中山大学 SUN YAT-SEN UNIVERSITY 现代操作系统期末项目实验报告

一、 项目分工

学号	名字	角色	班级	职责	贡献
16340014	车鑫恺	组长	下午班	菜单界面、历史成绩等相关实现	25%
16340015	陈彬彬	组员	下午班	角色的移动、攻击、死亡等相关实现	25%
16340016	陈冬禹	组员	下午班	怪物的移动、攻击、死亡等相关实现	25%
16340017	陈帆	组员	下午班	地图制作、角色移动、显示信息等	25%

二、开发环境

Visual Studio2017、cocos2d-x-3.16

三、 项目阐述

名称	生存岛
简介	这是一个 1p/2p 的动作射击类游戏，玩家在不同的环境下面临怪物的追击，需要获取补给来得到武器和弹药，活下去就得杀死怪物，时间越久怪物会越来越多，看看你能拿多少分吧！
功能	1. 进入游戏有玩法介绍 2. 可选择地图、单人/双人模式 3. 通过键盘来分别控制角色移动、攻击怪物 4. 屏幕会根据角色的移动而移动 5. 怪物主动追击角色，并攻击 6. 杀死怪物可以获得分数，游戏结束后自动记录成绩
亮点	1. 枪械、子弹种类丰富 2. 角色的移动、攻击流畅 3. 界面动画丰富、美观

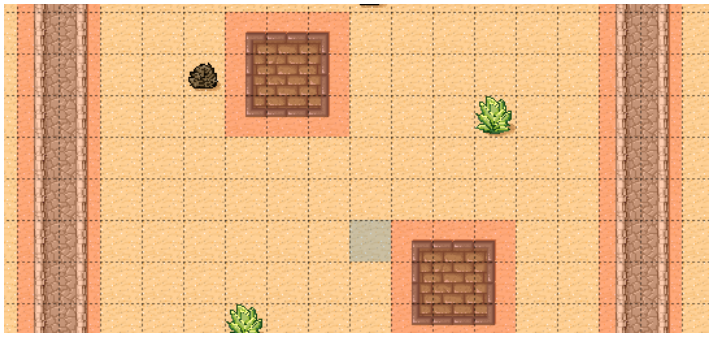
四、 项目展示

直接运行提供的 exe 文件即可，具体可见展示视频

五、 项目难点及解决方案

1. **难点**：tileMap 制作地图要考虑到障碍物的问题，不能让角色太随心所欲地移动

   解决方法：这需要在地图上弄好要设障碍物的位置(要单独弄一个图层出来)：



上图红色部分即为有障碍物的地方。这些红色的是用图块覆盖的，图块要增加自定义属性以设置

成障碍物

核心代码：

```
Point tileCoord = this->tileCoordForPosition(position);
int tileGid = blockage->getTileGIDAt(tileCoord);
if (tileGid) {
    auto properties = tileMap->getPropertiesForGID(tileGid).asValueMap();
    if (!properties.empty()) {
        //地图上的障碍物，不能移动到这里
        auto collision = properties["Block"].asString();
        if ("true" == collision) return;
    }
}
```

先获得对应的瓦片地图中的坐标，再获得相应瓦片地图上的瓦片，判断其是否有障碍物属性，为 true 则说明这是障碍物

2. **难点**：角色移动时，因为地图大过屏幕大小，会出现角色移动没有超过地图范围，但超过了屏幕返回的情况，但不知道为什么一开始在设置视角后又会自动跳转到别的视角，之后又可以了，调试了很久也没能发现问题源头

解决方法：角色移动时屏幕视角也应该相应移动，可得到当前视角中心与目标位置的偏移，再移动屏幕，要注意的是不能让屏幕显示地图之外的部分，具体代码在 setViewPoint 函数中：

核心代码：

```
//显示屏幕中心点的坐标大于屏幕宽和高的一半,防止部分视图在屏幕之外，即屏幕移出地图边界下/左方
int x = MAX(position.x, winSize.width / 2);
int y = MAX(position.y, winSize.height / 2);
```

```
int mapWidth = tileMap->getMapSize().width * tileMap->getTileSize().width;
x = MIN(x, mapWidth - winSize.width / 2);
//整个瓦片地图的高
int mapHeight = tileMap->getMapSize().height * tileMap->getTileSize().height;
y = MIN(y, mapHeight - winSize.height / 2);
auto actualPosition = Point(x, y);
```

```
auto centerOfView = Point(winSize.width / 2, winSize.height / 2);
auto viewPoint = centerOfView - actualPosition;
//重置显示屏幕的中心点
this->setPosition(viewPoint);
```

3. **难点**：与玩家相关的信息(枪械、分数等时)通过 label 显示时，要使其始终在窗口左(右)上角，label 位置难以设置

解决方法：要注意信息也要随着窗口移动而移动，视觉上感觉一直在屏幕左(右)上角，实现思路来自与设置屏幕视角，在得到 actualPosition 之后，就得到了屏幕视角的中点，再加上一些偏移量就可以实现信息的移动了

核心代码：

```
// 设置位置
label_player1->setPosition(actualPosition +
    Vec2(-visibleSize.width / 2, visibleSize.height / 2 - 10));
hp1->setPosition(label_player1->getPosition() - Vec2(0, 30));
attackType1->setPosition(hp1->getPosition() - Vec2(0, 30));
count1->setPosition(attackType1->getPosition() - Vec2(0, 30));
score1->setPosition(count1->getPosition() - Vec2(0, 30));
```

4. **难点**：关于界面跳转时的传参问题，由于生成人物之前就需要获得一些变量，例如地图名称和单人或者双人模式，因此变量需要在创建新的场景之前传递过去，所以不能存在各个 Scene 类里，不管是否是静态或者非静态都不行。

解决方法：为了便于管理，创建了一个新的类用于控制全局变量。

核心代码：

```
class GlobalVar : public cocos2d::Ref{
private:
    GlobalVar();
    static GlobalVar* instance;
    //是否双人模式
    bool double_player;
    //文件来源
    string mapFile;
public:
    static GlobalVar* getInstance();
    string getMapFile();
    bool getDoublePlayer();
    void setMapFile(string m);
    void setDoublePlayer(bool b);
    //获得武器类型
    int getAttack1();
    int getAttack2();
```

5. **难点**：在实现主界面点击显示帮助信息和历史成绩时，最初是产生一个新的界面并跳转，但这样会使代码结构复杂，新产生的界面没有什么实际用处

解决方法：参考最后一周的作业，改为用 TxtField 来实现

核心代码：

```
private:
    cocos2d::Sprite* mapsprite;
    TextField *helpMessage;
    TextField *historygrade;
};
```

```
    if (helpMessage->getString().length() != 0)
        helpMessage->setString("");
    else {
        CCDictionary* message = CCDictionary::createWithContentsOfFile("chinese.xml");
        auto helpKey = message->valueForKey("helpMessage");
        const char* helpString = helpKey->getCString();//获取中文玩法信息
        helpMessage->setString(helpString);
    }
```

6. **难点**：在记录历史最佳成绩时，要考虑是否三个成绩已经存满，还有是单人模式还是双人模式

解决方法：在 Userdefault 里分别存储单人和双人成绩，并根据存储数据进行逻辑判断是否有成绩空位

核心代码：

```
if (grade_d[0] == 0) {
    database->setIntegerForKey("first_d", grade);
}
else if (grade_d[1] == 0) {
    if(grade <= grade_d[0])
        database->setIntegerForKey("second_d", grade);
    else {
        database->setIntegerForKey("first_d", grade);
        database->setIntegerForKey("second_d", grade_d[0]);
    }
}
else if (grade_d[2] == 0) {
    if (grade >= grade_d[0]) {
        database->setIntegerForKey("first_d", grade);
        database->setIntegerForKey("second_d", grade_d[0]);
        database->setIntegerForKey("third_d", grade_d[1]);
    }
    else if (grade >= grade_d[1]) {
        database->setIntegerForKey("second_d", grade);
        database->setIntegerForKey("third_d", grade_d[1]);
    }
    else {
        database->setIntegerForKey("third_d", grade);
    }
}

  else {
      if (grade >= grade_d[0]) {
          database->setIntegerForKey("first_d", grade);
          database->setIntegerForKey("second_d", grade_d[0]);
          database->setIntegerForKey("third_d", grade_d[1]);
      }
      else if (grade >= grade_d[1]) {
          database->setIntegerForKey("second_d", grade);
          database->setIntegerForKey("third_d", grade_d[1]);
      }
      else if(grade >= grade_d[2]){
          database->setIntegerForKey("third_d", grade);
      }
  }
```

7. **难点**：怪物向角色移动时，会互相卡住而不能移动

   解决方法：原因是在判断怪物是否能够移动时，需要判断怪物目的地坐标（怪物位置+移动向量）是否包含障碍物或其他怪物，而我把移动向量（距离）设置得比较小，所以一旦怪物十分接近，判断结果就会一直为不能移动。所以增大移动向量就能解决问题。

   核心代码：

```
// 判断目的地是否有障碍物
Point position = monster.at(i)->getPosition() + dir * 15;
Point tileCoord = this->getPositionInMap(position);
int tileGid = blockage->getTileGIDAt(tileCoord);
if (tileGid) {
  auto properties = tileMap->getPropertiesForGID(tileGid).asValueMap();
  if (!properties.empty()) {
    //地图上的障碍物，不能移动到这里
    auto collision = properties["Block"].asString();
    if ("true" == collision) canMove = false;
  }
}
for (auto j : monster) {
  if (monster.at(i) != j) {
    Rect otherMonster = j->getBoundingBox();
    if (otherMonster.containsPoint(position)) {
      canMove = false;
    }
  }
```

8. 难点：维护怪物相关列表时，容易出现越界错误

   解决方法：手动判断临界条件。

   核心代码：

```
if (i >= 0 && i < monster.size()) {
```

9. 难点：怪物的帧动画可能会发生冲突

   解决方法：

   对每个怪物使用一个互斥锁来控制帧动画的播放

   维护一个锁的列表

   播放前，判断 isDone 为真才播放帧动画

   播放后，把 isDone 设置为真

   核心代码：

```
if (isDone[i]) {
  // 播放动画后将对应isDone的值设为true

  //auto seq = Sequence::createWithTwoActions(animate, setDoneToTrue);
  monster.at(i)->runAction(animate);
  if (i >= 0 && i < monster.size()) {
    isDone[i] = true;
  }
}
```

10. 角色移动方面：

    难点：

    1. 八方向的位移问题：该如何做到很流畅地用四个方向键完成八方向的位移，摁住方向键不松开实现连续移动，松开摁键能实现快速停下。• 移动与射击的互斥问题，因为射击我也设计了帧动画，于是摁射击键的时候，人物要停下，并且执行完射击帧动画后才允许继续移动。

    2. 位移与帧动画的流畅结合，停下时帧动画也要停下，转向时帧动画也要转换

    解决方法：

    1. 角色移动的帧动画是八方向的，一开始以为只要确定人物当前的朝向即可，后来发现没这么简单，因为人物朝向不代表它在动，所以还要**加上是否在移动的判断**。另外如果简单用 sequence：动画 MoveBy,MoveTo+八方向帧动画来实现人物移动位置和实现帧动画的话，会出现移动不够流畅的效果，于是决定改**用 setPosition 来实现移动，帧动画另外分开进行 runAction（sequence），用调度器 update 来调度转向**，这样实现了迅速地转向与停下，一个较为流畅的移动效果。

2.用当前的执行方向——（八方向 or stop）与之前的执行方向进行对比，分 stop，与之前不同，与之前相同三种情况进行逻辑判断。

核心代码：

```
//让玩家更新移动动作的帧动画
void Factory::update_Player_Animation(Sprite* player, bool isPlayer1) {
    // 如果玩家已经死亡，不更新
    if ((isPlayer1&&GlobalVar::getInstance()->player1HP <= 0) || (!isPlayer1) && GlobalVar::getInstance()->player2HP <= 0) return;
    bool isMove_hori = isPlayer1 ? GlobalVar::getInstance()->player1_isMove_hori : GlobalVar::getInstance()->player2_isMove_hori;
    bool isMove_vert = isPlayer1 ? GlobalVar::getInstance()->player1_isMove_vert : GlobalVar::getInstance()->player2_isMove_vert;
    char moveKey_hori = isPlayer1 ? GlobalVar::getInstance()->player1_moveKey_hori : GlobalVar::getInstance()->player2_moveKey_hori;
    char moveKey_vert = isPlayer1 ? GlobalVar::getInstance()->player1_moveKey_vert : GlobalVar::getInstance()->player2_moveKey_vert;
    bool done = isPlayer1 ? GlobalVar::getInstance()->player1_done : GlobalVar::getInstance()->player2_done;
    string current_direction = isPlayer1 ? GlobalVar::getInstance()->player1_moving_direction : GlobalVar::getInstance()->player2_moving_direction;
    string prename = (isPlayer1) ? "player1_move_" : "player2_move_";
    //bool isShotDone = (isPlayer1) ? GlobalVar::getInstance()->player1_isShotDone : GlobalVar::getInstance()->player2_isShotDone;
    // 获得player1下一步的移动方向
    string nextDirection = getDirection(isMove_hori, isMove_vert, moveKey_hori, moveKey_vert);

    //如果玩家当前在射击，移动帧动画不进行更新，直接返回
    if (current_direction == "shot") return;

    // 若下一步移动方向为静止，则停止所有帧动画
    if (nextDirection == "stop") {
        player->stopAllActions();

        if (isPlayer1) {
            GlobalVar::getInstance()->player1_done = true;
            GlobalVar::getInstance()->player1_moving_direction = nextDirection;
        }
        else {
            GlobalVar::getInstance()->player2_done = true;
            GlobalVar::getInstance()->player2_moving_direction = nextDirection;
        }
        return;
    }
```

```
// 若下一步移动方向跟现在的方向一致， 接连循环播放该方向动画
if (nextDirection == current_direction && done) {
    GlobalVar::getInstance()->player1_moving_direction = isPlayer1 ? nextDirection : GlobalVar::getInstance()->player1_moving_direction;
    GlobalVar::getInstance()->player2_moving_direction = isPlayer1 ? GlobalVar::getInstance()->player2_moving_direction : nextDirection;
    string name = prename + nextDirection;
    Animate* animate = Animate::create(AnimationCache::getInstance()->getAnimation(name));
    auto setDoneToTrue = CallFunc::create([&,isPlayer1]() {
        //player1_done = isPlayer1 ? true : player1_done;
        //player2_done = isPlayer1 ? player2_done : true;
        if (isPlayer1) GlobalVar::getInstance()->player1_done = true;
        else GlobalVar::getInstance()->player2_done = true;
    });
    auto sequence = Sequence::create(animate, setDoneToTrue, NULL);
    // 将done设为false
    GlobalVar::getInstance()->player1_done = isPlayer1 ? false : GlobalVar::getInstance()->player1_done;
    GlobalVar::getInstance()->player2_done = isPlayer1 ? GlobalVar::getInstance()->player2_done : false;
    player->runAction(sequence);

}
```

```
// 若下一步移动方向跟现在的方向不一致
// 终止当前方向的帧动画，更新为新方向的帧动画
if (nextDirection != current_direction) {
    GlobalVar::getInstance()->player1_moving_direction = isPlayer1 ? nextDirection : GlobalVar::getInstance()->player1_moving_direction;
    GlobalVar::getInstance()->player2_moving_direction = isPlayer1 ? GlobalVar::getInstance()->player2_moving_direction : nextDirection;
    // 获得新方向的帧动画
    string name = prename + nextDirection;
    Animate* animate = Animate::create(AnimationCache::getInstance()->getAnimation(name));
    auto setDoneToTrue = CallFunc::create([&, isPlayer1]() {

        if (isPlayer1) {
            GlobalVar::getInstance()->player1_done = true;
        }
        else {
            GlobalVar::getInstance()->player2_done = true;
        }
        //this->player1_done = isPlayer1 ? true : player1_done;
        //this->player2_done = isPlayer1 ? player2_done : true;
    });
    auto sequence = Sequence::create(animate, setDoneToTrue, NULL);
    // 停止当前帧动画
    player->stopAllActions();
    // 将done设为false
    GlobalVar::getInstance()->player1_done = isPlayer1 ? false : GlobalVar::getInstance()->player1_done;
    GlobalVar::getInstance()->player2_done = isPlayer1 ? GlobalVar::getInstance()->player2_done : false;
    // 更新为新方向的移动帧动画
    player->runAction(sequence);
    //修改player朝向
    if (nextDirection != "stop") {
        GlobalVar::getInstance()->player1_toward = isPlayer1 ? nextDirection : GlobalVar::getInstance()->player1_toward;
        GlobalVar::getInstance()->player2_toward = isPlayer1 ? GlobalVar::getInstance()->player2_toward : nextDirection;
    }
}
```

11. **角色攻击方面：**

难点：

1. 移动位移+帧动画与射击帧动画的冲突
2. 6 种类型子弹的不同效果，包括样式，射程，伤害，能否连发各性质的不同
3. 子弹与怪物的碰撞条件的判断，迭代器的问题

解决方法：

1. 将执行方向 string 加一个 shot 状态，当这个状态来临时，要停止之前的移动和移动帧动画，进行射击。另外增加 toward 表示现在的朝向，子弹发射沿这个方向进行旋转。

2. 得到 toward 的朝向，要设置子弹精灵的锚点，旋转角度，然后设置射出轨迹，当超出范围时还要移除。另外用 Bullet 这个类来记下当前在场景飞的子弹队列，用于后续的子弹与怪物的碰撞条件判断。Bullet 还记下了剩余子弹的数目。因为有 6 种不同类型的子弹，所以要进行复杂的逻辑判断，来实现多种多样子弹的效果。

3. 子弹与怪物的碰撞条件的判断的难点在于迭代器，我们先要写好 Player 类，Bullet 类和 Monster 类的 collide()函数，用来碰撞检测的辅助。然后就是遍历各种各样的队列 Vector，然后用迭代器进行响应的处理。

核心代码：

```cpp
void PlayGameScene::meet(EventCustom * event)
{
    // 1.判断子弹是否打中怪物
    // 遍历两个Vector进行判断
    //1.1首先是Player1，有6个子弹队列s，判断子弹是否打中怪物

    for (int i = 0; i < 6; i++) {

        Vector<Sprite*> bullets = bulletFactory::getInstance()->getBulletNumInScene(i, true);
        Vector<Sprite*>::iterator it = bullets.begin();

        // 遍历第i类型子弹队列看是否有击中怪物
        for (it = bullets.begin(); it != bullets.end(); ) {
            Rect b = (*it)->getBoundingBox();
            Sprite* mon = MonsterFactory::getInstance()->collider(b);
            // 没有击中怪物
            if (mon == NULL) {
                //std::string str = "Player1---Type: " + Value(i).asString() + " --- not Hit!";
                //CCLOG(str.c_str());
                it++;
            }
            // 击中怪物
            else {
                std::string str = "Player1---Type: " + Value(i).asString() + " --- Hit!";
                CCLOG(str.c_str());
                //1. 移除子弹
                (*it)->stopAllActions();
                //(*it)->removeFromParent();
                bulletFactory::getInstance()->removeBullet((*it), i, true);
                if (it != bullets.begin()) it--;
                // 怪物扣血
                MonsterFactory::getInstance()->wasHit(mon, i);
                // 伤害得分
                if (i == 0) GlobalVar::getInstance()->player1Score += 3;
                else if (i == 1) GlobalVar::getInstance()->player1Score += 2;
                else if (i == 2) GlobalVar::getInstance()->player1Score += 5;
                else if (i == 3) GlobalVar::getInstance()->player1Score += 8;
                else if (i == 4) GlobalVar::getInstance()->player1Score += 15;
                else if (i == 5) GlobalVar::getInstance()->player1Score += 10;

                break;
            }
            if (it != bullets.end())it++;
        }
    }
```

```
//1.2.首先是Player2，有6个子弹队列s,判断子弹是否打中怪物
for (int i = 0; i < 6; i++) {
    Vector<Sprite*> bullets = bulletFactory::getInstance()->getBulletNumInScene(i, false);
    Vector<Sprite*>::iterator it = bullets.begin();
    //Vector<Sprite*>::iterator it2 = MonsterFactory::getInstance()->getMonster().begin;
    for (it = bullets.begin(); it != bullets.end(); ) {
        Rect b = (*it)->getBoundingBox();
        Sprite* mon = MonsterFactory::getInstance()->collider(b);
        // 没有击中怪物
        if (mon == NULL) {
            //std::string str = "Player2---Type: " + Value(i).asString() + " --- not hit!";
            //CCLOG(str.c_str());
            it++;
        }
        // 击中怪物
        else {
            std::string str = "Player2---Type: " + Value(i).asString() + " ---- Hit!";
            CCLOG(str.c_str());
            //1. 移除子弹
            (*it)->stopAllActions();
            //(*it)->removeFromParent();
            bulletFactory::getInstance()->removeBullet((*it), i, false);
            if (it != bullets.begin()) it--;
            //怪物扣血
            MonsterFactory::getInstance()->wasHit(mon, i);
            // 伤害得分
            if (i == 0) GlobalVar::getInstance()->player2Score += 3;
            else if (i == 1) GlobalVar::getInstance()->player2Score += 2;
            else if (i == 2) GlobalVar::getInstance()->player2Score += 5;
            else if (i == 3) GlobalVar::getInstance()->player2Score += 8;
            else if (i == 4) GlobalVar::getInstance()->player2Score += 15;
            else if (i == 5) GlobalVar::getInstance()->player2Score += 10;
            break;
        }
        if (it != bullets.end())it++;
    }
}
```

```
// 2.判断大怪的子弹是否打中帅气的玩家
Vector<Sprite*> monsterBullets = MonsterFactory::getInstance()->getMonsterBullet();
Vector<Sprite*>::iterator it = monsterBullets.begin();
// 遍历大怪子弹队列
for (it = monsterBullets.begin(); it != monsterBullets.end();) {
    Rect b = (*it)->getBoundingBox();
    Vec2 pos = (*it)->getPosition();
    Sprite* player = Factory::getInstance()->collider(pos);
    // 没有打中玩家
    if (player == NULL) {
        it++;
    }
    // 打中玩家
    else if (player == player1 && GlobalVar::getInstance()->player1HP > 0) {
        //1. 移除子弹
        (*it)->stopAllActions();
        MonsterFactory::getInstance()->removeMonsterBullet((*it));
        (*it)->removeFromParent();
        if (it != monsterBullets.begin()) it--;
        //2.人物扣血
        Factory::getInstance()->wasHit(player, 100, true);
    }
    else if (player == player2 && GlobalVar::getInstance()->getDoublePlayer() && GlobalVar::getInstance()->player2HP > 0) {
        //1. 移除子弹
        (*it)->stopAllActions();
        MonsterFactory::getInstance()->removeMonsterBullet((*it));
        (*it)->removeFromParent();
        if (it != monsterBullets.begin()) it--;
        //2.人物扣血
        Factory::getInstance()->wasHit(player, 100, false);
    }
    if (it != monsterBullets.end()) it++;
}
```

12. **角色死亡方面：**

难点：

1. Monster 打中玩家的判断，玩家如何进行扣血
2. 玩家血量降至 0 后的反应，如何判断游戏结束等
3. 游戏关卡的涉及，升级趣味性

解决方法：

1. 跟角色攻击子弹与怪物的碰撞类似，这里检测的是大怪物发射的“子弹”与玩家进行碰撞检测，另外检测小怪物是否靠近了角色，是否对角色进行攻击，列出攻击玩家的队列，然后根据攻击玩家的队列的 size 进行响应的扣血。玩家扣血，血量直接减少，到 0 自然就是死亡，然后每次扣血都有响应的受伤音效。

2. 角色死亡即血量到零，角色变成一个坟墓，不能再移动，也不能再射击，若单人模式，自然死了就游戏结束，双人模式要两个都死才能结束。另外为了允许玩家直接结束游戏，按键 T 可以直接返回菜单界面，并保留积分。

3. 涉及每一关的关卡，每一关的通关时间随着关卡数的增加而减小，难度越来越高，生存岛看看玩家在难度逐渐增加的同时，是如何实现高分的

核心代码：

```
// 被击中
void Factory::wasHit(Sprite* Player, int HP, bool isPlayer1) {
    if (isPlayer1) {
        GlobalVar::getInstance()->player1HP -= HP;
        if (GlobalVar::getInstance()->player1HP <= 0) removePlayer(Player);
        // 设置音效
        CocosDenshion::SimpleAudioEngine::getInstance()->playEffect("music/dead1.mp3", false);
    }
    else {
        GlobalVar::getInstance()->player2HP -= HP;
        if (GlobalVar::getInstance()->player2HP <= 0) removePlayer(Player);
        // 设置音效
        CocosDenshion::SimpleAudioEngine::getInstance()->playEffect("music/dead2.mp3", false);
    }

}
```

```
void Factory::removePlayer(Sprite* sp) {
    sp->stopAllActions();
    //GlobalVar::getInstance()->player1HP = 0;
    auto player_dead_texture = Director::getInstance()->getTextureCache()->addImage("grave.png");
    auto frame = SpriteFrame::createWithTexture(player_dead_texture, CC_RECT_PIXELS_TO_POINTS(Rect(0, 0, 32, 43)));
    //auto grave = Sprite::createWithSpriteFrame(frame);
    sp->setSpriteFrame(frame);
    //GlobalVar::getInstance()->player1_dead = true;
    //sp->setScale(1.0);
    //sp->removeFromParent();
    player.eraseObject(sp);

}
```

六、 项目总结

16340017 陈帆：

这次的项目因为与期末复习时间重叠，因此挺赶的，到最后我们实现的也只是一个“无敌版”，没有时间把之前想要的各种道具都加进去，可能再有一天时间就够了吧，觉得挺可惜的，但目前实现的也不少了，因此还是有挺多收获的：

1. 之前只是简单使用了 tileMap 没有像这次一样使用加了对象层、障碍层等，而且地图比屏幕大得多
2. 整合了之前玩家移动的思路，最终实现的玩家移动还是很流畅的，这种思路或许也会对以后有所帮助
3. 这次作业完成之后必须得学学用 git 来实现团队开发，这次作业过程中好像因为整合大家的代码出了点问题而导致出了一些 bug

16340015 陈彬彬：

这次大作业每个舍友都很用心，本身预想的游戏要更加复杂一点，武器的类型还包括围墙、油桶等，可以进行更深层次的游戏逻辑，两人游戏也可以更具有战略意义。后来因为时间的有限和逻辑的过于复杂，还是放弃了，稍微简化了一点，不过还是具有可玩性的。

要说学到了什么，其实更重要的是，这次大作业基本把之前所有的 cocos 作业，除了网络访问部分都涉及到了，让我基本都重温了一遍，印象深刻的有精灵动画序列的执行，帧动画的切割，还有最让人头疼的迭代器判断碰撞。总之收获还是蛮大的，终于也来到 20 周的末尾了，正式地准备告别高二了,希望以后的自己代码能力越来越好，学更多的知识来丰实自己。

16340016 陈冬禹：

本次项目中，我学习到了设计项目结构的重要性，一般需要使用面向对象的思想，尽可能地使程序模块化，降低代码的耦合度。这样可以隐藏细节，便于修改局部代码，不会牵一发而动全身。例如将子弹、怪物单独作为一个类。

其次，版本管理是协作开发的关键，不同成员的代码发生冲突会给合并带来麻烦，

应该选择合适的版本管理工具，避免冲突

16340014 车鑫恺：

这次项目是对整个 cocos2d 的总结学习，界面 UI 运用了精灵的创建，Lable 的创建，菜单项的使用，还有本地数据的存储，这里采用的是 UserDefault 来存储，界面的布局包括图标的大小、渲染的层次，各个项之间的相对位置及绝对位置，各个场景之间的切换等等。