程序设计实践答辩

Designed by Marcus 2024-11-01

作者：梁俊斌

目录

1. 项目简介
2. 项目功能
3. 设计思路
4. 模块介绍

项目简介

项目名称: 扫雷小游戏

项目作者: 梁俊斌

实现方式: `C++` 和 `QT`

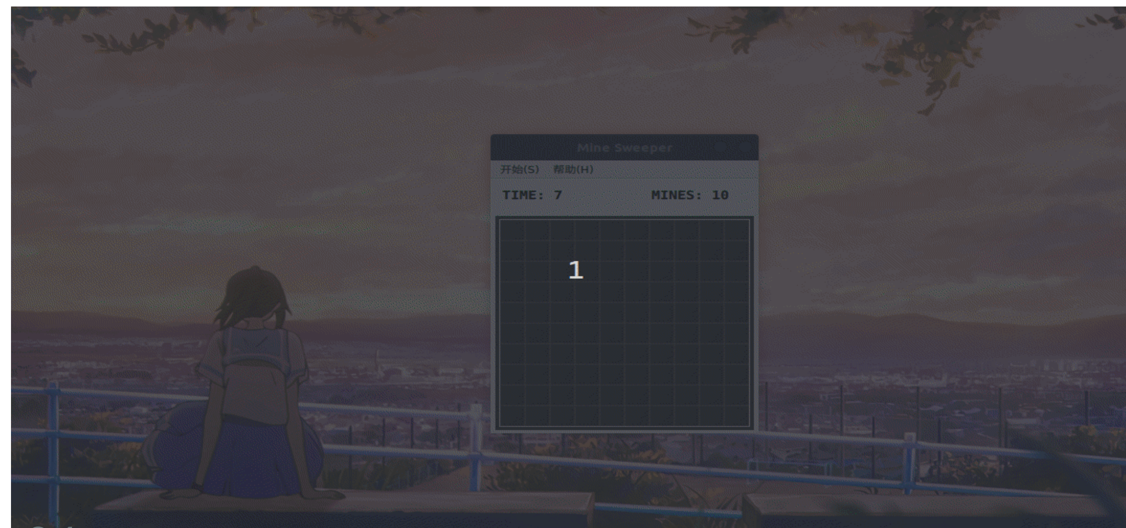
管理形式: `qmake` 和 `qrc` QT产生的文件来管理

该项目是一个经典的扫雷游戏实现，旨在通过编写一个完整的 `GUI` 应用程序来展示C++和QT的综合应用。游戏支持多种难度选择，玩家可以自定义地图大小和雷数。游戏界面友好，操作简单，适合各类用户体验。

项目功能

- **多种难度选择**：初级、中级、高级和自定义模式。
- **游戏计时**：记录玩家的游戏时间。
- **标记功能**：玩家可以标记雷的位置。
- **胜利提示**：玩家成功扫雷后会有胜利提示界面。

运行效果如下:



项目结构如下:

```
MineSweeper
├─── .vscode/                  # 开发环境配置文件
│    ├─── c_cpp_properties.json
│    └─── settings.json
├─── build/                    # 编译生成文件夹
├─── include/                  # 头文件
│    ├─── commons/              # 公共资源和常量
│    ├─── cores/                # 游戏逻辑核心组件
│    └─── views/                # 界面视图
├─── res/                      # 资源文件夹
│    ├─── images/               # 图片资源
│    └─── sounds/               # 音频资源
├─── src/                      # 源代码
```

采用典型的项目化目录结构有助于提高代码的可维护性、可读性和可扩展性。

核心模块介绍

这个部分介绍了`MineBlock`类，它用于处理地雷区块的各种状态和操作。`MineBlock`类包含多个成员变量和方法，用于管理区块的类型、数字、标记状态、覆盖状态等。通过这些方法，游戏可以精确地控制每个区块的行为和显示。

```
enum class BlockType
{
EMPTY,
```

以上就是`block.h`的完整的代码,下面我们就拿部分函数的实现出来简单介绍

```cpp
void MineBlock::init()
{
    this->type = BlockType::EMPTY;  // 设置区块类型为空
    this->number = 0;               // 初始化区块数字为0
    this->covered = true;           // 设置区块为未揭开状态
    this->mistaken = false;         // 初始化错误标记状态为false
    this->marked = false;           // 初始化标记状态为false
    this->touched = false;          // 初始化点击状态为false
}
// 将区块设置为地雷
void MineBlock::setAsMine()
{
    this->type = BlockType::MINE;  // 设置区块类型为地雷
}
```

从具体函数代码实现可以看到就是 `MineBlock类` 是由一个初始化函数和很多的用于变量接口的函数(`get函数`)组成

在这一部分，我们将详细介绍 `GameController` 类的实现。`GameController` 类是整个扫雷游戏的核心控制器，负责管理游戏状态、处理用户输入以及更新游戏界面。

`GameController` 类的主要功能

- **游戏初始化**：设置游戏的初始状态，包括地雷的分布和数字的计算。
- **游戏状态管理**：控制游戏的开始、暂停、失败和成功状态。
- **用户输入处理**：处理用户的左键和右键点击操作。
- **游戏逻辑更新**：根据用户的操作更新游戏界面和状态。

主要函数介绍

```cpp
void GameController::setLevel1()
{
    this->level = GameLevel::LEVEL1;
    this->rows = Level1::ROWS;
    this->cols = Level1::COLS;
    this->mineInitCount = Level1::MINE_INIT_COUNT;
}

void GameController::setLevel2()
{
    this->level = GameLevel::LEVEL2;
    this->rows = Level2::ROWS;
    this->cols = Level2::COLS;
    this->mineInitCount = Level2::MINE_INIT_COUNT;
}

void GameController::setLevel3()
{
    this->level = GameLevel::LEVEL3;
    this->rows = Level3::ROWS;
    this->cols = Level3::COLS;
    this->mineInitCount = Level3::MINE_INIT_COUNT;
}
```

```cpp
void GameController::setResume()
{
    if (status == GameStatus::PAUSE)
    {
        status = GameStatus::PLAYING;
    }
}

void GameController::restart()
{
    status = GameStatus::PLAYING;

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            blocks[i][j].init();
        }
    }
    remainFlagCount = mineInitCount;
}
```

从上述代码可以看出，`GameController`类通过不同的函数来设置游戏的难度，并调用`restart`函数来重新初始化游戏。

用户输入处理

```cpp
bool GameController::leftButtonClick(int x, int y)
{
    // 处理左键点击事件
    if (status != GameStatus::PLAYING) return false;
    MineBlock& block = getBlock(x, y);
    if (block.isCovered() && !block.isMarked())
    {
        block.setUncovered();
        if (block.getType() == BlockType::MINE)
        {
            status = GameStatus::FAILURE;
            return false;
        }
        else if (block.getType() == BlockType::EMPTY)
        {
            uncoverEmptyBlocks();
        }
    }
    return true;
}
```

```cpp
bool GameController::rightButtonClick(int x, int y)
{
    // 处理右键点击事件
    if (status != GameStatus::PLAYING) return false;
    MineBlock& block = getBlock(x, y);
    if (block.isCovered())
    {
        block.setMarked(!block.isMarked());
        remainFlagCount += block.isMarked() ? -1 : 1;
    }
    return true;
}
```

上述代码展示了 `GameController` 类如何处理用户的左键和右键点击事件，并根据点击的结果更新游戏状态。

通过 `GameController` 类，这些函数和逻辑共同构成了一个完整的游戏控制器，使得游戏能够顺利运行并响应用户的操作。

资源管理模块介绍

资源管理模块主要负责加载和管理游戏中的图片和音频资源，以及提供给其他模块使用。通过资源管理模块，游戏可以方便地访问和使用各种资源，使得游戏正确渲染,提高游戏的可维护性和可扩展性。

下面我们来看一下 `GameResource` 类的部分实现

```
// 加载各类游戏图标资源,资源路径以 : 开头是为了指示这是一个资源文件路径, 而不是文件系统
pBlockPixmap = new QPixmap("qrc:/images/block.png"); // 墙块图标
pCoverPixmap = new QPixmap("qrc:/images/cover.png"); // 覆盖/未揭示图标
pErrorPixmap = new QPixmap("qrc:/images/error.png"); // 错误图标
pFlagPixmap = new QPixmap("qrc:/images/flag.png"); // 旗帜图标
pMinePixmap = new QPixmap("qrc:/images/mine.png"); // 矿井图标
pTouchPixmap = new QPixmap("qrc:/images/touch.png"); // 触碰/揭示图标

// 循环加载数字图标资源 (1-9)
for (int i = 0; i < SceneProperties::NUMBER_COUNT; i++)
{
    // 这里使用QString是因为可以使用.arg()方法实现使用占位符, 方便循环加载。如果像下面
    pNumberPixmaps[i] = new QPixmap(QString(":/images/number_%1.png").arg(i + 1));
}

// 初始化画刷：黑色和灰色
pBlackBrush = new QBrush(QColor(Colors::BLACK)); // 黑色画刷
pGrayBrush = new QBrush(QColor(Colors::GRAY)); // 灰色画刷
pWhiteBrush = new QBrush(QColor(Colors::WHITE)); // 白色画刷
```

```
// 初始化声音效果
pClickSound = new QSoundEffect(); // 点击声音效果
pSuccessSound = new QSoundEffect(); // 成功声音效果
pFailureSound = new QSoundEffect(); // 失败声音效果

// 设置声音文件路径
pClickSound->setSource(QUrl("qrc:/sounds/click.wav")); // 点击声源
pFailureSound->setSource(QUrl("qrc:/sounds/failure.wav")); // 失败声源
pSuccessSound->setSource(QUrl("qrc:/sounds/success.wav")); // 成功声源
```

代码分析

从代码图片中可以看出，`GameResource`类通过`QPixmap`和`QSound`类来加载和管理游戏中的图片和音频资源。它还通过指针成员变量和`get`函数，`GameResource`类提供了一种简单的方式来访问和使用这些资源。

```
QPixmap* GameResources::getBlockPixmap()
{
    return pBlockPixmap;
}

QPixmap* GameResources::getCoverPixmap()
{
    return pCoverPixmap;
}

QPixmap* GameResources::getErrorPixmap()
{
    return pErrorPixmap;
}

QPixmap* GameResources::getFlagPixmap()
{
    return pFlagPixmap;
}
```

项目的另外资源管理文件

通过 `.pro` 文件以及 `qrc` 文件来管理项目中的资源文件，这样可以方便的访问和使用这些资源。同时,将图片以及音频资源放在 `res` 文件夹中,方便项目整体调用

```
# 指定源文件列表
SOURCES += \
        src/cores/block.cpp \
        src/views/dialogs/aboutdialog.cpp \
        src/views/dialogs/customdialog.cpp \
        src/views/dialogs/successdialog.cpp \
        src/views/widgets/scenewidget.cpp \
        src/main.cpp \
        src/views/mainwindow.cpp \
        src/cores/gamecontroller.cpp \
        src/commons/gameresources.cpp


# 指定头文件列表
HEADERS += \
        include/cores/block.h \
        include/views/dialogs/aboutdialog.h \
        include/views/dialogs/customdialog.h \
        include/views/dialogs/successdialog.h \
        include/views/widgets/scenewidget.h \
        include/views/mainwindow.h \
        include/cores/gamecontroller.h \
        include/commons/gameresources.h \
        include/commons/constants.h
```

```
<RCC>
    <qresource prefix="/">
        <file>images/icon.ico</file>
        <file>images/block.png</file>
        <file>images/cover.png</file>
        <file>images/error.png</file>
        <file>images/flag.png</file>
        <file>images/mine.png</file>
        <file>images/touch.png</file>
        <file>images/number_1.png</file>
        <file>images/number_2.png</file>
        <file>images/number_3.png</file>
        <file>images/number_4.png</file>
        <file>images/number_5.png</file>
        <file>images/number_6.png</file>
        <file>images/number_7.png</file>
        <file>images/number_8.png</file>
        <file>images/title.png</file>
        <file>sounds/click.wav</file>
        <file>sounds/failure.wav</file>
        <file>sounds/success.wav</file>
    </qresource>
</RCC>
```

界面设计模块介绍

界面设计模块主要负责实现游戏的用户界面，包括游戏主界面、游戏设置界面、游戏胜利界面和游戏失败界面等。通过界面设计模块，游戏可以展示给用户友好的界面，提高用户体验。

下面我们来看一下 `SceneWidget` 类的部分实现

`SceneWidget` 类通过 `QPainter` 类来绘制游戏界面，包括地雷区块、数字区块、标记区块等各种情况。编写不同情况的函数,然后通过重写 `paintEvent` 函数从而调用各情况的函数， `SceneWidget` 类实现了游戏界面的绘制。

代码部分截图

```
void SceneWidget::paintEvent(QPaintEvent* event)
{
    QPainter painter(this);

    // 设置平滑渲染提示
    painter.setRenderHints(QPainter::SmoothPixmapTransform);

    // 绘制外层背景
    paintOuterBackground(painter);

    // 绘制内层背景
    paintInnerBackground(painter);

    // 遍历所有方块
    for (int rows = 0; rows < pGameController->getRows(); rows++)
    {
        for (int cols = 0; cols < pGameController->getCols(); cols++)
        {
            MineBlock& block = pGameController->getBlock(rows, cols);

            // 计算方块的位置
            int x = SceneProperties::MARGIN + rows * SceneProperties::BLOCK_SIZE;
            int y = SceneProperties::MARGIN + cols * SceneProperties::BLOCK_SIZE;

            // 根据游戏状态绘制方块
            if (pGameController->getStatus() == GameStatus::PAUSE)
            {
                painter.drawPixmap(x, y, *pGameResources->getCoverPixmap());
            }
            else if (block.isCovered())
            {
                paintCoveredBlock(painter, block, x, y);
            }
            else
            {
                paintUncoveredBlock(painter, block, x, y);
            }
        }
    }
}
```

通过ui文件设计界面

通过 `QT` 的 `ui` 文件来设计界面，这样可以方便的实现界面的布局和设计。通过 `ui` 文件，可以直观地看到界面的布局和组件，方便调整和修改。同时，通过 `ui` 文件和 `QT` 的信号槽机制，可以方便地实现界面和逻辑的交互。

> 通过 `ui` 文件设计主界面,胜利结算,计时界面，可以直观地看到界面的布局和组件，方便调整和修改。同时，通过 `ui` 文件和 `QT` 的信号槽机制，可以方便地实现界面和逻辑的交互。

主要ui界面有:

- `mainwindow.ui`
- `aboutdialog.ui`
- `customdialog.ui`
- `successdialog.ui`

总结

> “ 这个项目使用`C++`和`QT`来实现一个完整的`GUI`扫雷游戏。在项目中，通过设计和实现游戏的核心逻辑、资源管理、界面设计等模块，以及如何将这些模块整合在一起，实现一个完整的游戏。 ”

感谢聆听

Designed by Marcus 2024-11-01

作者：梁俊斌