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
15331262	饶宇熹	组长	上午班	物理属性设置、UI 美化、视频录制	25%
15331281	苏惠玲	组员	上午班	玩家控制改进、项目优化、实现两个关卡	25%
15331236	马朝露	组员	上午班	动画设置、项目优化、素材寻找、裁剪	25%
15331242	明友芬	组员	上午班	实现三个关卡、寻找素材、撰写实验报告	25%

二、开发环境

操作系统 ： Windows10

开发 IDE ： Visual Studio 2015

开发语言 ： C++

游戏引擎 ： cocos2d-x-3.10

三、 项目阐述

名称 ： 趣味推箱子

简介 ： 经典的推箱子是一个来自日本的古老游戏，目的是在训练你的逻辑思考能力。在一个狭小的仓库中，要求把木箱放到指定的位置，稍不小心就会出现箱子无法移动或者通道被堵住的情况，所以需要巧妙的利用有限的空间和通道，合理安排移动的次序和位置，才能顺利的完成任务。

本次期末作品在经典推箱子的基础上进行改良创新，新增了先找到灯泡在将箱子推到灯泡处等趣味功能，通关方法奇特。

主要运用了以下知识点：

(1) Tiledmap —— 利用 Tiledmap 工具，制作推箱子地图；

(2) 动画帧 —— 玩家的移动等；

(3) 事件处理 —— 碰撞、墙体等；

(4) 音效 —— 预加载背景音和动作音效；

(5) 物理引擎 —— 将玩家和箱子设置为刚体，实现玩家推箱子、碰撞检测等；

玩法： 点击屏幕上方 help，显示游戏玩法。玩家需要将箱子放到指定的位置。

1、 方向控制：

（1） 方向键左←屏幕上的 A：左移；

（2） 方向键→或屏幕上的 A：右移；

（3） 方向键↑或屏幕上的 A：上移；

（4） 方向键↓或屏幕上的 A：下移。

2、 玩家可在地图的空地上自由行走。当玩家与箱子解除时，可推动箱子。箱子和玩家无法穿过墙体。

3、 设置了重玩按钮，当游戏进入死角时可以点击重玩按钮冲完本关。

亮点： 1、 在经典推箱子的基础上进行改良创新，有声控灯泡、陷阱、趣味寻找灯泡等趣味功能；

2、 通关方法奇特，部分指定位置隐藏在墙体下面；

3、一共有 6 关，难度从小到大排列，有的关卡需要玩家脑洞大开才能寻找到灯泡的位置然后将箱子推到制定的位置才能继续下一关。

四、 项目展示

1、 项目框架以及初始界面

项目整体框架图如下图 1 所示：

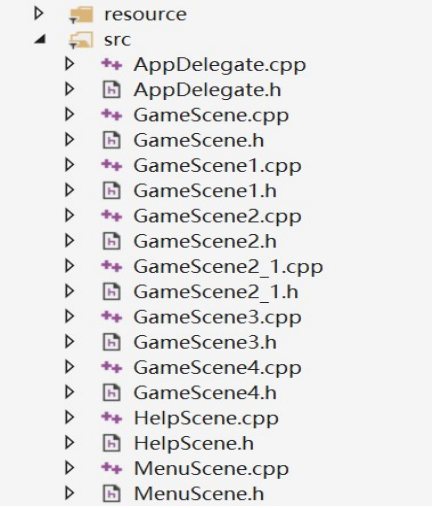

图 1

一共有 5 个游戏界面，一个帮助界面和一个开始界面组成，在开始界面可以查看

游戏的规则并且开始游戏，每一个关卡完成之后可以选择重完本关或者进入下一个关卡。游戏的初始界面如下图 2 所示：

图 2

点击 help 按钮之后会有简单的游戏介绍如何实现的，如图 3 所示：

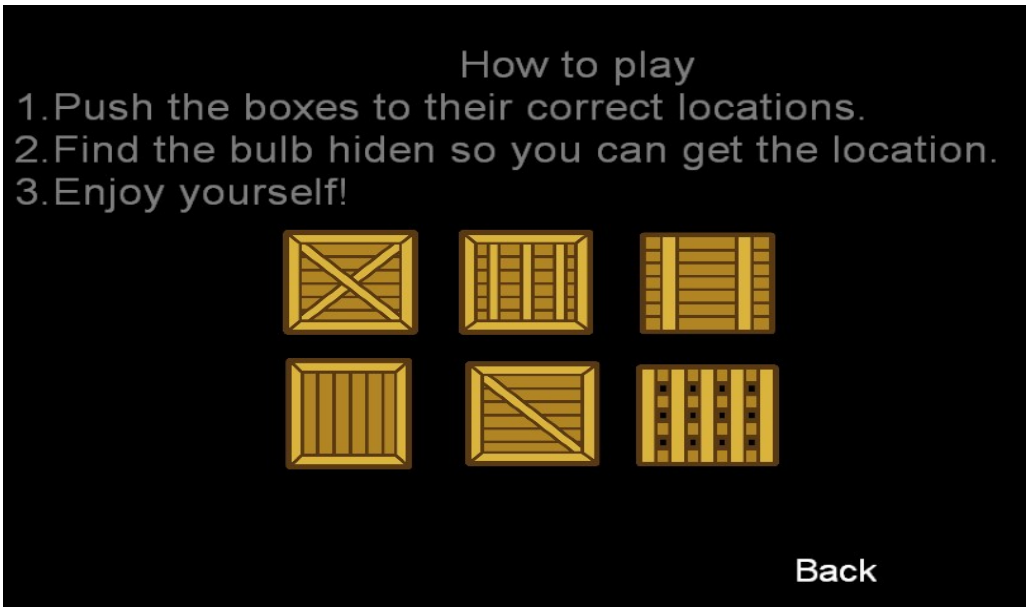

图 3

点击 help 和 start 的对应代码实现如下图 4 所示，点击按钮之后就会跳转到对应的界面完成操作：

```
void MenuScene::helpReplaceScene(Ref* pSender)
{
    SimpleAudioEngine::getInstance()->playEffect("music/kaca.mp3", false);
    Director::getInstance()->replaceScene(TransitionSlideInR::create(1, HelpScene::createScene()));

#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    exit(0);
#endif
}

void MenuScene::startReplaceScene(Ref* pSender)
{
    SimpleAudioEngine::getInstance()->playEffect("music/music2.mp3", false);
    Director::getInstance()->replaceScene(TransitionSlideInR::create(1, GameScene ::createScene()));

#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    exit(0);
#endif
}
```

图 4

2、 游戏第一关展示及重点代码分析

跳转到第一关游戏开始时界面如图 5 所示， 第一关比较简单，直接就将灯泡显示出来了，玩家只需要将箱子推到灯泡处即可过关，在轻快的背景音乐下，玩家可以选择点击屏幕左下方的 W、S、A、D 字样控制玩家的移动，也可以按键盘上的方向控制按钮控制玩家的移动，如果玩家很不幸将箱子推进了死角，玩家可以点击 Restart 按钮，重玩本关。

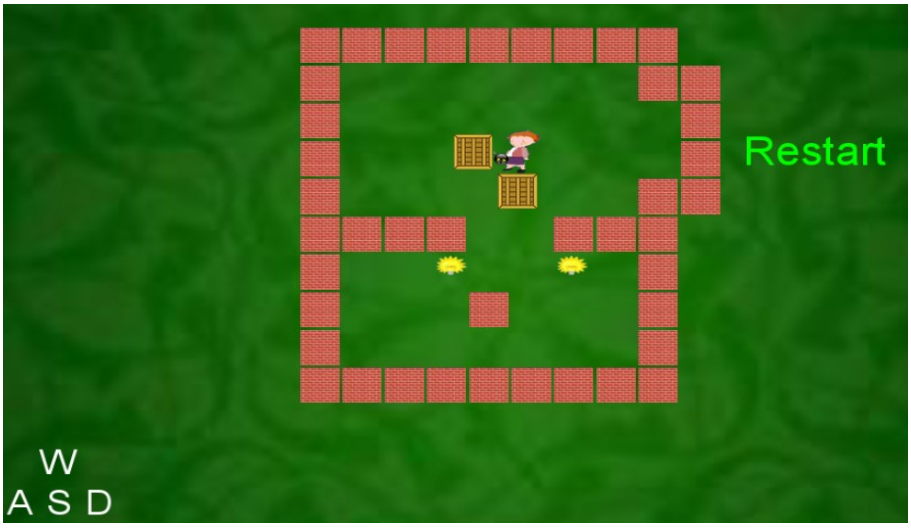

图 5

重点代码分析：

首先总览一下第一关的情况，第一关的私有成和公有函数如下图 6 所示：

```
public:
    static cocos2d::Scene* createScene();
    virtual bool init();
    static PhysicsWorld* world;
    void setPhysicsWorld(PhysicsWorld * world);
    void addPlayer();
    void addBox();
    void addWall();
    void moveUp(Ref* sender);
    void moveDown(Ref* sender);
    void moveLeft(Ref* sender);
    void moveRight(Ref* sender);
    void addKeyboardListener();
    void onKeyPressed(EventKeyboard::KeyCode code, Event * event);
    void onKeyReleased(EventKeyboard::KeyCode code, Event * event);
    void judgeWin(float dt);
    void nextCallback(Ref * pSender);
    void restartCallback(Ref * pSender);
    CREATE_FUNC(GameScene);

private:
    cocos2d::Size visibleSize;
    PhysicsWorld* m_world;
    cocos2d::Vec2 origin;
    cocos2d::Label* time;
    int dtime;
    cocos2d::ProgressTimer* pT;
    char lastCid;
    cocos2d::Sprite* player;
    cocos2d::Sprite* box1;
    cocos2d::Sprite* box2;
    cocos2d::Sprite* light1;
    cocos2d::Sprite* light2;
    cocos2d::Vector<SpriteFrame*> run;
```

图 6

这之后的几关都是在第一关的基础上更改的，所以之后的几关就不再重复这些内容，只讲更改的部分。界面的初始化部分代码就不贴上来了，比较简单。

添加玩家，并设置玩家的物理属性如下图 7 所示：

```
void GameScene::addPlayer() {
    //使用第一帧创建精灵
    player = Sprite::create("3.png");
    player->setPosition(480, 400);
    addChild(player, 3);

    // 设置角色刚体属性
    player->setPhysicsBody(PhysicsBody::createBox(Size(48, 48), PhysicsMaterial(1.0f, 0.0f, 0.001f)));
    player->getPhysicsBody()->setCategoryBitmask(2);
    player->getPhysicsBody()->setCollisionBitmask(2);
    player->getPhysicsBody()->setContactTestBitmask(2);
    player->getPhysicsBody()->setRotationEnable(false);  // 防止小人旋转
}
```

图 7

添加箱子，并且设置箱子的物理属性，两个箱子的设置除了位置以外，其余的基本是一样的：

```
void GameScene::addBox() {
    // 创建箱子
    box1 = Sprite::create("box1.png");
    box1->setPosition(480, 362); //第二下
    box1->setPhysicsBody(PhysicsBody::createBox(box1->getContentSize(), PhysicsMaterial(1.0f, 0.0f, 0.001f)));
    box1->getPhysicsBody()->setCategoryBitmask(3);
    box1->getPhysicsBody()->setCollisionBitmask(3);
    box1->getPhysicsBody()->setContactTestBitmask(3);
    box1->getPhysicsBody()->setRotationEnable(false);
    addChild(box1, 3);
```

图 8

创建墙体，并且设置墙体的物理属性，如图 9 所示，四堵墙的设置基本是一样的：

```
void GameScene::addWall() {
    // 创建墙体
    //左墙体
    for (int i = 0; i < 10; i++) {
        auto wall = Sprite::create("wall.jpg");
        wall->setScale(0.7f);
        wall->setPosition(300, 162 + 55 * 0.7*i);
        wall->setPhysicsBody(PhysicsBody::createBox(Size(55, 55), PhysicsMaterial(10000.0f, 1.0f, 1.0f)));
        wall->getPhysicsBody()->setCategoryBitmask(2);
        wall->getPhysicsBody()->setCollisionBitmask(2);
        wall->getPhysicsBody()->setContactTestBitmask(2);
        wall->getPhysicsBody()->setRotationEnable(false);  // 防止旋转
        addChild(wall, 1);
    }
```

图 9

控制人儿的上下左右移动，如图 10 所示：

```
void GameScene::moveUp(Ref* sender) {
    auto animation = Animation::createWithSpriteFrames(run, 0.05f);
    auto animate = Animate::create(animation);
    float x = player->getPosition().x;
    float y = player->getPosition().y;
    player->runAction(MoveTo::create(0.2, Vec2(x, y + 43)));

    player->runAction(animate);
}
```

图 10

添加事件的侦听器，如图 11 所示：

```
// 添加键盘事件监听器
void GameScene::addKeyboardListener() {
    auto keyboardListener = EventListenerKeyboard::create();
    keyboardListener->onKeyPressed = CC_CALLBACK_2(GameScene::onKeyPressed, this);
    keyboardListener->onKeyReleased = CC_CALLBACK_2(GameScene::onKeyReleased, this);
    this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(keyboardListener, this);
}
```

图 11

控制键盘的按下以及放开的函数的处理如图 12 所示：

```
void GameScene::onKeyPressed(EventKeyboard::KeyCode code, Event* event) {
    switch (code) {
    case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
    case EventKeyboard::KeyCode::KEY_CAPITAL_A:
    case EventKeyboard::KeyCode::KEY_A:
        moveLeft(NULL);
        break;

void GameScene::onKeyReleased(EventKeyboard::KeyCode code, Event* event) {
    switch (code) {
    case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
    case EventKeyboard::KeyCode::KEY_A:
    case EventKeyboard::KeyCode::KEY_CAPITAL_A:
    case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
    case EventKeyboard::KeyCode::KEY_D:
    case EventKeyboard::KeyCode::KEY_CAPITAL_D:
    case EventKeyboard::KeyCode::KEY_UP_ARROW:
    case EventKeyboard::KeyCode::KEY_CAPITAL_W:
    case EventKeyboard::KeyCode::KEY_W:
    case EventKeyboard::KeyCode::KEY_DOWN_ARROW:
    case EventKeyboard::KeyCode::KEY_CAPITAL_S:
    case EventKeyboard::KeyCode::KEY_S:
        //isMove = false;
        break;
    }
}
```

图 12

对于是否进入下一关的判断，如下图 13 所示：

```
void GameScene::judgeWin(float dt) {
    if (light1 != nullptr && light2 != nullptr) {
        if  (((box1->getBoundingBox().intersectsRect(light1->getBoundingBox()))&&
            (box2->getBoundingBox().intersectsRect(light2->getBoundingBox())))||
            ((box1->getBoundingBox().intersectsRect(light2->getBoundingBox()))&&
            (box2->getBoundingBox().intersectsRect(light1->getBoundingBox())))) {

            // 关闭调度器
            unschedule(schedule_selector(GameScene::judgeWin));
            // 设置下一关按钮
            auto label2 = Label::createWithTTF("Next", "fonts/arial.ttf", 40);
            label2->setColor(Color3B::GREEN);
            auto nextBtn = MenuItemLabel::create(label2, CC_CALLBACK_1(GameScene::nextCallback, this));
            Menu* next = Menu::create(nextBtn, NULL);
            next->setPosition(750 + origin.x / 2,  origin.y + 300);
            this->addChild(next);

        }
    }
}
```

图 13

图 14 展示了下一关的回调函数和重玩函数：

```
// 下一关按钮响应函数.
void GameScene::nextCallback(Ref * pSender) {
    Director::getInstance()->replaceScene(GameScene1::createScene());
}
// 重玩按钮响应函数.
void GameScene::restartCallback(Ref * pSender) {
    Director::getInstance()->replaceScene(GameScene::createScene());
}
```

图 14

以上就是代码的基本分析，比较难实现的就是对于物理属性的使用的那一块。

第一关完成之后的界面如下图 15 所示，选择 next 继续冲关：

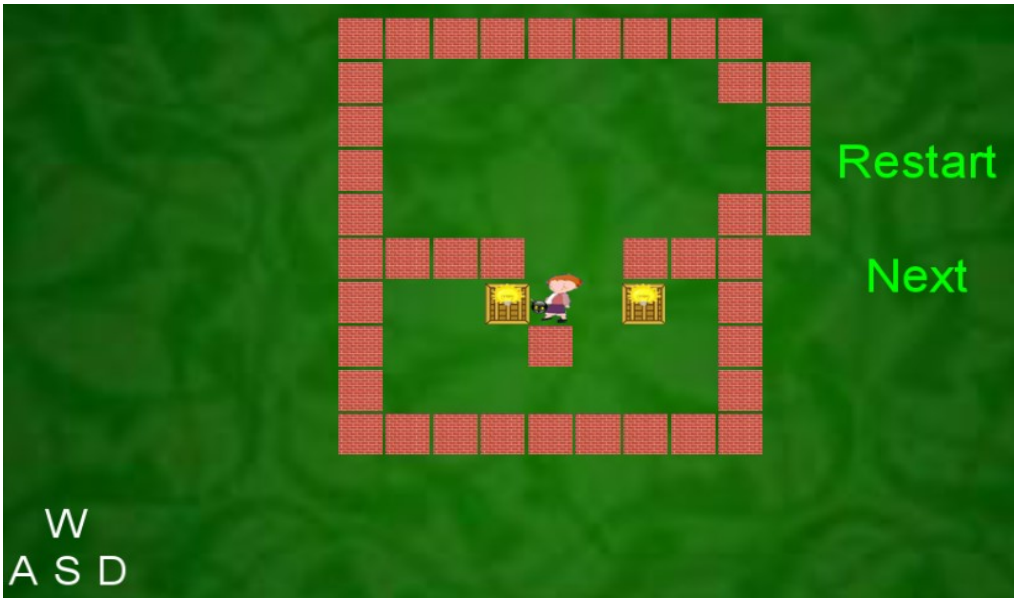

图 15

3、 第二关基本情况展示

第二关初始界面如下图 16 所示，在图 16 中并没有灯泡来让玩家将箱子推动到指定的位置，这就需要玩家自己想办法了：

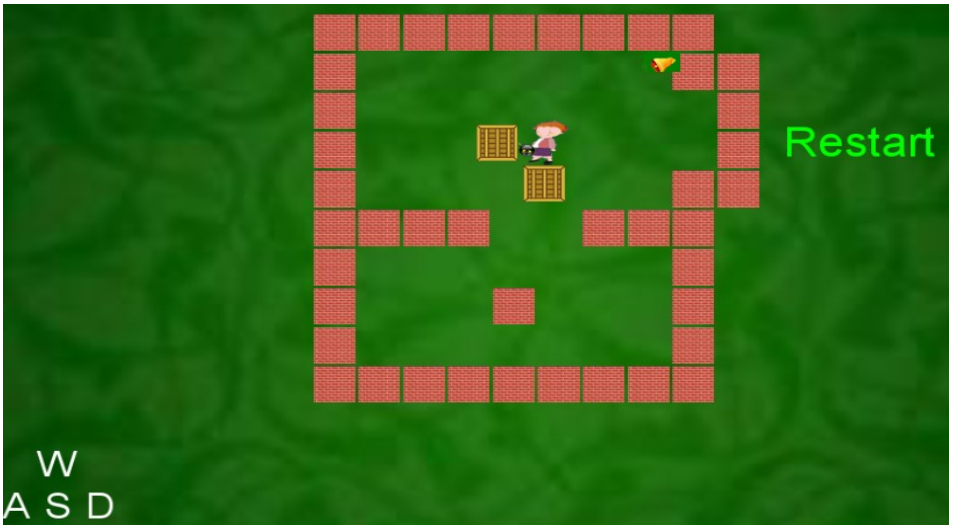

图 16

第二关是一个声控的开关，玩家需要做的首先是要到铃铛的位置摇摇铃铛，产生声音来让声控灯亮起来。如何控制灯亮呢，这需要借助三个函数来实现：

```
void meet(EventCustom* event);
void addCustomListener();
void update(float f);
```

Meet 函数用来检测图中小人儿与铃铛的位置，addCustomListener()函数用来添加自定义的侦听器监听小人儿的移动情况，update 函数则实时的更新界面，具体实现如下图 17 所示：

```
void GameScene1::meet(EventCustom* event) {
    int distance = 0;
    distance = bell->getPosition().getDistance(player->getPosition());
    if (distance < 20) {
        light1 = Sprite::create("light.png");
        light1->setPosition(420, 280); //第左
        addChild(light1, 3);

        light2 = Sprite::create("light.png");
        light2->setPosition(530, 280); //第三左
        addChild(light2, 3);
    }
}

void GameScene1::addCustomListener() {
    auto meetListener = EventListenerCustom::create("meet", CC_CALLBACK_1(GameScene1::meet, this));
    _eventDispatcher->addEventListenerWithFixedPriority(meetListener, 1);
}

void GameScene1::update(float f) {
    EventCustom e("meet");
    _eventDispatcher->dispatchEvent(&e);
}
```

图 17

在使用自定义的监听器和添加调度器之后，在 restart 函数和 next 函数里面要相应的将监听器和调度器给关闭掉，否则函数会在点击 next 按钮的时候出现断点如下图 18 所示：

```
void GameScene1::nextCallback(Ref * pSender) {
    //关闭调度器和侦听器
    unschedule(schedule_selector(GameScene1::update));
    this->getEventDispatcher()->removeAllEventListeners();
    Director::getInstance()->replaceScene(GameScene2::createScene());
}
// 重玩按钮响应函数.
void GameScene1::restartCallback(Ref * pSender) {
    unschedule(schedule_selector(GameScene1::update));
    this->getEventDispatcher()->removeAllEventListeners();
    Director::getInstance()->replaceScene(GameScene1::createScene());
}
```

图 18

4、 第三关基本情况展示

第三关初始界面如下图 19 所示,界面中最开始的时候也是没有灯泡的，需要玩家自己寻找：

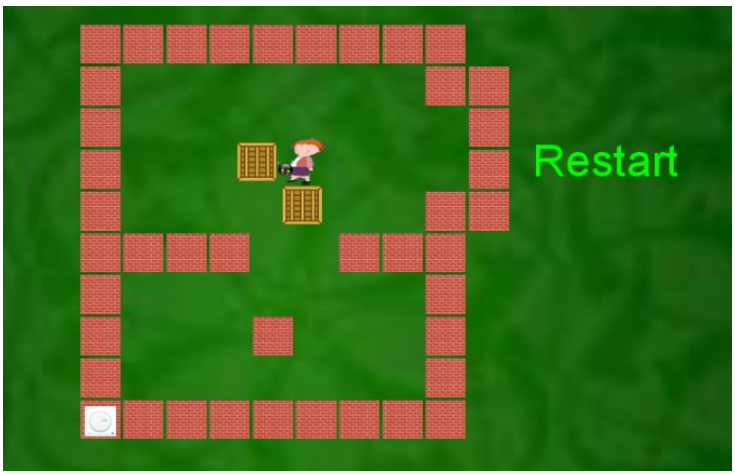

图 19

第三关是开关控制灯亮，需要玩家点击开关，控制灯亮之后，将箱子推动到灯泡的位置，关于开关的实现和点击开关灯亮的实现如下图 20 所示：

```
Switch = MenuItemImage::create("switch.png", "switch.png", CC_CALLBACK_1(GameScene2::SwitchScene, this));
auto menu2 = Menu::create(Switch, NULL);
menu2->setPosition(300, 160);
this->addChild(menu2, 1);

void GameScene2::SwitchScene(cocos2d::Ref* pSender) {
    light2 = Sprite::create("light.png");
    light2->setPosition(530, 280); //第三左
    addChild(light2, 3);

    light1 = Sprite::create("light.png");
    light1->setPosition(350, 200);
    addChild(light1, 3);
}
```

图 20

5、 第四关基本情况展示

第四关初始界面如下图 21 所，第四关一开始就显示了两个灯泡，但是新增加了一个陷阱，玩家在推箱子的过程中需要小心的避开陷阱，否则会掉进陷阱，游戏就结束了：

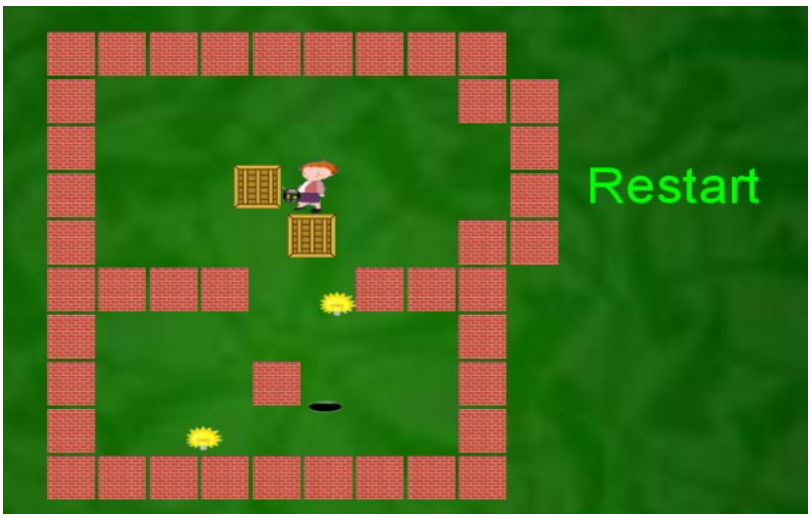

图 21

关于玩家与陷阱距离的监测实现方式与第二关的铃铛的实现方式类似，但是在实现的时候玩家在碰到陷阱的时候有一个掉进陷阱的动画，实现方式如图 22 所示：

```
void GameScene2_1::meet(EventCustom* event) {
    int distance = 0;
    distance = trap->getPosition().getDistance(player->getPosition());
    if (distance < 10) {
        //将player从画面上移走；
        player->runAction(ScaleTo::create(0.4, 0.0f));
        auto gameover = Sprite::create("gameOver1.png");
        gameover->setAnchorPoint(Vec2(0.5, 0.5));
        gameover->setPosition(visibleSize / 2);
        this->addChild(gameover, 5);
        return;
    }
}
```

图 22

其余的关于侦听器和调度器的使用与第二关的是一样的。

6、 第五关基本情况展示

第五关初始界面如图 23 所示，这一关玩家需要打开脑洞，找到另一个灯泡的位置：

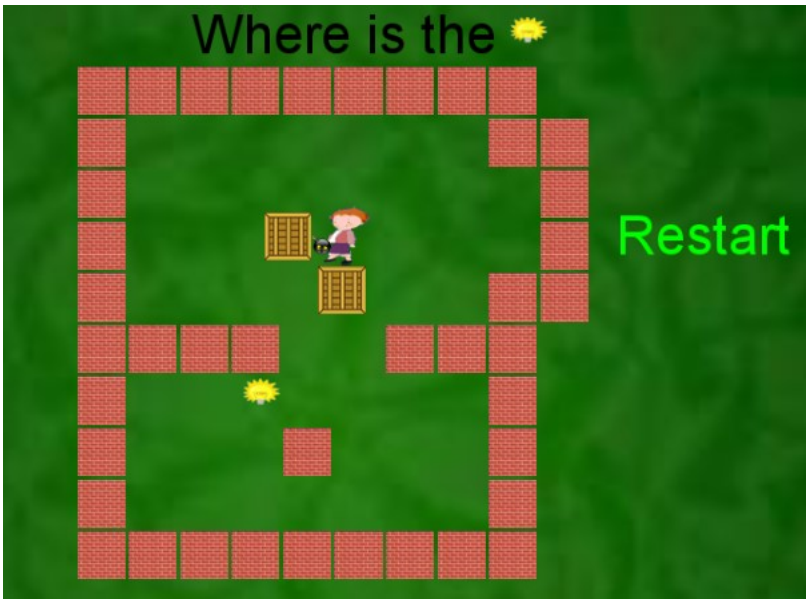

图 23

这一关需要玩家打破成规，事实上另一个灯泡就是图中英文字母后面的那个，这里设置了上面那堵墙有一个地方是可以穿过的，这个地方比较新颖。

更改墙，可以穿过墙的实现如下图 24 所示：

```
//上墙体
for (int i = 1; i < 8; i++) {
    auto wall = Sprite::create("wall.jpg");
    wall->setScale(0.7f);
    wall->setPosition(300 + 55 * 0.7*i, 508.5);
    wall->setPhysicsBody(PhysicsBody::createBox(Size(55, 55), PhysicsMaterial(10000.0f, 1.0f, 1.0f)));
    wall->getPhysicsBody()->setCategoryBitmask(2);
    wall->getPhysicsBody()->setCollisionBitmask(2);
    wall->getPhysicsBody()->setContactTestBitmask(2);
    wall->getPhysicsBody()->setRotationEnable(false);  // 防止旋转
    if (i == 5 || i == 4) {
        wall->getPhysicsBody()->setCategoryBitmask(0);
        wall->getPhysicsBody()->setCollisionBitmask(0);
        wall->getPhysicsBody()->setContactTestBitmask(0);
    }
    addChild(wall, 1);
}
```

图 24

7、 第六关基本情况展示

第六关两个箱子是重叠的，玩家需要想办法将箱子分开，然后推到对应的位置，如图 25 所示：

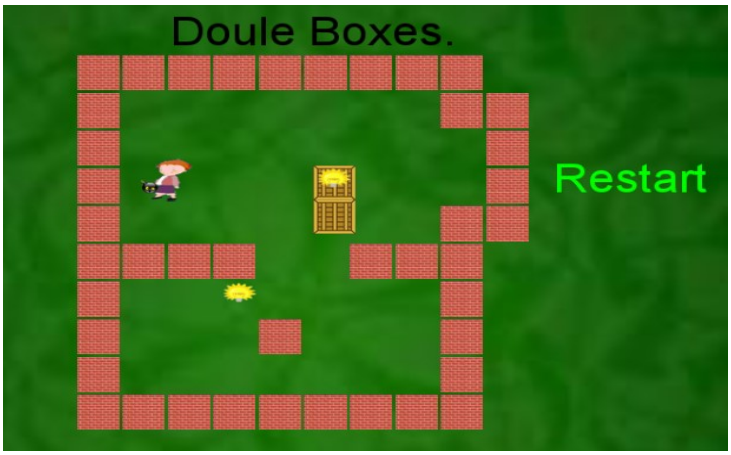

图 25

这一关真的需要大开脑洞，很多玩家一开始很可能会想法设法将两个箱子分开，课实际上两个箱子是不能分开的，这里需要做的是移动灯泡，将灯泡移动到箱子的位置。

代码实现如图 26 所示：

```
void GameScene4::addBox() {
    // 创建箱子
    box1 = Sprite::create("box1.png");
    box1->setPosition(500, 362); //第二下
    box1->setPhysicsBody(PhysicsBody::createBox(box1->getContentSize(), PhysicsMaterial(10.0f, 0.0f, 1.001f)));
    box1->getPhysicsBody()->setCategoryBitmask(3);
    box1->getPhysicsBody()->setCollisionBitmask(3);
    box1->getPhysicsBody()->setContactTestBitmask(3);
    box1->getPhysicsBody()->setRotationEnable(false);
    addChild(box1, 3);

    box2 = Sprite::create("box1.png");
    box2->setPosition(500, 396);
    box2->setPhysicsBody(PhysicsBody::createBox(box2->getContentSize(), PhysicsMaterial(10.0f, 0.0f, 1.001f)));
    box2->getPhysicsBody()->setCategoryBitmask(3);
    box2->getPhysicsBody()->setCollisionBitmask(3);
    box2->getPhysicsBody()->setContactTestBitmask(3);
    box2->getPhysicsBody()->setRotationEnable(false);
    addChild(box2, 3);

     bool GameScene4::onConcactBegin(PhysicsContact & contact) {
         auto shapeA = contact.getShapeA();
         auto shapeB = contact.getShapeB();
         auto BitmaskA = shapeA->getCollisionBitmask();
         auto BitmaskB = shapeB->getCollisionBitmask();

         if (BitmaskA == 3 && BitmaskB == 3) {
             auto joint1 = PhysicsJointDistance::construct(box1->getPhysicsBody(), box2->getPhysicsBody(),
                 box1->getAnchorPoint(), box2->getAnchorPoint());
             if (m_world)
                 m_world->addJoint(joint1);
         }
     return true;
     }
```

图 26

以上就是基本的情况展示。

五、 项目难点及解决方案

1、 Tiled 工具导入的 map 和玩家的碰撞检测老是做不好，使用刚体墙解决了此问题。

2、 动画有些不协调，解决办法是裁剪掉多余的帧。

3、 最开始时用 tiledmap 画的地图，但是总是不能进行碰撞检测，上网查的教程总是说没有某个函数，最后 team 成员决定改变方针每一个墙体都是一个精灵，并赋予其物理属性来实现墙的部分。

4、 页面跳转时会出现断点，想了一下发现是在跳转页面之前没有关闭当前页面使用的侦听器和调度器，在页面的跳转之前将使用的调度器和侦听器关闭解决了此问题；

5、 在有调度器和侦听器的界面重玩界面也会产生断点，在页面的重置之前将使用的调度器和侦听器关闭解决了此问题；

六、 项目总结

饶宇熹：我在本次实验中主要负责的是 MenuScene 和 HelpScene 的部分。MenuScene 的功能是实现主页面和游戏帮助页面的跳转，有点类似封面的作用。HelpScene 是对游戏玩法的简单介绍，是一个游戏具备完整性必不可少的部分。MenuScene 和 HelpScene 的实现相对简单，主要时间花在找素材上。另外，在游戏实现过程中，为了解决 tiled map 墙体和玩家碰撞的问题，我们改用了刚体墙。

简单总结就是，麻雀虽小五脏俱全，虽然我们做的是小游戏，但是注重细节和完整性，游戏帮助和关卡指引都齐了，看起来还挺像样的。

马朝露：我的主要任务是素材寻找，裁剪。最开始时我用 tiledmap 画的地图，但是总是不能进行碰撞检测，上网查的教程总是说没有某个函数，最后 team 成员决定改变方针每一个墙体都是一个精灵，并赋予其物理属性来实现墙的部分。

明友芬：本次项目我的主要任务是寻找音乐素材，寻找自己设计关卡的素材和设计是哪个关卡，在设计关卡的时候考虑的灯亮的控制方法与日常生活中灯亮的控制是一样的，有声控和开关控制灯亮这两种，另外还设计了一个有陷阱的界面，在有陷阱的那个界面，玩家需要大胆的尝试，充分利用空间才能完美的避开陷阱，从而通关。在实现这几个关卡的时候基本都是先将前面关卡的代码赋值过来，然后在原来的基础上更改，因为界面之间的差距不是很大，所以代码能很好的实现重用。在实现的过程

中遇到的问题就是在使用了侦听器和调度器的页面点击 next 和 restart 按钮都会出现断点，检查之后发现是因为没有在跳转之前关闭侦听器和调度器，关闭之后问题就解决了。总之本次项目蛮有收获的。

苏惠玲：在拿到了有精灵的 Demo 之后，实现了键盘控制人物移动，添加物理世界，给人物和箱子设置了刚体属性，尝试做了一些功能，觉得有点无趣，突然想起新的玩法，和组员讨论过确定可行后做了每一关卡的模板，（实时检查是否通关，跳转到下一关卡）并且设计实现了其中两个关卡．有一关是将灯泡移到了墙壁外，并且用一句话来伪装，墙体的其中两块的刚体属性被我修改为不会产生碰撞的，这样玩家需要突破惯性思维才能通关． 另一个关卡用到了关节，还有一些触发事件．将两个箱子用关节连接到了一起，但是两个灯泡的初始位置是分开的，盲目推动箱子是没有用的，开一下脑洞之后就能想到灯泡是可以移动的．设计这些关卡都很有趣，设计的时候还要结合物理世界，物理世界真的很神奇，但是有时候又会与意愿相违背，要使两者协调的过程真是痛并快乐着。