十米体感游戏设计大赛答辩

姚**

湖南大學
HUNAN UNIVERSITY

湖南大学
HUNAN UNIVERSITY

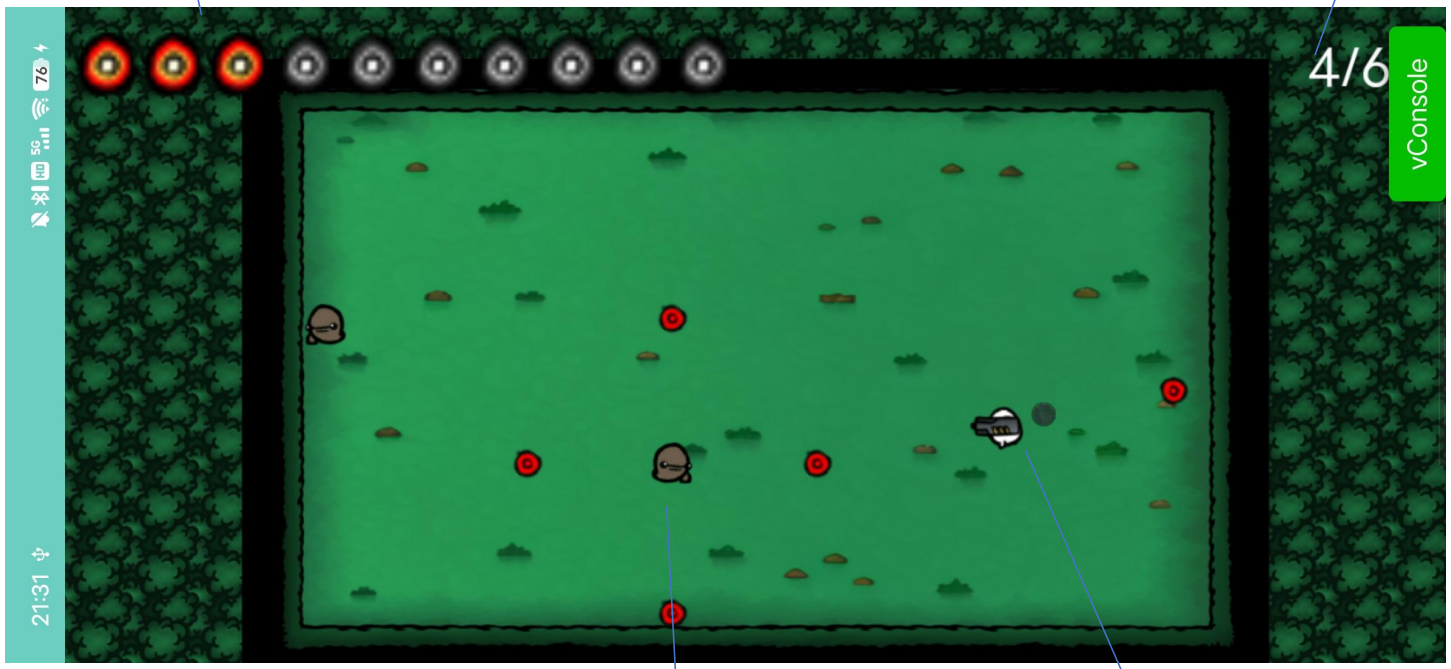
1

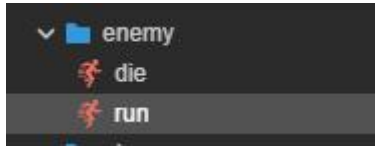
YISA

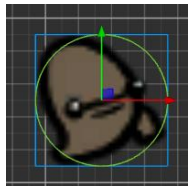
游戏主场景

玩家血条

敌人数量

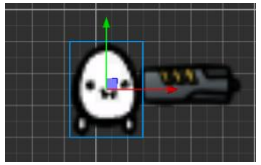






敌人

玩家



角色实现

在xy平面内移动

跑步（Run）

```
update(deltaTime: number): void {
    this.velocity.set(this.actor.input.x, this.actor.input.y);//设置速度
    this.velocity.multiplyScalar(this.actor.linearSpeed);//乘以速度系数
    this.actor.rigidbody.linearVelocity = this.velocity;//设置刚体速度

    if (this.actor.input.length() <= math.EPSILON) {//如果输入向量的长度小于等于0
        this.actor.stateMgr.transit(StateDefine.Idle);//切换到Idle状态
    }
}
```

冲刺（Dash）

```
time: number = 0;//计时器
duration: number = 0.8;//持续时间
dashFactor: number = 2.0;//冲刺系数
dashVelocity: Vec2 = v2();//冲刺速度
```

死亡（Die）

```
onEnter(): void {
    this.actor.rigidbody.linearVelocity = Vec2.ZERO;//速度归零
    this.animation.play(StateDefine.Die);//播放死亡动画

    this.animation.once(Animation.EventType.FINISHED, this.onDieEnd, this)//监听死亡动画播放完毕事件

    this.actor.dead = true;//标记死亡
}

onDieEnd(type: Animation.EventType, state: AnimationState) {
    if (type == Animation.EventType.FINISHED) {//死亡动画播放完毕
        if (state.name == StateDefine.Die) {//播放的是死亡动画
            //TODO: remove from parent
            this.actor.scheduleOnce(() => {//延迟0.1秒后销毁
                this.actor.node.destroy();
                director.emit(GameEvent.OnDie, this.actor.node);
            }, 0.1);
        }
    }
}
```

等待（Idle）

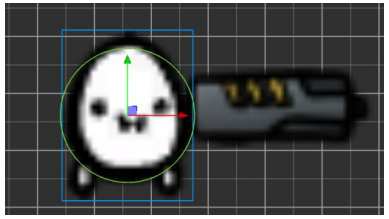
```
onEnter(): void {
    this.actor.rigidbody.linearVelocity = Vec2.ZERO;//速度归零
    let hasIdle = this.animation.getState(StateDefine.Idle);//是否有Idle动画
    if (hasIdle){
        this.animation.play(StateDefine.Idle);//播放Idle动画
    }
}
```

角色实现

发射子弹

```
fire(direction: AimDirection) {//开火
    this.gun.setWorldRotationFromEuler(0, 0, direction as number);//设置枪的旋转
    this.projectileEmitter.emit();//发射子弹
}
```

```
export enum AimDirection {
    RIGHT = 0,
    UP = 90,
    LEFT = 180,
    DOWN = -90,
    RIGHT_UP = 45,
    LEFT_UP = 135,
    LEFT_DOWN = -135,
    RIGHT_DOWN = -45,
}
```



玩家和子弹碰撞

```
//在角色与投射物碰撞开始时被调用。在这个方法中，检测了碰撞的投射物是否可以造成伤害
onProjectileTriggerEnter(ca: Collider2D, cb: Collider2D, contact: IPhysics2DContact) {
    if (colliderTag.isProjectileHitable(cb.tag, ca.tag)) {
        // console.log('project tigger enter', contact);
        let hurtSrc = cb.node.getComponent(Projectile).host;
        let hitNormal = v3();
        Vec2.subtract(hitNormal, ca.node.worldPosition, cb.node.worldPosition);//计算碰撞法线
        hitNormal.normalize();
        const v2Normal = v2(hitNormal.x, hitNormal.y);
        this.onHurt(this.attack, hurtSrc, v2Normal);
    }
}
```

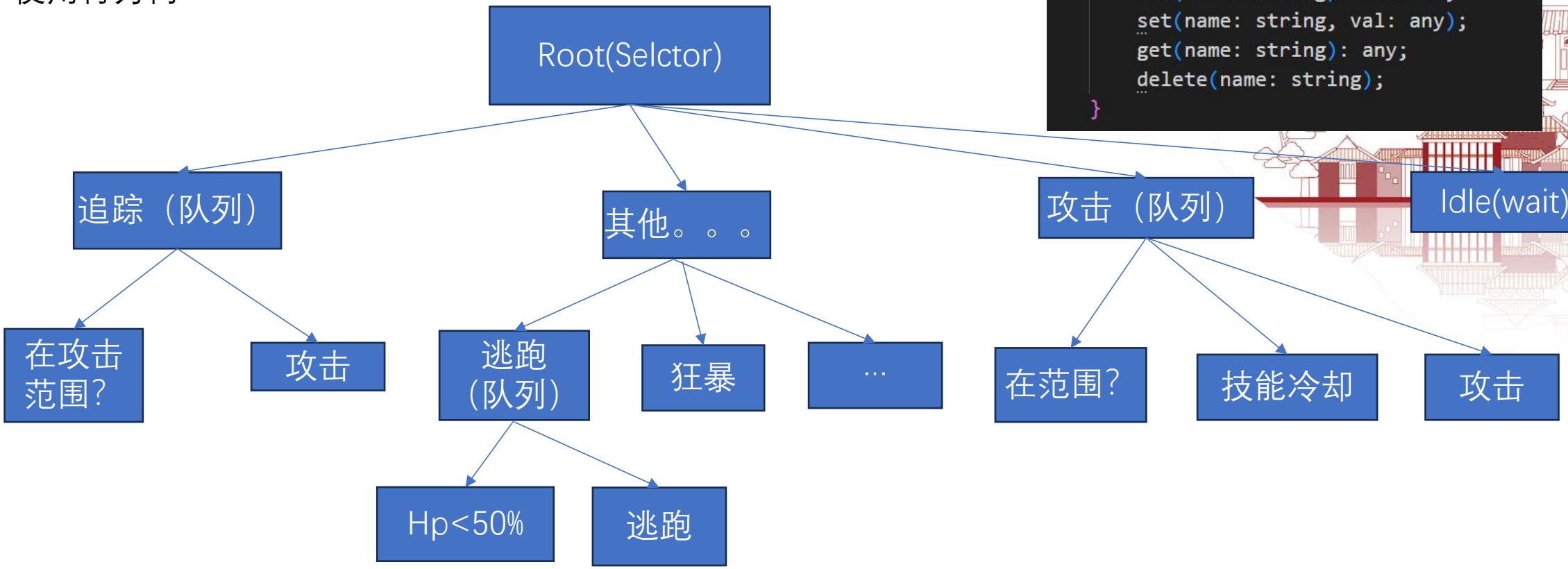
```
//在角色受到伤害时被调用。在这个方法中，减少了角色的生命值，如果生命值降到 0 或以下，则将角色的状态转换为死亡状态
onHurt(damage: number, from: Actor, hurtDirection?: Vec2) {
    this.hp = Math.floor(math.clamp(this.hp - damage, 0, this.maxHp));
    //受力
    this.rigidbody.applyLinearImpulseToCenter(hurtDirection,true);
    //受伤闪硕
    this.mainRenderer.color = Color.RED;
    this.scheduleOnce(() => {
        this.mainRenderer.color = Color.WHITE;
    }, 0.2);

    if (this.hp <= 0) {
        this.stateMgr.transit(StateDefine.Die);
    }
}
```

区分玩家和敌人的碰撞

```
export function isProjectileHitable(tag: number, other: number): boolean {
    if (tag == Define.PlayerProjectile) {
        return other == Define.Scene || other == Define.Enemy;
    }

    if (tag == Define.EnemyProjectile) {
        return other == Define.Scene || other == Define.Player;
    }
    return false;
}
```

敌人

使用行为树




```ts
/**
* AI 的黑板
*/
export interface Blackboard {
    has(name: string): boolean;
    set(name: string, val: any);
    get(name: string): any;
    delete(name: string);
}
```

敌人

使用行为树

```
createAI() {
    this.ai = new bt.BehaviourTree();

    // root
    let rootNode = new bt.Fallback();
```

行为树结点

1.顺序节点 (bt.Sequence)：一系列子节点按顺序执行，直到有一个失败为止。
2.选择节点 (bt.Fallback)：一系列子节点按顺序执行，直到有一个成功为止。
3.并行节点 (bt.Parallel)：同时执行所有子节点，可以设置成功或失败的条件。
4.条件节点 (bt.IsTrue)：检查黑板上的条件，根据条件的真假决定成功或失败。
5.反转节点 (bt.InvertResultDecorator)：反转子节点的执行结果，成功变为失败，失败变为成功。
6.行为节点 (bt.Action)：执行具体的游戏行为，例如移动、攻击、逃跑等。
7.等待节点 (bt.Wait)：暂停执行一段时间，然后返回成功。

```
//优先级最高逃跑
if (1) {
    // escape
    let escapeSeq = new bt.Sequence();//逃跑
    rootNode.addChild(escapeSeq);

    // has the escaped key?
    let invertHasEscapedKey = new bt.InvertResultDecorator();//反转->没有逃跑

    let hasEscapeKey = new bt.IsTrue();
    hasEscapeKey.key = BlackboardKey.Escaped;
    invertHasEscapedKey.child = hasEscapeKey;
    escapeSeq.addChild(invertHasEscapedKey);

    let lowHp = new IsLowHp();//血量低于一定值
    escapeSeq.addChild(lowHp);

    //TODO: add escape action
    let escape = new EscapeDash();
    escapeSeq.addChild(escape);
}
```

逃跑

```
// Partrol .... move to dest position
if (1) {
    let moveDestSeq = new bt.Sequence();//移动到目标位置

    let hasMoveDest = new bt.IsTrue();
    hasMoveDest.key = BlackboardKey.MoveDest;
    moveDestSeq.addChild(hasMoveDest);

    let moveDest = new MoveToDest();
    moveDestSeq.addChild(moveDest);

    rootNode.addChild(moveDestSeq);
}
```

移动到目标位置

```
if (1) {
    let emitSeq = new bt.Sequence();//发射子弹

    let simpleEmitter = this.node.getComponentInChildren(SimpleEmitter);
    let cooldown = new IsCooldown();//冷却
    cooldown.emitter = simpleEmitter;
    emitSeq.addChild(cooldown);

    let emit = new Emit();
    emit.emitter = simpleEmitter;
    emitSeq.addChild(emit);

    rootNode.addChild(emitSeq);
}
```

发射子弹

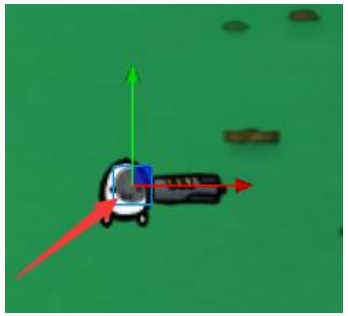
```
let idleSeq = new bt.Sequence();
rootNode.addChild(idleSeq);

idleSeq.addChild(new StayIdle());
let wait = new bt.Wait();//保底策略->发呆
wait.elapsed = 1.0;
idleSeq.addChild(wait);
idleSeq.addChild(new SetMoveDest());
```

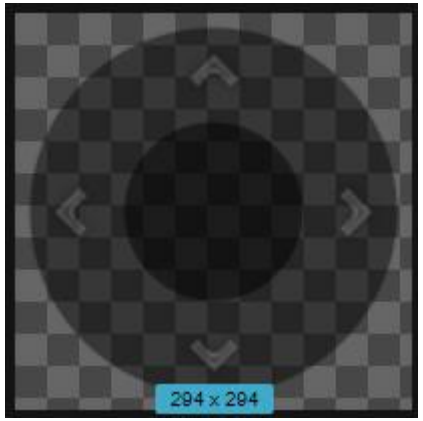
等待

体感动作算法

玩家移动逻辑:

通过xy坐标系的动作移动控制一个光标的移动
（类似于虚拟摇杆）






```protected _oldPos1: Vec2;
protected thumbnailPosition=v3(0,0,0);
onGsencor(X, Y, Speed, X_Throw, Y_Throw, Z_Gravity, Count_Throw, X_Gravity, Y_Gravity) {
    let delta;

    if (this._oldPos1) {
        delta = v2(this._oldPos1.x - X, this._oldPos1.y - Y);
        //解决移动超出陀螺仪边界问题
        if (delta.x > 1000) {
            delta.x = delta.x - 3600;
        } else if (delta.x < -1000) {
            delta.x = delta.x + 3600;
        }
        if (delta.y > 900) {
            delta.y = delta.y - 1800;
        } else if (delta.y < -900) {
            delta.y = delta.y + 1800;
        }

        this.thumbnailPosition.x+=delta.x*0.25;
        this.thumbnailPosition.y+=delta.y*0.35;

        let distanceToCenter = Math.sqrt(Math.pow(this.thumbnailPosition.x, 2) + Math.pow(this.thumbnailPosition.y, 2));
        // 如果 thumbnailPosition 超出了圆的范围
        if (distanceToCenter > this.radius) {
            // 将 thumbnailPosition 重新定位到圆上
            let angle = Math.atan2(this.thumbnailPosition.y, this.thumbnailPosition.x); // 计算角度
            this.thumbnailPosition.x = this.radius * Math.cos(angle);
            this.thumbnailPosition.y = this.radius * Math.sin(angle);
        }

        this.thumbnail.node.setPosition(this.thumbnailPosition);
        VirtualInput.horizontal = this.thumbnail.node.position.x / this.radius;
        VirtualInput.vertical = this.thumbnail.node.position.y / this.radius;

    }
    this._oldPos1 = v2(X, Y);
}
```

体感动作算法

玩家射击逻辑

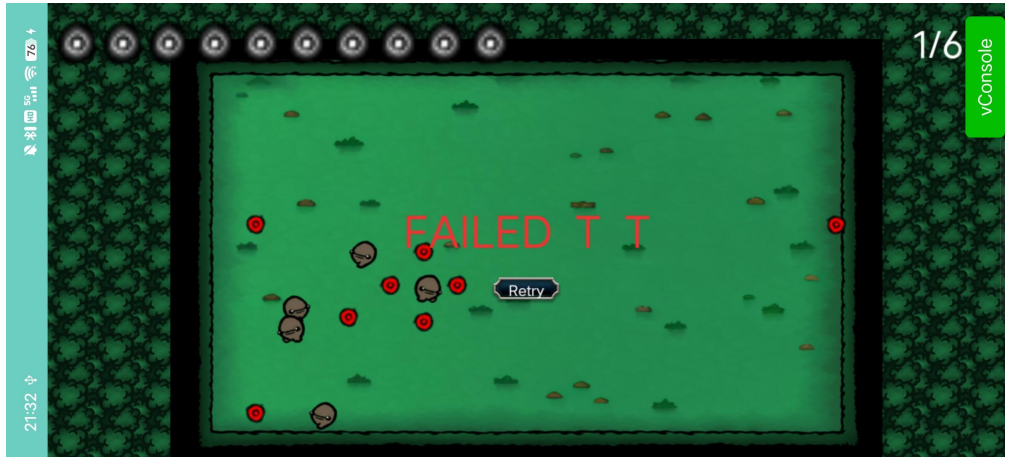
使用四个虚拟按键，控制四个发射子弹的方向

```
/**--------------------游戏操作 相关控制---------------------- */
_lastKeyboardNum: number = 0;
onVKeyboardNumCb(num:number) {
    //上
    if(num==1 && this._lastKeyboardNum==0){
        this.fire(90);
    }
    //下
    if(num==2 && this._lastKeyboardNum==0){
        this.fire(-90);
    }
    //左
    if(num==3 && this._lastKeyboardNum==0){
        this.fire(180);
    }
    //右
    if(num==4 && this._lastKeyboardNum==0){
        this.fire(0);
    }
    if(num==0 && this._lastKeyboardNum!=0){

    }
    this._lastKeyboardNum=num;
}
```

```
fire(direction: AimDirection) {//开火
    this.gun.setWorldRotationFromEuler(0, 0, direction as number);//设置枪的旋转
    this.projectileEmitter.emit();//发射子弹
}
```

游戏结束

失败->重新开始



成功->下一关（增加难度）

敌人射击移动速度和射击速度增加

湖南大学
HUNAN UNIVERSITY

2

光之剑

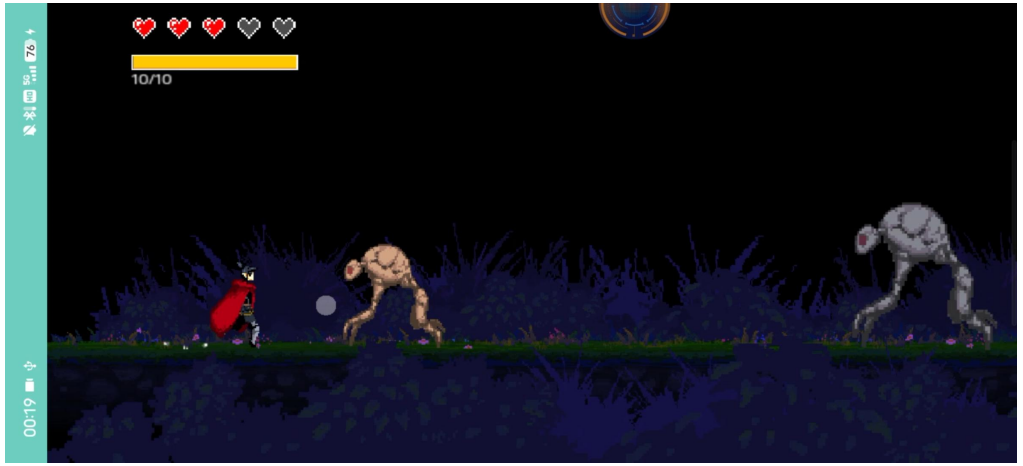
游戏主场景

游戏主场景

游戏初始化场景

死亡

第一个场景通过后



角色实现

左右移动

跳跃
连续按下两次虚拟按键连续跳跃两次


```
        if(this.jumpCount < 2){
            this.node.getComponent(cc.AudioSource).play()
            this.dust()
            this.fall = false
            this.Anim.play("jump")
            this.jumpSpd = 500
            this.jumpCount += 1
            this.spriteScript.state = "jump"
        }
```

```
    //播放降落动画  (Play the falling animation)
    if(this.jumpSpd < 0){
        if(this.spriteScript.state != "jumpAtk"){
        if(this.fall == false ){
            this.Anim.play("fall")
            this.fall = true
        }
        }
    }
    //模拟重力  (The gravity)
    if(this.spriteScript.state == "jump" || this.spriteScript.state == "jumpAtk"){

        this.jumpSpd -= 25
}

    //角色跳起与下落物理变化    (character jumping and falling)
    this.player.y += this.jumpSpd * dt
```

```
    //角色落地判断  (Landing)
    if (this.player.y < 0) {
        if (this.spriteScript.state == "jump") {
            this.spriteScript.state = "stand"

            if (this.playerScript.dir != 0) {
                this.Anim.play("run")
            } else {
                this.Anim.play("landing")
            }
        }
        if (this.spriteScript.state == "jumpAtk") {
            this.spriteScript.state = "atk"
        }

        this.btnDash.IsDash = false
        this.OnPlatform = false
        this.jumpSpd = 0
        this.player.y = 0
        this.jumpCount = 0
        var landingDust = cc.instantiate(this.landingDust)
        landingDust.parent = this.player.parent
        landingDust.setPosition(this.player.getPosition())
```

角色实现

突击

```
if (this.IsDash == false) {

    //开始移动 (Start to dash)
    this.dashSPD = 800

    //转到冲刺状态 (State to "dash")
    this.spriteScript.state = "dash"
    this.jbS.jumpSpd = 0

    //冲刺特效 (Instantiate dash effect)
    this.DashEFF()

    //播放冲刺动画 (Play dash animation)
    this.Anim.play("dash")

    //冲刺状态结束 (Set a timer to end the dash)
```

攻击

```
    //状态切换 (State changed)
    if (this.spriteScript.state == "jump" || this.spriteScript.state == "dashAtk") {
        this.spriteScript.state = "jumpAtk"
    }
    if (this.spriteScript.state != "jump" && this.spriteScript.state != "jumpAtk") {
        if (this.player.y > 0) {
            this.spriteScript.state = "jumpAtk"
        } else {
            this.spriteScript.state = "atk"
        }
    }
```

发射技能

```
    if (this.spriteScript.state == "stand" && this.player.st >= 2) {

        //扣除耐力
        this.player.st -= 2
        this.HPnST.getComponent('HPnST').HPchange()

        this.spriteScript.state = "atks"
        this.Anim.play("atks")
        //音效 (Sound effect)
        this.node.getComponent(cc.AudioSource).play()
    }
}
```

角色实现

部分事件

走路声音

```
//动画事件,走路声音
soundRun(){

    this.node.getComponent(cc.AudioSource).play()
},
```

走路扬尘

```
//动画事件,走路扬尘  (Animation event----Show the dust of walking)
Dust(){
    var dust = cc.instantiate(this.dust)
    dust.parent = this.player.parent
    dust.setPosition(this.player.getPosition())
},
```

攻击特效

```
//加载特殊攻击特效  (This is a Animation event ----- Load special attack effect)
ball(){
    var eff = cc.instantiate(this.Ball)
        eff.parent= this.player.parent
        eff.scaleX = this.node.scaleX
        eff.x = this.player.x + this.node.scaleX * 60
        eff.y = this.player.y + 45
},
```

挥剑声音

```
//攻击动画帧事件内触发此函数    (This function is
soundsword(){
    //播放挥剑声音  (Play sword swing sound)
    this.sound.play()
},
```

复活

```
//复活
revive(){
    this.finish()
    cc.find("Canvas/uiCamera/ui").active = true
}
```

敌人

```
onLoad () {
    //初始血量  (Origin HP)
    this.hp = this.maxHP
    //出生时的坐标  (Origin position)
    this.startPos = this.node.x
    //左折返点  (When monster move to this position,turn left)
    this.l = this.startPos - this.range
    //右折返点  (When monster move to this position,turn right)
    this.r = this.startPos + this.range
    //当前状态  (Current state)
    this.state = "L"
},

//重新开始追踪主角  (Restart tracking player)
ReChase(){
    if(this.state != "die"){
        this.Anim.play("mon01_move")
        this.state = "chase"}
},
```

```
1  //处于追踪状态  (Walk towards the player)
2  if(this.state == "chase"){
3      if(this.node.x > playerPos.x+1){
4          this.node.x -= 1
5          this.node.scaleX = 1
6      }else if(this.node.x < playerPos.x-1){
7          this.node.x += 1
8          this.node.scaleX = -1
9      }
10     //进入可攻击范围    (Player enter the attackable range)
11     if(dist < 80){
12         this.state = "atk"
13         this.Anim.play("mon01_atk")
14         setTimeout(() => {
15         this.ReChase()
16         }, 2000+Math.random()*500);
17     }
18 }
19 //向左走    (Move to the left)
20 if(this.state == "L"){
21     this.node.x += -1
22     if(this.node.x <= this.l){
23         this.state = "R"
24         this.node.scaleX = -1
25     }
26      //切换到追踪   (Enter the tracking range)
27      if(dist < 120){
28         this.state = "chase"
29     }
30 }
31 //向右走    (Move to the right)
32 if(this.state == "R"){
33     this.node.x += 1
34     if(this.node.x >= this.r){
35         this.state = "L"
36         this.node.scaleX = 1
37     }
38      //切换到追踪   (Enter the tracking range)
39      if(dist < 120){
40         this.state = "chase"
41     }
42 }
```

体感动作算法

玩家移动逻辑:

通过x轴的动作移动控制一个光标的移动

```
_oldPos1: cc.v2,
MoveGsensor(x, y, speed, xThrow, yThrow, zG, countThrow, xG, yG){
    // console.log('1111')
    if (this._oldPos1) {
        let delta = cc.v2(this._oldPos1.x - x, this._oldPos1.y - y);
        //解决移动超出陀螺仪边界问题
        if (delta.x > 1000) {
            delta.x = delta.x - 3600;
        } else if (delta.x < -1000) {
            delta.x = delta.x + 3600;
        }
        if (delta.y > 900) {
            delta.y = delta.y - 1800;
        } else if (delta.y < -900) {
            delta.y = delta.y + 1800;
        }
        this.thumbnailPosition.x+=delta.x*0.2;
        if (this.thumbnailPosition.x < -70) {
            this.thumbnailPosition.x = -70
        }
        if (this.thumbnailPosition.x > 70) {
            this.thumbnailPosition.x = 70
        }
        this.bnt.x=this.thumbnailPosition.x;
        this.playerScript.dir=this.bnt.x/70;
        if(this.spriteScript.state == "stand" && Math.abs(this.playerScript.dir) >= 0.4){

            if(this.jbS.jumpCount ==0){
                this.Anim.play("run");
                this.spriteScript.state = "run"
            }
        }
        else if(this.spriteScript.state == "run" && Math.abs(this.playerScript.dir) < 0.4){
            this.Anim.play("stand")
            this.spriteScript.state = "stand"
        }
        if (this.playerScript.dir >= 0) {
            this.playerSprite.scaleX = 1;
        }
        else {
            this.playerSprite.scaleX = -1;
        }

    }
    this._oldPos1 = cc.v2(x, y);
},
```

体感动作算法

玩家其他行为逻辑:
虚拟按键

跳跃

```
    _lastKeyboardNum: 0,
    onVKeyboardNumCb(num){
        if(num==1 && this._lastKeyboardNum==0){
```

突击

```
    _lastKeyboardNum: 0,
    onVKeyboardNumCb(num) {
        if (num == 4 && this._lastKeyboardNum == 0) {
```

攻击

```
    _lastKeyboardNum: 0,
    onVKeyboardNumCb(num) {
        if (num == 3 && this._lastKeyboardNum == 0)
```

发射技能

```
    _lastKeyboardNum: 0,
    onVKeyboardNumCb(num) {
        if (num == 2 && this._lastKeyboardNum == 0)
```

跳跃

攻击

突击

发射技能

体感动作算法

玩家生命值耗尽

进入死亡场景

复活
游戏重新初始化


```
//没血了
if(this.player.getComponent('player').hp <=0 ){
    this.playerspriteAnim = this.sprite.getComponent(cc.Animation).play('die')
    cc.find("Canvas/uiCamera/ui").destroy()
    //死亡黑幕
    this.blackmask = cc.find("Canvas/blackmask")
    this.blackmask.active = true
    this.blackmask.zIndex = 999
    this.blackmask.setPosition(this.player.x,this.player.y)
    this.blackmask.getComponent(cc.Animation).play('blackmask')
    setTimeout(() => {
        cc.director.loadScene('test2')
        WebviewBridge.setVKeyboardNumCb(null, null);
        WebviewBridge.setP1MoveCallback(null, null);
    }, 4500);

}
```