2D Platform Controller

JNA Mobile

Quick Start

To familiarize yourself with the controller use the Sample Scene.

For your own scenes it is recommended that you start with one of the prefabs, such as **Character** or **CharacterForSlopes**. Drag this on to your scene. This square character is controlled with arrow keys and space bar and will interact with unity colliders.

If you are having issues ensure the z-axis is aligned for all the objects (for example all of them have z = 0).

The controller ONLY works in an X-Y plane.

Changing the model

1) Remove the mesh renderer from the square controller.
2) Add your model or 2D sprite as a child of the controller.
3) Do not rotate the controller object. If you need to adjust rotation rotate the child object.

Updating the Raycasts

Use the editor gizmos to change offset and distance parameters for your raycasts. Use the **Collider Editor Options** to show only the handles that you need to use. Alternatively enter values in the Inspector.

The green raycasts are downwards and should be used as feet colliders, they push your character up from the ground. They are also used to detect slopes and ladders. Feet colliders should be the same length and at the same *y* offset as each other, use the **Align Feet** button to ensure this is the case.

The purple raycasts are upwards and should be used as head colliders. They push your character downwards from obstacles above them.

The red and yellow colliders are left and right colliders. They push your character away from obstacles to the left or right.

Number of Raycast Colliders

For a simple controller you can use 2 feet, 2 head, and 4 side raycasts (2 left, 2 right).

A **climbing** controller will need 3 feet.

A **sloped** controller will need 3 feet colliders, for better results particularly with steep platforms use 5 feet colliders.

The gap between your **side** colliders, should be smaller than the thinnest platform colliders. If you want complex shapes or thin platforms you will likely need to add additional **side** colliders.

For mobiles you may need to restrict the total number of colliders. Note that this is usually only a problem if you have multiple characters, even low-end mobiles can support 12 or more raycasts without issue.

Layers

There are three layers used by the controller:

> The **background** layer is for normal platforms.
>
> The **pass through** layer lets characters walk and jump through the platform, but they can still stand on it.
>
> The **climbable** layer is used for ladders, vines, and other climbable objects. See the notes on ladders for setting up climbable objects.

Other layers are ignored.

Platform Behaviour

To add special behaviour to your platforms extend the platform class. The DoAction method is triggered by each raycast that collides with the platform. Check the direction of the raycast to ensure it's the right type (for example if something has an action when stood on ensure its being hit by a DOWN raycast).

Check out the samples for more ideas.

You can also add a trigger or (kinematic) collider to your character. In this case change the physics settings to ensure the trigger or collider doesn't interact with the platform controller layers (background, passthrough, and climbable).

Ladders

Ladders are constructed of multiple small colliders that belong to the climbable layer. The distance between these colliders in the y direction should be smaller than the distance of the feet colliders. The ladder prefabs are a good starting point for any ladder system.

If you are building your own ladders also note that:

If a ladder is to touch the ground then the ladder colliders must be smaller in the x direction than the width of the character (i.e. the distance between the characters outer feet colliders).

If you want the top of the ladders to act like a platform add the TopStepPlatform script to the top step (or set **AutoStick = true**).

Input

Your input class should extend the provided abstract class (**RaycastCharacterInput**).

X values control direction. A magnitude of one or greater represents a run value, a magnitude between 0 and 1 (exclusive) represents a walk value.

Y values are used for climbing.

Jump and JumpHeld control the start and duration of a jump.

Animation

Animations are sent as events. By default only changes of state are sent. An animation event can also be sent every frame by selecting the appropriate checkbox in the editor settings.

See the example animation classes for details.

You can also use the **State** property of the controller to inspect animation state.

Other properties may also be useful when animating for example the Velocity property of the controller, and the input properties available from the RaycastCharacterInput.

Working in UnityScript (Unity JavaScript)

Move the scripts to a directory called **Plugins** to ensure they are available to UnityScript.

Because Unity Script does not support c# events you have two options for working with Animations:

1) Write a c# class which sends the events to your Javascript code (for example using SendMessage).
2) Inspect the State and Velocity property of the controller.

Description of All Settings

public class RaycastCharacterController

```
     public RaycastCollider[] feetColliders
The feet colliders.

     public RaycastCollider[] headColliders
The head colliders.

     public RaycastCollider[] sides
The side colliders.

     public MovementDetails movement
Movement details.

     public JumpDetails jump
Jump details.

     public SlopeDetails slopes
Slope handling details.

     public ClimbDetails climbing
Climbing details.

     public int backgroundLayer
The layer used for normal collisions.

     public int passThroughLayer
The layer used for pass through collisions.

     public int climableLayer
The layer used for climbable colliders.

     public bool controllerActive
Set to false to turn the controller off.

     public RaycastCharacterInput characterInput
The class providing input. You could use an AI here too.

     public bool sendAnimationEventsEachFrame
If true send events each frame. If false only send when
```

state changes.

public class MovementDetails

public float walkSpeed
The walk speed, any faster than this and the run animation will be played.

public float runSpeed
Maximum X speed that can be obtained.

public float acceleration
How quickly the character gets to speed.

public float drag
How much the character slows down. 1 means no drag.

public float terminalVelocity
The maximum speed in the y direction.

public float skinSize
The minimum amount of overlap before the character is moved by a collision. High values will make the character shake.

public class SlopeDetails

public bool allowSlopes
Do slope calculations?

public float slopeLookAhead
How much extra we raycast from each foot in order to determine if we are on a slope.

public float groundError
How much leeway we allow when detecting the grounded state.

public float rotationSpeed
How quickly we rotate to a new slope.

public class JumpDetails

```
    public bool canDoubleJump
```
Can we double jump?

```
    public bool canWallJump
```
Can we wall jump?

```
    public float jumpVelocity
```
How fast do we jump?

```
    public float doubleJumpVelocity
```
How fast do we double jump?

```
    public float jumpTimer
```
How much time do we allow to get off the ground before we are considered to be jumping?

```
    public float jumpHeldTime
```
How long we allow the jump button to be held to increae our jump height.

```
    public float jumpFrameVelocity
```
How much extra velocity we get each frame if we are holding the jump button down.

```
    public float wallJumpTime
```
How much leeway we have between pushing against a wall, and pressing jump and the opposite direction in order to wall jump.

public class ClimbDetails

```
    public bool autoStick
```
If we hit a climbable do we automatically stick to it or do we need to press up or down.

```
    public bool allowClimbing
```
Do we allow climbing?

```
    public float speed
```
How fast we climb.

```
    public int collidersRequired
```
The number of feet colliders required to be hitting a

limable before we allow climbing. More colliders means ladders feel more narrow. Smaller amount of colliders means ladders can be moved about on.

public class RaycastCollider

```
public Transform transform
```

The transform this collider operates from. Usually the same as the controller, but it can be set to something else in order to mutate the shape of the controller (e.g. attach it to a bone, or attach it to a transform that changes when you duck).

```
public Vector3 offset
```

Offset of the raycast starting point from the transform.

```
public float distance
```

How far the ray is cast.

```
public RC_Direction direction;
```

Direction the ray is cast in.