Scivolo Character Controller

User Manual

Introduction

Scivolo Character Controller is a set of components for helping to build a custom character controller. It essentially provides the basic building blocks with which one can build the movement mechanics of his own character. Its aim is to propose an alternative to the Unity's built in `CharacterController` component besides expanding its functionalities by allowing for free rotation 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.

Every component doesn't hold any movement logic and that is to allow for great flexibility for 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]`.

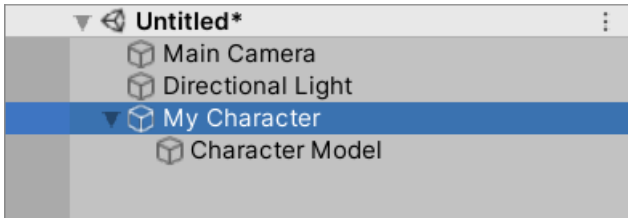

Figure 1: Character hierarchy

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 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.

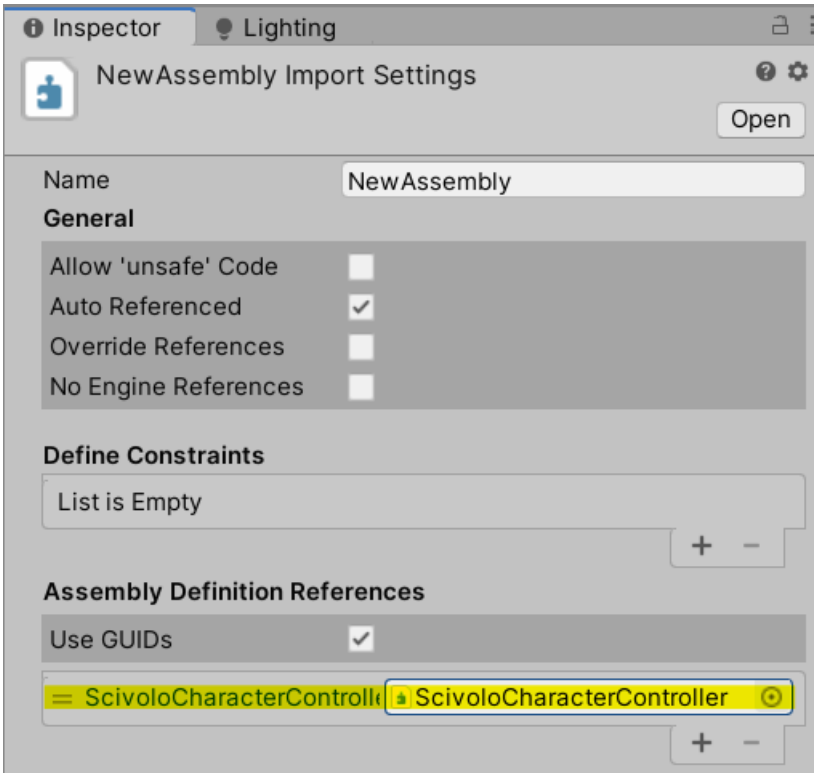

Figure 2: Reference to assembly definiton.

Package Overview

The controller is made up of three separate `MonoBehaviour` components:

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

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

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

Overview

The `CharacterCapsule` is responsible for defining the shape of the capsule collider which represents how the character interacts with the other colliders in the game world.

Unlike Unity's `CapsuleCollider` component, the `CharacterCapsule` has its main axis direction restricted to its local Y axis which also defines the character's up direction, of course the character itself can be freely rotated in any direction in world space.

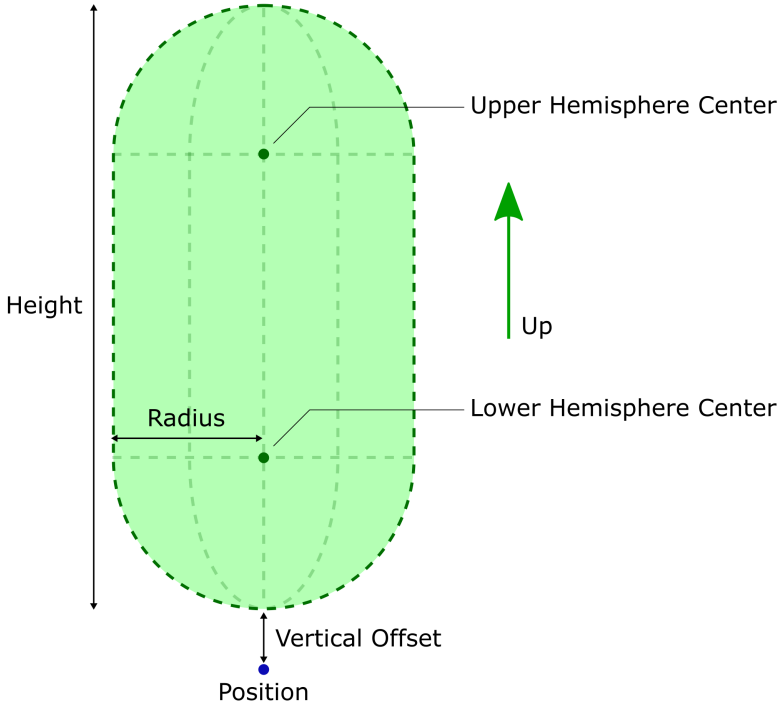

Figure 3: Character capsule parameters.

When then 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.

See figure 3 for a visual reference.

CharacterMover Component

Overview

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. Also, if the movement starts with the `CharacterCapsule` overlapping other colliders, it will try to resolve the overlap .

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

Interaction with other colliders is defined by the *Layer Collision Matrix*. So the `CharacterMover` will ignore all the colliders on layers that are not set to collide with the character game object layer. Also it will always ignore trigger colliders.

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.

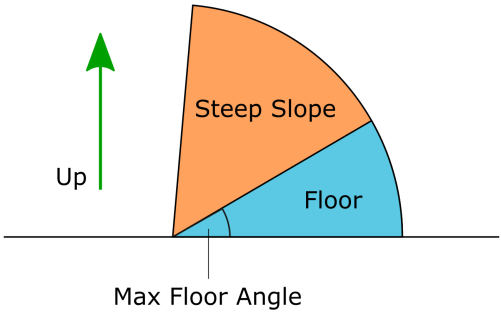

Figure 4: Ground types.

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 *Prevent Climbing Steep Slope* field is set to false. However, even if it is not allowed to move up on a steep slope, it can still slide along its side.

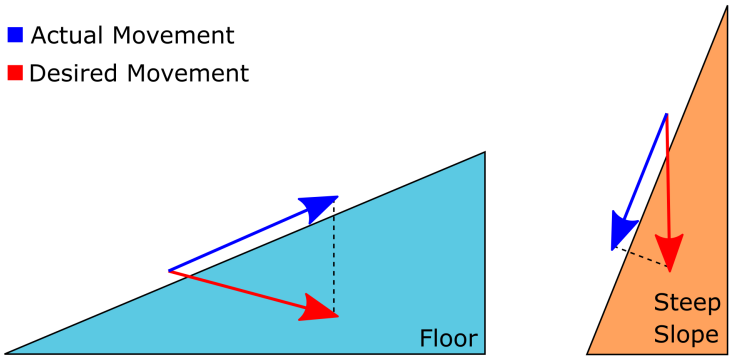

Figure 5: Movement on floor (left) and on a steep slope (right).

It is possible to clamp the character on ground when moving on a floor just by adding a downward movement, when on floor it won't slide down because horizontal movement is conserved so it will stay in place if there is no horizontal movement.

Step Climbing

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.

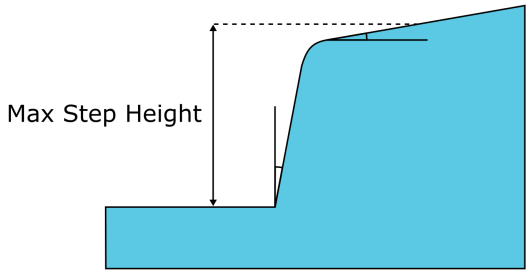

Figure 6: A valid step.

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 if a step 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 is safer that they have 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 the *Can Climb Steps* field is true

it will hang on ledges, instead of sliding down, even if they are higher than a 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 *Can Climb Steps* field, for example to put your character in an airborne state.

Contact Offset

For the `CharacterMover` to work properly, a small contact offset is needed so that contacts are detected slightly before the character capsule actually touches the collider it is moving to. Therefore this will result in a small distance between the capsule and the contact surface on the other collider.

In particular there will always be some space between the capsule bottom and the ground on which it stands. For this reason the contact offset value should be kept very small so hopefully this won't be very noticeable. Its default value should be fine for most cases, anyway it can still be adjusted by setting the *Contact Offset* field.

Serialized Fields

Field	**Description**
Max Floor Angle	Maximum angle a slope can have in order to be considered floor.
Prevent Moving Up Steep Slope	Prevents the character from moving up slopes which exceed the maximum floor angle.
Can Climb Steps	Allows the character to climb steps.
Max Step Height	Maximum height of climbable step.
Contact Offset	Small distance padding from the collider surface within which contacts are detected.

Public Methods

Move

```
void Move(Vector3 desiredMovement, List<MoveContacts> moveContacts = null)
```

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 contact information it has found during the movement. List size can grow depending on how many contacts are detected, however there is an upper bound for the maximum number of contacts it can retrieve and it will stop allocating memory once it reaches that limit.

GroundDetector Component

Overview

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

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

The same way as the CharacterMover, only colliders on layers with which the layer of the character game object is set to collide in the *Layer Collision Matrix* are detected.

Ground Detection Functioning

It is worth giving a brief description of the ground detection functioning to better explain the meaning of the component parameters.

Its principle of operation is quite simple: at first it casts down from the capsule bottom a sphere with the same radius as the capsule for a small tolerance distance (given by the value of the *Tolerance* field). If it hits something it returns false, which means that the character is not standing on any ground, otherwise it does further operations to evaluate if the ground is also a floor.

It uses the same definition of floor as for the CharacterMover but it makes a more robust check by probing the ground at different positions. In particular, it casts down some rays from the capsule bottom to check for other ground points. These rays should reach for a greater distance down below the capsule bottom to make sure they detect ground points even if they are not close to the bottom of the capsule.

The reason why these more far points are needed is that, at some point during the floor evaluation, it considers groups of three points for approximating the surface of the ground with a triangle, therefore it only needs that at least one of its vertex is close.

Points are considered close if their distance from the capsule bottom is less than the maximum floor distance, which represents how far the floor can be from the capsule bottom for the character to be considered standing on floor. It is usually the same as the tolerance distance used in the initial phase.

However, when it detects that the character is standing on a ledge, the maximum floor distance is adjusted up to the value of the *Max Step Height* field in `CharacterMover`, so that steps can also be detected as floor.

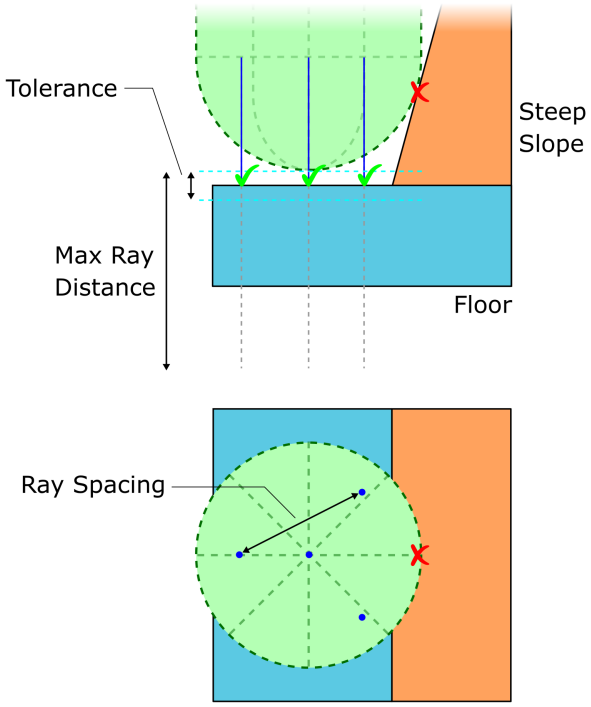

Figure 7: Side and top views of the ground detection when the bottom of the capsule touches a steep floor and rays are used to check for the floor below.

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.
Ray Spacing	Spacing between probe rays in relation to the capsule radius.
Max Ray Distance	Max distance for the rays used for probing ground.

See figure 7 for a visual reference.

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.