Let’s Build Our Own Wack-A-Mole Game with Cocos Creator!

Preface

Before you start building our game, let’s download the tutorial from Github. You can download the completed version as well, but try to build your game with us first.

If you have trouble with our lesson, please study the following documentation:

- Prepare the development environment
  - Cocos Creator 2.1.2
  - Visual Studio Code
- Document
  - Animation system
  - Collision system
- API
  - Animation
  - CollisionManager

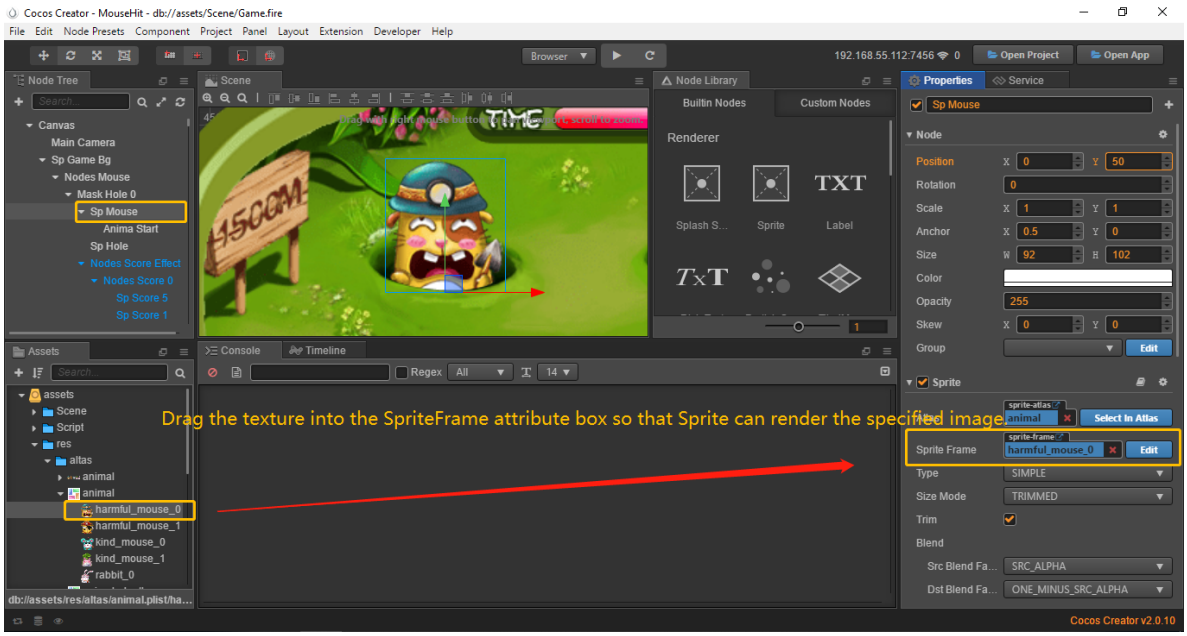
Let's get started!

1. Use Animation System

We can use the Animation Editor in Cocos Creator to animate a simple mouse exit.

1.1 Add mouse sprite to the scene:

The first thing we need to do is add the sprite to the game. To do this, we need to create a node in Sp Mouse, which is not yet capable of rendering images. Let’s add a Sprite component to Sp Mouse that can render the specified picture to the screen. Then all we have to do is drag the picture “harmful_mouse_0” into the SpriteFrame property box in the Sprite component.

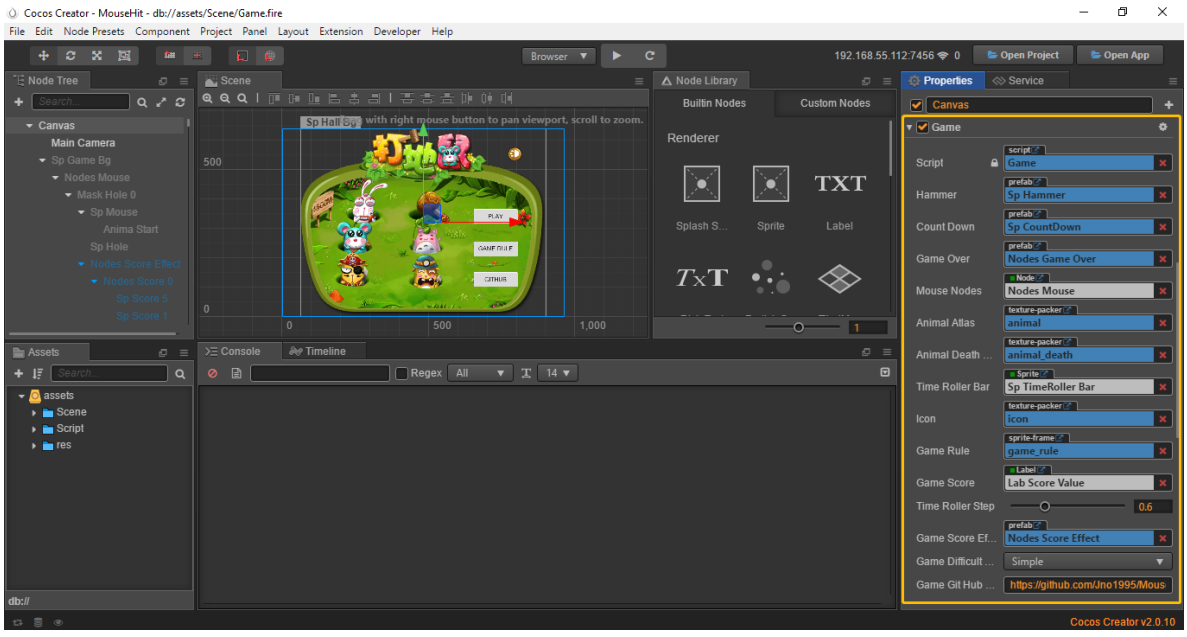


1.1.1 Add game logic:

Now we need to add the gameplay script Game.js into the game and add the required property inspector parameters to it:

1.1.2 Code for Game.js:

Please add the following code to Game.js



```
var Difficult = cc.Enum({
    Simple: 1000,
    Ordinary: 2500,
    Difficult: 5000
});

properties: {
        hammer: {
            default: null,
            type: cc.Prefab
        },

        countDown: {
```

```
        default: null,
        type: cc.Prefab
    },

    gameOver: {
        default: null,
        type: cc.Prefab
    },

    mouseNodes: {
        default: null,
        type: cc.Node
    },

    animalAtlas: {
        default: null,
        type: cc.SpriteAtlas
    },

    animalDeathAtlas: {
        default: null,
        type: cc.SpriteAtlas
    },

    timeRollerBar: {
        default: null,
        type: cc.Sprite
    },

    icon: {
        default: null,
        type: cc.SpriteAtlas
    },

    gameRule: {
        default: null,
        type: cc.SpriteFrame
    },

    gameScore: {
        default: null,
        type: cc.Label
    },

    timeRollerStep: {
        default: 1,
        range: [0, 2, 0.1],
        slide: true
    },

    gameScoreEffect: {
        default: null,
        type: cc.Prefab
    },

    gameDifficultScore:{
        default: Difficult.Simple,
        type: Difficult,
```

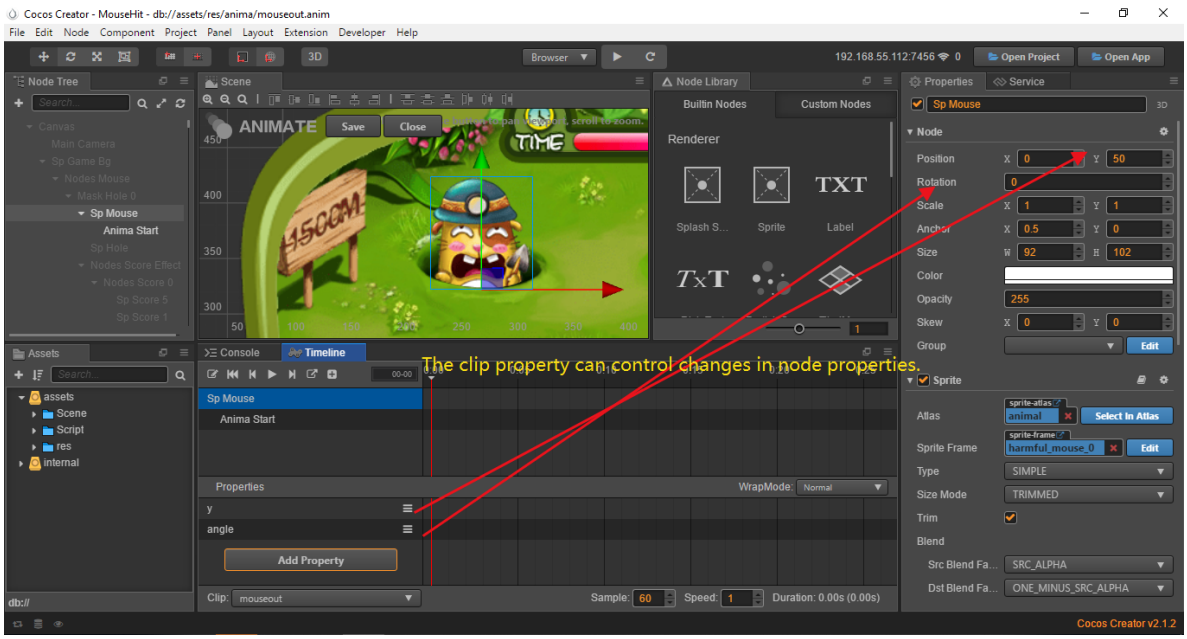
```
            tooltip: "Simple:2000\n Ordinary:4000\n Difficult:6000",
        },

        gameGitHubUrl:"",

        _isCollider: false,
        _mouseNode: null,
        _mouseIndexArr: [],
        _score: 0
},
```

1.2 Add animation effects to the mouse sprite：

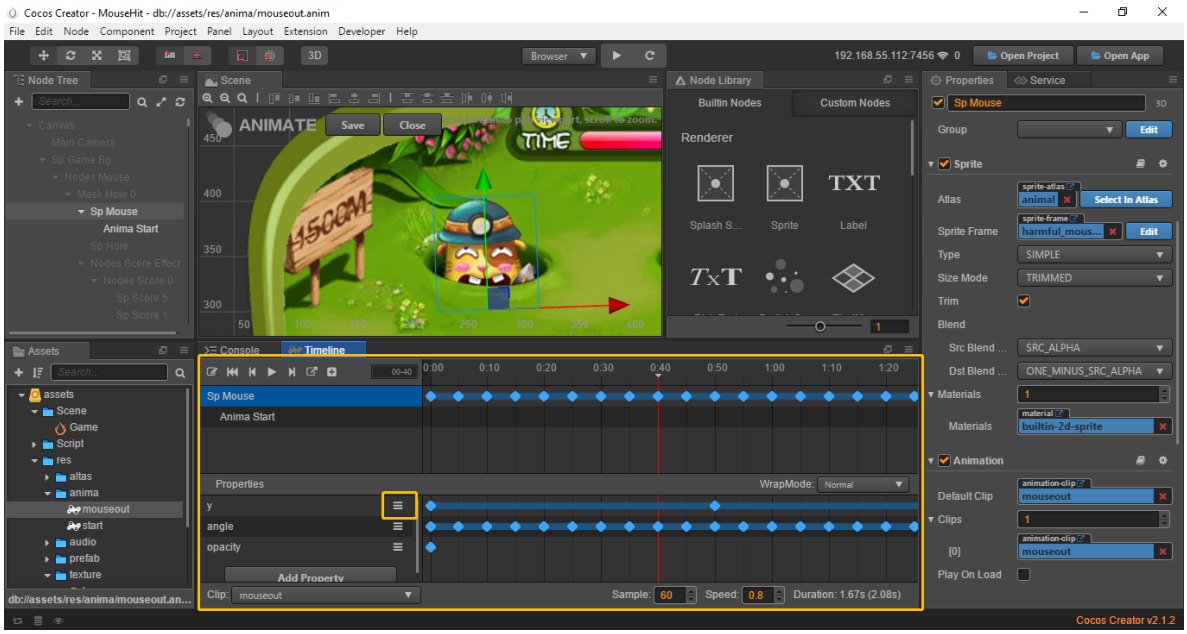
Now we need to add an Animation component to the node and create a new AnimaClip resources inside the assets folder named mouseout. The AnimaClip we just created records when the node property changes, and the Animation component can read this data to make the node change. Open the anima folder in the Assets section and choose the “mouseout” animation frame. The editor does not open the animation editor by default. We click the Animation Editing button to enter the animation editor and add two properties to the mouseout animation: y coordinates and angle.



1.3 Insert keyframes for the clip and adjust the animation

There is a small button on the right side of the animation frame property. After right clicking, choose insert key frame, you can now create an animated frame on the specified timeline. When it is created, the properties related to the Clip attribute of the node are recorded in Clip. If we continue to add animation frames with different Clip attributes in the latter timeline, we can produce continuous animation changes.

At this time, we want the mouse to constantly shake from the rat hole and disappear, so we set the Y coordinate of the node at `0:00`, `0:50`, `1:40`, etc., and the corresponding Y coordinate is `-50`, `0`, `50`. Because we want the mouse to shake constantly, it is necessary to set different angles in a number of time periods, with a corresponding angle equal to `-2`, `0`, `2`.



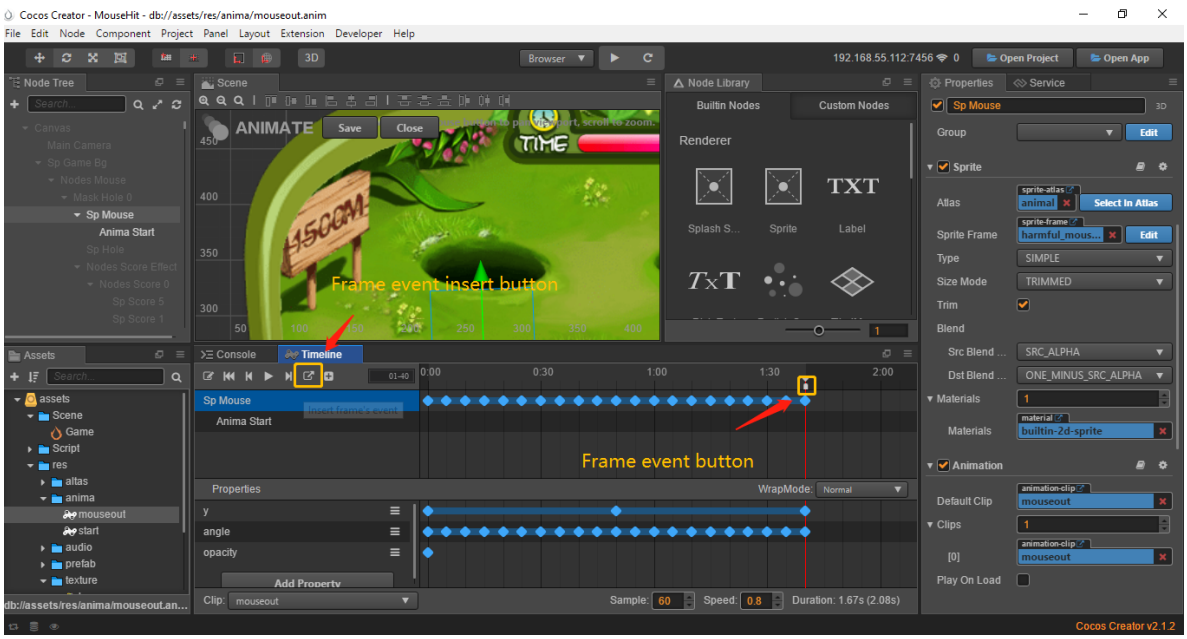
1.4 Continue adding a frame callback for this animation frame

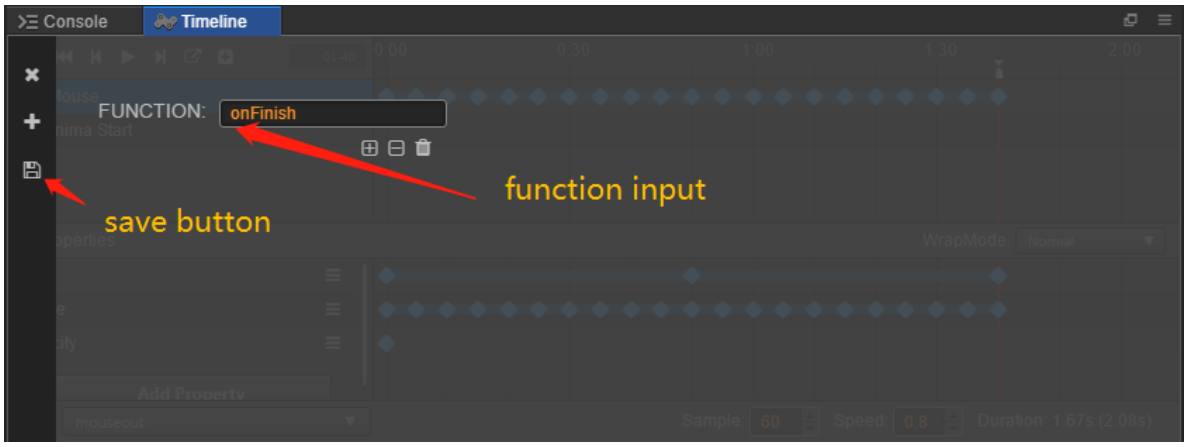
We first create a JS script named AnimationCallBack.js, to add a function called onFinish to the cc.Class and add it to the Sp Mouse node:

1.4.1 Code：

```
onFinish () {
    this.node.opacity = 0;
    this.node.getChildByName("Anima Start").opacity = 0;
    this.node.getChildByName("Anima Start").getComponent(cc.Animation).stop();
    this.node.getComponent(cc.BoxCollider).enabled = false;
},
```

Back in the animation editor, we select the timeline of the last frame and click the button to insert the frame event so that the frame event is inserted in the current frame. Right-click the generated frame event icon, select Edit, to enter the frame event editing interface, enter onFinish, in the input box, and then click the save button to exit the animation editor mode, so that the frame event will be added successfully.





At this point, we have completed the mouse wizard animation settings.

---

2 Turn the mouse into a hammer

Mice need to use a hammer to faint, so we turn the mouse into a hammer, so that wherever the mouse moves, you can stun the mouse at any time.

2.1 Turn the mouse into a hammer by listening to system events.

System events are the most basic game input mode, and each game should manage their system events well. Cocos Creator engine encapsulates the management of system events, and we use on function to monitor all system events. When listening to user Touch events or mouse events, we can access the touch object data touched by the current game by accessing the callback parameter event. It is a EventTouch object, and we can use API in ccTouch to get the information contained in this object, which contains the coordinate letter of the current mouse Interest.

We need to write a function called 'initEventListener' , which can listen to critical system events and manipulate the game content in the event callback. After the function is created, this function takes effect once in the start callback of Game.js.

2.1.1 Code：

```
initEventListener () {
        this.node.on(cc.Node.EventType.MOUSE_MOVE, (event)=>{
            this.onBeCreateHammerEvent(event.getLocation());
        },this);

        this.node.on(cc.Node.EventType.TOUCH_MOVE, (event)=>{
            this.onBeCreateHammerEvent(event.getLocation());
        },this);

        this.node.on(cc.Node.EventType.TOUCH_START, (event)=>{
            this.onBeCreateHammerEvent(event.getLocation());
            this.onHammerClicked();
            if (this.gameRuleNode) {
                var gameRuleFadeOut = cc.fadeOut(1);
                this.gameRuleNode.runAction(gameRuleFadeOut);
            }
        },this);
```

```
        this.node.on(cc.Node.EventType.TOUCH_END, (event)=>{
            this.onHammerClicked();
        },this);
}
```

2.2 Generate a hammer

We create a function called 'onBeCreateHammerEvent' to generate a hammer. Let's first create a Sp Hammer node in the scene and add Sprite, to it to render the 'hammer.png' image. When the addition is complete, drag it under the assets folder and form Prefab. When we let the mouse enter the game screen, we instantiate the Sp Hammer prefab into the game and become the hammer we need, and it will move with the mouse.

2.2.1 Code：

```
onBeCreateHammerEvent (position) {
     if (!cc.isValid(this.hammerNode)) {
         this.hammerNode = cc.instantiate(this.hammer);
         this.hammerNode.zIndex = cc.macro.MAX_ZINDEX;
         this.hammerNode._isCollider = false;
         this.node.addChild(this.hammerNode);
     }
     this.hammerNode.position = this.node.convertToNodeSpaceAR(position);
},
```

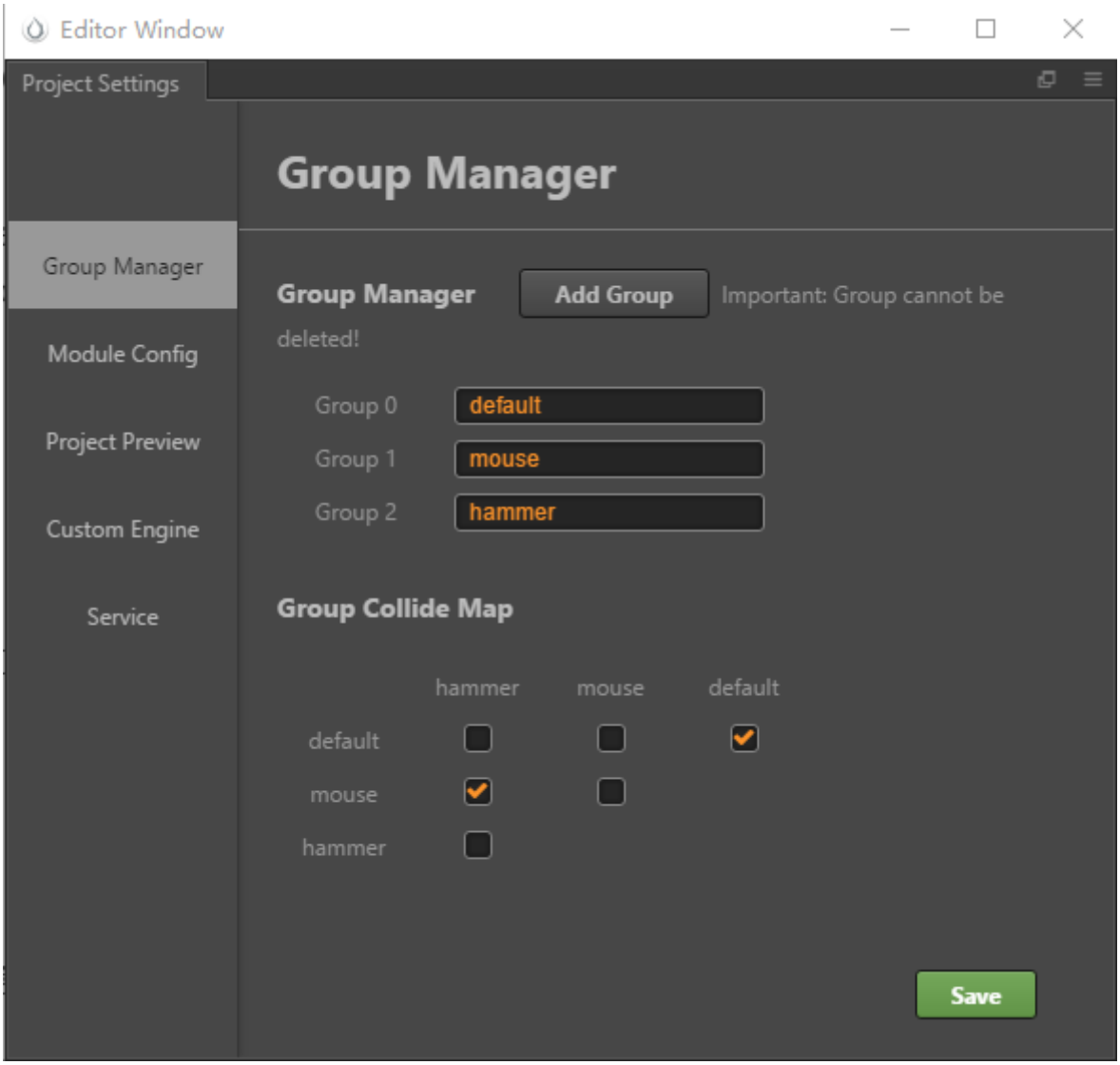
---

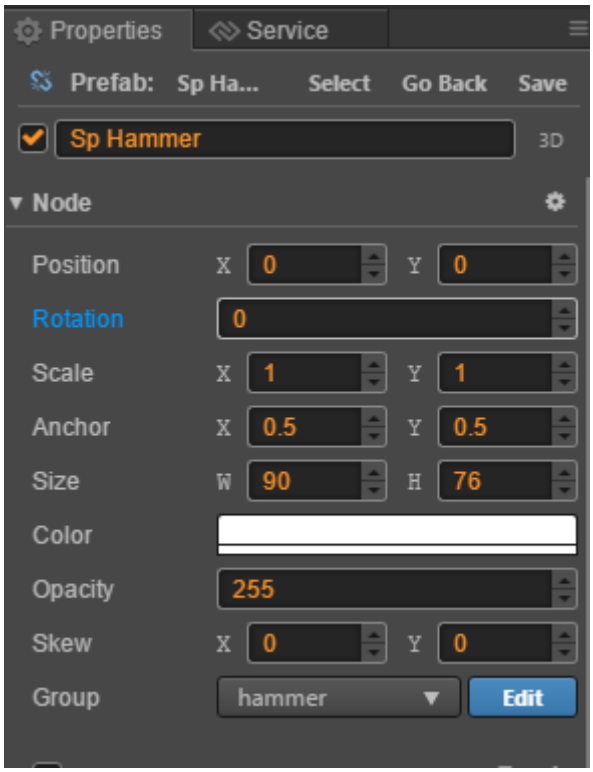
3 Use collision system

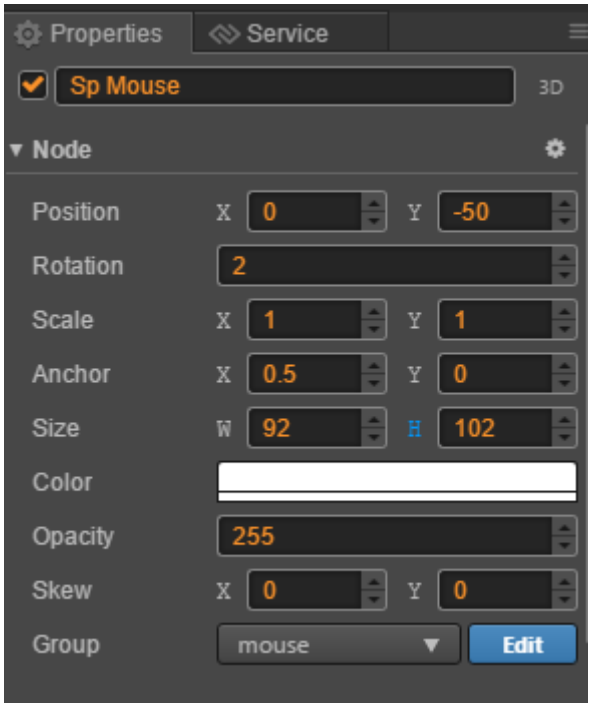
Now we need to have to have it so that if the mouse hits the hammer, it is stunned. But if you don't click on the mouse, it's fine. We can use a collision system to help us quickly implement the ability to detect whether the mouse meets the mouse.

3.1 Change packets for nodes that need collision detection and use collision components

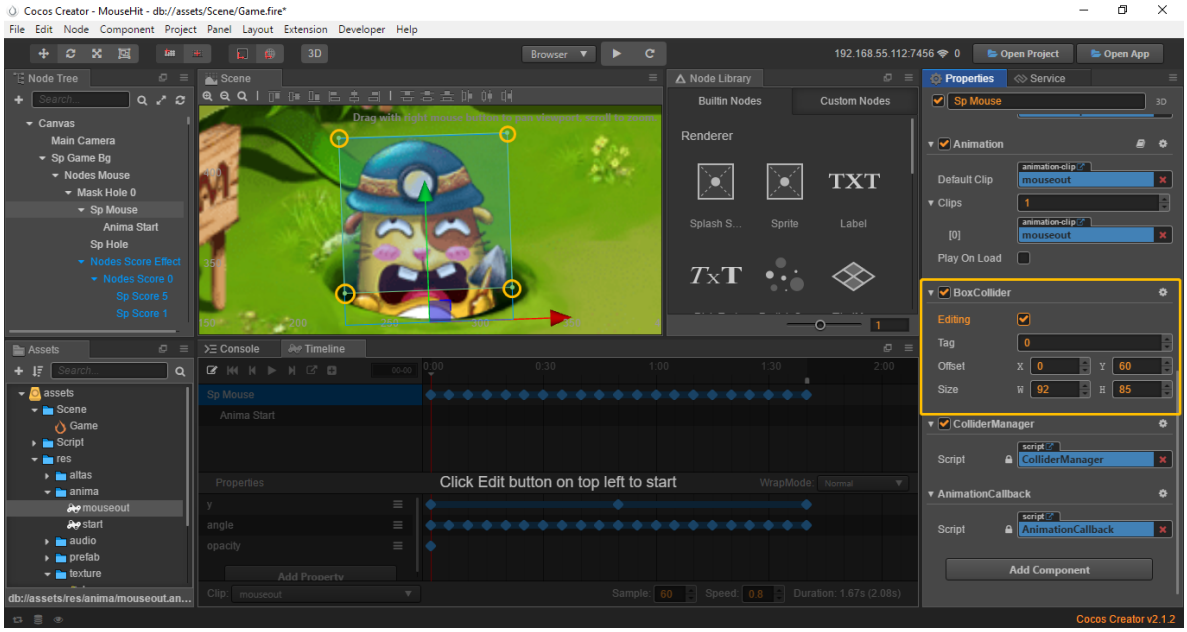
We need to use the editor to create two new groups, mouse and hammer. The menu path is Project-> Project Settings.. > Group Manager. Follow the image to help know what collides with what. We can modify the grouping of nodes so that they can adapt to collision systems. So we change the Group of the Sp Mouse node to mouse, and change the Prefab packet of Sp Hammer to hammer.


Editor Window
Project Settings
Group Manager
Group Manager
Module Config
Project Preview
Custom Engine
Service
Group Manager
Add Group
Important: Group cannot be deleted!
Group 0
default
Group 1
mouse
Group 2
hammer
Group Collide Map
hammer
mouse
default
default
mouse
hammer
Save


Properties
Service
Prefab:
Sp Ha...
Select
Go Back
Save
Sp Hammer
3D
Node
Position
X
0
Y
0
Rotation
0
Scale
X
1
Y
1
Anchor
X
0.5
Y
0.5
Size
W
90
H
76
Color
Opacity
255
Skew
X
0
Y
0
Group
hammer
Edit



We can now add BoxCollider components to the two nodes, click Edit and add it to the two componnets, and start building the collision box. You are free to drag the four-sided spot to change the size of the collision box. You can turn the mouse's BoxCollider off temporarily and on when it comes out of the rat hole.



3.2 Using scripts to control collision systems

The collision system is turned off by default in CocosCreator, so we need to start it manually. And we need to add callback functions for collision generation and collision end to the collision system. So we create a JS script called 'ColliderManager.js' and add it to the node.

```
cc.Class({
    extends: cc.Component,

    editor: {
        menu: "CustomComponent/CollisionManagement",
    },
```

```
    properties: {
    },

    onLoad () {
        var colliderManager = cc.director.getCollisionManager();
        colliderManager.enabled = true;
        // colliderManager.enabledDebugDraw = true;
    },

    start () {
        this.gameComponent =  cc.find("Canvas").getComponent("Game");
    },

    onCollisionEnter (other, self) {
        if (this.node.group === cc.game.groupList[1]) {
            this.node._isCollider = true;
            this.gameComponent._mouseNode = this.node;
        }
        else if (this.node.group === cc.game.groupList[2]) {

        }
    },

    onCollisionExit (other, self) {
        if (this.node.group === cc.game.groupList[1]) {
            this.node._isCollider = false;
        }
        else if (this.node.group === cc.game.groupList[2]) {

        }
    },
});
```

Get to the collision manager by accessing cc.director, and change its enabled property to true, to turn on the collision system. Then add the name 'onCollisionEnter','onColli to the script The two functions of sionExit' are the callback of collision generation and collision end, and control the content of the game in the callback, so that the simple collision management system is completed.

4、Perfect the logic of the game

We need to continue to supplement the logic of the game.

4.1 Let the mouse show up at random

At present, there are nine mouse holes in the game. Let's first determine how many mice there will be this time, and where these mice will appear at random, and what kind of mice will appear randomly. Once this idea has been determined, we can design the code.After the function is created, it takes effect once in the start callback of Game.js.

4.1.1 Code:

```
initMouseOutEvent () {
```

```
        if (this._mouseIndexArr.length === 0) {
            let mouseAmount = Math.ceil(Math.random() *
(this.mouseNodes.childrenCount - 1));

            if (mouseAmount === 0) {
                mouseAmount = 1;
            }

            for (let i = 0; i < mouseAmount; i++) {
                let randomNodeIndex = Math.ceil(Math.random() *
(this.mouseNodes.childrenCount - 1));
                let randomSpriteFrameIndex = Math.ceil(Math.random() *
(this.animalAtlas.getSpriteFrames().length - 1))

                if (this._mouseIndexArr.indexOf(randomNodeIndex) === -1) {
                    var mouseNode =
this.mouseNodes.children[randomNodeIndex].getChildByName("Sp Mouse");
                    this.updateMouseNodeInfo(mouseNode, randomSpriteFrameIndex);
                    mouseNode.getComponent(cc.BoxCollider).enabled = true;
                    this._mouseIndexArr.push(randomNodeIndex);
                    mouseNode.getComponent(cc.Sprite).spriteFrame =
this.animalAtlas.getSpriteFrames()[randomSpriteFrameIndex];
                    mouseNode.getComponent(cc.Animation).play();
                }
            }
        }
    },
```

4.2 We need to get the mouse to reappear.

The mouse appears once and then falls into the hole. Then, the mouse repeats this in another hole. Because we used animation components to get mice in and out of rat holes, we can add a monitor to the animation components and reappear randomly when the animation is over, that is, when the mouse enters the rat hole.

In order for mice to appear in droves, we use the _mouseIndexArr array to record the hole number of each random mouse, and when the mouse returns to the hole, the serial number is eliminated. So only when all the mice go back to the hole can the hole be used again. Let's have the mice randomly appear. First add a callback at the end of the animation named 'onAnimationFinishEvent':

4.2.1 Code:

```
onAnimationFinishEvent () {
     this._mouseIndexArr.pop();
     this.initMouseOutEvent();
},
```

Next, in 'initEventListener'Fcuntion, add animation listening:

```
for (let i = 0; i < this.mouseNodes.childrenCount; i++) {
     this.mouseNodes.children[i].getChildByName("Sp Mouse").
         getComponent(cc.Animation).on(cc.Animation.EventType.FINISHED,
     this.onAnimationFinishEvent, this);
}
```

4.3 Logic and expression of Hammer under Hammer

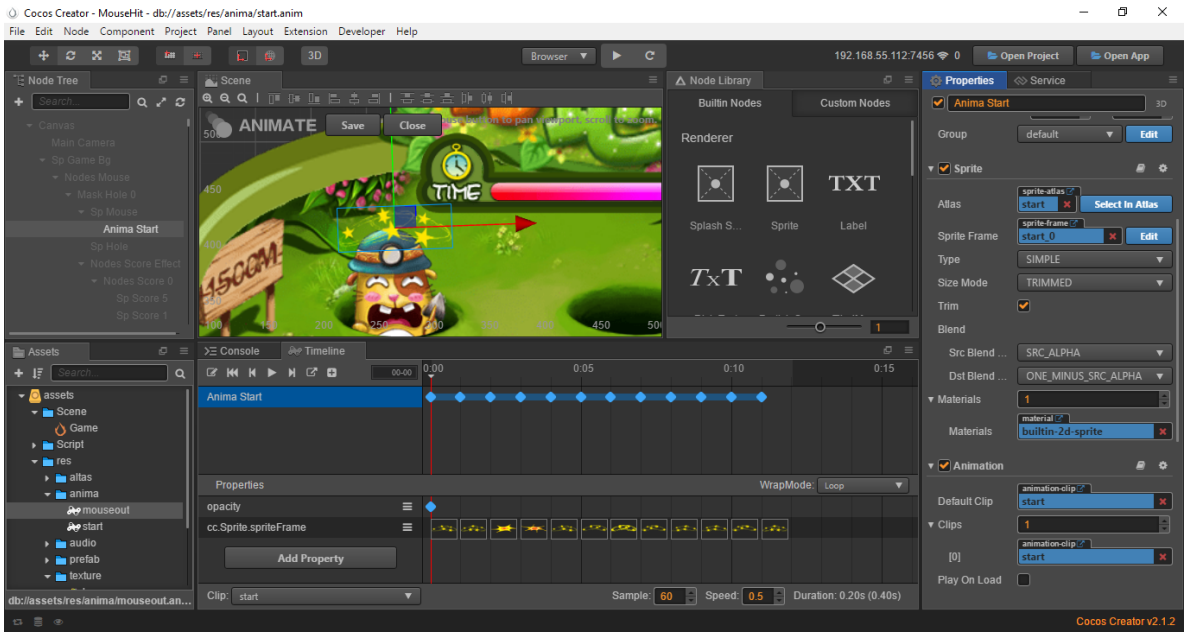
When the hammer is down, the hammer will fly up and down, so we need to change its angle attribute to achieve this effect. When the hammer is hammered, the mouse will be knocked out, should be knocked out, and should not be knocked out continuously.

4.3.1 Code：

```
onHammerClicked () {
        this.hammerNode.angle = this.hammerNode.angle === 0 ? 30 : 0;
        if (this._mouseNode && this._mouseNode._isCollider &&
this._mouseNode._isLive && cc.find("Canvas/Sp Game Bg")) {
            this._mouseNode._isLive = false;
            let oldSpriteFrameName =
this._mouseNode.getComponent(cc.Sprite).spriteFrame.name;
            let newSpriteFrameName = oldSpriteFrameName + "_death";
            this._mouseNode.getComponent(cc.Sprite).spriteFrame =
this.animalDeathAtlas.getSpriteFrame(newSpriteFrameName);
            this._mouseNode.getChildByName("Anima
Start").getComponent(cc.Animation).play();
        }
}
```

To add animation effects to mice when knocked out, we need to use animation components. First, create a new Anima Start node for Sp Mouse, add animation components to it, and create a new Animation Clip, in assets named 'start', which will play back until the mouse returns to the hole.

The start' animation needs to use a set of pictures to show the vertigo effect, and we need to control the SpriteFrame attribute in the Sprite component of the node. So we need to add multiple animation frames and set a different SpriteFrame. for each animation frame.



5、Written at the end.

Now go ahead and test if your game is running normally, if there is an issue, go back and check if you made an error with your code. If not, congratulations on getting the core game design of the Whack-A- Mole game, in the next article will supplement the game score, game sound effect, game peripheral interaction and so on.

Thanks

[New project GitHub link]

[Complete project GitHub link]