《软件设计文档》

1. 软件技术选型理由

（1）开发平台

候选 1：Linux

候选 2：Windows

最终选择：Windows

选择原因：

A) Windows 的开发工具丰富，开发环境搭建难度较低

B) Windows 系统是大部分目标用户的常用系统，开发基于此系统的软件符合用户需求

（2）软件形式

候选 1：桌面应用

候选 2：WEB 应用

最终选择：桌面应用

选择原因：

A) 一般的游戏开发都是以桌面软件的形式，各种游戏引擎，开发工具库齐全且成熟

B) WEB 应用一般需要网络和服务器支持，游戏质量容易受到网络服务器的影响

C) WEB 应用需要依赖浏览器，需要进行不同版本的浏览器兼容这一额外工作

D) 我们的游戏是单人游戏，没有多人互动的信息交互需求

（3）框架选择

候选 1：Unity3D

候选 2：Cocos2d-x

最终选择：Cocos2d-x

选择原因：

A) Cocos2d-x 是一款开源的且十分成熟的游戏引擎软件，具有优秀的性能以及较为全面的技术文档，能够很好地满足我们的游戏开发需求。

B) Cocos2d-x 使用 C++进行游戏开发，语言门槛较低。

C) 小组成员都曾经专门学习使用 Cocos2d-x 进行游戏开发，有一定的编程经验。

2. 软件架构设计

游戏主进程由 Cocos2d-x 游戏引擎主导，负责各个自定义游戏场景类的调度和渲染。
场景类负责游戏场景的设计以及用户交互逻辑的实现，供 Cocos2d-x 游戏引擎调用。
资源类中的成员函数提供游戏资源的加载与控制的数据接口，供各场景类使用。
资源类中的变量用于保存已加载的游戏资源以及游戏实时状态信息。
游戏信息使用配置文件进行持久化，玩家信息使用 SQLite 数据库进行保存。

3. 软件模块划分

本作品可分为以下几个模块：

（1）游戏菜单模块：主要用于游戏页面的切换控制

（2）游戏主模块：主要进行与玩家的交互以及游戏逻辑的实现

（3）游戏音乐模块：主要用于实现游戏音乐的控制

（4）游戏砖块生成模块：主要用于生成游戏中的跳跃砖块

（5）游戏奖励生成模块：主要用于生成游戏中的奖励物品

（6）游戏数据库模块：主要用于保存游戏设置以及游戏玩家的相关信息

（7）游戏设置模块：主要用于设置游戏音乐等游戏内容
（8）游戏规则说明模块：主要用于显示游戏的规则说明等内容
（9）游戏排行榜模块：主要用于显示游戏最高历史得分等内容
（10）游戏提示模块：主要用于显示游戏提示信息等内容
（11）游戏资源模块：主要用于存放本项目所使用的图片，音乐等内容
（12）游戏自定义地图模块：主要用于让玩家自主设计游戏地图。
（13）游戏测试模块：主要用于测试数据库交互，音效播放以及游戏内物品的属性。

4. 软件设计技术

（1）Structure Programming（结构编程）

以 MainScene 中的 init 函数为例，其执行操作顺序如下：

基本块结构	代码示例
顺序结构	`void MainScene::initMap() {` `  auto blockPos = Vec2(origin.x, origin.y + 200);` `  auto blockSize = Size(visibleSize.width, 20);` `  auto block = BlockCreator::getInstance()->createBlock(0, blockPos, blockSize);` `  this->addChild(block);` `}`
选择结构	`void MainScene::scheduleRestTime(float dt)` `{` `  if (restTimeProgressTimer->getPercentage() == 0) {` `  jumpCount = 1;` `  }` `  else if (!isStop) {` `  auto percentage = restTimeProgressTimer->getPercentage();` `  percentage -= dt * 100 / shoeTime;` `  restTimeProgressTimer->setPercentage(percentage);` `  }` `}`
循环结构	`void BlockCreator::stopBlock()` `{` `  for (auto i = blockList.begin(); i != blockList.end(); i++) {` `  (*i)->getPhysicsBody()->setVelocity(Vec2(0, 0));` `  }` `}`

（2）Object-Oriented Programming（面向对象编程）

本作品中的每个功能模块都被抽象成一个类，而模块中的所有操作都被封装成每个类的成员函数，模块中的私有资源被封装成每个类的私有变量。

类名	Database
说明	用于与 SQL 数据库进行数据交互的模块。这个模块拥有的内部方法总共有三大类：一是用于保存玩家游戏状态，如：记录玩家的生命值，玩家的跳跃次数等；二是用于保存玩家得分，如：保存玩家本局游戏得分以及历史最高分等；三是用于查询玩家信息，如：获取玩家当前信息，玩家历史得分排行等。
代码	`class Database` `{` `public:` `  Database();` `  ~Database();` `  static Database* getInstance();` `  void createDatabase();` `  int getPlayerLife(int player);` `  void setPlayerLife(int player, int life);` `  int getPlayerJumpCount(int player);` `  void setPlayerJumpCount(int player, int jumpCount);` `  int getPlayerHighScore(int player);` `  void setPlayerHighScore(int player, int high);` `  int getPlayerCurrentScore(int player);` `  void setPlayerCurrentScore(int player, int score);` `  void resetPlayer(int player);` `  void addRank(int player, int score);` `  void refreshRank();` `  std::vector<std::vector<string>> getRank();` `private:` `  static Database* instance;` `  sqlite3* db;` `  bool is_new_record;` `};`

类名	Music
说明	用于管理游戏音乐资源。这个模块的操作主要有总共有两大类：一是游戏资源的加载，如：不同类型游戏音乐资源的加载；二是游戏音乐的播放控制，如：游戏音乐的暂停与播放，游戏音量的调节等。
代码	`class Music` `{` `public:` `  static Music* getInstance();` `  void preloadMusic();` `  void playMusic(int mode);` `  void setEffectVolume(float volumn);` `  void setBackgroundVolume(float volumn);` `  void stopCurrentEffectMusic();` `  void playBackgroundMusic();` `  void pauseBackgroundMusic();` `  void stopBackgroundMusic();` `private:` `  Music();` `  static Music* instance;` `  int backgroudSongId,` `  effectSongId;` `  float  MusicVolumn,` `  EffectVolumn;` `};`
类名	propsFactory
说明	用于游戏奖励物品的生成。这个模块的成员函数总共有三大类：一是游戏基础环境的设置，用于管理物品渲染时的基础场景；二是奖励物品的生成函数，用于生成如：飞鞋，金币等的奖励物品；三是奖励物品的更新函数，用于根据游戏进度实时更新已有的奖励物品的属性，使其能够被合理地渲染。
代码	`class propsFactory` `{` `public:` `  void setPhysicsWorld(PhysicsWorld* physicWorld);` `  void setLayer(Layer *layer);` `  static propsFactory* getInstance();` `  Vector<Sprite*> createRandomProps(Sprite* block, Layer* a);` `  Sprite* createGold(Vec2& pos, Layer* placeLayer = nullptr);` `  Sprite* createShoe(Vec2& pos, Layer* placeLayer = nullptr);` `  Sprite* createRocket(Vec2& pos, Layer* placeLayer = nullptr);` `  Sprite* createMedicine(Vec2& pos, Layer* placeLayer = nullptr);` `  void removeIfOut();` `  void removeAll();` `  void stopAll();` `  void speedUp(double multi);` `  void removeProps(Sprite* obj);` `  list<Sprite*> getPropsList();` `  void resetSpeed();` `  void resetGoldPosibilityRange();` `  void resetShoePosibilityRange();` `  void resetMedicinePosibilityRange();` `  void resetAll();` `private:` `  propsFactory();` `  const double  oriGoldPossibilityRange = 0.3,` `  oriShoePossibilityRange = 0.4,` `  oriMedicinePossibilityRange = 0.5;` `  double  shoePossibilityRange,` `  goldPossibilityRange,` `  medicinePossibilityRange,` `  speed,` `  medicineLimitSpeed = GameMinSpeed + 30;` `  PhysicsWorld * m_physicsWorld = NULL;` `  Layer* defaultLayer = NULL;` `  Sprite* createProps(Vec2& pos , const char path[] , Layer* placeLayer );` `  static propsFactory* instance;` `  list<Sprite*> propsList;` `};`
类名	BlockCreator
说明	用于生成游戏内的砖块。这个模块的功能分为三大类：一是新砖块的生成函数，根据一定规则生成新砖块；二是砖块的获取函数，这部分供主场景进行

	调用，用于获取已有砖块和新砖块的信息；三是砖块状态的更新函数，主要用于根据游戏进度调节砖块的生成，以及根据已有砖块的位置更新装块的位置。
代码	 ``` class BlockCreator { public: static BlockCreator* getInstance(); Sprite * createBlock(int mode, Vec2 & position, Size & size); Sprite * getBlock(int mode, Vec2 pos, Size size); void setNextBlock(int mode); void removeBlocksIfOut(); int getSize() { return blockList.size(); } void stopBlock(); void clearBlock(); std::list<Sprite*> getBlockList(); void speedUp(double multi); Sprite* changeBlockStyle(int mode, Sprite* block, Vec2& position, Size& size); private: BlockCreator(); //Layer * defaultLayer; static BlockCreator* instance; std::list<Sprite*> blockList; int status; Vec2 pos; Size size; double maxHeigtht; double speed; double g; double degree; Vec2 origin; Size visibleSize; }; ```
类名	RuleScene
说明	用于渲染规则说明页面的场景。这部分主要是有两部分的功能：一是重写了 Scene 类的场景渲染函数，绘制设置说明页面场景；二是为场景内部的按钮设置事件回调函数，响应用户输入。
代码	 ``` class RuleScene : public cocos2d::Scene { public: static cocos2d::Scene* createScene(); virtual bool init(); // a selector callback void backButtonCallback(cocos2d::Ref* pSender); // implement the "static create()" method manually CREATE_FUNC(RuleScene); }; ```
类名	SettingScene
说明	用于渲染游戏设置页面的场景。这部分主要是有三部分的功能：一是重写了 Scene 类的场景渲染函数，绘制设置页面场景；二是为场景内部的按钮设置事件回调函数，响应用户输入；三是游戏资源管理，将用户自定义的游戏设置保存到游戏文件中实现设置的持久化。

代码	 USING_NS_CC; class SettingScene : public cocos2d::Scene { public: static cocos2d::Scene* createScene(); static void createSettingFileIfNotExit(); static Dictionary* getSettingDict(); virtual bool init(); ui::Slider* SettingScene::createSlider(Vec2 sliderPos, int oriPercentage); Label* createLabel(Vec2 labelPos, std::string text, int fontSize = 30, Color3B textColor = Color3B(147, 68, 0)); void MusicVolumnSliderEvent(Ref* pSender, ui::Slider::EventType type); void EffectVolumnSliderEvent(Ref * pSender, ui::Slider::EventType type); // a selector callback void backButtonCallback(cocos2d::Ref* pSender); void saveButtonCallback(Ref* pSender); // implement the "static create()" method manually CREATE_FUNC(SettingScene); private: static std::string settingPath; Label *MusicVolumnText, *EffectVolumnText; ui::Slider *MusicVolumnSlider, *EffectVolumnSlider; int oldMusicVolumnPercentage, oldEffectVolumnPercentage; };
类名	RankScene
说明	用于渲染游戏排行榜页面的场景。这部分主要是有三部分的功能：一是重写了 Scene 类的场景渲染函数，绘制排行榜页面场景；二是为场景内部的按钮设置事件回调函数，响应用户输入；三是游戏数据交互，从数据库中查询用户的历史游戏得分并显示。
代码	 class RankScene : public cocos2d::Scene { public: static cocos2d::Scene* createScene(); virtual bool init(); Label* createLabel(Vec2 labelPos, std::string text, int fontSize = 30, Color3B textColor = Color3B(147, 68, 0)); // a selector callback void backButtonCallback(cocos2d::Ref* pSender); // implement the "static create()" method manually CREATE_FUNC(RankScene); private: int showRankNum = 8; };
类名	MenuScene
说明	用于渲染游戏菜单页面的场景。这部分主要是有三部分的功能：一是重写了 Scene 类的场景渲染函数，绘制菜单页面场景；二是为场景内部的按钮设置事件回调函数，响应用户输入，进行页面跳转。
代码	 class MenuScene : public cocos2d::Scene { public: static cocos2d::Scene* createScene(); virtual bool init(); // a selector callback void startButtonCallback(cocos2d::Ref* pSender); void ruleButtonCallback(cocos2d::Ref* pSender); void settingButtonCallback(cocos2d::Ref* pSender); void exitButtonCallback(cocos2d::Ref* pSender); void rankButtonCallback(cocos2d::Ref* pSender); // implement the "static create()" method manually CREATE_FUNC(MenuScene); };
类名	MapScene
说明	用于渲染游戏自定义地图的场景。这部分主要是有三部分的功能：一是重写

	了 Scene 类的场景渲染函数，绘制提示自定义地图场景；二是为场景内部的按钮和鼠标事件设置监听和事件回调函数，响应用户输入，进行游戏地图的设计。
代码	

```
enum DrawStates {
    DRAW_NONE,
    DRAW_READY,
    DRAW_ACTIVE,
    DRAW_SELECT,
    DRAW_MOVE,
};

class MapScene : public cocos2d::Scene
{
public:
    static cocos2d::Scene* createScene();

    virtual bool init();

    //绘制背景
    void drawBackground();

    //菜单
    void addMenu();

    //绘制基本地图元素
    void drawMapElement();

    //temp
    void print(string str);
    Label* printLabel;


    // implement the "static create()" method manually
    CREATE_FUNC(MapScene);
private:

    Vec2 origin;
    Size visibleSize;

    //绘制状态
    DrawStates state;
    DrawStates lastState;
    Point drawStart;

    //选取地图单元
    Sprite* selSprite;

    //当前绘制元素
    Sprite* drawSprite;

    //地图绘制单元
    vector<Sprite*> mapElement;

    // 地图实例
    MapClass map;


    //背景
    TMXTiledMap* background;

    //菜单
    Sprite* backphoto;
    Label* saveLabel;
    Label* loadLabel;

private:

    //是否在绘制区域
    bool MapScene::isInDrawLocal(const Point& loc);

    //添加监听
    void addListener();

    //鼠标监听
    bool onTouchBegan(Touch* touch, Event* event);
    bool onTouchMoved(Touch* touch, Event* event);
    bool onTouchEnded(Touch* touch, Event* event);

    //地图加载和保存
    void loadMapButtonCallback();
    void saveMapButtonCallback();

    //键盘监听
    void onKeyPressed(EventKeyboard::KeyCode code, Event * event);
    void onKeyReleased(EventKeyboard::KeyCode code, Event * event);

    // 坐标转换
    static inline Point localConvert(const Point& loc) {
        int w = int(loc.x) / MapScene::board;
        int h = int(loc.y) / MapScene::board;
        return Point(w*MapScene::board, h*MapScene::board);
    }
    static const int board = 32;
};
```

类名	Hint
说明	用于渲染游戏提示信息页面的场景。这部分主要是有三部分的功能：一是重写了 Scene 类的场景渲染函数，绘制提示信息页面场景；二是为场景内部的按钮设置事件回调函数，响应用户输入，进行页面跳转。
代码	`class Hint` `{` `public:` `    static Hint* getInstance();` `    void RestartHint(Vec2& origin, Size& visibleSize, Layer* defaultLayer);` `    void restartButtonCallback(Ref* sender);` `    void backMenuButtonCallback(Ref* sender);` `private:` `    Hint();` `    static Hint* instance;` `};`
类名	MainScene
说明	用于渲染游戏主界面的场景。这部分主要是有三部分的功能：一是重写了 Scene 类的场景渲染函数，绘制游戏主页面场景；二是为场景内部的按钮设置事件回调函数，响应用户输入，进行页面跳转；三是游戏资源控制，负责与各资源模块进行数据交互，实时更新游戏内物品的状态和属性。
代码	`class MainScene : public cocos2d::Layer` `{` `public:` `    void setPhysicsWorld(PhysicsWorld * world);` `    static cocos2d::Scene* createScene();` `    virtual bool init();` `    void scheduleRestTime(float dt);` `    void createPlayer();` `    void createMap();` `    void createBackground();` `    void createRestTimeBar();` `    void setJoint();` `    void addListener();` `    void removeListener();` `    void explosion(Vec2 pos, float radius);` `    bool onContactBegin(PhysicsContact & contact);` `    bool onContactEnd(PhysicsContact & contact);` `    bool pairMatch(int a1, int a2, int b1, int b2);` `    void onKeyPressed(EventKeyboard::KeyCode code, Event * event);` `    void onKeyReleased(EventKeyboard::KeyCode code, Event * event);` `    void updateMap(float dt);` `    void StopAction(float dt);` `    void updateScoreLabel();` `    int getCurrentScore();` `    void updateHighScoreLabel();` `    void initMap();` `    void contactGold();` `    void contactShoe();` `    void contactMedicine();` `    void speedUp(float dt);` `    void changeSpeed(double multi);` `    void resetShoePosibilityRange(float dt);` `    void resetMedicinePosibilityRange(float dt);` `    // implement the "static create()" method manually` `    CREATE_FUNC(MainScene);` `private:` `    PhysicsWorld * m_physicWorld;` `    Layer* worldLayer;` `    Vec2 origin;` `    Size visibleSize;` `    float brickSize = 13;` `    int canJump;` `    bool isStop = false;` `    float playerleftOfRightVel = 500;` `    Sprite* player;` `    std::list<Sprite*> blockList;` `    cocos2d::Vector<SpriteFrame*> playerRun;` `    float playerVelocity;` `    const float maxSpeed = 200,` `        minSpeed = 800;` `    int jumpCount;` `    int Money;` `    int highScore;` `    int life;` `    int difficulty = 0;` `    float shoeTime = 15;` `    bool isOnBoard = true;` `    // 标签` `    Label* blocknum;` `    Label* MostHigh;` `    Label* currentScore;` `    ProgressTimer* restTimeProgressTimer;` `};`

类名	AppDelegate
说明	游戏主控制类。负责游戏窗口的建立以及场景的初始化。
代码	`class AppDelegate : private cocos2d::Application` `{` `public:` `AppDelegate();` `virtual ~AppDelegate();` `virtual void initGLContextAttrs();` `/**` `@brief Implement Director and Scene init code here.` `@return true Initialize success, app continue.` `@return false Initialize failed, app terminate.` `*/` `virtual bool applicationDidFinishLaunching();` `/**` `@brief Called when the application moves to the background` `@param the pointer of the application` `*/` `virtual void applicationDidEnterBackground();` `/**` `@brief Called when the application reenters the foreground` `@param the pointer of the application` `*/` `virtual void applicationWillEnterForeground();` `};`

（3）Design Patterns（设计模式）

项目中对资源管理模块均使用了单例模式，场景渲染模块则使用了工厂模式。

设计模式	样例（节选）	说明
单例模式	`Database* Database::instance = nullptr;` `Database* Database::getInstance() {` `if (instance == nullptr) {` `instance = new Database();` `}` `return instance;` `}`	使用一个静态 Database 类指针。外部函数可以通过调用 Database::getInstance()这个静态方法获取全局唯一的 Database 类实体。
工厂模式	`class RuleScene : public cocos2d::Scene` `{` `public:` `static cocos2d::Scene* createScene();` `virtual bool init();` `// a selector callback` `void backButtonCallback(cocos2d::Ref* pSender);` `// implement the "static create()" method manually` `CREATE_FUNC(RuleScene);` `};`	每个场景渲染类都继承了 cocos2d::Scene 这个基类，不同的场景类通过重载基类中的 init()函数实现不同场景的渲染。

5. 外部软件库以及资源

（1）Cocos2d-x 游戏引擎程序文件（cocos2d.h/main.cpp/main.h/resource.h）

（2）Sqlite3 开源数据库引擎程序文件（sqlite3.h/sqlite3.cpp）

6. 测试

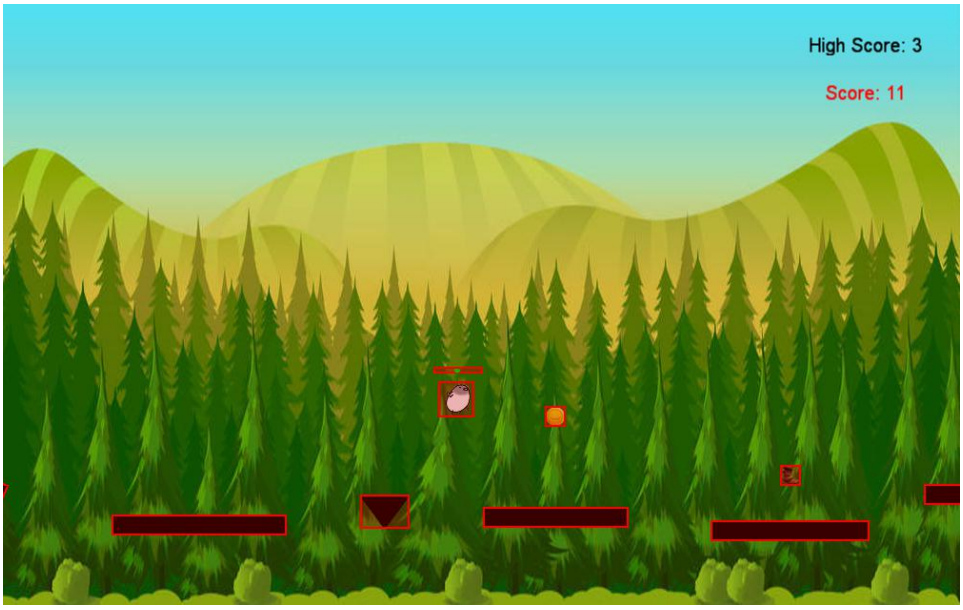
本项目的测试主要分为两个部分：数据测试和画面测试

数据测试部分主要测试玩家数据与数据库的交互以及音乐资源的加载和处理函数，使用 Test 类进行测试；画面测试主要测试游戏物品和游戏逻辑的实现，使用 Cocos2d-x 内置的 DEBUG 模式进行测试。

数据测试部分				
测例	测例说明	测例代码	测例输出	测试结果

1	测试玩家生命值保存与读取	// 测试生命值记录 Database::getInstance()->setPlayerLife(0, 1); CCLOG("life: %d", Database::getInstance()->getPlayerLife(0));	set life ok! life: 1	通过
2	测试玩家跳跃次数保存于读取	// 测试跳跃次数记录 Database::getInstance()->setPlayerJumpCount(0, 2); CCLOG("JumpCount: %d", Database::getInstance()->getPlayerJumpCount(0));	set jump count ok! JumpCount: 2	通过
3	测试玩家历史最高分的保存与读取	// 测试最高分纪录 Database::getInstance()->setPlayerHighScore(0, 3); CCLOG("HighScore: %d", Database::getInstance()->getPlayerHighScore(0));	set high score ok! HighScore: 3	通过
4	测试玩家本局得分的保存与读取	// 测试本次得分纪录 Database::getInstance()->setPlayerCurrentScore(0, 4); CCLOG("CurrentScore: %d", Database::getInstance()->getPlayerCurrentScore(0));	set high score ok! CurrentScore: 4	通过
5	测试排行榜的输入与读取	// 测试排行榜记录 Database::getInstance()->addRank(0, 1); Database::getInstance()->addRank(0, 8); Database::getInstance()->addRank(0, 4); std::vector<std::vector<string>> rank = Database::getInstance()->getRank(); CCLOG("RANK LIST"); for (int i = 0; i < rank.size(); i++) { CCLOG("%s %s", rank[i][0].c_str(), rank[i][1].c_str()); }	insert rank ok! insert rank ok! insert rank ok! RANK LIST 0 8 0 4 0 1	通过
6	测试背景音效的播放	case 4: // 测试背景音乐播放 Music::getInstance()->playBackgroundMusic(); break;	AudioPlayer::play2d, _alSource: 2, player id=13 AudioPlayer::destroy begin, id=13 Exit rotate buffer thread ... cocos2d::experimental::AudioPlayer::rotateBufferThread exited	通过
7	测试背景音效的暂停	case 5: // 测试背景音乐暂停 Music::getInstance()->pauseBackgroundMusic(); break;	背景音乐停止	通过
8	测试背景音效的停止	case 6: // 测试背景音乐停止 Music::getInstance()->stopBackgroundMusic(); break;	rotateBufferThread exited! Before alSourceStop Before alSourcei AudioPlayer::destroy end, id=13 ~AudioPlayer() (14FC4AB0), id=13	通过
9	测试特效音效的播放	default: // 测试特效音乐播放 Music::getInstance()->playMusic(mode); break;	AudioPlayer::play2d, _alSource: 2, player id=14 AudioPlayer::destroy begin, id=14 Before alSourceStop Before alSourcei AudioPlayer::destroy end, id=14 ~AudioPlayer() (14FC3EA8), id=14	通过
10	测试特效音效的停止	case 7: // 测试特效音乐停止 Music::getInstance()->stopCurrentEffectMusic(); break;	AudioPlayer::play2d, _alSource: 1, player id=6 ~AudioPlayer() (10BA4980), id=6 AudioPlayer::destroy begin, id=6 Before alSourceStop Before alSourcei AudioPlayer::destroy end, id=6 AudioPlayer::play2d, _alSource: 1, player id=7 ~AudioPlayer() (10BA4980), id=7 AudioPlayer::destroy begin, id=7 Before alSourceStop Before alSourcei AudioPlayer::destroy end, id=7 AudioPlayer::play2d, _alSource: 1, player id=8 ~AudioPlayer() (10BA4980), id=8 AudioPlayer::destroy begin, id=8 Before alSourceStop Before alSourcei AudioPlayer::destroy end, id=8 AudioPlayer::play2d, _alSource: 1, player id=9 ~AudioPlayer() (10BA4980), id=9 AudioPlayer::destroy begin, id=9 Before alSourceStop Before alSourcei AudioPlayer::destroy end, id=9	通过

画面测试部分	
测试代码	// Debug 模式 if (DEBUG) { scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL); Test::getInstance()->OpenTestMode(); }

测试画面	
测试结果	砖块、玩家、奖励物品、计分板均能正常显示，程序能够正确响应用户的键盘输入，测试通过

7. 特别说明

本项目使用现代操作系统应用开发的期末项目作为本课程设计，代码和文档均为本小组成员原创。