**目录**

1. **设计方案简介**
2. **具体实现原理介绍**

2.1状态机（StateMachine）和状态基类（BaseState）

2.2 五个游戏状态

2.2.1 开始状态（StartState）

2.2.2 游戏状态（PlayState）

2.2.3 游戏结束状态（GameoverState）

2.2.4 高分榜状态（HighScoreState）

2.2.5 结束状态（EndState）

2.2.6 状态转换示意

2.3 Cube类和Map类

2.3.1 Cube类

2.3.2 Map类

1. **成果展示**
2. **设计方案简介**

编写一个命令行的小程序，实现俄罗斯方块小游戏。实验中用状态机实现游戏中各个状态的转换和过渡。状态机利用c++的多态实现。每一个状态则由专门的类负责。其中游戏状态（PlayState）配合Cube类和Map类实现俄罗斯方块的核心游戏逻辑。

1. **具体实现原理介绍**

**2.1状态机（StateMachine）和状态基类（BaseState）**

StateMachine是一个类。在main（）函数中有一个死循环反复调用StateMahcine的update（）和render（）以及反复判断StateMachine的end是否为真，如果end为真，则跳出循环，结束游戏。

StateMachine成员函数功能简介：

void change（）：状态切换

void update（）：更新游戏数据

void render（）：将游戏内容打印输出至屏幕上

void stop（）：停止工作，将成员变量bool end设置为真

bool exit（）：判断是否退出，返回end的值

状态切换原理：

a)StateMachine配合BaseState这个类一起使用

b)BaseState是所有状态类的基类，它有五个虚函数，分别是enter（）（进入状态函数），bgrender（）（背景打印函数），update（）（游戏数据更新函数），render（）（实时打印函数），exit（）（退出状态函数）。五个函数的函数体皆为空。需要添加的状态通过继承重写BaseState的这五个函数的部分函数，即可让StateMachine进行调用。

c)StateMachine中有一个BaseState \*\*list和一个int current，用于维护一个BaseState \*的数组。数组中的每一个BaseState \*指向一个由BaseState派生出来的状态类。StateMachine的成员函数将会调用list[current]的相应函数。通过改变current的值，实现对不同派生状态类的调用。

**2.2 五个游戏状态**

* + 1. **开始状态（StartState）**

主要功能：呈现开始的界面。输出游戏标题tetris，玩家通过敲击空格键切换选项，回车选择开始游戏或者查看高分榜。

主要成员函数的功能和原理：

void update（）：检测用户输入，如果输入空格，切换选项。如果输入回车，根据选项调用StateMahicne的change（）函数切换游戏状态至PlayState，HighScoreState或者EndState。

void bgrender（）：背景的打印输出，即游戏标题和选项Start和HighScore的输出。

void render（）：根据update（）更新的选项信息，打印一个“=>”箭头符号显示玩家选的是Start还是HighScore。

* + 1. **游戏状态（PlayState）**

主要功能：实现俄罗斯方块的核心游戏逻辑。方块随机产生，可以旋转，左右移动，快速下落，下落以后满一行可以消除。实时显示玩家分数，提示下一个方块是什么。支持游戏暂停。

核心游戏原理：

a)PlayState配合Cube类和Map类一起使用（Cube类和Map类的具体细节将会在后文介绍）。其中Cube类负责管理正在下落的方块，Map类负责管理已经落下的方块。

b)PlayState有两个Cube \*指针，一个是Cube \*current，指向正在下落的方块，一个是Cube \*next，指向下一个方块。游戏过程中，current指向的方块落下以后，current将会指向next所指向的方块。next则指向current原本的方块，并且更新其数据，随机产生新的方块。

c)方块落下以后，PlayState会调用Map的addCube方法，将新落下的方块信息记录下来。

d)方块下落速度的快慢是通过调节Sleep（speed）函数中的speed参数的大小实现的。

主要成员函数的功能和原理：

void enter（）：进入PlayState的时候将会被StateMahcine调用，负责游戏数据的初始化，例如将玩家分数（由成员变量score记录）清零。

void update（）：调用Cube类和Map类的成员函数，实现游戏的暂停功能，实时更新游戏数据。将检测到的玩家输入交给Cube类的方法进行处理。游戏实现了暂停的功能，利用getch()的阻塞特性实现空格暂停。

void render（）：输出显示游戏界面，包括方块，玩家分数，下一个方块等等。

* + 1. **游戏结束状态（GameoverState）**

主要功能：游戏结束以后显示玩家分数和分数等级。之后提示玩家输入名字，并过渡到HighScoreState。

核心原理：主要难点在于状态变化时，不同状态信息的传递。PlayState需要告诉GameoverState玩家分数。GameoverState需要告诉HighScoreState玩家姓名和分数。这些在重写的enter（）函数中实现。

enter（string enterParams）函数接受一个string参数，将状态过渡的信息记录在string中完成信息传递。不同的信息用‘&’分隔，例如“name&score”传递名字-分数对。

* + 1. **高分榜状态（HighScoreState）**

主要功能：显示历史最高分的十个玩家名字及其分数。

核心原理：

a)为了使玩家再次进入游戏，高分榜依然存在，HighScoreState会把高分榜的信息写入同一个文件夹下的本地文件highscore.lst中。每次打开游戏，将会读取highscore.lst的信息。

b)高分榜规则如下：1）重复的玩家名字只取一个最高分2）显示历史最高分的十个玩家3）相同分数的玩家按照字典序排列

c)这里利用map和vector完成名字的不重复和按照分数打印信息这两个任务。其中vector配合sort函数完成高分榜排序。

* + 1. **结束状态（EndState）**

主要功能：游戏结束的过渡页面。倒计时三秒钟窗口关闭。

核心原理：利用Sleep函数完成计时功能。反复打印数字和‘\b’实现倒计时的效果。

* + 1. **状态转移关系**

GameoverState

PlayState

StartState

HighScoreState

EndState

注：“->”表示可由箭头始端状态转移至箭头指向的状态。StartState是游戏的起点。其中StartState，PlayState以及HighScoreState都可以由玩家按‘q’直接转化至EndState，结束游戏。

* 1. **Cube类和Map类**
     1. **Cube类**

主要功能：管理掉落的方块。可以实现方块的打印输出，信息更新，旋转，下落，碰撞检测等等。

核心原理简介：

a)方块的形状和颜色：利用了一个全局变量shapesheet记录方块的形状，Cube显示方块时，直接查表获取相应方块的形状信息。每个方块有一个（x， y）坐标，shapesheet则记录了四个小方块相对（x， y）的坐标。从而实现形状信息的记录。每一个新方块都通过随机数产生不同的形状和颜色。

b)碰撞检测：每个方块由四个小方块组成，检测碰撞时逐个检测小方块是否发生碰撞。

c)移动，下落和旋转：Cube类中有两套（x， y， rotation）（坐标和旋转角度）数据。当玩家旋转移动方块时，更新x, y, rotation。同时记录旧的x，y，rotation。如果碰撞了，则将x，y，rotation恢复，如果没有碰撞则不恢复。打印输出时先将旧的一套坐标的方块用打印‘ ’的方式擦去，再在新的一套坐标位置上打印方块。从而实现屏幕上显示的更新。注：此处有一个不同步更新的问题，方块每隔一段时间一定会下落一格。但是每下落一格的时间里方块可以发生多次平移和旋转。所以这里采用了循环多次检测用户输入，之后再下落一格的方法解决。

* + 1. **Map类**

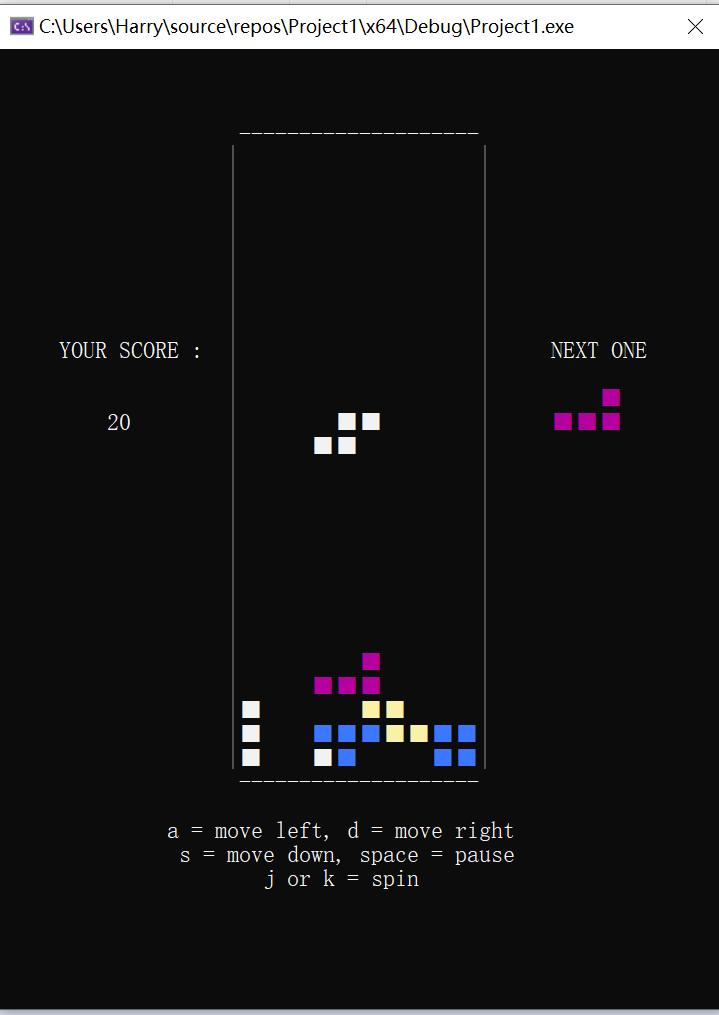
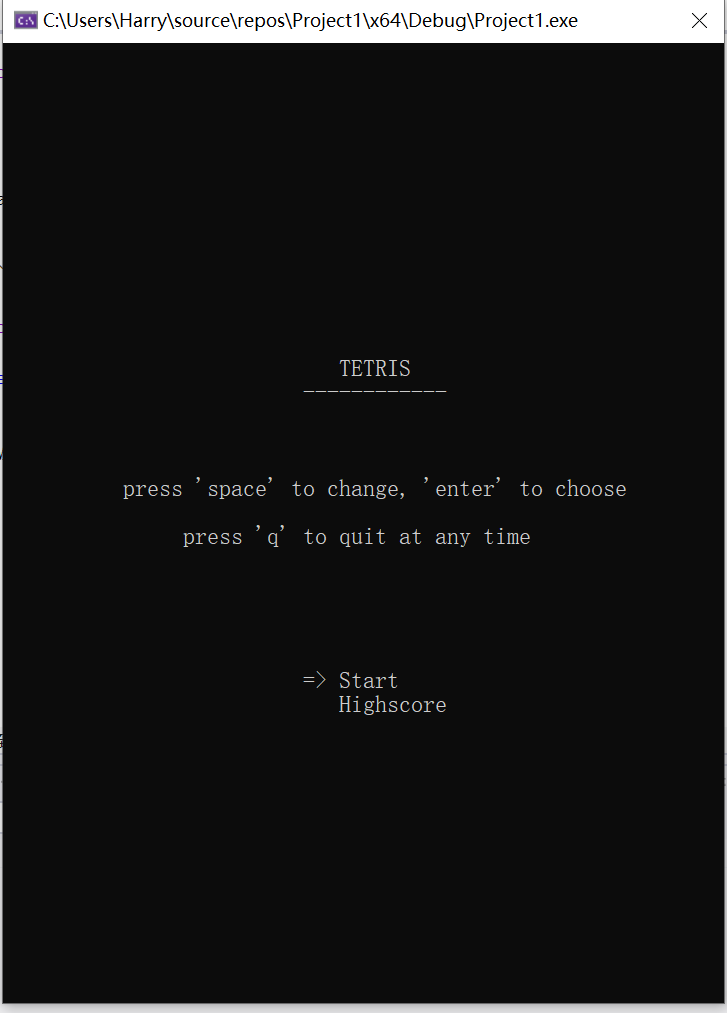
主要功能：负责管理已经落下的方块。

核心原理：

a)每落下一个方块，Map类的addCube方法可以将新的Cube信息记录在一个二位数组中。之后检测每一行是否已经满了，所有满行检测完以后，将会消除已满的行的信息并且将上面相应的行信息平移下来。清除的行数可以反馈至PlayState，用于PlayState中玩家分数的更新。

b)Map类有一个成员maxh，用于记录下面方块堆的最大高度。这个参数超过一定值，PlayState就会进入GameoverState，表示游戏结束。

1. **成果展示**



图**1** 开始状态 图**2** 游戏状态



图**3** 游戏结束状态 图**4** 高分榜状态



图**5** 结束状态