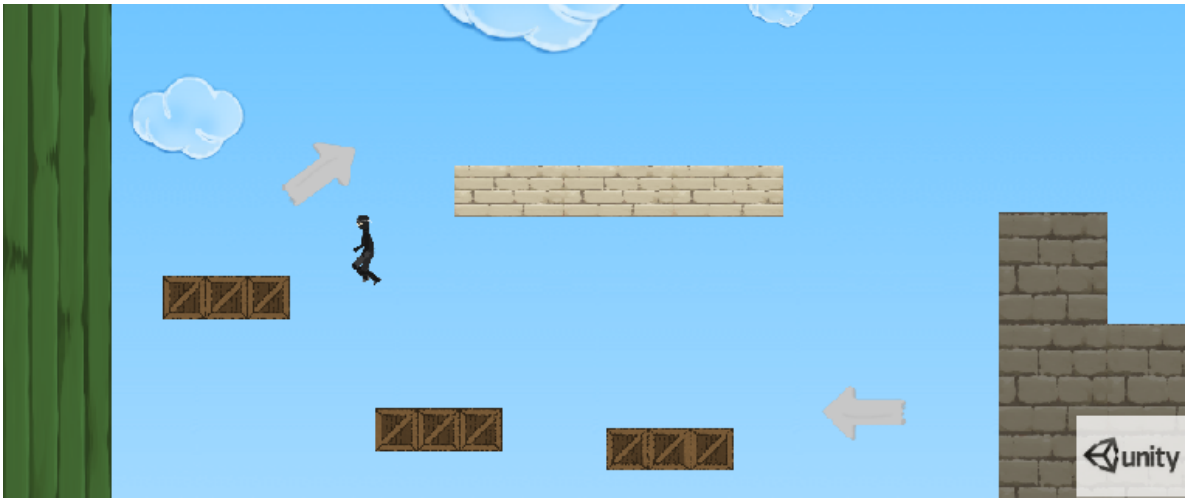
Advanced Platformer 2D

Documentation v1.6



Make your own 2D platformer!

Advanced Platformer 2D is a complete toolkit package dedicated to 2D platformer video games using Unity game engine.

For beginners and experienced users, it is made of a full set of features to help you creating your own 2D platformer.

Use available sample objects or create yours. Quickly customize behaviors with user interface settings or do some scripting for more advanced customization.

Features:

- **Optimized** for 2D : it only uses Unity 2D native objects included from version **4.3** (physic, sprites, animations....).
- Real **kinematic character** : movement is entirely controlled and mastered (not using physic dynamic motion).
- a 2D character raycast "**move and collide**" toolkit => used to move a character in kinematic motion and collide properly with environment
- a complete **character controller** for common behaviors : **walk, jump, slide, crouch, wall jump**...
- many **game objects & toolkits** : camera, ladders, railings, moving platforms, parallax scrolling, finite state machine...
- powerful and generic **melee/ranged attack** mechanism
- full **user interface setup :** configure everything immediately
- lots of **prefabs** objects to help you create your game quickly, including **sprites & animations sets**
- **samples** scripts to initiate begginers and advanced users on how to create custom behaviors and interactive objects
- **EasyTouch** asset support for touchpad management (add your own plugin quickly if needed)
- all Youtube **demo and tutorial levels** included
- full **C# accessible source code**, optimized and documented
- quick **support** and custom development

Visit online website for more uptodate information at https://sites.google.com/site/advancedplatformer2d
More information may be available on Unity forum : http://forum.unity3d.com/threads/246017-RELEASED-Advanced-Platformer-2D
You can contact me at **uniphys2D@gmail.com**

Contents

Important Notice

This is the online documentation page of the latest release : **version 1.6**
This documentation is also available offline, check into your asset folder.
Text with an orange background is the updated documentation from previous release, this helps users who want to know what happened here.

Keep in mind that you can reuse this product for any commercial or non commercial projet.
Any modification is allowed but please keep a link to this asset somewhere and don't hesitate to send me small video of your project, I will post it on Youtube.

Asset is using only Unity native 2D objects integrate from version 4.3. I mean sprites, physics and all elements using 2D suffix if possible.
This way the asset is fully optimized for 2D. For now 2.5D is not supported (render 3D mixed with physic 2D) but I will work on it as soon as possible.
Stay tune!

All values are expressed in meters, kilograms and seconds (position, velocity, mass...).
This is the convention of most physic engines, that's why we respects this.
You will have to scale these values to fit your needs if needed.
All visible settings are serialized, undoable and prefabizable.

Animations are using new animation mechanism. You'll have to make sure you are using new Animator component.
Notice that all animations names corresponds to an AnimationState inside the Animator and not a key name.

Installation

Asset is a complete project template. This allows you to build a game from scratch with no coding at all, just by using provided prefabs and scripts.
You can customize behavior easily by tuning values accessible in UI.
For more advanced users having some coding knowledge, it is also possible to customize behavior in a more advanced way, by using provided samples scripts and toolkits.
Please have a look on Youtube channel for tutorial and demos if you can't get enough information here.

Some important projects settings are provided inside ProjectSettings folder : **InputsManager, TagManager** and **Physic2DSettings**.
Please use them directly (overwrite into your project folder) or update your own settings to make sure everything is correctly configured.
Have a look a script execution order, this must match AP2D project setting too.

Load the Basics level and make sure everything works well. You can have a tour on other levels too and see what is available.

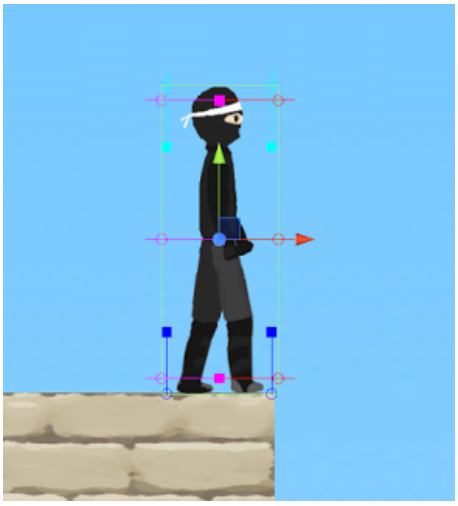
Normally you should not reference directly any provided resource in your game (sprite, animation, prefabs, animator etc...), because these may be updated/deleted during new versions and this could break your game. Instead you must duplicate each resources and put it in your own game folder. Then customize these resources at your will : settings, sprite, animation, fx etc....The only thing that you do not need to duplicate are all the scripts as they should be always compatible from one version to another.

Please delete all the files of the package just before updating to a new version to avoid error (as some files are moved from one folder to another).
Warn me if any mistake happens at this time.

CharacterMotor

This script offers same capabilities of classic CharacterController but exclusively for new Physic2D engine.
It is responsible for handling collision detection properly when moving the character.
For now it is only compatible with **BoxCollider2D**, so you must add one to your game object before using this script.

Currently collision detection is done using ray casts in Physic2D world.
This is an approximation as you don't detect collisions on entire box volume, so you can get drawbacks but in common case this works very well for 2D platformers.
You must be aware of this when designing you character and your level.
All of this is well explained into this nice article.You should look at it, even if AP2D do not use exactly the same principles.

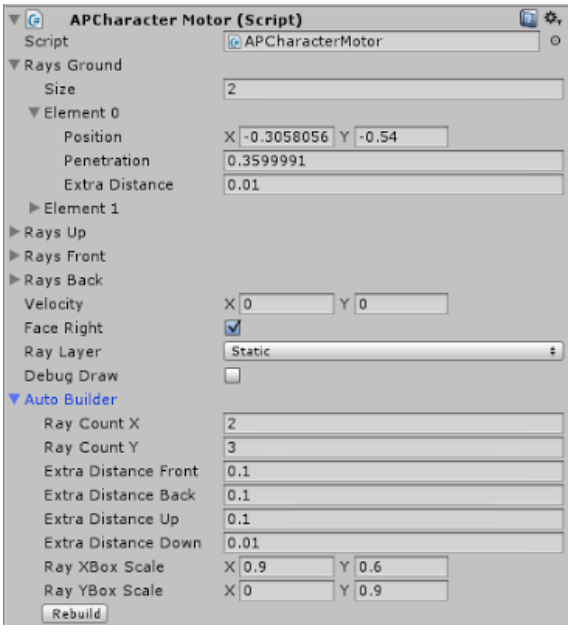


A ray has a starting position (represented as a square) and a ending position.
The small sphere is used to check the intersection with the box collider, indeed each ray will try to prevent penetration inside the collider.
Remaining path, after the sphere, is called **extra distance**, this is only for detecting surrounding environment and this is often used for scenaric purpose.
In this example we use two rays for detecting ground/ceiling and three rays for front/back walls.

Finally keep in mind that a ray detects collisions only along its path, so take care with your environment or you may experience penetration issues.
Add more rays if needed or tune your collisions to prevent this.

Configuring the rays can be a complex task, that's why you should use the **AutoBuilder** feature.
Default settings are well suited in most situations so you should keep it as this.
Ask for help if needed, I can help you in configuring your rays for you specific character.

If needed you can adjust positions in editor, each ray can be grabbed and moved freely.
You can also adjust the **Extra Distance**, this is only used to gather environment information around character.
Ground skin width should always be near zero otherwise you character may float in some cases.



Finally if you want to use it in your script, update motor linear velocity (rotation is not supported for now), then use the **Move**() API.
The character will move using its current velocity and will stop properly on any detected collision along path. Linear velocity will be recomputed by the way.
We will try to correct penetration errors too, thus by pushing character in right direction.
Lots of service are available, as getting collisions information during last move, changing collision filter, scale the rays...

Notice that triggers colliders are ignored.
Please have a look at source code if interested.

- **Rays** : list of rays used for collision detection
  - Position = local position of ray relative to game object transform
  - Penetration = distance between sphere and square, you should never change this value, this is computed for you
  - Extra distance = distance to add after the sphere for detection surrounding environment
- **Velocity** => current velocity of character (in m/s)
- **Face Right** => tells if character is facing right, must be valid at init
- **Ray Layer** => collision filter layer used by the rays
- **Debug Draw** => enable debug drawing in Scene view, this information is serialized
- **Auto Builder**
  - Ray Count X = number of ray along X axis
  - Ray Count Y = number of ray along Y axis
  - Extra Distance = extra distance to add for each ray
  - Ray Box Scale X = scale factor to apply to box collider for positioning rays along X axis
  - Ray Box Scale Y = scale factor to apply to box collider for positioning rays along Y axis
  - Rebuild = launch rebuild process

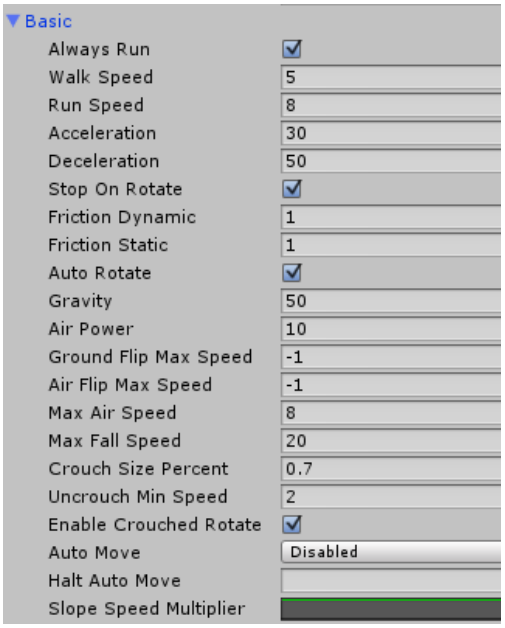
CharacterController

Motion could be handled by Physic engine in dynamic motion, but this is not the recommended method for many reasons.
That's why CharacterController uses CharacterMotor to move properly and handle collisions, thus we have full control over character dynamic.
So you must attach a CharacterMotor to your character before attaching this script.

Finally script handles complete character dynamic in function of inputs, configuration and context in which player is.
More over this offer many services and API so it can be used by other scripts.

Basics

This concerns settings for standard behavior (walk, run, jump, in air, crouch...).

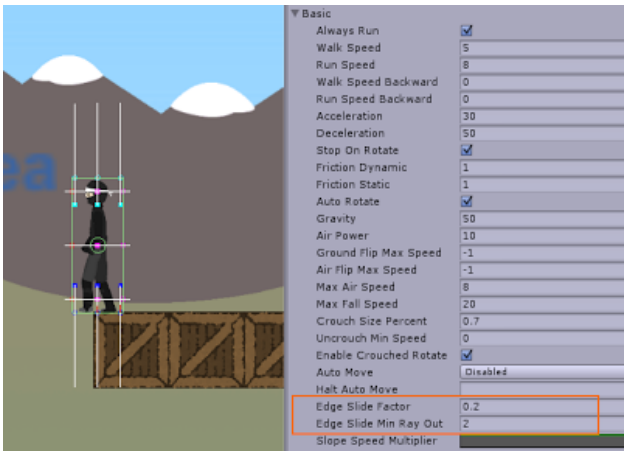


- **Always Run** => is character always running, i.e always using Run Speed
- **Walk Speed** => speed when walking (m/s)
- **Run Speed** => speed when running (m/s)
- **Acceleration** => max acceleration (m/s²)
- **Deceleration** => max deceleration (m/s²)
- **Stop On Rotate** => makes player stops immediately when changing walk direction, otherwise progressively decelerate
- **Friction Dynamic** => friction when pushing input, must be in [0,1] range
- **Friction Static**=> friction when releasing input, must be in [0,1] range

- **Auto Rotate** => character flips itself when input direction change is detected
- **Gravity** => gravity acceleration while in air (m/s²)
- **Air Power** => input acceleration while in air (m/s²)
- **Ground Flip Max Speed** => maximum speed at which player can rotate while on ground
- **Air Flip Max Speed** => maximum speed at which player can rotate while in air
- **Max Air Speed** => maximum horizontal speed while in air (m/s)
- **Max Fall Speed** => maximum downfall speed (m/s)
- **Crouch Size Percent** => size scale when crouched
- **Uncrouch Min Speed** => minimum speed applied to character if stuck while trying to uncrouch
- **Enable Crouch Auto Rotate** => enable/disable player flipping while crouched
- **Auto Move** => force character to move automatically in one direction
- **Halt Auto Move** => auto move is paused while this input is pushed
- **Slope Speed Multiplier** => curve to scale speed in function of slope angle (in degree)

Edge Slide

This allows us to make the character slide down when located near an edge.
This can be needed sometimes if you want your character to fall down when just a small part of the sprite is lying on the ground but most of it is in the air.
These settings are in the ***Basic*** configuration settings of the APCharacterController.

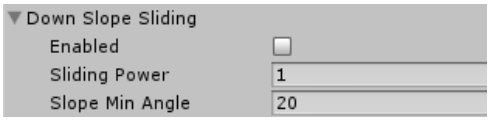


Technically, the engine detect if one or more rays from the front of the character are not touching the ground whenever character is on the ground.
If this is the case, an horizontal slide force is applied onto the character until it falls down.

- **Edge Slide Factor** => enable the feature if positive only. Amount of power to make the character slide along the ground in order to fall down
- **Edge Slide Min Ray Out** => minimum count of rays to lie in the air for activating the sliding force

Down Slope Sliding

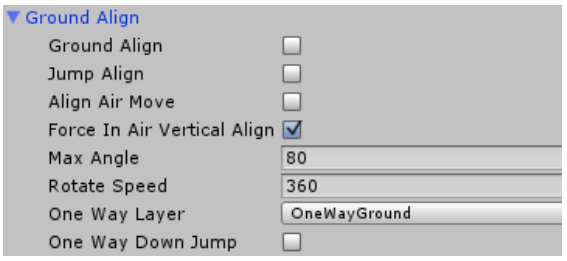
If enabled, this allows player to slide down along slopes only if in stand position (i.e not moving forward).
Indeed this is enabled only in player is releasing any forward input.
When it is active, this is like character is on a icy surface with some gravity applied to it.



- **Enabled** => enable or not this behavior
- **Sliding Power** => sliding power (acts like gravity)
- **Slope Min Angle** => minimum slope angle at which behavior is activated

Ground Align

You can setup some special behavior related to ground alignment.

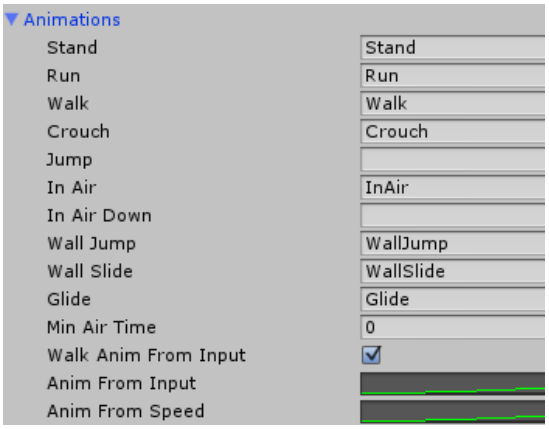


- **Ground Align** => align player along ground normal
- **Jump Align** => jump in direction of ground normal
- **Align Air Move** => move player while in air along its facing direction if enabled or only horizontally if disabled

- **Force In Air Vertical Align** => enforce player to always stay aligned with vertical axis
- **Max Angle** => maximum slope angle for which ground aligned is applied (0 = horizontal, 90 = vertical)
- **Rotate Speed** => maximum rotation speed (degrees/s) used for aligning player with ground normal
- **One Way Layer** => collision layer used to detect one way grounds (i.e all game objects using this layer will be recognized as "Mario style" one way ground objects)
- **One Way Down Jump** =>allows you to jump down from a one way ground when jump & down buttons are pressed at the same time

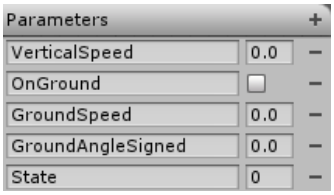
Animations

List of animations name for most moves.
Animations are using new animation mechanism. You'll have to make sure you are using new Animator component.
Notice that all animations names corresponds to an AnimationState inside the Animator.



- **Min Air Time** : minimum time while in air before launching **"In Air"** animation
- **Walk Anim From Input** : if checked, speed of walk/run animation is computed from filtered input, otherwise from ground speed
- **Anim From Input** : animation speed in function of input
- **Anim From Speed** : animation speed in function of ground speed

You can completely customize/override default animations behavior.
There are differents ways to achieve this, please check Youtube tutorial for more information.
Just keep in mind that each animation can be overrided easily,
plus you can add more complex behaviors by using available Animator parameters listed below and cool Unity Animator transition system :



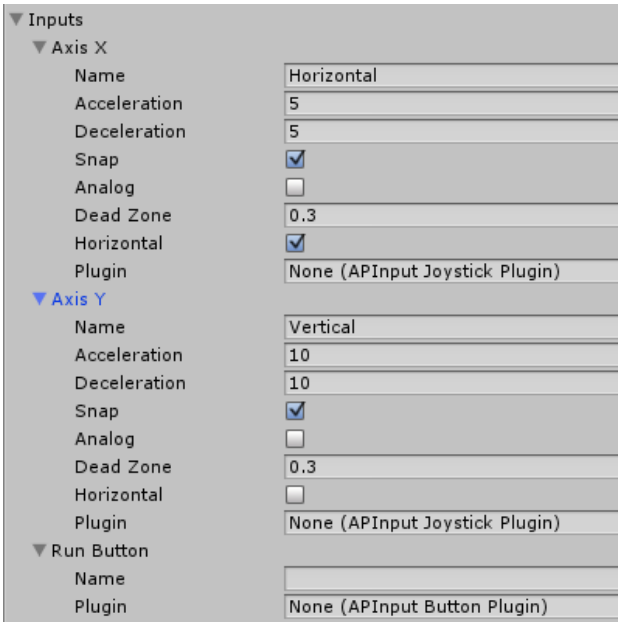
- **VerticalSpeed :** the vertical speed of the character in m/s, can be used to switch between inair up and inair down animation for example
- **OnGround :** tells if the character is touching a ground with its feet or not
- **GroundSpeed :** speed of the character along ground surface (valid only if touching the ground)
- **GroundAngleSigned :** angle in degrees of the ground the character is lying on, for exemple:
  - 0 = horizontal
  - 45 = slope up of 45 degrees
  - -30 = slope down of 30 degrees
- **State :** current internal state of the character, list is available in State enum of the APCharacterController script :

- Standard = 0
- Crouch = 1
- WallJump = 2
- WallJumpInAir = 3
- MeleeAttack = 4
- RangedAttack = 5
- Glide = 6
- WallSlide = 7
- Shift = 8

Inputs

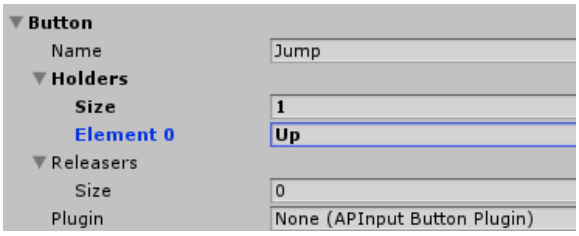
Each input has a name, this must match input name in project input settings, this is the Unity input.
When axis is used, the raw value (i.e value at which input is physically) is always used instead of the Unity filtered value.
Indeed, we handle our own input filtering mechanism but we use same principles.

When a digital input is used for an axis (e.g keyboard or touchpad), some parameters are nonsense and then they are not used.
But some analogical devices (such as joysticks) can have different analog values and may be filtered if needed (e.g DeadZone).
Default values should fit all needs but you can tweak them if really needed.



- **Axis X/Y** : used for horizontal/vertical moves. Each axis has value in [-1, 1] range.
  - **Name** : name of Unity axis input to use (check you project input settings for name matching => Input Manager)
  - **Acceleration** : max speed at which input can accelerate (only used for **Anim From Input** mode, cf. Anims)
  - **Deceleration** : max speed at which input can decelerate (only used for **Anim From Input** mode, cf. Anims)
  - **Snap** : immediate switch from one sign to the other (i.e does not decelerate if changing axis direction suddenly on device input)
  - **Force Digital (analog devices)** : force value to be digital (result value will only be -1, 0 or 1)
  - **Dead Zone (analog devices)** : value is always forced to 0 if input device raw value is inside this threshold (only useful for analog devices)
  - **Plugin** : allows you to override default Unity input raw value, but use yours instead (for example to handle touch input)
    - this is where you reference an EasyTouch plugin script for example
- **Run Button** : button to use if Always Run is disabled
  - **Name** : name of the button in your project input settings => Input Manager
  - **Plugin** : allows you to override default Unity input raw value, but use yours instead (for example to handle touch input)
    - this is where you reference an EasyTouch plugin script for example

When a button is used inside a specific action (e.g Jump), you will see this kind of configuration if unfolded :



Name is always the Button name used in Unity Input Manager.

Holders is the list of buttons that must be maintained while pressing the main button ('Jump' button in this example)
This allows you to make combination of buttons for launching a specific action.

Releasers is the opposite, this is the list of buttons that must be released while pushing the main button.
This is to handle properly different actions with same main button but with differents Holders.
For example, you want to jump with the 'Jump' button and you want to launch an attack with 'Jump' + 'A' buttons.
You must add 'A' as a Holder in your attack action, and add it as a Releaser in your Jump action.
Otherwise, pressing 'Jump' + 'A' will sometimes launch the classic Jump action instead of the attack.

EasyTouch Plugin

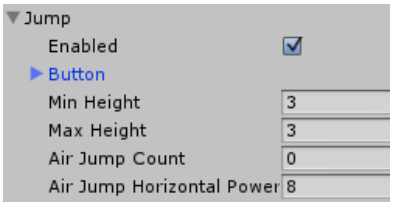
Inputs behavior can be overriden by EasyTouch (only if you bought this asset!).
Simply download and add these scripts to your project : APEasyTouchPlugin
Add each script to your matching instantiated EasyTouch input :

- APEasyTouchJoystick to EasyTouchJoystick
- APEasyTouchButton to EasyTouchButton

Finally reference the APEasyTouchJoystick/APEasyTouchButton script instance in **Plugin** property in your APCharacterController input slots.

Jump

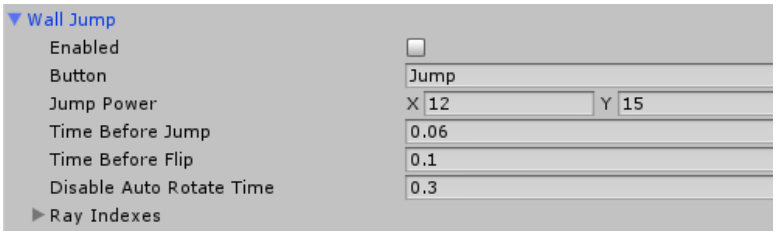
A character can jump if touching the ground and if jump input is pushed.
Input must have been released prior jumping.
There is a small tolerance if player push again jump button 0.1 second just before landing.



- **Enabled** : enabled status (can be updated in script)
- **Button** : Unity input button name to use
- **Min Height** : minimum height when jumping (in meters)
- **Max Height** : maximum height if player continues pushing jump button
- **Air Jump Count** : number of additional jumps you can make while in air (put 1 for double jump)
- **Air Jump Horizontal Power**: defines the horizontal power when doing additional jumps

Wall Jump

Character can do wall jump. Wall jumping is possible if facing a wall close enough (used by Front/Back ray extra distance) and if jumping at this time.
Player must not touch the ground at this time.
You can activate/deactivate this feature for all objects in the scene.
More over you can override this per game object by using material.



- **Enabled** : enabled status (can be updated in script or per game object)
- **Button** : Unity input button name to use
- **Jump Power** : power to use when jump against wall
- **Time Before Jump** : time to snap player on wall before jumping
- **Time Before Flip** : time to flip player after jumping on wall
- **Disable Auto Rotate Time** : time to deactivate auto rotate after jumping on wall
- **Ray Indexes** : list of rays index to use for detecting front wall, 0 ray means all rays

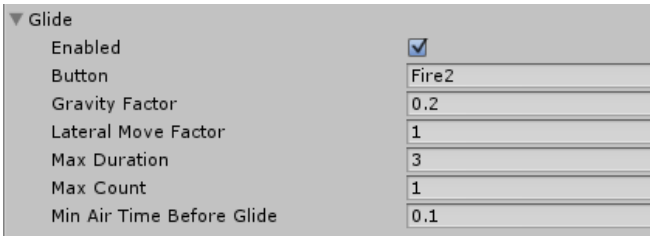
Wall Slide

When player is moving down while facing a wall and pushing input against this wall then it is going into wall slide.
Wall sliding will switch player animation and change dynamic to simulate wall friction.

- **Enabled** : enabled status (can be updated in script or per game object)
- **Friction**: wall friction during sliding
- **Min Time** : minimum time player is sliding before switching animation and changing friction
- **Min Speed** : minimum vertical down speed to switch into wall sliding state
- **Ray Indexes** : list of rays index to use for detecting front wall, 0 ray means all rays

Glide

You can add special glide ability to your player.

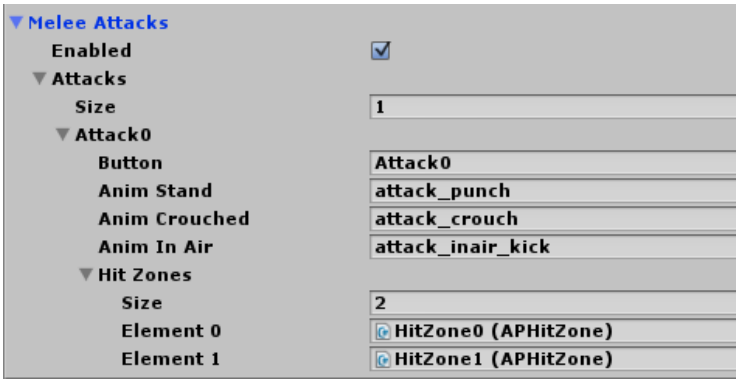


- **Enabled** : enabled status (can be updated in script or per game object)

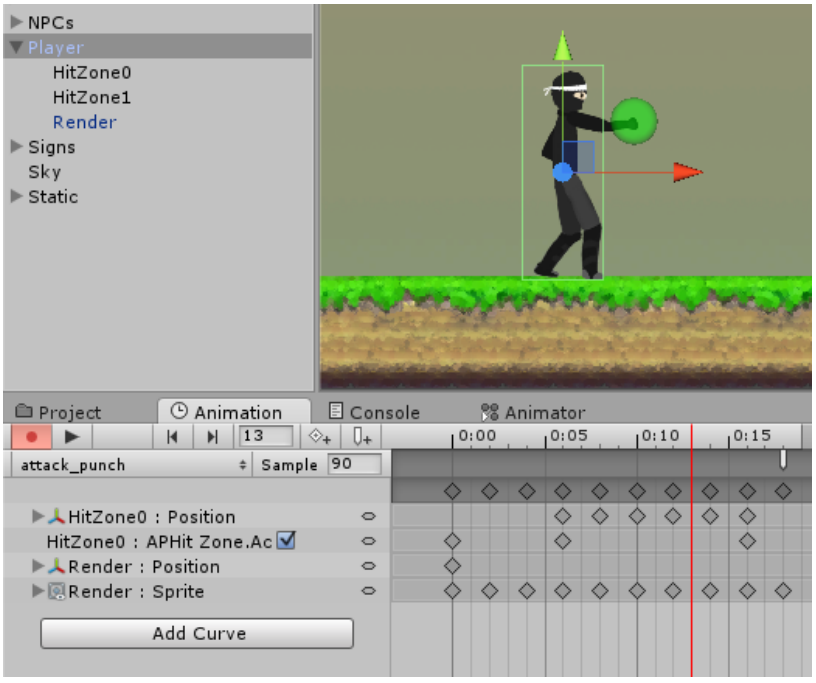
- **Button** : glide button in Unity inputs
- **Gravity Factor** : gravity scale factor
- **Lateral Move Factor** : you can add some lateral air friction
- **Max Duration** : maximum time a glide can occur
- **Max Count** : maximum glide player can make before touching ground/wall jumping
- **Min Air Time Before Glide** : minimum time player is in air before glide is allowed

Melee Attack

You character can launch melee attacks. Mechanism is relatively simple and generic.
You can add as melee attack as wanted.
A melee attack is defined by an input key, a list of animations (one per context : stand, inair, crouch...), some settings and a list of hit zones.



- **Enabled** : enabled status (can be updated in script or per game object)
- **Size**: number of melee attacks
- **Button**: name of input for launching this attack
- **Anim**: name of animation in function of context (while standing, crouched, in air, etc...)
- **Hit Zones**: list of hit zones used for collision detection



A **HitZone** is simply a small sphere used for hit detection with other game objects. You can add as many hit zone as desired to your game object.
To achieve this, you have to create an empty game object under your player game object and add a HitZone script to it.

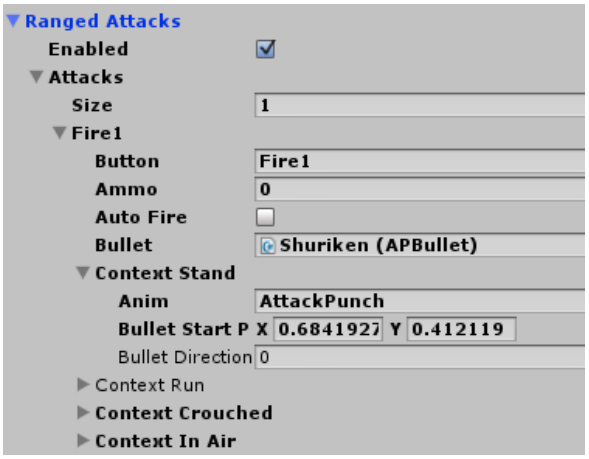
Then under your melee attack configuration, add an element to the array of Hit Zones and point your newly created HitZone game object.
Notice that any hit zone not listed in this array will be ignored.
You can animate your hit zone as you wish in your melee attack animation (local position, radius, active status).

By default an HitZone is not active at init. You must change active status inside animation. **Don't forget to uncheck active status at end of animation.**
Notice that a HitZone can be shared between many melee attacks.

Finally your animation must tell the engine when the attack has ended, to achieve this you have to add a script event at the last frame.
This event must point to the **LeaveMeleeAttack** API. If not doing so, there is a security test in engine not allowing more than 10 seconds for playing the attack.
Please check MeleeAttack demo level for easy to learn sample.

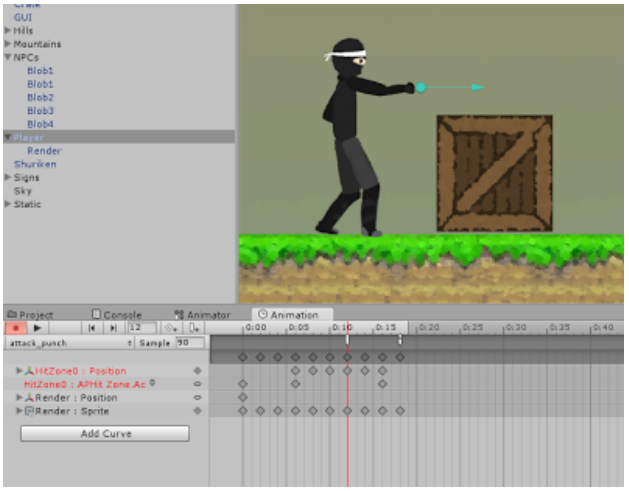
Ranged Attack

Ranged attack is similar to Melee Attack mechanism.
You can add as ranged attack as wanted.
A ranged attack is defined by an input name, a list of animations and settings for each context (stand, inair, crouch...) + some common settings.
Each attack launch bullets at desired starting position and in a given direction, these settings are also available for each context.



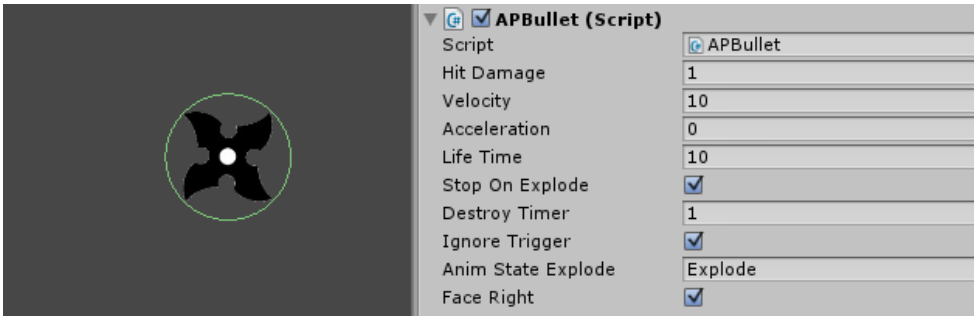
- **Enabled** : enabled status (can be updated in script or per game object)
- **Size**: number of ranged attacks
- For each attack
  - **Button**: name of input for launching this attack (disabled if null)
  - **Ammo**: number of remaining ammo
  - **Auto Fire**: enable auto fire (player can hold button to repeat fire)
  - **Bullet**: bullet game object to instantiate when spawning a new bullet (see below)
  - For each context :
    - **Anim**: name of animation state to play (disable if null)
    - **Bullet Start position**: start position of the bullet (in local character space), you can use game scene to move it instead of setting directly its value
    - **Bullet Direction :** launch direction of the bullet

Notice that bullet start position is shown as a green sphere when a context is unfolded.
You can grab the sphere to move it freely in the game scene and position it correctly according to your animation.



If a context does not use any animation (i.e. animation string is empty), it means that ranged attack is not available when player is into this context.
Notice the special case where Stand context is filled and not Run context, player will be stopped automatically while firing.

A bullet is a simple game object owning the APBullet script.
This script is used to control the dynamic of the bullet and its behavior when touching an element of the game.
The package is provided with a simple bullet script that moves straight forward and explode as soon as it touch something.
If the touched object has an APHitable script, it will be warned that a bullet has touched him.
You can override this behavior to implement your own : simply create a new script that inherits APBullet class.
Asset may provide more complex bullet dynamics later.



- **Hit Damage**: number of hit damage done (used for custom scripting)

- **Velocity**: move velocity (m/s)
- **Acceleration** : acceleration (m/s²), can be negative to slow down
- **Life Time** : bullet is destroyed from scene after this time (in seconds)
- **Stop On Explode** : immediate stop bullet when touching something
- **Destroy Timer** : time after exploding before destroying bullet, for example to allow an explosion animation to play entirely
- **ignore Trigger** : ignore trigger colliders for collision detection
- **Anim State Explode** : animation to play when exploding
- **Face Right** : is bullet facing right in Editor (needed to know if we must reverse bullet at spawn)

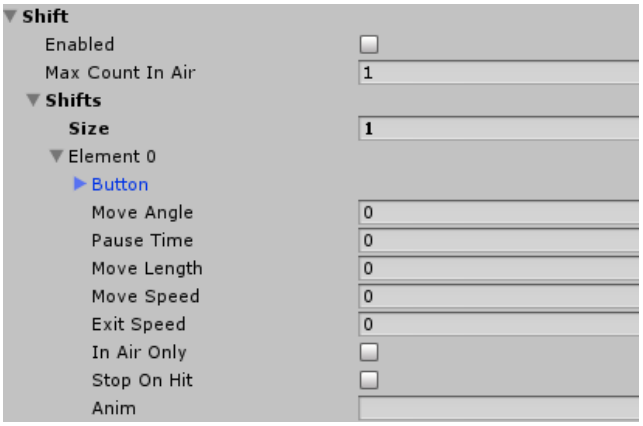
As in the Melee Attack system, your animation must tell when to launch a bullet and when animation is ended.
Simply add these events to your animation :

- **FireRangedAttack**
- **LeaveRangedAttack**

You can launch many bullets in one animation. Check AttackCrouch animation of provided ninja character for example.
A tutorial is provided on YouTube to help you creating your own attacks.

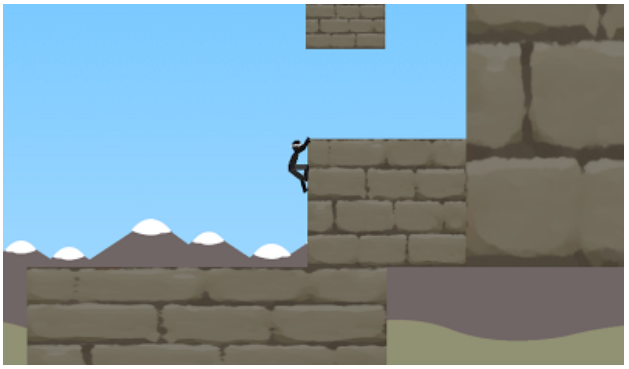
Shift

Shifting is the ability to make the character move along a straight line.
Once a shift key is pressed, the player starts to move from its current position, along a predefined direction, during an amount of time and at a desired speed.
You can add as many as shifting configuration as needed, one for each button.
With this generic approach, you can make an horizontal slide move and also a vertical down jump (check samples scripts for cool vertical down jump crate breakage feature!).



- **Enabled** : activation status of shifting ability
- **Max Count In Air**: maximum consecutive shift in air
- For each shift configuration
  - **Button** : button used to launch this shift configuration
  - **Move Angle:** direction move angle (0 = horizontal, 90 = vertical up, -90 = vertical down)
  - **Pause Time** : pause character for this amount of time before shifting (allows you to play your animation)
  - **Move Length** : length of path to move along (in meters)
  - **Move Speed** : move speed (in meters/seconds)
  - **Exit Speed** : speed at which player should move (in current moving direction) when shifting is ended
  - **In Air Only** : allow this shift ability only while in air
  - **Stop On Hit** : stop trying to move if a wall along the path is reached
  - **Anim** : animation state to launch

Edge Grab



Character is able to grab an edge.
An edge can be grabbed while in the air or while the character is lying on the ground upon the edge.
Depending on the initial grab state, appropriate animation is played.
When the character is grabbing the edge, he is allowed to :

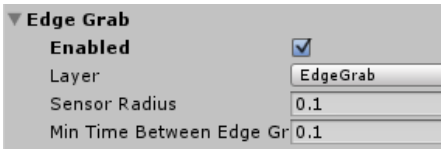
- release the grab by pushing a move key (according to settings)
- jump
- climb up the edge with a special climb animation

So you must add 3 animations for your character if you want the 3 features activated, but you can just to enable one if needed.

In order to make an edge grabbing in your scene, you have to add a EdgeGrab game object.
Please use the provided prefab for easier implementation.

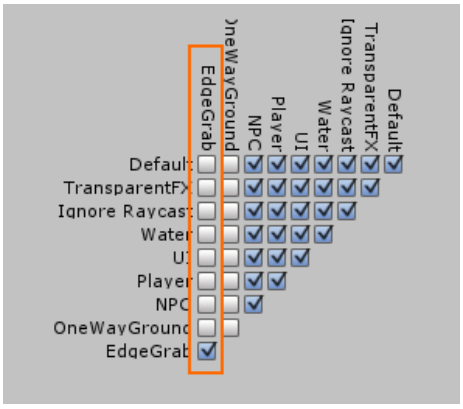
Technical details :

Settings for the character :

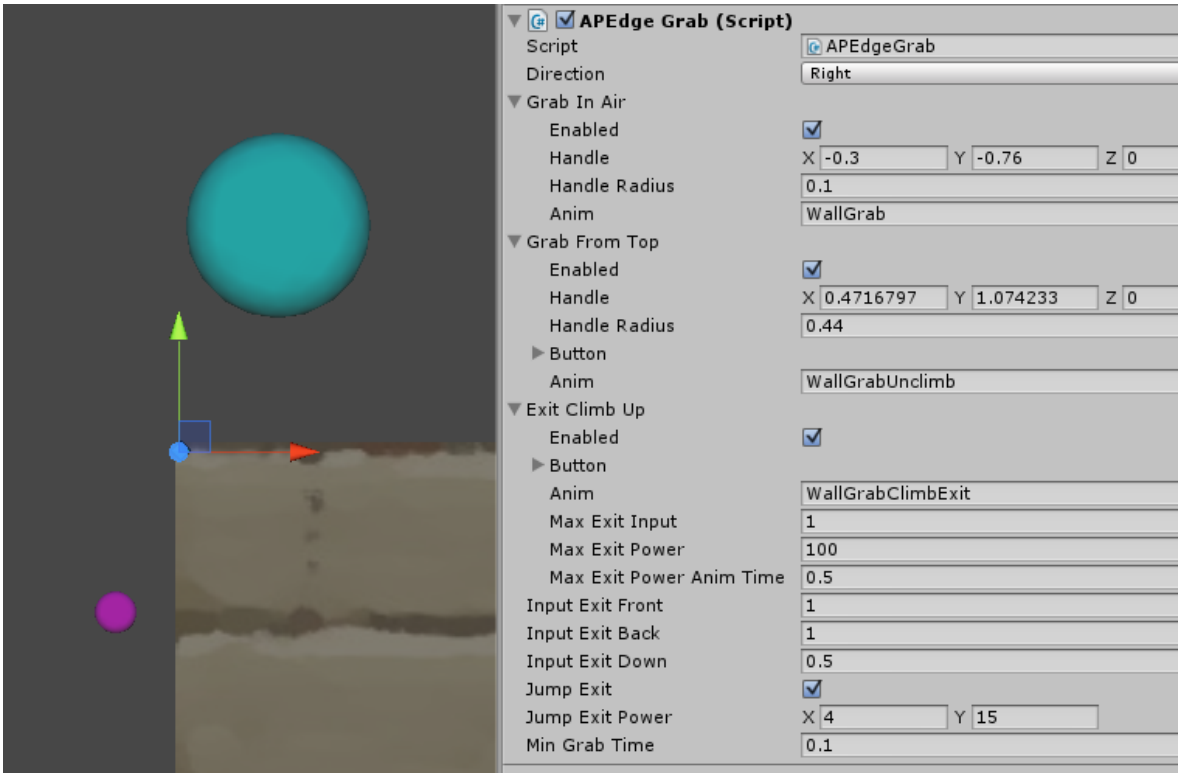


Internally, a sphere to detect edge grabs game objects is created at start of the game.
This sphere follows the center of the characters, its radius can be tuned but default provided value should be fine.
It acts like a trigger, thus it is assigned to a specific Layer in your character settings (here called EdgeGrab layer).
Whenever this trigger detects a collisions with an edge grab game object, appropriate animation is launched.

Important notice : same collision layer than the one specified in your character must be put on your EdgeGrab game objects.
This layer should collide only with itself for maximum performance. Your Physic2D layer collision matrix should be as this :



To make an edge "grabbable", you must put a specific game object on it.
Please use position of the game object as the reference point for your edge corner (as shown below).
This game object's Layer must be assigned to the previously defined Layer in your character settings.



First, an edge grab object is defined by a grab direction so that we know in which direction it is allowed to grab this edge : left or right.
Then you can setup your different grabbing enter context, each one having a detection sphere called **Handle** :

- Grab in the air (**magenta sphere**)
- Grab from the top of the edge (**cyan sphere**)

You can change each sphere position and radius.
Anim slot is used to launch character animation when grab is detected for each entering context.
Please notice that when character enter of the detection sphere, **it is first snapped onto this sphere's center**, then animation is played.

You can setup **Input Exit** values, theses are the minimum value (between 0 and 1) at which the player must press the corresponding input in order to directly ungrab the edge.
Special value 1 is used to disabled this.

You can also allow the player to jump with the **Jump Exit** setting. A specific exit jump impulse is defined too.
Min Grab Time is used to force the player to grab at least this amount of time in seconds before allowing him to exit.

When the edge is grabbed, you can also launch a **Climb Up** animation with any input key so the player climbs up the edge and leave.
Max Exit Input is used to clamp the input value when animation ends, indeed if you want to stop your character even if player is pushing forward just put 0 here.
Max Exit Power is used to add additional moving force to your animation if player is pressing move forward key. This force starts to apply at specified **Max Exit Power Anim Time,** which is an animation ratio between 0 and 1.

Animation is quite handy here, indeed moving a physic object into an animation must be done by moving character root node directly, but this is not possible in Unity to make an animation that move your character in world space. That's why a special node has been added under character game object, this is called the "**PhysicAnim**" node.
When creating your animations, you can animate this node locally, and the engine will use this in order to move properly the character in world space.
This allows you to move the character locally to the Edge Grab game object within an animation so that it's world space position is updated (and not only the sprite).

This part is the most complicated part here, and I recommend you to make your hands with provided prefabs before making your own settings and animations.
Finally, please have a look at my Youtube tutorial in which things should be much clearer.

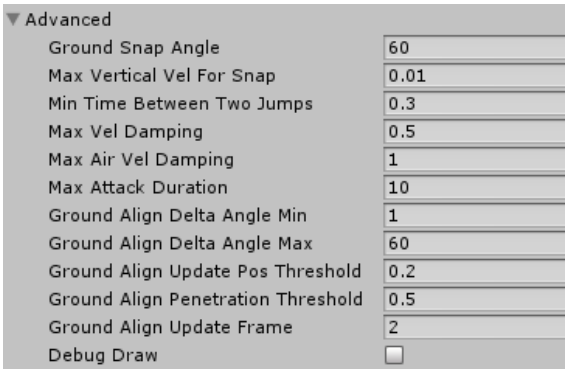
Events

If you want to do some specific scripting, you can listen to any event launched by the character.
To achieve this, simply create a new class object that inherits the APCharacterEventListener class.
Then override any appropriate method, each method is called when corresponding event occur.

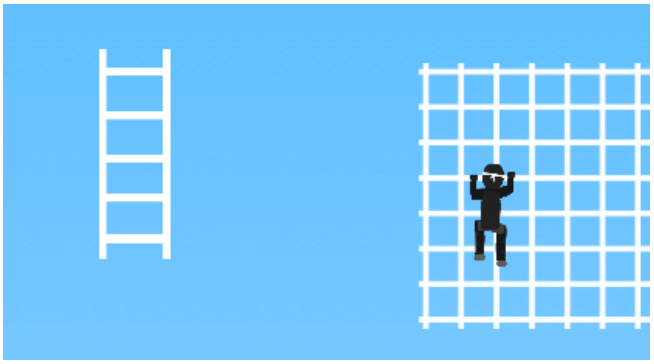
You can implement any of desired effect inside each method, this can be used for example to play an audio/fx event whenever the characters jumps, touch the ground, attacks....
Finally make sure to attach your listener properly to the listened character into your Awake method.
Please check the Audio samples to see how it is done!

Advanced Settings

Controller has some more advanced settings that are contained here.
Normally default settings should fit all needs, and you should never change these values.
In some really rare cases this may be tuned if you got some specific issues.
Please check comments for more information in APCharacterController.Settings.cs file.



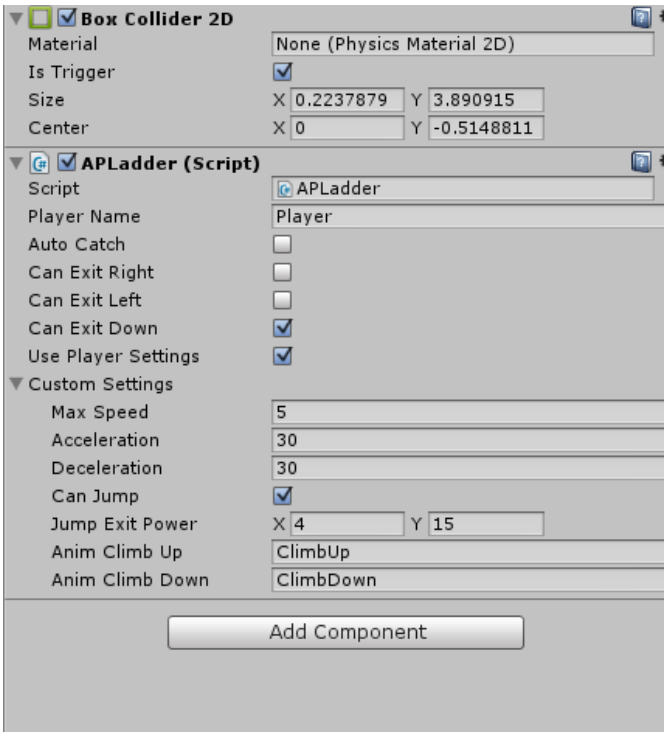
Ladder / Railings



A ladder is a game object on which character can climb up and down.
Player is snapped onto ladder as soon as it lies completely into ladder box collider zone and if pushing vertical axis (unless autocatch is enabled).
You must adds a BoxCollider2D and attach the ladder scripts to enable ladder.

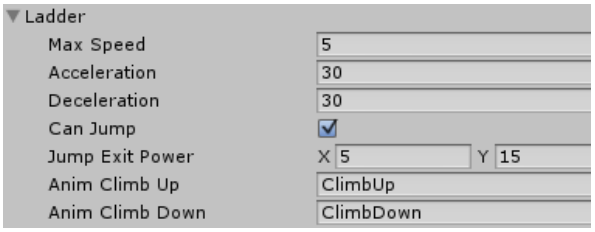
BoxCollider must be flagged as "**Is Trigger**" or be put into non colliding physic layer to prevent character from colliding with it.

Same rules apply to railings except that player can move freely on horizontal axis too.
More over you can add many BoxColliders to handle complex railings (see Basics demo level).



- **Player Name** : name of player to survey to catching
- **Auto Catch** : player is catched as soon as overlapping box collider zone, otherwise he must push up axis
- **Can Exit XXX** : allows player to exit in XXX direction if trying to
- **Use Player Settings** : use default player settings or Custom Settings defined below

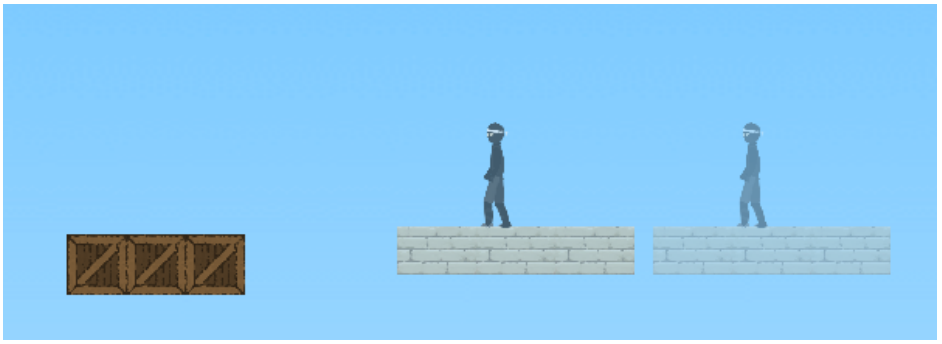
Default settings regarding behavior on a ladder/railings are carried by the CharacterController script.
But each one can be overridden by game object if needed.
For example we could make a special ladder on which we move slower than others.
You just have to uncheck "**Use Player Settings**" on concerned ladder/railings and tweak custom settings for it.



- **Max Speed** : maximum speed (m/s)
- **Acceleration** : acceleration (m/s²)
- **Deceleration** : deceleration (m/s²)
- **Can Jump** : allows jumping in facing direction while catched
- **Jump Exit Power** : jump power
- **Anim Climb XXX** : animation to use for moving in XXX direction

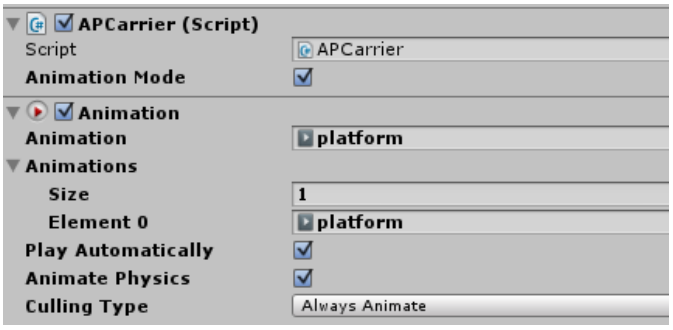
---

Carrier

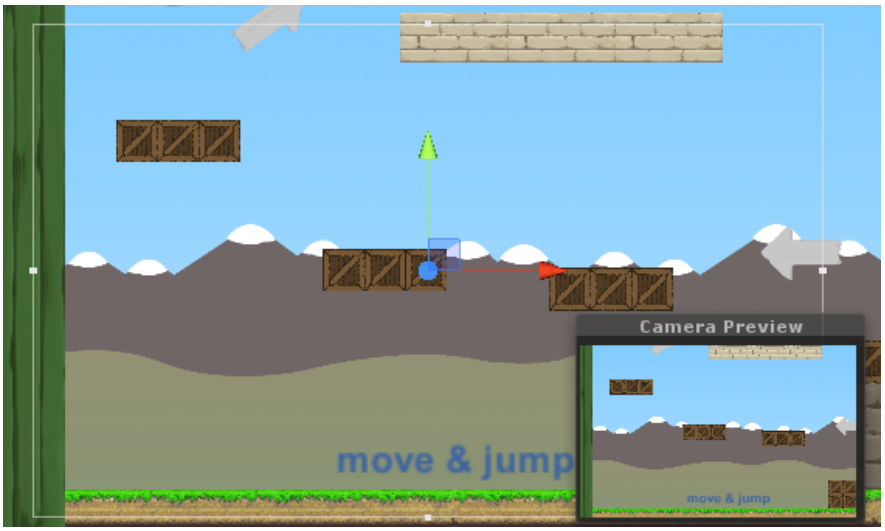


When lying on a moving object, your character may not be carried.
To enable this, you have to attach a APCarrier script on it.
Then you have 2 options :

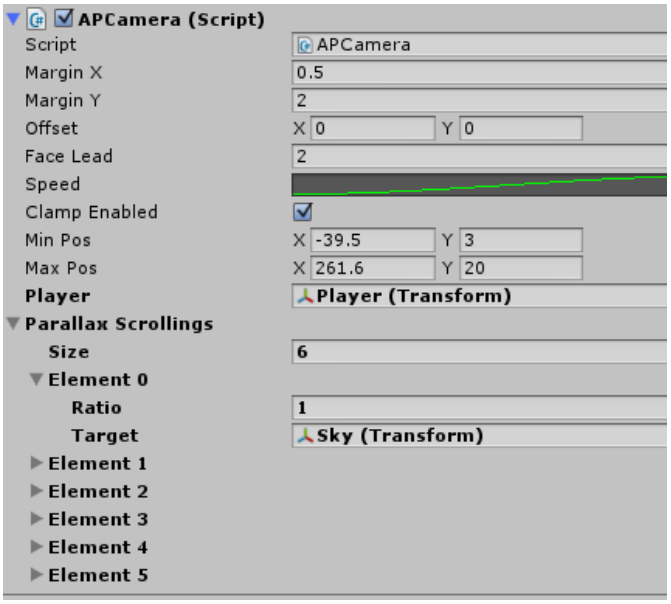
1. your carrier object is moved by native Unity animation system:
    - make sure that **Animate Physics** is checked in the **Animation** properties
    - check the box **Animation Mode** on the APCarrier script
2. carrier object is updated by code in a FixedUpdate method (like in the SampleMovingObject script):
    - uncheck **Animation Mode** on the APCarrier script.
    - make sure that your script is executed before APCarrier script (Edit > Project Settings > Script Execution Order), by default this should be ok



Camera

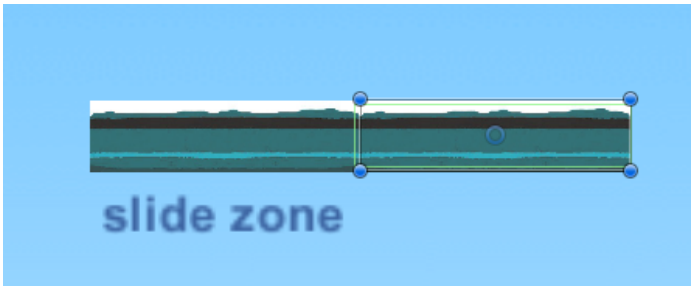


Provided camera should fits most need of any 2D platformer.
This fits standard 'Mario' style camera. Please have a look here for concise explanation : the-ideal-platformer-camera-should-minimize
Attach it to your game camera and select target transform to use.



- **Margin X/Y** : allowed inside margin for Player in which camera does not move
- **Offset** : add static offset from Player
- **Face Lead** : camera horizontal offset when Player is moving forward
- **Speed** : camera speed in function of distance to Player
- **Clamp Enabled** : enable camera position clamping
- **Min Pos / Max Pos** : min/max camera position when clamp is enabled
- **Player** : transform to follow
- **Parallax Scrolling** : enable parallax scrolling on specified transform using this camera
    - Ratio : speed ratio of scrolling, must be in [0,1] range. 1 means we follow camera at its full speed, 0 means we never move
    - Target : target transform to update

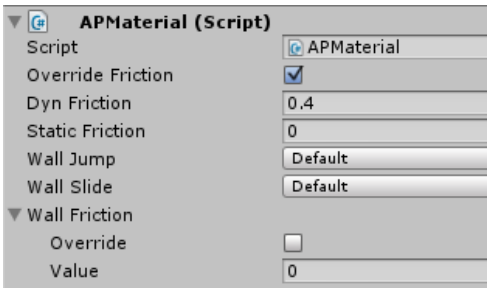
Material



The CharacterController has some settings that can be overridden in some situations : for example friction, max velocity, can walljump etc...
This is the goal of Material script if attached to a game object. So you can override default properties of CharacterController for some specific game objects.
This property will be used if character is touching a game object owning a material script.

A perfect example is the friction. By default you want no friction for your character, so you put 1 in its default setting value.
But if you want a specific ground to slide, you just have to put a new script on it, add Material and override friction value.
This new friction will be used if the character is lying on this specific ground.

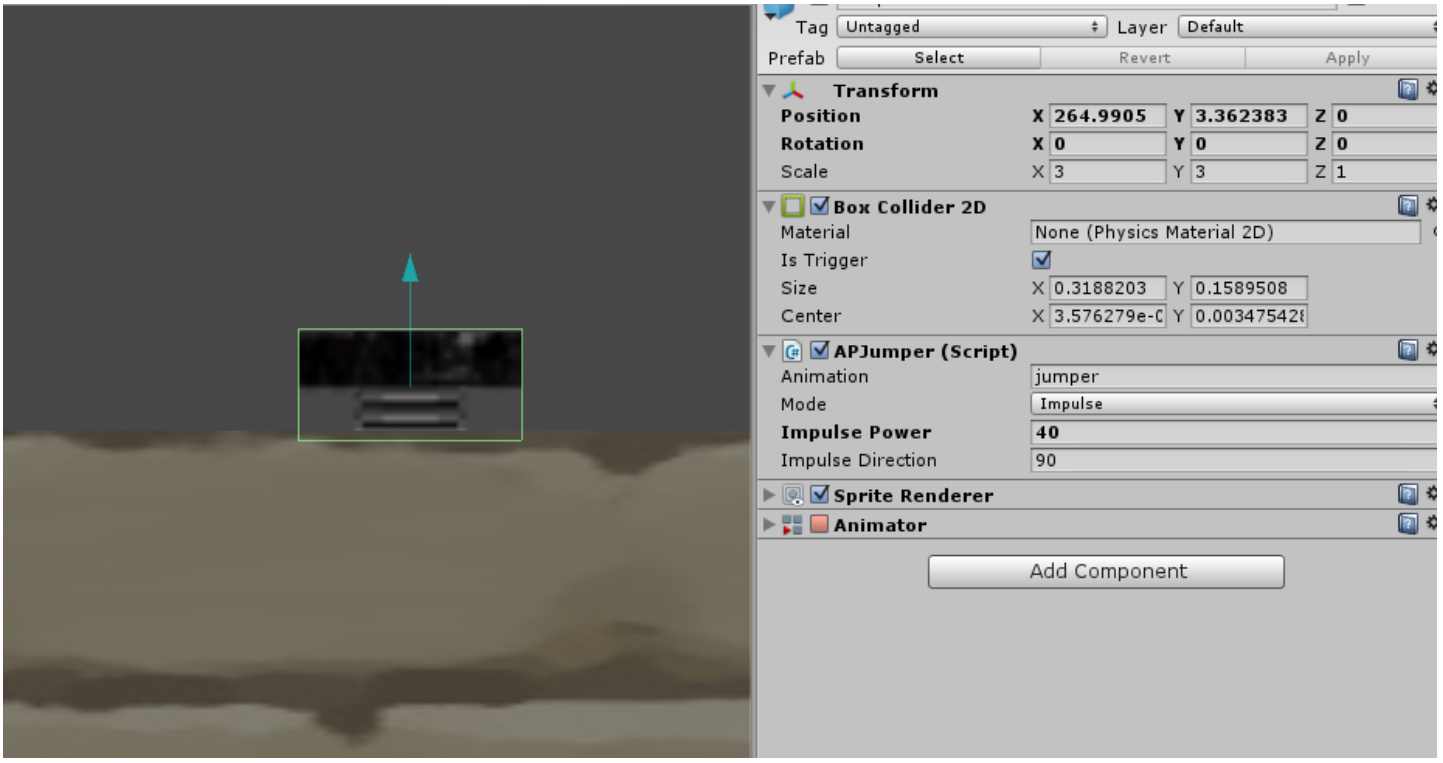
Some others settings can be overridden for example if player can do wall jump or not.
For example by default you don't allow wall jumping on your character except form some walls, this is done in demo level.
We have attached a material script on wall and override Wall Jump default value to True.



- **Override Friction** : override default player frictions or not
- **Wall Jump** : wall jump enabled status (default = keep player value, true = force enabled, false = force disabled)

Jumper

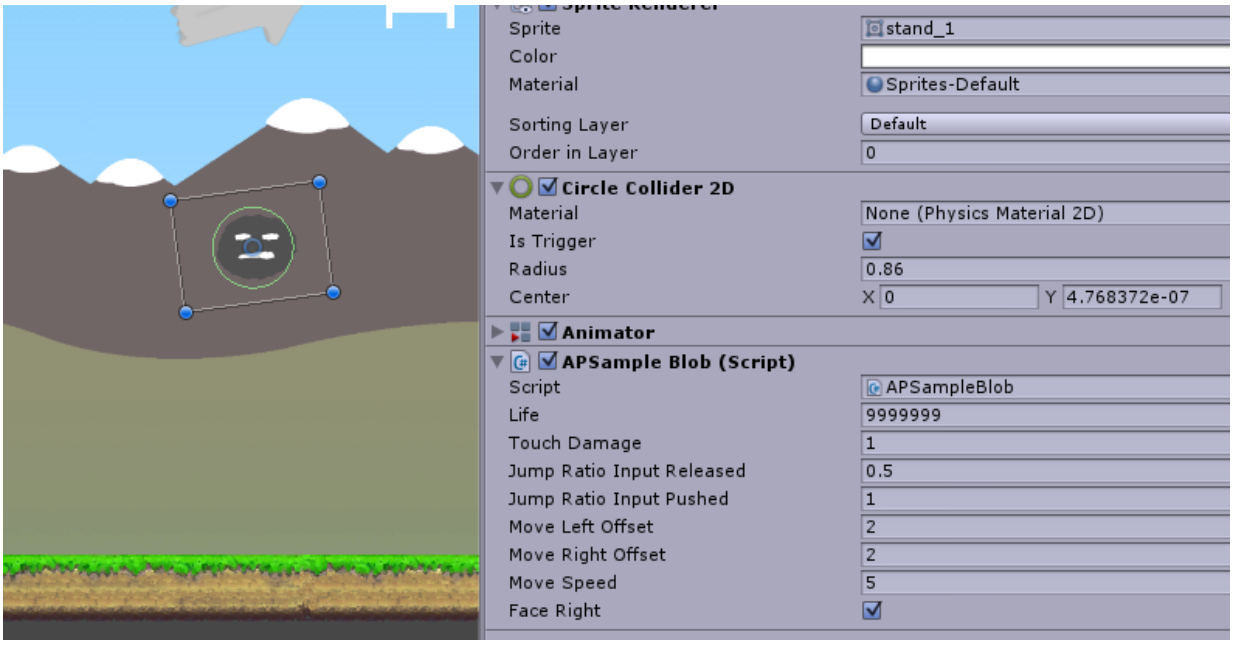
This special game object (which could have been a sample game object) behaves like a jump effector when character touches it.
An animation is played once when touched by the character.
You can choose the direction and power of impulse if you are in an Impulse Mode.
Otherwise this is a classic vertical Jump impulse with a minimum height and a maximum height as long as Jump input is pushed.
Just try it !



Technical details :

It must have a collider as a trigger for hit detection with the character.
This shape can be any supported 2D shape in Unity.
Just make sure that collision layer is correctly setup (so this layer collides with character's layer).
An animation state is played when player touches the trigger, this animation does not loop and last keyframe is the standby position.
Please use new Animator mechanism otherwise this won't work.

Hitable



A **Hitable** is a game object for which you want to be notified of some events, for example when hit by a melee attack, touched by a projectile or simply touched by a character.
Create your own script that inherits this script then add it to your game object.
You will have to override some methods for your specific scripting behavior.
Finally add a 2D collider and design your own detection zone.
Notive that character never collides with triggers and then they never receive the `OnCharacterTouch` events.

```
public class APHitable : MonoBehaviour
{
    // called when we have been hit by a melee attack
 // - launcher : character controller launching the attack
 // - hitZone : hitzone detecting the hit
 // - return false if you want engine to ignore the hit
 virtual public bool OnMeleeAttackHit(APCharacterController launcher, APHitZone hitZone) { return false; }
```

```
 // called when we have been hit by a bullet
 // - launcher : character controller launching the bullet
 // - bullet : reference to bullet touching us
 // - return false if you want engine to ignore the hit, true if you want bullet to be destroyed
 virtual public bool OnBulletHit(APCharacterController launcher, APBullet bullet) { return false; }

 // called when character is touching us with a ray
 // - motor : character controller touching the hitable
 // - rayType : type of ray detecting the hit
 // - hit : hit info
 // - penetration : penetration distance (from player box surface to hit point, can be negative)
 // - return false if contact should be ignored by engine, true otherwise
 virtual public bool OnCharacterTouch(APCharacterController launcher, APCharacterMotor.RayType rayType, RaycastHit2D hit,
float penetration) { return true; }
}
```

Please check the script APSampleBlob for more information about scripting your own NPC.
In this sample, we handle the case where the player is jumping on us or touching us.
We also manage a kind of life system for our player and npc.
Check the Melee Attack demo level for more informations.

Samples

A game can be very specific and the package could not cover every aspects of a gameplay.
That's why we try to stay as generic as possible to allow you scripting any kind of behavior.
Some samples scripts are provided to help you in creating your own scripts and behaviors, and some samples scripts may be interdependent.
All these scripts are well documented and should be easy to understand.

Look at the folder Samples and check the prefabs using this, this will help you a lot creating your own script.
Here is a non exhaustive list of behaviors created by these scripts :

- Player life system with GUI
- Restart level when falling in void
- Falling platform
- Moving platform when landing on it
- Basic npc move / hit
- Breakable crate
- Collectable (life, ammo...)
- Vertical down jump
- Audio events

This is a good place to starts scripting from.
Ask for support if you don't understand something, new samples could be added to the package!