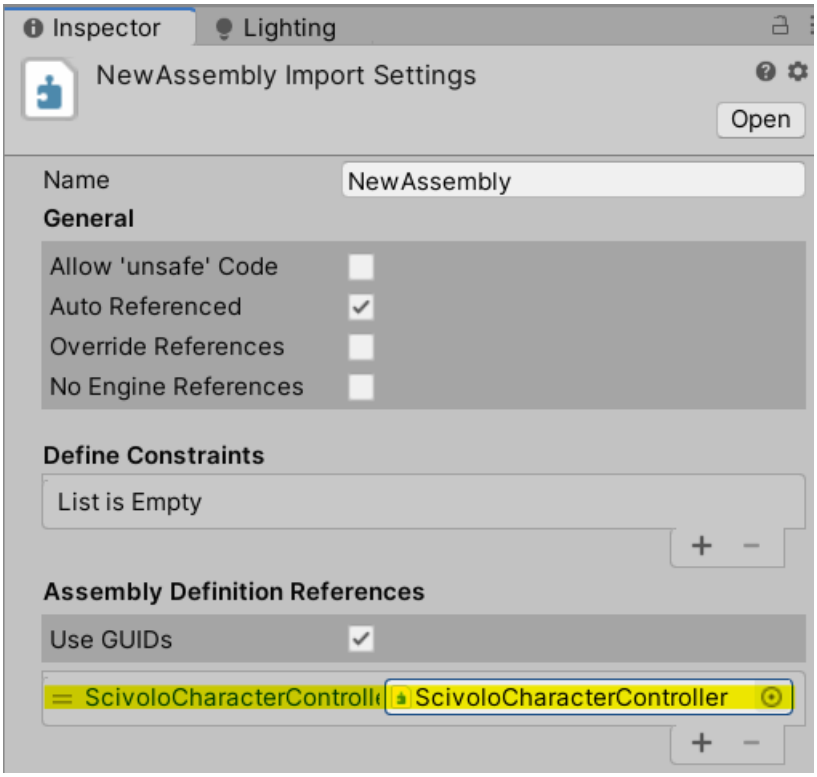


Figure 2: Reference to assembly definiton.

- `CharacterCapsule`
- `CharacterMover`
- `GroundDetector`

Each component has its own specific purpose. Details about these components will be provided in the following sections.

There are also two simple structs that serves as containers for data that can be retrieved:

- `GroundInfo`
- `MoveContact`

The package also contains a demo project that provides a demonstrative playable scene and can also serves as an usage example for the controller. It is not necessary to import the Demo folder to use the controller.

CharacterCapsule Component

The `CharacterCapsule` defines the shape of the capsule collider which represents how the character interacts with the other colliders in the game world. It also provides for the resolution of overlaps with the other colliders.

If the component is enabled, the overlap resolution occurs automatically at each update, but it can also be done manually by calling the appropriate method. Automatic overlap resolution should happen before any other update, so it has been assigned a low execution order. It can be changed in Project Settings.

It resolves overlaps only with colliders in the layers it is supposed to collide with as defined in the *Layer Collision Matrix* in the *Physics* section of the *Project Settings*.

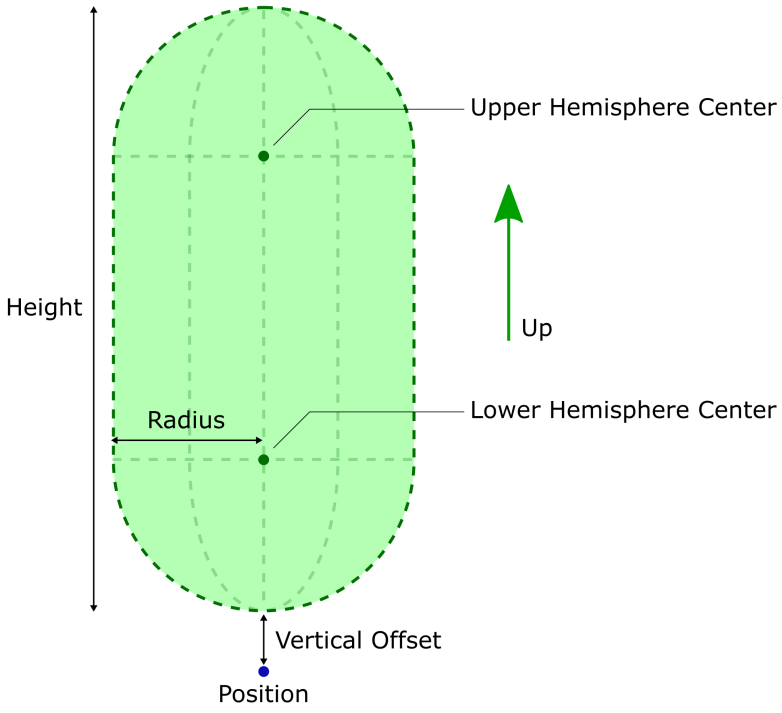


Figure 3: Character capsule parameters.

Unlike Unity's `CapsuleCollider` component, the `CharacterCapsule` has its main axis direction restricted to its local Y axis which also defines the char-

acter's up direction, of course the character itself can be freely rotated in any direction in world space.

When the game starts it instantiates a `CapsuleCollider` component with the corresponding shape and a `Rigidbody` with the *Is Kinematic* field set to true.

Serialized Fields

Field	Description
Vertical Offset	Vertical offset from the game object position to the bottom of the capsule. When it's zero the capsule bottom is at the game object position.
Height	Height of the capsule.
Radius	Radius of the capsule.
Contact Offset	Small distance from the surface of the capsule used as a safety margin to avoid that the capsule is directly in contact with other colliders.
Prioritized Overlap	Overlaps with colliders in layers excluded from this mask will be ignored if the attempt to resolve all overlaps fails. Colliders already ignored at the first attempt will still be ignored even if included in this mask. In practice this mask is an additional filter and can only exclude layers. It can be used for example to give higher priority to the geometry of the level than other characters, small props or moving objects.

See figure 3 for a visual reference.

Public Methods

TryResolveOverlap

```
bool TryResolveOverlap()
```

Tries to resolve overlaps with every colliders it is supposed to collide with. If the first attempt fails, it tries again considering only the high priority

colliders. Returns true if it managed to resolve all overlaps, including those with non-prioritized colliders, false otherwise.

CharacterMover Component

The `CharacterMover` component is responsible for moving the character around the game world. It deals with collision detection and handle sliding on surfaces and climbing on steps.

When it moves, it takes into account the contact offset of the capsule by trying to maintain a small distance between the surface of the capsule and the other colliders.

Collisions with other colliders are defined by the *Layer Collision Matrix* in the *Physics* section of the *Project Settings*.

It requires the `CharacterCapsule` component attached to the same game object in order to work.

Ground Types

Sliding on ground is handled according to its type, there are two kind of grounds:

- Floor
- Steep Slope

Ground types are defined by their slope angle, which is the angle that the ground surface forms with the horizontal plane, where the horizontal plane is the plane perpendicular to the character up direction. This means that a ground could change its type if the character up direction changes.

Ground is considered a floor when its slope angle is smaller than the value set in the *Max Floor Angle* field otherwise it is considered a steep slope.

Ground Movement

When the character moves on a floor it always tries to maintain the horizontal component of the movement. When moving on a steep slope the character is always allowed to slide down but is only allowed to move up on it if the *Can Climb Steep Slope* field is set to true and it is not in walk mode. However, even if it is not allowed to move up on a steep slope, it

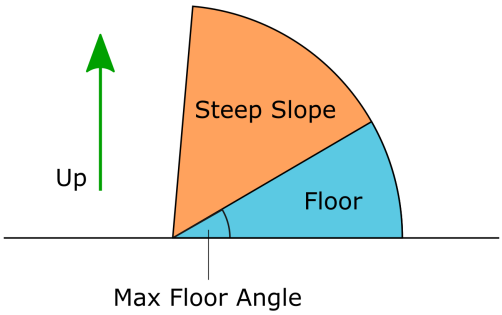


Figure 4: Ground types.

can still slide along its side. This holds whether the mover is in walk mode or not.

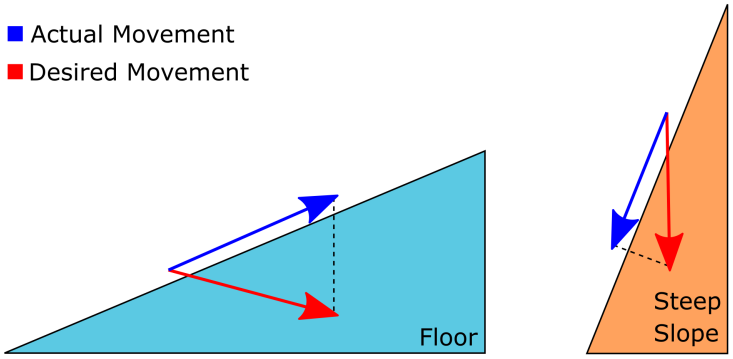


Figure 5: Movement on floor (left) and on a steep slope (right).

Walk Mode

When the *Is In Walk Mode* field is set to true, the `CharacterMover` switch to walk mode. This mode is better suited for when the character is moving by foot, so not only for walking but also jogging, running and also for standing up idle.

The walk mode allows the character to climb on steps and it tries to clamp the character on the floor, so there is no need to apply a downward movement to prevent the character to leave the floor, for example when moving down on a ramp.

Walk mode works whether the character is grounded on floor or not, so if the character starts the movement while already on floor it will try to keep him on the floor, otherwise it will try to snap him on the nearest floor below if found within a certain distance. However, if the floor has not been found, the character will not be abruptly pushed down more than necessary.

To allow the character to easily leave the floor, for example when jumping, floor clamping is only enabled when the desired movement is not directed upward, that is when the movement component along the up direction is not positive.

When in walk mode, climbing steep slope is always prevented, even if the *Can Climb On Steep Slope* field is set to true. This is mostly due to the fact that floor clamping interferes with the ability to climb on steep slope.

Step Climbing

In walk mode, if when moving it touches an obstacle that it recognizes as a step it can climb, it will climb and move past it. Steps are considered floor so it will try to maintain the horizontal component of the movement no matter if climbing up or down a step.

For a step to be considered as such, it needs to be composed of a nearly vertical surface which form an angle with a not very sloped surface (there isn't a hard limit actually, but it is recommended to not go past 30 degrees to stay safe). It accounts to some beveling so the edge doesn't need to be perfectly sharp. See figure 6 for a visual reference. Also there has to be a distance of at least a half of the capsule radius between two consecutive steps.

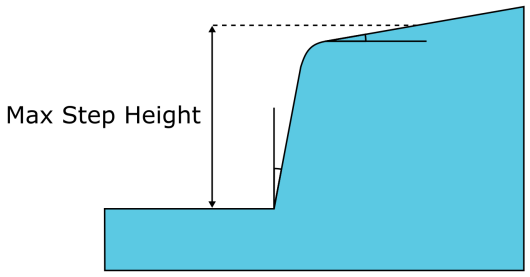


Figure 6: A valid step.

Although there is no limit to the height of a climbable step, values greater

than the capsule radius could cause inconsistent behaviour. This is because the CharacterMover relies on the contact position to determine the height of a step, so if its side is not a perfectly vertical surface it will always touch the capsule at the top of its bottom hemisphere. Therefore, for steps higher than the capsule radius, it would be safer if they had a more rectangular profile.

For performance reasons, the height of a step is measured starting from the capsule bottom and not the ground below. It works fine for climbing steps up as it assumes that the capsule is touching the ground below before starting climbing. The downside is that if it is in walk mode then it will hang on ledges, instead of sliding down, even if they are higher than a climbable step. In that case however, the GroundDetector component should no more detect the ground as floor and that information could be used to turn to false the *Is In Walk Mode* field, for example to put the character in an airborne state.

Serialized Fields

Field	**Description**
Is In Walk Mode	If true, sets the mover in walk mode.
Max Floor Angle	Maximum angle a slope can have in order to be considered floor.
Max Step Height	Maximum height of climbable step.
Can Climb Steep Slope	Allow the character to climb slopes which exceed the maximum floor angle. Only if not in walk mode.

Public Properties

Property	**Description**
static MaxContactsCount	Maximum possible number of contacts that can be detected, useful for instantiating the contact list without the need for further resizing.

Public Methods

Move

```
void Move(Vector3 desiredMovement, List<MoveContact> moveContacts)
```

```
void Move(Vector3 desiredMovement)
```

Moves the character according to the movement settings trying to fulfill the desired movement as close as it can. If a list of MoveContact is provided it populates the list with all the relevant contact information it has found during the movement. The MaxContactsCount property can be used to instantiate a list of the optimal size.

Parameter	Description
desiredMovement	The desired movement.
moveContacts	Reference to a list of MoveContact that will be filled in.

GroundDetector Component

The GroundDetector component is responsible for scanning the ground below the character capsule and retrieving useful information about the ground if found.

It only detects colliders that CharacterMover can collide with.

In order to work it needs both the CharacterCapsule and CharacterMover components.

Serialized Fields

Field	Description
Tolerance	Small tolerance distance so that ground is detected even if the capsule is not directly touching it but just close enough.

Public Methods

DetectGround

```
bool DetectGround(out GroundInfo groundInfo)
```

It detects ground below the capsule bottom and retrieves useful information on the ground. Returns true if it detects ground false otherwise. If ground is detected it fills in the passed groundInfo variable with the retrieved informations.

Parameter	Description
out groundInfo	Info about the detected ground.

GroundInfo Struct

The GroundInfo struct is a container for the ground data retrieved by the DetectGround method when it has detected ground.

Public Fields

Field	Description
point	Ground point considered in contact with the capsule bottom hemisphere.
normal	Ground normal, it accounts for steps by returning the normal of the upper floor.
tangentNormal	Normal of the plane tangent to the ground and to the capsule bottom hemisphere at the contact point.
collider	Ground collider.
isOnFloor	Should the character be considered on floor on this ground?

As shown in figure 7, when the capsule is on a flat surface the normal and the tangent normal are the same (upper image), instead, if the capsule is on the edge of a step, the normal is the direction of the normal of the floor above the step while tangent normal always points toward the center of the capsule bottom hemisphere (lower image).

MoveContact Struct

The MoveContact struct is a container for the contact data retrieved by the Move method.

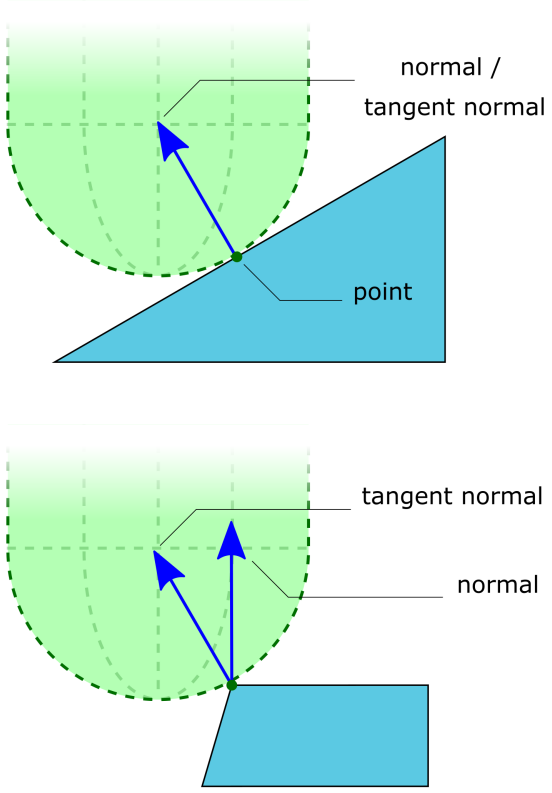


Figure 7: Normal and tangent normal for different ground.

Public Fields

Field	Description
position	World space position of the contact point.
normal	Normal of the plane tangent to the capsule at the contact point.
collider	Collider of the object on which the contact happened.