现代操作系统应用开发实验报告

姓名：余漫霖

学号：16340280

实验名称：lab5

一、参考资料

http://www.effecthub.com/particle2dx

https://blog.csdn.net/wiyun_beijing/article/details/17912127

http://blog.csdn.net/fansongy/article/details/14142323

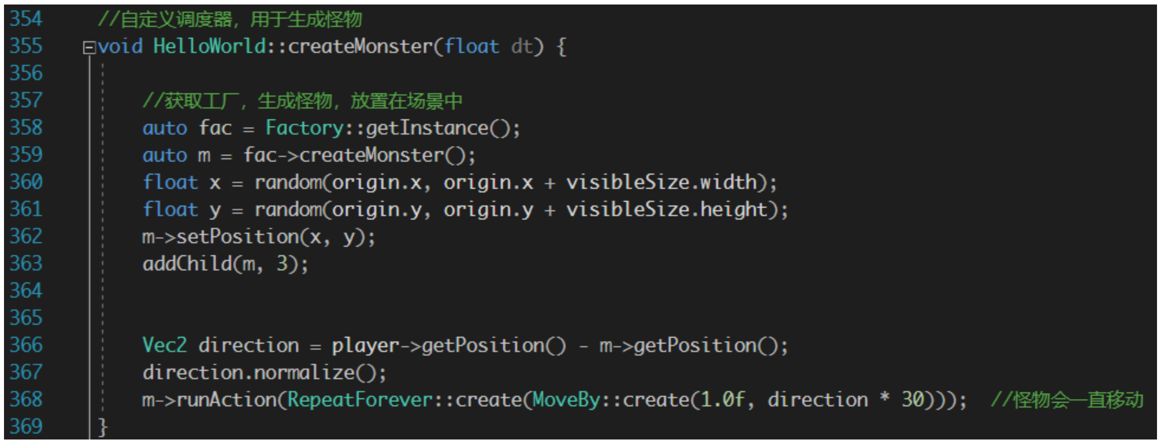
http://www.w3cschool.cc/sqlite/sqlite-tutorial.html

http://www.cocos.com/doc/tutorial/show?id=2455

二、实验步骤

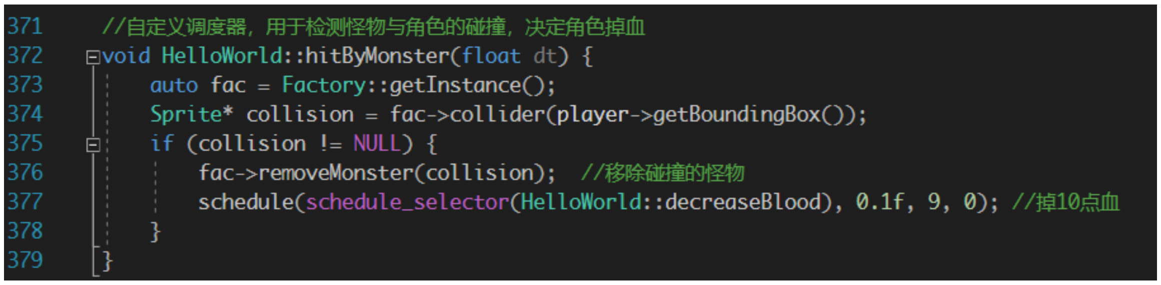
Week13:

1. 随机产生怪物并且怪物会向角色靠近

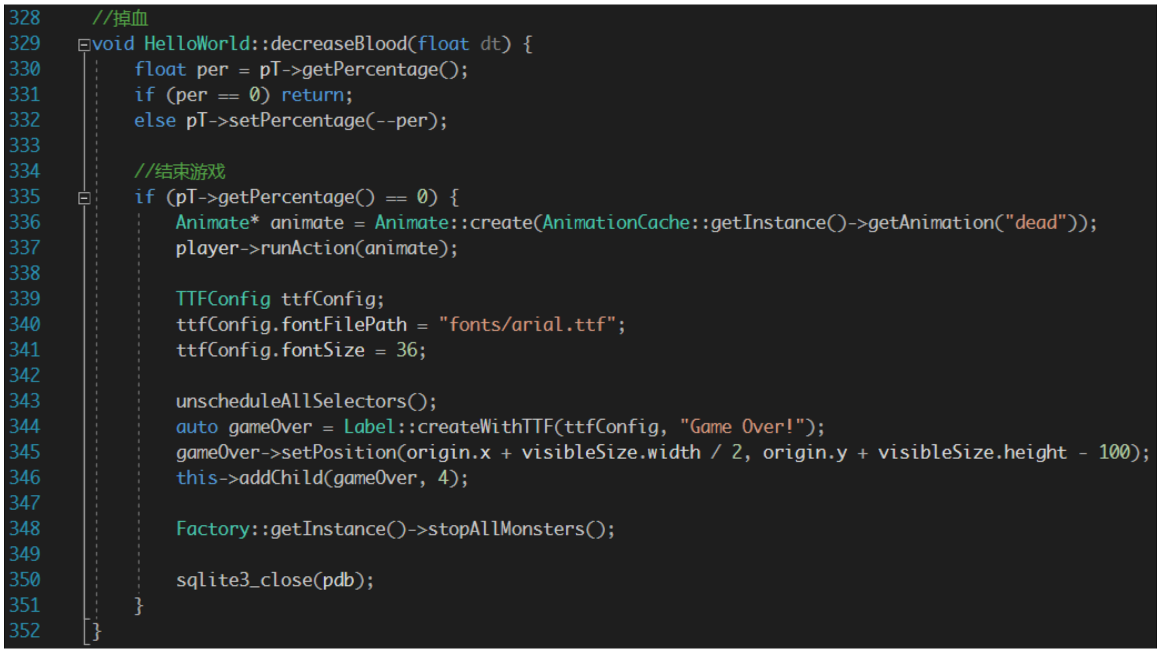


```cpp
//自定义调度器，用于生成怪物
void HelloWorld::createMonster(float dt) {

    //获取工厂，生成怪物，放置在场景中
    auto fac = Factory::getInstance();
    auto m = fac->createMonster();
    float x = random(origin.x, origin.x + visibleSize.width);
    float y = random(origin.y, origin.y + visibleSize.height);
    m->setPosition(x, y);
    addChild(m, 3);


    Vec2 direction = player->getPosition() - m->getPosition();
    direction.normalize();
    m->runAction(RepeatForever::create(MoveBy::create(1.0f, direction * 30)));  //怪物会一直移动
}
```

2. 怪物碰到角色后，角色掉血，角色血量为空则播放死亡动画并解除所有事件

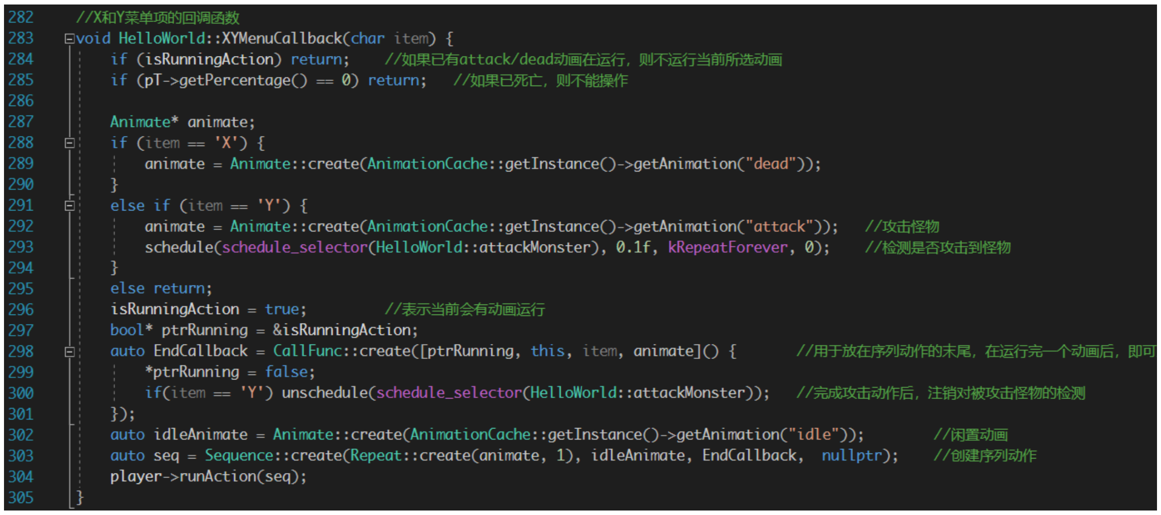


```cpp
//自定义调度器，用于检测怪物与角色的碰撞，决定角色掉血
void HelloWorld::hitByMonster(float dt) {
    auto fac = Factory::getInstance();
    Sprite* collision = fac->collider(player->getBoundingBox());
    if (collision != NULL) {
        fac->removeMonster(collision);  //移除碰撞的怪物
        schedule(schedule_selector(HelloWorld::decreaseBlood), 0.1f, 9, 0); //掉10点血
    }
}
```

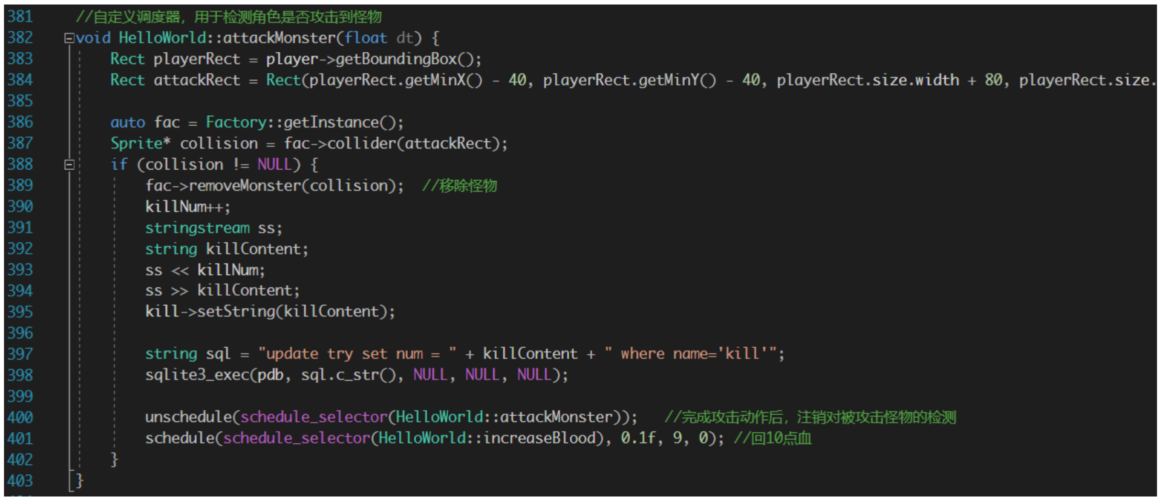


```cpp
//掉血
void HelloWorld::decreaseBlood(float dt) {
    float per = pT->getPercentage();
    if (per == 0) return;
    else pT->setPercentage(--per);

    //结束游戏
    if (pT->getPercentage() == 0) {
        Animate* animate = Animate::create(AnimationCache::getInstance()->getAnimation("dead"));
        player->runAction(animate);

        TTFConfig ttfConfig;
        ttfConfig.fontFilePath = "fonts/arial.ttf";
        ttfConfig.fontSize = 36;

        unscheduleAllSelectors();
        auto gameOver = Label::createWithTTF(ttfConfig, "Game Over!");
        gameOver->setPosition(origin.x + visibleSize.width / 2, origin.y + visibleSize.height - 100);
        this->addChild(gameOver, 4);

        Factory::getInstance()->stopAllMonsters();

        sqlite3_close(pdb);
    }
}
```

3. 角色可以攻击怪物

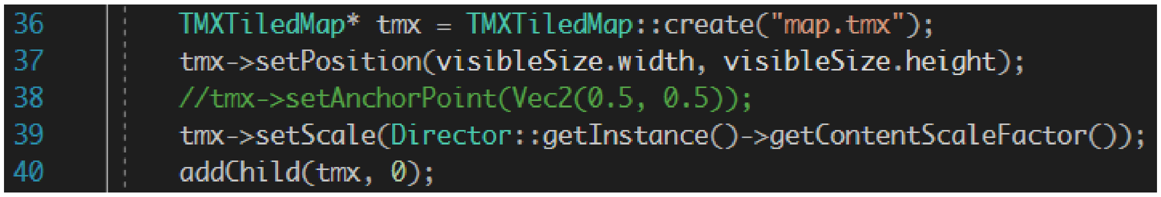


```cpp
//X和Y菜单项的回调函数
void HelloWorld::XYMenuCallback(char item) {
    if (isRunningAction) return;    //如果已有attack/dead动画在运行，则不运行当前所选动画
    if (pT->getPercentage() == 0) return;   //如果已死亡，则不能操作

    Animate* animate;
    if (item == 'X') {
        animate = Animate::create(AnimationCache::getInstance()->getAnimation("dead"));
    }
    else if (item == 'Y') {
        animate = Animate::create(AnimationCache::getInstance()->getAnimation("attack"));   //攻击怪物
        schedule(schedule_selector(HelloWorld::attackMonster), 0.1f, kRepeatForever, 0);    //检测是否攻击到怪物
    }
    else return;
    isRunningAction = true;         //表示当前会有动画运行
    bool* ptrRunning = &isRunningAction;
    auto EndCallback = CallFunc::create([ptrRunning, this, item, animate]() {       //用于放在序列动作的末尾，在运行完一个动画后，即可
        *ptrRunning = false;
        if(item == 'Y') unschedule(schedule_selector(HelloWorld::attackMonster));   //完成攻击动作后，注销对被攻击怪物的检测
    });
    auto idleAnimate = Animate::create(AnimationCache::getInstance()->getAnimation("idle"));        //闲置动画
    auto seq = Sequence::create(Repeat::create(animate, 1), idleAnimate, EndCallback,  nullptr);    //创建序列动作
    player->runAction(seq);
}
```



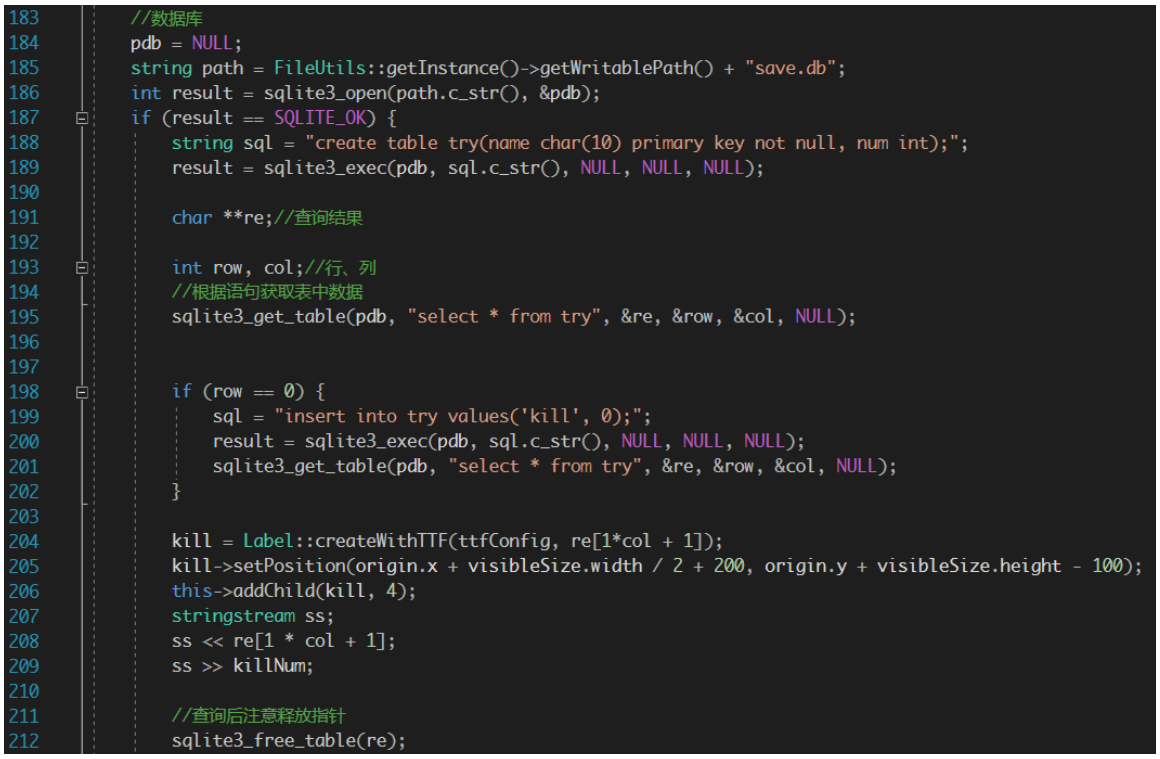
```cpp
//自定义调度器，用于检测角色是否攻击到怪物
void HelloWorld::attackMonster(float dt) {
    Rect playerRect = player->getBoundingBox();
    Rect attackRect = Rect(playerRect.getMinX() - 40, playerRect.getMinY() - 40, playerRect.size.width + 80, playerRect.size.

    auto fac = Factory::getInstance();
    Sprite* collision = fac->collider(attackRect);
    if (collision != NULL) {
        fac->removeMonster(collision);  //移除怪物
        killNum++;
        stringstream ss;
        string killContent;
        ss << killNum;
        ss >> killContent;
        kill->setString(killContent);

        string sql = "update try set num = " + killContent + " where name='kill'";
        sqlite3_exec(pdb, sql.c_str(), NULL, NULL, NULL);

        unschedule(schedule_selector(HelloWorld::attackMonster));   //完成攻击动作后，注销对被攻击怪物的检测
        schedule(schedule_selector(HelloWorld::increaseBlood), 0.1f, 9, 0); //回10点血
    }
}
```

4. 使用 tilemap 创建地图

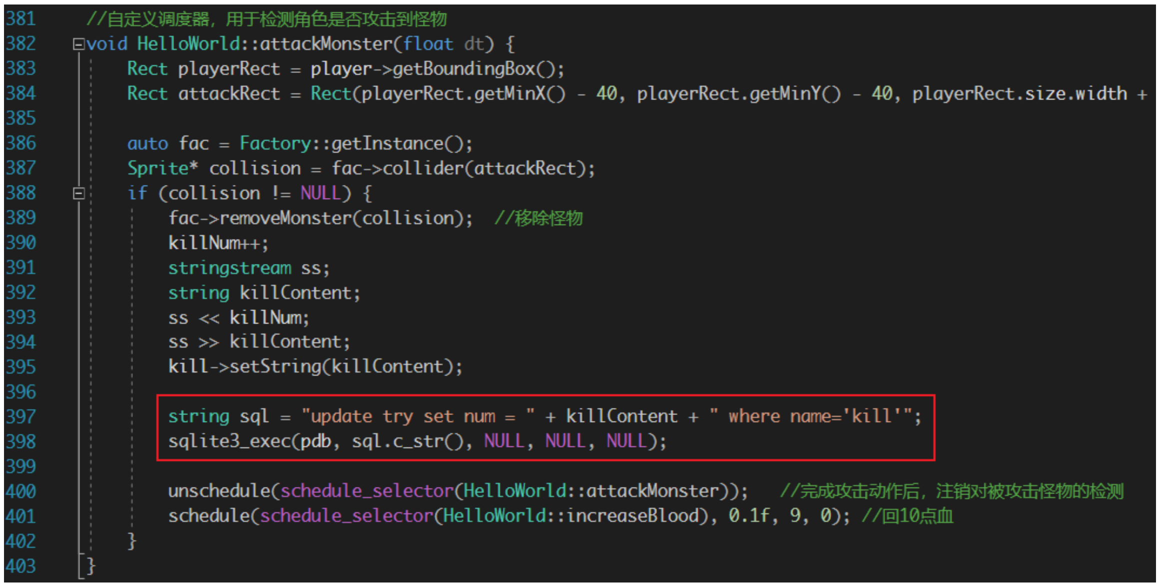


```cpp
TMXTiledMap* tmx = TMXTiledMap::create("map.tmx");
tmx->setPosition(visibleSize.width, visibleSize.height);
//tmx->setAnchorPoint(Vec2(0.5, 0.5));
tmx->setScale(Director::getInstance()->getContentScaleFactor());
addChild(tmx, 0);
```

5. 加分项：使用本地数据存储，记录打到的怪物数量，同时在游戏中显示打倒数量



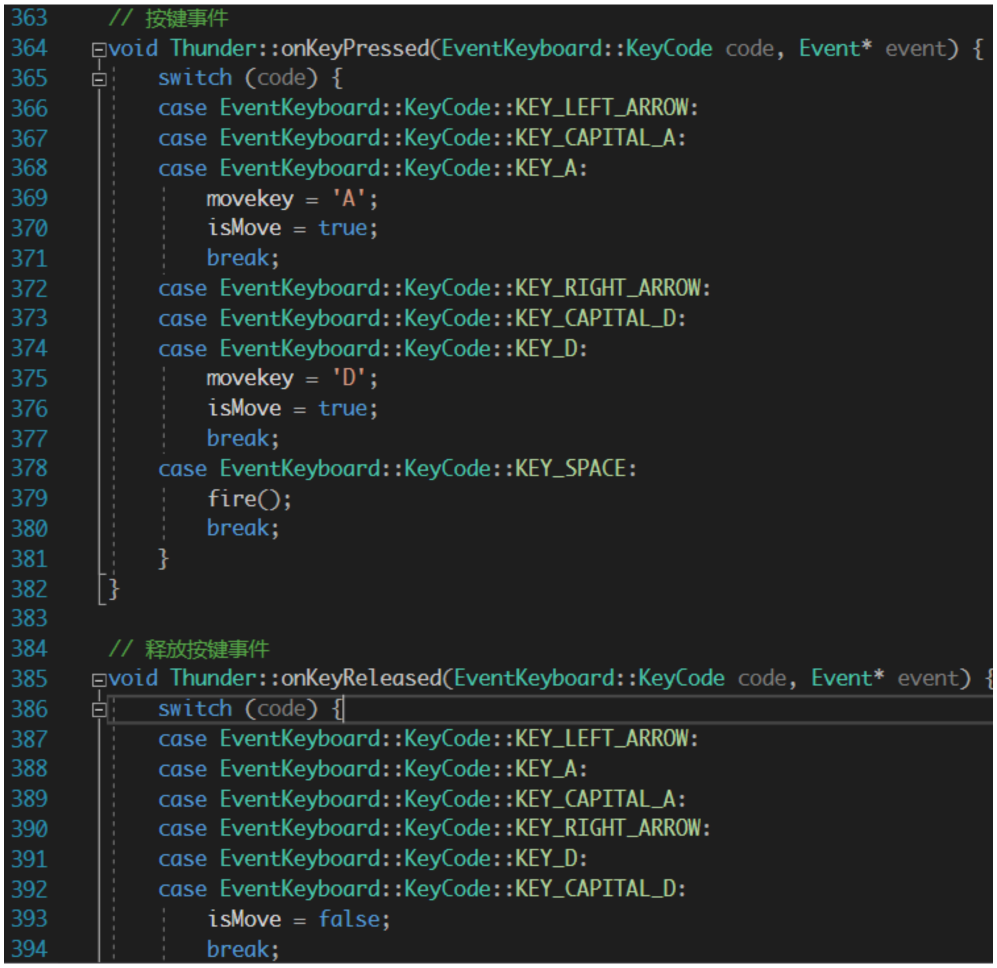
```cpp
    //数据库
    pdb = NULL;
    string path = FileUtils::getInstance()->getWritablePath() + "save.db";
    int result = sqlite3_open(path.c_str(), &pdb);
    if (result == SQLITE_OK) {
        string sql = "create table try(name char(10) primary key not null, num int);";
        result = sqlite3_exec(pdb, sql.c_str(), NULL, NULL, NULL);

        char **re;//查询结果

        int row, col;//行、列
        //根据语句获取表中数据
        sqlite3_get_table(pdb, "select * from try", &re, &row, &col, NULL);


        if (row == 0) {
            sql = "insert into try values('kill', 0);";
            result = sqlite3_exec(pdb, sql.c_str(), NULL, NULL, NULL);
            sqlite3_get_table(pdb, "select * from try", &re, &row, &col, NULL);
        }

        kill = Label::createWithTTF(ttfConfig, re[1*col + 1]);
        kill->setPosition(origin.x + visibleSize.width / 2 + 200, origin.y + visibleSize.height - 100);
        this->addChild(kill, 4);
        stringstream ss;
        ss << re[1 * col + 1];
        ss >> killNum;

        //查询后注意释放指针
        sqlite3_free_table(re);
```



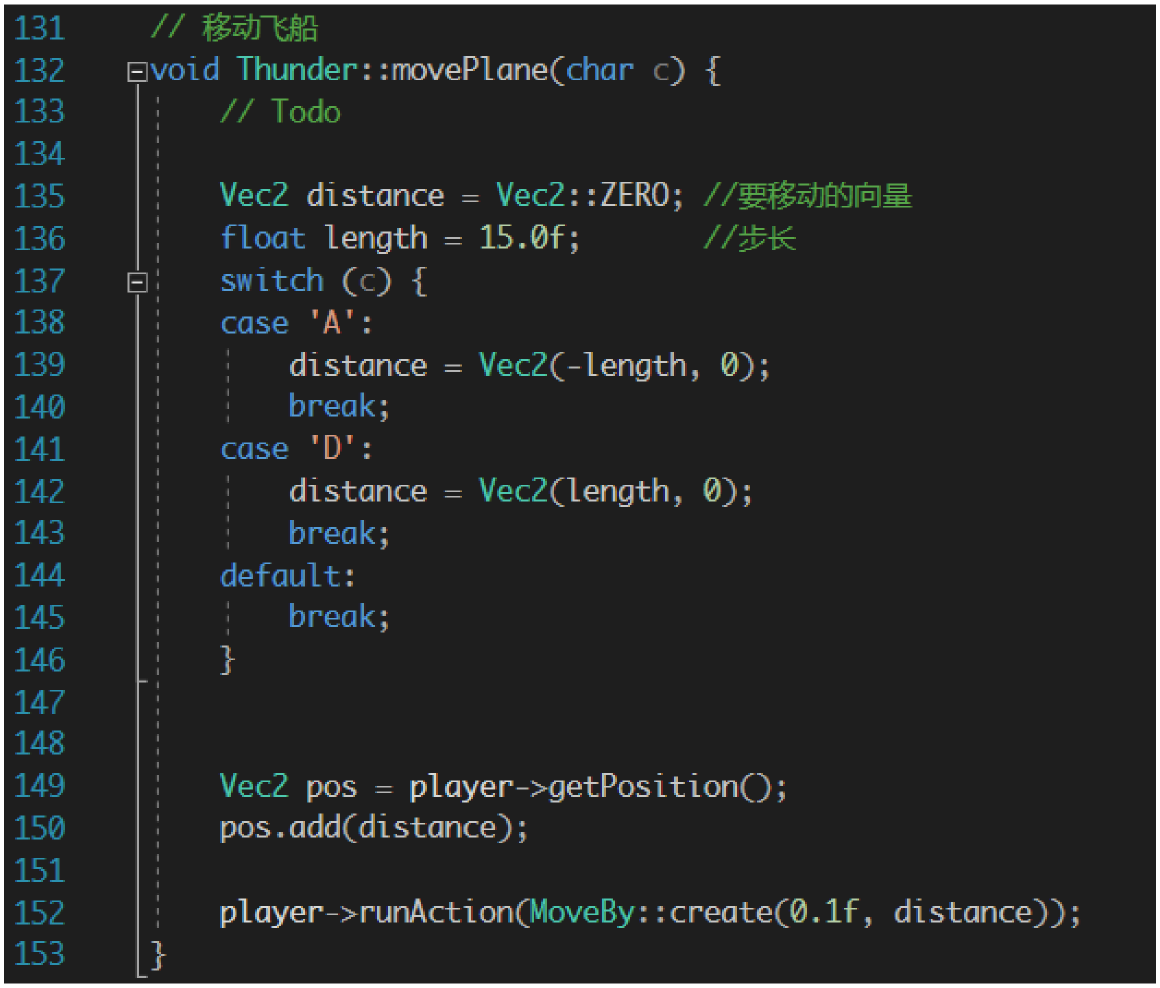
```cpp
//自定义调度器，用于检测角色是否攻击到怪物
void HelloWorld::attackMonster(float dt) {
    Rect playerRect = player->getBoundingBox();
    Rect attackRect = Rect(playerRect.getMinX() - 40, playerRect.getMinY() - 40, playerRect.size.width +

    auto fac = Factory::getInstance();
    Sprite* collision = fac->collider(attackRect);
    if (collision != NULL) {
        fac->removeMonster(collision);  //移除怪物
        killNum++;
        stringstream ss;
        string killContent;
        ss << killNum;
        ss >> killContent;
        kill->setString(killContent);

        string sql = "update try set num = " + killContent + " where name='kill'";
        sqlite3_exec(pdb, sql.c_str(), NULL, NULL, NULL);

        unschedule(schedule_selector(HelloWorld::attackMonster));   //完成攻击动作后，注销对被攻击怪物的检测
        schedule(schedule_selector(HelloWorld::increaseBlood), 0.1f, 9, 0); //回10点血
    }
}
```

Week14:

1. 利用键盘事件实现飞船左右移动。

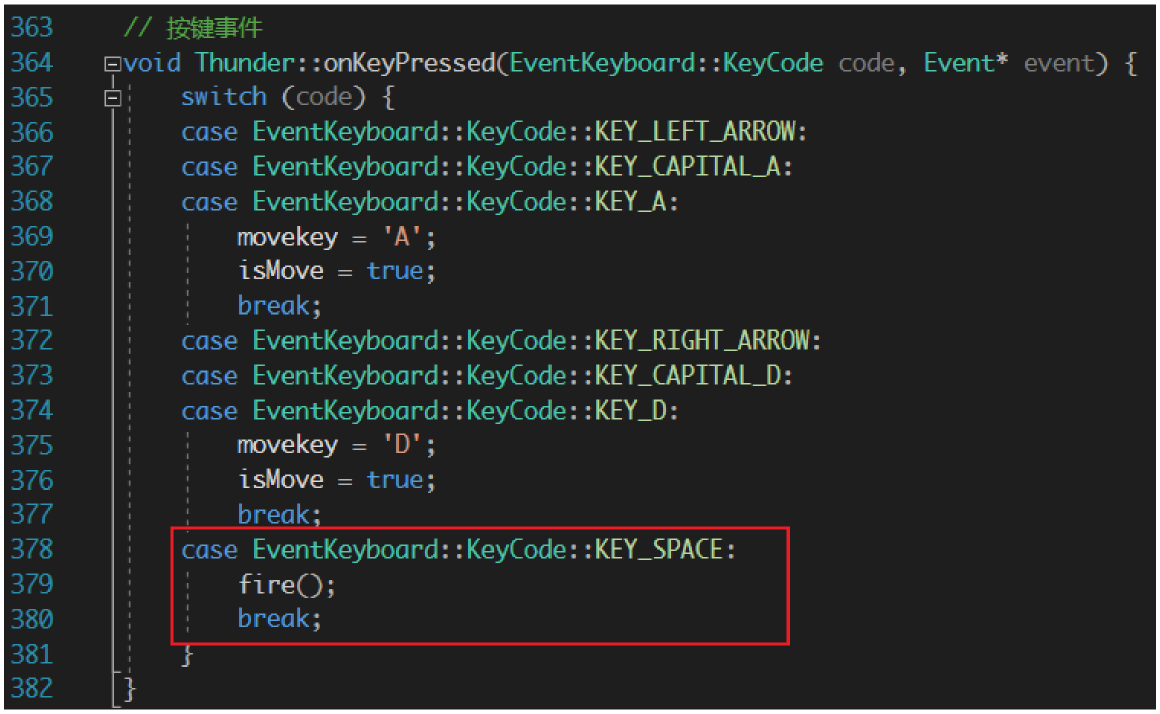


```
// 按键事件
void Thunder::onKeyPressed(EventKeyboard::KeyCode code, Event* event) {
    switch (code) {
    case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
    case EventKeyboard::KeyCode::KEY_CAPITAL_A:
    case EventKeyboard::KeyCode::KEY_A:
        movekey = 'A';
        isMove = true;
        break;
    case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
    case EventKeyboard::KeyCode::KEY_CAPITAL_D:
    case EventKeyboard::KeyCode::KEY_D:
        movekey = 'D';
        isMove = true;
        break;
    case EventKeyboard::KeyCode::KEY_SPACE:
        fire();
        break;
    }
}

// 释放按键事件
void Thunder::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:
        isMove = false;
        break;
```

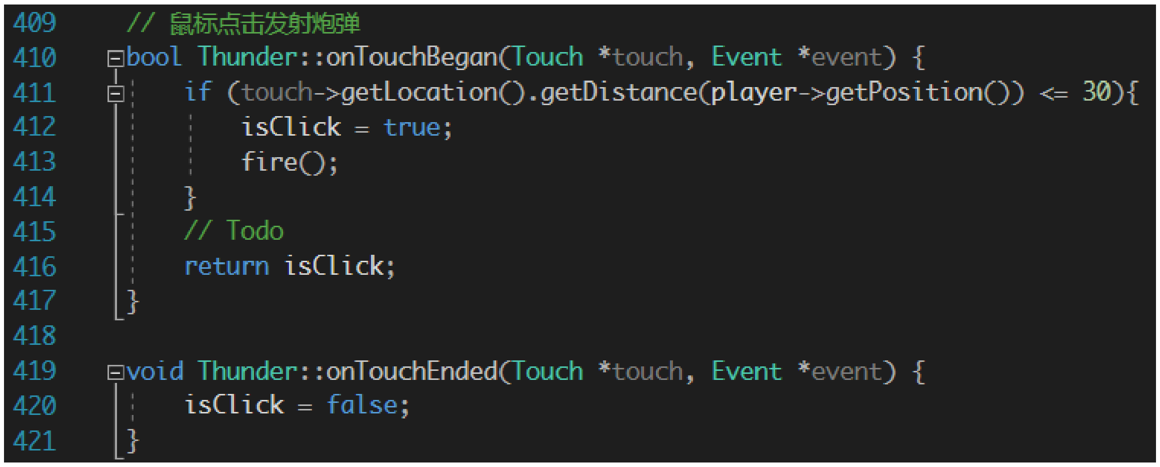


```
// 移动飞船
void Thunder::movePlane(char c) {
    // Todo

    Vec2 distance = Vec2::ZERO; //要移动的向量
    float length = 15.0f;       //步长
    switch (c) {
    case 'A':
        distance = Vec2(-length, 0);
        break;
    case 'D':
        distance = Vec2(length, 0);
        break;
    default:
        break;
    }


    Vec2 pos = player->getPosition();
    pos.add(distance);

    player->runAction(MoveBy::create(0.1f, distance));
}
```

2. 利用键盘和触摸事件实现子弹发射。

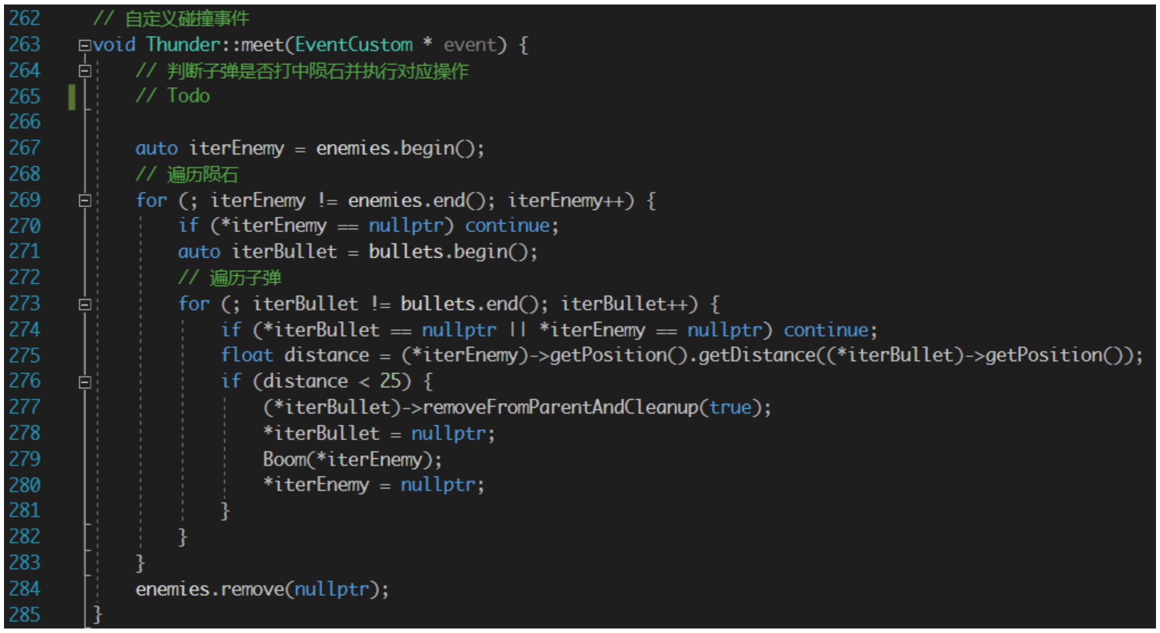


```
    // 按键事件
void Thunder::onKeyPressed(EventKeyboard::KeyCode code, Event* event) {
    switch (code) {
    case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
    case EventKeyboard::KeyCode::KEY_CAPITAL_A:
    case EventKeyboard::KeyCode::KEY_A:
        movekey = 'A';
        isMove = true;
        break;
    case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
    case EventKeyboard::KeyCode::KEY_CAPITAL_D:
    case EventKeyboard::KeyCode::KEY_D:
        movekey = 'D';
        isMove = true;
        break;
    case EventKeyboard::KeyCode::KEY_SPACE:
        fire();
        break;
    }
}
```

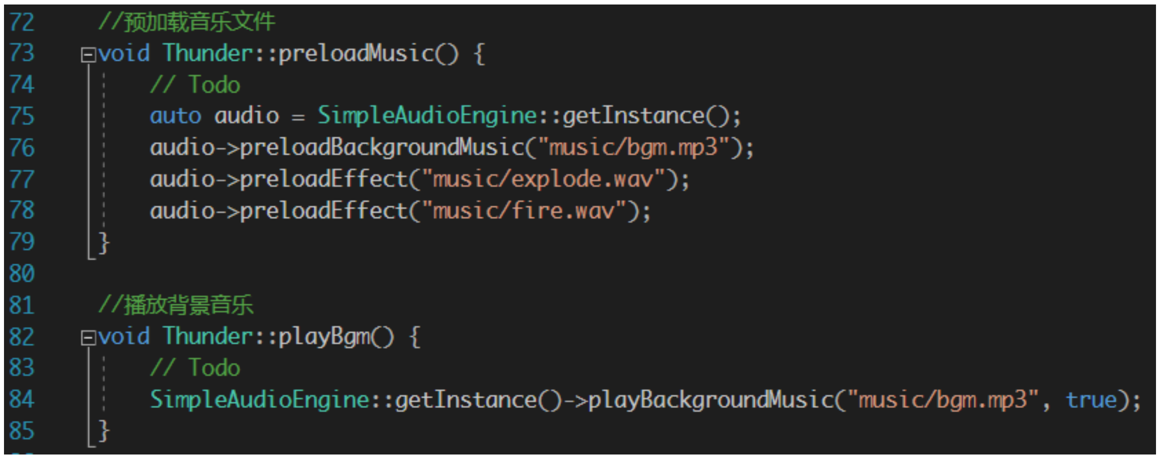


```
    // 鼠标点击发射炮弹
bool Thunder::onTouchBegan(Touch *touch, Event *event) {
    if (touch->getLocation().getDistance(player->getPosition()) <= 30){
        isClick = true;
        fire();
    }
    // Todo
    return isClick;
}

void Thunder::onTouchEnded(Touch *touch, Event *event) {
    isClick = false;
}
```

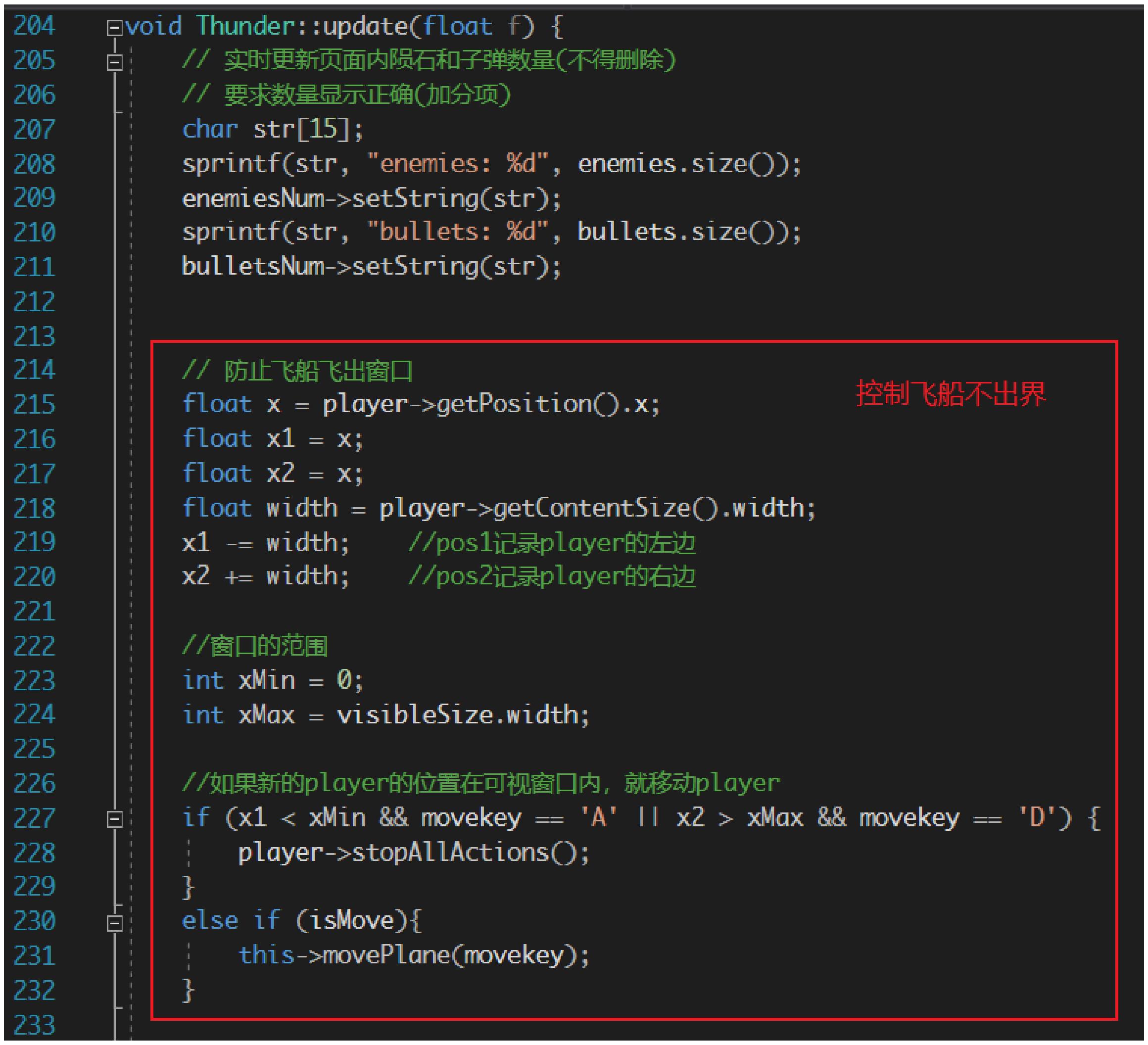
3. 用自定义事件实现：子弹和陨石相距小于一定距离时，陨石爆炸，子弹消失。


```cpp
// 自定义碰撞事件
void Thunder::meet(EventCustom * event) {
    // 判断子弹是否打中陨石并执行对应操作
    // Todo

    auto iterEnemy = enemies.begin();
    // 遍历陨石
    for (; iterEnemy != enemies.end(); iterEnemy++) {
        if (*iterEnemy == nullptr) continue;
        auto iterBullet = bullets.begin();
        // 遍历子弹
        for (; iterBullet != bullets.end(); iterBullet++) {
            if (*iterBullet == nullptr || *iterEnemy == nullptr) continue;
            float distance = (*iterEnemy)->getPosition().getDistance((*iterBullet)->getPosition());
            if (distance < 25) {
                (*iterBullet)->removeFromParentAndCleanup(true);
                *iterBullet = nullptr;
                Boom(*iterEnemy);
                *iterEnemy = nullptr;
            }
        }
    }
    enemies.remove(nullptr);
}
```

4. 游戏过程中有背景音乐，发射子弹、击中陨石有音效。

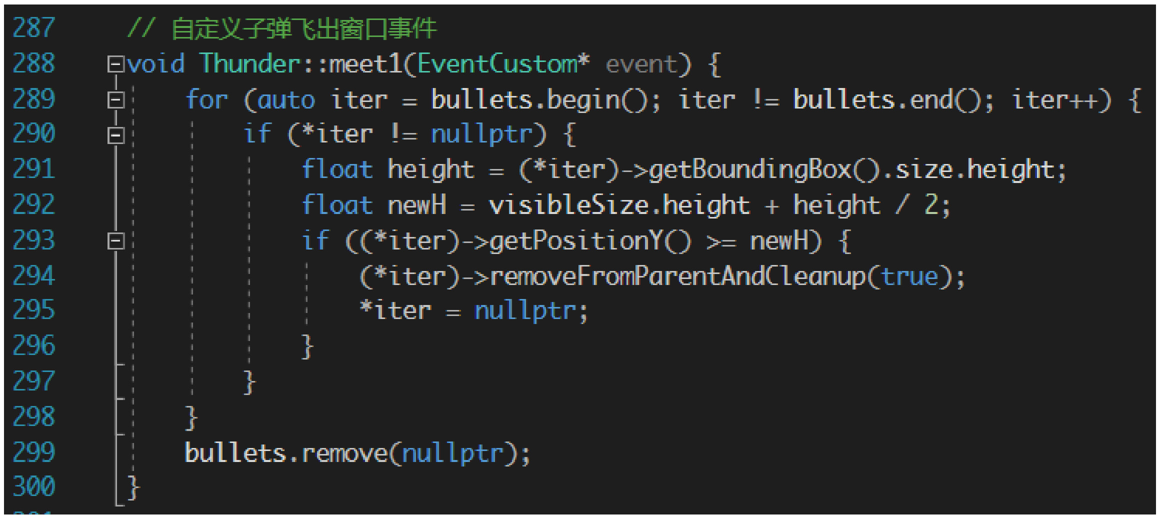

```cpp
//预加载音乐文件
void Thunder::preloadMusic() {
    // Todo
    auto audio = SimpleAudioEngine::getInstance();
    audio->preloadBackgroundMusic("music/bgm.mp3");
    audio->preloadEffect("music/explode.wav");
    audio->preloadEffect("music/fire.wav");
}

//播放背景音乐
void Thunder::playBgm() {
    // Todo
    SimpleAudioEngine::getInstance()->playBackgroundMusic("music/bgm.mp3", true);
}
```

5. 注意飞船、子弹的移动范围。



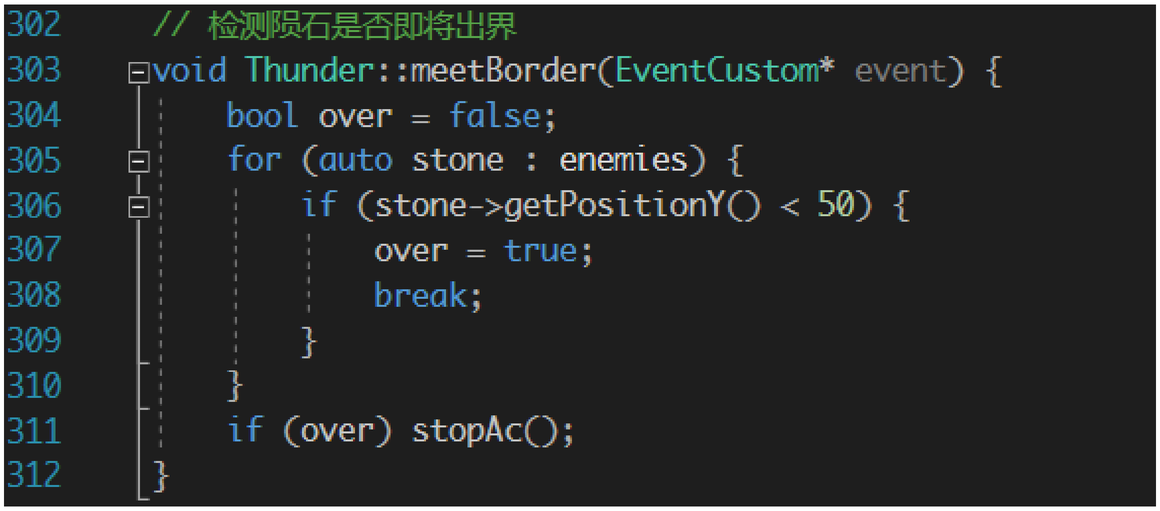
```cpp
void Thunder::update(float f) {
    // 实时更新页面内陨石和子弹数量(不得删除)
    // 要求数量显示正确(加分项)
    char str[15];
    sprintf(str, "enemies: %d", enemies.size());
    enemiesNum->setString(str);
    sprintf(str, "bullets: %d", bullets.size());
    bulletsNum->setString(str);


    // 防止飞船飞出窗口
    float x = player->getPosition().x;
    float x1 = x;
    float x2 = x;
    float width = player->getContentSize().width;
    x1 -= width;    //pos1记录player的左边
    x2 += width;    //pos2记录player的右边

    //窗口的范围
    int xMin = 0;
    int xMax = visibleSize.width;

    //如果新的player的位置在可视窗口内，就移动player
    if (x1 < xMin && movekey == 'A' || x2 > xMax && movekey == 'D') {
        player->stopAllActions();
    }
    else if (isMove){
        this->movePlane(movekey);
    }

```

如果子弹飞出窗口，则移除子弹：

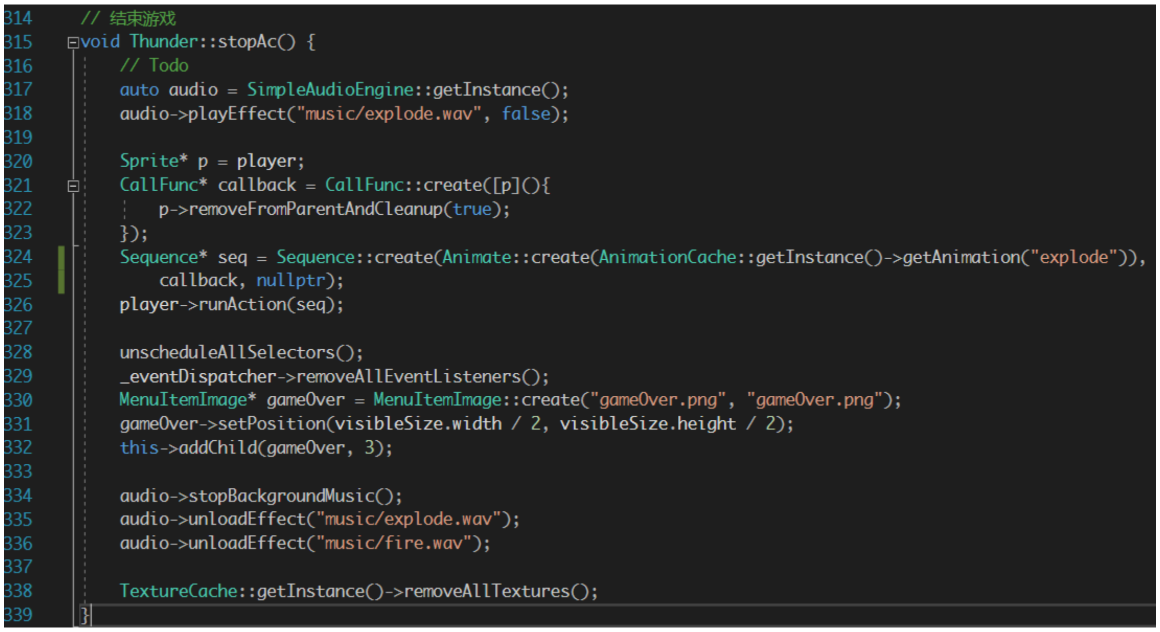


```cpp
// 自定义子弹飞出窗口事件
void Thunder::meet1(EventCustom* event) {
    for (auto iter = bullets.begin(); iter != bullets.end(); iter++) {
        if (*iter != nullptr) {
            float height = (*iter)->getBoundingBox().size.height;
            float newH = visibleSize.height + height / 2;
            if ((*iter)->getPositionY() >= newH) {
                (*iter)->removeFromParentAndCleanup(true);
                *iter = nullptr;
            }
        }
    }
    bullets.remove(nullptr);
}
```

6. 游戏结束飞船爆炸，移除所有监听器



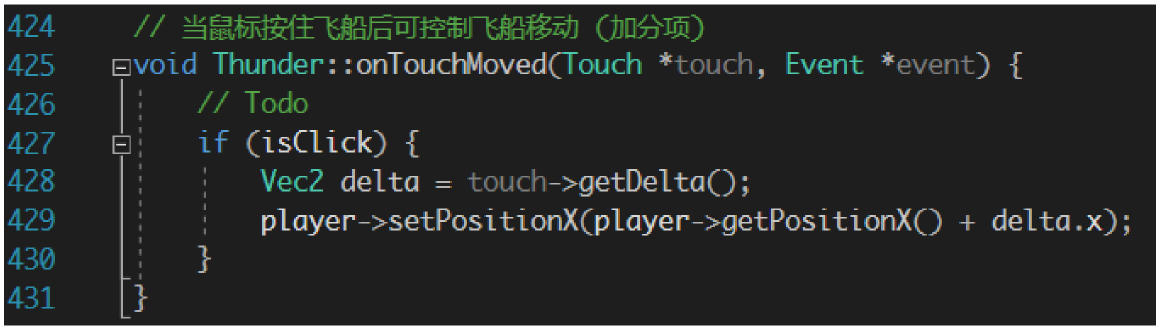
```cpp
// 检测陨石是否即将出界
void Thunder::meetBorder(EventCustom* event) {
    bool over = false;
    for (auto stone : enemies) {
        if (stone->getPositionY() < 50) {
            over = true;
            break;
        }
    }
    if (over) stopAc();
}
```



```cpp
// 结束游戏
void Thunder::stopAc() {
    // Todo
    auto audio = SimpleAudioEngine::getInstance();
    audio->playEffect("music/explode.wav", false);

    Sprite* p = player;
    CallFunc* callback = CallFunc::create([p](){
        p->removeFromParentAndCleanup(true);
    });
    Sequence* seq = Sequence::create(Animate::create(AnimationCache::getInstance()->getAnimation("explode")),
        callback, nullptr);
    player->runAction(seq);

    unscheduleAllSelectors();
    _eventDispatcher->removeAllEventListeners();
    MenuItemImage* gameOver = MenuItemImage::create("gameOver.png", "gameOver.png");
    gameOver->setPosition(visibleSize.width / 2, visibleSize.height / 2);
    this->addChild(gameOver, 3);

    audio->stopBackgroundMusic();
    audio->unloadEffect("music/explode.wav");
    audio->unloadEffect("music/fire.wav");

    TextureCache::getInstance()->removeAllTextures();
}
```

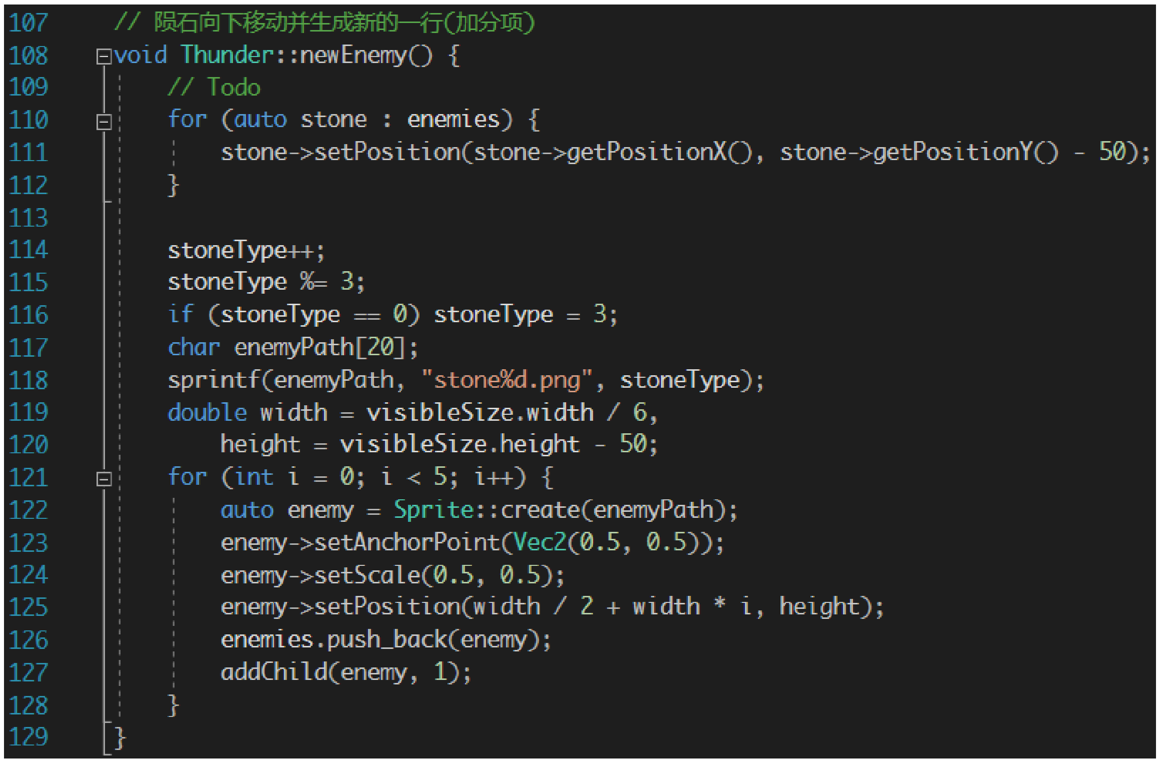
加分项：

1. 利用触摸事件实现飞船移动。（点击飞船后拖动鼠标）



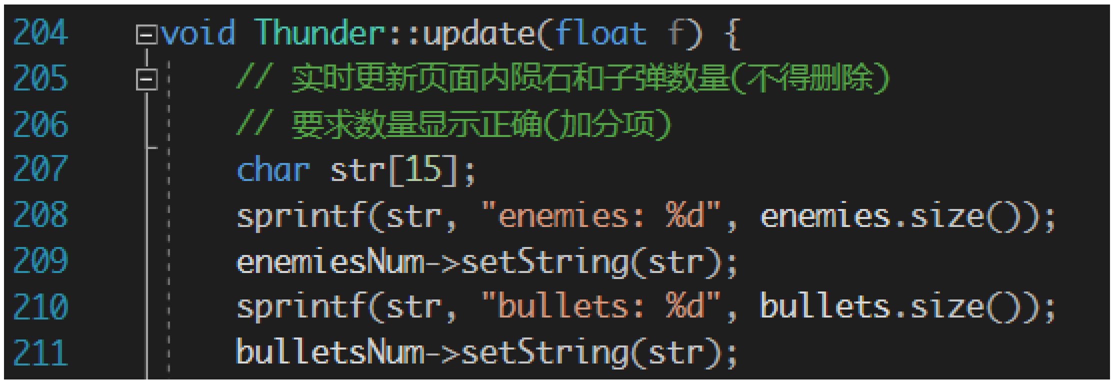
```cpp
// 当鼠标按住飞船后可控制飞船移动 (加分项)
void Thunder::onTouchMoved(Touch *touch, Event *event) {
    // Todo
    if (isClick) {
        Vec2 delta = touch->getDelta();
        player->setPositionX(player->getPositionX() + delta.x);
    }
}
```

2. 陨石向下移动并生成新的一行陨石



```cpp
// 陨石向下移动并生成新的一行(加分项)
void Thunder::newEnemy() {
    // Todo
    for (auto stone : enemies) {
        stone->setPosition(stone->getPositionX(), stone->getPositionY() - 50);
    }

    stoneType++;
    stoneType %= 3;
    if (stoneType == 0) stoneType = 3;
    char enemyPath[20];
    sprintf(enemyPath, "stone%d.png", stoneType);
    double width = visibleSize.width / 6,
        height = visibleSize.height - 50;
    for (int i = 0; i < 5; i++) {
        auto enemy = Sprite::create(enemyPath);
        enemy->setAnchorPoint(Vec2(0.5, 0.5));
        enemy->setScale(0.5, 0.5);
        enemy->setPosition(width / 2 + width * i, height);
        enemies.push_back(enemy);
        addChild(enemy, 1);
    }
}
```

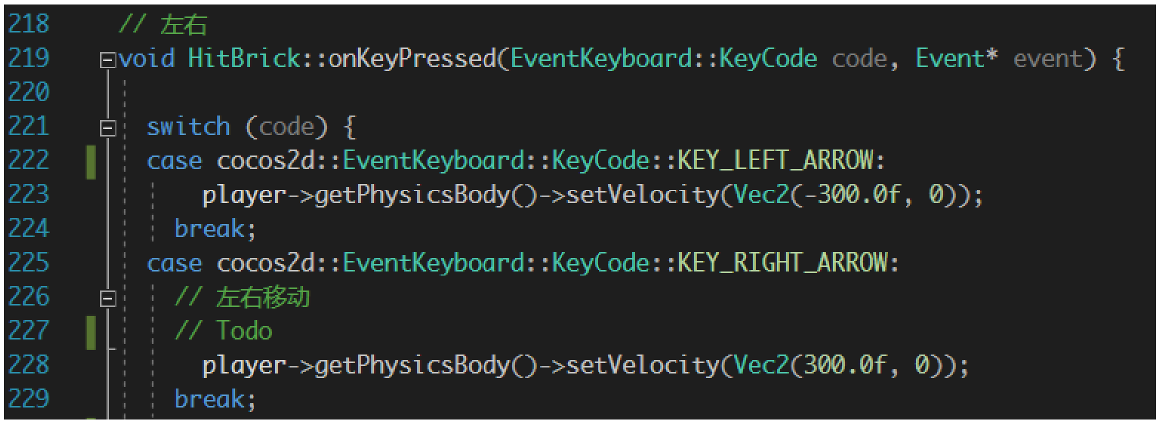
3. 子弹和陨石的数量显示正确



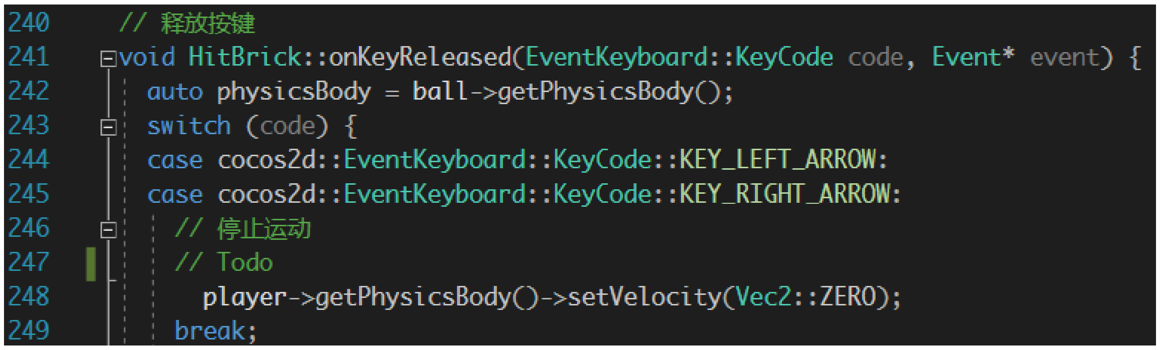
```cpp
void Thunder::update(float f) {
    // 实时更新页面内陨石和子弹数量(不得删除)
    // 要求数量显示正确(加分项)
    char str[15];
    sprintf(str, "enemies: %d", enemies.size());
    enemiesNum->setString(str);
    sprintf(str, "bullets: %d", bullets.size());
    bulletsNum->setString(str);
```

Week15:

1. 控制板子左右移动

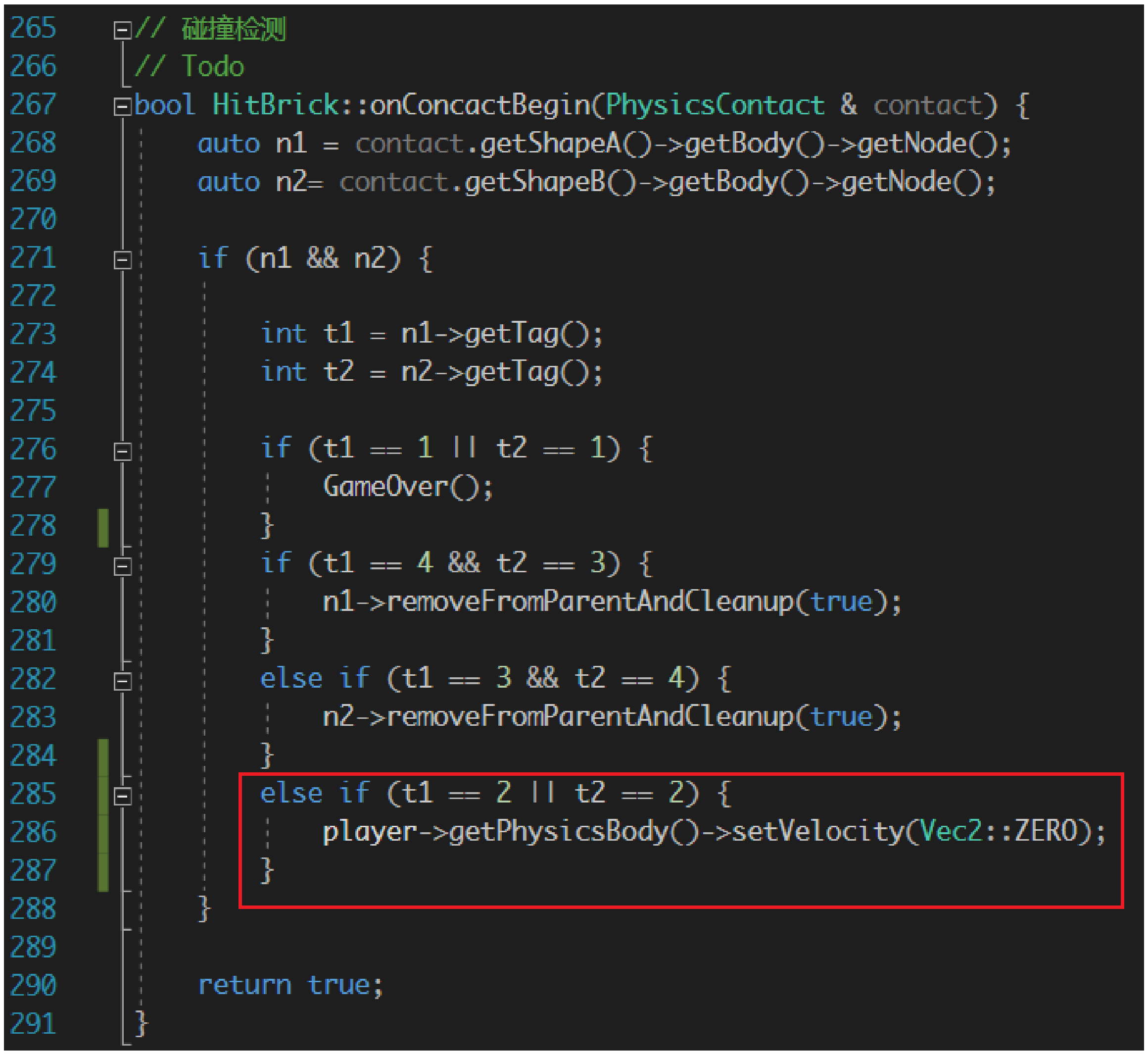


```
    // 左右
void HitBrick::onKeyPressed(EventKeyboard::KeyCode code, Event* event) {

    switch (code) {
    case cocos2d::EventKeyboard::KeyCode::KEY_LEFT_ARROW:
        player->getPhysicsBody()->setVelocity(Vec2(-300.0f, 0));
        break;
    case cocos2d::EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
        // 左右移动
        // Todo
        player->getPhysicsBody()->setVelocity(Vec2(300.0f, 0));
        break;
```



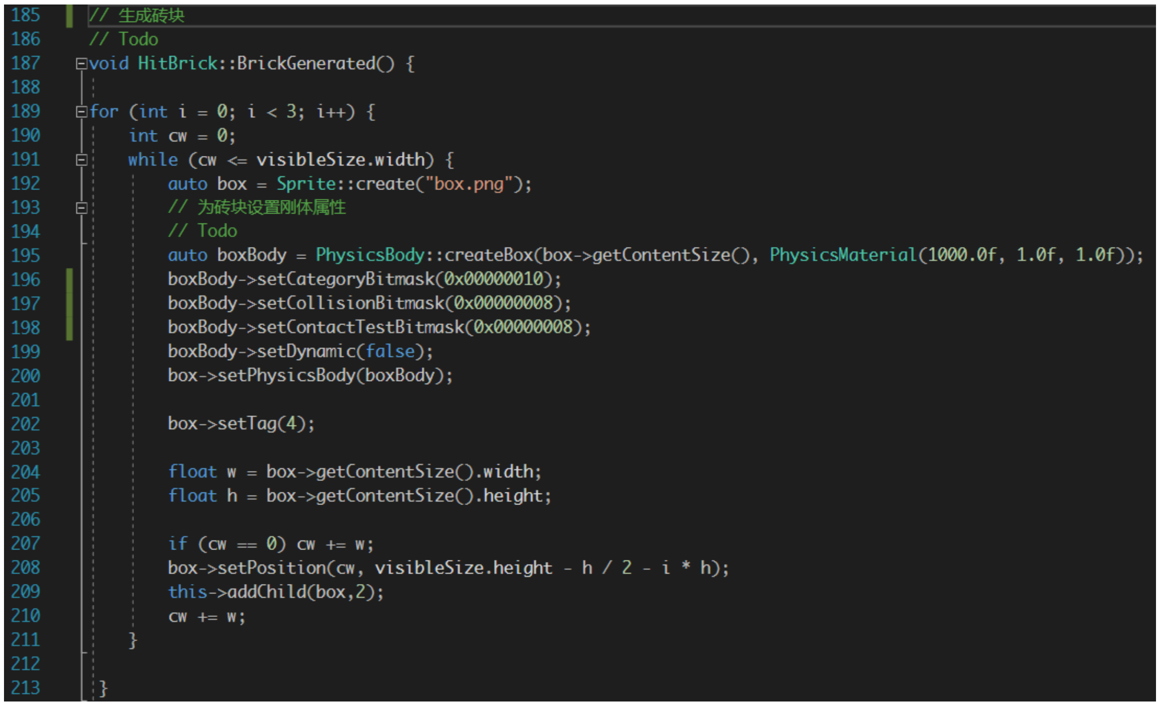
```
    // 释放按键
void HitBrick::onKeyReleased(EventKeyboard::KeyCode code, Event* event) {
    auto physicsBody = ball->getPhysicsBody();
    switch (code) {
    case cocos2d::EventKeyboard::KeyCode::KEY_LEFT_ARROW:
    case cocos2d::EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
        // 停止运动
        // Todo
        player->getPhysicsBody()->setVelocity(Vec2::ZERO);
        break;
```

如果撞到墙，在碰撞检测事件中将板子速度设为 0：



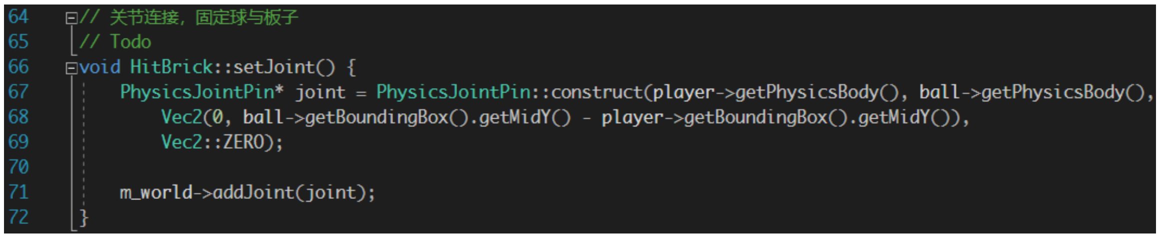
```
// 碰撞检测
// Todo
bool HitBrick::onConcactBegin(PhysicsContact & contact) {
	auto n1 = contact.getShapeA()->getBody()->getNode();
	auto n2= contact.getShapeB()->getBody()->getNode();

	if (n1 && n2) {

		int t1 = n1->getTag();
		int t2 = n2->getTag();

		if (t1 == 1 || t2 == 1) {
			GameOver();
		}
		if (t1 == 4 && t2 == 3) {
			n1->removeFromParentAndCleanup(true);
		}
		else if (t1 == 3 && t2 == 4) {
			n2->removeFromParentAndCleanup(true);
		}
		else if (t1 == 2 || t2 == 2) {
			player->getPhysicsBody()->setVelocity(Vec2::ZERO);
		}
	}

	return true;
}
```

2. 在顶部生成小砖块



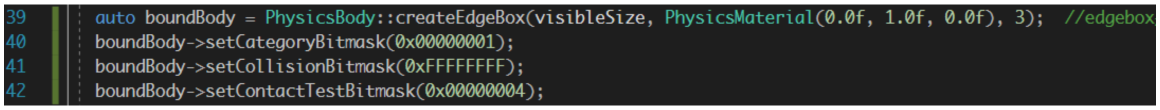
```cpp
// 生成砖块
// Todo
void HitBrick::BrickGenerated() {

	for (int i = 0; i < 3; i++) {
		int cw = 0;
		while (cw <= visibleSize.width) {
			auto box = Sprite::create("box.png");
			// 为砖块设置刚体属性
			// Todo
			auto boxBody = PhysicsBody::createBox(box->getContentSize(), PhysicsMaterial(1000.0f, 1.0f, 1.0f));
			boxBody->setCategoryBitmask(0x00000010);
			boxBody->setCollisionBitmask(0x00000008);
			boxBody->setContactTestBitmask(0x00000008);
			boxBody->setDynamic(false);
			box->setPhysicsBody(boxBody);

			box->setTag(4);

			float w = box->getContentSize().width;
			float h = box->getContentSize().height;

			if (cw == 0) cw += w;
			box->setPosition(cw, visibleSize.height - h / 2 - i * h);
			this->addChild(box,2);
			cw += w;
		}

	}
}
```

3. 使用关节固定球与板子

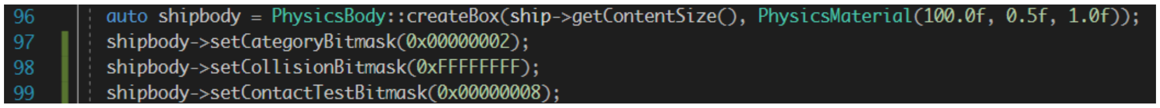


```cpp
// 关节连接，固定球与板子
// Todo
void HitBrick::setJoint() {
	PhysicsJointPin* joint = PhysicsJointPin::construct(player->getPhysicsBody(), ball->getPhysicsBody(),
		Vec2(0, ball->getBoundingBox().getMidY() - player->getBoundingBox().getMidY()),
		Vec2::ZERO);

	m_world->addJoint(joint);
}
```

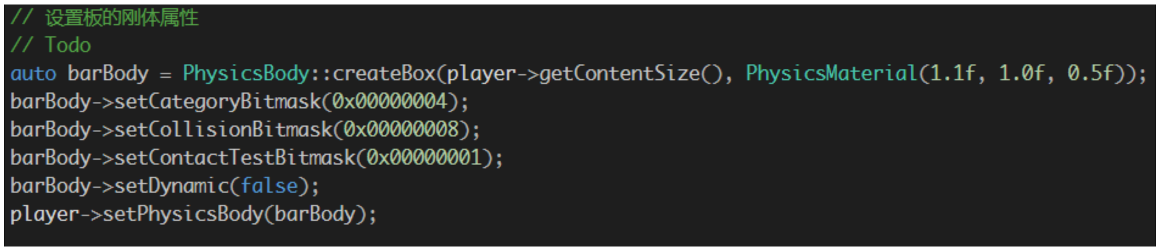
4. 为板子、球、以及砖块设置物理属性



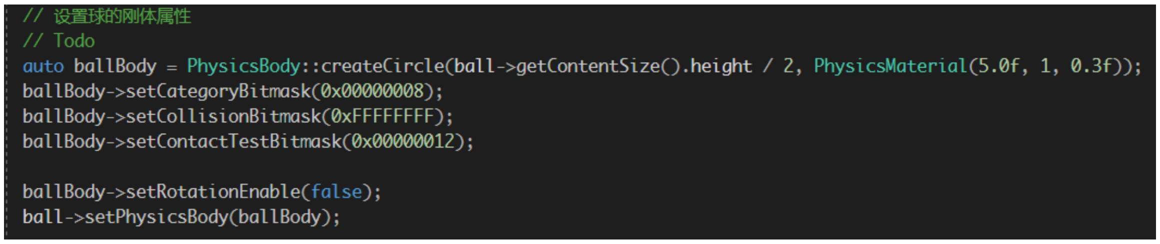
```cpp
	auto boundBody = PhysicsBody::createEdgeBox(visibleSize, PhysicsMaterial(0.0f, 1.0f, 0.0f), 3);  //edgebox
	boundBody->setCategoryBitmask(0x00000001);
	boundBody->setCollisionBitmask(0xFFFFFFFF);
	boundBody->setContactTestBitmask(0x00000004);
```



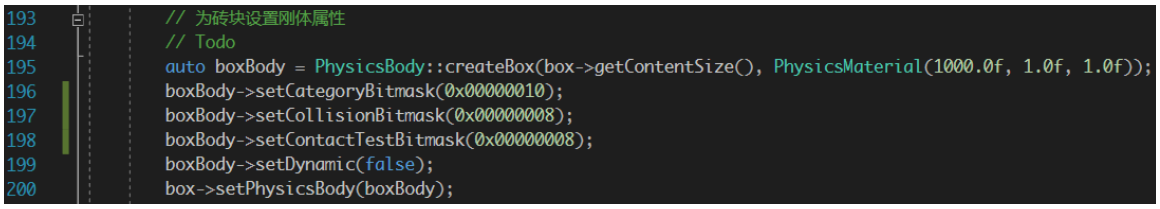
```cpp
	auto shipbody = PhysicsBody::createBox(ship->getContentSize(), PhysicsMaterial(100.0f, 0.5f, 1.0f));
	shipbody->setCategoryBitmask(0x00000002);
	shipbody->setCollisionBitmask(0xFFFFFFFF);
	shipbody->setContactTestBitmask(0x00000008);
```



```cpp
// 设置板的刚体属性
// Todo
auto barBody = PhysicsBody::createBox(player->getContentSize(), PhysicsMaterial(1.1f, 1.0f, 0.5f));
barBody->setCategoryBitmask(0x00000004);
barBody->setCollisionBitmask(0x00000008);
barBody->setContactTestBitmask(0x00000001);
barBody->setDynamic(false);
player->setPhysicsBody(barBody);
```

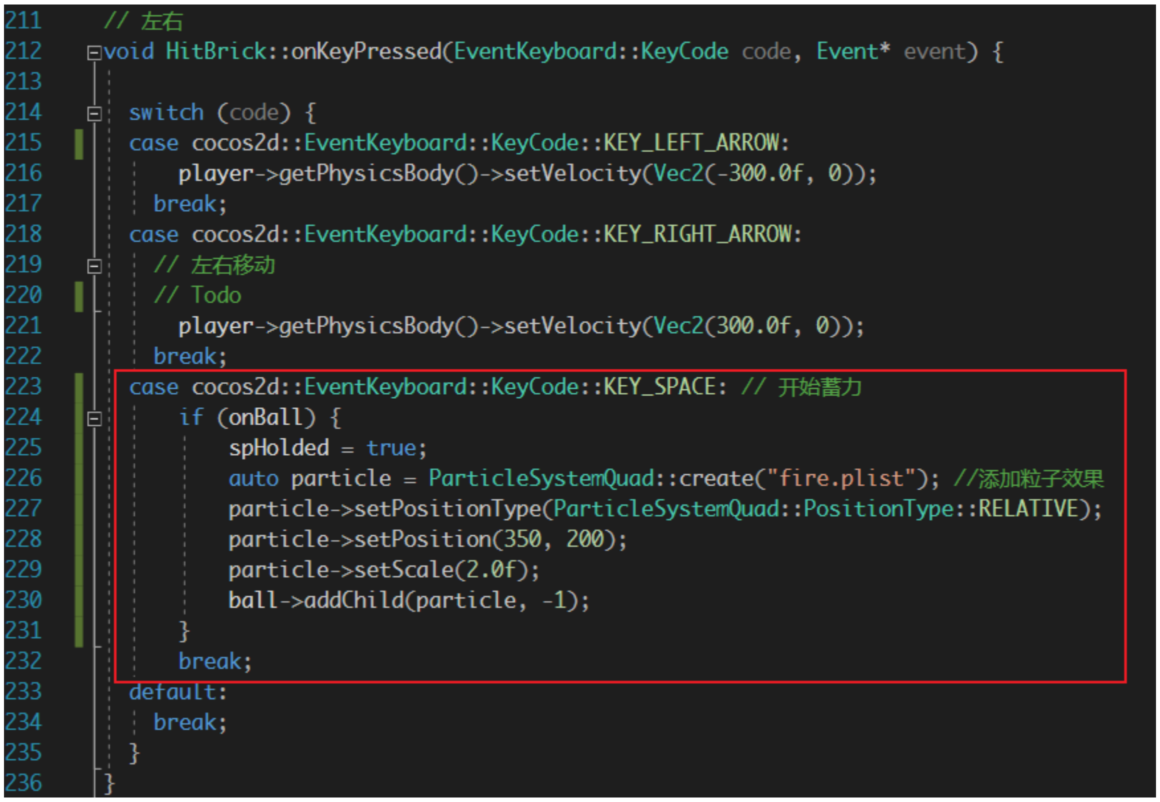


```
// 设置球的刚体属性
// Todo
auto ballBody = PhysicsBody::createCircle(ball->getContentSize().height / 2, PhysicsMaterial(5.0f, 1, 0.3f));
ballBody->setCategoryBitmask(0x00000008);
ballBody->setCollisionBitmask(0xFFFFFFFF);
ballBody->setContactTestBitmask(0x00000012);

ballBody->setRotationEnable(false);
ball->setPhysicsBody(ballBody);
```

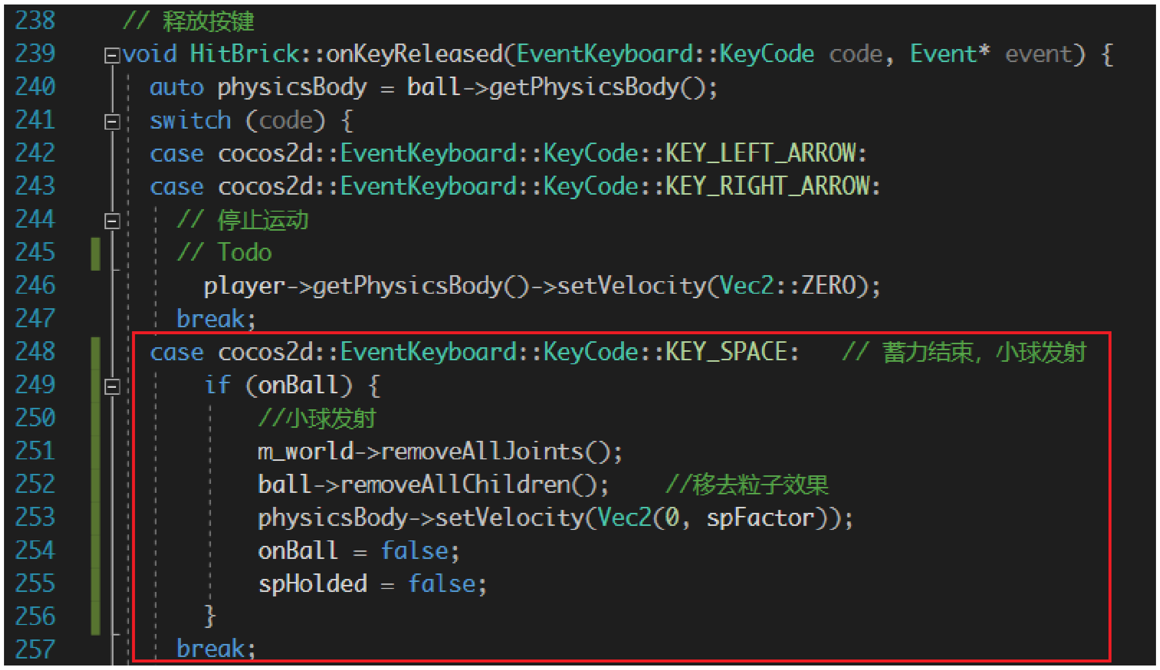


```
        // 为砖块设置刚体属性
        // Todo
        auto boxBody = PhysicsBody::createBox(box->getContentSize(), PhysicsMaterial(1000.0f, 1.0f, 1.0f));
        boxBody->setCategoryBitmask(0x00000010);
        boxBody->setCollisionBitmask(0x00000008);
        boxBody->setContactTestBitmask(0x00000008);
        boxBody->setDynamic(false);
        box->setPhysicsBody(boxBody);
```

5. 蓄力发射小球

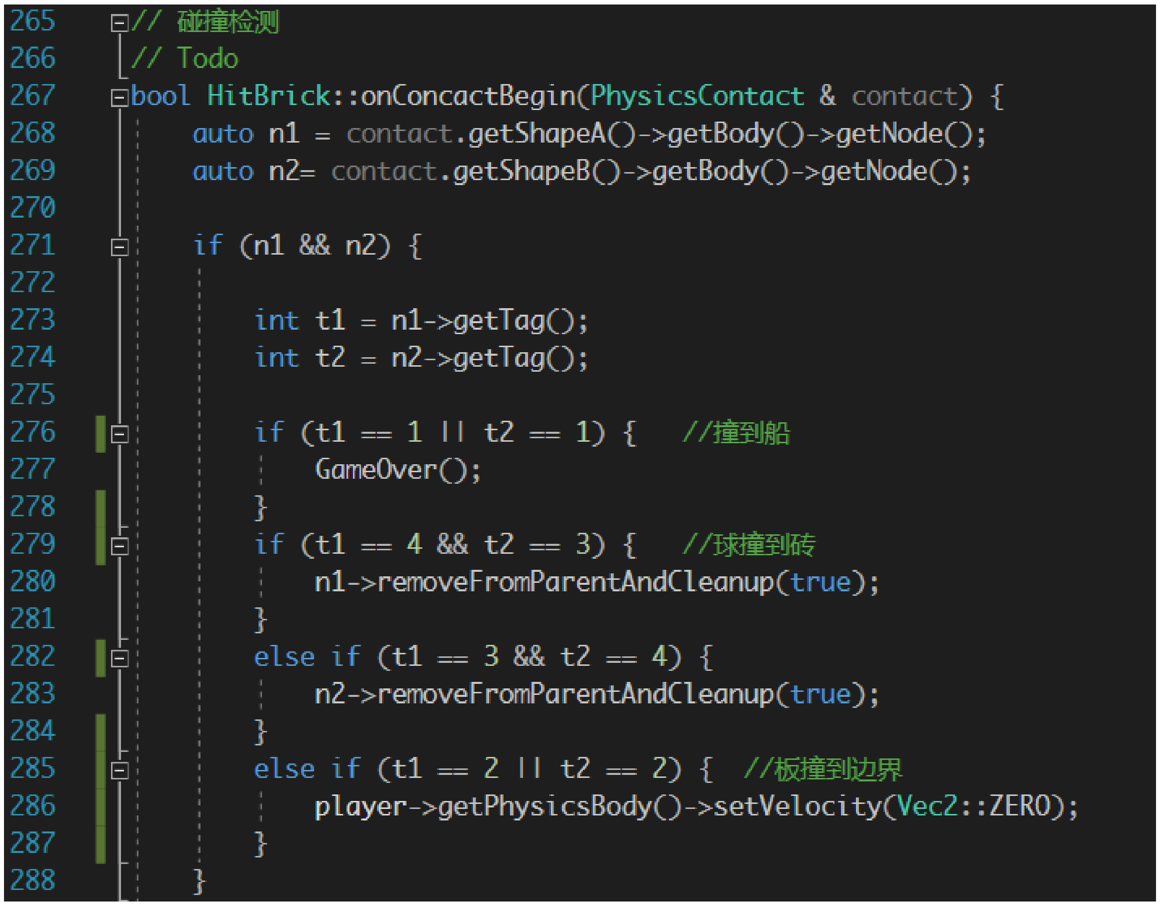


```
 // 左右
void HitBrick::onKeyPressed(EventKeyboard::KeyCode code, Event* event) {

  switch (code) {
  case cocos2d::EventKeyboard::KeyCode::KEY_LEFT_ARROW:
      player->getPhysicsBody()->setVelocity(Vec2(-300.0f, 0));
    break;
  case cocos2d::EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
    // 左右移动
    // Todo
      player->getPhysicsBody()->setVelocity(Vec2(300.0f, 0));
    break;
  case cocos2d::EventKeyboard::KeyCode::KEY_SPACE: // 开始蓄力
      if (onBall) {
          spHolded = true;
          auto particle = ParticleSystemQuad::create("fire.plist"); //添加粒子效果
          particle->setPositionType(ParticleSystemQuad::PositionType::RELATIVE);
          particle->setPosition(350, 200);
          particle->setScale(2.0f);
          ball->addChild(particle, -1);
      }
      break;
  default:
    break;
  }
}
```



```
// 释放按键
void HitBrick::onKeyReleased(EventKeyboard::KeyCode code, Event* event) {
  auto physicsBody = ball->getPhysicsBody();
  switch (code) {
  case cocos2d::EventKeyboard::KeyCode::KEY_LEFT_ARROW:
  case cocos2d::EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
    // 停止运动
    // Todo
      player->getPhysicsBody()->setVelocity(Vec2::ZERO);
    break;
  case cocos2d::EventKeyboard::KeyCode::KEY_SPACE:   // 蓄力结束，小球发射
      if (onBall) {
        //小球发射
        m_world->removeAllJoints();
        ball->removeAllChildren();    //移去粒子效果
        physicsBody->setVelocity(Vec2(0, spFactor));
        onBall = false;
        spHolded = false;
      }
    break;
```

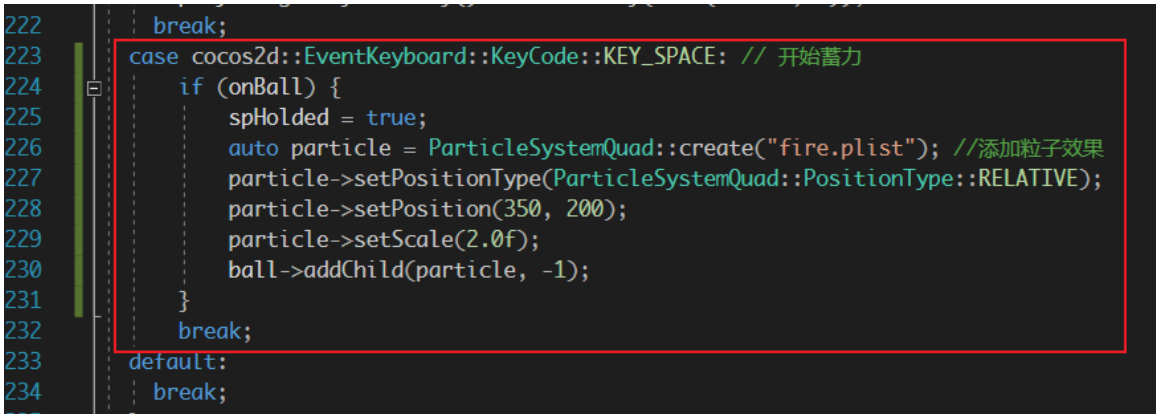
6. 砖、球碰撞则消去砖头，球与地板碰撞则游戏结束



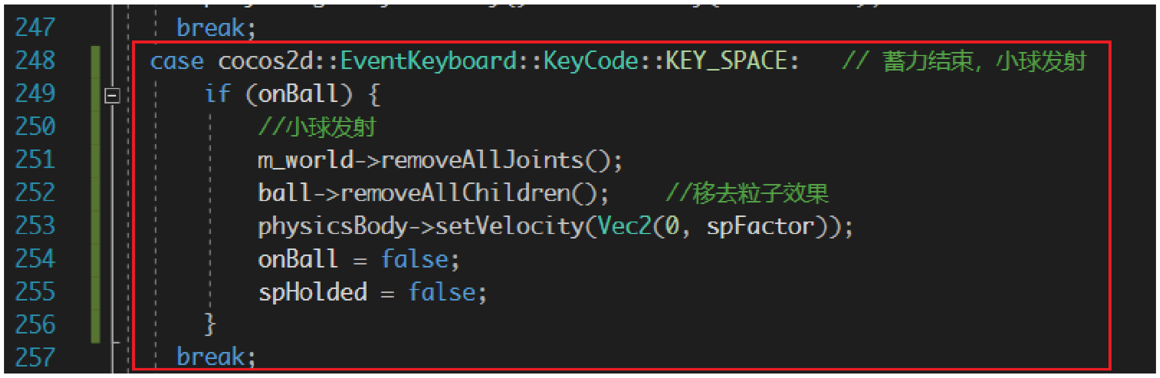
```
// 碰撞检测
// Todo
bool HitBrick::onConcactBegin(PhysicsContact & contact) {
    auto n1 = contact.getShapeA()->getBody()->getNode();
    auto n2= contact.getShapeB()->getBody()->getNode();

    if (n1 && n2) {

        int t1 = n1->getTag();
        int t2 = n2->getTag();

        if (t1 == 1 || t2 == 1) {   //撞到船
            GameOver();
        }
        if (t1 == 4 && t2 == 3) {   //球撞到砖
            n1->removeFromParentAndCleanup(true);
        }
        else if (t1 == 3 && t2 == 4) {
            n2->removeFromParentAndCleanup(true);
        }
        else if (t1 == 2 || t2 == 2) {  //板撞到边界
            player->getPhysicsBody()->setVelocity(Vec2::ZERO);
        }
    }
```

加分项：至少使用一种粒子效果。

（在蓄力时有粒子效果，发射后除去粒子效果。）



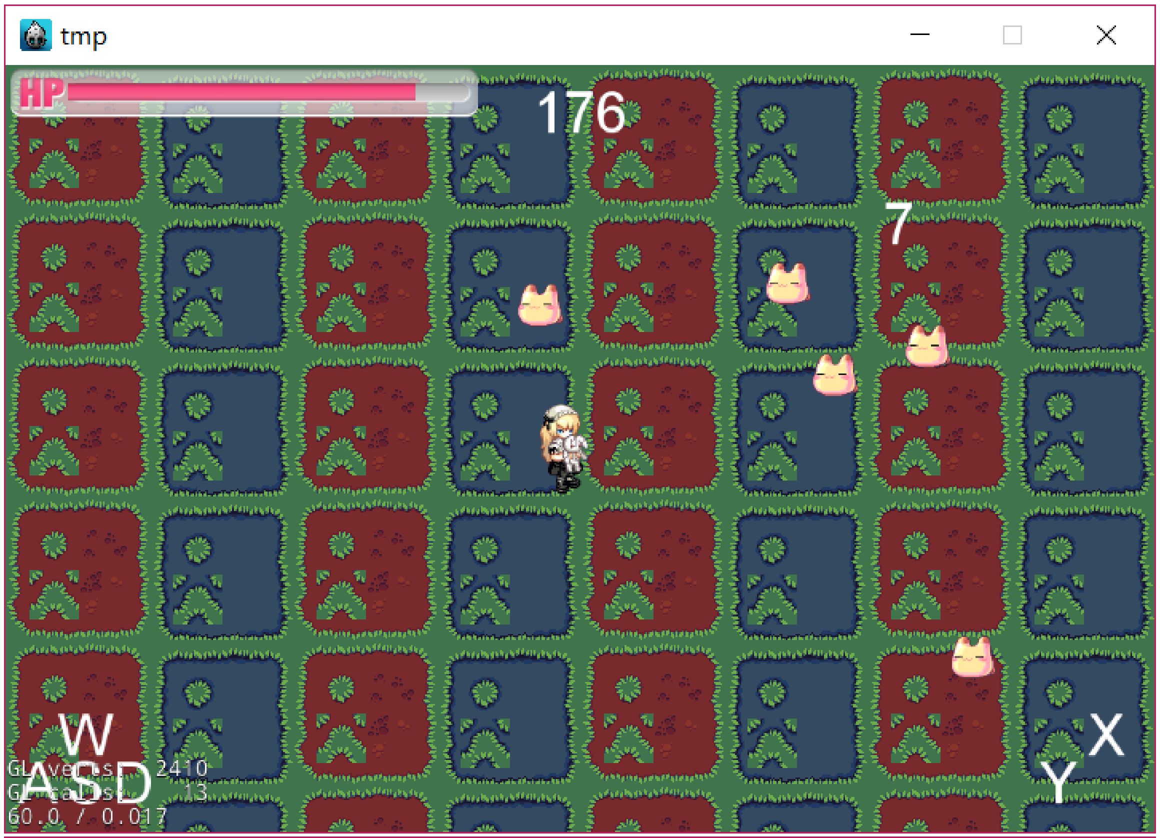
```
    break;
case cocos2d::EventKeyboard::KeyCode::KEY_SPACE: // 开始蓄力
    if (onBall) {
        spHolded = true;
        auto particle = ParticleSystemQuad::create("fire.plist"); //添加粒子效果
        particle->setPositionType(ParticleSystemQuad::PositionType::RELATIVE);
        particle->setPosition(350, 200);
        particle->setScale(2.0f);
        ball->addChild(particle, -1);
    }
    break;
default:
    break;
```

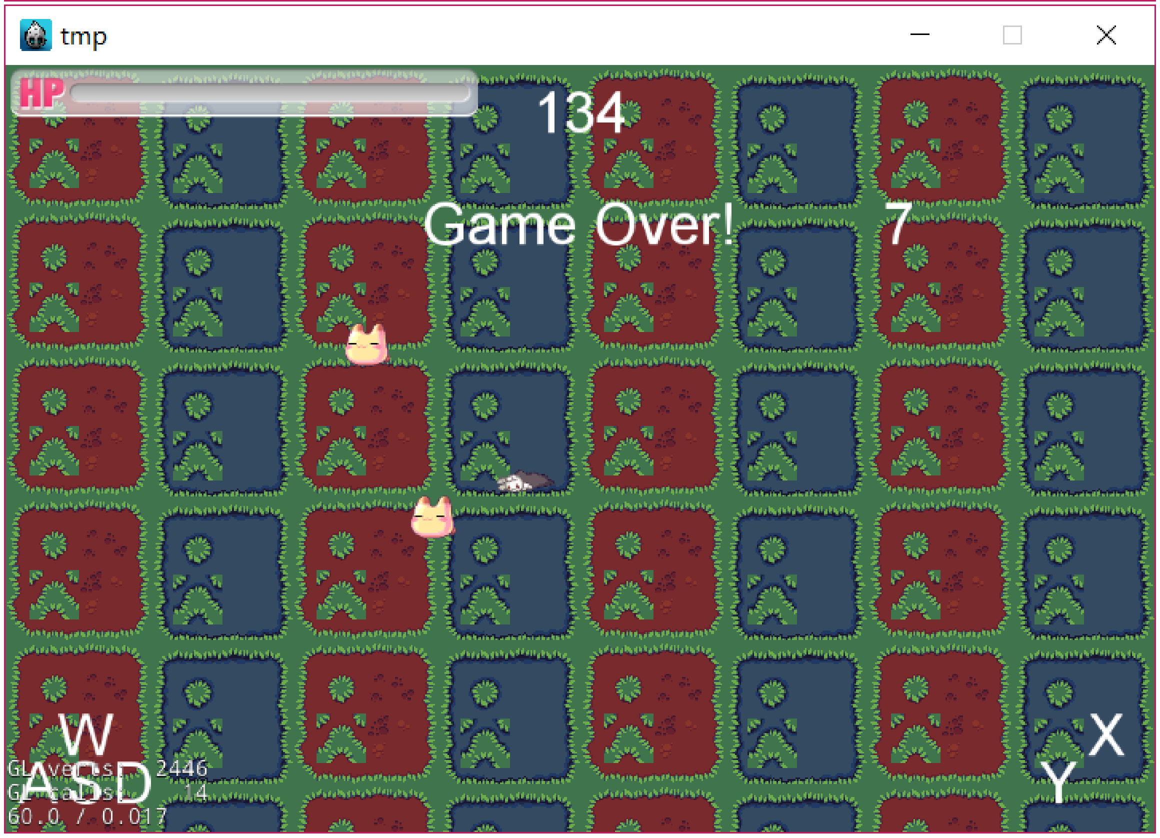


```
    break;
case cocos2d::EventKeyboard::KeyCode::KEY_SPACE:   // 蓄力结束，小球发射
    if (onBall) {
        //小球发射
        m_world->removeAllJoints();
        ball->removeAllChildren();    //移去粒子效果
        physicsBody->setVelocity(Vec2(0, spFactor));
        onBall = false;
        spHolded = false;
    }
    break;
```

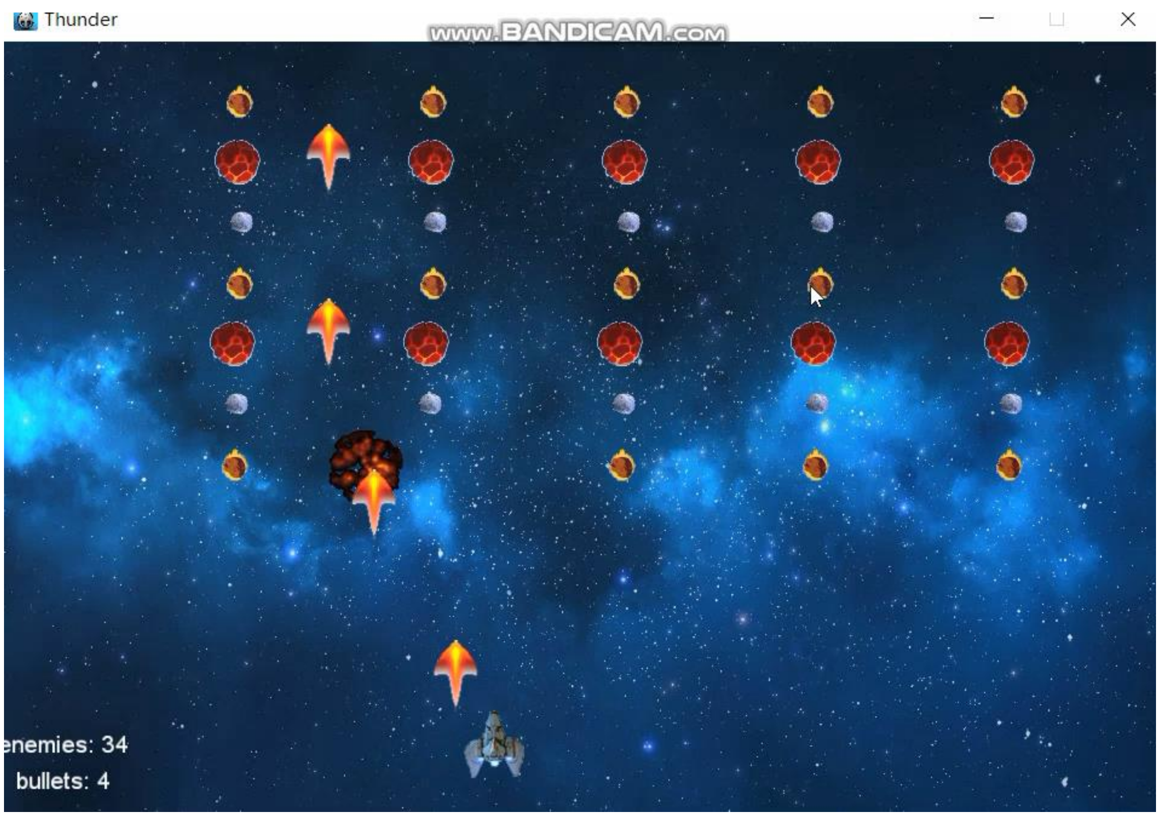
三、关键步骤截图

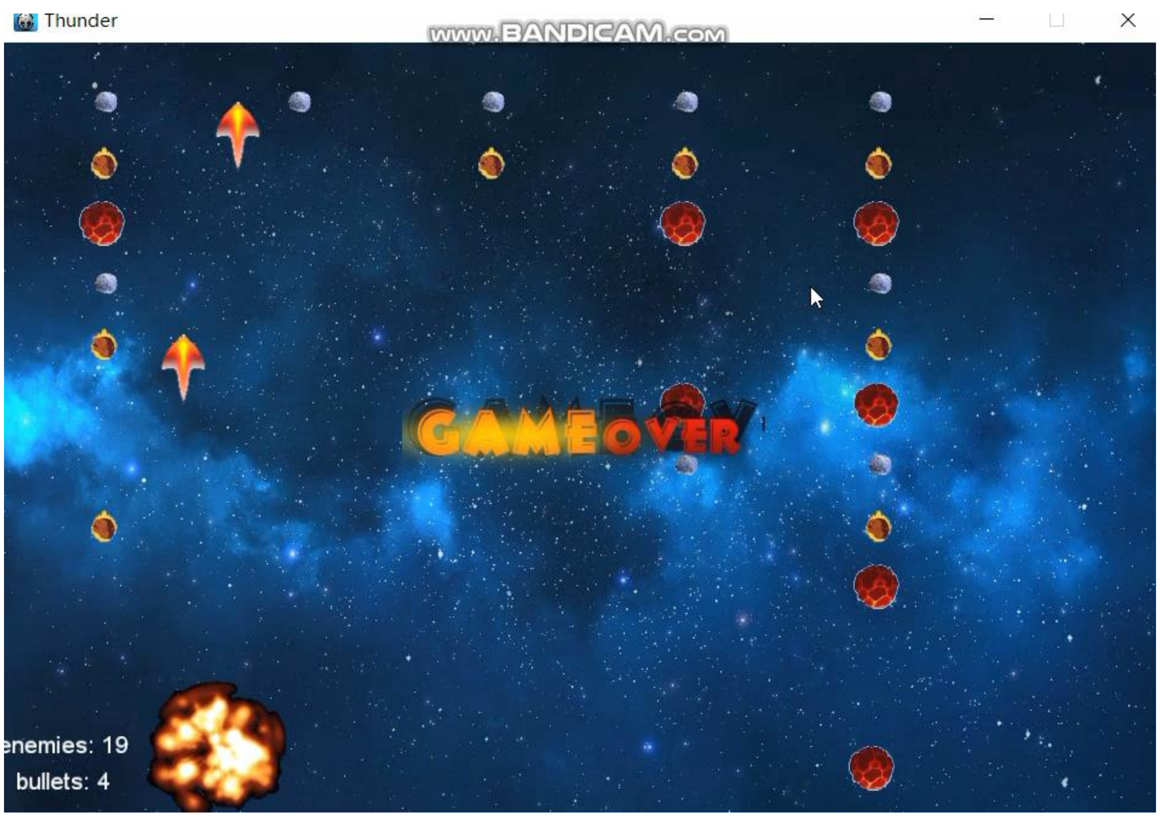
Week13:


tmp
HP
176
7
W
A S D
X
Y
GL verts: 2410
GL calls: 13
60.0 / 0.017


tmp
HP
134
Game Over!
7
W
A S D
X
Y
GL verts: 2446
GL calls: 14
60.0 / 0.017

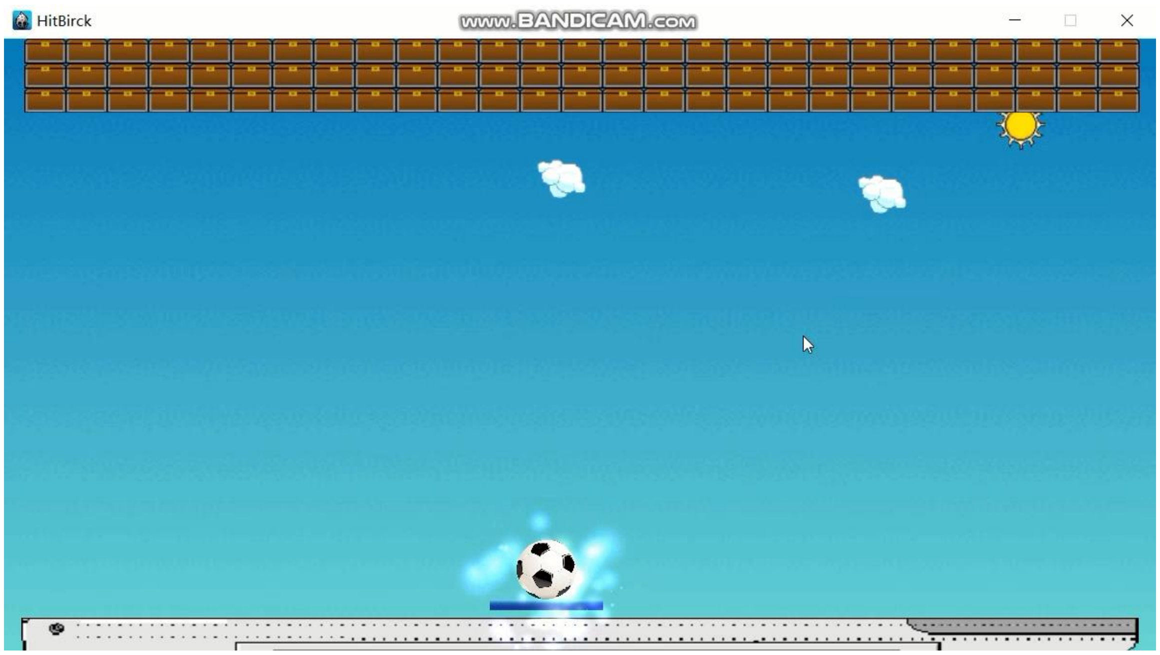
Week14:



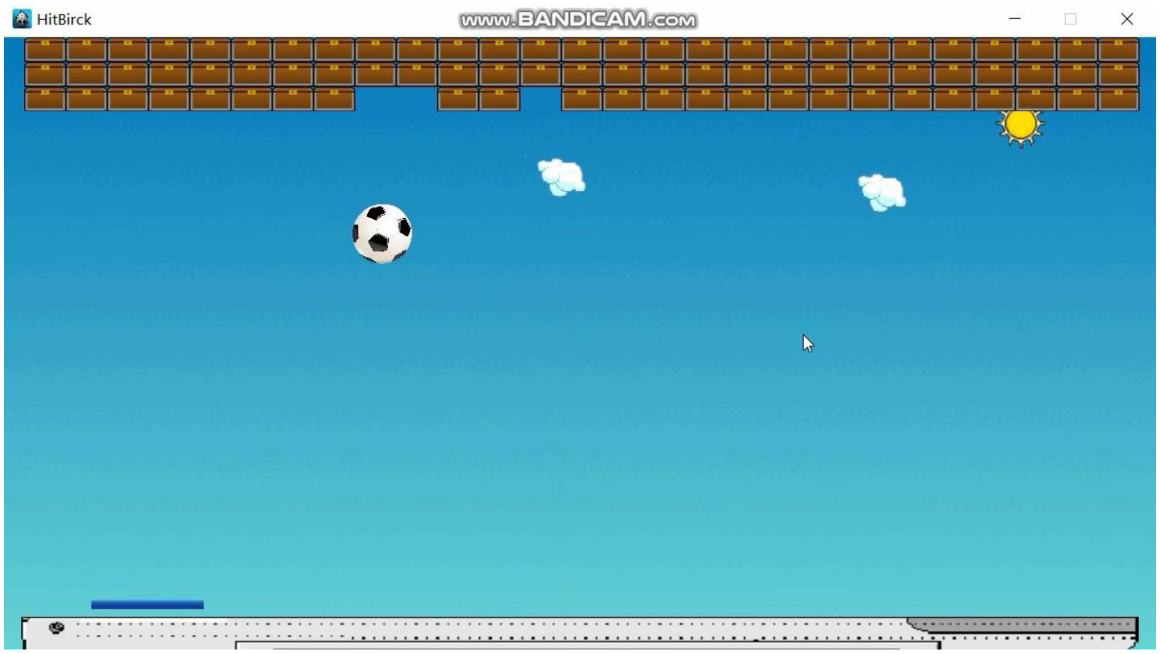


Week15:

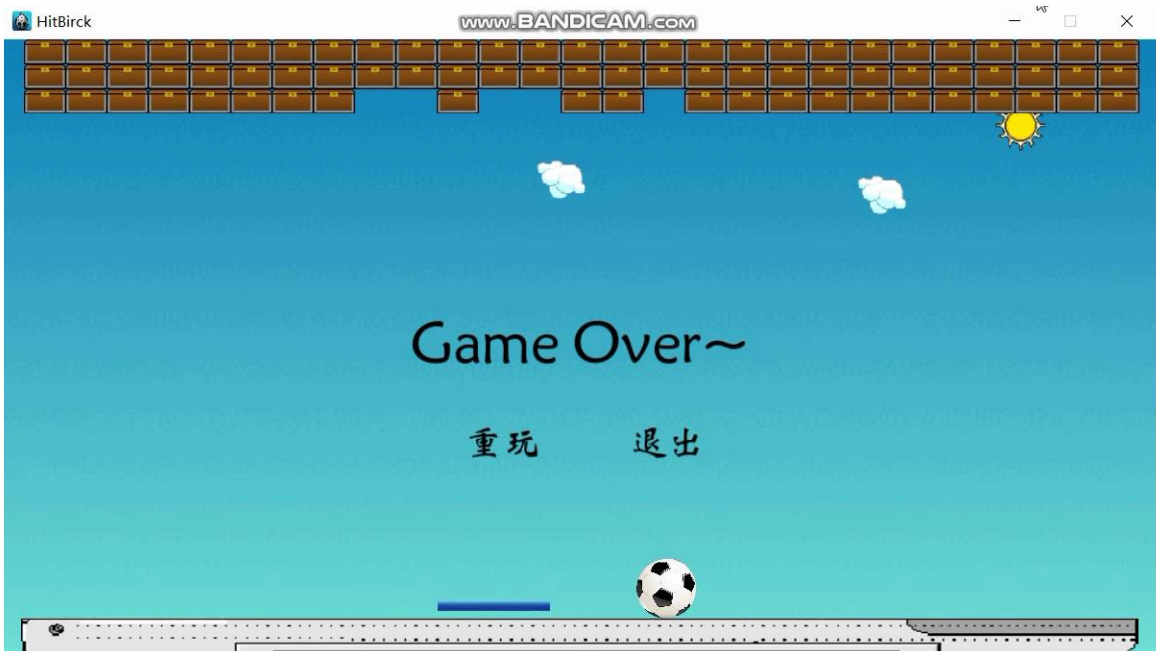
蓄力中:



消灭砖块并反弹：



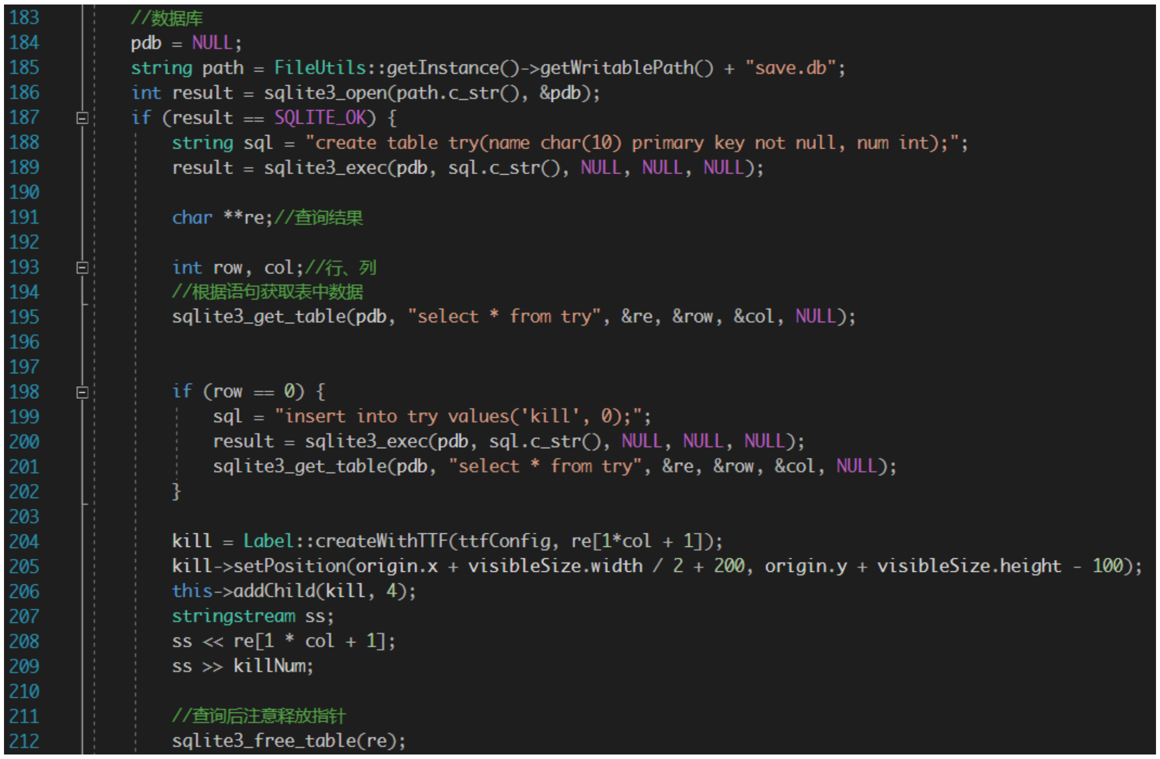
游戏结束：



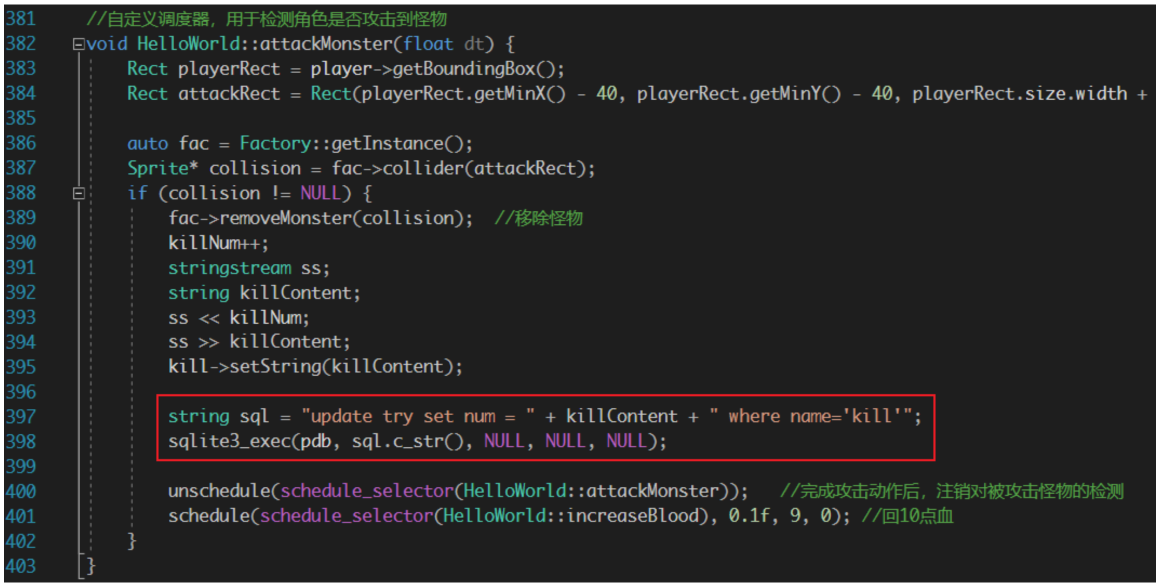
四、亮点与改进（可选）

Week13:

使用本地数据存储，记录打到的怪物数量，同时在游戏中显示打倒数量



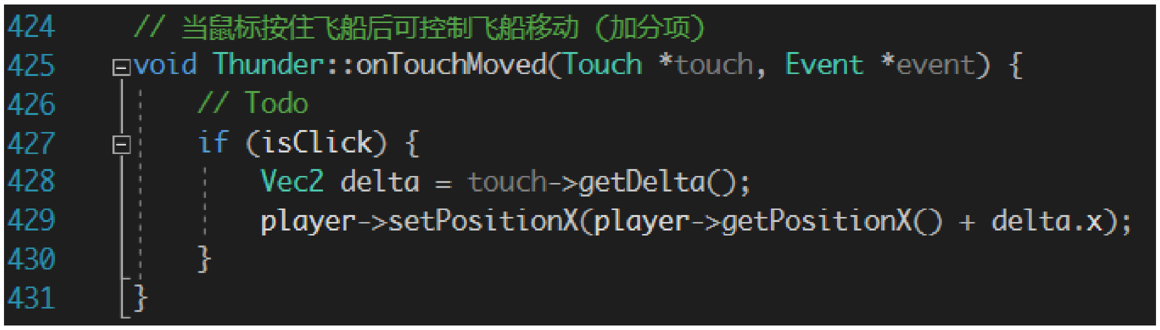
```cpp
    //数据库
    pdb = NULL;
    string path = FileUtils::getInstance()->getWritablePath() + "save.db";
    int result = sqlite3_open(path.c_str(), &pdb);
    if (result == SQLITE_OK) {
        string sql = "create table try(name char(10) primary key not null, num int);";
        result = sqlite3_exec(pdb, sql.c_str(), NULL, NULL, NULL);

        char **re;//查询结果

        int row, col;//行、列
        //根据语句获取表中数据
        sqlite3_get_table(pdb, "select * from try", &re, &row, &col, NULL);


        if (row == 0) {
            sql = "insert into try values('kill', 0);";
            result = sqlite3_exec(pdb, sql.c_str(), NULL, NULL, NULL);
            sqlite3_get_table(pdb, "select * from try", &re, &row, &col, NULL);
        }

        kill = Label::createWithTTF(ttfConfig, re[1*col + 1]);
        kill->setPosition(origin.x + visibleSize.width / 2 + 200, origin.y + visibleSize.height - 100);
        this->addChild(kill, 4);
        stringstream ss;
        ss << re[1 * col + 1];
        ss >> killNum;

        //查询后注意释放指针
        sqlite3_free_table(re);
```



```cpp
//自定义调度器，用于检测角色是否攻击到怪物
void HelloWorld::attackMonster(float dt) {
    Rect playerRect = player->getBoundingBox();
    Rect attackRect = Rect(playerRect.getMinX() - 40, playerRect.getMinY() - 40, playerRect.size.width +

    auto fac = Factory::getInstance();
    Sprite* collision = fac->collider(attackRect);
    if (collision != NULL) {
        fac->removeMonster(collision);  //移除怪物
        killNum++;
        stringstream ss;
        string killContent;
        ss << killNum;
        ss >> killContent;
        kill->setString(killContent);

        string sql = "update try set num = " + killContent + " where name='kill'";
        sqlite3_exec(pdb, sql.c_str(), NULL, NULL, NULL);

        unschedule(schedule_selector(HelloWorld::attackMonster));   //完成攻击动作后，注销对被攻击怪物的检测
        schedule(schedule_selector(HelloWorld::increaseBlood), 0.1f, 9, 0); //回10点血
    }
}
```

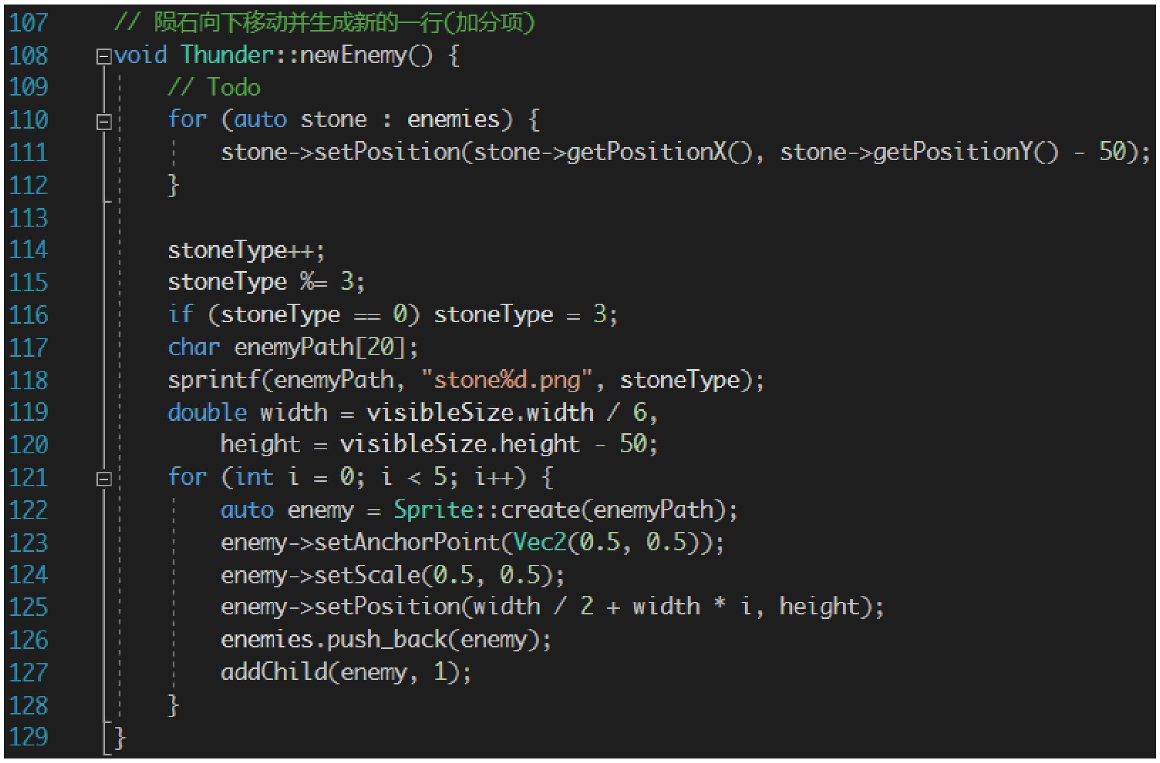
Week14:

1. 利用触摸事件实现飞船移动。（点击飞船后拖动鼠标）



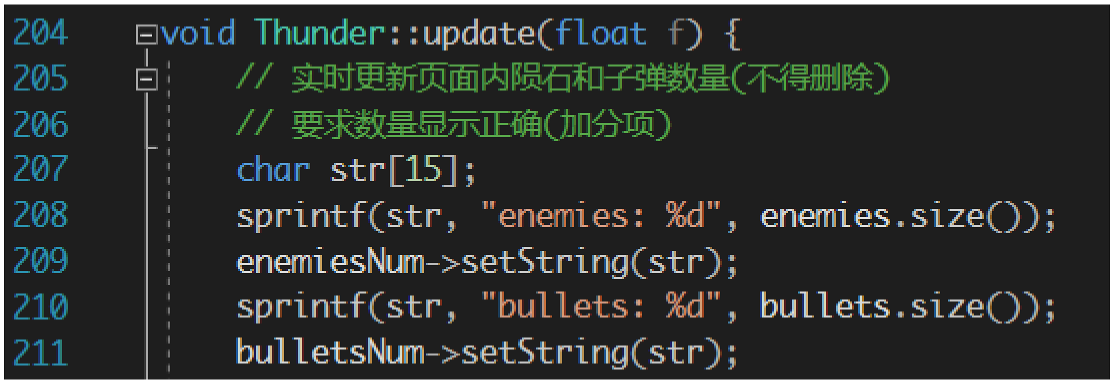
```
// 当鼠标按住飞船后可控制飞船移动 (加分项)
void Thunder::onTouchMoved(Touch *touch, Event *event) {
	// Todo
	if (isClick) {
		Vec2 delta = touch->getDelta();
		player->setPositionX(player->getPositionX() + delta.x);
	}
}
```

2. 陨石向下移动并生成新的一行陨石



```
// 陨石向下移动并生成新的一行(加分项)
void Thunder::newEnemy() {
	// Todo
	for (auto stone : enemies) {
		stone->setPosition(stone->getPositionX(), stone->getPositionY() - 50);
	}

	stoneType++;
	stoneType %= 3;
	if (stoneType == 0) stoneType = 3;
	char enemyPath[20];
	sprintf(enemyPath, "stone%d.png", stoneType);
	double width = visibleSize.width / 6,
		height = visibleSize.height - 50;
	for (int i = 0; i < 5; i++) {
		auto enemy = Sprite::create(enemyPath);
		enemy->setAnchorPoint(Vec2(0.5, 0.5));
		enemy->setScale(0.5, 0.5);
		enemy->setPosition(width / 2 + width * i, height);
		enemies.push_back(enemy);
		addChild(enemy, 1);
	}
}
```

3. 子弹和陨石的数量显示正确

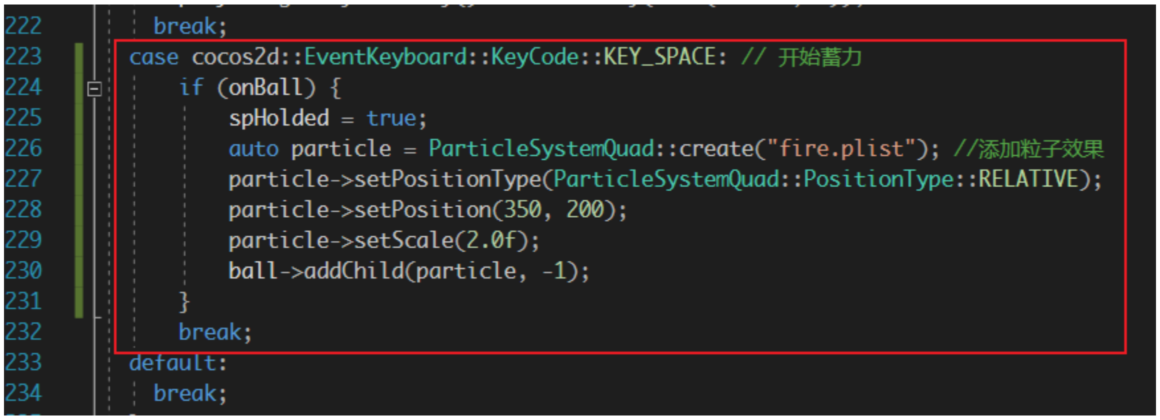


```
void Thunder::update(float f) {
	// 实时更新页面内陨石和子弹数量(不得删除)
	// 要求数量显示正确(加分项)
	char str[15];
	sprintf(str, "enemies: %d", enemies.size());
	enemiesNum->setString(str);
	sprintf(str, "bullets: %d", bullets.size());
	bulletsNum->setString(str);
```

Week15:

至少使用一种粒子效果。

（在蓄力时有粒子效果，发射后除去粒子效果。）



```
    break;
case cocos2d::EventKeyboard::KeyCode::KEY_SPACE: // 开始蓄力
    if (onBall) {
        spHolded = true;
        auto particle = ParticleSystemQuad::create("fire.plist"); //添加粒子效果
        particle->setPositionType(ParticleSystemQuad::PositionType::RELATIVE);
        particle->setPosition(350, 200);
        particle->setScale(2.0f);
        ball->addChild(particle, -1);
    }
    break;
default:
    break;
```

五、遇到的问题

Week14:

如果没有使用迭代器来修改列表中的值,而是取列表中的存放的某个指针给它赋值,

类似于

```
for each(Sprite* shoot in enemies){
    //...
    shoot = nullptr;
}
```

这样实际上只是改变了这个引用的指向，而没有修改实际上要修改的变量。会在清除子弹的过程中出错。

Week15:

关于板子的移动，一开始是使用 setPosition 方法，但是这样会导致足球无法和板子同步移动，足球会出现滞后。所以必须改用物理引擎的 setVelocity 方法来使板子移动。

六、思考与总结

1. 使用 C++编写游戏，基础概念很重要，尤其是指针，如果对它理解失当，就会出现类似于上面所说的那样直接修改指针类型变量而引发的错误。
2. 通过使用物理引擎和碰撞检测，可以在玩家碰撞到边界时就使玩家停下，相较于先前的在 Update 方法里利用位置反复检测玩家是否出界，这大大方便了对防止玩家出界的限制。同时，使用物理引擎的 setVelocity 方法来移动玩家也比利用 moveTo 或 setPosition 方便很多。合理利用好物理引擎，我们可以实现和简化很多事情。
3. 这三周我也学习到了怎么给游戏添加丰富多彩的效果，比如使用 Tilemap 来搭建游戏地图，比如使用粒子系统来添加华丽的特效。