面向对象程序设计课程项目文档

项目名称：单机版俄罗斯方块		年级：	16 级
指导教师：朱宏明		姓名：	毕家瑞
		学号：	1551615

目录

1. 概述

1.1 游戏类型：

2d 消除类、音乐节奏类

1.2 内容概述

俄罗斯方块是一款从红白机时代就经久不衰的的经典游戏，上亿的销量证实了这款游戏的游戏性与耐玩度。而我设计的这款俄罗斯方块除了传统的游戏方式外，极具创造性的将俄罗斯方块与现在非常热门的音乐节奏类游戏进行了结合，创造出了节奏俄罗斯方块的玩法，这种玩法为游戏界首创，虽然设计尚未成熟，但以足以颠覆以往玩家对俄罗斯方块的认识，将音乐与俄罗斯方块结合，大大提高了俄罗斯方块的趣味性与游戏性。

1.3 目标人群

俄罗斯方块游戏爱好者、节奏音乐类游戏爱好者

1.4 主要特点

1.画面：游戏界面简洁，游戏场景的背景画面采用色彩鲜艳的扁平化背景设计，消除和连击时有炫酷特效。

2.游戏内容：目前游戏有经典俄罗斯方块、本地多人和音乐节奏 3 个模式。

3.游戏体积小，便于后续的进一步开发 。

1.5 文档目的

对游戏的玩法，游戏开发过程遇到的问题及解决方法，游戏中各个系统和之间的关联及游戏用到的多媒体素材等进行介绍。说明游戏设计的思路。这样可以在后续的开发过程中能够有所参考游戏中的各种设置，避免开发后期对重要模块的修改。

2. 游戏实现的功能

2.1 基础功能：

随机方块、方块下落、碰撞检测、鼠标或键盘控制

排行榜

可修改难度

支持动画

2.2 额外功能：

预览下一个方块

本地双人模式

音乐节奏模式

可更换歌曲

2.3 用到的 c++特性

STL 容器，如：std::vector、std::array

类和多态

函数重载

初始化列表

3. 游戏操作

单人模式中使用 AD（或左右方向键）控制方块移动，S（或下方向键）控制方块快速下落，W(或上方向键)控制方块翻滚。

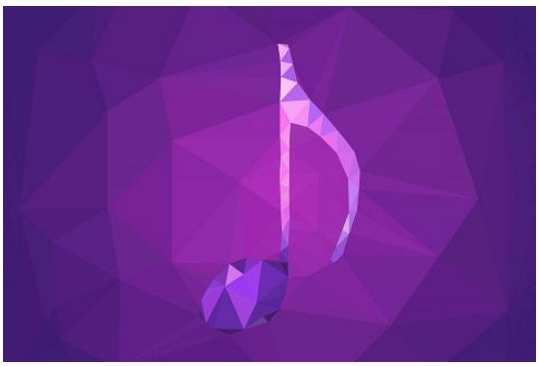
双人模式中 WASD 控制 Player1 界面，方向键控制 Player2 界面。

4. 多媒体素材

4.1 美工

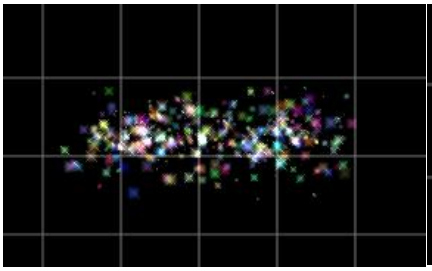
主要有游戏场景的背景制作和 GameLayer 中用到的粒子效果的制作。场景的背景设计和制作采用了扁平化的设计风格，





经典游戏模式的背景采用了玩家们熟知的游戏形象并进行了扁平化艺术处理，拉近了玩家与这款游戏的距离，在游戏制作中，背景的设计花费了很多时间，让各个模式的背景都有自己的鲜明特点，同时又让整个游戏的风格和谐一致，对增加游戏的趣味性有很大帮助。

4.2 粒子效果




游戏使用的是自己设计的粒子效果，第一个粒子效果会在消除方块时使用，多彩的粒子模拟出方块爆炸消除的画面，第二种特效在玩家一击消除多个方块时出现，并伴随有甜美的加油声音，提高玩家游戏的兴致。

4.3 音乐及音效

4.3.1 音效

游戏的音效有方块旋转音效，下落音效，方块消除音效和连击消除的人声音效。

4.3.2 音乐

单人经典模式采用的是一个慵懒但节奏分明的歌曲，能让人有玩上一整个下午的冲动。而本地双人模式的音乐采用最经典的俄罗斯方块背景音乐《Korobelniki》的管乐版本，歌曲旋律更悠扬轻快。使用经典音乐也能激起玩家的怀旧之情。

音乐模式目前选入了 3 首节奏分明的歌《Bad Apples!!》《The Hampster Dance Song》和《We No Speak Americano》，使游戏过程更欢快愉悦。游戏制作过程中为了使节奏模式达到最好的体验，在寻找节拍稳定，节奏明显并且十分欢乐的歌曲上下了很大功夫，并且对歌曲进行了一定的剪辑，以达到节奏模式的最佳效果。

5. 游戏主要架构相关

5.1 方块逻辑 Tetris 类

方块逻辑类是俄罗斯方块最核心逻辑，包含了方块的随机产生，移动，翻滚，下落和碰撞检测等，是这款俄罗斯方块游戏成型的基础。俄罗斯方块的逻辑受上学期五子棋项目的影响，采用二维数组来布置俄罗斯方块的棋盘，数字 1~7 为方块类型号，0 为空棋盘的值。方块类中重要的成员变量有 `BrickCellValue *m_brickMatrix;`整个方块矩阵 `int m_iPosRow, m_iPosCol;BrickType m_currentBrickType;`这 3 个成员变量代表当前移动的方块的位置和数值。`BrickCellValue m_currentBrick[4][4];`这个成员变量储存的内容是当前正在控制的方块。`BrickType m_nextBrickType; BrickCellValue m_nextBrick[4][4];`为预览方块信息。

7 种方块的产生创造性的使用 7 个长 8 位的 16 进制数通过位移操作来填充 4*4 方块矩阵，实现代码如下：

```
unsigned char Tetris::m_ucBrickMode[7] = { 0x66, 0xe4, 0xe2, 0xe8, 0x6c, 0xc6, 0xf0 };

for (int r = 0; r < 2; ++r) {
```

```
        for (int c = 0; c< 4; ++c) {
            if (((Tetris::m_ucBrickMode[brickType] << (r * 4 + c)) & 0x80) > 0) {
                    currentBrick[r][c] = brickCellValue;
                // 方块矩阵的这个位置有效，且方块矩阵的这个位置也有值，则游戏结束
                int offset = (posRow + (1 + r))*column + (posCol + c);
                if (posRow + (1 + r) >= 0 && brickMatrix[offset] != BrickCellValue_NULL) {
                    return false;
                }
            }
        }
    }
```

新方块的产生逻辑中如果无法产生，就判断游戏失败 return false。方块的下落，左移和右移操作尤其是旋转操作的实现进行了很多尝试和努力。最终对如何判断方块可以进行移动和旋转采用复制出一个临时方块的方法，用临时方块去尝试移动和旋转的方法解决了方块运动和碰撞的问题。旋转的函数由于没有想到一个简单的表达式来表达旋转的逻辑，所以代码过于复杂，在后续的开发中可以考虑适当修改。

方块类中方块操作的函数声明静态函数，这样即使声明多个 Tetris 变量核心函数仍然是保存在内存中的那一套函数，而为了方便调用，在 Tetris 类中声明了一组与静态函数重载的函数，使调用函数是更简洁并减少参数出错。

5.2 游戏的核心实现，GameLayer 类和 ControLayer 类

5.2.1 GameLayer

GameLayer 类实现了游戏层，通过调用一些 cocos 的方法将 Tetris 类抽象的逻辑实现为可视化的游戏界面。包括游戏矩阵的形成，预览方块的形成，游戏中各种声效和粒子效果的添加以及最重要的方块下落的实现。下落的实现在 starGame()方法里面用动作回调函数 CallFunc 实现：

```
    CallFunc *callFunc = CallFunc::create([=] {
        if (m_bGameRun) {
            this->fall();
        }
        else {
            m_delegate->gameOverAction();
        }
    });
        this->runAction(RepeatForever::create(Sequence::create(DelayTime::create(level()), callFunc,
NULL)));
```

其中 DelayTime::create(level())来控制方块的下落延迟，m_delegate->gameOverAction();是各个游戏模式结束游戏的行为各不相同，通过 GameLayerDelegate 类来提供接口。具体游戏场景继承 GameLayerDelegate 类后可通过将虚函数 gameOverAction()具体化来实现控制不同模式的游戏结束行为。GameLayer 类中控制粒子效果的函数是这样的：

```
void GameLayer::initPartical(int r)
{
    Size visibleSize = Director::getInstance()->getVisibleSize();
    m_Particle = ParticleSystemQuad::create("particles/2444.plist");
    m_Particle->setPosition(visibleSize.width*0.441, visibleSize.height *0.93 - r * 16);
    this->addChild(m_Particle,5);

    this->scheduleOnce(schedule_selector(GameLayer::deletePartical), m_Particle->getLife() + 1.0f);
}

void GameLayer::deletePartical(float dt)
{
    this->removeChild(m_Particle, true);
}
```

这两个函数是产生方块消除的粒子效果使调用的，由于方块消除的操作大量存在，然而粒子效果并不会自己移除，所以导致随着游戏进行场景中的 Node 数量越来越多，最终可能引起卡顿，这两个函数可以避免这种情况的发生。

5.2.2 ControlLayer

ControlLayer 中实现了一个摁键监听，并与 GameLayer 类似的有一个 Delegate 虚基类提供接口。使不同的游戏模式实现自己的操作方法，将 Control 的方法单独封装成类也方便以后为了移植其他平台添加虚拟摁键或滑动屏幕的监听提供方便。

5.3 3 个游戏场景

5.3.1 经典模式

由于 GameLayer 和 ControlLayer 已经将游戏最主要的功能实现完毕，经典模式的场景要做的工作非常少，只需要将场景设定好，控制设定好，并完成结束后保存分数和前往 ClassicGameOver 场景的 gameOverAction();就可以了。

5.3.2 本地双人模式

双人模式与经典模式类似，只是建立了两个 GameLayer 层并重写那两个 delegate 来控制这两个层。最初设置双人模式仅仅是为了试验联网双人应该是什么样子，然而游戏体验实在是很糟糕，游戏界面变小，几乎不存在任何对抗性，另一位玩家的操作还会让人走神，无法专心游戏，可以说俄罗斯方块的对战无法给人带来任何的愉悦性。这是我放弃联网对战模式专心写节奏模式的原因之一，文档最后会简述联网模式的大体实现方法。

5.3.3 节奏音乐模式

节奏音乐模式，我认为是整个游戏最精华也是最有趣的部分。俄罗斯方块诞生距今已经 33 年，然而这应该是历史上首次将节奏类游戏和俄罗斯方块结合起来。这种独创的节奏游戏既有一般节奏游戏的紧张感又有俄罗斯方块消除行的快乐。这个场景类使用 std::vector<std::string> music;和 std::vector<float> myRhythm;来存放歌曲名和歌曲的节奏，利用 vector 的性质方便在后续开发中添加歌曲和加入循环播放功能（由于游戏制作时间有限，未能实现）。

对于一款音乐游戏来说最重要的设计就是音乐节奏，和根据节奏进行操作的反馈。在这两点的设计上我曾查阅过一些资料，优秀的音乐游戏的节拍点都是人工输入进去的，但由于代码

水平和乐理知识有限，无法做到在极其精确的时间点显示节拍，所以歌曲选择只能采取稳定节拍的歌曲，并用 schedule 来控制产生稳定的节拍。节拍器的设计为了使玩家专心于方块矩阵操作而不分心于节拍，没有采用《太鼓达人》式的节拍器，而是使用模仿《啪嗒砰》的屏幕边框节拍器，用更简单的方法达到了更好的效果。节拍器的产生和消失以及摁键的配合采用了一组 scheduleOne 来实现，例如：

```
scheduleOnce(schedule_selector(RhythmGameScene::canClickTrue), 0.08f);
scheduleOnce(schedule_selector(RhythmGameScene::canSeeTrue), 0.15f);
scheduleOnce(schedule_selector(RhythmGameScene::canSeeFalse), 0.40f);
scheduleOnce(schedule_selector(RhythmGameScene::canClickFalse), 0.43f);
```

不同的歌曲可以通过微调上面四个函数的参数达到节拍和歌曲的完美配合。歌曲的选择界面使用了 TabView 实现歌曲目录的滚动，未来的开发中可能会在 TabView 加入歌曲的图片来提升游戏体验。音乐模式开发最大的难度在于歌曲的选择，剪辑去掉节奏紊乱的前奏和结尾，获取 BPM 和设置节拍，使音乐节拍和游戏的节拍器完美结合。制作场景时节拍器图片的构思和制作也费了不少心思。

5.3.4 游戏场景总结

真正实现三个完全不同的游戏场景的代码量都不多，主要是由于较好的设计了 Tetris 类以及 GameLayer 和 ControlLayer 类。使得开发新的游戏模式的工作量大大降低，重复的代码量也尽可能降至了最低。降低了后续编程和维护的难度。

5.4 场景切换

由于游戏是多模式多场景的游戏，所以高效的管理切换场景是游戏设计所必须的。场景切换的统一管理在 SceneManager 类中实现。在需要进行切换的场景或层中有成员变量 SceneManager * tsm;指针能够调用 SceneManager 类中实现的方法。

6. 一些遗憾

6.1 美术等游戏体验方面

在最初的游戏设想中，就将游戏的背景风格和色彩风格定位为扁平化图案并且有绚丽色彩的艺术风格。然而由于时间和精力有限，只完成了 3 个游戏场景的背景设计和制作，在未来的进一步开发中会逐步将游戏的背景进行完善，例如将经典游戏结束场景的背景继续用人们熟知的游戏人物形象进行设计，音乐游戏结束场景沿用紫色色系并拥有独特的音乐的符号等等。

当前版本的游戏中所有摁键都是采用了 Label 来制作的，其实 Label 制作摁键只是一种暂时性方案，同样由于时间有限，并没有专门进行摁键图片的设计，用 Label 来制作摁键确实影响了游戏体验，在未来的版本中可能会进行摁键图片的更新。

6.2 游戏内容相关

由于 Tetris 类和 GameLayer 类的设计比较完善，所以游戏添加新模式时的工作量会减少很多，但由于开发时间有限，一些游戏模式确实无法完成，这里只做一些思路设计，以便后续开发参考。

6.2.1 未实现的初始自定义开局

自定义开局的思路实际上是这样的，在 Tetris 类中加入一个 static vector<array<array<int,m_iRow>,m_iColumn > >来储存各种定义好的初始化开局，Tetris 的构造函数中多加入一个传入参数来代表使用何种初始开局，并改写当前构造函数和刷新矩阵的函数使矩阵按照 vector 中储存的模板填充，GameLayer 相应作出修改，GameLayerDelegate 添加一个函数实现具体场景可控制初始开局。这样设计的话之前 3 个游戏模式的代码只要根据新的 GameLayerDelegate 作出少量修改即可，但自定义开局的难度在于关卡的设计，新颖的关卡很可能需要手动输入坐标点来设计，工作量太大，所以最终没有制作这个模式。

6.2.2 未实现的联网对战

联网对战打算用 boost.asio 的 udp 通信来实现，重载 GameLayer 类中的 startGame（）、fall()和 throwDown()方法，使产生方块不是通过随机数，而是通过传入参数产生，GameLayer 再中添加两个返回型为 int 的 get 函数，返回方块类型和预览方块类型。联网设计的场景中设计两种游戏场景，分别对应服务端和客户端，服务端需要输入房间号作为开放的端口并调用 bind 方法绑定创建的端口，而客户端除了房间号还需要有一个额外文本输入框输入服务端 IP。由客户端发出开始游戏的指令。游戏场景的创建类似本地双人，只不过游戏层一个是从键盘读入信息，一个是从 scoket 获得信息。信息的获取类似碰撞监听或摁键监听，利用 schedule 来实现。

以上就是实现联网游戏的大体思路，我对 asio 网络编程的理解还十分有限，仅仅是按照教程编写过 2 个 test 来测试 udp 异步通信，如果要具体实现联网对战肯定同样要参考一些教程和文档，所以上述思路仅是一些初步构想，实际开发中肯定还会遇到各种各样的坑。若有任何不妥，还希望能多多指教。由于开发时间有限，双人模式体验糟糕，对联网编程的理解极其有限，最终放弃了联网对战的游戏模式，不过假期中时间充裕，会仔细学习一下 boost.asio 并使游戏加入联网模式。

7. 项目总结及心得

7.1 项目总结

这款游戏是一款独自开发完成的俄罗斯方块游戏，虽然游戏简单并且不支持联网，游戏的艺术资源并未制作完全，但玩家应该能体验到一种完全不同的俄罗斯方块游戏方式和一种新颖的艺术风格。游戏的代码我认为在几个核心类的设计上花费了很多心思，使游戏能够很方便的扩展功能，为游戏的后续开发和维护打下了良好了基础。

7.2 心得体会

写文档比写代码累多了 Orz