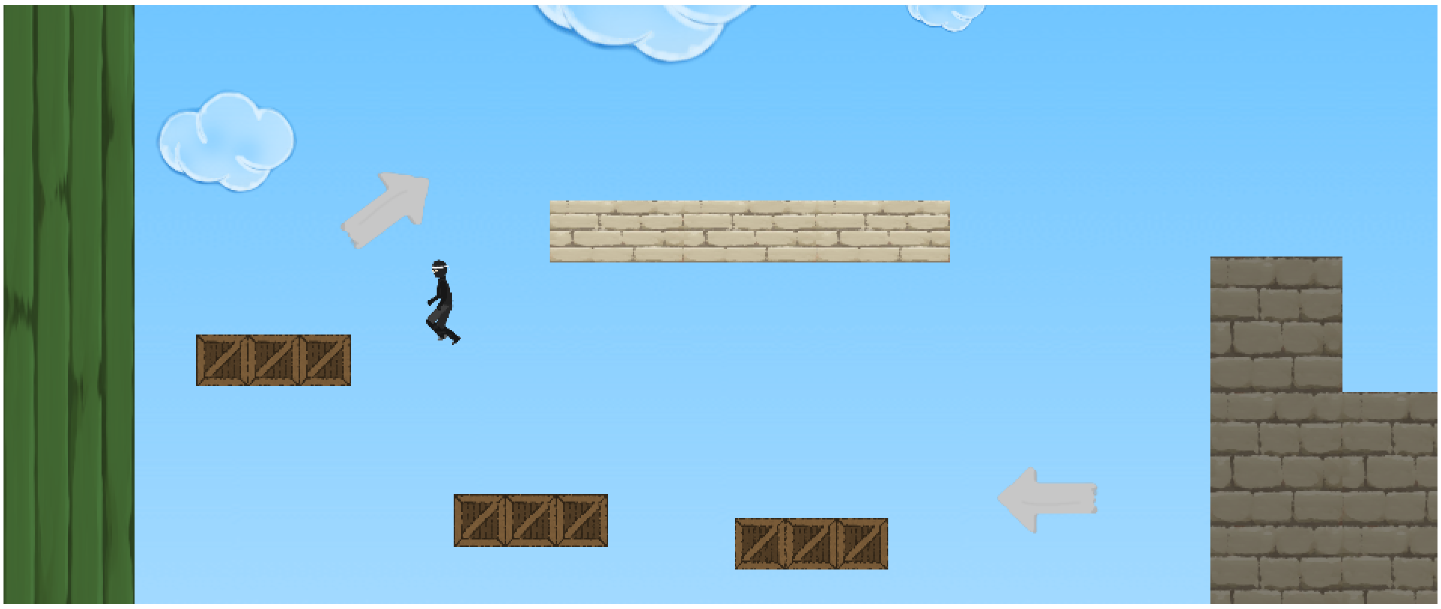
Advanced Platformer 2D

Documentation v1.1



Advanced Platformer 2D is a toolkit package for Unity dedicated to **2D platformer** video games.
It is built upon new **Unity 2D** toolkit included from version **4.3** (physic, sprites, animations....).
It is here to help you in creating your own 2D platformer upon Unity game engine.
You can reuse every provided elements and modify them to get 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 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
- 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

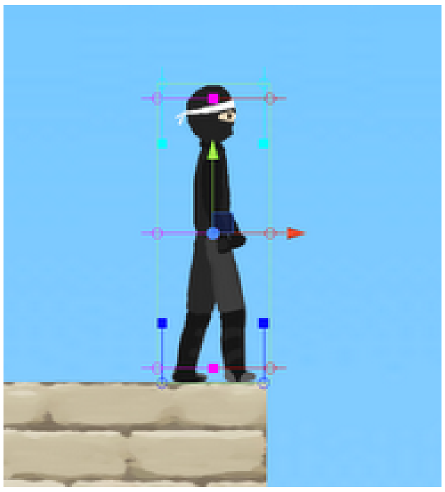
All values are expressed in meters, kilograms and seconds (position, velocity, mass...).
This is the convention of most physic engines, that's why we respects this.
You will have to scale these values to fit your needs if needed.
All visible settings are serialized, undoable and prefabizable.

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.

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.

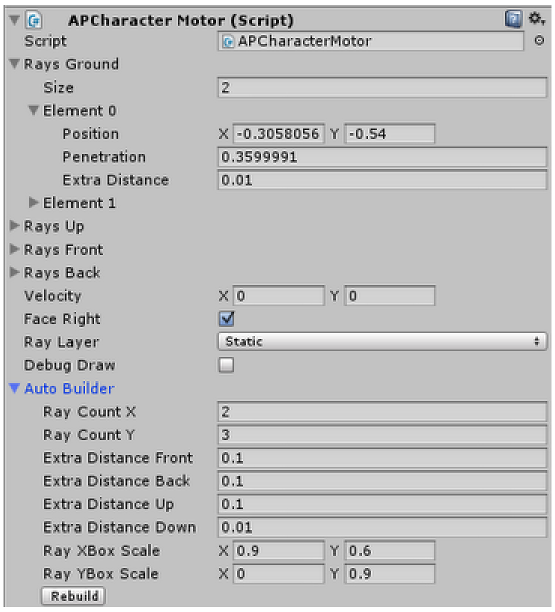


A ray has a starting position (represented as a square) and a ending position.
The small sphere is used to check the intersection with the box collider, indeed each ray will try to prevent penetration inside the collider.
Remaining path, after the sphere, is called **extra distance**, this is only for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

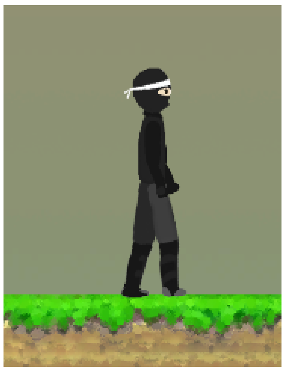


```
APCharacter Motor (Script)
Script                  APCharacterMotor
Rays Ground
  Size                  2
  Element 0
    Position            X -0.3058056  Y -0.54
    Penetration         0.3599991
    Extra Distance      0.01
  Element 1
Rays Up
Rays Front
Rays Back
Velocity                X 0  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
Rebuild
```

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
- **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

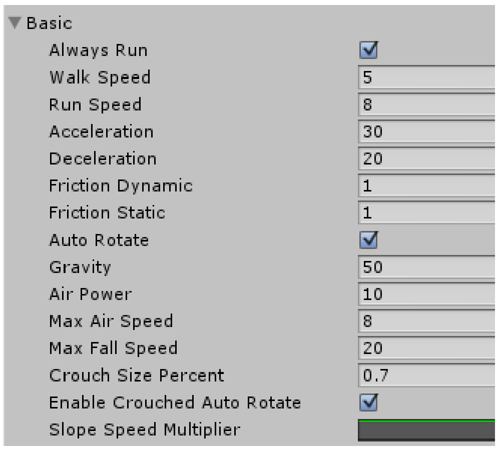
CharacterController



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



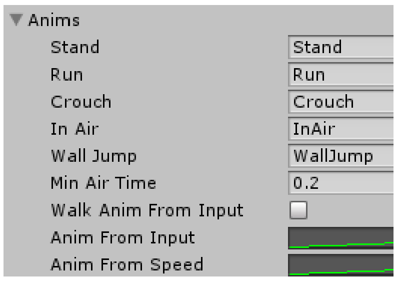
Basic

Always Run	✓
Walk Speed	5
Run Speed	8
Acceleration	30
Deceleration	20
Friction Dynamic	1
Friction Static	1
Auto Rotate	✓
Gravity	50
Air Power	10
Max Air Speed	8
Max Fall Speed	20
Crouch Size Percent	0.7
Enable Crouched Auto Rotate	✓
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)
- **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 force while in air (m/s²)
- **Air Power** => input force while in air (m/s²)
- **Max Air Speed** => maximum horizontal speed while in air (m/s)
- **Max Fall Speed** => maximum downfall speed (m/s)
- **Slope Speed Multiplier** => curve to scale speed in function of slope angle (in degree)
- **Crouch Size Percent** => size scale when crouched
- **Enable Crouch Auto Rotate** => enable/disable player flipping while crouched

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.



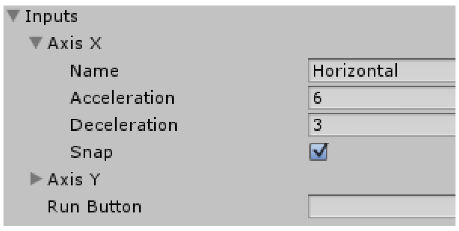
Anims

Stand	Stand
Run	Run
Crouch	Crouch
In Air	InAir
Wall Jump	WallJump
Min Air Time	0.2
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** : speed of walk/run animation is always computed from filtered input, otherwise from ground speed
- **Anim From Input** : animation speed in function of input, this also used when walking on surface with friction < 1 even if "**Walk Anim From Input**" is disabled
- **Anim From Speed** : animation speed in function of ground speed

Inputs

Inputs filtering is done by a specific script, this replace Unity input filtering mechanism but use same principles.
An input have a raw value, i.e value at which input is physically. This is in range [-1, 1].
You can filter this value, for example to prevent sudden changes, this is often used for keyboards as keys don't have analogical state.
Filtered value is used only when computing animation speed from input (see **Anim From Input** in Anims).
Otherwise we always use raw value.



Inputs
Axis X

Name	Horizontal
Acceleration	6
Deceleration	3
Snap	✓

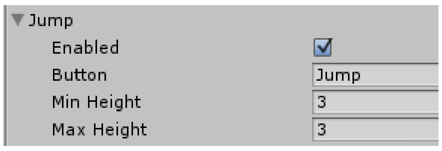
Axis Y

Run Button	

- **Axis X/Y** : used for horizontal/vertical moves
  - Name : name of Unity axis input to use
  - Acceleration : max speed at which input can accelerate
  - Deceleration : max speed at which input can decelerate
  - Snap : immediate switch from one sign to the other.
- **Run Button** : button to use if Always Run is disabled

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.

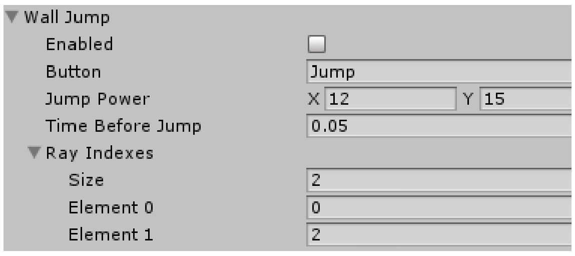


▼ Jump	
Enabled	✓
Button	Jump
Min Height	3
Max Height	3

- **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

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.

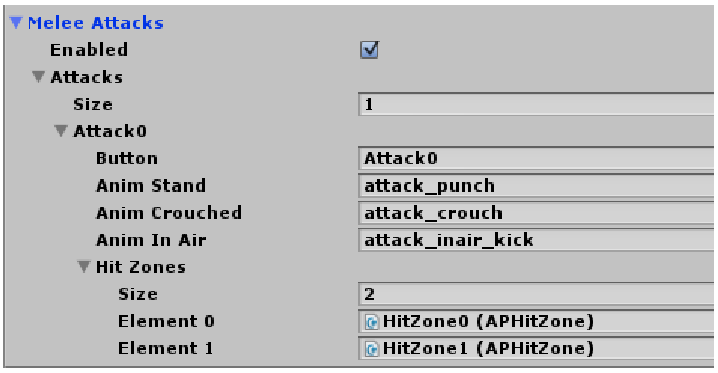


▼ Wall Jump	
Enabled	☐
Button	Jump
Jump Power	X 12 Y 15
Time Before Jump	0.05
▼ Ray Indexes	
Size	2
Element 0	0
Element 1	2

- **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

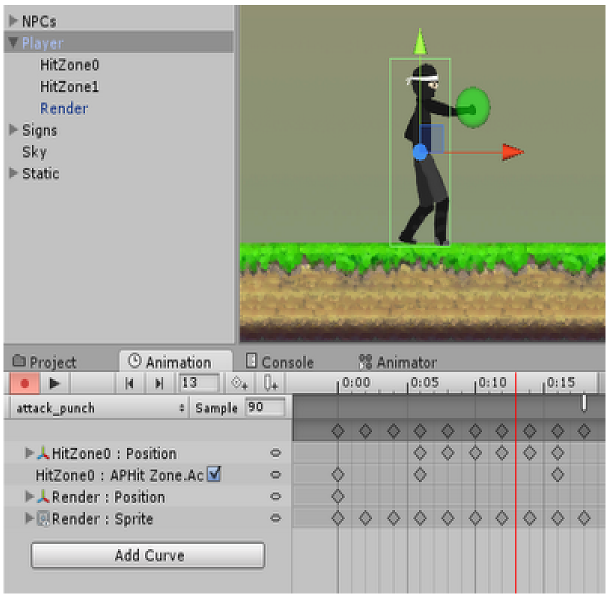
Melee Attack

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.

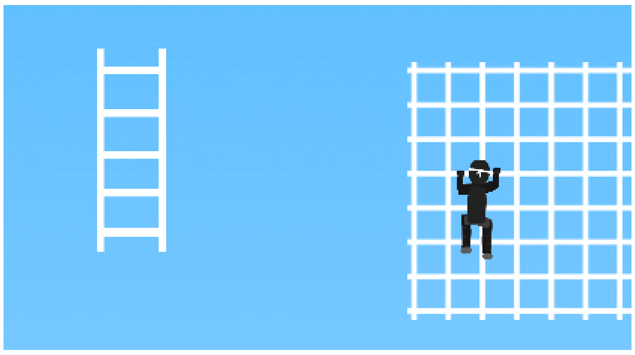
Finally your animation must tell the engine when the attack has ended, to achieve this you have to add a script event at the last frame.
This event must point to the **LeaveMeleeAttack** API. If not doing so, there is a security test in engine not allowing more than 10 seconds for playing the attack.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

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 :

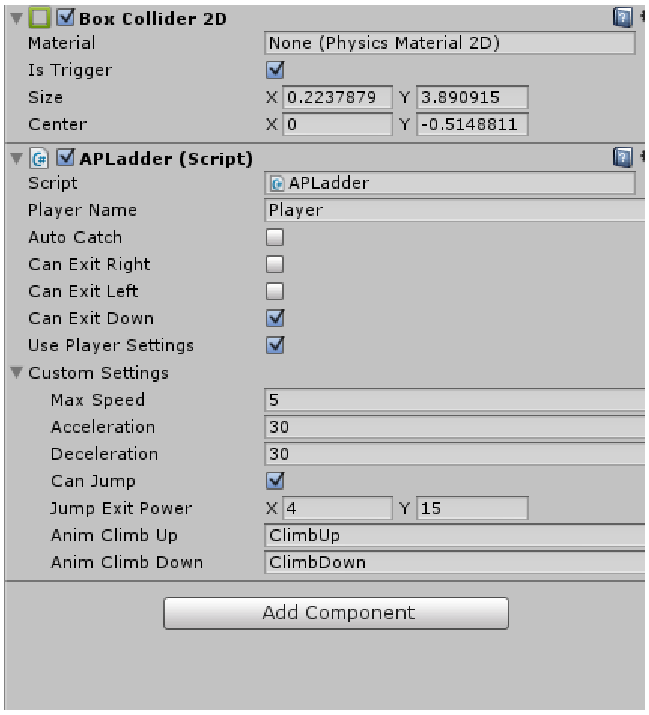
- APOnJump => *when player jumps*
- APOnLand => *when landing on any ground*
- APOnInAir => *when loosing ground*
- APOnWallJump => *when doing wall jump*
- APOnCrouch / APOnUncrouch => *when crouching/uncrouching*
- APOnMeleeAttack => *when launching melee attack*

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

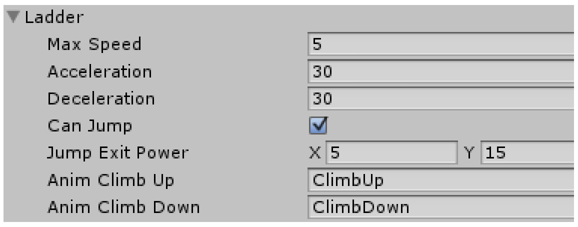
Material	None (Physics Material 2D)
Is Trigger	✓
Size	X 0.2237879 Y 3.890915
Center	X 0 Y -0.5148811

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

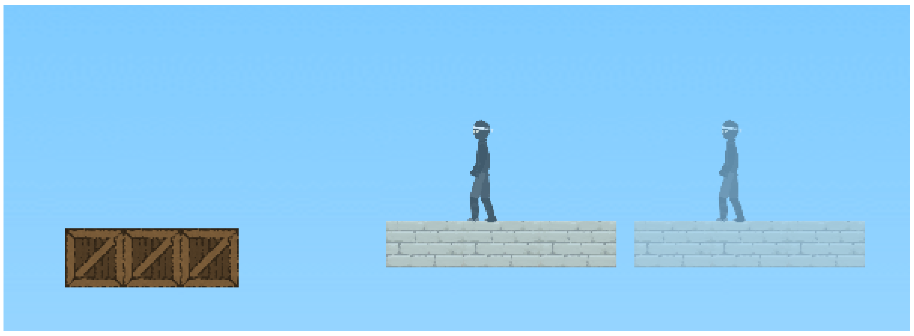
- **Max Speed** : maximum speed (m/s)
- **Acceleration** : acceleration (m/s²)
- **Deceleration** : deceleration (m/s²)
- **Can Jump** : allows jumping in facing direction while catched
- **Jump Exit Power** : jump power
- **Anim Climb XXX** : animation to use for moving in XXX direction

Events

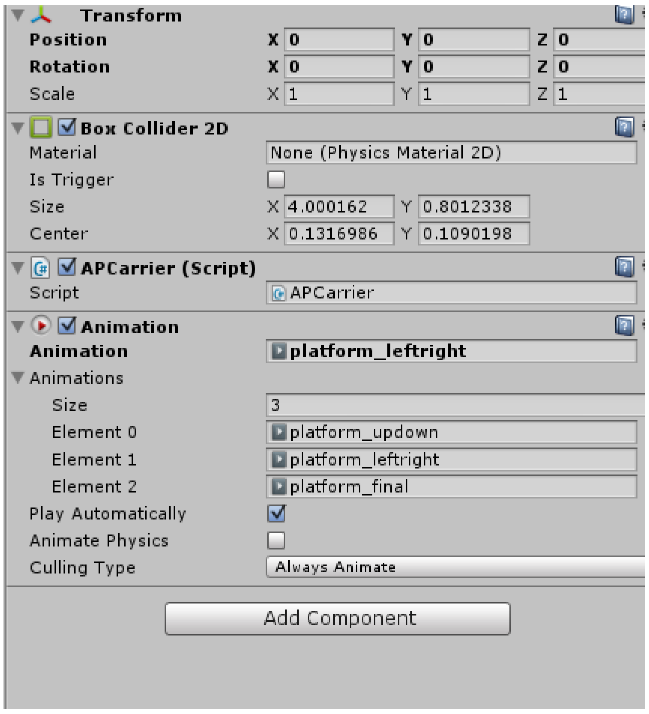
Some scenaric events are launched by a ladder/grid. This is done by calling the SendMessage API onto the game object owning the script.
Finally if you want to be notified of a particular scenaric event, you just have to add a script onto the player game object and adding the corresponding method :

- APOnCatchLadder
- APOnReleaseLadder
- APOnCatchRailings
- APOnReleaseRailings

Moving Platform



Moving platforms are used to carry properly the character when moving.
You just have to attach an Carrier script on it. Finally move it with standard animation.
Do not check "Animate Physics", this is not supported for now.



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

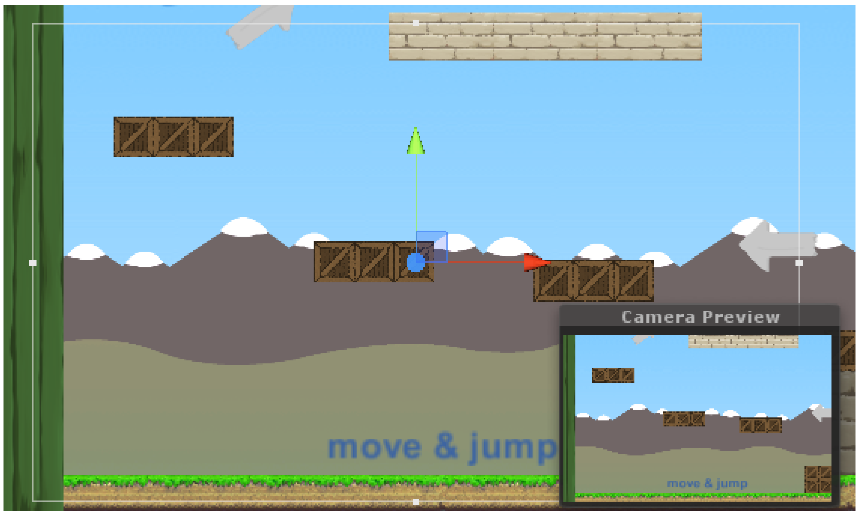
APCarrier (Script)

Script	APCarrier

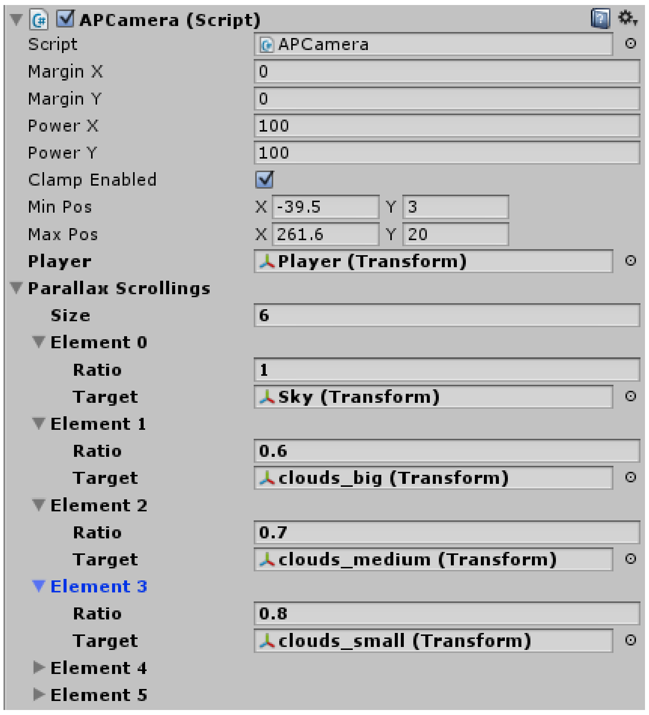
Animation

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



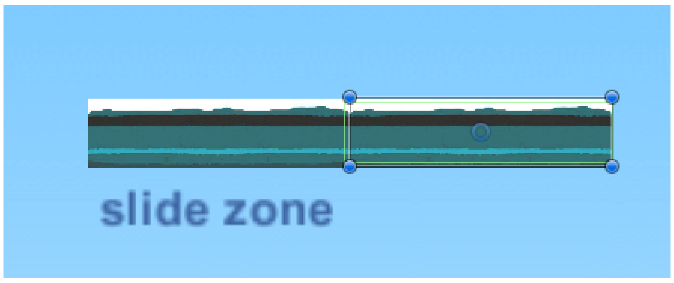
We offer a simple camera follow script.
Attach it to your camera and select target transform to use.
Some basic features are available.



```
APCamera (Script)
Script                APCamera
Margin X              0
Margin Y              0
Power X               100
Power Y               100
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
    Ratio             0.6
    Target            clouds_big (Transform)
  Element 2
    Ratio             0.7
    Target            clouds_medium (Transform)
  Element 3
    Ratio             0.8
    Target            clouds_small (Transform)
  Element 4
  Element 5
```

- **Margin X/Y** : allowed inside margin for player
- **Power X/Y** : camera power to follow player
- **Clamp Enabled** : enable camera position clamp
- **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

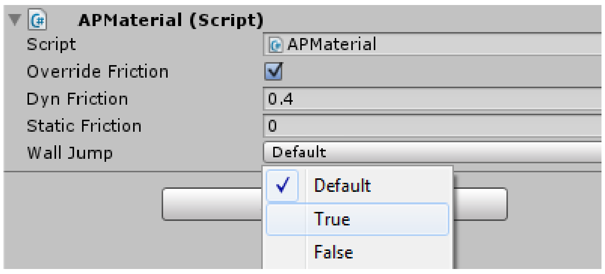
Material



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.

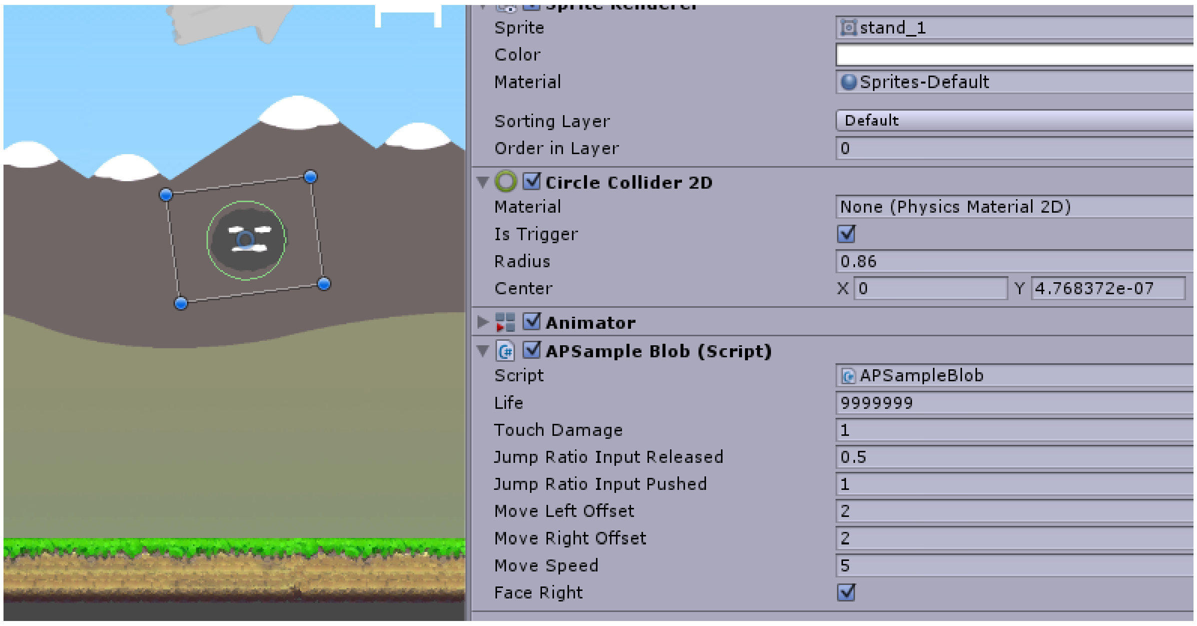
Some others settings can be overridden for example if player can do wall jump or not.
For example by default you don't allow wall jumping on your character except form some walls, this is done in demo level.
We have attached a material script on wall and override Wall Jump default value to True.



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

- **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.

```
public class APSampleBlob : APHitable
{
         // called when we have been hit by a melee attack
 override public void OnMeleeAttackHit(APCharacterController character, APHitZone hitZone)
 {
 ...
 }

 // called when character motor ray is touching us
 override public void OnCharacterTouch(APCharacterController character, APCharacterMotor.RayType rayType)
 {
 ...
 }
}
```

Please check the script APSampleBlob for an sample of use.
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.