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

preface

In the last tutorial, we have completed the core play of hamster games, but a complete game not only needs a suitable game, but also needs modules such as lobby, sound effects, rules, scores and so on. Now let's continue to complete these modules to make the game more complete.

If you have trouble in our tutorial, please learn the documentation:

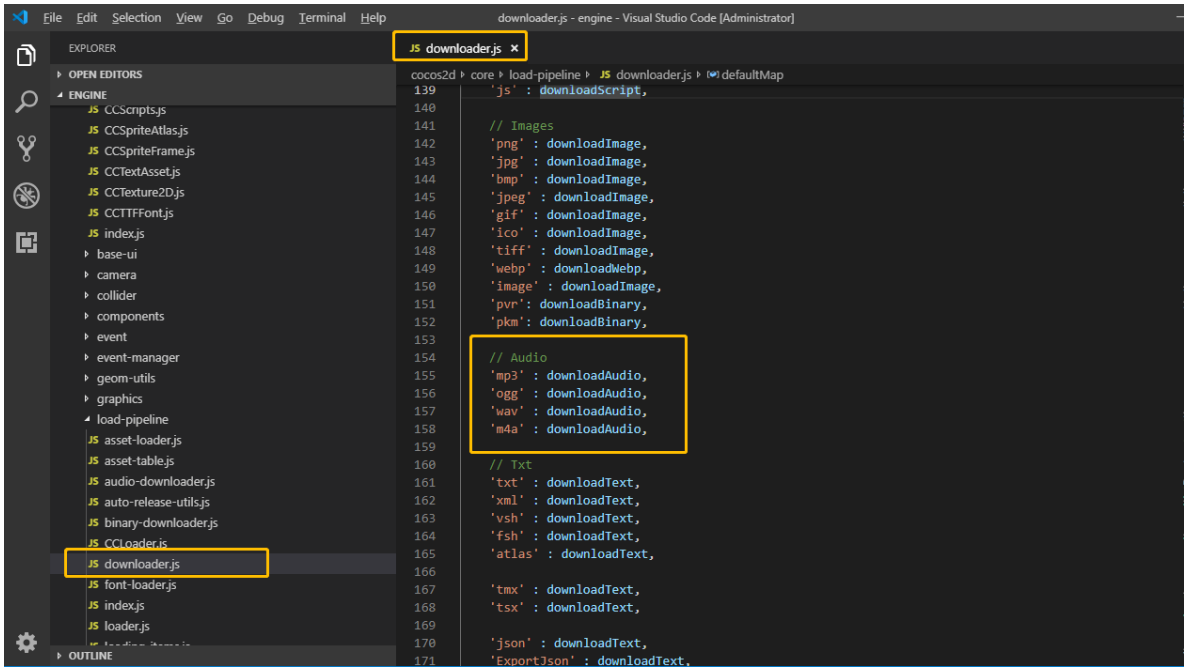
- Prepare the development environment
  - Cocos Creator 2.1.2
  - Visual Studio Code
- Document
  - Node and Component
  - Script Development Guid
  - AudioEngine
  - Scheduler
- API
  - Node
  - AudioEngine
  - Scheduler

Let's get started!

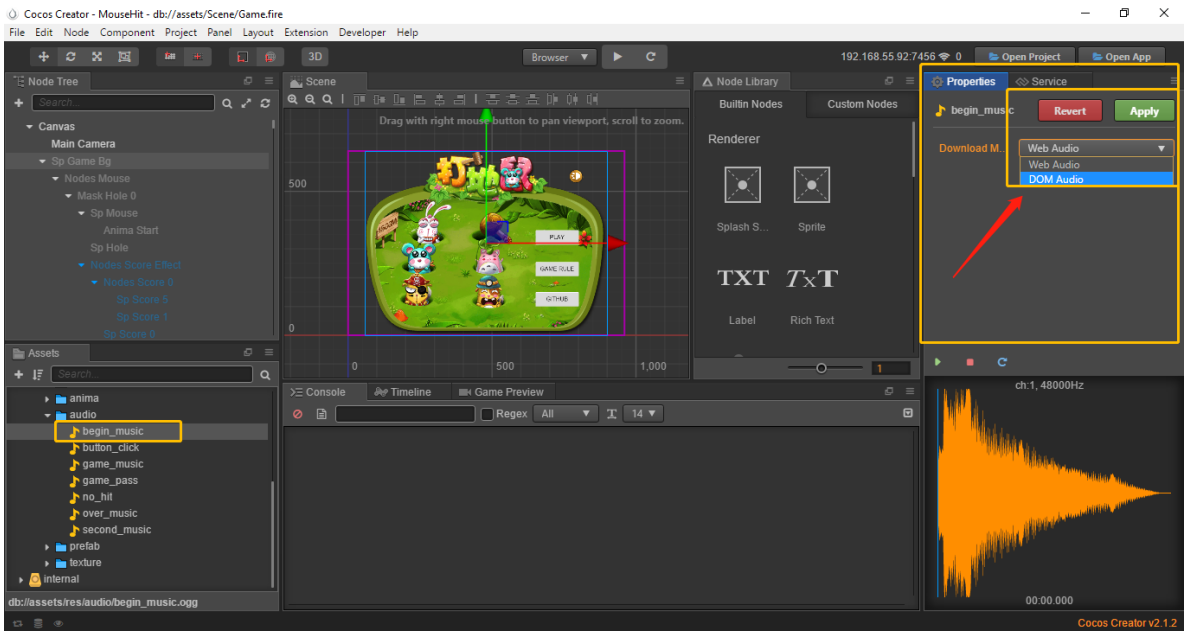
1. Understand and use AudioEngine

In CocosCreator, there are two audio playback systems, AudioEngine and AudioSource. At present, most developers are using AudioEngine because it has more features than AudioSource.

In the CocosCreator2.1.2 version, audio playback formats in 4 are currently supported, namely: `MP3`, `OGG`, `WAV`, `M4A`. This information is recorded in the engine source code downloader.js.



CoossCreator supports two audio loading modes: Web Audio and Dom Audio, and the engine defaults to the Web Audio mode. You can select an audio file in the editor and change its audio load mode in the upper right corner.



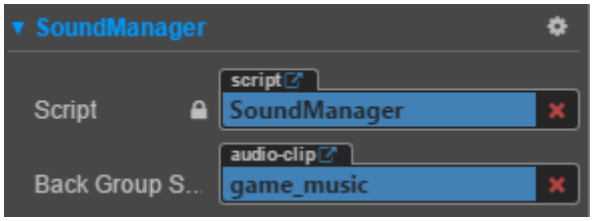
For the time being, we don't have to understand the advantages and disadvantages of these audio formats, nor do we need to know the difference between the two loading modes. At present, we usually use MP3 format and select Web Audio mode, which is enough to deal with most game items.

1.1 Use AudioEngine

We can use cc.audioEngine to access the AudioEngine module and use its various interfaces to control the audio of the game. First, create a script on the project: SoundManager.js, and add it under the Canvas node. It will be responsible for the control of the game. In CocosCreator, audio resources are packaged as AudioClip resources, so if you want to add a background audio resource property to SoundManager.js, you should do this Sample add code:

```
cc.Class({
    extends: cc.Component,
    properties: {
        backGroupSound: {
            default: null,
            type: cc.AudioClip
        }
    }
});
```

After saving the code, back to the editor, we will see a property box of type AudioClip, and there is nothing in the box. We drag the game background music "game_music.mp3" file into the property box called "backGroupSound" to complete the addition of an audio resource.



After consulting the AudioEngine's API, we understand that you can use the play method to play the specified AudioClip resource, using the following example:

```
var audioId = cc.audioEngine.play(audioClip, loop, volume);
```

In this demonstration code, audioId is the ID, of the audio playback instance returned by the cc.audioEngine.play function, the ID changes once, so the label of each audio instance. AudioClip is the specified audio resource, only resources that meet the above four audio formats can be played. Loop refers to whether the loop is cyclic. Volume is the volume of the playback.

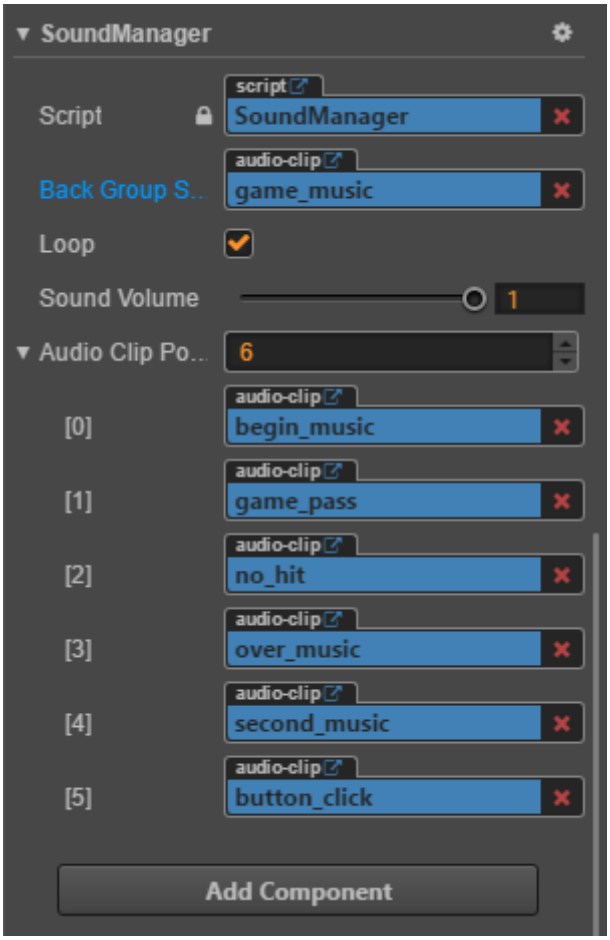
After you know these, we can formally use the AudioEngine.

Loop and volume should be able to be adjusted at any time, so we no longer have it hard-coded into the program. We add it to properties, so that not only can it be adjusted in the editor, even if the game is in code control.

```
properties: {
        backGroupSound: {
            default: null,
            type: cc.AudioClip
        },

        loop: true,

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

        audioClipPool: {
            default: [],
            type: cc.AudioClip
        },
```

```
        _isPlaying: false,
        _audioId: null,
        _EffectId: null,
    },
```

The property inspector interface after the addition is complete (looks good):



If you don't know much about the property inspector parameters, you can refer to this document:Property inspector parameters。 In the code, the audiClipPool parameter is used to record all the AudioClip resources that need to be used in the game, which is convenient for us to play any of the audio at any time. `_isPlaying` is used to record if the current background music is playing. In that audio engine, each audio play instance will also record its own play state, but it is not the play state of the global music, so we still need a tag for representing the global music state. `_audiId` and `_EffectId` are background music and The ID of the audio play instance for the game sound effect.

Next, we should have the music be played.

```
// SoundManager.js code
playBackGroupSound () {
        cc.audioEngine.stopAll();
        this._audioId = cc.audioEngine.play(this.backGroupSound, this.loop,
this.soundVolume);
}
```

Because AudioEngine does not pause the current audio when playing audio resources, the playBackGroupSound method pauses all music once before playing background music to ensure that there are no strange sounds. Then the play method is called to play the background music according to the resource, loop mode and volume size we specify. Let's try this at Game.js:

```
this.node.getComponent("SoundManager").playBackGroupSound();
```

After execution, friends who can't play sound don't have to worry, because the Google browser kernel stipulates that sound can only be played after effective interface interaction, that is to say, we can create a button to start the game, and when we click on the button to execute the playback code, we can play audio normally. We can find a listening callback for the click event in the initEventListener method and add this code to test if the following is correct:

```
this.node.on(cc.Node.EventType.TOUCH_START, (event)=>{
    //audioEngine play function test
    this.node.getComponent("SoundManager").playBackGroupSound();
},this);
```

In addition, we have designed a way to play different audio by passing different instructions:

```
playEffectSound (command, loop) {
        if (loop === null && loop === undefined) {
            var loop = loop;
        }
        if (command !== null || command !== undefined) {
            switch (command) {
                case "begin":
                case "score":
                    this._EffectId =
cc.audioEngine.playEffect(this.audioClipPool[0], loop);
                    break;
                case "pass":
                    this._EffectId =
cc.audioEngine.playEffect(this.audioClipPool[1], loop);
                    break;
                case "hit":
                    this._EffectId =
cc.audioEngine.playEffect(this.audioClipPool[2], loop);
                    break;
                case "lose":
                    this._EffectId =
cc.audioEngine.playEffect(this.audioClipPool[3], loop);
                    break;
                case "second":
                    this._EffectId =
cc.audioEngine.playEffect(this.audioClipPool[4], loop);
                    break;
                case "click":
                    this._EffectId =
cc.audioEngine.playEffect(this.audioClipPool[5], loop);
                    break;
                default:
                    console.error("Command is invalid");
            }
        }
    },
```

For example， we can add sound effects to a hammer hit event. We can add the following code in the onHammerClicked method, and play 'no_hit.mp3' by default:

```
this.node.getComponent("SoundManager").playEffectSound("hit");
```

And when it meets the conditions in the if statement, another sound effect 'second_music.mp3' is played:

```
this.node.getComponent("SoundManager").playEffectSound("score");
```

1.2 Supplementary audio control method:

In the course of the game, we often need to pause or resume the background music and adjust the volume size. So we need to supplement these two methods. First of all, to pause the music, we can use the stopAll method directly to pause all the audio in the game, but he will also pause the sound in the game. In addition, we can use stop to pause the audio that specifies the audio play instance ID. In the current project, we choose to use stopAll directly because our sound effect is some simple sound effect, and it won't affect the game.

```
// SoundManager.js code
stopAll () {
        cc.audioEngine.stopAll();
        this._audioId = null;
    }
```

```
// test in Game.js code
this.node.getComponent("SoundManager").stopAll();
```

In addition, we can use the setVolume method to control the background music volume:

```
// SoundManager.js code
setVolume(value) {
        if (!isNaN(value)) {
            this.soundVolume = value;
            cc.audioEngine.setVolume(value);
        }
    },
```

Let's test the event callbacks on the screen:

```
///test in Game.js code
this.node.on(cc.Node.EventType.TOUCH_START, (event)=>{
    //audioEngine setVolume function test
    this.node.getComponent("SoundManager").setVolume(0);
    this.node.getComponent("SoundManager").playBackGroupSound();
},this);
```

In this way, after the code is executed, the background music can not make a sound.

In addition, we have added pause and recovery methods, as follows:

```
// SoundManager.js code
pauseMusic () {
    cc.audioEngine.pauseAll();
},

resumeMusic () {
    cc.audioEngine.resumeAll();
},
```

In summary, it is a simple game audio control module script.

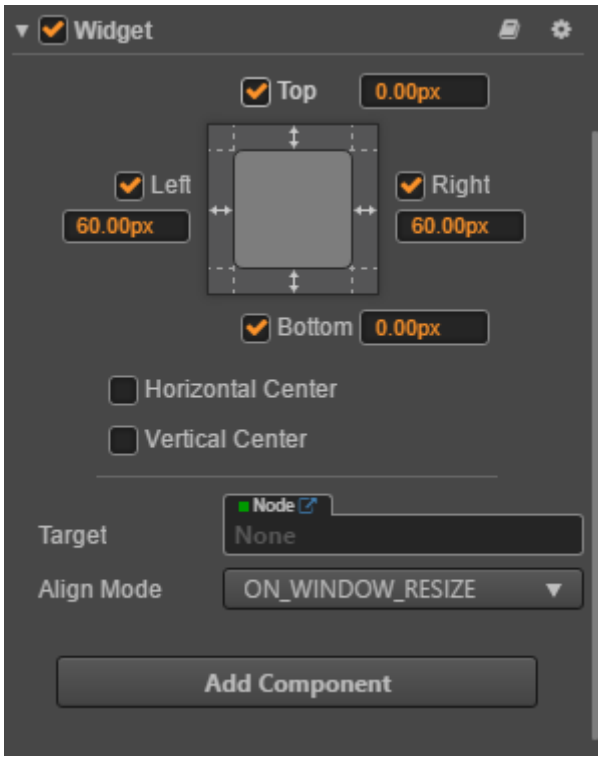
---

2 Make the interface of the game hall

Each game needs a game hall interface. As the first interface of the game, the game hall interface can not only guide users, introduce rules, set the game, but also add a variety of game activities to increase the vitality of the game and bring benefits to the game developers. So a good game hall is very important.

2.1 Add pictures and UI components

Find the "hall.png" picture under "assets/res/texture", which is the background picture resource of the hall interface. We drag it into the Game scenario and set the position to: (0, 0), then add the widget component and adjust the properties to look like this:



The widget component can help us adapt to different screen resolutions, suggesting that all background pictures add a widget component.

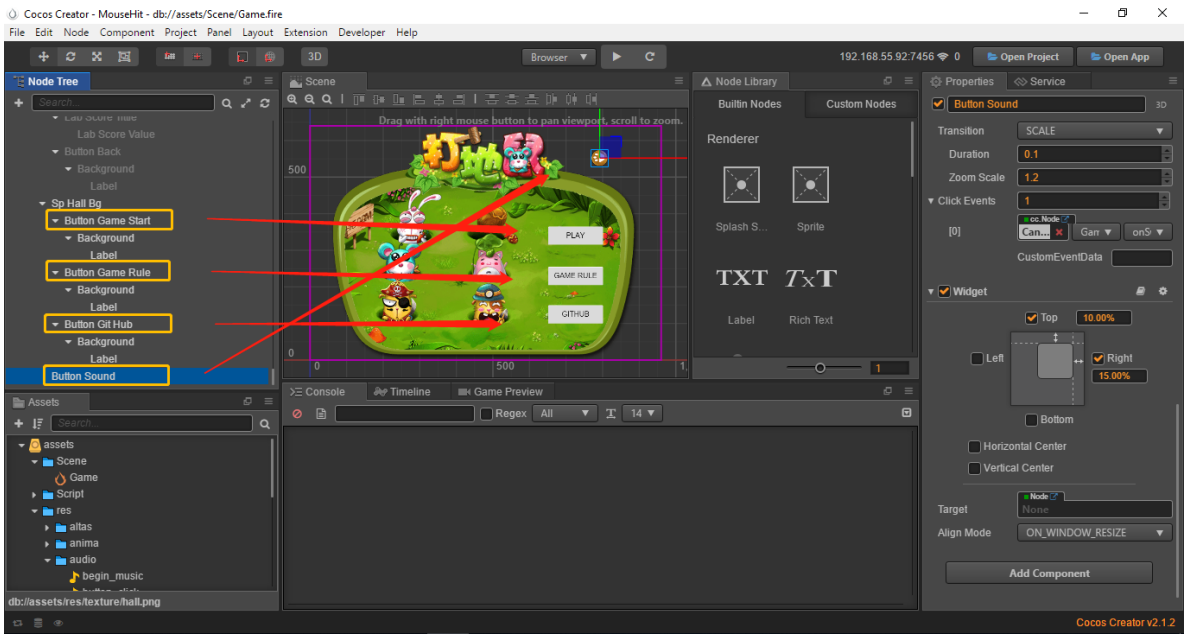
Then, we add four child nodes with Button components to the lobby nodes, namely:

“Button Game Start”：Game start button；

“Button Game Rule”：Game rule button；

“Button GitHub“：Game github button；

”Button Sound“：Audio switch button。



Also add widget components to these buttons so that they can also adapt to different screen resolutions. Please refer to the layout on the complete project for the location layout of these buttons. Of course, you can also play at will.

In the scene design of this game, we did not use the second scene to show the game login, because our game content is relatively simple. If you make more complex projects, you can use a dual-scene architecture, that is, a scene is responsible for game loading and login, and a scene is responsible for the lobby and core play. Or a better way.

Because the interface of the hall and the interface of the core play method are overlapped, the problem of clicking the penetration can occur, so that the core play method interface can be shielded, so that the option of the upper left corner of the attribute checker of the "Sp Game Bg" node is removed. The effect of doing this is equivalent to node.active = false; you can then click the lobby interface as you do.

2.2 Add a click callback for each button.

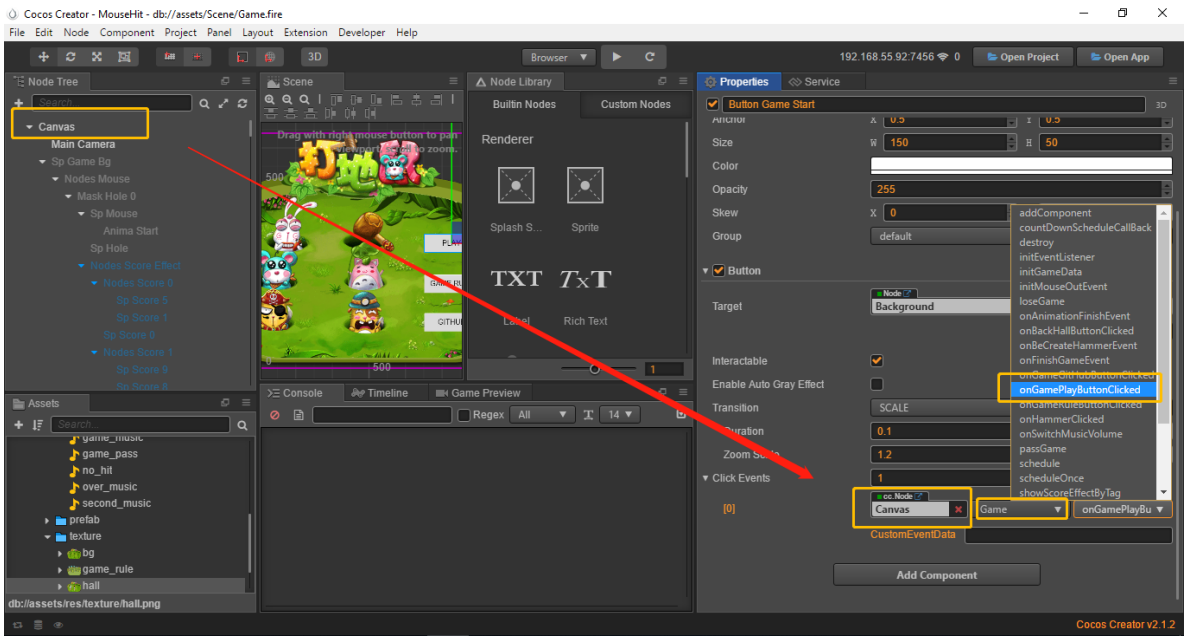
Modify the label string for buttons with Label components for child nodes and modify their styles accordingly. After that, we began to add click-back functions for each button.

2.2.1 Add a click callback function to the "Button Game Start" button

Add a callback function onGamePlayButtonCicked in the Game.js code, and the code is as follows:

```
onGamePlayButtonClicked() {
    //Play background music
        this.node.getComponent("SoundManager").playBackGroupSound();
    //Control the operation of the game
    cc.find("Canvas/Sp Hall Bg").active = false;
    cc.find("Canvas/Sp Game Bg").active = true;
}
```

Then go back to the editor and add a callback function to the button so that the button will execute it after the button is clicked, and the following three callback functions will be added in a similar way, and I will no longer demonstrate:



2.2.2 Add a click callback function to the "Button Game Rule" button

Add the callback function onGameRuleButtonClicked, to the Game.js code with the following code:

```
onGameRuleButtonClicked () {
        //play the general click sound effect
        this.node.getComponent("SoundManager").playEffectSound("click", false);
        if (!this.gameRuleNode) {
            //Create a node that displays a picture of the rules of the game
            this.gameRuleNode = new cc.Node();
            this.gameRuleNode.addComponent(cc.Sprite).spriteFrame =
this.gameRule;
            this.node.addChild(this.gameRuleNode);
        }
        //Adjust the opacity value of the game rule node
        this.gameRuleNode.opacity = 255;
    },
```

2.2.3 Add a click callback function to the "Buton GitHub" button

Add the callback function onGameGitHubButtonClicked, to the Game.js code with the following code:

```
onGameGitHubButtonClicked () {
        //Play the general click sound effect
        this.node.getComponent("SoundManager").playEffectSound("click", false);
        //To determine whether the current environment is a browser environment,
execute only in a browser environment
        if (cc.sys.isBrowser) {
            //Open the game GitHub link in a new window
            cc.sys.openURL(this.gameGitHubUrl);
        }
    },
```

2.2.4 Add a click callback function to the "Button Sound" button

Add the callback function onSwitchMusicVolume, to the Game.js code with the following code:

```
onSwitchMusicVolume (event) {
        //Play the general click sound effect
        this.node.getComponent("SoundManager").playEffectSound("click");
        //changing the audio play state in the audio management system
        this.node.getComponent("SoundManager")._isPlaying =
!this.node.getComponent("SoundManager")._isPlaying;
        //Changes the display picture of the button according to the current
audio playback state and controls the playback of the background audio
        if (this.node.getComponent("SoundManager")._isPlaying) {
            event.target.getComponent(cc.Sprite).spriteFrame =
this.icon.getSpriteFrame("sound_close");
            this.node.getComponent("SoundManager").stopAll();
        }
        else {
            event.target.getComponent(cc.Sprite).spriteFrame =
this.icon.getSpriteFrame("sound_open");
            this.node.getComponent("SoundManager").playBackGroupSound();
        }
    },
```

In summary, we have completed the hall UI design and related button click callback.

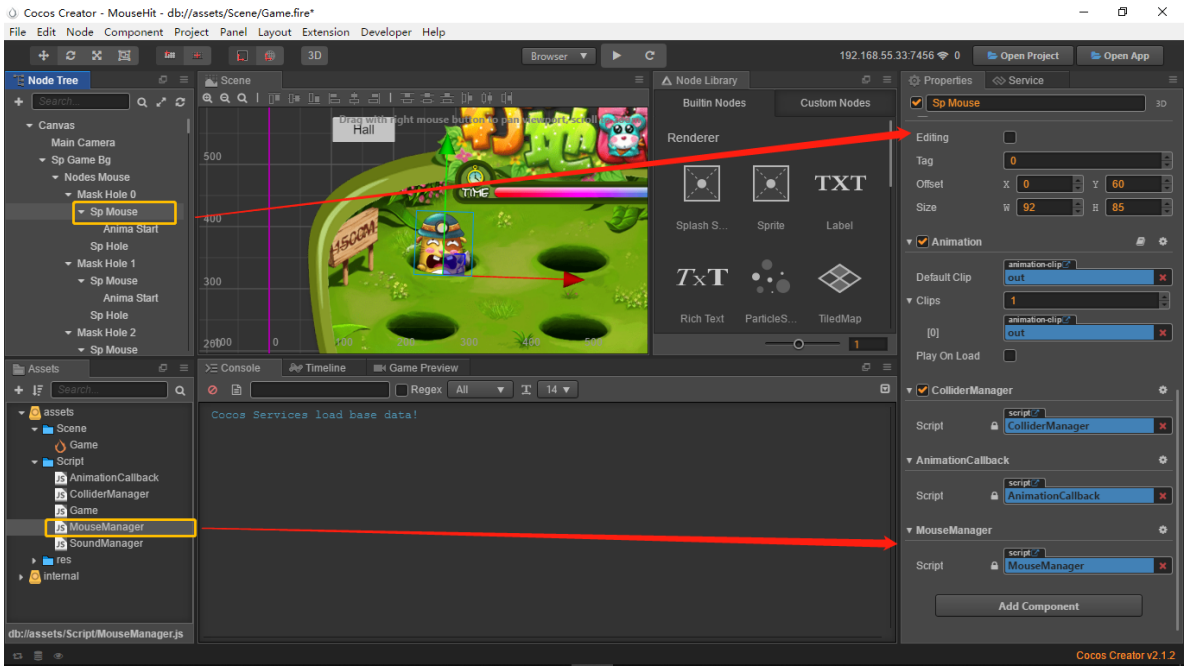
---

3 Increase the scoring mechanism of the game

Game score is the best embodiment of the results of the game, he can best stimulate the desire of players for the game. Different games have different scoring methods, and the main scoring characteristics of hamster games are that the scoring methods of each mouse are different. Next, we will implement this scoring mechanism.

3.1 Configure data for each mouse

First, we create a new JavaScript script called "MouseManager" and add it to each Sp Mouse node.



code detial

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

    properties: {
        _isLive: false,  //代表老鼠是否被击晕
        _tag: null,  //节点标签
    },
});
```

Each mouse has a different name and a different way of calculating points. We can set their own data for each mouse when the game is initialized. You can refer to the following approach, we need to create a initGameData function in Game.js, which will be executed in the start callback. Friends who do not know about start life cycle callbacks can view this document: Life cycle。

```
initGameData () {
        //Create a set of mouse data and set different mouse names and
subdividing functions for each element
        this._mouseDataTable = [
            {
                mouseName: "harmful_mouse_0",
                scoreUpdateFunc: function () {
                    this._score += 100;
                }
            },
            {
                mouseName: "harmful_mouse_1",
                scoreUpdateFunc: function () {
                    this._score += 500;
                }
            },
            {
                mouseName: "kind_mouse_0",
                scoreUpdateFunc: function () {
                    if (this._score === 0) {
                        this._score += 200;
```

```
                    }
                    else {
                        this._score = Math.floor(this._score * 1.2);
                    }
                }
            },
            {
                mouseName: "kind_mouse_1",
                scoreUpdateFunc: function () {
                    this._score -= 100;
                }
            },
            {
                mouseName: "rabbit_0",
                scoreUpdateFunc: function () {
                    this._score = Math.floor(this._score / 2);
                }
            }
        ];
    },
```

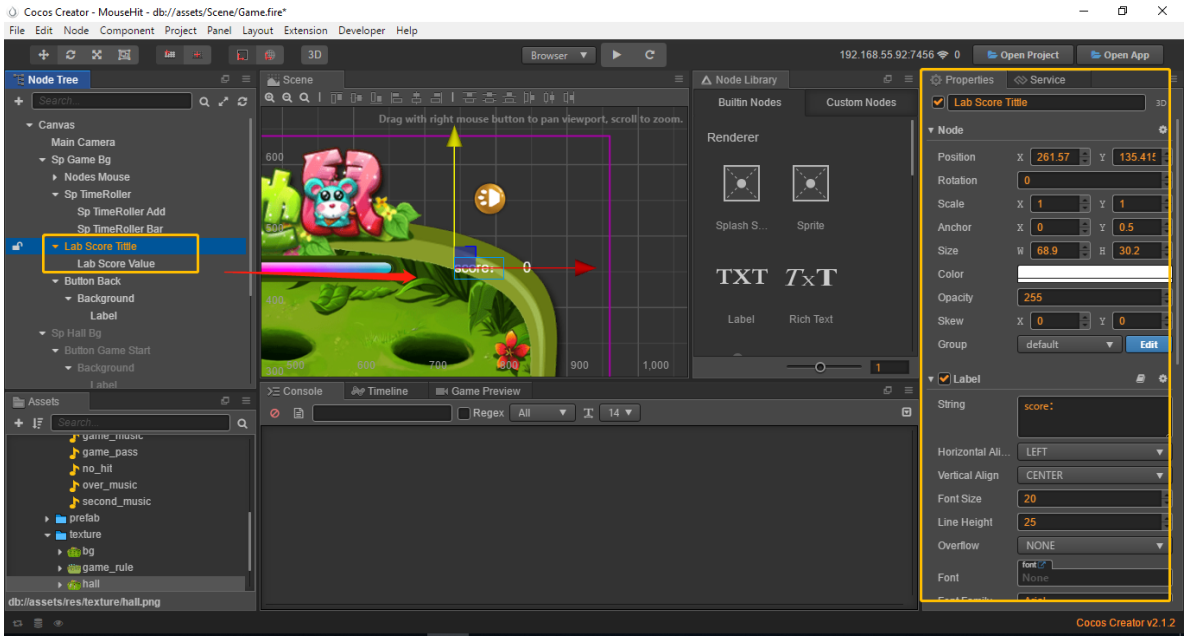
We need to know exactly what each random mouse does, so we add a updateMouseNodeInfo function to do it. It is executed in the initMouseOutEvent function, and when the mouse type is determined, the ordinal number of the current type is recorded on the mouse node.

```
updateMouseNodeInfo(mouseNode, tag) {
        //Set the current mouse node to live
        mouseNode._isLive = true;
        //Bind the corresponding score function according to the received mouse
type ordinal number
        mouseNode._scoreUpdateFunc =
this._mouseDataTable[tag].scoreUpdateFunc.bind(this);
        //Add the mouse type serial number as a label to the mouse node
        mouseNode.getComponent("MouseManager")._tag = tag;
    },
```

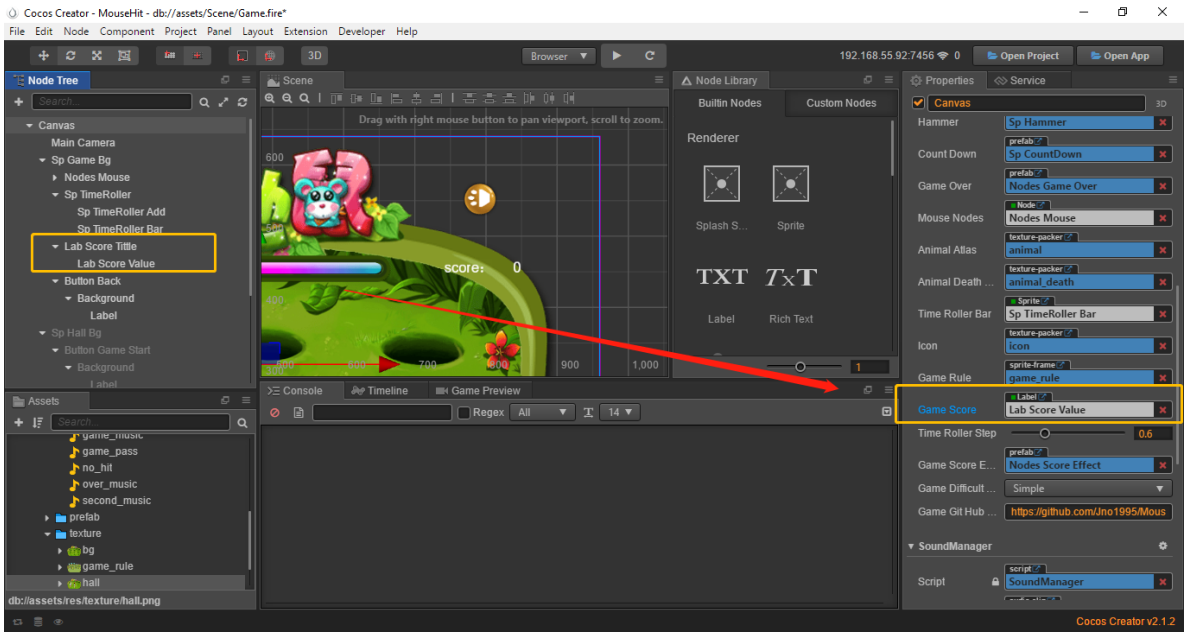
```
initMouseOutEvent () {
        if (this._mouseIndexArr.length === 0) {
            let mouseAmount = Math.ceil(Math.random() *
(this.mouseNodes.childrenCount - 1));

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

            for (let i = 0; i < 5; 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");
                    //Perform the mouse node data refresh function
                    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();
                }
            }
        }
    },
```

We also need a label that shows the score, so we add two nodes with the Label component under the "Sp Game Bg" node, their position can reference the design of the full project, but you can also play it freely.



Then drag the Lab Score Tittle node into the ”gameScore“ property box of Game.js.



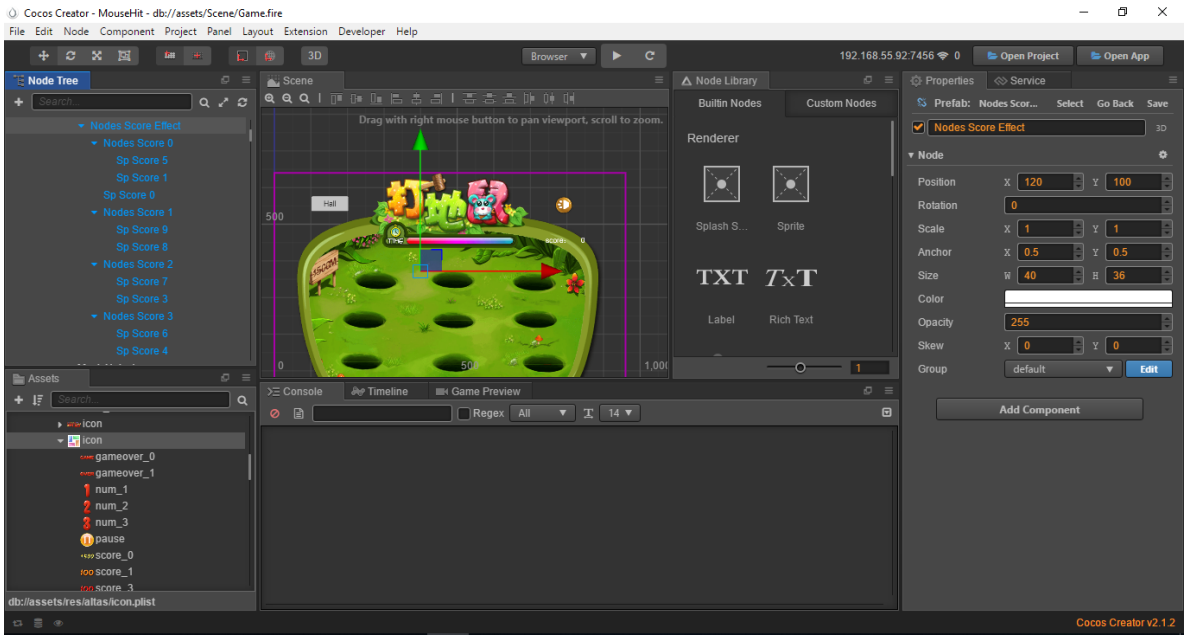
Every time we hit a mouse effectively, we get a different score. For example, if you knock out harmful mice, you will add points, and if you knock out harmless mice, you will lose points. This design is very interesting. Of course, we can't allow scores below zero. So we added these lines to the onHammerClicked function:

```
onHammerClicked () {
        this.hammerNode.angle = this.hammerNode.angle === 0 ? 30 : 0;
        this.node.getComponent("SoundManager").playEffectSound("hit");
        if (this._mouseNode && this._mouseNode._isCollider &&
this._mouseNode._isLive && cc.find("Canvas/Sp Game Bg")) {
            //Play score sound effect
            this.node.getComponent("SoundManager").playEffectSound("score");
            //Perform the mouse score function to refresh the current game score
            this._mouseNode._scoreUpdateFunc();
            //Determine if the current score is below 0 and reset to 0 if less
than zero
            this._score = this._score < 0 ? 0 : this._score;
            //Let the game score text display the current score
            this.gameScore.string = this._score;
            this._mouseNode.getComponent("MouseManager")._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();
        }
    },
```

Save the code, run the game, and you can see that when we knock out the mouse, the score in the upper right corner begins to change.

3.2 Add scores to display special effects

In order to enrich the performance of the game, we can have a special effect of scoring every time we knock out a mouse score. First, we add a node Nodes Score Effect, and then add five sub-nodes to it to represent five kinds of scoring special effects. The resources they use are pictures from the atlas icon. By default, we set the opacity of all five subnodes to 0. 0. Then drag the node into the "assets/res/prefab" to make Prefab, and then make the pref Ab is dragged into each child node of the "Canvas/Sp Game Bg/Nodes Mouse".



Now every random mouse carries a tag that represents its type _ tag, and we can use this tag to specify the score effect that the mouse hole should present in the current mouse. We can add a showScoreEffectByTag function to do this. We can see the effect by executing this function in the onHammerClicked function.

```
showScoreEffectByTag (node, scoreEffectNode) {
        //the node receives each of the different mouse nodes, the
scrotEffectNode receives the score special-effect parent node,
        for (let i = 0; i < scoreEffectNode.childrenCount; i++) {
            //Traversal score special-effect parent nodes, only the nodes that
match the order of nodes and the order of the mouse type are displayed, and the
rest of the nodes are hidden.
            scoreEffectNode.children[i].opacity =
node.getComponent("MouseManager")._tag === i ? 255 : 0;
            //to perform a fade-out action
            scoreEffectNode.children[i].runAction(cc.fadeOut(1));
        }
    },
```

```
onHammerClicked () {
        this.hammerNode.angle = this.hammerNode.angle === 0 ? 30 : 0;
        this.node.getComponent("SoundManager").playEffectSound("hit");
        if (this._mouseNode && this._mouseNode._isCollider &&
this._mouseNode._isLive && cc.find("Canvas/Sp Game Bg")) {
            this.node.getComponent("SoundManager").playEffectSound("score");
            this._mouseNode._scoreUpdateFunc();
            // New add : Show score special effects
            this.showScoreEffectByTag(this._mouseNode,
this._mouseNode.parent.getChildByName("Nodes Score Effect"));
            this._score = this._score < 0 ? 0 : this._score;
            this.gameScore.string = this._score;
            this._mouseNode.getComponent("MouseManager")._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();
        }
},
```

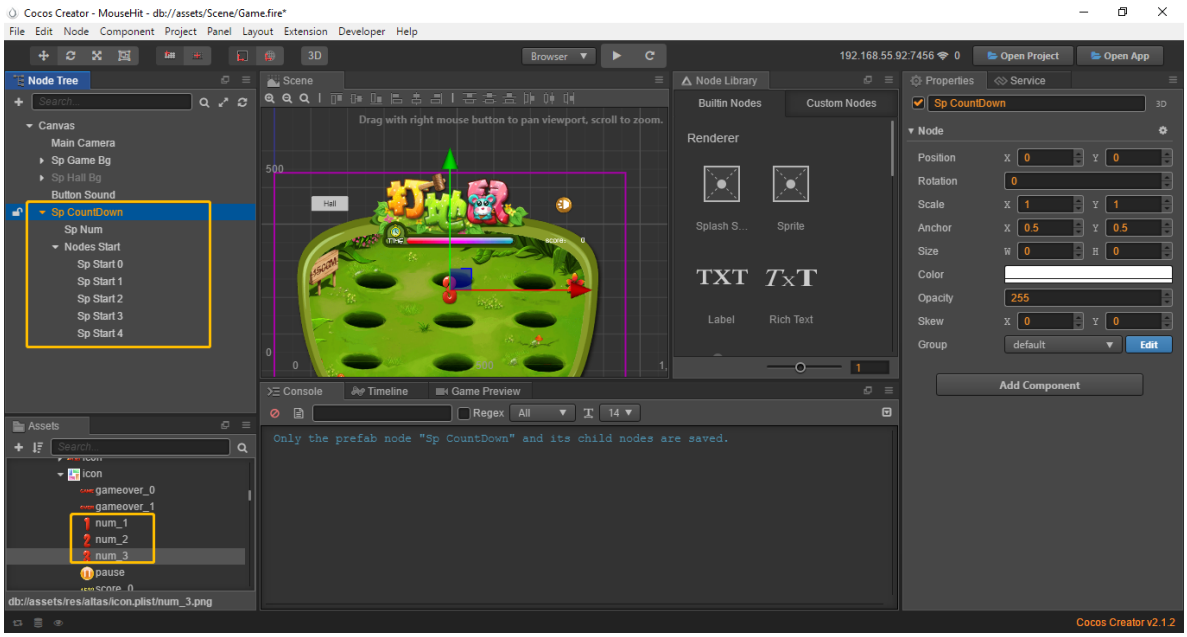
---

4. Join the game countdown and settlement mechanism

The current game can be played without time limit, which is obviously unreasonable, which will also make the game can not stop, can not settle the score.

4.1 Join the game to start the countdown

We add a function startTimeRoller to do this, and the execution of this function is triggered by the "Button Game Play" button on the lobby interface. Now we can reverse the comment code in the callback function added by this button. Let's first add a node to display the countdown. We create a node named Sp CountDown, under Canvas to add pictures in the "icon" diagram set related to the countdown display.



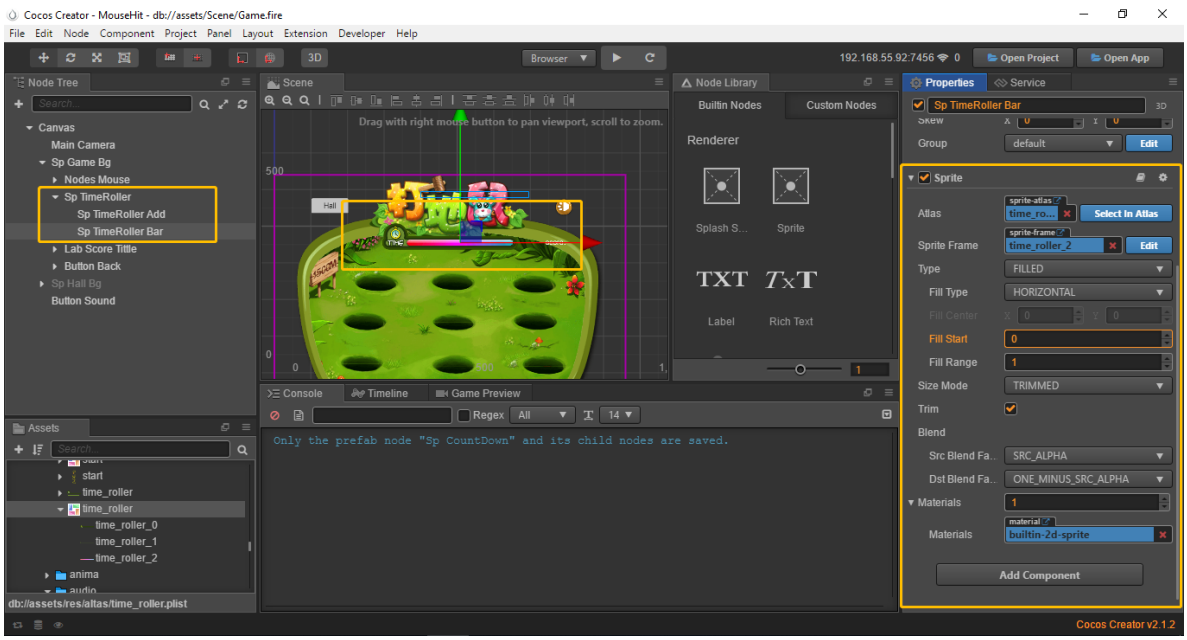
After dragging the Sp CountDown node to prefab under assets/res/prefab, drag it into the “countDown” property box of Game.js. We also need to use the CocosCreator packaged timer Scheduler to help us with the countdown, so we can add a function called startTimeRoller to do this.

```
startTimeRoller () {
        //Default value
        var times = 3;
        //start-up timer
        this.schedule(()=> {
            //When the count value is not 0, the picture of the countdown node
is changed
            if (times !== 0) {
                if (!this.countDownNode) {
                    //Instantiate the preform of the countdown node
                    this.countDownNode = cc.instantiate(this.countDown);
                    //Add a countdown node to the current component's node
                this.node.addChild(this.countDownNode);
                }
                //Display countdown node
                this.countDownNode.getChildByName("Sp Num").opacity = 255;
                //Hide the game start parent node in the countdown node
                this.countDownNode.getChildByName("Nodes Start").opacity = 0;
                //Toggles the timing picture according to the current count
value
                let spriteFrameName = "num_" + times;
                this.countDownNode.getChildByName("Sp
Num").getComponent(cc.Sprite).spriteFrame =
this.icon.getSpriteFrame(spriteFrameName);
                //Play the countdown sound
                this.node.getComponent("SoundManager").playEffectSound("second",
false);
            }
```

```
            //When the count is 0, that is, the countdown ends, the game start
logic is executed
            else {
                //Hide the timing sub-node in the countdown node
                this.countDownNode.getChildByName("Sp Num").opacity = 0;
                //Display the game start child node in the countdown node
                this.countDownNode.getChildByName("Nodes Start").opacity = 255;
                //Play the game and start the sound effect.
                this.node.getComponent("SoundManager").playEffectSound("begin",
false);
                //The countdown node performs a concealment action
                this.countDownNode.runAction(cc.fadeOut(1));
                //Start the game.
                this.startGame();
            }
            //Count value self-subtraction
            times--;
        }, 1, 3);
    },
```

4.2 Join the countdown to the end of the game:

In the current startTimeRoller function, we have completed the countdown to the start of the game, in which we can count the countdown to the end of the game. Let's first add a wizard Sp TimeRoller. For the countdown to the end of the game. The resources it uses come from the atlas "time_roller".



SpTimeRoller's child node SpTimeRollerBar uses FILLED rendering mode, which causes the wizard to determine the area of the display picture according to the current fillStart to fillRange ratio. We choose fillType as horizontal, set fillStart to 0, and fillRange to 1, so we can well display the game end countdown. We'll put SpTimeRollerBar. Add to the Game.js' s "timeRolleBar" property box, and then start adding the startTimeRoller code.

```
startTimeRoller () {
        var times = 3;
        //Reset the initial fill value of the countdown wizard at the end of the
game
        this.timeRollerBar.fillStart = 0;
```

```
        this.schedule(()=> {
            if (times !== 0) {
                if (!this.countDownNode) {
                    this.countDownNode = cc.instantiate(this.countDown);
                    this.node.addChild(this.countDownNode);
                }
                this.countDownNode.getChildByName("Sp Num").opacity = 255;
                this.countDownNode.getChildByName("Nodes Start").opacity = 0;
                let spriteFrameName = "num_" + times;
                this.countDownNode.getChildByName("Sp
Num").getComponent(cc.Sprite).spriteFrame =
this.icon.getSpriteFrame(spriteFrameName);
                this.node.getComponent("SoundManager").playEffectSound("second",
false);
            }
            else {
                this.countDownNode.getChildByName("Sp Num").opacity = 0;
                this.countDownNode.getChildByName("Nodes Start").opacity = 255;
                this.node.getComponent("SoundManager").playEffectSound("begin",
false);
                this.countDownNode.runAction(cc.fadeOut(1));
                //Start the countdown to the end of the game,
this.timeRollerStep can control the countdown frequency
            this.schedule(this.countDownScheduleCallBack, this.timeRollerStep);
                this.startGame();
            }
            times--;
        }, 1, 3);
    },
```

Add execution callback function counter to the end of the game countDownScheduleCallBack

```
countDownScheduleCallBack () {
        //The countdown wizard fills 1%each time a callback is performed.
        this.timeRollerBar.fillStart += 0.01;
        //The end of the game when fully populated
        if (this.timeRollerBar.fillStart === this.timeRollerBar.fillRange) {
            //Turn off the countdown timer for the end of the game
            this.unschedule(this.countDownScheduleCallBack);
            //Turn off game listening events.
            this.unEventListener();
        }
},
```

4.3 Settlement game score

When the game is finished, it is possible to determine whether the current score satisfies the score of the current game difficulty request. If you can't meet the requirements, it's a loss. If you meet the requirements, it's a win. This logical code can be added to the counter DownScheduler CallBack function.

```
countDownScheduleCallBack () {
        this.timeRollerBar.fillStart += 0.01;
        if (this.timeRollerBar.fillStart === this.timeRollerBar.fillRange) {
            this.unschedule(this.countDownScheduleCallBack);
            this.unEventListener();
```

```
            //judging whether the score exceeds the currently set game
difficulty score, exceeding the score, executing the passGame function,
            if (this._score > this.gameDifficultScore) {
                this.passGame();
            }
            //No more than, generate the game failure interface
            else {
                if (!this.gameOverNode) {
                    this.gameOverNode = cc.instantiate(this.gameOver);
                    this.node.addChild(this.gameOverNode);
                }
                //Display game failure interface
                this.gameOverNode.opacity = 255;
                //the game failure interface performs a fade-out action,
                this.gameOverNode.runAction(cc.fadeOut(1.5));
                //Execute loseGame function
                this.loseGame();
            }
            //Execute the game completion function
            this.onFinishGameEvent();
        }
    },
```

```
passGame() {
    //Play the game through the sound effect
    this.node.getComponent("SoundManager").playEffectSound("pass", false);
},
```

```
loseGame () {
    //Play game failure sound effect
    this.node.getComponent("SoundManager").playEffectSound("lose", false);
},
```

```
onFinishGameEvent () {
    //Stop all mouse animations
    for (let i = 0; i < this.mouseNodes.childrenCount; i++) {
        this.mouseNodes.children[i].getChildByName("Sp
Mouse").getComponent(cc.Animation).stop();
    }
    //Restart the game in two seconds
    setTimeout(()=>{
        cc.director.loadScene("Game");
    },2000);
}
```

So far, we have finished playing hamster games. If you have any questions, you can ask your questions in issues or go to the forum to ask questions. You can let me know at @337031709 on the forum.

Thank you

新项目的GitHub链接

完整项目的GitHub链接