软件设计实践报告

制作小游戏2048

单 位：计算机·网络空间安全学院

班 级：2018级计算机科学与技术3班

|  |  |  |
| --- | --- | --- |
| **组长** | （填学号） | （填姓名） |
| **组员1** | 201805551018 | 龙少华 |
| **组员2** | 201805551030 | 刘美君 |
| **组员3** | 201805551035 | 程煜 |

任课教师： 谢勇

湘 潭 大 学

2021年7月

1. 项目概述
2. 概述

本项目使用qt实现小游戏2048，2048是一款非常经典并且独具魅力的游戏，游戏的玩法是在一个n\*n的正方形网格上随机生成数字2，然后通过键盘上的方向键来控制数字的移动，例如按下左键所有的数字都会滑向左边，此时相邻的两个数字如果相同则会相加，并且消除掉远端的数字，靠近移动方向的数字翻倍，直到棋盘上有数字达到了2048，则游戏成功。在游戏进行中，右下角会根据玩家的操作显示分数。

当面板中所有位置都有数字，并且不论怎么移动都不能在合并数字来减少的话，游戏结束，提示玩家游戏结束，可以再次尝试。

1. 项目小组成员

小组成员：龙少华、刘美君、程煜

各自负责的工作：

龙少华：需求分析、进行项目设计、部分代码的编写、测试、撰写部分文档

刘美君：查询相关资料、进行项目设计、部分代码的编写、撰写部分文档

程煜：阅读相关技术文档，编写文档技术部分，负责视频录制

自评总结：

本次软件设计实践，我们初开始就选定做一个小游戏，想了许多中实现方法，包括DEV C++，Visual Studio等等。都不太好实现，最后我们发现了QT框架，发现它有比较好的跨平台行，且界面设计非常精妙，很满足我们的要求。

QT支持的是C++语言，我们需要C++源代码直接描述界面和功能，在整个过程中，技术实现最困难的部分在于，如何使用二维数组模拟棋盘的变化。通过指针控制每一个位置的棋子，执行判断条件并且执行相应操作。这部分需要精准地控制约束条件，考虑到每一种可能的情况，才能防止程序崩溃，这对于我们来说是一个巨大的挑战。

通过本次软件设计实践，我学到了很多，包括C++代码的编写能力，以及对QT框架的使用，最重要的是，我们学到了一个完整的项目制作的流程，从需求分析到制作到测试完工的整个过程。这对于我们以后的工作有很大帮助。这次项目，也锻炼了我们的合作能力，一个项目，特别是大项目，更需要大家的团结合作。最后，我么一定要多实践，才能真正地掌握一门技术。

自评：优

1. 项目工作进度

2021年6月20日 寻找项目大致方向与思路

2021年6月24日 确定项目具体实现目标和编码语言

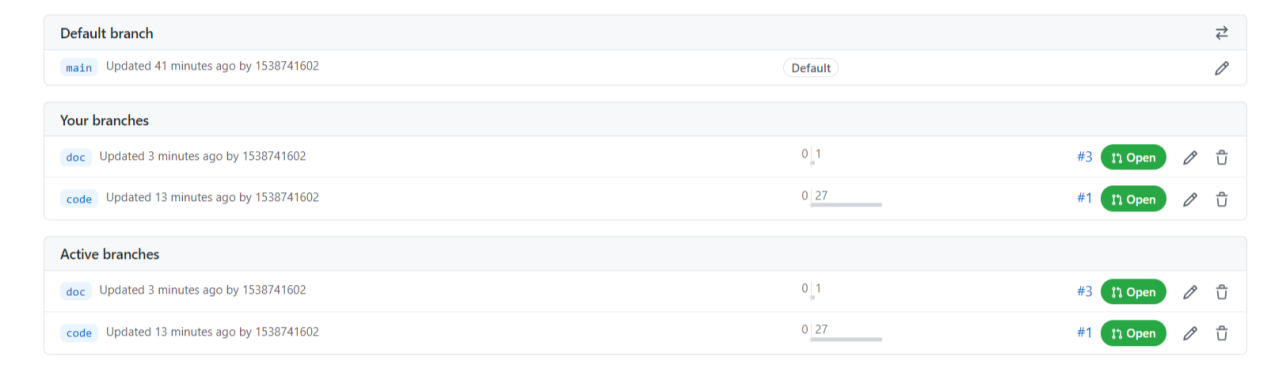
2021年6月28日 项目需求分析完成

2021年7月2日 项目代码部分撰写基本完成

2021年 7 月5日 项目代码部分优化完毕，运行正常

2021年 7 月9日 撰写文档完成

1. 项目代码托管地址

https://github.com/1538741602/2048.git

1. 软件演示视频连接

https://www.bilibili.com/video/BV1iM4y1M7gE

1. 需求分析
2. 用例模型

|  |  |
| --- | --- |
| **用例编号与名称** | UC1 处理开始游戏请求 |
| **范围** | 游戏控制 |
| **级别** | 子功能 |
| **主要参入者** | 用户 |
| **涉众及其关注点** | 用户希望运行可执行文件后即可进入游戏游玩 |
| **前置条件** | 无 |
| **成功保证** | 可执行文件和环境配置准确 |
| **主成功场景** | 用户点击可执行文件后项目开始运行 |
| **扩展** | 在可执行文件构建失败时提示报错 |
| **特殊需求** | 需要显示设备和鼠标 |
| **技术与数据变元素** | 无 |
| **发生频率** | 间隔一段时间发生，一般发生一次 |
| **杂项** | 暂无 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC2 随机生成数字块 |
| **范围** | 游戏功能 |
| **级别** | 子功能 |
| **主要参入者** | 系统 |
| **涉众及其关注点** | 系统希望在可执行文件运行后初始界面可以随机生成两个数字块2 |
| **前置条件** | 无 |
| **成功保证** | 程序中的函数freePosition()实现随机生成数字块 |
| **主成功场景** | 打开游戏后，界面随机出现两个数字块2 |
| **扩展** | 无 |
| **特殊需求** | 需要显示设备 |
| **技术与数据变元素** | 指令由游戏系统自动生成 |
| **发生频率** | 每次重新开启游戏进行一次 |
| **杂项** | 暂无 |
| **用例编号与名称** | UC3 数字块向左移动 |
| **范围** | 游戏操作 |
| **级别** | 子功能 |
| **主要参入者** | 用户 |
| **涉众及其关注点** | 用户希望在按动键盘上的左方向键后，数字块可以整体向左进行移动 |
| **前置条件** | 游戏处于开启状态且没有达到结束条件 |
| **成功保证** | Board.cpp文件中的move()操作函数 |
| **主成功场景** | keyPressEvent.cpp文件中的key\_left操作 |
| **扩展** | 如果游戏界面中数字块移动后界面不再发生改变则不再向左移动并提示退出游戏 |
| **特殊需求** | 键盘和显示设备 |
| **技术与数据变元素** | 指令由键盘输入← |
| **发生频率** | 在游戏期间不间断地发生 |
| **杂项** | 无 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC4 数字块向右移动 |
| **范围** | 游戏操作 |
| **级别** | 子功能 |
| **主要参入者** | 用户 |
| **涉众及其关注点** | 用户希望在按动键盘上的右方向键后，数字块可以整体向右进行移动 |
| **前置条件** | 游戏处于开启状态且没有达到结束条件 |
| **成功保证** | keyPressEvent.cpp文件中的key\_right操作 |
| **主成功场景** | 当用户按下→后，数字块向右进行移动 |
| **扩展** | 如果游戏界面中数字块移动后界面不再发生改变则不再向右移动并提示退出游戏 |
| **特殊需求** | 键盘和显示设备 |
| **技术与数据变元素** | 指令由键盘输入→ |
| **发生频率** | 在游戏期间不间断地发生 |
| **杂项** | 无 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC5 数字块向上移动 |
| **范围** | 游戏操作 |
| **级别** | 子功能 |
| **主要参入者** | 用户 |
| **涉众及其关注点** | 用户希望在按动键盘上的向上方向键后，数字块可以整体向上进行移动 |
| **前置条件** | 游戏处于开启状态且没有达到结束条件 |
| **成功保证** | keyPressEvent.cpp文件中的key\_up操作 |
| **主成功场景** | 当用户按下↑后，数字块向上进行移动 |
| **扩展** | 如果游戏界面中数字块移动后界面不再发生改变则不再向上移动并提示退出游戏 |
| **特殊需求** | 键盘和显示设备 |
| **技术与数据变元素** | 指令由键盘输入↑ |
| **发生频率** | 在游戏期间不间断地发生 |
| **杂项** | 无 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC6 数字块向下移动 |
| **范围** | 游戏操作 |
| **级别** | 子功能 |
| **主要参入者** | 用户 |
| **涉众及其关注点** | 用户希望在按动键盘上的下方向键后，数字块可以整体向下进行移动 |
| **前置条件** | 游戏处于开启状态且没有达到结束条件 |
| **成功保证** | keyPressEvent.cpp文件中的key\_down操作 |
| **主成功场景** | 当用户按下↓后，数字块向左进行移动 |
| **扩展** | 如果游戏界面中数字块移动后界面不再发生改变则不再向下移动并提示退出游戏 |
| **特殊需求** | 键盘和显示设备 |
| **技术与数据变元素** | 指令由键盘输入↓ |
| **发生频率** | 在游戏期间不间断地发生 |
| **杂项** | 无 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC7 游戏计算积分 |
| **范围** | 游戏功能 |
| **级别** | 子功能 |
| **主要参入者** | 系统 |
| **涉众及其关注点** | 系统希望在用户移动数字块后，自动计算出此时棋盘上的积分 |
| **前置条件** | 游戏处于开启状态且没有达到结束条件 |
| **成功保证** | Board.cpp中获取分数的函数运行正常 |
| **主成功场景** | 当用户移动数字块后游戏面板更新分数 |
| **扩展** | 如果游戏界面中数字块移动后界面不再发生改变则分数不再改变 |
| **特殊需求** | 显示设备 |
| **技术与数据变元素** | 指令由系统自动完成 |
| **发生频率** | 在游戏期间不间断地发生 |
| **杂项** | 用户在一次移动过程中没有出现分数变化进行提醒 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC7 游戏结束 |
| **范围** | 游戏功能 |
| **级别** | 子功能 |
| **主要参入者** | 系统 |
| **涉众及其关注点** | 系统希望在用户移动数字块后，判断此时棋盘还能否移动 |
| **前置条件** | 游戏处于开启状态 |
| **成功保证** | Board.cpp中changed函数判断棋盘是否和和前一次棋盘界面一致，一致则表示棋盘已经无法移动 |
| **主成功场景** | 当棋盘中数字块已经满了，用户再进行移动则提示报错，游戏结束 |
| **扩展** | 无 |
| **特殊需求** | 显示设备 |
| **技术与数据变元素** | 指令由系统自动完成 |
| **发生频率** | 在游戏结束时出现 |
| **杂项** | 用户在游戏结束后可添加一个bgm提示 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC8 游戏结束后点击try again |
| **范围** | 游戏操作 |
| **级别** | 子功能 |
| **主要参入者** | 用户 |
| **涉众及其关注点** | 用户在游戏失败后想要点击try again 再进行一次游戏 |
| **前置条件** | 游戏处于结束状态 |
| **成功保证** | QGameOverWindow.cpp中的QGameOverWindow（）函数 |
| **主成功场景** | 用户点击try again然后重启游戏 |
| **扩展** | 无 |
| **特殊需求** | 显示设备和鼠标 |
| **技术与数据变元素** | 指令由用户点击try again 按钮后触发 |
| **发生频率** | 在游戏结束时出现 |
| **杂项** | 无 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC9 棋盘样式 |
| **范围** | 游戏界面 |
| **级别** | 子功能 |
| **主要参入者** | 系统 |
| **涉众及其关注点** | 系统生成棋盘界面的配色和棋盘大小 |
| **前置条件** | 无 |
| **成功保证** | QGameBoard.cpp中的QGameBoard函数能够初始化棋盘 |
| **主成功场景** | 运行可执行文件后初始化棋盘样式 |
| **扩展** | 棋盘样式更为多样一些 |
| **特殊需求** | 显示设备 |
| **技术与数据变元素** | 指令由系统触发函数 |
| **发生频率** | 在游戏开始时发生 |
| **杂项** | 无 |

|  |  |
| --- | --- |
| **用例编号与名称** | UC10 数字块的样式 |
| **范围** | 游戏界面 |
| **级别** | 子功能 |
| **主要参入者** | 系统 |
| **涉众及其关注点** | 系统对不同的数字块配上不同的颜色用于区分 |
| **前置条件** | 无 |
| **成功保证** | Qtile中的draw函数用于处理棋子样式 |
| **主成功场景** | 运行可执行文件后初始化棋子样式 |
| **扩展** | 棋盘样式更为多样一些 |
| **特殊需求** | 显示设备 |
| **技术与数据变元素** | 指令由系统触发函数 |
| **发生频率** | 在游戏开始时或游戏进行时发生，频率较高 |
| **杂项** | 无 |

1. 补充性规格说明

可用性：

1. 采用图形化界面，游戏操作使用键盘上下左右，其他操作仅需轻点鼠标即可实现。
2. 采用不同颜色来区分不同的数字块，让用户更方便辨别，提高游戏体验度

可靠性：

1. 这个游戏是个极度轻量的游戏，因此其运行相对来说较为可靠，不会出现较大的bug。
2. 游戏对于用户的点击响应速度较快。
3. 词汇表

网格板：2048游戏界面由n\*n个数字块组成，整体外观像一个网络图，可具有不同的背景颜色

消除：相同的邻接数字块在靠拢过程中会去掉远端的数字，保留离移动方向边框远一点的数字并且大小为初始的两倍

生成：在移动过程中，程序会在移动方向的相反方向随机生成数字块2。

计算：移动到一起之后，非零数字之间没有空隙了，对相邻的两个数字相同的 进行相加计算。将计算的结果保存在离手指滑动方向较远的那个位置，那么，被消 除的那个位置就要重置数值为0。

1. 设想

本项目已经实现了2048的基本玩法，但是附加功能仍旧较少。我们计划在将来实现能够让用户自己选择棋盘大小，以及选择棋盘的背景颜色，以增加用户体验度。同时这个游戏应当还添加一个更多的玩法选择，不仅仅局限于数字，可扩展到朝代、军衔等等创意性的玩法。更进一步的，开发充值系统，当游戏失败时可以通过充值挽救局面，让游戏继续进行。

1. 业务规则
2. 游戏开始，页面随机出现数字块2，用户需上下左右滑动使每两个邻接的相同数字块融合成为一块，数字块的值变为两倍。
3. 玩家每一次操作，相邻的同种数字会融合，也会出现新的几个数字块2，直到所有格子被占满，游戏结束。

（2）玩家能通过方向键上下左右来控制数字块的上下左右移动。

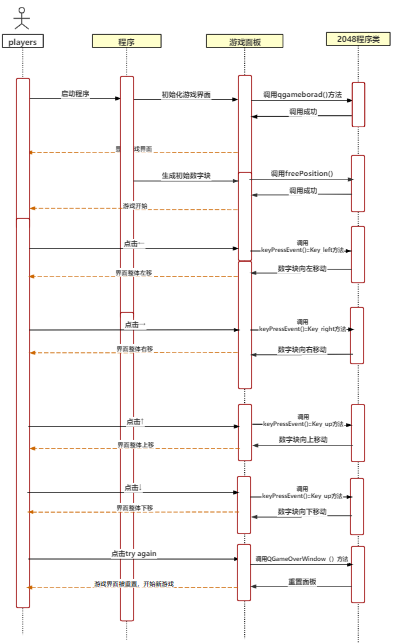
（3）每种大小的数字块有不同的颜色。

（4）游戏能够在玩的过程中，给出玩家的分数，分数根据数字块融合的数量和大小决定，每移动一次，就将分数累加到总分中。

（5）游戏有开始，游戏分数，再来等。

1. 系统设计

系统时序图



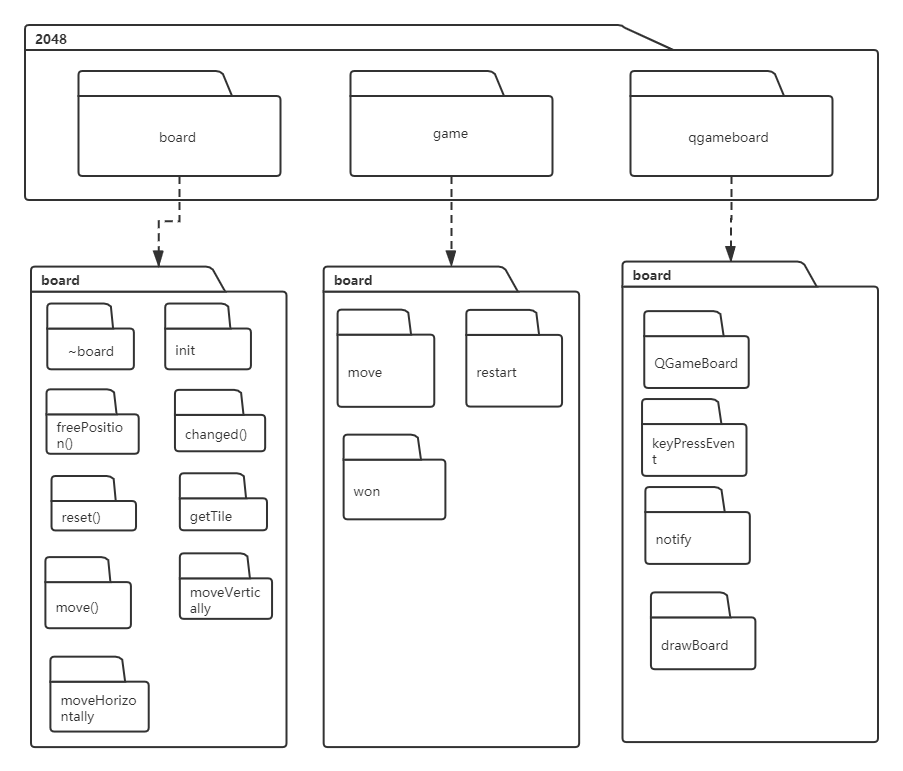
操作契约

|  |  |
| --- | --- |
| 操作 | resetGame() |
| 交叉引用 | UC8 |
| 前置条件 | Try again按钮可用 |
| 后置条件 | 重置游戏界面后，游戏界面重新生成数字块，开始新的一轮游戏 |

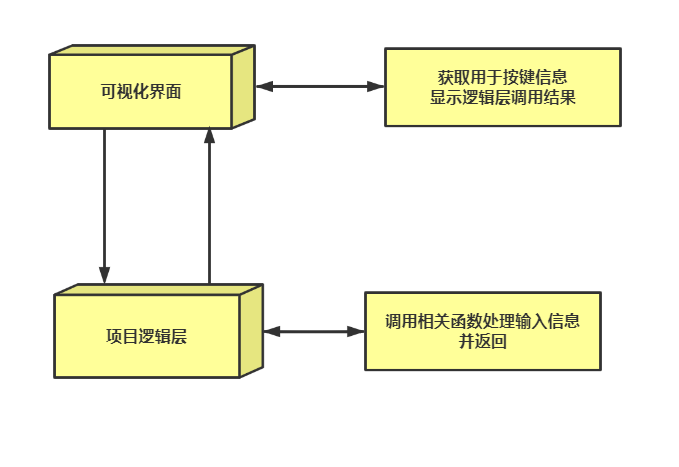
|  |  |
| --- | --- |
| 操作 | 再来一局 |
| 交叉引用 | 无 |
| 前置条件 | 数字块已经无法移动，游戏已经终止 |
| 后置条件 | 游戏界面被重置，游戏重新开始 |

|  |  |
| --- | --- |
| 操作 | 上下左右移动数据块 |
| 交叉引用 | 无 |
| 前置条件 | 游戏已经开始 |
| 后置条件 | 根据移动的指令，界面进行变换 |

包图

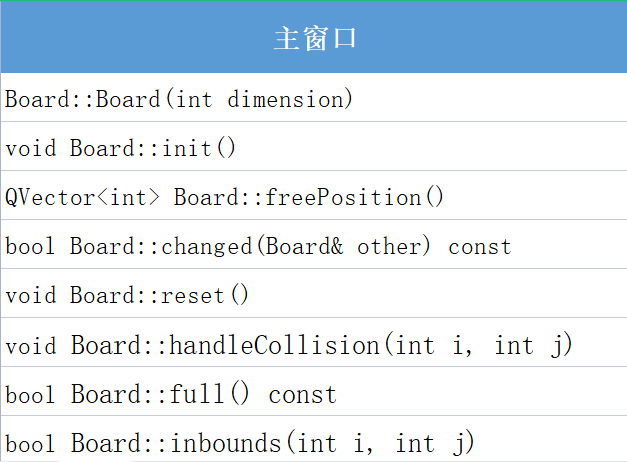


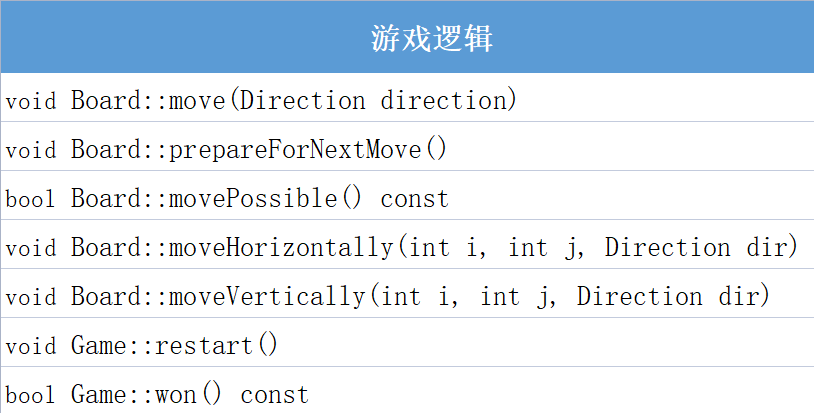
**逻辑分层**



1. 详细设计

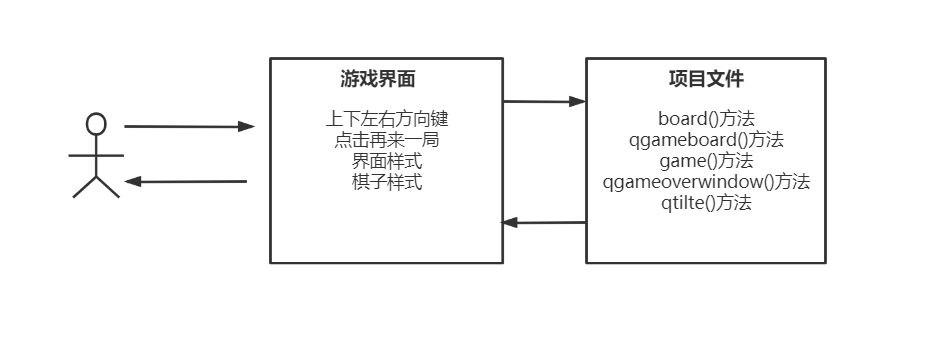
类图





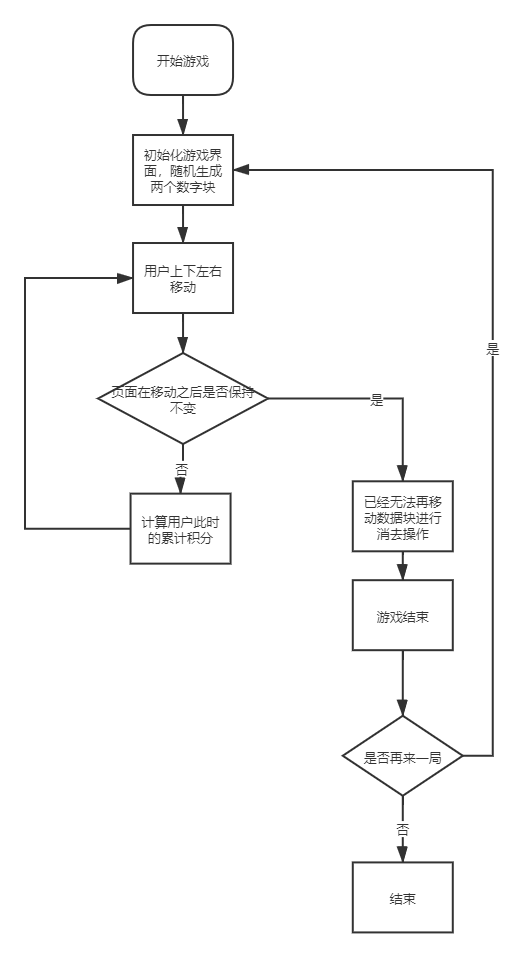


交互图



1. 代码实现

流程图



核心代码解析

1. 初始化面板

Board::**Board**(int dimension)

{

pointsScoredLastRound = 0;

tileCollisionLastRound = false;

this->dimension = dimension;

init();

}

1. 销毁棋子，当棋子合并的时候

Board::~**Board**()

{

// destroy the tiles

for (int i = 0 ; i < dimension; ++i)

for (int j = 0; j < dimension; ++j)

delete board[i][j];

}

1. 初始化游戏界面

void Board::**init**()

{

//用resize函数初始化面板，注意是二维数组的逐层初始化

board.resize(dimension);

for (int i = 0; i < dimension; ++i)

board[i].resize(dimension);

// for loop run through board create tiles

for (int i = 0; i < dimension; ++i)

for (int j = 0; j < dimension; ++j)

board[i][j] = NULL;

}

1. 初始化的时候产生一个随机位置生成数字块2

QVector<int> Board::**freePosition**()

{

QVector<int> pos;

if (full()) {

pos.append(-1);

pos.append(-1);

} else {

int i,j;

do {

//rand产生随机数模dimension就能得到0，1，2，3

i = rand() % dimension;

j = rand() % dimension;

} while (board[i][j] != NULL);

pos.append(i);

pos.append(j);

}

return pos;

}

1. 判断面板在移动前后是否发生改变

bool Board::**changed**(Board& other) const

{

if (dimension != other.dimension)

return false;

for (int i = 0; i < dimension; i++)

for (int j = 0; j < dimension; ++j)

if ( ( (board[i][j] == NULL && other.board[i][j] != NULL) ||

(board[i][j] != NULL && other.board[i][j] == NULL) ) ||

( (board[i][j] != NULL && other.board[i][j] != NULL) &&

board[i][j]->getValue() != other.board[i][j]->getValue()) )

return true;

return false;

}

1. 利用枚举数字存储移动方向

void Board::**move**(Direction direction)

{

Board pre\_move\_board(\*this);

prepareForNextMove();

switch (direction) {

case UP:

for (int i = 0; i < dimension; ++i)

for (int j = 0; j < dimension; ++j)

moveVertically(i,j,UP);

break;

case DOWN:

for (int i = dimension-1; i >= 0; --i)

for (int j = 0; j < dimension; ++j)

moveVertically(i,j,DOWN);

break;

case LEFT:

for (int i = 0; i < dimension; ++i)

for (int j = 0; j < dimension; ++j)

moveHorizontally(i,j,LEFT);

break;

case RIGHT:

for (int i = 0; i < dimension; ++i)

for (int j = dimension-1; j >= 0; --j)

moveHorizontally(i,j, RIGHT);

}

// 如果面板已经改变了，添加一个新的2棋子，也可以选择如果没有棋子融合再添加新棋子

if (changed(*pre\_move\_board*) /\*&& !tileCollisionLastRound\*/) {

QVector<int> newpos = freePosition();

board[newpos[0]][newpos[1]] = new Tile();

}

notifyObservers();

}

1. 实现上下左右移动操作的函数

void Board::**moveHorizontally**(int i, int j, Direction dir)

{

if (board[i][j] != NULL) {

bool tileCollision = false;

int newj;

if (dir == RIGHT)

newj = j + 1;

// to the left to the left

else

newj = j - 1;

// 一直向一个方向走，直到遇到不为空的棋子，或者到边界

while (inbounds(i,newj) && board[i][newj] == NULL) {

if (dir == RIGHT)

newj++;

else

newj--;

}

// out of bounds or ...

if (!inbounds(i,newj)) {

if (dir == RIGHT)

board[i][dimension-1] = board[i][j];

else

board[i][0] = board[i][j];

}

// ... collision

else {

// 遇到相同数值的相加

if (board[i][newj]->getValue() == board[i][j]->getValue() &&

!board[i][newj]->getUpgratedThisMove()) {

tileCollision = true;

handleCollision(i, newj);

}

// 遇到不同数值的则相邻排列

else {

if (dir == RIGHT)

board[i][newj-1] = board[i][j];

else

board[i][newj+1] = board[i][j];

}

}

// 移动后要把原来的棋子清除掉

if ( (dir == RIGHT && newj-1 != j) || (dir == LEFT && newj+1 != j) || tileCollision )

board[i][j] = NULL;

if (tileCollision)

tileCollisionLastRound = true;

}

}

// 垂直方向上的移动和水平的是类似的

void Board::**moveVertically**(int i, int j, Direction dir)

{

if (board[i][j] != NULL) {

bool tileCollision = false;

int newi;

if (dir == UP)

newi = i - 1;

// down

else

newi = i + 1;

while (inbounds(newi,j) && board[newi][j] == NULL) {

if (dir == UP)

newi--;

else

newi++;

}

if (!inbounds(newi,j)) {

if (dir == UP)

board[0][j] = board[i][j];

else

board[dimension-1][j] = board[i][j];

}

else {

if (board[newi][j]->getValue() == board[i][j]->getValue() &&

!board[newi][j]->getUpgratedThisMove()) {

tileCollision = true;

handleCollision(newi, j);

}

else {

if (dir == UP)

board[newi+1][j] = board[i][j];

else

board[newi-1][j] = board[i][j];

}

}

if ( (dir == UP && newi+1 != i) || (dir == DOWN && newi-1 != i) || tileCollision )

board[i][j] = NULL;

if (tileCollision)

tileCollisionLastRound = true;

}

}

1. 在按动方向键的时候将按键转化为代码中枚举数组中的元素

void QGameBoard::***keyPressEvent***(QKeyEvent \*event)

{

switch (event->key()) {

case Qt::Key\_Up:

game->move(UP);

break;

case Qt::Key\_Left:

game->move(LEFT);

break;

case Qt::Key\_Right:

game->move(RIGHT);

break;

case Qt::Key\_Down:

game->move(DOWN);

break;

}

}

1. 绘制游戏面板

void QGameBoard::**drawBoard**()

{

delete boardLayout;

boardLayout = new QGridLayout();

for (int i = 0; i < game->getGameBoard()->getDimension(); ++i) {

for (int j = 0; j < game->getGameBoard()->getDimension(); ++j) {

delete gui\_board[i][j];

gui\_board[i][j] = new QTile(game->getGameBoard()->getTile(i,j));

boardLayout->addWidget(gui\_board[i][j], i, j);

gui\_board[i][j]->draw();

}

}

mainLayout->insertLayout(0, boardLayout);

}

1. 再来一次的弹窗界面

QGameOverWindow::**QGameOverWindow**(QWidget \*parent) :

QWidget(parent)

{

setStyleSheet("QGameOverWindow { background: rgb(84, 199, 160); }");

setFixedSize(425,205);

QVBoxLayout \*layout = new QVBoxLayout(this);

// game over label

QLabel\* gameover = new QLabel("Game Over!", this);

gameover->setStyleSheet("QLabel { color: rgb(51, 122, 99); font: 40pt; font: bold;} ");

// reset button

reset = new QResetButton(this);

reset->setFixedHeight(50);

reset->setFixedWidth(100);

// add game over label to window

layout->insertWidget(0,gameover,0,Qt::AlignCenter);

// add reset button to window

layout->insertWidget(1,reset,0,Qt::AlignCenter);

}

1. 为数字块附上样式

void QTile::**draw**()

{

if (tile == NULL) {

setText("");

setStyleSheet("QTile { background: rgb(66, 199, 139); border-radius: 10px; }");

} else {

setText(QString::number(tile->getValue()));

switch (tile->getValue()) {

case 2: {

setStyleSheet("QTile { background: rgb(238,228,218); color: rgb(119,110,101); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 4: {

setStyleSheet("QTile { background: rgb(237,224,200); color: rgb(119,110,101); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 8: {

setStyleSheet("QTile { background: rgb(242,177,121); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 16: {

setStyleSheet("QTile { background: rgb(245,150,100); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 32: {

setStyleSheet("QTile { background: rgb(245,125,95); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 64: {

setStyleSheet("QTile { background: rgb(245,95,60); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 128: {

setStyleSheet("QTile { background: rgb(237,207,114); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 256: {

QGraphicsDropShadowEffect \*dse = new QGraphicsDropShadowEffect();

dse->setColor(Qt::yellow);

dse->setBlurRadius(20);

dse->setOffset(-1);

setGraphicsEffect(dse);

setStyleSheet("QTile { background: rgb(237,204,97); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 512: {

QGraphicsDropShadowEffect \*dse = new QGraphicsDropShadowEffect();

dse->setColor(Qt::yellow);

dse->setBlurRadius(30);

dse->setOffset(-1);

setGraphicsEffect(dse);

setStyleSheet("QTile { background: rgb(237,204,97); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 1024: {

QGraphicsDropShadowEffect \*dse = new QGraphicsDropShadowEffect();

dse->setColor(Qt::yellow);

dse->setBlurRadius(40);

dse->setOffset(-1);

setGraphicsEffect(dse);

setStyleSheet("QTile { background: rgb(237,204,97); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

case 2048: {

QGraphicsDropShadowEffect \*dse = new QGraphicsDropShadowEffect();

dse->setColor(Qt::yellow);

dse->setBlurRadius(50);

dse->setOffset(-1);

setGraphicsEffect(dse);

setStyleSheet("QTile { background: rgb(237,204,97); color: rgb(255,255,255); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

default: {

setStyleSheet("QTile { background: rgb(238,228,218); color: rgb(119,110,101); font: bold; border-radius: 10px; font: 40pt; }");

break;

}

}

}

}

1. 两个相同的数字块碰撞后相加

// 处理两个数遇到后相加的情况

void Board::**handleCollision**(int i, int j)

{

board[i][j]->upgrade();

board[i][j]->setUpgratedThisMove(true);

pointsScoredLastRound += board[i][j]->getValue();

}

1. 测试

1、测试环境

设备：PC机

操作系统：windows 10

处理器：Intel(R)Core(TM) i7-8750H CPU @2.20GHz 2.21 GHz

内存：8.00GB

系统类型：64位操作系统，基于x64的处理器

测试软件：5.9.8Qt

1. 测试用例

（1）游戏结束后的再来一次模块测试

|  |  |
| --- | --- |
| 输入等价类 | 鼠标点击 |
| 无效等价类 | 对非按钮部分的点击 |
| 有效等价类 | 对界面中按钮的点击 |
| 测试结果 | 游戏结束后点击再来一次即可重新开局 |

（2）游戏控制模块测试

|  |  |
| --- | --- |
| 输入等价类 | 键盘输入 |
| 无效等价类 | 除方向键外的其它按键 |
| 有效等价类 | 键盘上的方向键 |
| 测试结果 | 按动方向键后界面可正确移动 |

（3）游戏积分计算

|  |  |
| --- | --- |
| 输入等价类 | 无 |
| 无效等价类 | 无 |
| 有效等价类 | 无 |
| 测试结果 | 消去重复数字块后，可正确计算积分 |

3、bug修改记录

（1）qtcreator运行项目代码后报错

检查后发现是因为路径上带有汉字，路径必须保持纯英文

（2）在点击try again后游戏界面没有重置为初始状态

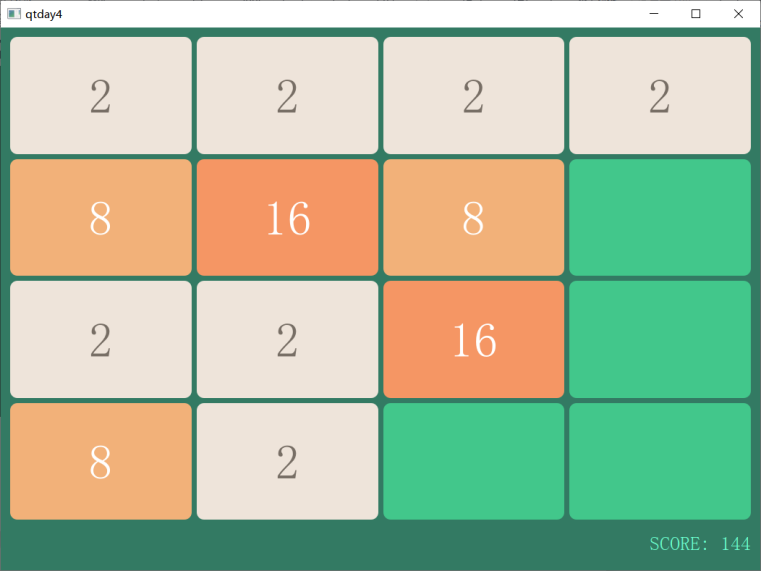
模型中按钮函数调用错误，导致鼠标点击后没有反应

1. 软件运行效果与使用说明

（1）项目运行初始状态：



（2）游戏进行期间：



（3）游戏失败时提示game over 并再来一局



使用说明：

1. 按动键盘上的方向键实现界面上下左右移动，消除相同的数据块
2. 在游戏失败时可以点击try again再次进行游戏
3. 界面右下角的积分用于统计当前步骤已经获得了多少分数
4. 设计体会

在开始进行这们课程的时候，之前主要接触的代码书写大多是一些oj题目，对于系统的项目还没有啥经验，web管理系统或者app的制作都不大具备必要的实践经历。这次课程没有具体的项目范围，小组对于c++语言更为熟悉一些，再加上要考虑做一个可视化的项目后，我们便采用qtcreator进行项目编写。

选取具体项目时，目光注意到了一些经典的小游戏复现，而且qt的练习时大部分是一些游戏方面的练习，游戏的实现会让人更有一些兴奋感。

在整个2048的实现过程中，卡壳的地方主要是对整个数据的上下左右的操作的实现。总是想着一下子将整个操作步骤写完，认为自己想的实现方法在逻辑上没有问题，那么实现之后也就不会出问题。然而总会发现有着自己没有考虑到的地方，并且程导致序运行后出+现的错误不止如何改正。后来学乖了，老老实实的将整个步骤一步步拆开，一步步的实现。

在写将二维数组的整个数据向某个方向移动时（上述类 myGame中写的游戏操作），虽然移动的方向不同，但整体逻辑是一样的，并且在每一的方向上的移动，都可以将二维数组拆分为MODE个一维数组，而对于每个一维数组的操作都是相同的，可以先实现。

本次项目并没有做出来一个真正意义上的app来，对于2048这个小游戏的改进之处还有很多地方，例如：游戏的积分排名、奖励，游戏的难度设置，游戏的游玩模式等方面均有改进之处，同时也可以把这款游戏部署到服务器上让用户可以实现跨域游玩。

1. 参考文献
2. qt官方参考文档 <https://doc.qt.io/>
3. qt项目练习 <https://gitee.com/kriko/qt_game_project_exercise>
4. Qt核心技术 <https://blog.csdn.net/bjtuwayne/article/details/97422833>
5. 《构建之法》
6. 附录
   1. 代码上传地址和记录

https://github.com/1538741602/2048.git

