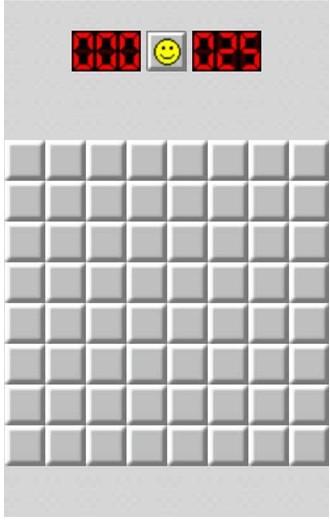
# 扫雷概要设计书

小组成员：曹炳阳、孙国瑞、葛炯佑

1. 扫雷问题描述

扫雷游戏的玩法是在一个9\*9 (初级)，16\*16 (中级)，16\*30 (高级)，或自定义大小的矩阵中随机布置一定量的地雷(初级为 10 个，中级为 40 个，高级为 99 个) ，但是雷区大小不能超过 24 *×* 30。由玩家逐个翻开格子，以找出所有地雷为最终游戏目标。 如果玩家翻开的格子有地雷，则游戏失败。如果玩家翻开所有没有地雷的格子，则游戏胜利。

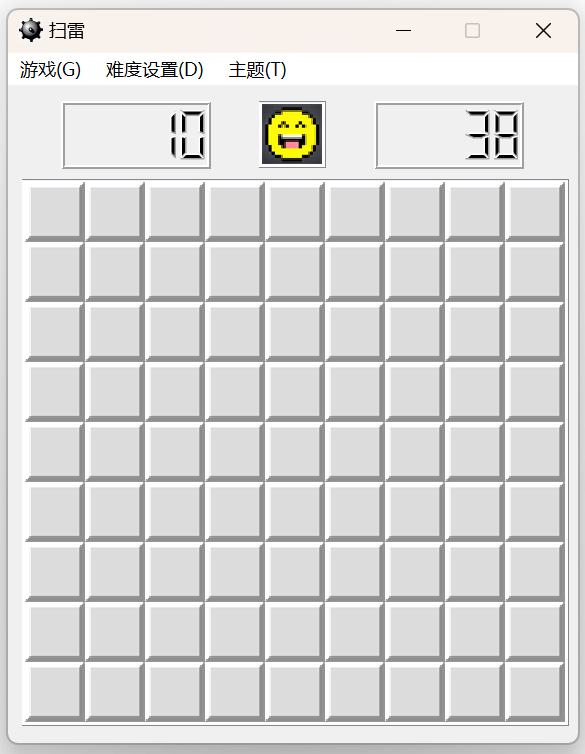


现在，我们需要通过编程来实现游戏的运行以及玩家的操作。

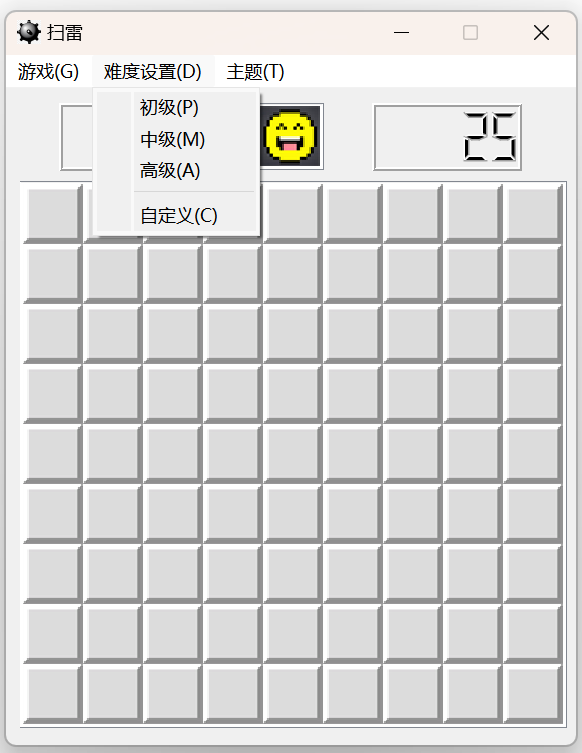
1. 玩家操作分析

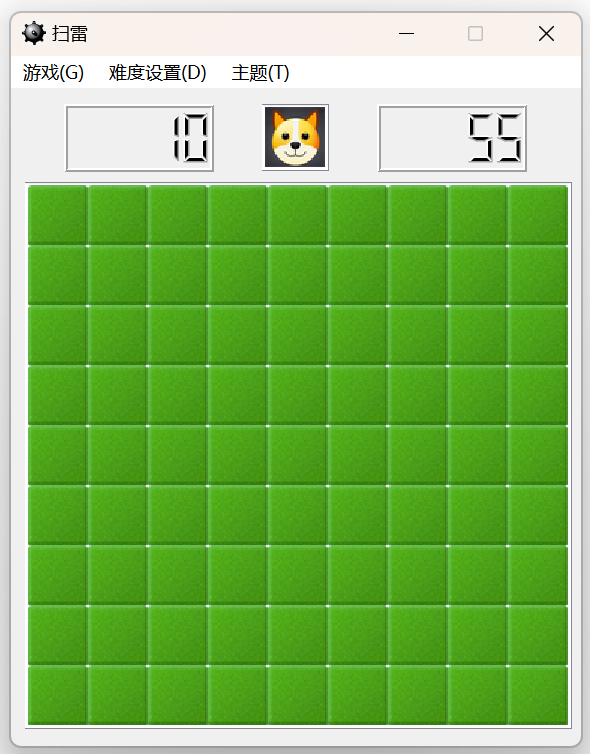
玩家的操作分为：单击打开格子、用旗子标记地雷位置、用问号标记不确定的格子、双击打开某个格子周围8个格子（当旗子的数量和格子显示的数字相等的时候）、取消标记等操作。

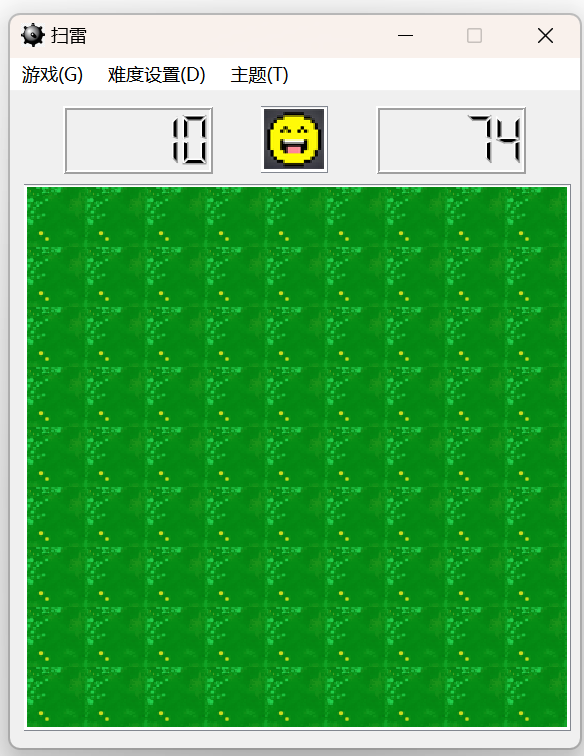
1. 用户界面设计

游戏主窗口：

图形用户界面, 应用程序

描述已自动生成难度选择：

小狗主题：

植物大战僵尸主题：

我们设计了雷数统计以及时间统计还有难度设置和更换主题等小工具，增加游戏的趣味性和可玩性并帮助玩家判断地雷。

1. 系统模块划分

此游戏程序部分划分为dailogdifficulty.cpp、emojiitem.cpp、emojiscene.cpp、latticeitem.cpp、main.cpp、mainwindow.cpp、playscene.cpp等七个模块。各模块功能如下：

1. 模块名称：dailogdifficulty.cpp

模块功能简要描述：调整游戏难度以及雷数的控制。

1. 模块名称：emojiitem.cpp

模块功能简要描述：控制游戏主界面的笑脸和哭脸（游戏进行中和游戏胜利显示笑脸，游戏失败时显示哭脸）。

1. 模块名称：emojiscene.cpp

模块功能简要描述：将新表情放进表情框。

1. 模块名称：latticeitem.cpp

模块功能简要描述：对每个格子的属性的控制。

1. 模块名称：main.cpp

模块功能简要描述：主函数，程序的入口。

1. 模块名称：mainwindow.cpp

模块功能简要描述：游戏的主窗口、信号与槽的链接。

1. 模块名称：playscene.cpp

模块功能简要描述：游戏版的场景控制。

1. 全局常量定义：enum enum\_latticeStatus

{

CLOSED,

OPENED,

FLAG,

QUESTION,

MINE\_GAMEOVER

};

// 指示主题

enum enum\_theme

{

CLASSIC,

DOG,

PVZ

};

// 指示emoji的状态（笑脸或哭脸）

enum enum\_emojiStatus

{

SMILE,

CRY

};

// 指示游戏结束状态（胜利或失败）

enum

{

LOSE,

WIN

};

const int SIDE\_LENTH = 40; // 一个格子的边长/像素

6.类及其成员设计

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **类** | | | | **类说明** | | | | | **私有成员** | | **说明** | | | | | | |
| DialogDifficulty | | | | 用于设计自定义难度对话框的类 | | | | | Ui::DialogDifficulty \*ui; | | 指向DialogDifficulty类对象的指针 | | | | | | |
| **公有成员** | | **说明** | | | | | | |
| explicit DialogDifficulty(QWidget \*parent = nullptr); | | DialogDifficulty类的构造函数 | | | | | | |
| ~DialogDifficulty(); | | DialogDifficulty类的析构函数 | | | | | | |
| int rowCount(); | | 返回自定义难度对话框中spinBoxRow的值 | | | | | | |
| int columnCount(); | | 返回自定义难度对话框中spinBoxColumn的值 | | | | | | |
| int mineCount(); | | 返回自定义难度对话框中spinBoxMine的值 | | | | | | |
| **类** | | | | | **类说明** | | | | | **私有成员** | | | | | **说明** | | |
| EmojiItem | | | | | 用于更新emoji的主题和状态的类 | | | | | int m\_theme;  int m\_status; | | | | | 接受emoji的主题和状态 | | |
| **公有成员** | | | | | **说明** | | |
| EmojiItem(int theme = CLASSIC, QObject \*parent = nullptr); | | | | | EmojiItem类的构造函数 | | |
| int theme() { return m\_theme; } | | | | | 返回主题 | | |
| int status() { return m\_status; } | | | | | 返回状态 | | |
| void setTheme(int theme) { m\_theme = theme; } | | | | | 设置主题 | | |
| void setStatus(int status) { m\_status = status; } | | | | | 设置状态 | | |
|  | | | | | | | | | | void updateEmoji(); | | | | | 更新emoji的贴图 | | |
| **类** | | | **类说明** | | | | | **私有成员** | | | | | | **说明** | | | |
| EmojiScene | | | 将新表情放进表情框的类 | | | | | EmojiItem \*emoji; | | | | | | 指向EmojiItem的指针 | | | |
| **公有成员** | | | | | | **说明** | | | |
| explicit EmojiScene(QObject \*parent = nullptr); | | | | | | EmojiScene类的构造函数 | | | |
| void setEmojiTheme(int theme) | | | | | | 设置Emoji主题 | | | |
| void setEmojiStatus(int status) | | | | | | 设置Emoji状态 | | | |
| **类** | | **类说明** | | | | **私有成员** | | | | | | **说明** | | | | |
| LatticeItem | | 对每个格子的属性的控制的类 | | | | int m\_row;  int m\_column;  bool m\_isMine;  int m\_mineAround;  int m\_status;  int m\_theme; | | | | | | LatticeItem的属性 | | | | |
| **公有成员** | | | | | | **说明** | | | | |
| explicit LatticeItem(int theme = CLASSIC, QObject \*parent = nullptr); | | | | | | LatticeItem的构造函数 | | | | |
| ~LatticeItem(); | | | | | | LatticeItem的析构函数 | | | | |
| bool isMine(); | | | | | | 返回isMine的值 | | | | |
|  | | | | | | int mineAround(); | | | | | | 返回 m\_mineAround的值 | | | | |
| int status(); | | | | | | 返回m\_status的值 | | | | |
| int theme(); | | | | | | 返回 m\_theme的值 | | | | |
| **类** | **类说明** | | | | | | **私有成员** | | | | | | **说明** | | |
| MainWindow | 控制游戏的主窗口、信号与槽的链接的类 | | | | | | Ui::MainWindow \*ui;  PlayScene \*scene;  EmojiScene \*emoScene;  QTimer \*fTimer;  int timer\_time; | | | | | | MainWindow的属性 | | |
| **公有成员** | | | | | | **说明** | | |
| MainWindow(QWidget \*parent = nullptr); | | | | | | MainWindow的构造函数 | | |
| ~MainWindow(); | | | | | | MainWindow的析构函数 | | |
| void on\_gameOver(int result); | | | | | | 游戏结束后的操作 | | |
|  | | | | | | | void on\_flagNumChange(); | | | | | | 旗子数的改变 | | |
| void on\_changeRemainMineLCD(int remain\_mine); | | | | | | 改变显示的剩余雷数 | | |
| void on\_timerStart(); | | | | | | 计时开始 | | |
| void on\_timerStop(); | | | | | | 计时停止 | | |
| void on\_timer\_timeout(); | | | | | | 在timer发出timeout的信号后将计时器的值加一 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **类** | **类说明** | **私有成员** | **说明** |
| PlayScene | 控制游戏版场景的类 | int m\_row;  int m\_column; int m\_mineNum;  int m\_remainMine;  int m\_steps;  int m\_theme;  int open\_times;  LatticeItem \*board[MAX\_ROW][MAX\_COLUMN]; | PlayScene的属性 |
| **公有成员** | **说明** |
| explicit PlayScene(QObject \*parent = nullptr); | PlayScene的构造函数 |
| void initialize(); | 初始化游戏板 |
| void setMap(int row, int column, int mine); | 布置游戏面板 |
|  | | int remainMine(); | 返回剩余雷数 |
| int steps() { return m\_steps; }  int row() { return m\_row; }  int column() { return m\_column; }  int mineNum() { return m\_mineNum; }  int theme() { return m\_theme; }  void setTheme(int theme) { m\_theme = theme; } | 返回已进行的操作数 |
| void r\_leftClick(int x, int y); | 左键单击 |
| void r\_rightClick(int x, int y); | 右键单击 |
| void r\_doubleClick(int x, int y); | 左键双击 |
|  | | void openLattice(int x, int y); | 打开格子 |
| void setRandMine(int x, int y); | 随机布雷 |
| void setMineAround(); | 设置格子周围雷的数量 |
| void on\_gameStatusChange(); | 当游戏状态发生改变时做出的响应 |
| on\_flagNumChange(); | 当旗子数量发生改变时做出的响应 |
|  | | showAllMine(int result); | 展示所有的雷 |
| void changeLatticeTheme(); | 更改格子的主题 |

**7.函数说明：**

**Playscene**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序  号 | 函数原型 | 功能 | 参数 | 返回值 |
| **1** | **void r\_leftClick(int x, int y);** | **对左键单击做出回应** | **int x int y** | **void** |
| **2** | **void r\_rightClick(int x, int y);** | **对右键单击做出回应** | **int x int y** | **void** |
| **3** | **void r\_doubleClick(int x, int y);** | **对双击做出回应** | **int x int y** | **void** |
| **4** | **void openLattice(int x, int y);** | **打开格子** | **int x int y** | **void** |
| **5** | **void setRandMine(int x, int y);** | **随机布雷** | **int x int y** | **void** |
| **6** | **void setMineAround();** | **设置雷数** | **void** | **void** |
| **7** | **void on\_gameStatusChange();** | **改变游戏状态** | **void** | **void** |
| **8** | **void on\_flagNumChange();** | **改变旗数** | **void** | **void** |
| **9** | **void showAllMine(int result);** | **显示所有雷** | **int result** | **void** |
| **10** | **void changeLatticeTheme();** | **改变游戏主题** | **void** | **void** |

**MainWindow:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序  号 | 函数原型 | 功能 | 参数 | 返回值 |
| **1** | **void on\_actionCustomize\_C\_triggered();** | **实现自定义难度** | **void** | **void** |
| **2** | **void on\_actionPrimary\_P\_triggered();** | **实现初级难度** | **void** | **void** |
| **3** | **void on\_actionIntermediate\_M\_triggered()** | **实现中级难度** | **void** | **void** |
| **4** | **void on\_actionAdvanced\_A\_triggered();** | **实现高级难度** | **void** | **void** |
| **5** | **void on\_actionNewGame\_N\_triggered();** | **开始新游戏** | **void** | **void** |
| **6** | **void on\_actionClassic\_C\_triggered();** | **经典模式** | **void** | **void** |
| **7** | **void on\_actionDog\_D\_triggered();** | **小狗模式** | **void** | **void** |
| **8** | **void on\_actionPvZ\_P\_triggered();** | **植物大战僵尸模式** | **void** | **void** |
| **9** | **void on\_actionAbout\_A\_triggered();** | **游戏关于** | **void** | **void** |

**LatticeItem:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序  号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | **void mousePressEvent(QGraphicsSceneMouseEvent \*event) override;** | **检测鼠标点击事件** | **void** | **void** |
| 2 | **void mouseDoubleClickEvent(QGraphicsSceneMouseEvent \*event) override;** | **检测鼠标双击事件** | **void** | **void** |
| 3 | **bool isMine() { return m\_isMine; }** | **判断是否是雷** | **void** | **bool** |
| 4 | **int mineAround() { return m\_mineAround; }** | **判断周围雷数** | **void** | **int** |
| 5 | **int status() { return m\_status; }** | **返回状态** | **void** | **int** |
| 6 | **int theme() { return m\_theme; }** | **返回主题** | **void** | **int** |
| 7 | **void setRow(int row) { m\_row = row; }** | **设置行数** | **int row** | **void** |
| 8 | **void setColumn(int column) { m\_column = column; }** | **设置列数** | **int column** | **void** |
| 9 | **void setMine(bool isMine) { m\_isMine = isMine; }** | **布雷** | **Bool isMine** | **void** |
| 10 | **void setMineAround(int mineAround) { m\_mineAround = mineAround; }** | **布置周围的雷** | **int mineAround** | **void** |
| 11 | **void setStatus(int status) { m\_status = status; }** | **设置状态** | **int status** | **void** |
| 12 | **void setTheme(int theme) { m\_theme = theme; }** | **设置主题** | **int theme** | **void** |
| 13 | **void updateLattice();** | **更新格子** | **void** | **void** |
| 14 | **int r\_leftClick();** | **左键单击** | **void** | **int** |
| 15 | **int openLattice();** | **打开格子** | **void** | **int** |

**EmojiScene:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序  号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | **void setEmojiTheme(int theme)** | **设置表情主题** | **int theme** | **void** |
| 2 | **void setEmojiStatus(int status)** | **设置表情状态** | **int status** | **void** |

**EmojiItem:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序  号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | **int m\_theme;** | **设置emoji的主题** | **void** | **int** |
| 2 | **int m\_status;** | **设置emoji的状态（笑脸或哭脸）** | **void** | **int** |
| 3 | **void setTheme(int theme) { m\_theme = theme; }** | **设置主题** | **int theme** | **void** |
| 4 | **void setStatus(int status) { m\_status = status; }** | **设置状态** | **int status** | **void** |
| 5 | **void updateEmoji();** | **更新emoji的贴图** | **void** | **void** |

**DialogDifficulty：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序  号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | **int rowCount();** | **计算行数** | **void** | **int** |
| 2 | **int columnCount();** | **计算列数** | **void** | **int** |
| 3 | **int mineCount();** | **计算雷的个数** | **void** | **int** |
| 4 | **void iniDialogDifficulty(int row, int column, int mine);** | **在打开自定义难度对话框时初始化对话框中三个spinbox的值** | **int row, int column, int mine** | **void** |
| 5 | **void on\_spinBoxRow\_valueChanged(int arg1);** | **在行数设置发生变化时改变雷数设置的最大值** | **int arg1** | **void** |
| 6 | **void on\_spinBoxColumn\_valueChanged(int arg1);** | **在列数设置发生变化时改变雷数设置的最大值** | **int arg1** | **void** |

**5． 高层算法设计**

下面给出该项目中的高层算法的伪代码：

**难度选择对话框函数：**

**void DialogDifficulty::iniDialogDifficulty(int row, int column, int mine)**

**{**

**在行数设置发生变化时改变雷数设置的最大值**

**}**

**void DialogDifficulty::on\_spinBoxRow\_valueChanged(int rowValue)**

**{**

**设置雷数最小值为格子总数的0.1倍**

**设置雷数最大值为格子总数的0.7倍**

**}**

**在列数设置发生变化时改变雷数设置的最大值**

**void DialogDifficulty::on\_spinBoxColumn\_valueChanged(int colunmValue)**

**{**

**设置雷数最小值为格子总数的0.1倍**

**设置雷数最大值为格子总数的0.7倍**

**}**

**打开格子函数：**

**void PlayScene::openLattice(int x, int y)**

**{**

**if (如果该格子未被打开)**

**{**

**open\_times++;**

**if (open\_times == 1)**

**{**

**在该格子周围布雷并保证第一次点击的位置不是雷**

**}**

**将该格子状态更新**

**if (被打开的格子是0，则要递归打开周围的格子) {**

**在未越界的情况下，递归打开该格子的周围格子**

**}**

**布雷函数：**

**void PlayScene::setMineAround()**

**{**

**遍历整个棋盘**

**if (当前格子不是雷)**

**{ 统计这个格子周围的地雷数**

**if ( 搜寻的区域没有越界)**

**{**

**布雷；**

**}**

**}**

**游戏结束显示函数：**

**void PlayScene::showAllMine(int result)**

**{**

**if(有已标记的地雷)**

**{**

**已标记的地雷用“绿色地雷”显示**

**}**

**else if ( 游戏失败)**

**{**

**则未标记的地雷用“红色地雷”表示**

**}**

**else**

**{**

**若游戏胜利，则未标记的地雷也用“绿色地雷”表示**

**}**

**更新格子状态 }**

**}**

**更新格子状态函数：**

**void LatticeItem::updateLattice()**

**{**

**switch (主题)**

**{**

**case 经典模式:**

**将所有格子贴图改为经典模式贴图**

**break;**

**case 小狗模式：**

**将所有格子贴图改为小狗主题贴图**

**break;**

**case 植物大战僵尸主题:**

**将所有格子贴图改为植物大战僵尸主题贴图**

**break;**

**}**

**}**