武汉大学计算机学院

本科生课程设计报告

图形界面 2048 小游戏

专 业 名 称　：计算机类

课 程 名 称　：高级语言程序设计

团 队 名 称　：陈启明小组

指 导 教 师　：常军 讲师

团 队 成 员 一：陈启明（2018301040082）

团 队 成 员 二：陈航航（2018302141022）

团 队 成 员 二：胡雅芳（2018305232069）

团 队 成 员 二：吕欣　（2018302060114）

二〇一九年 6 月

摘要

实验目的是加深对 C++语言以及面向对象的程序设计方法的理解，提高实践编程能力。

实验设计主要遵循 UML 类图设计的原则以及自顶向下、逐步求精的设计方法

实验内容包括使用 Visual Studio 2017 开发工具，借助 Easy-X 第三方库，实现图形界面 2048 小游戏，并对其进行功能扩展。

实验结果：经过反复测试以及多次修改，实现了游戏的基本功能以及扩展功能；实现了图形化界面；在测试范围内，程序运行结果正常。

关键词：C++ 面向对象的程序设计方法 Easy-X 图形界面 小游戏

目录

1. 实验目的

为了将 C++理论知识付诸实践、培养程序设计思维、掌握程序设计方法、提高动手编程能力，我小组选择以实现图形界面 2048 小游戏作为实验内容，在实现基本功能的基础上，附加部分扩展功能，包括撤销、计分、存档、困难模式等功能。为了实验的顺利进行，我小组在开题报告前基本完成对功能的构想、小组成员分工工作以及总体思路的设计，并以 PPT 的形式做出了展示。在实验过程中，通过小组成员之间的配合，基于 C++语言顺利实现小游戏并完成了测试、修复等工作，实验结果达到预期。

2. 实验设计

2.1 概述

2048 是一款简单而流行的数字游戏，其游戏主界面是一个 4*4 的棋盘，基本规则如下：玩家每次可以通过键盘选择上下左右其中一个方向去滑动，每滑动一次，所有的数字方块都会往滑动的方向靠拢外，系统也会在空白的地方随机出现一个数字方块，相同数字的方块在靠拢、相撞时会相加。不断的叠加最终拼凑出 2048 这个数字则游戏成功。

为了游戏的可玩性以及实验过程的挑战性，我小组在游戏基本功能的基础上新增撤销、计分、存档、速度模式等功能，其具体内容以及模块的设计将在 2.2 中详细列出。

小组成员分工情况如下：

陈启明：设计并实现撤销功能以及速度模式的算法；对小组成员的基本游戏逻辑算法以及计分功能进行优化，并利用陈航航的图形界面逻辑框架对整个程序的各种功能进行串联、整合、优化，得到游戏的最终版本。

陈航航：负责设计图形界面，实现图形界面的逻辑框架，基本实现了游戏的图形化；设计了速度模式计时器的雏形；承担测试工作。

胡雅芳：以本地存储的方式实现了计分功能以及存档功能。

吕欣：基本模式游戏逻辑的实现。

2.2 功能与设计

2.2.1 需求与功能

实现 2048 游戏的基本功能以及扩展功能，其基本规则已在 2.1 中列出。其扩展功能包括

计分：每两个方块合并为一个方块后，就加上与新方块对应的分数值（比如两个 4 合成为 8，则分数+8），并将当前分数展示在界面上。

撤销：在普通模式中，可以进行至多两步的撤销操作（包括游戏状态的撤回与分数的撤回）。

存档：要求游戏关闭后能存储游戏的最高分，并在新游戏开始后将历史最高分展示于游戏界面上。

速度模式：进入该模式后，玩家可选择每两次操作之间限定的时间（比如 3 秒、5 秒、7 秒）。玩家每次操作的时间必须限定在该时间之内，否则游戏失败。

图形化界面：设计简洁而美观的图形界面，玩家可以通过鼠标点击直接在界面上操作，游戏主界面的棋盘也应实现图形化。

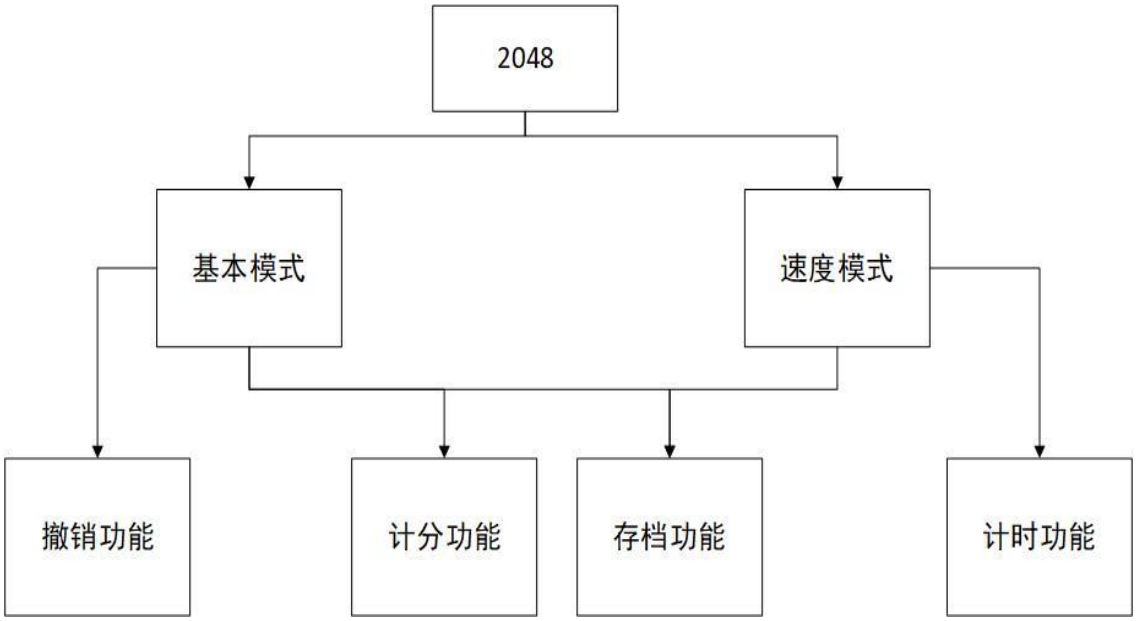


图 2.1 程序结构示意图

2.2.2 模块设计

本小组采用自顶向下、逐步求精的方法对功能进行设计、细化、实现，并借助 UML 类图展示思维过程。值得一提的是，在开题报告展示之前，我小组已对功能模块的设计有了初步的构想，并绘制了第一版 UML 类图（如图 2.2 所示）。但是在实际操作以及编写代码的过程中，发现部分成员函数和成员变量不是必须的，还有一部分是在最初设计过程中没有考虑到的，此外，为了程序的简化以及实现的容易，也从 UML 类图中剔除了 Account 类，所以最终的 UML 类图如图 2.3 所示。

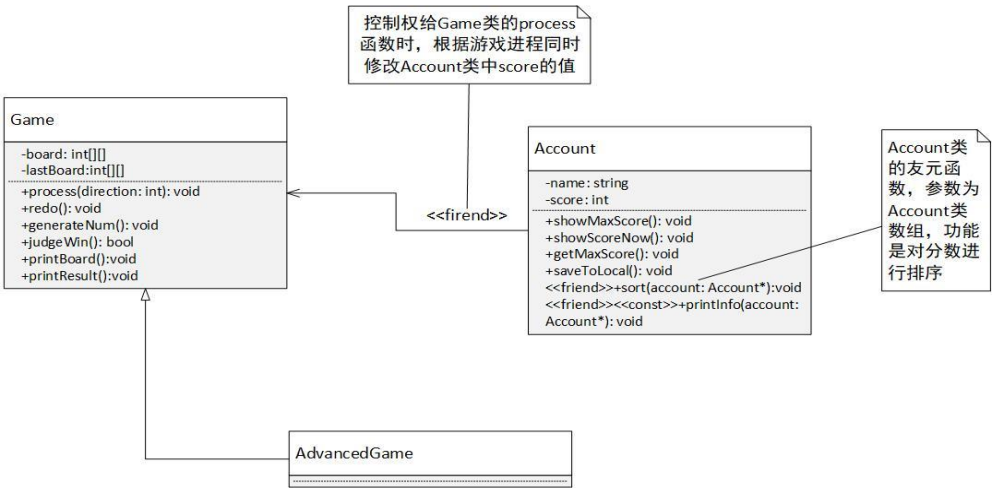
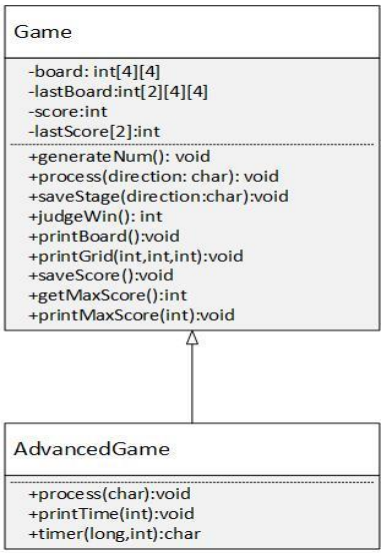


图 2.2 第一版类图



源代码方面共分为 3 个文件，Game.h、Game.cpp 和 main.cpp。Game.h 包含全局变量的定义以及类的定义，Game.cpp 包含类的实现，main.cpp 包含主函数以及一些与图形界面有关函数的声明和定义。

图 2.3 第二版类图

2.2.3 数据设计

在该版本 2048 小游戏中，数据主要以变量、数组、.dat 文件等形式存储。

普通变量是在程序实现过程中必需的，此处不再赘述。程序的全局变量在 Game.h 中声明并定义，主要包括 4*4 方格的行列值、游戏状态值（胜利、失败、继、续）、图形方格的大小、图形方格的间距、图形方格的颜色、图形界面中鼠标在某范围点击时函数返回的状态（该变量设计的主要目的是出于程序可读性方面考虑的）。具体代码如下：

```
1.  /*棋盘行列均为 4*/
2.  const int ROW = 4;
3.  const int COL = 4;
4.
5.
6.  /*输赢状态*/
7.  const int WIN = 1;
8.  const int CONTINUE = 2;
```

```
const int LOSE = 3;


const int GRID_SIZE = 100;     //格子大小
const int GRID_PADDING = 10;    //格子间距


/*鼠标响应状态*/
const int NORMAL_MODE = 0;
const int SPEED_MODE = 1;
const int RULE = 2;
const int EXIT = 3;


const int color[16] = {
    0x8a949e,0x2ec2ed,0xc8e0ed,0x79b1f2,
    0x6395f5,0x5f7cf6,0x3b5ef6,0x72cfed,
    0x61cced,0x50c8ed,0x3fc5ed,0x2ec2ed,
    0x50c8ed,0x2ec2ed,0xc8e0ed,0x79b1f2
};    //存储格子的颜色
```

基本的成员变量包括存储棋盘状态的二维数组和存储分数的 score 变量。除此以外，为了撤销功能的实现，还需要存储以往状态的二维数组以及存储以往分数的变量。代码如下：

```
protected:
    int board[4][4];
    int lastBoard[2][4][4]; //存储当前和上一次 4*4 方格中的状态（每次生成随机数后存储）
    int score;  //分数
```

```
5.      int lastScore[2];  //存储当前和上一次的分数
```

值得注意的是，此处将 lastBoard 设为三维数组，将 lastScore 设为一维数组，其目的是为了存放当前和前一次的两个状态。在没有撤回的情况下，每产生一次随机数后，需要将[1]的状态转移到[0]，同时把当前状态存放到[1]中，这样就能保证[0]中存储的状态永远都是上一次状态，这种方法类似于数据结构中的队列。如果需要考虑到撤回，情况就略微有些不同，因为撤回后状态的保存和上述过程是相反的，这时还需要在 saveStage 函数中定义静态变量存放“被挤出去的”状态，这一部分在 2.2.4 关键算法设计(4)中会提及。

至于.dat 文件，主要是为了存储历史最高分。在处理这部分数据时，需要用到 fopen，fclose，fprintf， fscanf 等与文件处理有关的函数。从文件中读取最高分的代码如下：

```
1.  int Game::getMaxScore()
2.  {
3.      int score = 0;
4.      FILE *fp;
5.      errno_t err;   //安全函数必须
6.      err = fopen_s(&fp, "scoreFile.dat", "r");
7.      if (err)   //如果文件没有被创建
8.      {
9.          err = fopen_s(&fp, "scoreFile.dat", "w");
10.         fprintf_s(fp, "%d", score, sizeof(int));   //就创建文件并将 0 写入文件
11.     }
12.     fscanf_s(fp, "%d", &score, sizeof(int));   //将文件内容读到内存
13.     fclose(fp);
14.
15.     return score;
16. }
```

游戏结束后，做出判断并保存历史最高分的代码如下：

```
void Game::saveScore()
{
    FILE *fp;
    errno_t err;
        //安全函数必须
    int bestScore;


    err = fopen_s(&fp,"scoreFile.dat","r");
    fscanf_s(fp, "%d", &bestScore, sizeof(int));   //取出文件中的分数存入 bestScore
    fclose(fp);

    if (score > bestScore)
    {
        err = fopen_s(&fp, "scoreFile.dat", "w");
        fprintf_s(fp, "%d", score, sizeof(int));   //将当前分数存入文件中
        fclose(fp);
    }
}
```

2.2.4 关键算法设计

(1) 产生随机数的算法

产生随机数的随机体现在两个方面：位置的随机以及方格内数字的随机。产生随机位置时，先要检查是否满格，如果没有满格，则生成两个 0 到 3 的随机数，表示行和列（需要对该位置进行判断，如果已经有数字了，则生成另外一对数字）。产生随机数字时，则按照一定的概率产生 2 或 4，具体实现方法是：产生 0 到 9 的随机数，若为 0，产生 4；否则，产生 2。流程图和代码如下：

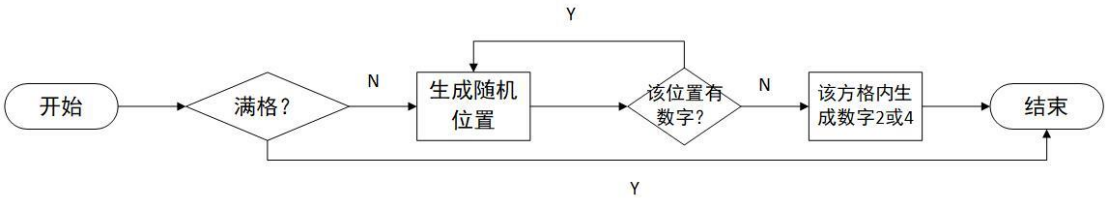


图 2.4 产生随机数的流程图

```
1.  void Game::generateNum()
2.  {
3.      bool isFull = true; //用于记录是否满格，起始值设为满格状态
4.      for (int i = 0; i < ROW; i++)
5.      {
6.          for (int j = 0; j < COL; j++)
7.          {
8.              if (board[i][j] == 0)
9.                  isFull = false;
10.         }
11.     }
12.
13.     if (isFull == false)    //如果没有满格，则生成随机数
```

```
14.     {
15.         srand((unsigned)time(NULL));    //时间作为种子值
16.         int row;
17.         int col;
18.         do {                    //生成随机的行数 row、列数 col(同时保证该位置没有数字）
19.             row = rand() % ROW;
20.             col = rand() % COL;
21.         } while (board[row][col] != 0);
22.
23.         int x = rand() % 10;
24.         if (x == 0)
25.             board[row][col] = 4;    //十分之一的概率生成 4
26.         else
27.             board[row][col] = 2;    //十分之九的概率生成 2
28.     }
29. }
```

(2) 游戏处理过程的算法（接收按键值）

该算法对棋盘状态和分数进行处理，主要分为两种情况。

如果接收的是方向键，做三步处理：数字向一边靠（一次靠边一格，最多操作三次），相同数字合并，数字再次靠边。

```
1.  switch (tolower(direction))
2.      {
3.      case'w':    //上
4.          for (int j = 0; j < COL; j++)    //按列遍历
5.          {
```

```
6.          for (int times = 0; times < 3; times++)     //最坏情况下，至少需要 3 次“靠边”操作
7.          {
8.            for (int i = 0; i < ROW - 1; i++)    //该列数字全部向上靠
9.            {
10.             if (board[i][j] == 0)
11.             {
12.               board[i][j] = board[i + 1][j];
13.               board[i + 1][j] = 0;    //如果 board[i][j]为 0，将其下方的数字赋给它，同时下方的数字变为 0
14.             }
15.           }
16.         }
17.
18.         for (int i = 0; i < ROW - 1; i++)       //相同数字合并
19.         {
20.           if (board[i][j] == board[i + 1][j])
21.           {
22.             board[i][j] *= 2;
23.             score+=board[i][j];     //加分
24.             board[i + 1][j] = 0;
25.           }
26.         }
27.
28.         for (int i = 0; i < ROW - 1; i++)    //该列数字全部向上靠
29.         {
30.           if (board[i][j] == 0)
31.           {
```

```
                board[i][j] = board[i + 1][j];
                board[i + 1][j] = 0;
            }
        }
    }
    break;
```

如果接收的是 x（撤销），那么就将当前状态复原为上一次状态，代码如下：

```
case'x':   //撤销
    /*对棋盘的操作*/
    for (int i = 0; i < ROW; i++)
    {
        for (int j = 0; j < COL; j++)
        {
            board[i][j] = lastBoard[0][i][j];
        }
    }

    /*对分数的操作*/
    score = lastScore[0];

    break;
```

(3)判断游戏是否结束的算法

遍历数组，如果存在 2048，游戏胜利；如果棋盘中存在空位或者存在两个相邻格子数字相同的情况，游戏继续；其它情况下游戏结束。

(4) 保存游戏状态的算法（为撤销操作服务）

该算法对每次操作后的棋盘状态和分数状态进行保存。主要分为两种情况：普通方向键操作后的保存。

每产生一次随机数后，需要将[1]的状态转移到[0]，同时把当前状态存放到[1]中，这样就能保证[0]中存储的状态永远都是上一次状态。除此以外，还需要将[0]中状态保存到一个静态变量中（全局生命期，下一次调用函数时静态变量中仍然保

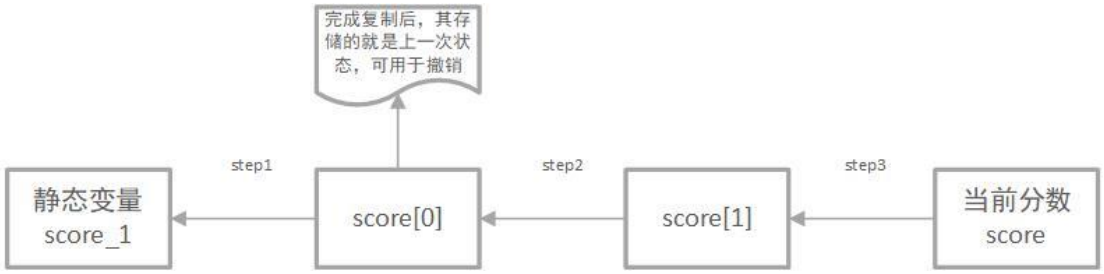


图 2.5 按下方向键后对数据操作的示意图

存有[0]中状态，防止[0]中状态丢失），这样做的目的是为下一种情况服务，因为撤销操作后的状态保存涉及前一次状态的还原。

```
1.  static int score_1;       //用于保存 score[0]，主要用于撤销
2.     static int board_1[4][4];  //用于保存 lastBoard[0]，主要用于撤销
3.
4.
5.     //基于普通方向键保存操作,lastBoard[0]存放上一次状态，[1]存放当前状态
6.     if (tolower(direction) == 'w' || tolower(direction) == 'a'
7.        || tolower(direction) == 's' || tolower(direction) == 'd')
8.     {
9.        /*对棋盘的操作*/
10.       for (int i = 0; i < ROW; i++)
11.       {
```

```
12.             for (int j = 0; j < COL; j++)
13.             {
14.                 board_1[i][j] = lastBoard[0][i][j];
15.                 lastBoard[0][i][j] = lastBoard[1][i][j];
16.                 lastBoard[1][i][j] = board[i][j];
17.
18.
19.             }
20.         }
21.
22.
23.         /*对分数的操作*/
24.         score_1 = lastScore[0];
25.         lastScore[0] = lastScore[1];
26.         lastScore[1] = score;
27.     }
```

撤销操作后的保存。

实质上与上一种情况方向相反，即实现对上一次数据状态的还原。

图 2.6 按下撤销键后对数据操作的示意图

```
//按下 x 为撤销，此为基于撤销的保存操作
    else if (tolower(direction) == 'x')
    {
        /*对棋盘的操作*/
        for (int i = 0; i < ROW; i++)
        {
            for (int j = 0; j < COL; j++)
            {
                lastBoard[1][i][j] = lastBoard[0][i][j];
                lastBoard[0][i][j] = board_1[i][j];
            }
        }


        /*对分数的操作*/
        lastScore[1] = lastScore[0];
        lastScore[0] = score_1;
    }
```

(5)速度模式计时方法

主要由成员函数 timer 实现，接收两个参数值，第一个是两次操作之间的时间限制（可以为 3 秒、5 秒、7 秒），第二个是计时开始的时间点，返回值是键盘按下的 WASD 方向值。主要实现原理是：使用 while 循环，循环条件为进入循环的时间小于 1s（可用当前时间 clock()和计时开始的时间之差实现），在循环体内，判断键盘上按下的值是否为 WASD，如果是，函数直接返回该方向值，此时显然倒计时会

停止，接着进行 process，generateNumber，printBoard 等操作。当程序再次运行到该函数时，倒计时又会重新开始显示。

于是倒计时功能得以实现——如果规定时间内不按下方向键，那么随着时间流逝，游戏失败；如果按下，程序对数据进行处理、显示后再次重新计时。代码如下：

```
1.  char AdvancedGame::timer(long startTime, int limit)
2.  {
3.      //limit 为限定时间，在设计的最初版本中，limit 只能选择 3s，为了使 Limit 可变，故将其作为函数的第二个形参，可根据玩家的选择而发生变化
4.      char direction;
5.
6.      for (int times = limit; times > 0; times--)
7.      {
8.          printTime(times);
9.          while (clock() - startTime < (limit + 1 - times) * CLOCKS_PER_SEC)
10.         {
11.             if (_kbhit())  //如果有 wasd 输入，立即停止计时
12.             {
13.                 direction = _getch();
14.                 if (tolower(direction) == 'w' || tolower(direction) == 'a'
15.                     || tolower(direction) == 's' || tolower(direction) == 'd')
16.                 {
17.                     return direction;
18.                 }
19.             }
20.
21.         }
22.     }
```

```
23.
24.     printTime(0);
25. }
```

(6) main 函数中基本游戏逻辑的实现

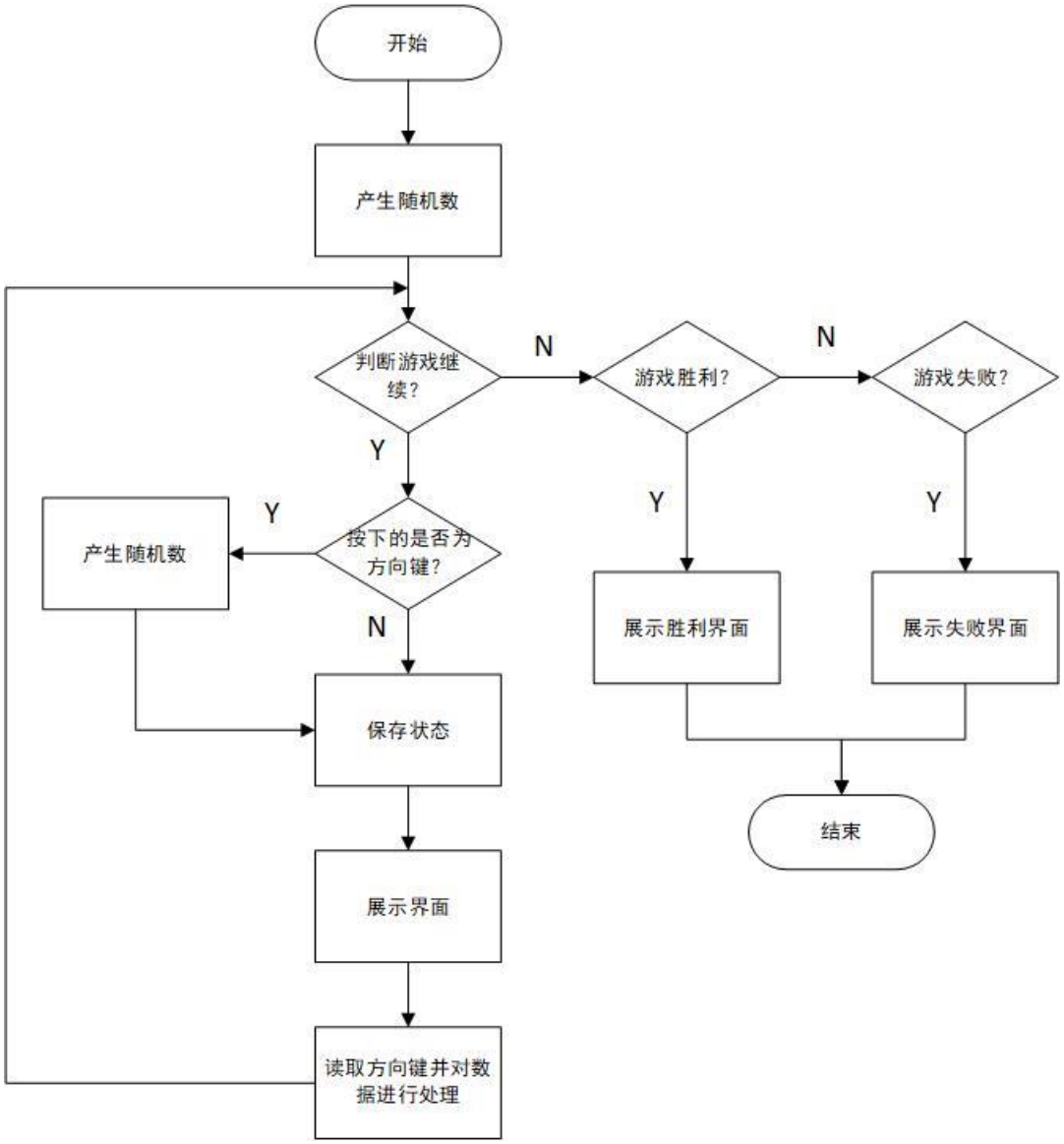


图 2.7 基本模式下游戏逻辑的流程图

2.2.5 运行结果

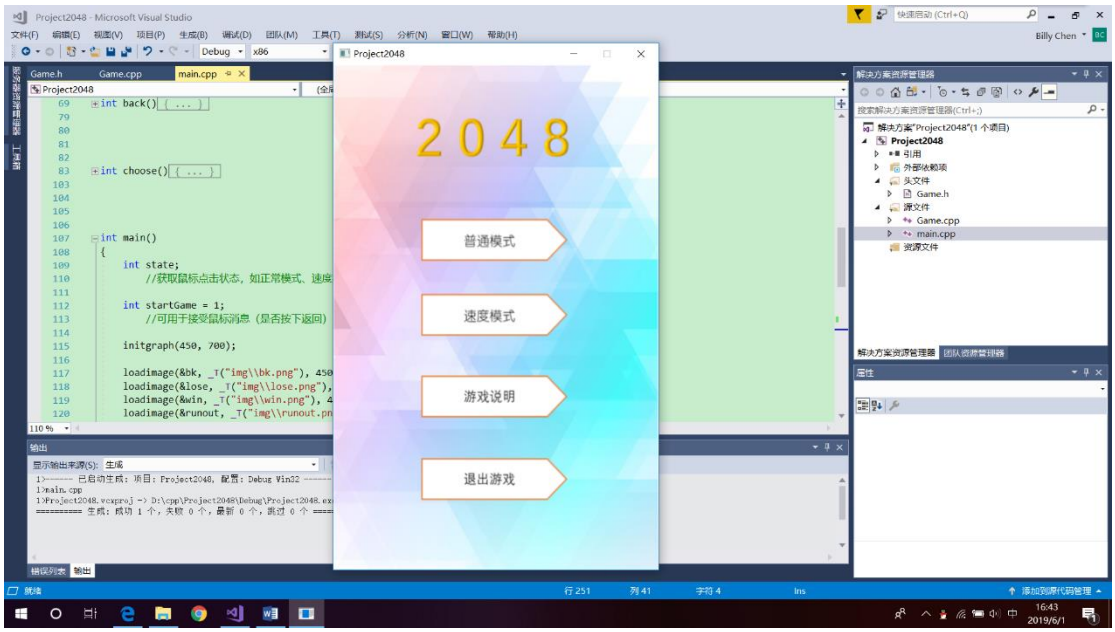


图 2.8 游戏主界面

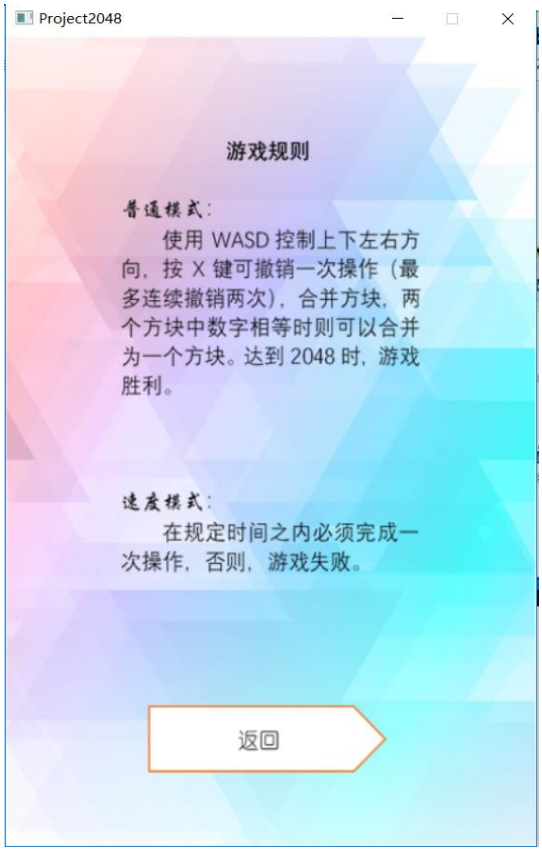


图 2.9 游戏说明界面

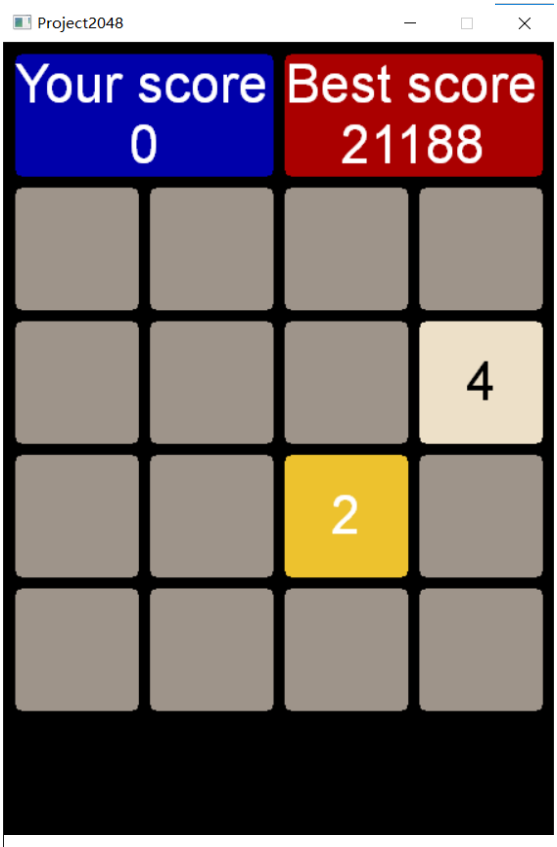


图 2.10 普通游戏开始时游戏界面

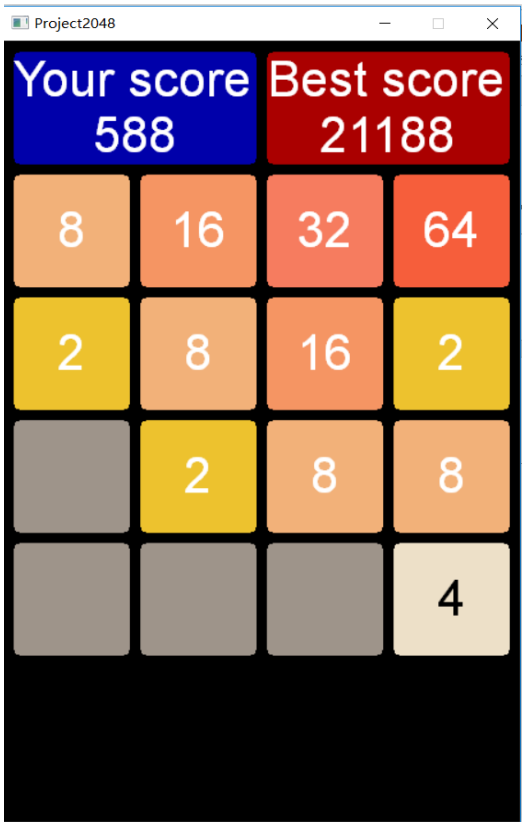


图 2.11 普通游戏进行时的界面

图 2.12 游戏失败的界面

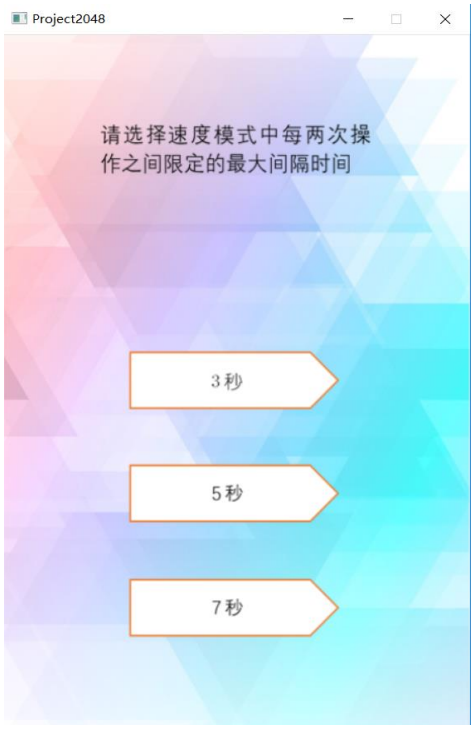


图 2.13 速度模式选择界面

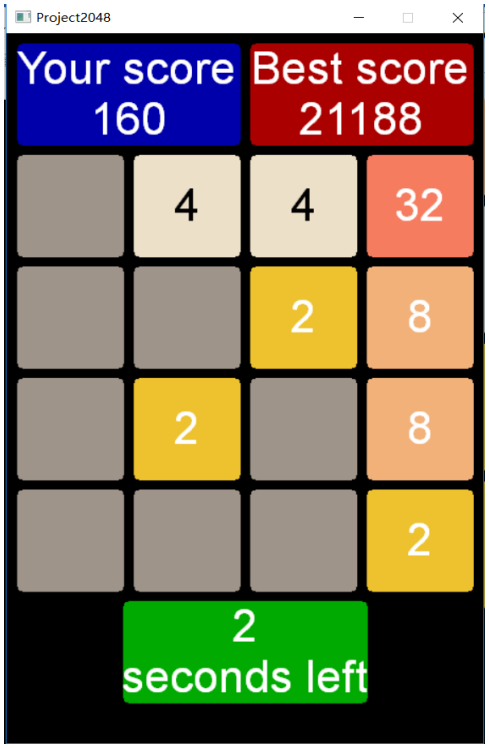


图 2.14 速度模式进行时的界面

图 2.15 时间耗尽的界面

图 2.16 游戏胜利的界面

Debug	2019/6/1 16:43	文件夹	
img	2019/5/25 14:37	文件夹	
Game	2019/5/25 9:57	C++ Source File	17 KB
Game	2019/5/31 20:56	C Header File	2 KB
main	2019/6/1 16:39	C++ Source File	6 KB
Project2048.vcxproj	2019/5/19 16:03	VC++ Project	6 KB
Project2048.vcxproj.filters	2019/5/19 16:03	VC++ Project Fil...	2 KB
Project2048.vcxproj.user	2019/5/20 22:52	Per-User Project...	1 KB
scoreFile.dat	2019/5/23 23:51	DAT 文件	1 KB

图 2.17 源文件、图片文件以及数据文件所在的文件。scoreFile.dat 存放历史最高分

3. 结论

本小组基本完成了实验任务，源文件的组织结构比较分明，程序也实现了既定的所有功能，在测试范围内，程序运行结果基本正常，达成了本次实验的基本目标。

但是在某些方面，程序仍有需要改进之处，首先，游戏界面是通过使用 Easy-X 库实现的，界面中的画面也是单独绘制的，不乏简陋之处，与实际的图形化编程有一定的差距。其次，用户在体验游戏时按键层级数过多，友好性方面略有不足。再次，游戏进行过程中，尽管画面以图形方式呈现，但由于缺少动画，直观性略差。

在代码质量方面，也需要有改进之处。尽管 main.cpp 文件中定义了一些函数，但 main 函数中内容过于繁杂，不利于阅读，而且存在几处代码重复的情况。在类的定义和实现上，类的数目过少，类的继承也显得有些突兀。

在数据结构方面，由于缺少相关知识，只能以数组的方式存储数据，数据存储效率不高，方式不完全合理，这在今后学习过程中是需要值得重视的。

总而言之，虽然仍存在些许问题，但程序能够正常运行，基本目标已经达成，编程能力得以提高。