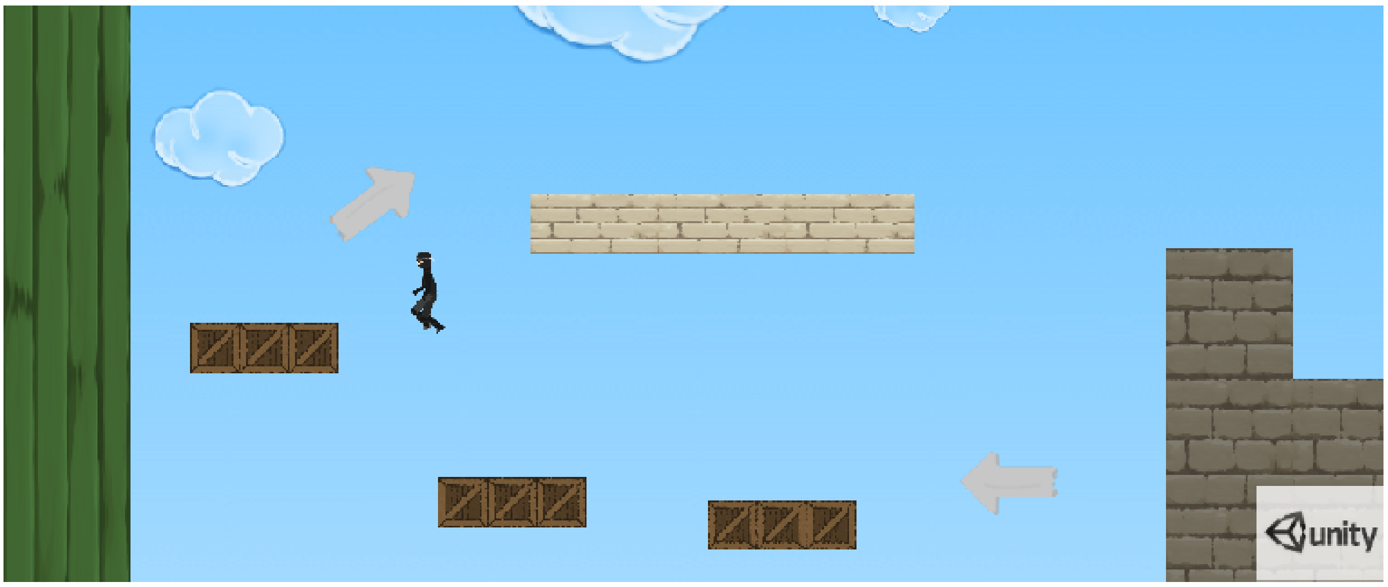
Advanced Platformer 2D

Documentation v1.4



Advanced Platformer 2D is a toolkit package for Unity dedicated to **2D platformer** video games.
It is made to assist you in creating your own 2D platformer using Unity game engine.
It is built upon new **Unity 2D** toolkit included from version **4.3** (physic, sprites, animations….).
Notice that character is not moved as a dynamic physic object, thus movement is entirely controlled and mastered.
Finnaly you can reuse every provided elements and modify them at your will to build your own game.

Features:

- 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**
- **samples** & **prefabs** objects to help you integrating, including **sprites & animations sets**
- many **demo levels** for a sample game & features explanations
- full **C# accessible source code**, optimized and documented
- implementation can be **customized** easily by inheriting classes and overriding methods (samples provided)
- quick **support** and custom development if required

Visit online website for more uptodate information at https://sites.google.com/site/advancedplatformer2d
You can contact me at **uniphys2D@gmail.com**.
Feel free to email me for any question, comment, bug or feature request. I could also make some specific code for you if needed.
Notice that everything is important for me in order to improve this package and I will try to answer as fast as possible.

Contents

Notes

This is the online documentation page of the latest release : **version 1.4**
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.

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, thus you should create a project from scratch with this project to avoid errors, otherwise simply import new files into your project.

Some important projects settings are provided inside ProjectSettings folder : InputsManager, TagManager and Physic2DSettings. Please use them to make sure everything is correctly setup.

Load the Basics level and make sure everything works well. You can have a tour on other levels too and see what is available.
Please have a look on Youtube channel for tutorial and demos.

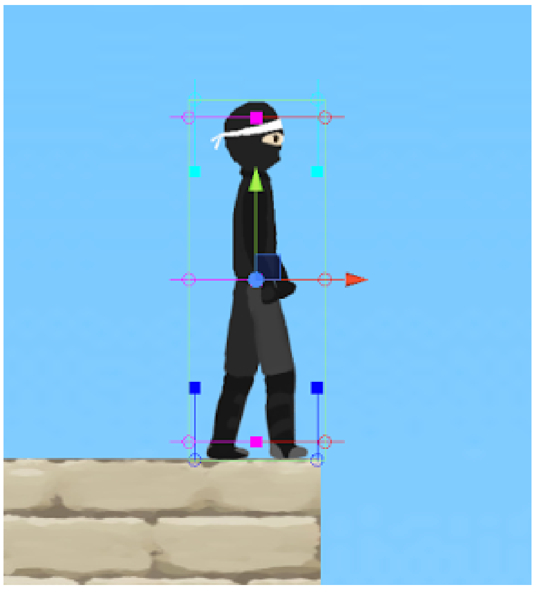
Normally you should not reference directly any provided resource in your game (sprite, animation, prefab etc...), because these may be updated/deleted at each new version and this could break your game. Instead you must duplicate resources and put it in your own game folder. Then customize these resources 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 warn me if any mistake happens after one update.

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.

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 have drawbacks but in common case this works very well for 2D platformers.
This is well explained into this nice article.
You should look at this even if we do not use exactly the same principles.
You must be aware of this when designing you character and your level.

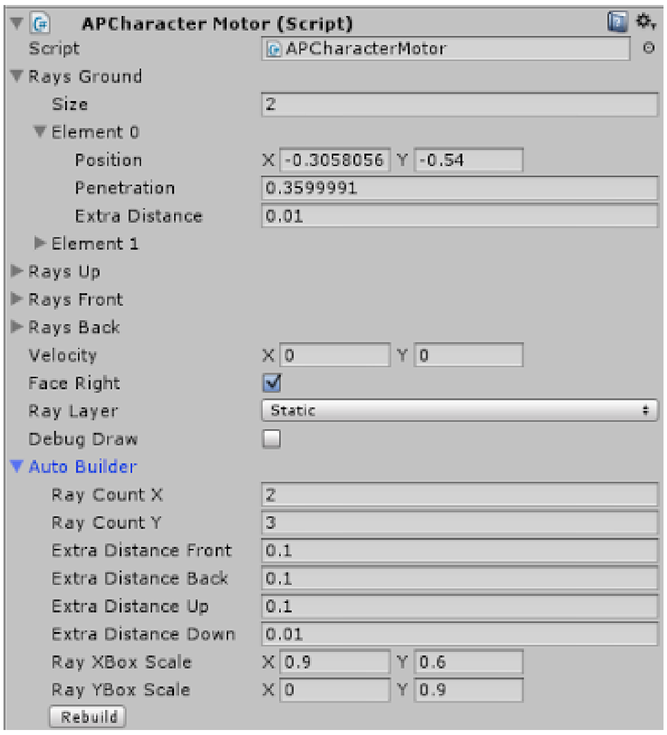


A ray has a starting position (represented as a square) and a ending position.
The small sphere is used to check the intersection with the box collider, indeed each ray will try to prevent penetration inside the collider.
Remaining path, after the sphere, is called **extra distance**, this is only for 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.


APCharacter Motor (Script)

Script	APCharacterMotor
Rays Ground	
Size	2
Element 0	
Position	X -0.3058056 Y -0.54
Penetration	0.3599991
Extra Distance	0.01
Element 1	
Rays Up	
Rays Front	
Rays Back	
Velocity	X 0 Y 0
Face Right	✓
Ray Layer	Static
Debug Draw	☐
Auto Builder	
Ray Count X	2
Ray Count Y	3
Extra Distance Front	0.1
Extra Distance Back	0.1
Extra Distance Up	0.1
Extra Distance Down	0.01
Ray XBox Scale	X 0.9 Y 0.6
Ray YBox Scale	X 0 Y 0.9

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

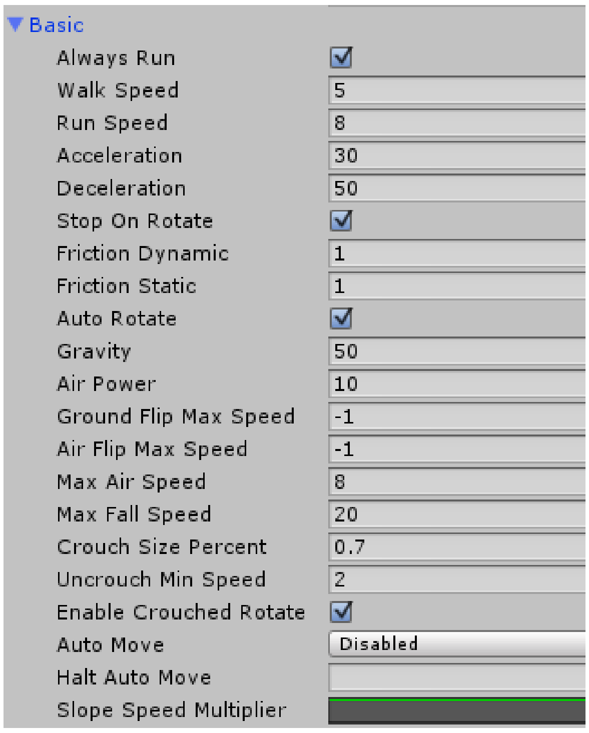
Motion could be handled by Physic engine in dynamic motion, but this is not the recommended method for many reasons.

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...).

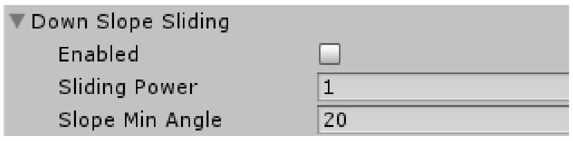


Basic	
Always Run	✓
Walk Speed	5
Run Speed	8
Acceleration	30
Deceleration	50
Stop On Rotate	✓
Friction Dynamic	1
Friction Static	1
Auto Rotate	✓
Gravity	50
Air Power	10
Ground Flip Max Speed	-1
Air Flip Max Speed	-1
Max Air Speed	8
Max Fall Speed	20
Crouch Size Percent	0.7
Uncrouch Min Speed	2
Enable Crouched Rotate	✓
Auto Move	Disabled
Halt Auto Move	
Slope Speed Multiplier	

- **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)

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.

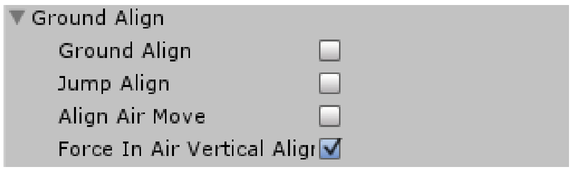


▼ Down Slope Sliding	
Enabled	☐
Sliding Power	1
Slope Min Angle	20

- **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.

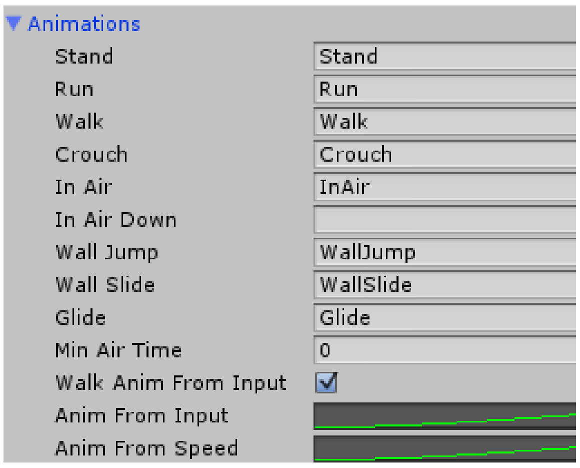


▼ Ground Align	
Ground Align	☐
Jump Align	☐
Align Air Move	☐
Force In Air Vertical Align	✓

- **Ground Aligne** => 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

Anims

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.

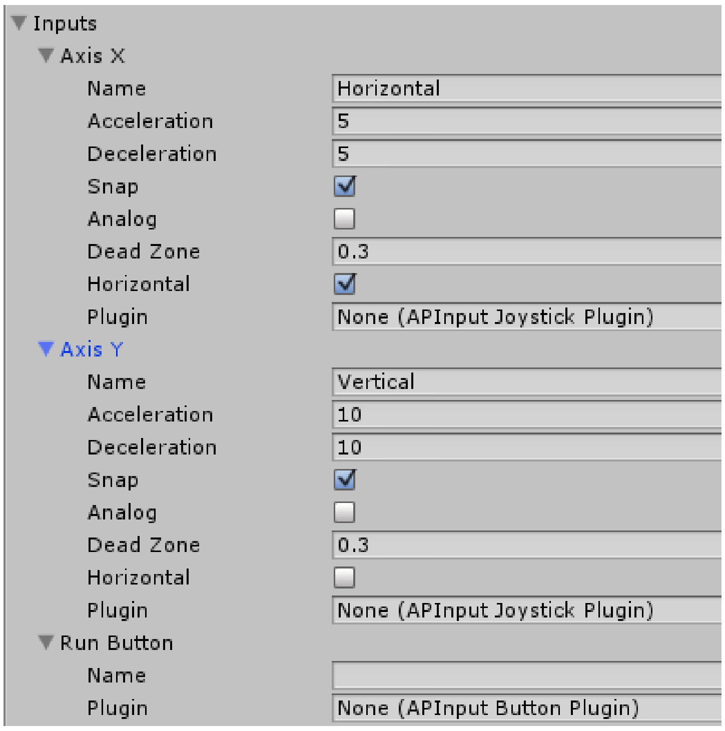


▼ Animations	
Stand	Stand
Run	Run
Walk	Walk
Crouch	Crouch
In Air	InAir
In Air Down	
Wall Jump	WallJump
Wall Slide	WallSlide
Glide	Glide
Min Air Time	0
Walk Anim From Input	✓
Anim From Input	
Anim From Speed	

- **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

Inputs

Inputs are handled by character, this replace Unity input filtering mechanism but use same principles.
An input have a raw value, i.e value at which input is physically, depending on input device.
Usually a keyboard only have two states, pushed (1) or not (0) .
But some analogic devices (such as joysticks) have analog values, which varies from 0 to 1.
This raw value may be filtered/smoothed when needed, for example to prevent sudden changes or to filter bad values.



▼ Inputs	
▼ Axis X	
Name	Horizontal
Acceleration	5
Deceleration	5
Snap	✓
Analog	☐
Dead Zone	0.3
Horizontal	✓
Plugin	None (APInput Joystick Plugin)
▼ Axis Y	
Name	Vertical
Acceleration	10
Deceleration	10
Snap	✓
Analog	☐
Dead Zone	0.3
Horizontal	☐
Plugin	None (APInput Joystick Plugin)
▼ Run Button	
Name	
Plugin	None (APInput Button Plugin)

- **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)
  - **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)
  - **Analog** : force raw value to be analog or not (result value will only be -1, 0 or 1 depending on input device raw value)
  - **Dead Zone** : raw value is always forced to 0 if input device raw value is inside this threshold
  - **Horizontal** : just tells the engine if this axis is vertical or not, you should not change this
  - **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
  - **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

EasyTouch Plugin

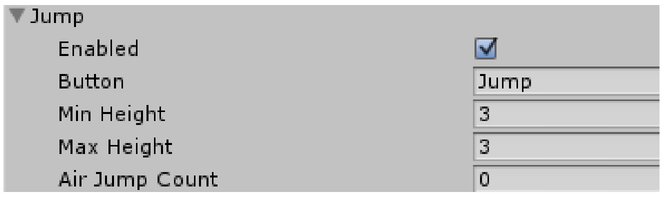
Inputs behavior can be overriden by EasyTouch if you bought the asset.
Then add these scripts to your project : https://drive.google.com/file/d/0B4Y3QGrSMqdzTTktNnVwaG1NX0k/edit?usp=sharing
Add each script to your matching instantiated EasyTouch input :

- 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.



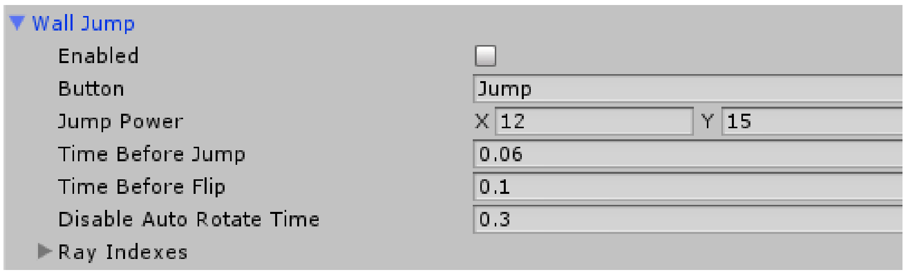
▼ Jump

Enabled	✓
Button	Jump
Min Height	3
Max Height	3
Air Jump Count	0

- **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)

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.



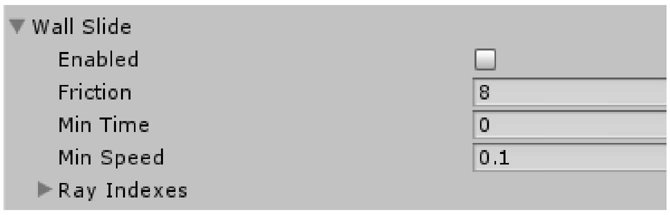
▼ Wall Jump

Enabled	☐
Button	Jump
Jump Power	X 12 Y 15
Time Before Jump	0.06
Time Before Flip	0.1
Disable Auto Rotate Time	0.3
▶ Ray Indexes	

- **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
- **Time Disable Auto Rotate** : 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.



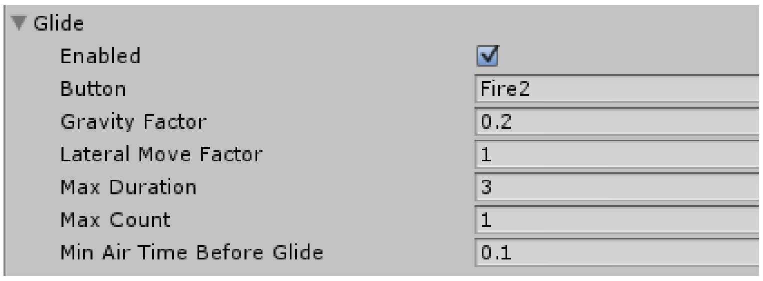
▼ Wall Slide

Enabled	☐
Friction	8
Min Time	0
Min Speed	0.1
▶ Ray Indexes	

- **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.

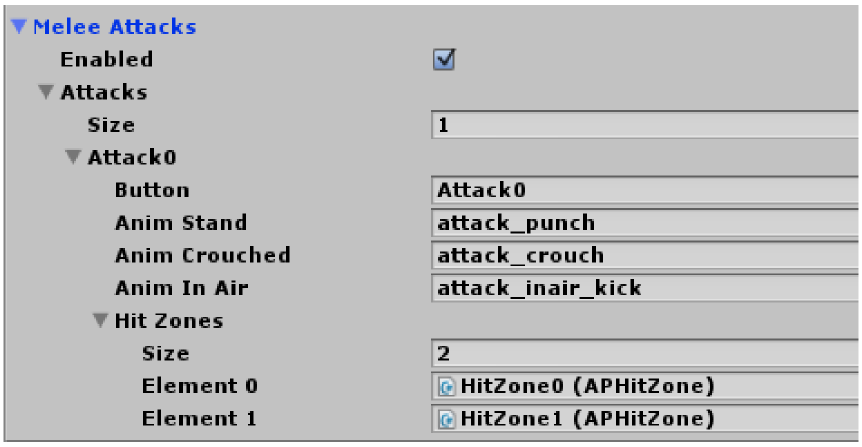


Glide	
Enabled	✓
Button	Fire2
Gravity Factor	0.2
Lateral Move Factor	1
Max Duration	3
Max Count	1
Min Air Time Before Glide	0.1

- **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

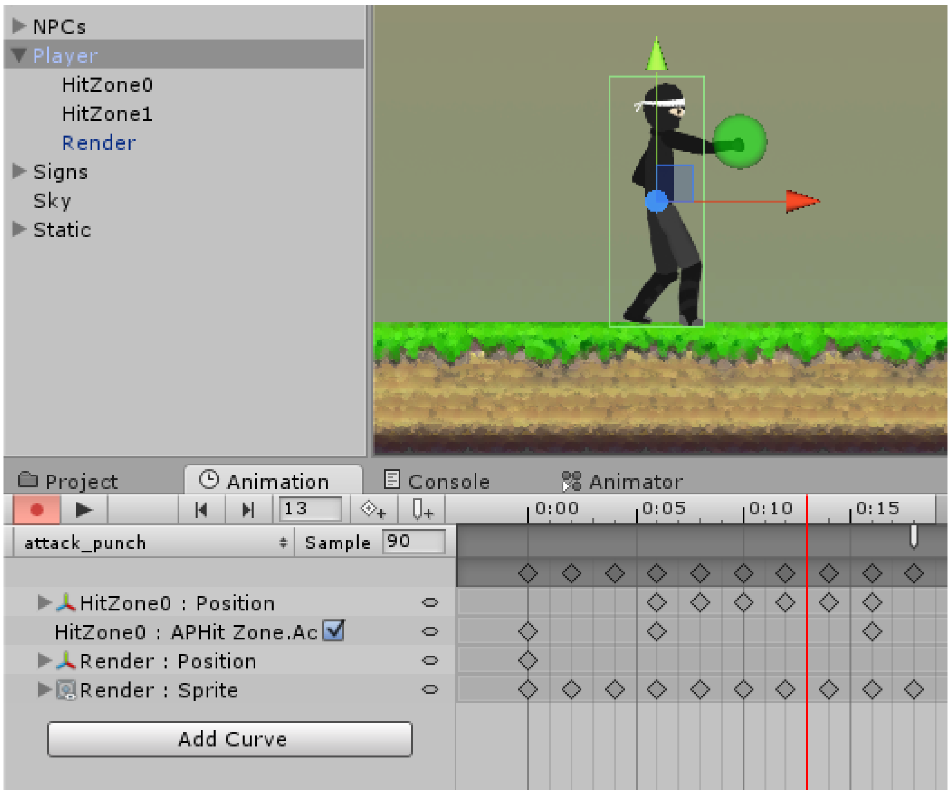
Melee Attack

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.



Melee Attacks	
Enabled	✓
Attacks	
Size	1
Attack0	
Button	Attack0
Anim Stand	attack_punch
Anim Crouched	attack_crouch
Anim In Air	attack_inair_kick
Hit Zones	
Size	2
Element 0	HitZone0 (APHitZone)
Element 1	HitZone1 (APHitZone)

- **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.

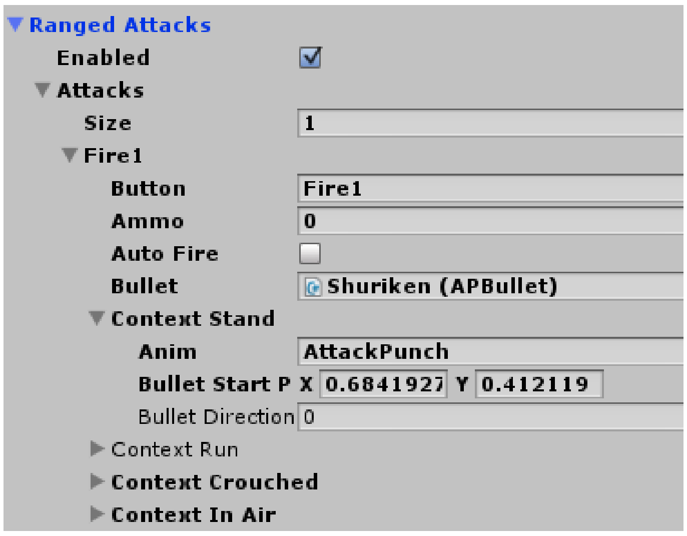
Then under your melee attack configuration, add an element to the array of Hit Zones and point your newly created HitZone game object.
Notice that any hit zone not listed in this array will be ignored.
You can animate your hit zone as you wish in your melee attack animation (local position, radius, active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

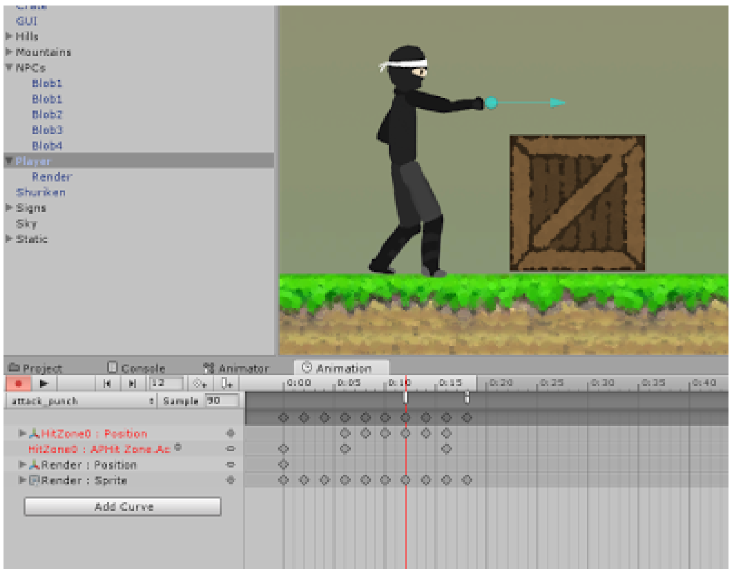
Ranged attack is similar to Melee Attack mechanism.
You can add as ranged attack as wanted.
A ranged attack is defined by an input name, a list of animations and settings for each context (stand, inair, crouch...) + some common settings.
Each attack launch bullets at desired starting position and in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



▼ Ranged Attacks	
Enabled	✓
▼ Attacks	
Size	1
▼ Fire1	
Button	Fire1
Ammo	0
Auto Fire	☐
Bullet	Shuriken (APBullet)
▼ Context Stand	
Anim	AttackPunch
Bullet Start P	X 0.6841927 Y 0.412119
Bullet Direction	0
▶ Context Run	
▶ Context Crouched	
▶ Context In Air	

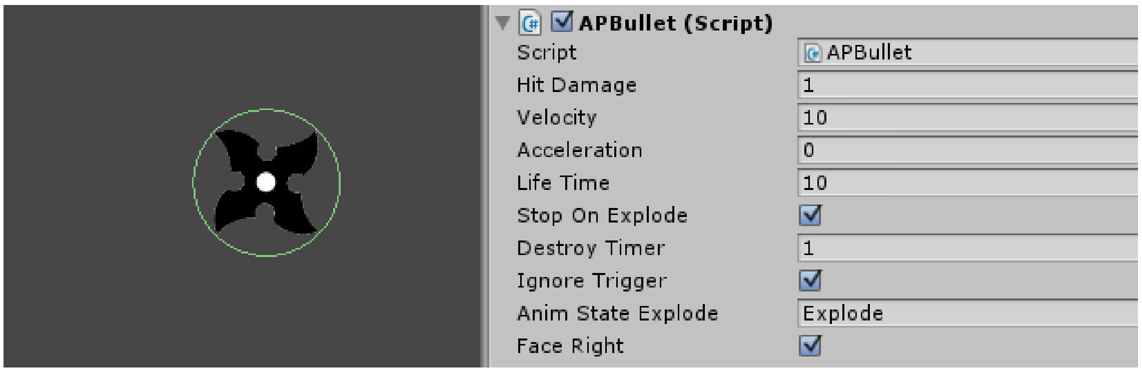
- **Enabled** : enabled status (can be updated in script or per game object)
- **Size**: number of ranged attacks
- For each attack
  - **Button**: name of input for launching this attack (disabled if null)
  - **Ammo**: number of remaining ammo
  - **Auto Fire**: enable auto fire (player can hold button 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 automattically 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 complexe bullet dynamics later.



APBullet (Script)	
Script	APBullet
Hit Damage	1
Velocity	10
Acceleration	0
Life Time	10
Stop On Explode	✓
Destroy Timer	1
Ignore Trigger	✓
Anim State Explode	Explode
Face Right	✓

- **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.

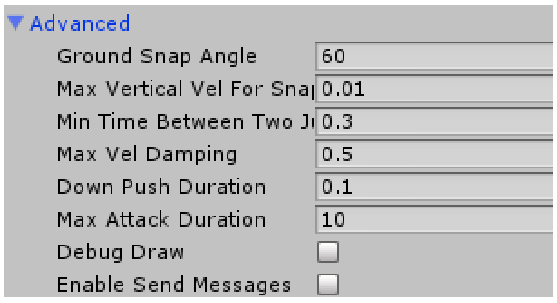
Events

Some scenaric events are launched by the script. This is done by calling the SendMessage API onto the game object owning the script.
Finally if you want to be notified of a particular scenaric event, you just have to add a script onto the player game object and adding the corresponding method.
Make sure that "Enable Send Messages" is checked in the character advanced settings otherwise no event is launched.

- APOnJump => *when player jumps*
- APOnLand => *when landing on any ground*
- APOnInAir => *when loosing ground*
- APOnWallJump => *when doing wall jump*
- APOnCrouch / APOnUncrouch => *when crouching/uncrouching*
- APOnMeleeAttack => *when launching melee attack*
- APOnRangedAttack => *when launching ranged attack*

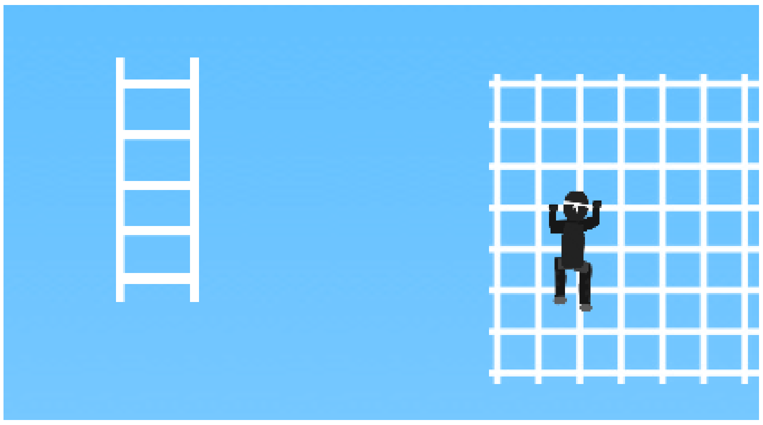
Advanced Settings

Controller has some more advanced settings that are contained here.
Normally default settings should fit all needs, but this may be tuned in some specific cases.
Only advanced users should set these values. Ask for support if you need some help.



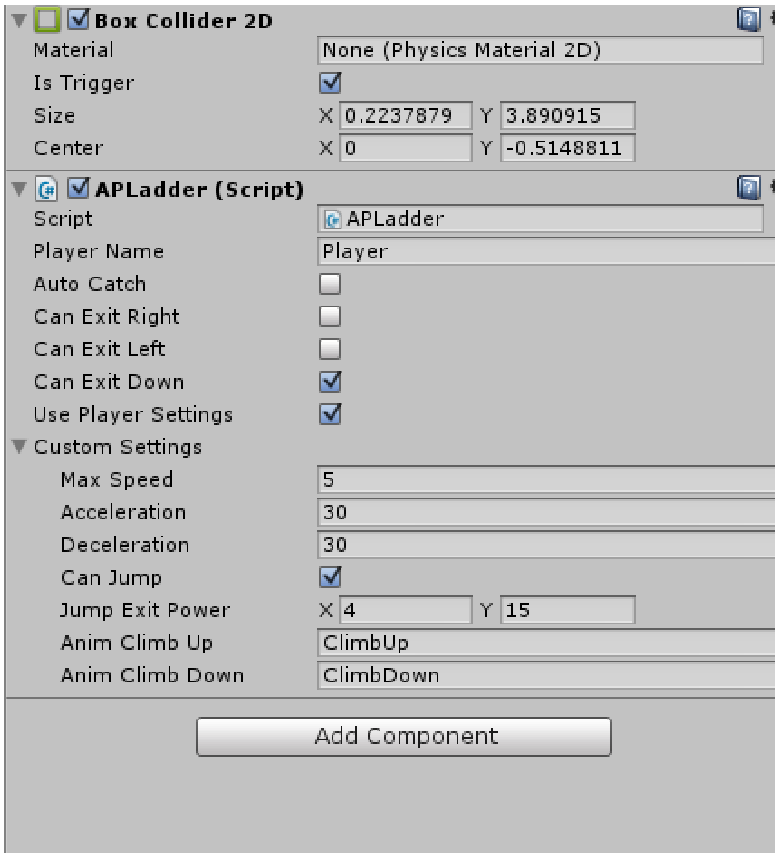
Advanced	
Ground Snap Angle	60
Max Vertical Vel For Snaı	0.01
Min Time Between Two Jı	0.3
Max Vel Damping	0.5
Down Push Duration	0.1
Max Attack Duration	10
Debug Draw	☐
Enable Send Messages	☐

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.

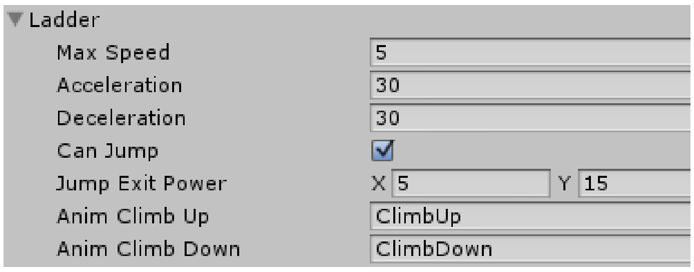
Same rules apply to railings except that player can move freely on horizontal axis too.
More over you can add many BoxColliders to handle complex railings (see Basics demo level).



Box Collider 2D	✓
Material	None (Physics Material 2D)
Is Trigger	✓
Size	X 0.2237879 Y 3.890915
Center	X 0 Y -0.5148811
APLadder (Script)	✓
Script	APLadder
Player Name	Player
Auto Catch	☐
Can Exit Right	☐
Can Exit Left	☐
Can Exit Down	✓
Use Player Settings	✓
Custom Settings	
Max Speed	5
Acceleration	30
Deceleration	30
Can Jump	✓
Jump Exit Power	X 4 Y 15
Anim Climb Up	ClimbUp
Anim Climb Down	ClimbDown

- **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.



▼ Ladder	
Max Speed	5
Acceleration	30
Deceleration	30
Can Jump	✓
Jump Exit Power	X 5 Y 15
Anim Climb Up	ClimbUp
Anim Climb Down	ClimbDown

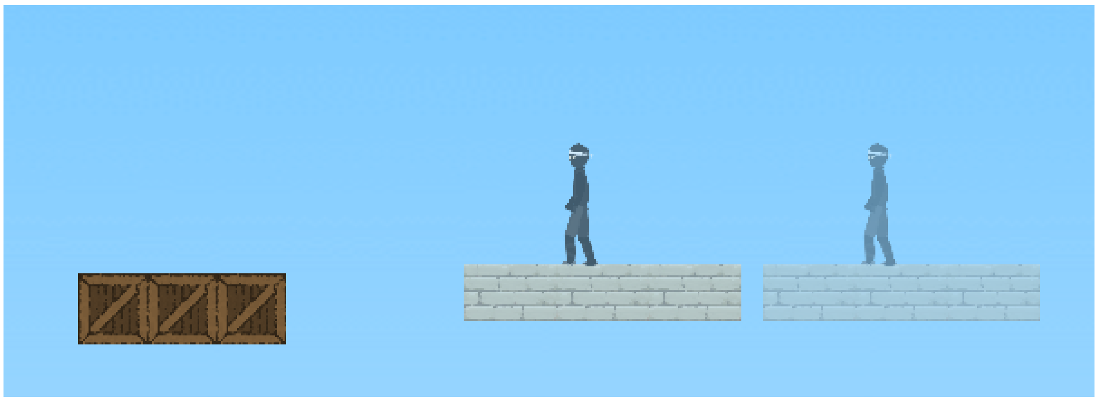
- **Max Speed** : maximum speed (m/s)
- **Acceleration** : acceleration (m/s²)
- **Deceleration** : deceleration (m/s²)
- **Can Jump** : allows jumping in facing direction while catched
- **Jump Exit Power** : jump power
- **Anim Climb XXX** : animation to use for moving in XXX direction

Events

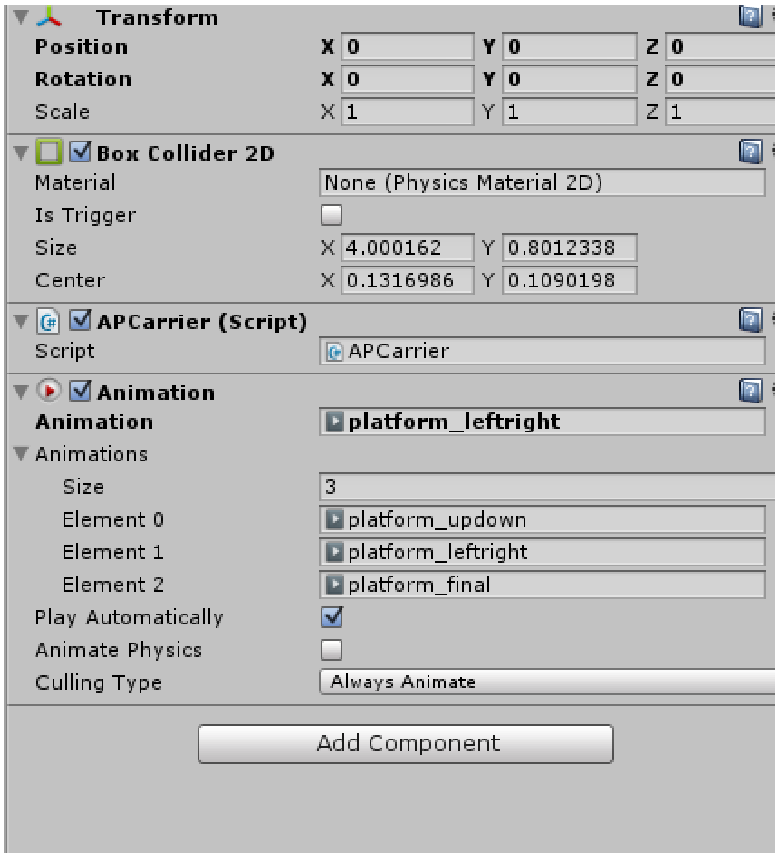
Some scenaric events are launched by a ladder/grid. This is done by calling the SendMessage API onto the game object owning the script.
Finally if you want to be notified of a particular scenaric event, you just have to add a script onto the player game object and adding the corresponding method.
Make sure that "Enable Send Messages" is checked in the character advanced settings otherwise no event is launched.

- APOnCatchLadder
- APOnReleaseLadder
- APOnCatchRailings
- APOnReleaseRailings

Carrier



When lying on a moving object, your character may not be carried.
To enabled this, you just have to attach a Carrier script on it.
Then handle moving of your object as your will (by animation or by script directly).



Transform

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

Box Collider 2D

Material	None (Physics Material 2D)	
Is Trigger	☐	
Size	X 4.000162	Y 0.8012338
Center	X 0.1316986	Y 0.1090198

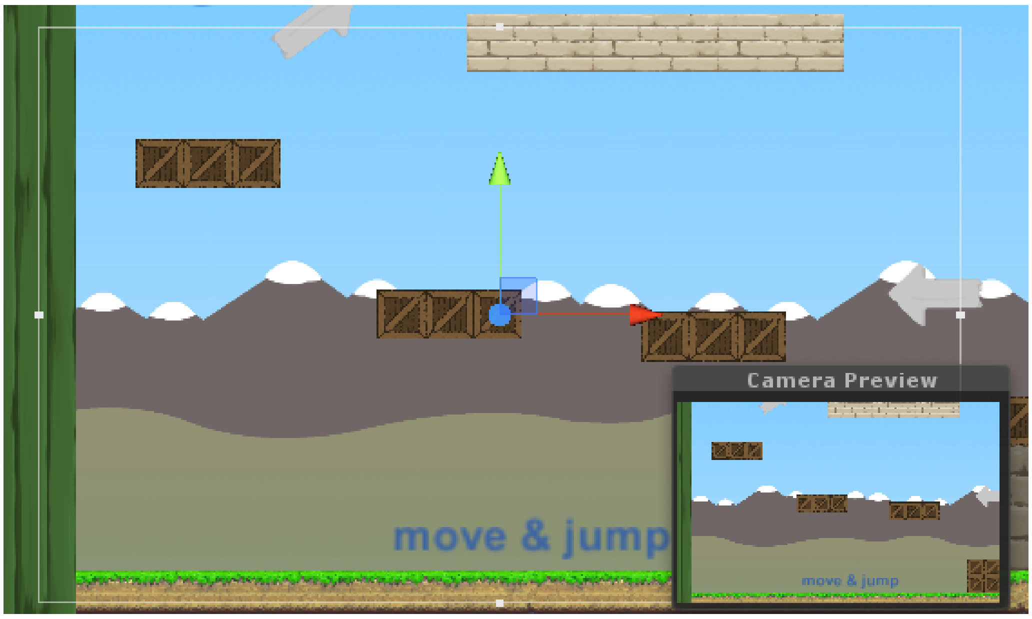
APCarrier (Script)

Script	APCarrier

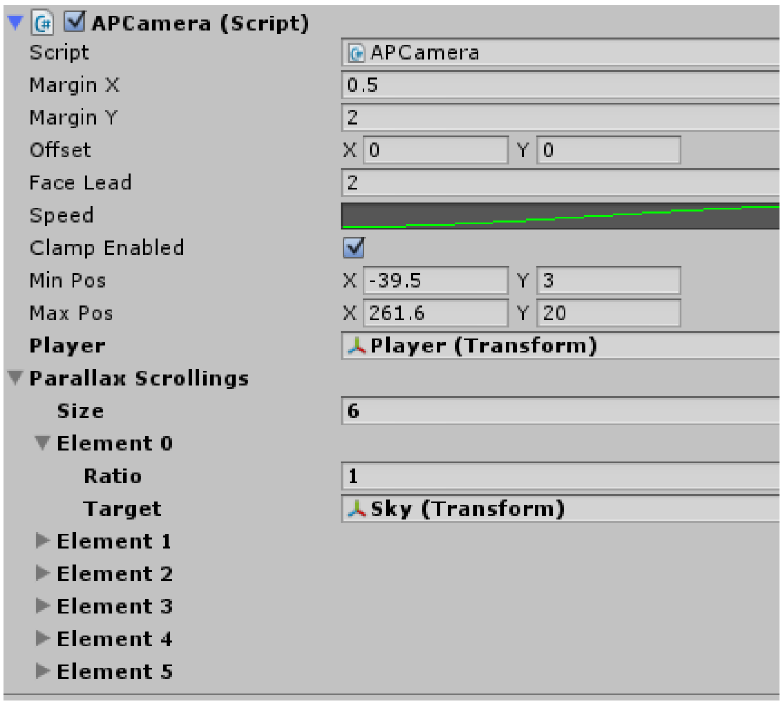
Animation

Animation	platform_leftright
Animations	
Size	3
Element 0	platform_updown
Element 1	platform_leftright
Element 2	platform_final
Play Automatically	✓
Animate Physics	☐
Culling Type	Always Animate

Camera



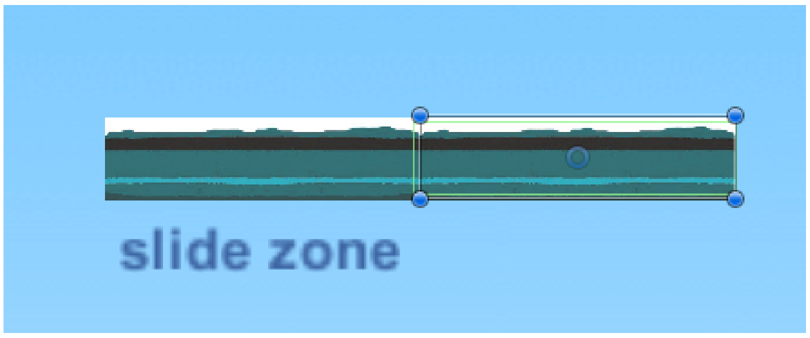
Provided camera should fits most need of any 2D platformer.
This fits standard 'Mario' style camera. Please have a look here for concise explanation : the-ideal-platformer-camera-should-minimize
Attach it to your game camera and select target transform to use.



APCamera (Script)	
Script	APCamera
Margin X	0.5
Margin Y	2
Offset	X 0 Y 0
Face Lead	2
Speed	
Clamp Enabled	✓
Min Pos	X -39.5 Y 3
Max Pos	X 261.6 Y 20
Player	Player (Transform)
Parallax Scrollings	
Size	6
Element 0	
Ratio	1
Target	Sky (Transform)
Element 1	
Element 2	
Element 3	
Element 4	
Element 5	

- **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

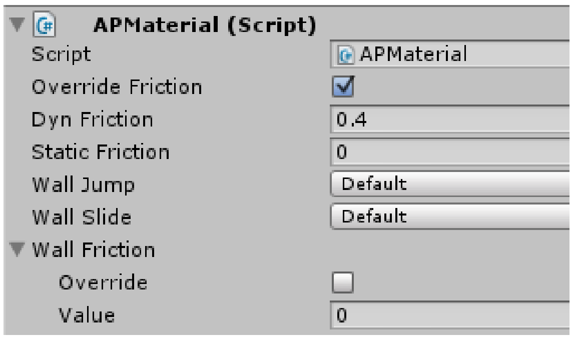
Material



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.

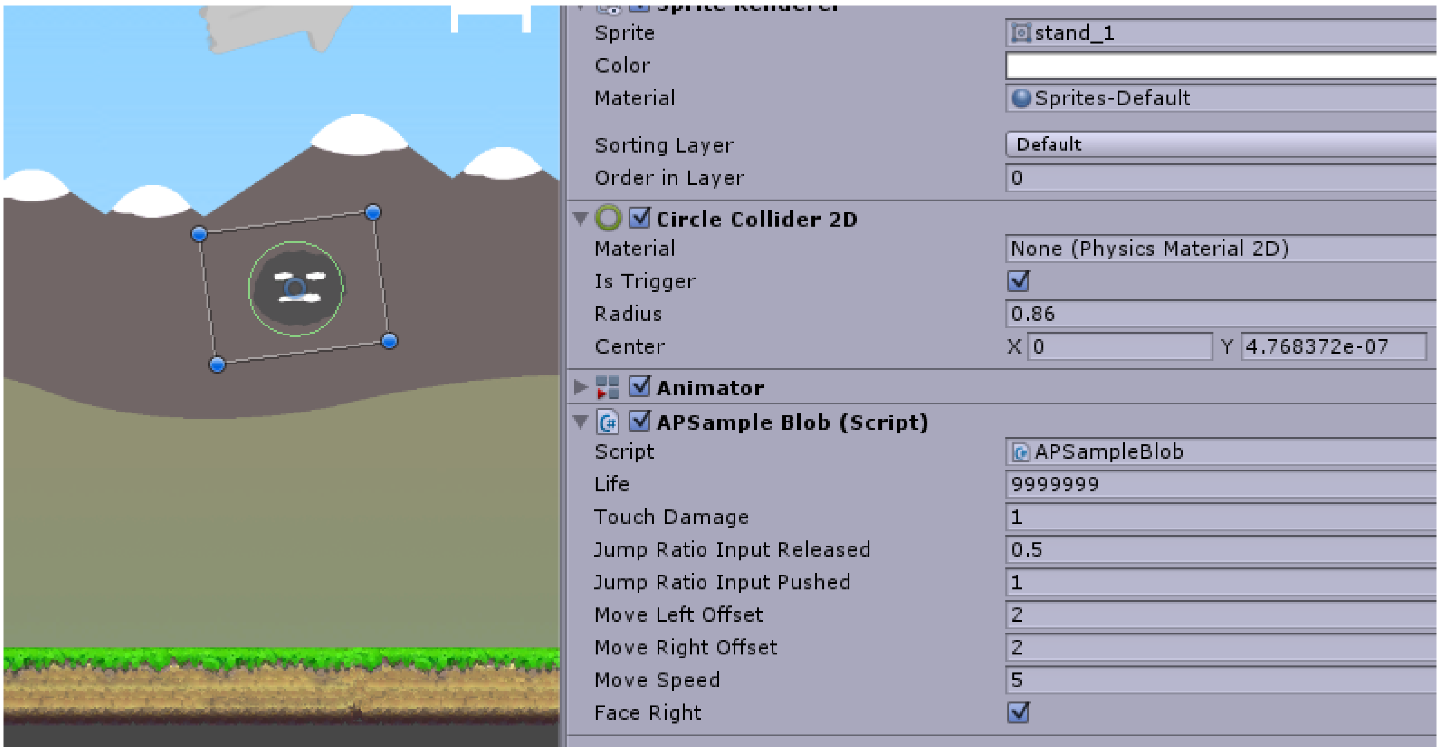
A perfect example is the friction. By default you want no friction for your character, so you put 1 in its default setting value.
But if you want a specific ground to slide, you just have to put a new script on it, add Material and override friction value.
This new friction will be used if the character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


```
APMaterial (Script)
Script                APMaterial
Override Friction     ✓
Dyn Friction          0.4
Static Friction       0
Wall Jump             Default
Wall Slide            Default
Wall Friction
    Override          ☐
    Value             0
```

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

Hitable


```
Sprite                       stand_1
Color
Material                     Sprites-Default

Sorting Layer                Default
Order in Layer               0
Circle Collider 2D
Material                     None (Physics Material 2D)
Is Trigger                   ✓
Radius                       0.86
Center                       X 0      Y 4.768372e-07
Animator
APSample Blob (Script)
Script                       APSampleBlob
Life                         9999999
Touch Damage                 1
Jump Ratio Input Released    0.5
Jump Ratio Input Pushed      1
Move Left Offset             2
Move Right Offset            2
Move Speed                   5
Face Right                   ✓
```

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 the asset tries 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.
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...)

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 !