Software Studio
軟體設計與實驗

Cocos Creator : Animation

Hung-Kuo Chu
Department of Computer Science
National Tsing Hua University

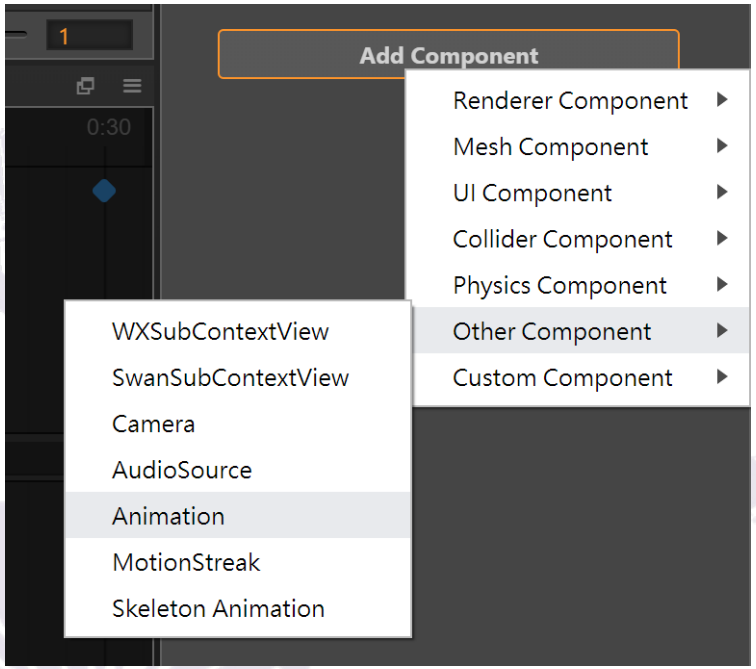
CS2410

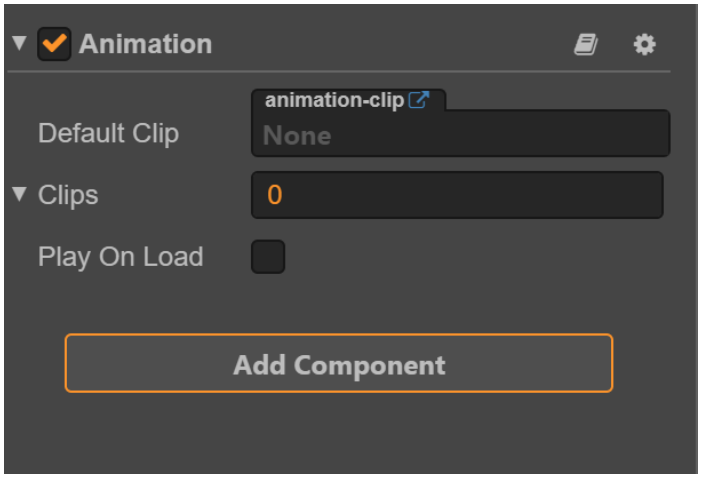
Animation

- Animation is a **component** in the Node.
- Animation includes **Animation Clips**, which are documents for saving animation data.
- Animation clips need to be mounted to Animation components to apply animation data to the Node.

Add an Animation Component

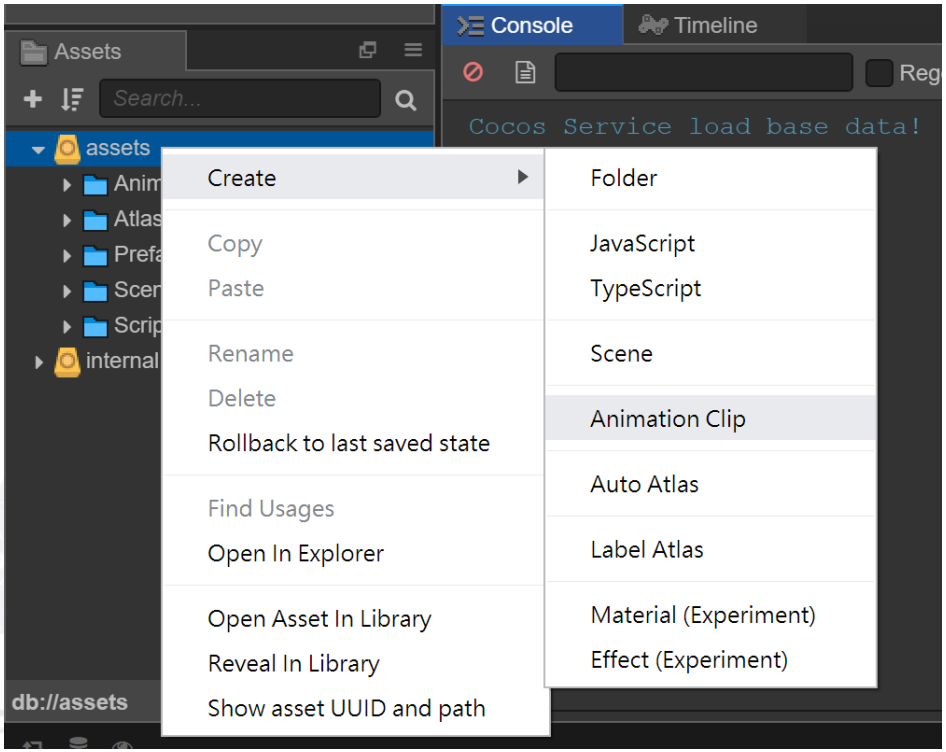
- Node → Add Component → Other Component → Animation





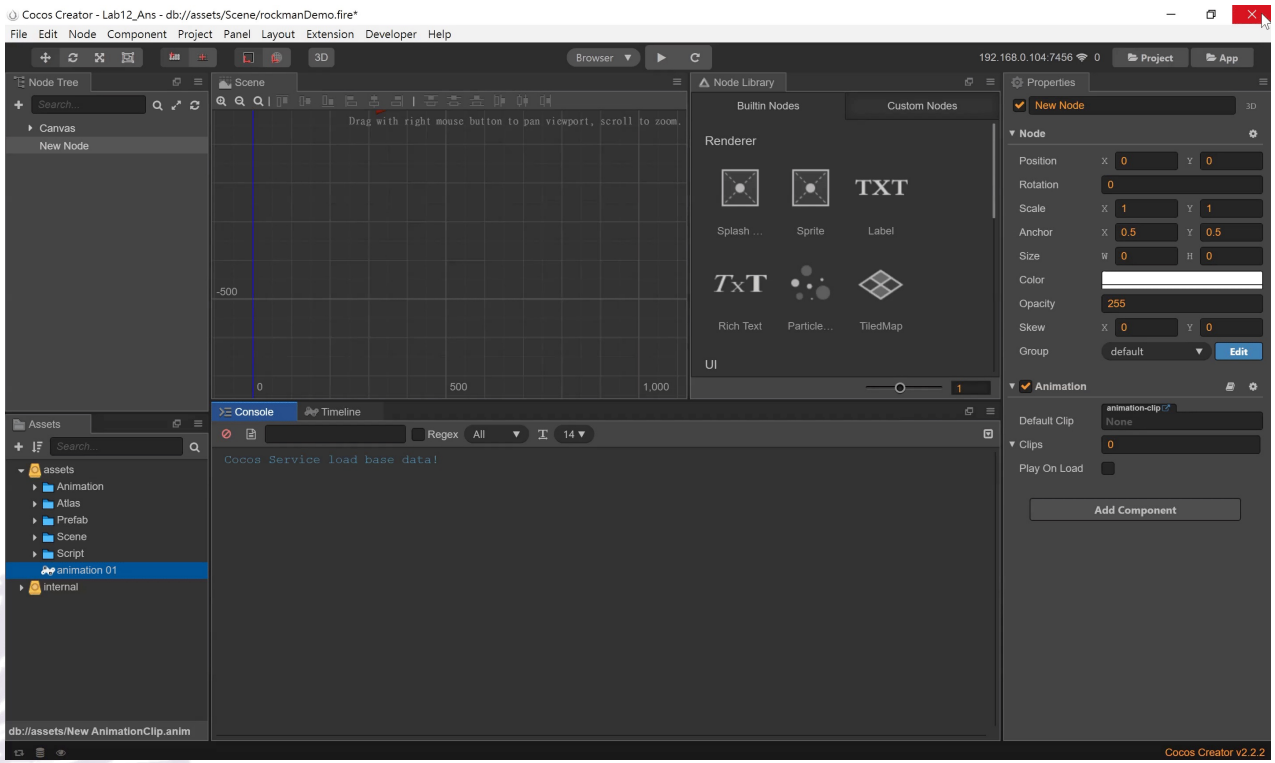
Add an Animation Clip

- assets → click right button → Create → Animation Clip



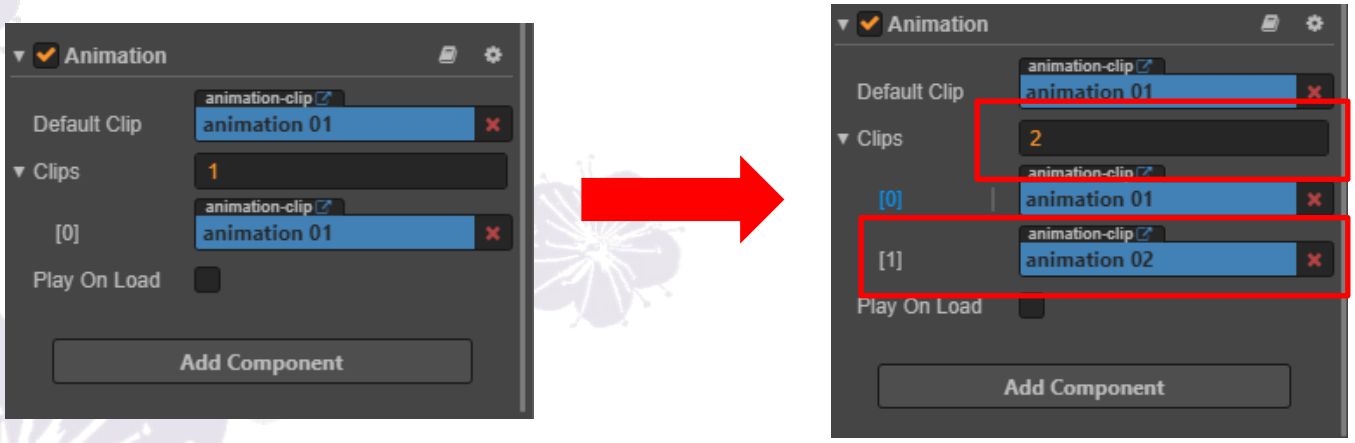
Mount an Animation Clip to Animation Component

- Drag the animation clip from asset to the “Default Clip” in an animation component.



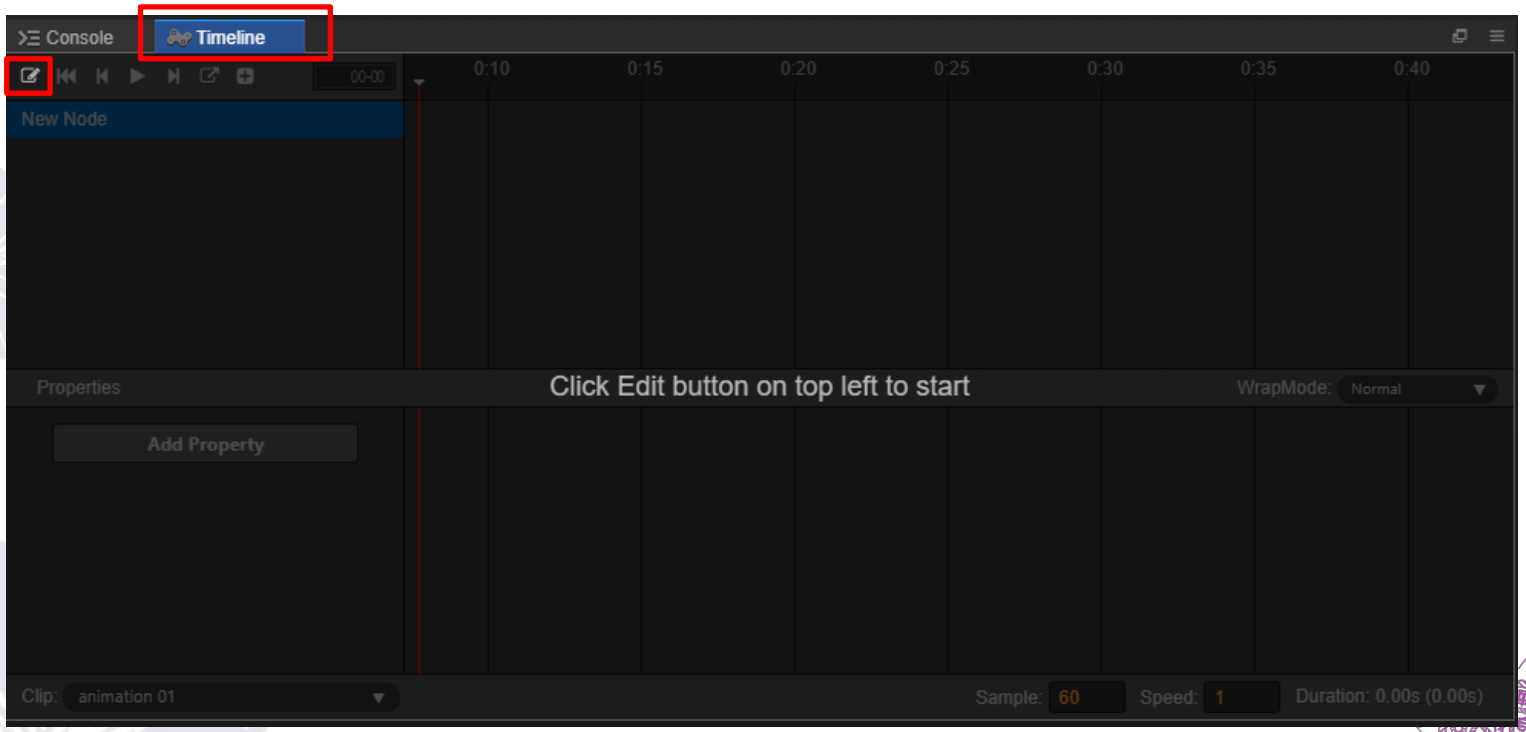
Mount Animation Clips to Animation Component

- If you want to mount more than one animation clip to the animation component, you can increase the number of **Clips** and mount additional animation clips.



Edit an Animation Clip

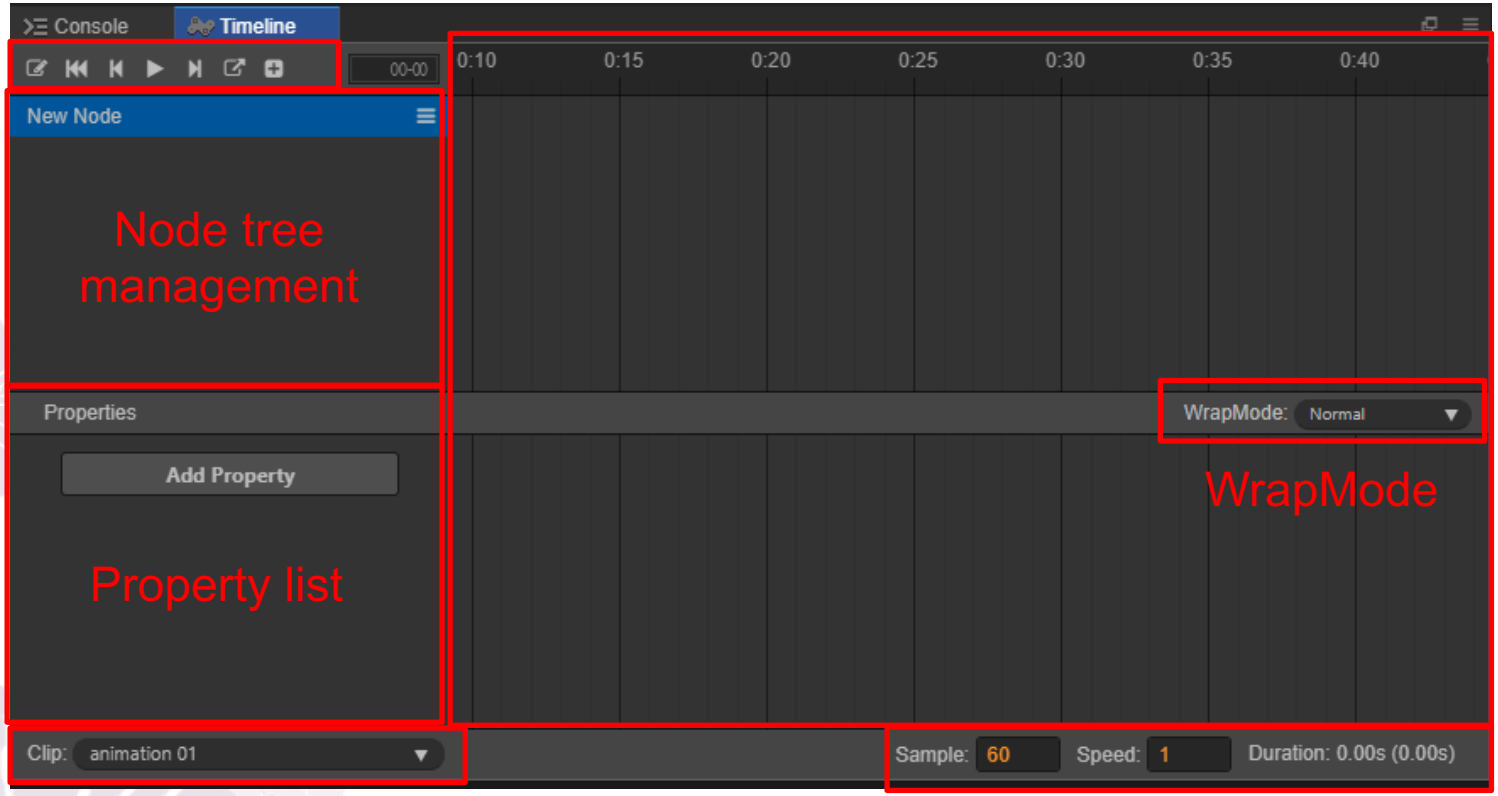
- Choose **Timeline** and click **edit** button to start editing of an animation clip.



Animation Editor: Layout

Common button field

Timeline & event



Current clip

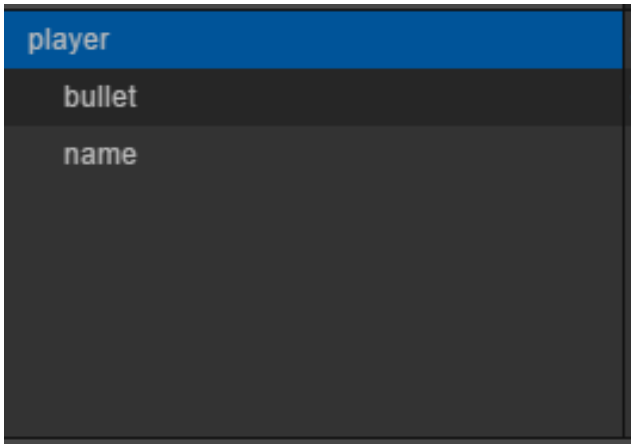
Clips params

Common Button Field

- Common button field contains common functional buttons, from left to right are:
  - **Recording status switch**
  - **Back to the first frame**
  - **Last frame**
  - **Play/stop**
  - **Next frame**
  - **Insert animation event**
  - **Create a new animation clip.**

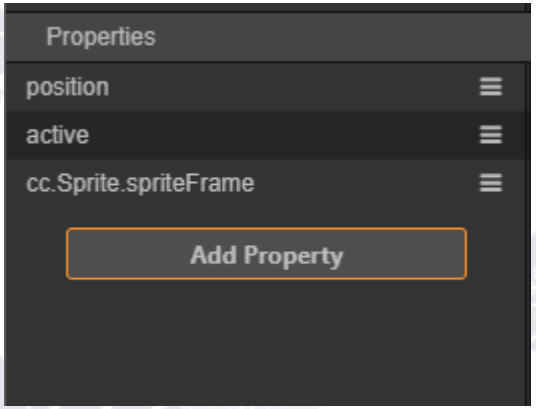
Node Tree Management

- Node tree management includes node data that is influenced by the current animation clip.
- Note that some parent node's animation properties **will affect all its child nodes.**



Property List

- Property list displays the animation related properties of the currently selected animation clip.

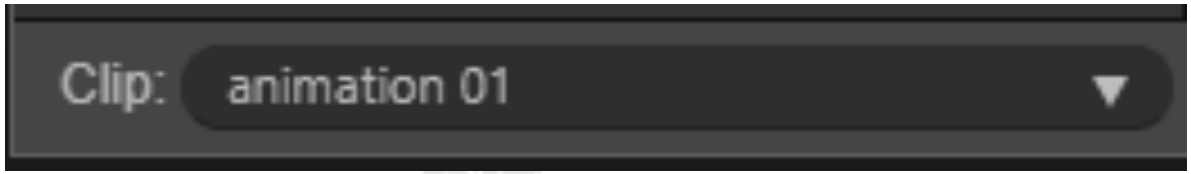


position
x
y
z
scale
scaleX
scaleY
scaleZ
angle
eulerAngles
width
height
color
opacity
anchorX
anchorY
skewX
skewY
cc.Sprite.spriteFrame
cc.Sprite.fillType
cc.Sprite.fillCenter
cc.Sprite.fillStart
cc.Sprite.fillRange

There are many types of properties you can use to enrich your animation!

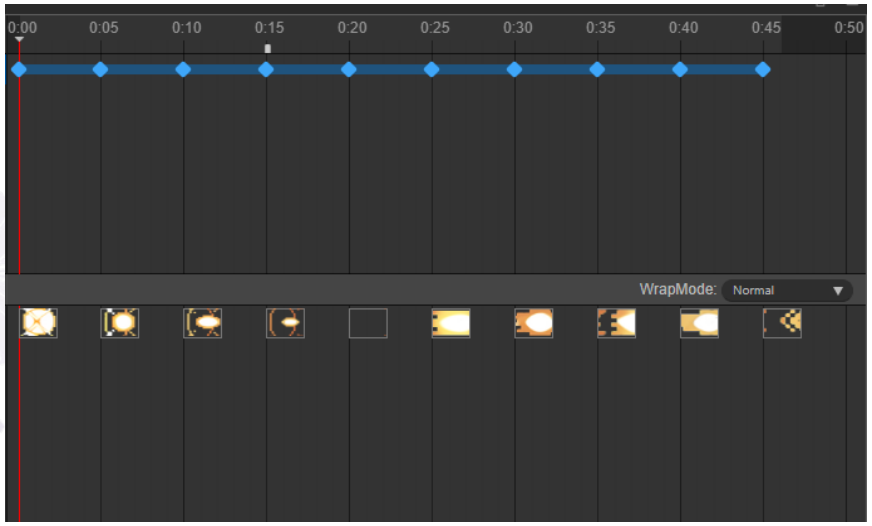
Current Clip

- If you have more than one animation clip mounted under the current animation component, you can switch to the clip you want to edit from here.



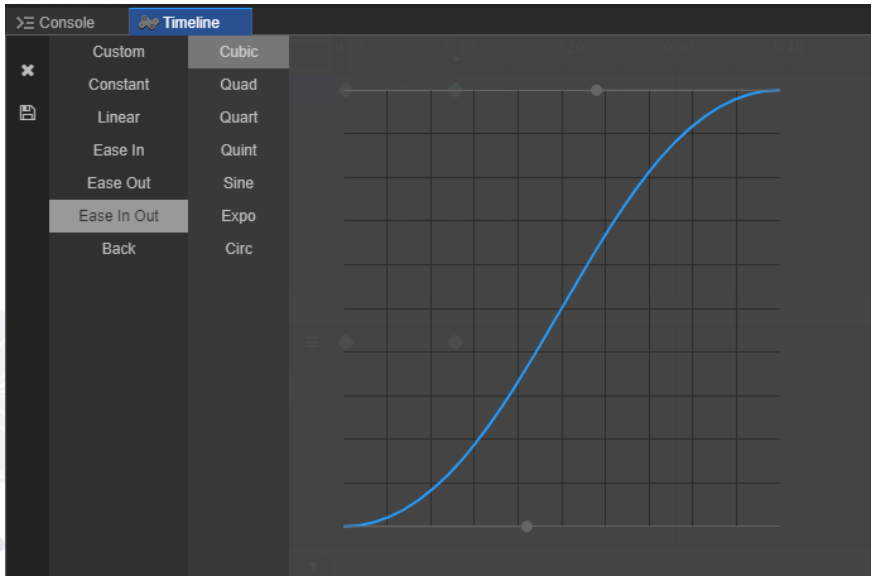
Timeline & Event

- **Timeline and event** mainly displays timeline and user-defined events.
- **Red line** represent the current frame, and every **blue nodes** are frames with user specified properties.



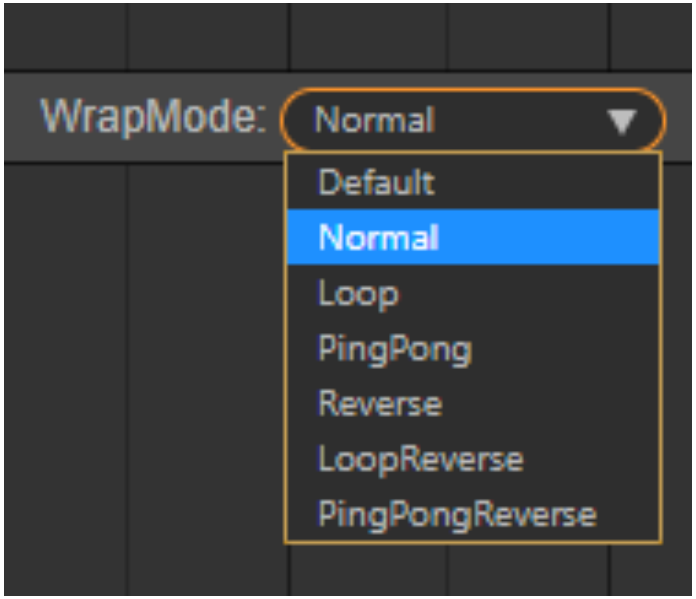
Edit Time Curve

- Double click the connecting line between two blue nodes to open the **time curve editor**.
- It is used to design the transition effect two nodes.



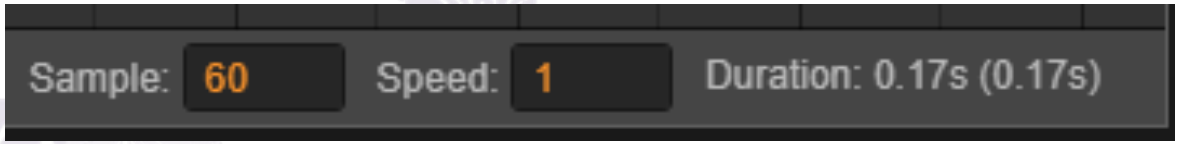
WrapMode

- **WrapMode** specifies the play mode of animation, such as **Normal**、**Reverse**、**Loop**…



Clip Params

- Clip params is a speed controller, you can modify the values of **sample** and **speed** to control the speed of animation.
- Note that the **duration** is the actual time the browser will take for playing the animation once.

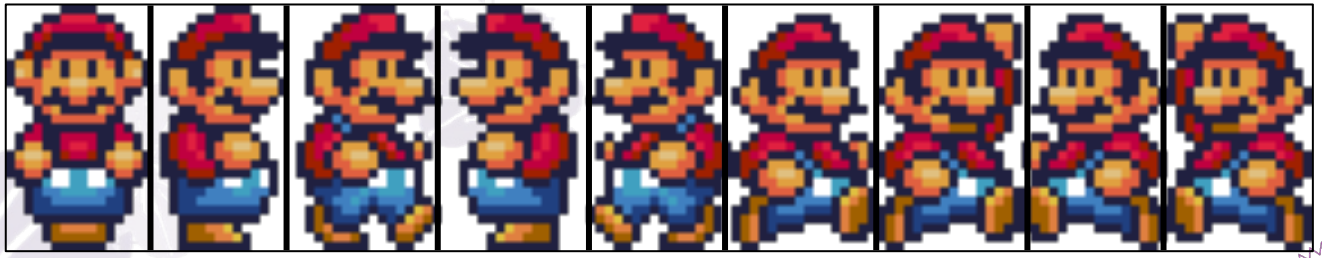


Animation using Spritesheet

- A spritesheet is an image that contains a sequence of images represents the player at different animation frames.
- Spritesheets tend to be smaller files since there's only 1 header for the whole lot, and they also load quicker as there’s just one disk access rather than several.

32px

54px



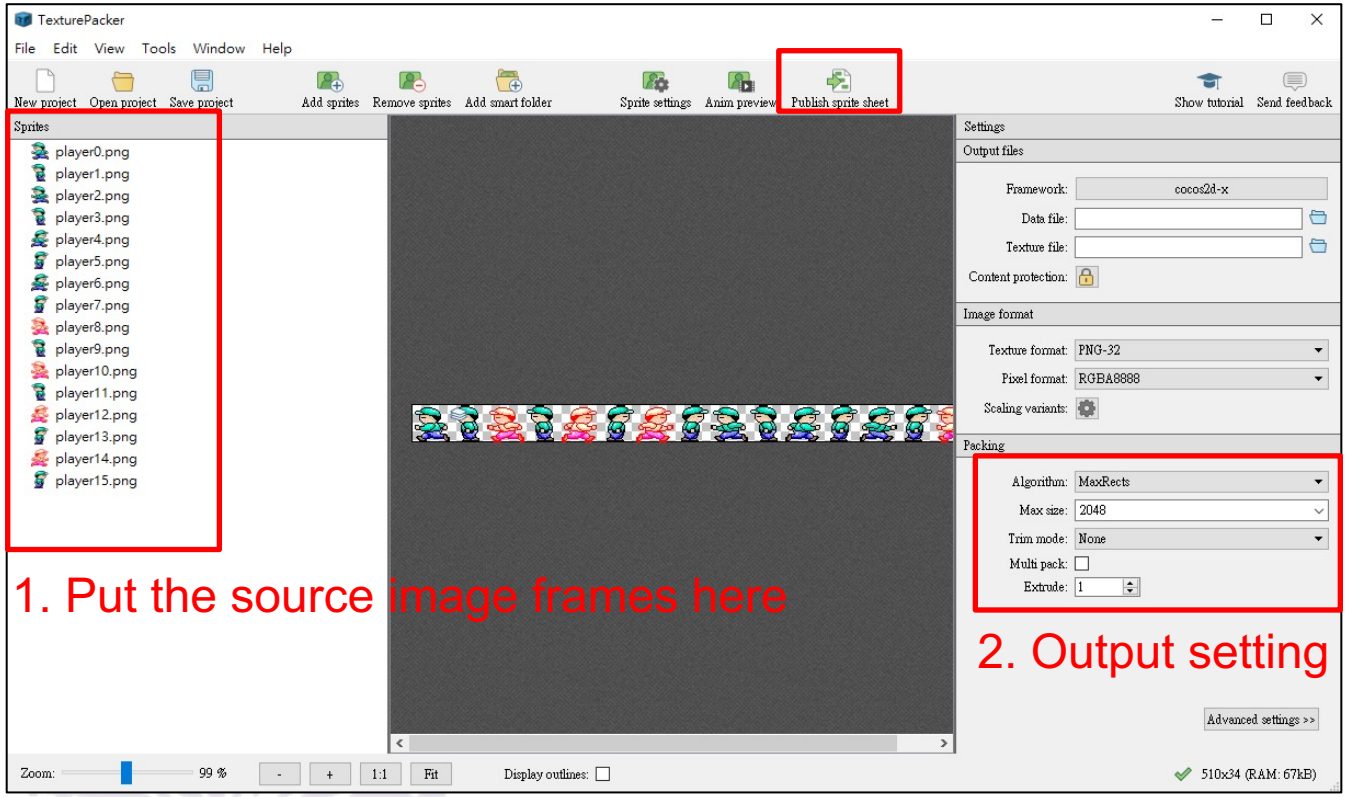
Atlas

- Cocos creator supports spritesheet.
- In Cocos creator, it is called **Atlas**. For better performance, we recommend using atlas rather than single picture.
- Some useful tools like **TexturePacker** or **Zwoptex** are recommended for generating atlas.

Ref: Spritesheet resource website

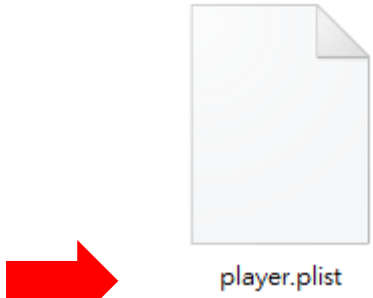
TexturePacker Example

3. Generate Atlas



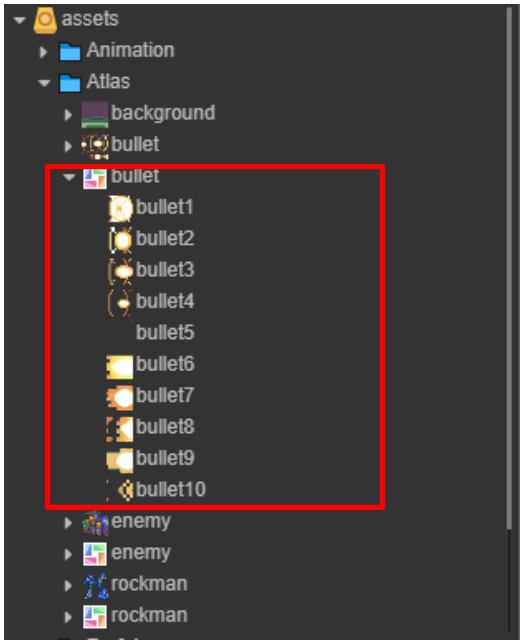
1. Put the source image frames here

2. Output setting



Atlas

- Once you get the **.plist** & **.png** files with the **same file name**, drag **both files** into **Assets Panel** to generate an Atlas in Cocos Creator.

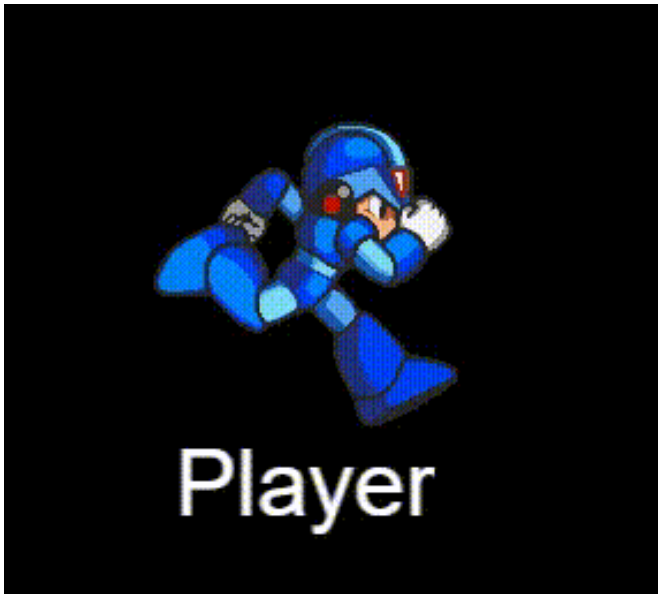


Let’s Create Cool Animations!

- We are going to show you how to add cool animations to the game, step-by-step.

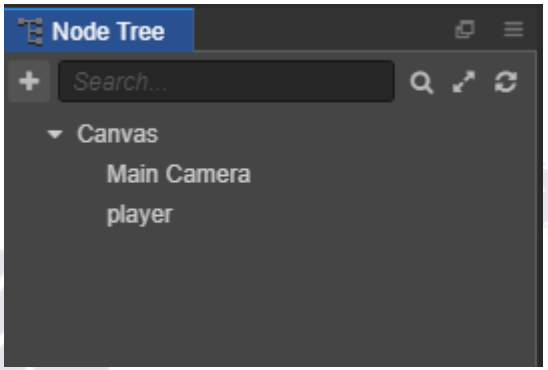
Start with Walking Animation

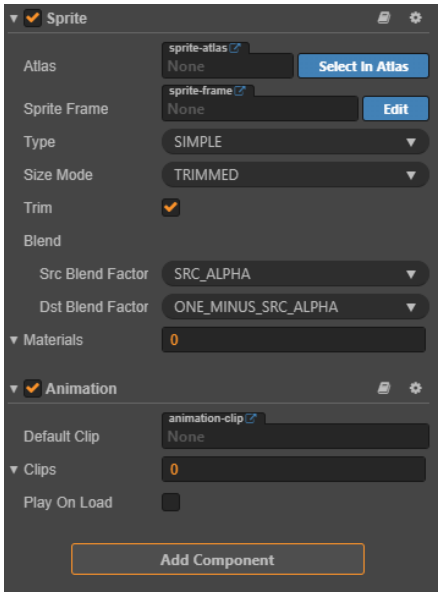
- First, we want to create a Rockman walking animation clip like this:



Create a Sprite Node

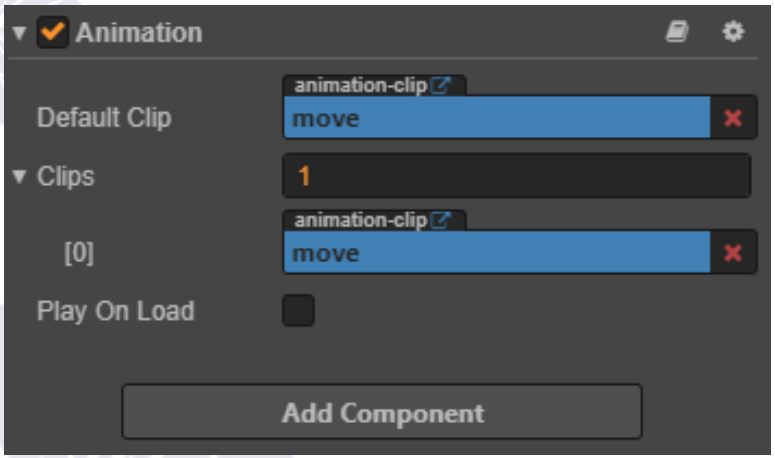
- Before creating animation, first we need to create the node for it.
- Add a sprite node with an **animation component** in the node tree.

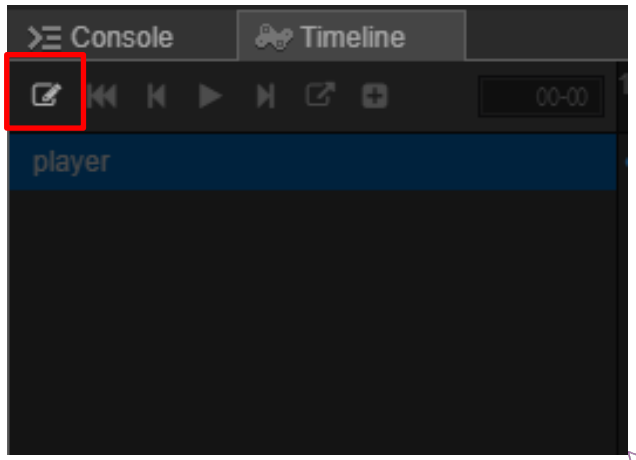




Mount an Animation Clip

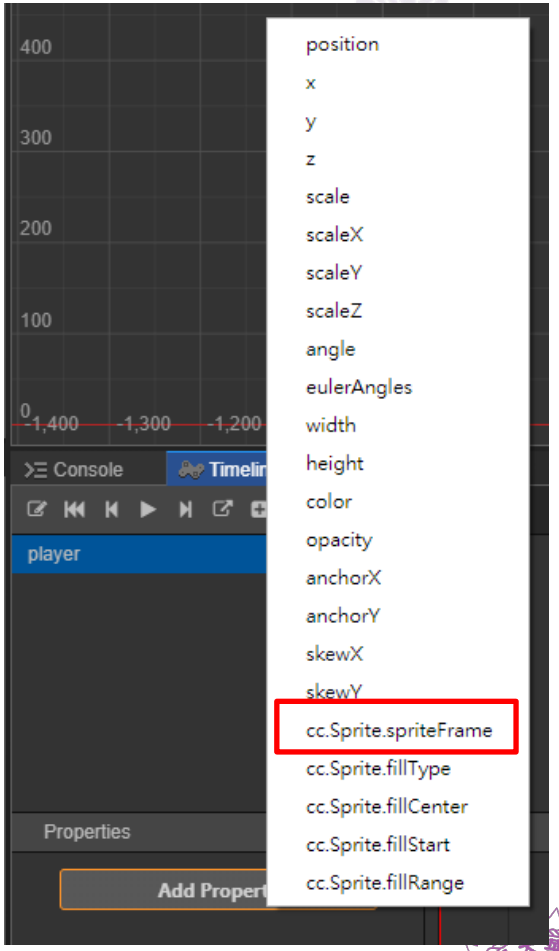
- Next, we create an animation clip and mount it to the animation component.
- Then, start editing the animation clip by click the edit button.



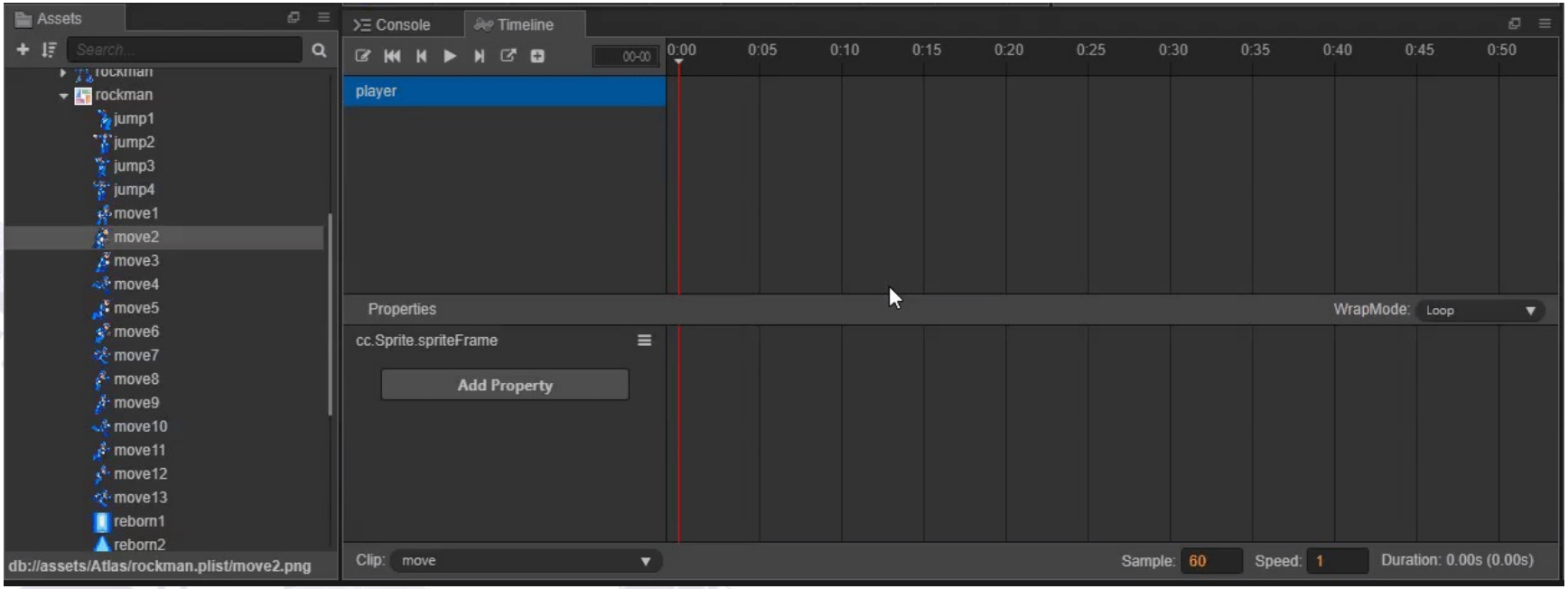


Add Sprite Frames

- We can switch Rockman's images at a specific time by adding a **spriteFrame** property.
- Next, we can use the atlas to edit our animation.

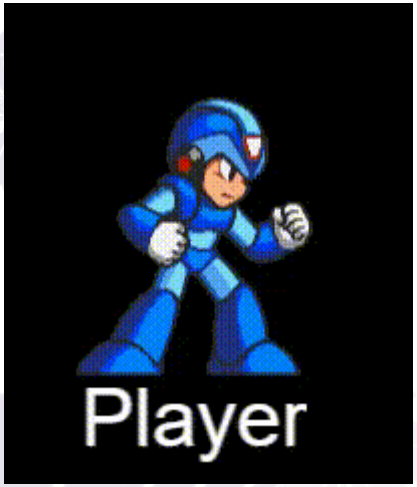


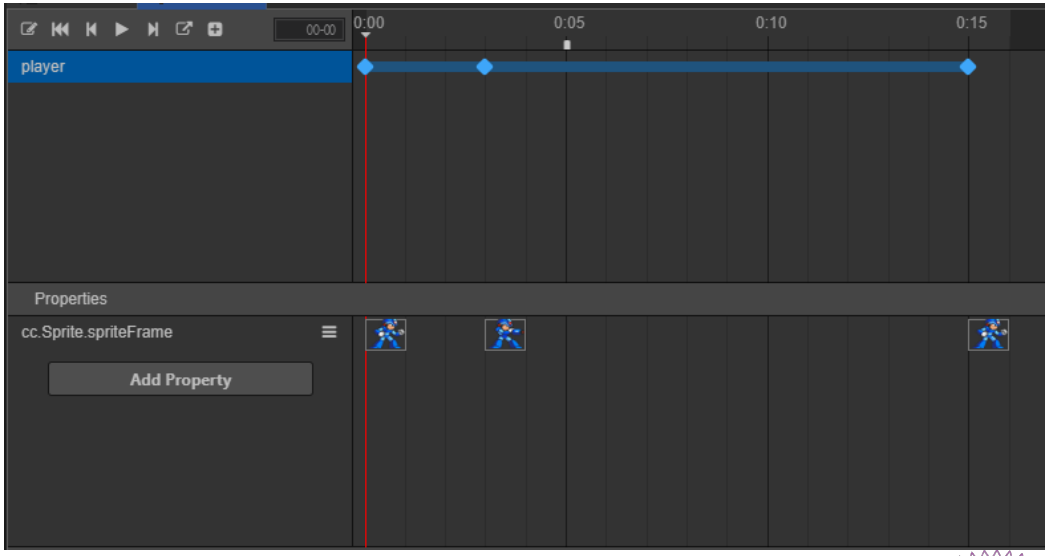
Add Sprite Frames



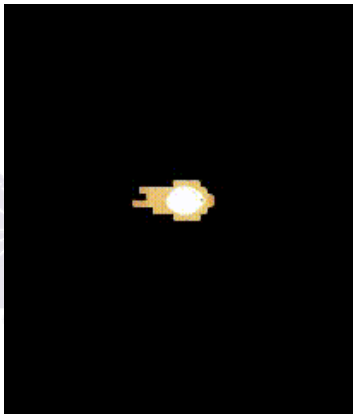
Add Other Animations

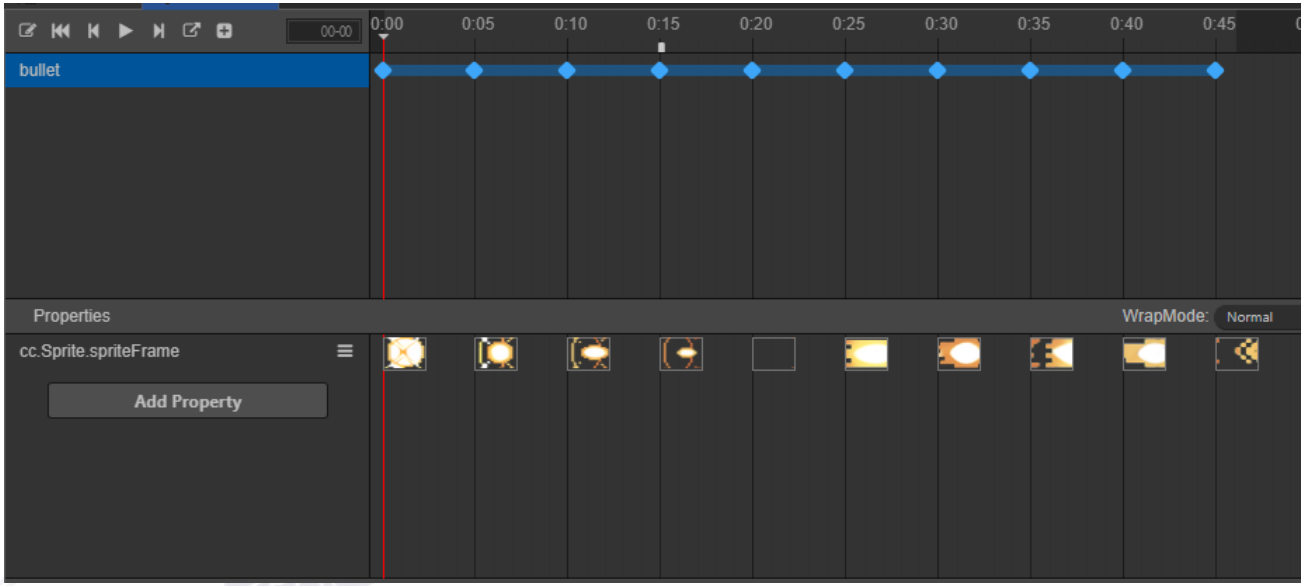
- Now, we can repeat the above steps to add more animation clips, like shoot animation, bullet animation…





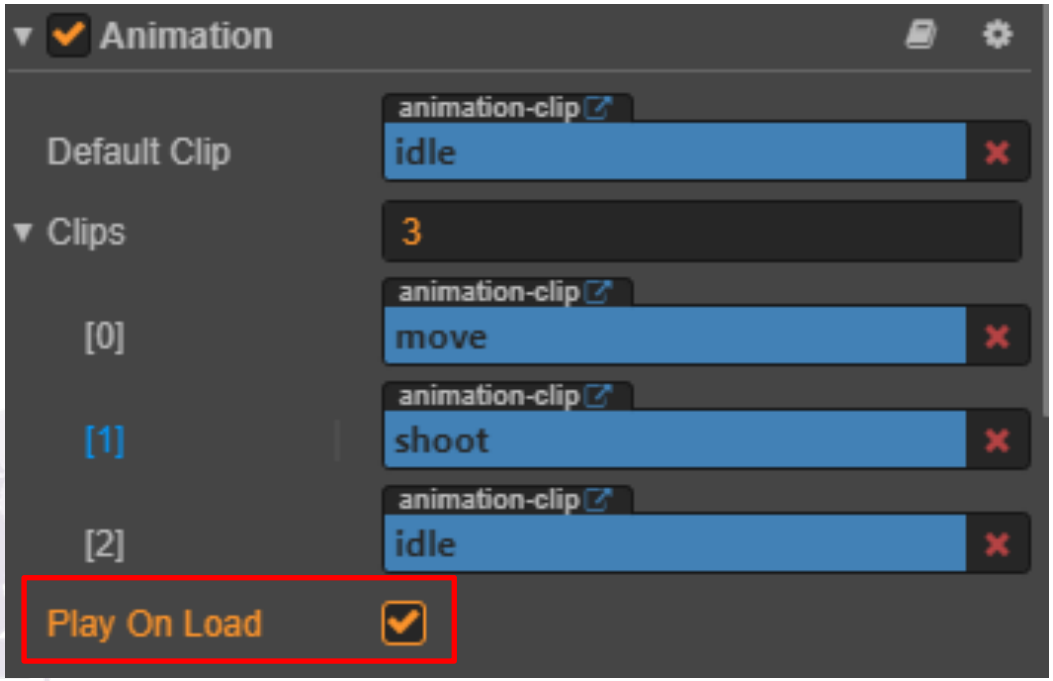
Add Other Animations

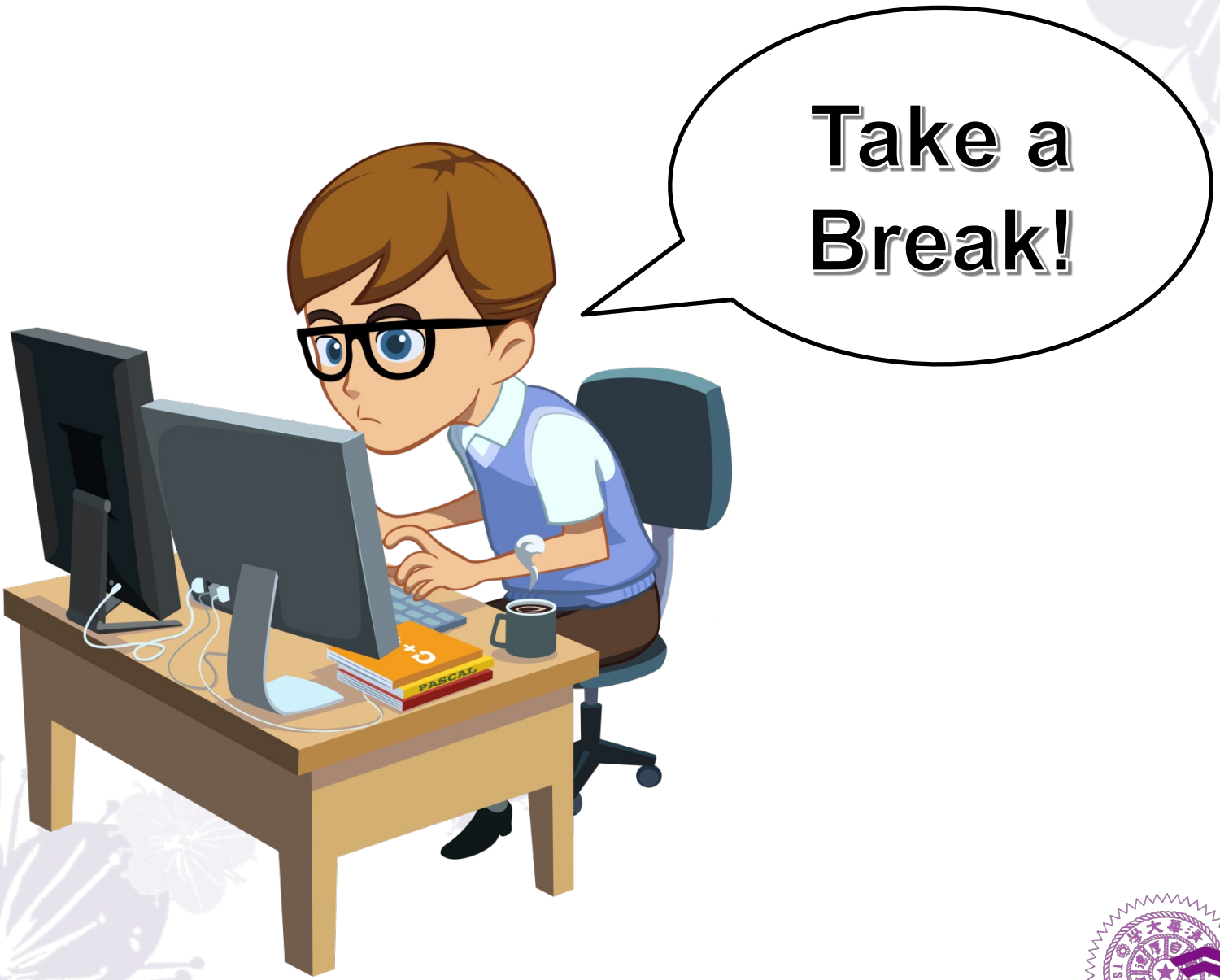




Add Sprite Frames

- If the **Play On Load** is checked, the default animation will play when the node is loaded.




Take a
Break!
C++
PASCAL

Control the Animation by Script

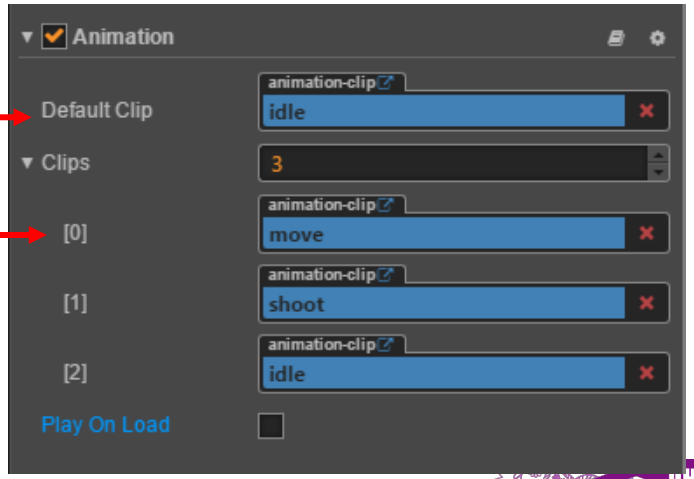
- Like any other components, animation can be controlled by script.
- We use **getComponent(cc.Animation)** to get the animation component from current node.

```
private anim = null;

onload() {
    this.anim = this.getComponent(cc.Animation);
}
```

Play the Animation

- We call **play()** function to play the **default** animation clip.
- We call **play(string)** function to play the animation clip with specified name.

```
this.anim.play(); // play the “idle” clip

this.anim.play(“move”) // play the “move” clip
```



Multiple Animations

- Multiple animations can be played simultaneously using **playAdditive** function.
- The playing of different animations will not affect the playing state of each other.

```
this.anim.playAdditive('position-anim'); // play the first animation
this.anim.playAdditive('rotation-anim'); // play the second animation
```

Pause, Resume, Stop

- Like play function, the approaches of these three functions are the same.

```
this.anim.play('idle');
this.anim.pause('idle');    // pause the "idle" animation
this.anim.pause();          // pause all the animations
this.anim.resume('idle');   // resume the "idle" animation
this.anim.resume();         // resume all the animations
this.anim.stop('idle');     // stop the "idle" animation
this.anim.stop();           // stop all the animations
```

Animation Implicit States

- **Play** state:
  - When we call the **play(clip)** function.
  - Calling the **play(clip)** function again will stop the current clip and play the specified clip from beginning.
- **Pause** state:
  - When we call the **pause(clip)** function.
  - Calling the **play(clip)** function to resume the playing of current clip.
- **Stop** state:
  - When we call the **stop(clip)** function.
  - Calling the **play(clip)** function to play the specified clip from beginning.

AnimationState

- **Animation** provides only simple control functions.
- **AnimationState** provides more animation information and controls.
- There are two ways to get AnimationState.

```
var animState = this.anim.play('idle'); // play() will return associated AnimationState

var animState = this.anim.getAnimationState('idle'); // we can also directly retrieve the AnimationState
```

Modify AnimationState

- After getting an animationState, we can modify parameters of associated animation, such as **speed, duration, wrapMode** …

```
animState.speed = 2; // change animation speed to 2

animState.wrapMode = cc.WrapMode.Loop; // set the wrapMode as "Loop"

animState.repeatCount = 2; // set the loop count to 2 times

animState.repeatCount = Infinity; // set the loop count to infinity
```

Ref: AnimationState Class

Animation Callback

- We can dynamically register a callback function on the specific state of animation.
- The supported states are as follows:
  - **Play**
  - **Stop**
  - **Pause**
  - **Resume**
  - **Lastframe**
  - **Finished**

Animation Callback: On

- You can use **On** to register a callback function.

```
var animation = this.node.getComponent(cc.Animation);

//register “playerMove” function on different states
animation.on(‘play’,          this.playerMove, this);
animation.on(‘stop’,          this.playerMove, this);
animation.on(‘lastframe’,     this.playerMove, this);
animation.on(‘finished’,      this.playerMove, this);
animation.on(‘pause’,         this.playerMove, this);
animation.on(‘resume’,        this.playerMove, this);
```

Animation Callback: Off

- You can use **Off** to cancel a callback function.

```
var animation = this.node.getComponent(cc.Animation);

//cancel the callback function
animation.off('play',        this.playerMove, this);
animation.off('stop',        this.playerMove, this);
animation.off('lastframe',   this.playerMove, this);
animation.off('finished',    this.playerMove, this);
animation.off('pause',       this.playerMove, this);
animation.off('resume',      this.playerMove, this);
```

Animation Callback: Once

- If you want to call the function once, you can use **once** instead of on.

```
var animation = this.node.getComponent(cc.Animation);

//register “playerMove” function once on different states
animation.once(‘play’,          this.playerMove, this);
animation.once(‘stop’,          this.playerMove, this);
animation.once(‘lastframe’,     this.playerMove, this);
animation.once(‘finished’,      this.playerMove, this);
animation.once(‘pause’,         this.playerMove, this);
animation.once(‘resume’,        this.playerMove, this);
```

Animation Event

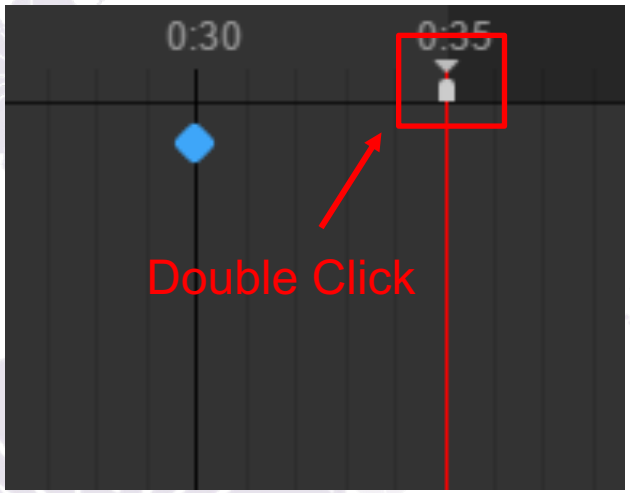
- We can also register callback functions at the specific frames in an animation clip by adding an **animation event**.

Animation Event

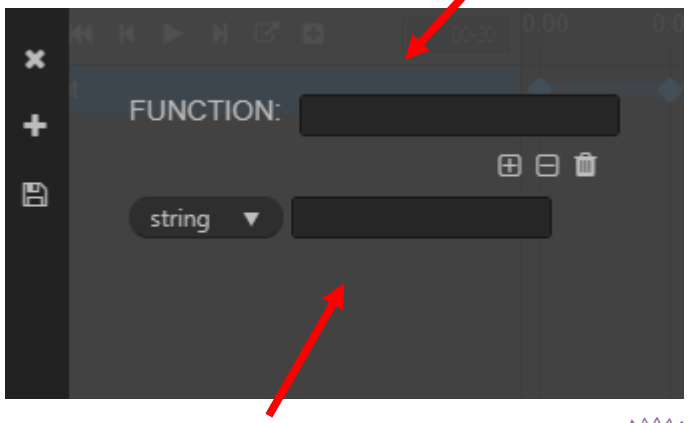
- The **white rectangles** on the timeline represents the added animation event.
- Double click on the white rectangle to open the event editor.
- Type in the name and parameters of the callback function in the editor.

Animation Event

- Note that only three types of parameters are supported:
  - **Boolean**, **String** and **Number**.



The callback function you want to call

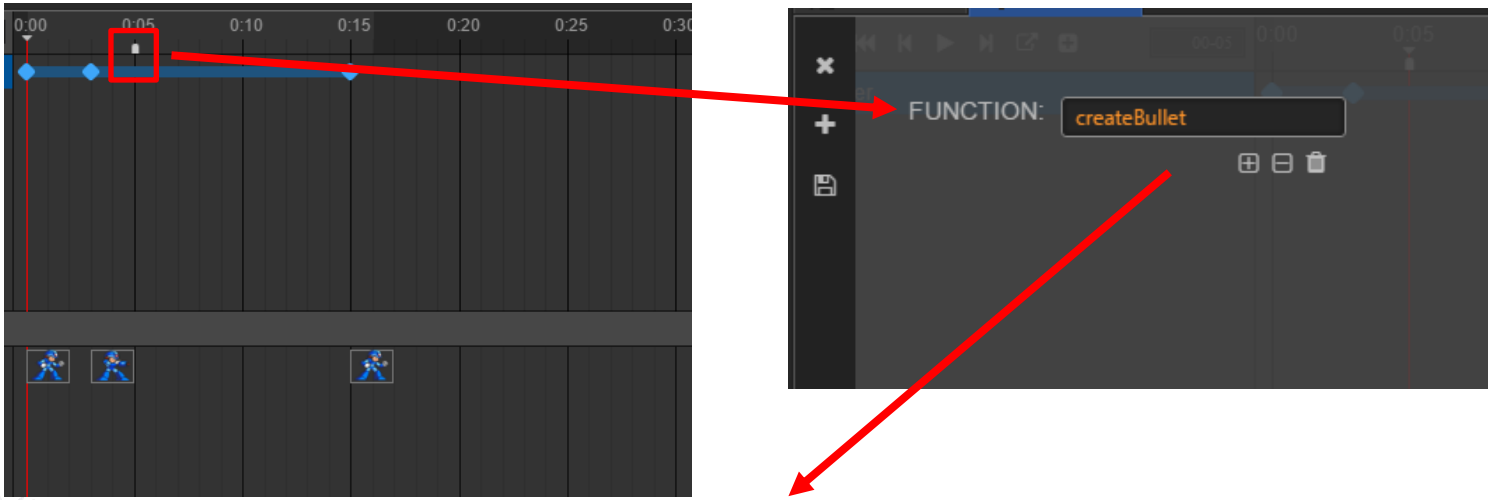


The parameter be passed to function

Add Bullet in Shoot Clip

- Add an **animation event** when the spriteframe of shoot clip switched to ‘shoot1.png’.
- Register **createBullet** function as a callback function.

Add Bullet in Shoot Clip



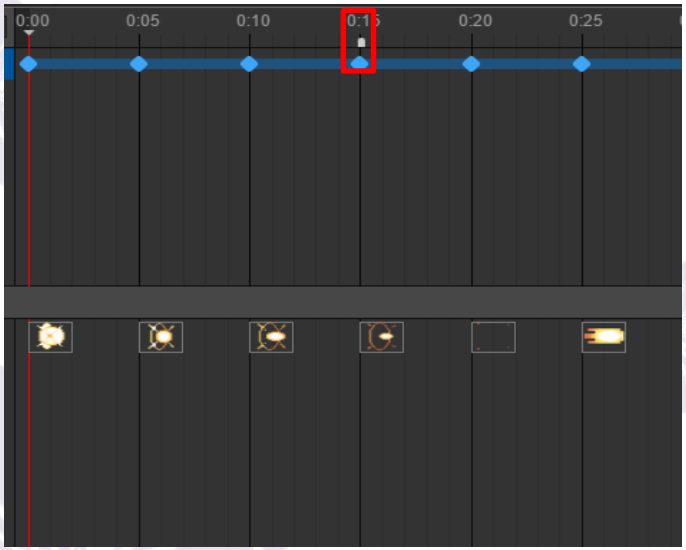

```
private createBullet() {
    let bullet = null;

    if(this.bulletPool.size() > 0)
        bullet = this.bulletPool.get(this.bulletPool);

    if(bullet != null)
        bullet.getComponent('Bullet').init(this.node);
}
```

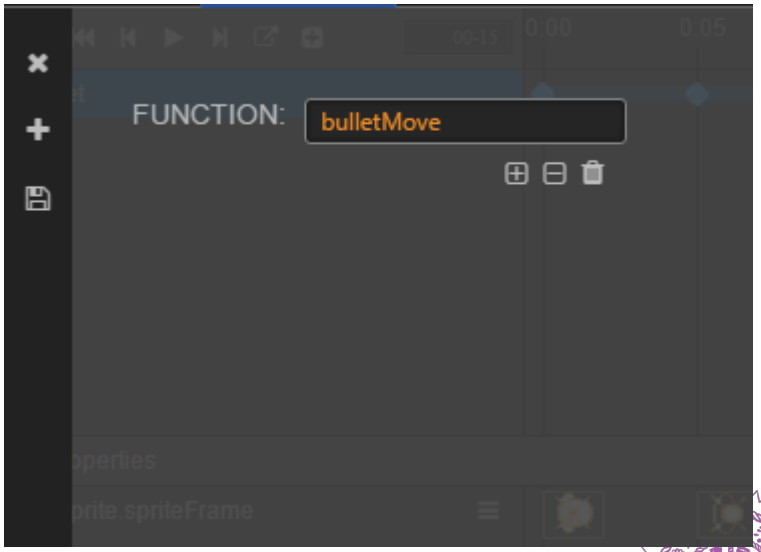
Now the function will execute when the frame is played.

Player.ts

Add BulletMove in Bullet Clip

- Register **bulletMove** function as callback function when the spriteframe of bullet clip switched to ‘bullet4.png’.





Player Control

- After finishing the shoot animation, now we need to make the player move and shoot.
- Use **AnimationState** to get current playing clip and switch between **idle, move and shoot** according to the user input.

Player Control: Logic

- When the shoot clip is playing, no other animations can interrupt.
- Use **isPlaying** property to get the current state of shoot clip.

```
playerAnimation() {
    if(!this.anim.getAnimationState('shoot').isPlaying) {
    }
}
```

Player Control: Logic

- If the shoot clip is not playing, we have to deal with three cases:
  - z key for left moving
  - x key for right moving
  - j key for shooting
- Use **zDown**, **xDown**, and **jDown** to get the current input status.

Player Control: Codes

```
playerAnimation() {
    if(!this.anim.getAnimationState('shoot').isPlaying) {
        if(this.jDown) {
        }
        else if(this.zDown) {
        }
        else if(this.xDown) {
        }
        else {
        }
    }
}
```

Player.ts

Player Control: Logic

- Use **this.node.scaleX** and **this.playerSpeed** to change player's direction and moving speed and use **this.animateState** to record the current playing clip.
- Use the value of **this.animateState** to determine whether we need to re-play move clip or need to play shoot clip.
- When all key is released, we need to **stop the animation and go back to the idle clip.**

Player Control: Codes

```
update(dt){
    this.node.x += this.playerSpeed * dt;
    this.playerAnimation();
}
```

Player.ts

Player Control: Codes

```
playerAnimation() {
    // we can play shoot anim only when shoot anim is not playing
    if(!this.anim.getAnimationState('shoot').isPlaying) {
        if(this.jDown) { // press j to shoot
            this.node.scaleX = (this.zDown)? -1 : (this.xDown) ? 1 : this.node.scaleX;
            this.playerSpeed = 0;
            this.animateState = this.anim.play('shoot');
        }
        else if(this.zDown) { // press z to turn left
            this.node.scaleX = -1;
            this.playerSpeed = -300;
            if(this.animateState == null || this.animateState.name != 'move')
                this.animateState = this.anim.play('move');
        }
        else if(this.xDown) { // press x to turn right
            this.node.scaleX = 1;
            this.playerSpeed = 300;
            if(this.animateState == null || this.animateState.name != 'move')
                this.animateState = this.anim.play('move');
        }
        else {
            if(this.animateState != null) {
                this.anim.play('idle');
                this.animateState = null;
            }
            this.playerSpeed = 0;
        }
    }
}
```

Player.ts

Mission Complete


thank you!
Question?