

Software Studio

軟體設計與實驗

Cocos Creator : Particle System

Hung-Kuo Chu

Department of Computer Science
National Tsing Hua University

CS2410

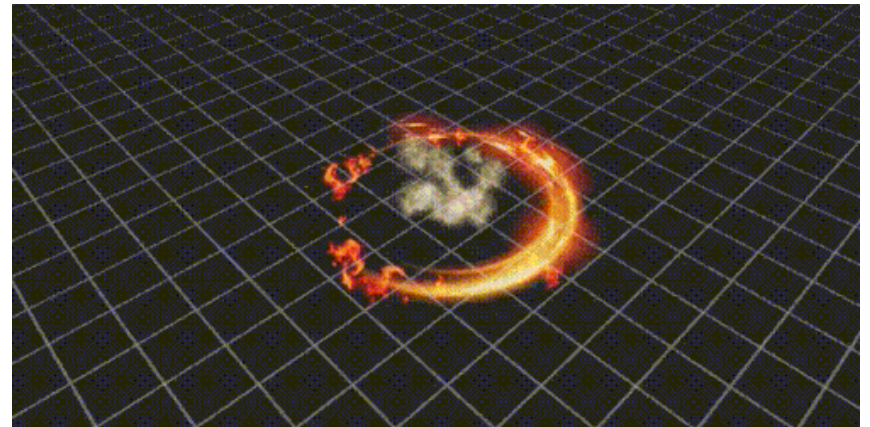
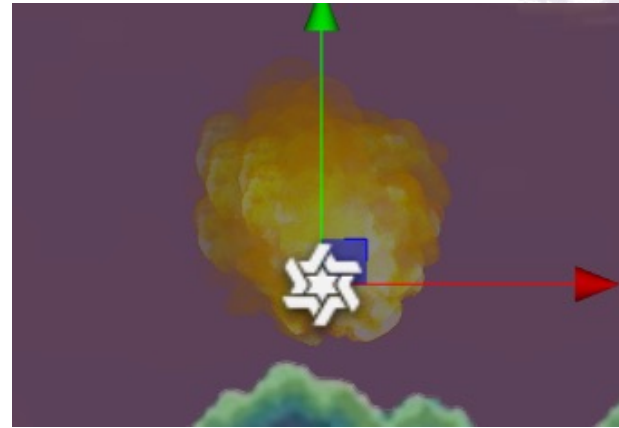


Particle System

- Particle system is a technology that can simulate some specific fuzzy phenomena in graphics.
- This component is used to read Particle Resources data and perform a series of operations such as play, pause, destroy, etc.

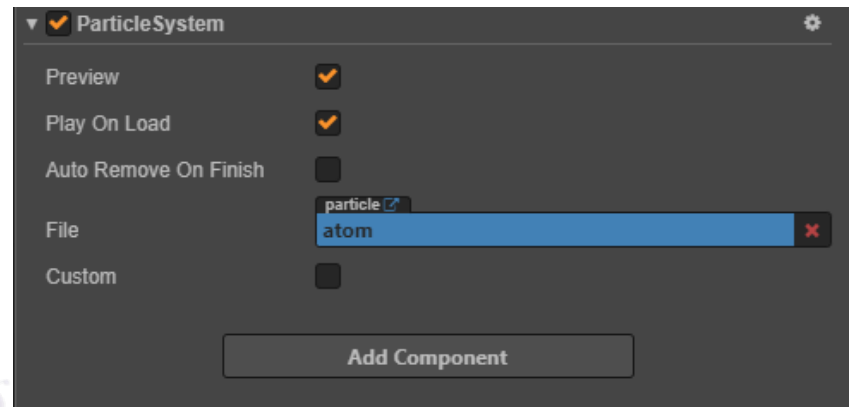
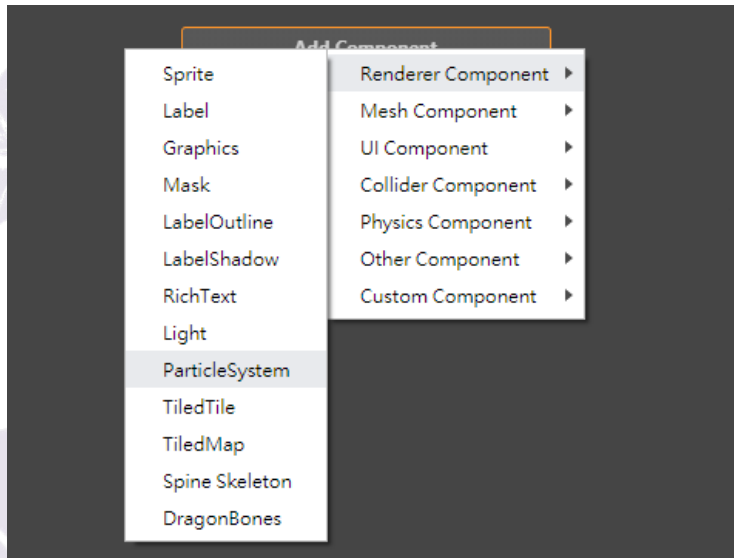


Example



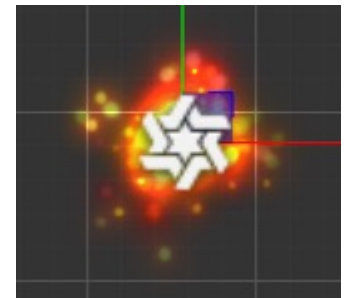
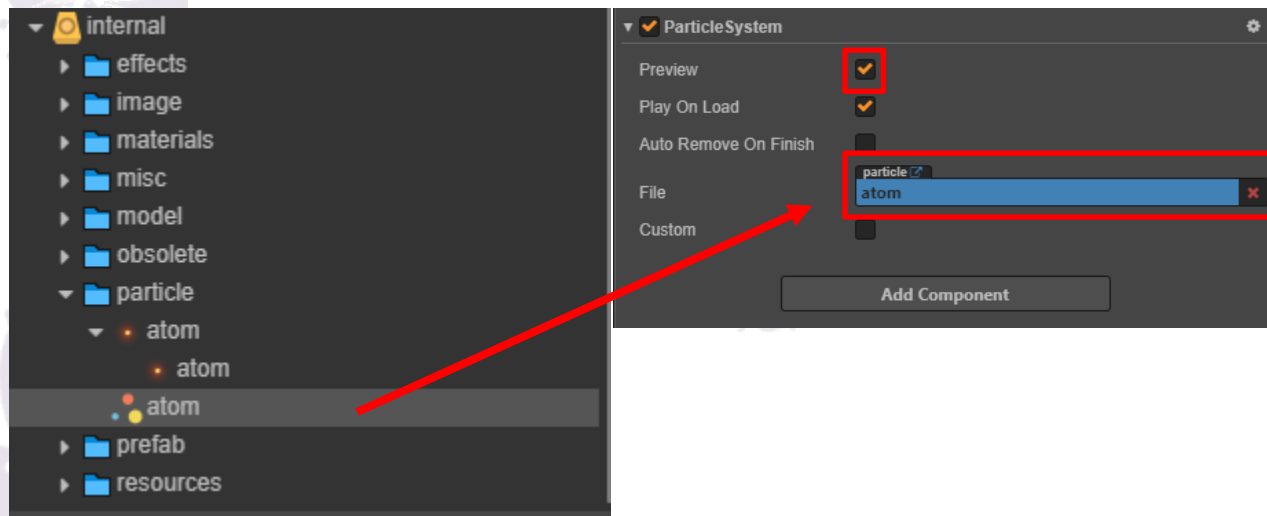
Create Particle System Components

- Node → Add Component → Renderer Component → ParticleSystem



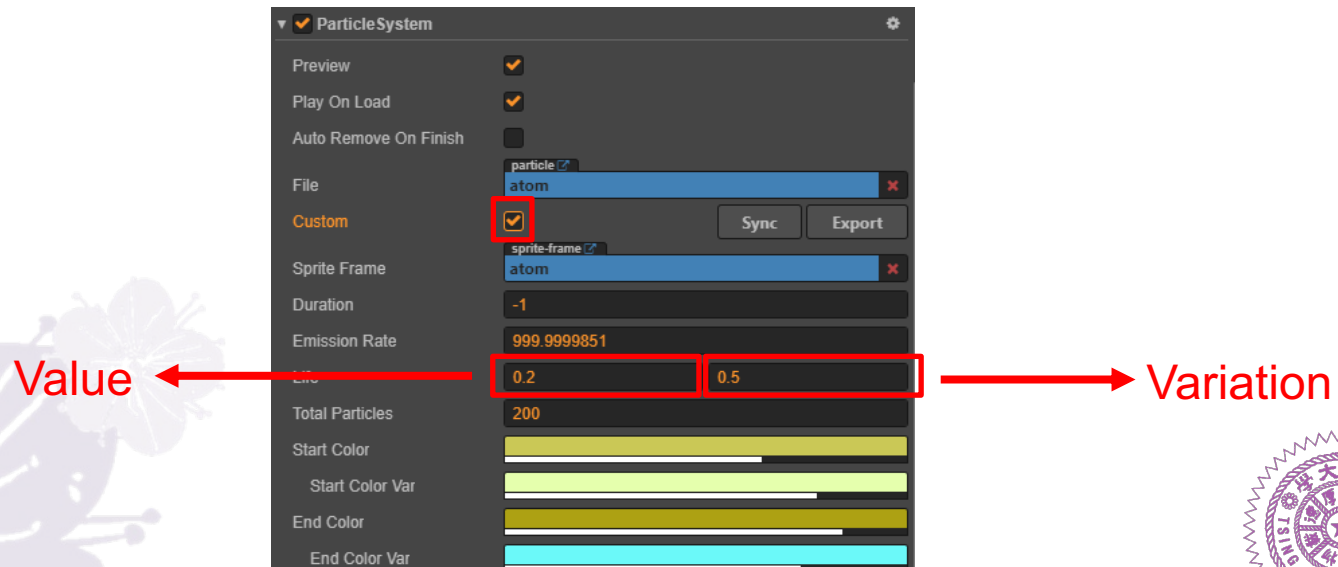
Create Particle System Components

- Cocos Creator provides a **Particle File** at internal
→ particle → atom
- Drag the particle file to the file column and check preview to see the effect



Custom Particle System

- Checking Custom allows you to change the properties of the particle system
- Most properties have two columns, the value on the left and the amount of variation on the right



Common Properties

Properties	Function Explanation
Preview	Play particle in edit mode.
Play On Load	If set to true, the particle system will automatically start playing on onLoad.
Auto Remove On Finish	Indicate whether the owner node will be auto-removed when it has no particles left.
File	The plist file.
Custom	If set custom to true, then use custom properties instead of read particle file. When this property is turned on, you can customize the following part of the particle properties
Sprite Frame	SpriteFrame of Particle System.
Duration	Duration
Emission Rate	Emission rate of the particles.
Life	Life and variation of each particle setter.



Common Properties(Cont'd)

Properties	Function Explanation
Total Particle	Maximum particles of the system.
Start Color	Start color of each particle.
Start Color Var	Variation of the start color.
End Color	Ending color of each particle.
End Color Var	Variation of the end color.
Angle	Angle and variation of each particle setter.
Start Size	Start size and variation in pixels of each particle.
End Size	End size and variation in pixels of each particle.
Start Spin	Start angle and variation of each particle.



Common Properties(Cont'd)

Properties	Function Explanation
End Spin	End angle and variation of each particle.
Source Pos	Source position of the emitter.
Pos Var	Variation of source position.
Position Type	Particles movement type. Including FREE, RELATIVE, GROUPED three types, refer to PositionType API for details.
Emitter Mode	Particles emitter modes. Including GRAVITY, RADIUS three types, refer to EmitterMode API for details.
Src Blend Factor	The source image blend mode. Refer to BlendFactor API for details.
Dst Blend Factor	The destination image blend mode. Refer to BlendFactor API for details.



Position Type

- **FREE:** Living particles are attached to the world and are unaffected by emitter repositioning.
- **RELATIVE:** Living particles are attached to the world but will follow the emitter repositioning.
 - Use case: Attach an emitter to an sprite, and you want that the emitter follows the sprite.
- **GROUPED:** Living particles are attached to the emitter and are translated along with it.



Emitter Mode

- **GRAVITY:** Uses gravity, speed, radial and tangential acceleration.
- **RADIUS:** Uses radius movement + rotation.

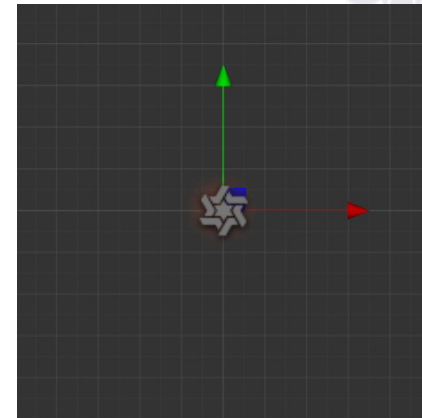
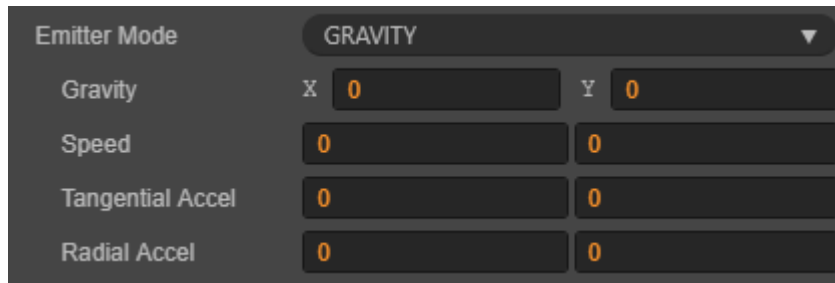
Emitter Mode	GRAVITY	
Gravity	X 0	Y 0
Speed	180	50
Tangential Accel	80	0
Radial Accel	0	0
Rotation Is Dir	<input type="checkbox"/>	

Emitter Mode	RADIUS	
Start Radius	0	0
End Radius	0	0
Rotate Per S	0	0

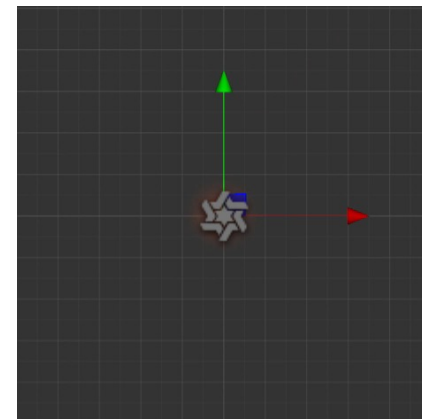
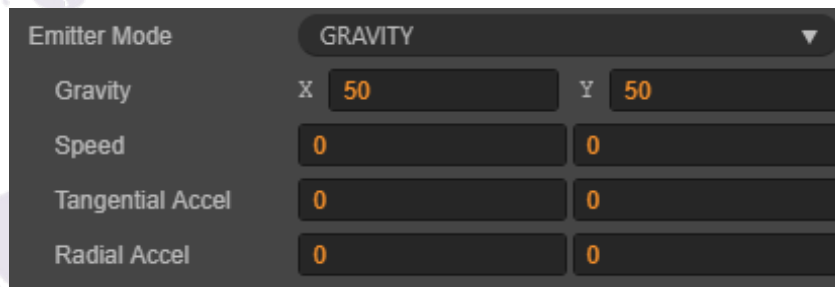


GRAVITY Mode

- No gravity

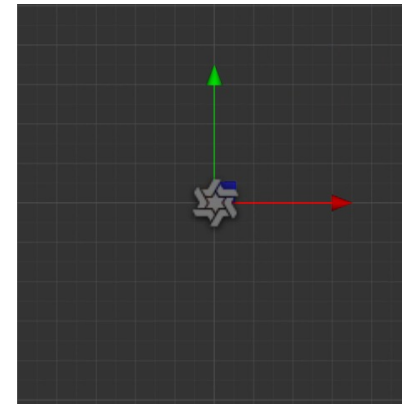
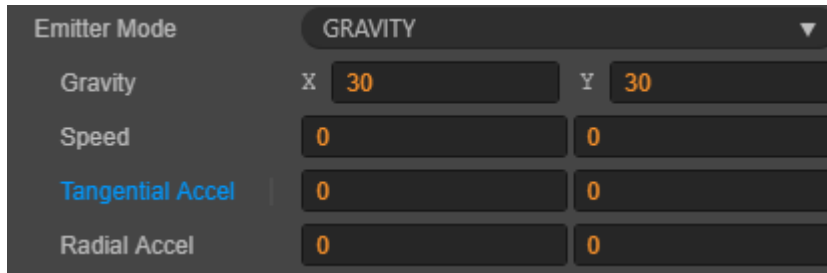


- Gravity in the (50, 50) direction

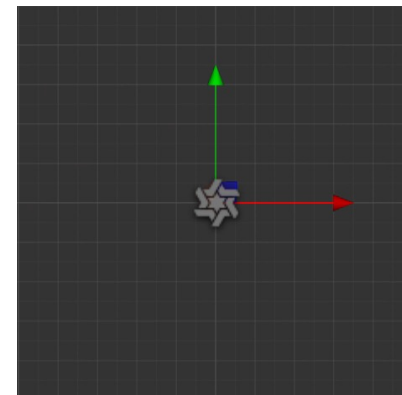
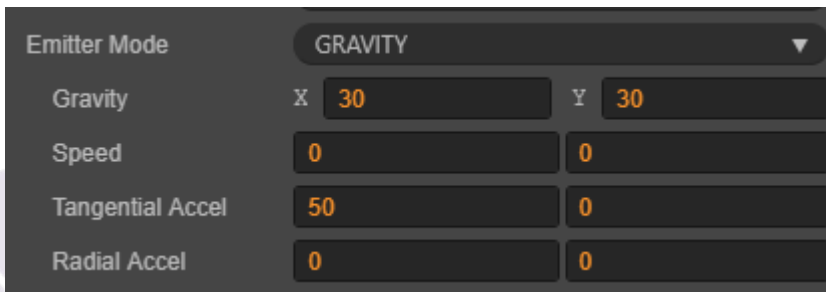


GRAVITY Mode(Cont'd)

- No tangential acceleration

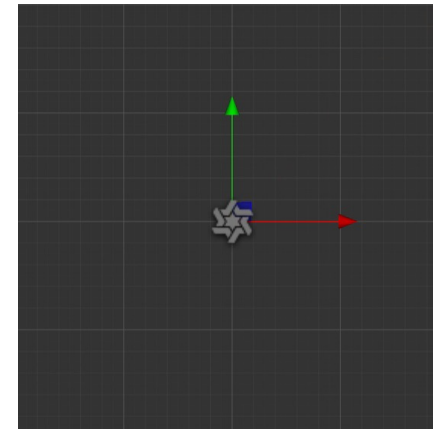
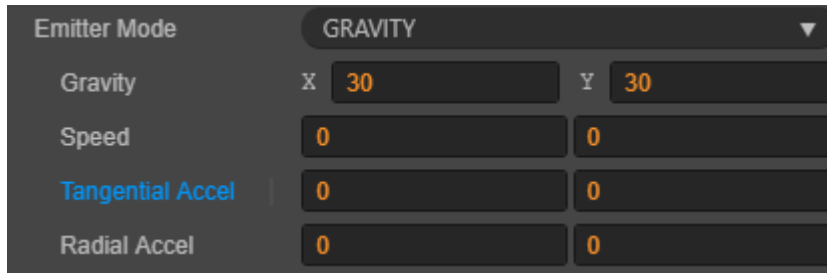


- Tangential acceleration 50

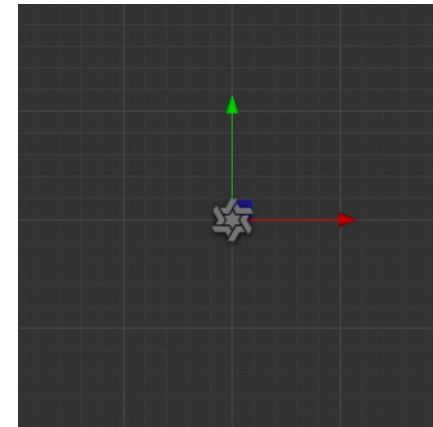
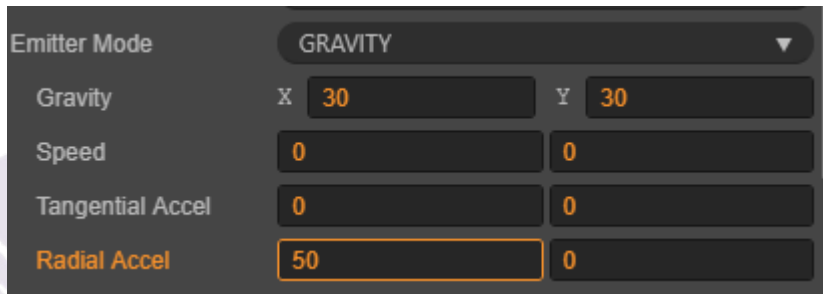


GRAVITY Mode(Cont'd)

- No radial acceleration



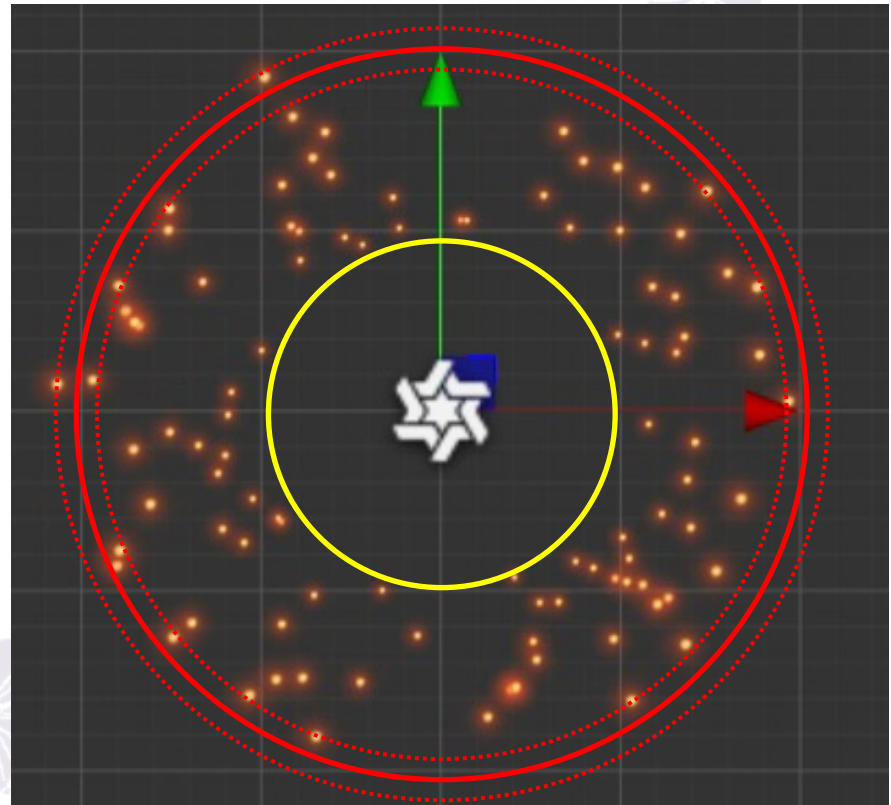
- Radial acceleration 50



RADIUS Mode

Angle	0	360
Start Size	20	0
End Size	10	0
Start Spin	0	0
End Spin	0	0

Emitter Mode	RADIUS ▾	
Start Radius	100	10
End Radius	50	0
Rotate Per S	60	30

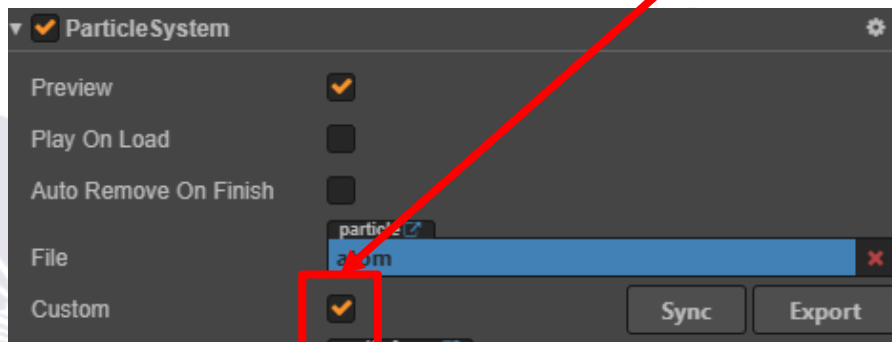
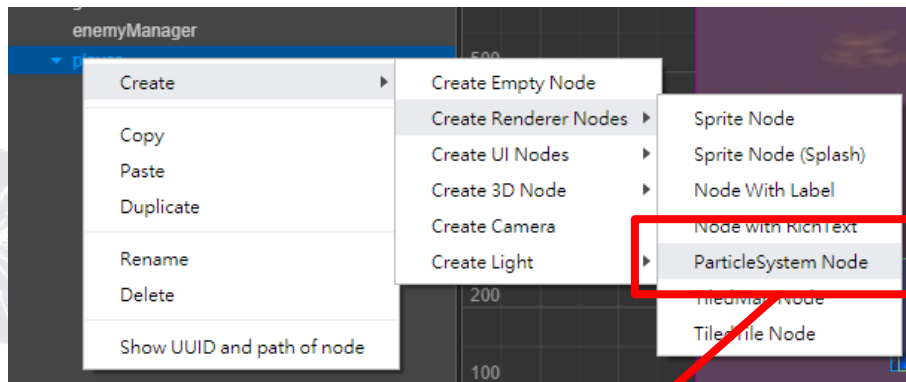


Effects Production Example: Charge Shot



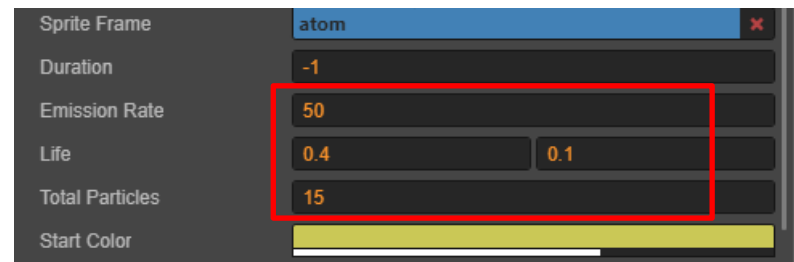
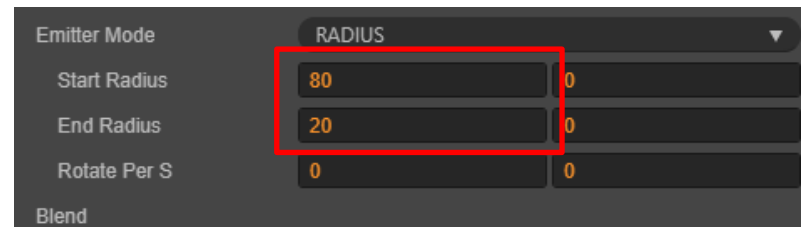
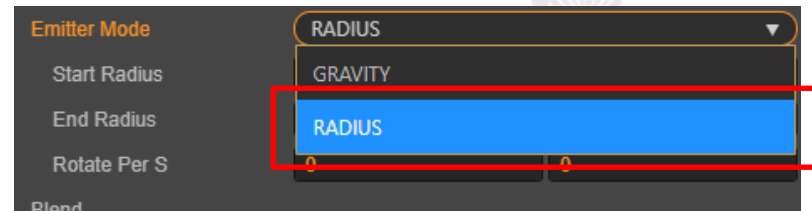
Create Particle Node

- Create a particle node under the player node
- Turn on custom to customize particle properties




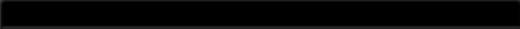


Modify Properties

- We want the particles to emit inward in the shape of a circle.
- So we change the **Emitter Mode** to **RADIUS**.
- Then set **Start Radius** to 80 and **End Radius** to 20.
- Next, we adjust **Emission Rate**, **Life** and **Total Particle** to reduce the number of particles.



Modify Properties

- Finally, we change the color and size of the particles to make them look better.

Total Particles	10	
Start Color		
Start Color Var		
End Color		
End Color Var		
Angle	360	360
Start Size	70	5
End Size	20	5
Start Spin	-47.3699989	0



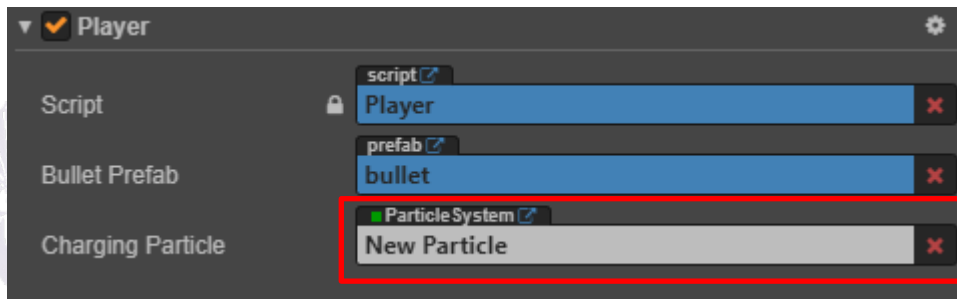
Control Particle By Script

- In order for the effect to appear while holding down the attack, we need to control the particle system by script.
- First, uncheck **Play On Load**.



Control Particle By Script(Cont'd)

- Let the particle system just completed be a property of the player script



```
@property(cc.ParticleSystem)  
private chargingParticle: cc.ParticleSystem = null;
```

Player.ts



Particle Reset Function

- We can use `resetSystem()` to kill all living particles and restart particle system.

```
private resetParticle()  
{  
    // kill all particles and restart particle system  
    this.chargingParticle.resetSystem();  
}
```

Player.ts

onKeyDown()

- Use **scheduleOnce()** to reset the particle system 0.2 seconds after pressing the attack key.

```
onKeyDown(event)
{
    switch(event.keyCode)
    {
        ...

        case cc.KEY.j:

            if(this.jDown == false){
                this.scheduleOnce(this.resetParticle, 0.2);
            }

            this.jDown = true;

            break;
    }
}
```

Player.ts



onKeyUp()

- Use **unschedule()** to ensure that the particle system will not be reset if the attack key is released in advance.
- Also use **stopSystem()** to make the particle system stop emitting when the attack key is released

```
onKeyUp(event)
{
    switch(event.keyCode)
    {
        ...

        case cc.KEY.j:

            this.jDown = false;

            // stop particle system
            this.chargingParticle.stopSystem();

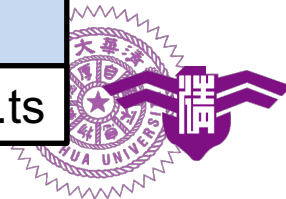
            if(!this.anim.getAnimationState('shoot').isPlaying)
            {
                this.node.scaleX = (this.zDown) ? -1 : (this.xDown) ? 1 : this.node.scaleX;

                this.playerSpeed = 0;

                this.animateState = this.anim.play('shoot');
            }

            break;
    }
}
```

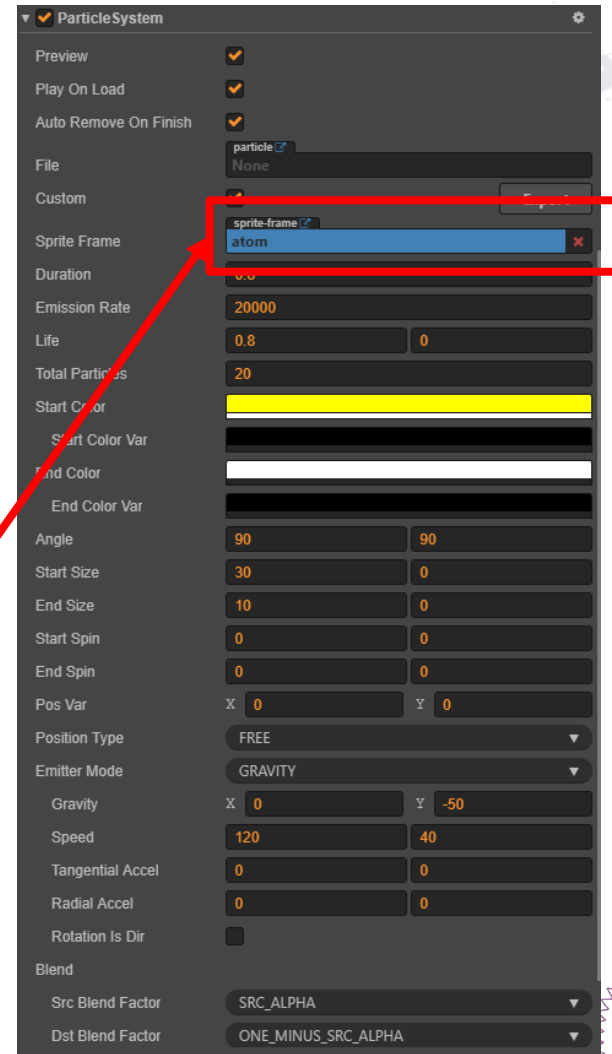
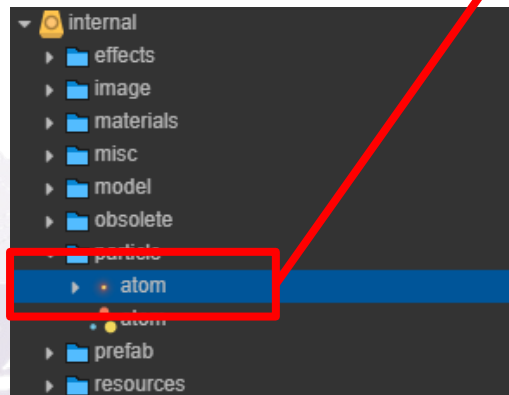
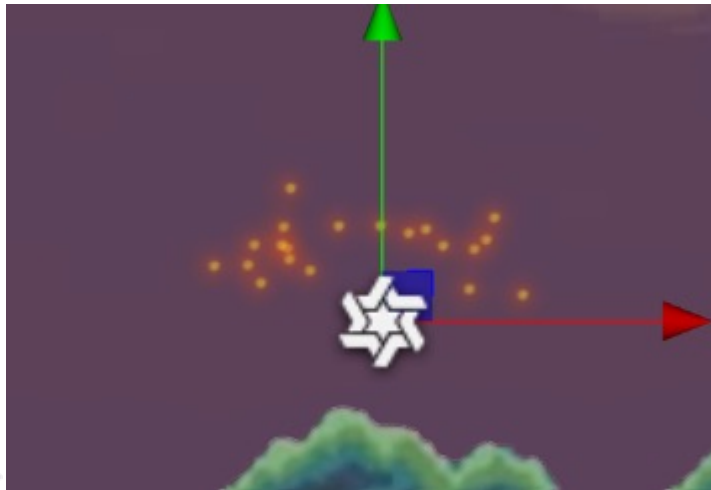
Player.ts



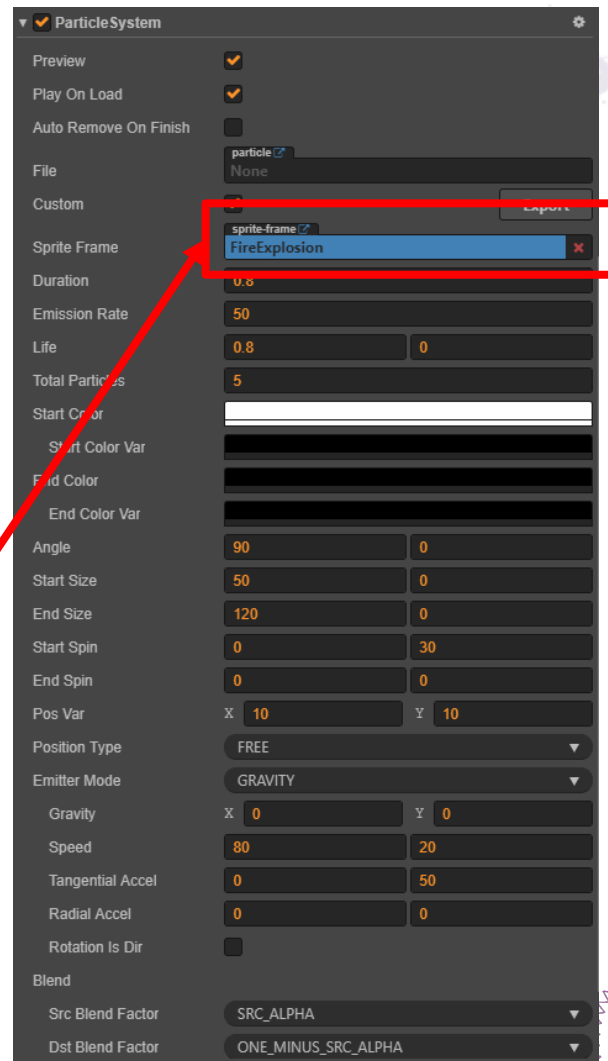
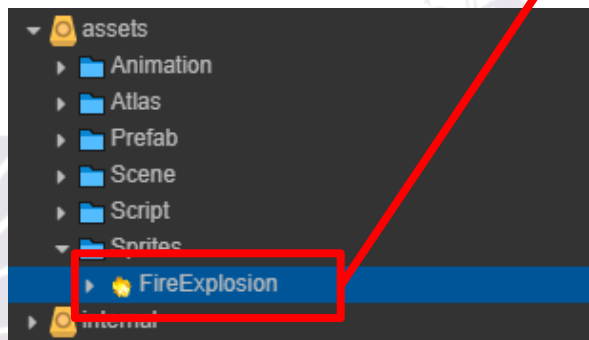
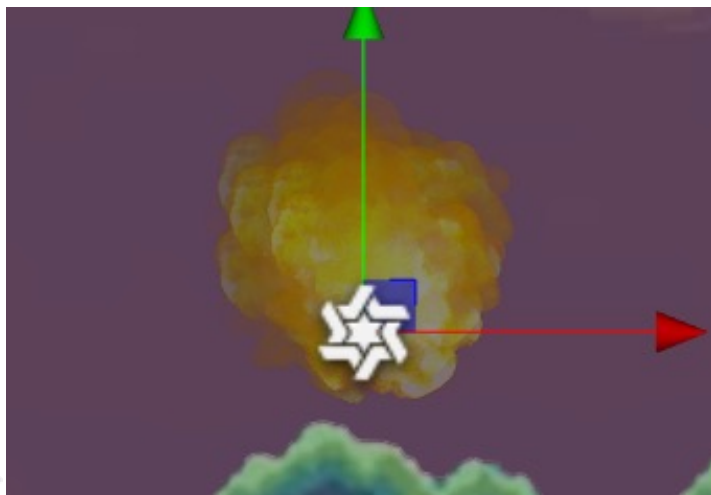
Additional Example: Explosion Effect



Explosion Effect - Embers

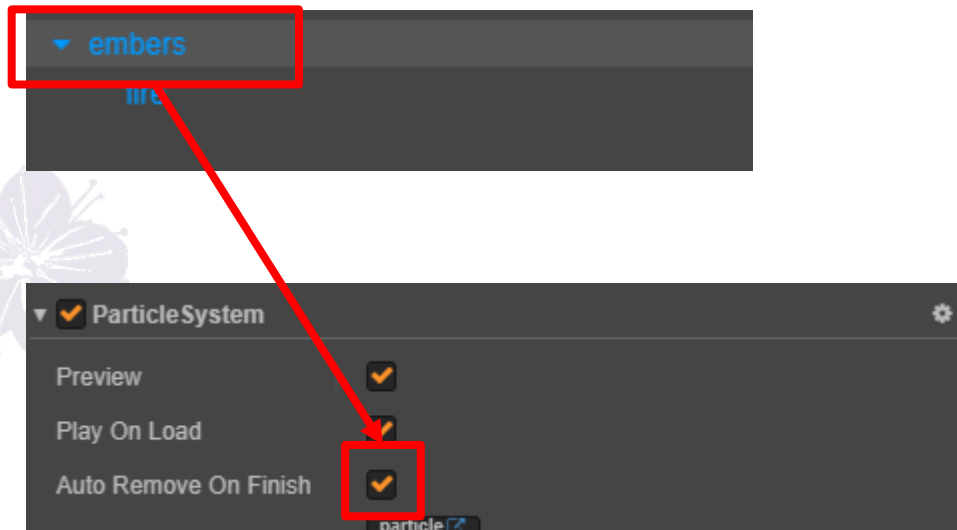


Explosion Effect - Fire



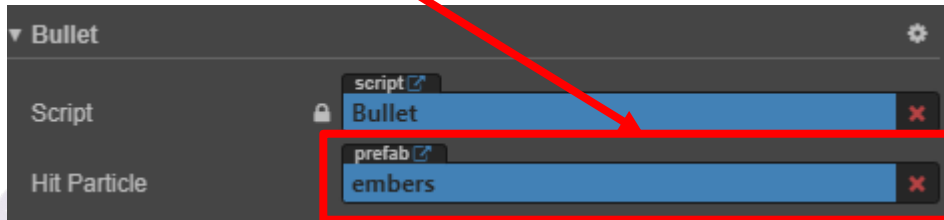
Trigger Explosion Effect

- We want the explosion effect to be automatically deleted when it is finished, so check **Auto Remove On Finish**.



Trigger Explosion Effect

```
@property(cc.Prefab)  
private hitParticle: cc.Prefab = null;
```



```
//detect collision with enemies  
onBeginContact(contact, selfCollider, otherCollider)  
{  
  this.scheduleOnce(() => {  
    let hitEffect = cc.instantiate(this.hitParticle);  
    this.node.parent.addChild(hitEffect);  
    hitEffect.setPosition(this.node.position);  
  
    this.node.stopAllActions();  
  
    this.anim.stop();  
  
    this.bulletManager.put(this.node);  
  }, 0.1); // for better animation effect, I delay 0.1s when bullet hits the enemy  
}
```

Bullet.ts



Online Particle Effect

- <http://www.effecthub.com/particle2dx>
- <https://pixijs.io/pixi-particles-editor/>



thank
you!

Question

