扫雷 UWP

软件设计文档

摘要

基于 UWP 开发的一个扫雷游戏，向经典的扫雷游戏致敬. 采用全新的图片素材，经典的玩法，不一样的体验。

Minesweeper

https://github.com/zjx386154110/Minesweeper

一、项目介绍

1. 项目目标 扫雷是一个经典的益智小游戏,因此选择扫雷这个经典的 windows 游戏作为开发目标，目的是在 windows10 新型通用平台上重现经典扫雷的界面和玩法，并融入一些新的元素，使扫雷游戏更加趣味耐玩。
2. 游戏玩法：
   1) 游戏区域包括雷区、地雷计数器、标记模式开关、退出按钮。
   2) 雷区中根据难度随机分布着一定数量的地雷，玩家必须找到雷区中所有不 是地雷的格子，而不能踩到地雷。
   3) 点击格子即可打开格子，格子中的数字代表周围 3*3 区域中含有的地雷数。 玩家需根据这些数字判断出安全的格子。
   4) 点击标记模式开关可切换到标记模式，标记模式下点击格子不会打开，而 是进行标记用以辅助判断，再点击一次格子即可取消标记。

二、项目设计

1. 开发环境

1) 操作系统：windows10 专业版
2) 开发工具：Visual Studio 2015
3) 编程语言：XAML 和 C#

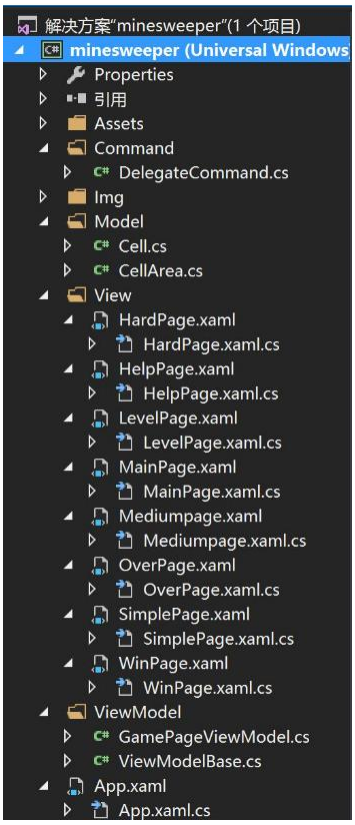
2. 技术选型

1) 技术选型表

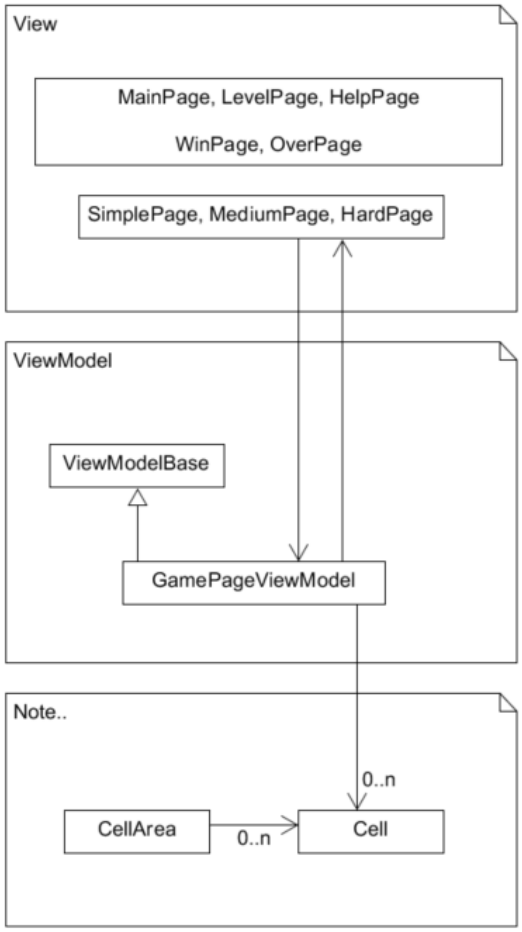
项目	UWP app	选型理由
1.终端支持		
1.1 终端类型	PC	性能强大，可以支持游戏流畅运行
1.1 开发语言框架	C#, XAML	开发简单，有现成的设计模式可参考
1.2 平台	Windows 通用平台	经 Windows 审核发布，易于推广
1.3 传感器	鼠标，触控屏	符合传统用户操作习惯
2.开发平台与工具		
2.1 IDE	Visual Studio 2015	开发 UWP 的首选
2.2 集成与测试	Visual Studio 2015	自带强大的调试工具
2.3 源代码管理	Github	世界知名的代码项目管理平台

3. 项目架构

1) 项目使用经典的 MVVM 模式来组织结构，如图所示。


解决方案"minesweeper"(1 个项目)
minesweeper (Universal Windows
Properties
引用
Assets
Command
DelegateCommand.cs
Img
Model
Cell.cs
CellArea.cs
View
HardPage.xaml
HardPage.xaml.cs
HelpPage.xaml
HelpPage.xaml.cs
LevelPage.xaml
LevelPage.xaml.cs
MainPage.xaml
MainPage.xaml.cs
Mediumpage.xaml
Mediumpage.xaml.cs
OverPage.xaml
OverPage.xaml.cs
SimplePage.xaml
SimplePage.xaml.cs
WinPage.xaml
WinPage.xaml.cs
ViewModel
GamePageViewModel.cs
ViewModelBase.cs
App.xaml
App.xaml.cs

2) MVVM 架构示意图


View
MainPage, LevelPage, HelpPage
WinPage, OverPage
SimplePage, MediumPage, HardPage
ViewModel
ViewModelBase
GamePageViewModel
Note..
0..n
CellArea
0..n
Cell

3) 各个模块的详细功能如下

MainPage 页面为游戏主菜单，

WinPage 页面为游戏胜利界面，

OverPage 页面为游戏失败界面。

这三个页面都包括开始游戏按钮、游戏帮助按钮、 离开游戏按钮。点击开始游戏按钮进入选择难度界面，点击游戏帮助按钮 进入游戏帮助界面，点击离开游戏按钮退出程序。

HelpPage 页面为游戏帮助页面，包含游戏元素介绍及退回祝词啊但的按 钮。

LevelPage 页面为选择难度界面，包括三种游戏难度。选择难度后即跳转 到相应难度的游戏界面。

SimplePage、MediumPage、HardPage 分别为简单、中等、困难三种 不同难度的游戏界面。简单难度为 10*10 个格子，10 个地雷；中等难度 为 14*16 个格子，30 个地雷；困难难度为 16*28 个格子，90 个地雷。 标记模式开关、地雷计数器、退出按钮以随机位置的形式嵌入在游戏区域 中。其 xaml 中使用 ItemsTemplate 及 ItemsSource 绑定来自 ViewModel 中的数据。

4) ViewModel 文件夹包括用以建立 View 和 Model 之间数据传递的文件。 在 GamePageViewModel.cs 中接受来自 View 传来的雷区属性参数，用 以初始化雷区并返回雷区的 ObservableCollection<Cell>集合数据。

5) Model 文件夹包括 Cell 和 CellArea 两个类文件。Cell 类代表一个格子按钮，包含格子坐标、显示图片路径、含有地雷数的 属性及点击命令的实现。 CellArea 类代表雷区，包含雷区行数、列数、雷数等属性以及初始化雷区、 点击格子、标记格子等方法的实现。

6) Commond 文件夹包括用以实现执行命令功能的文件。

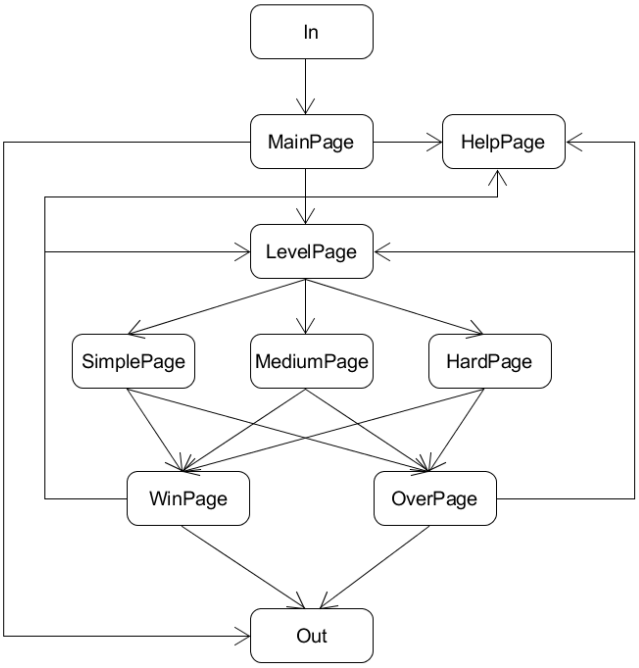
7) Img 文件夹包括用于显示在格子上的 png 图像文件。

4. 页面设计

游戏的页面总共包括：

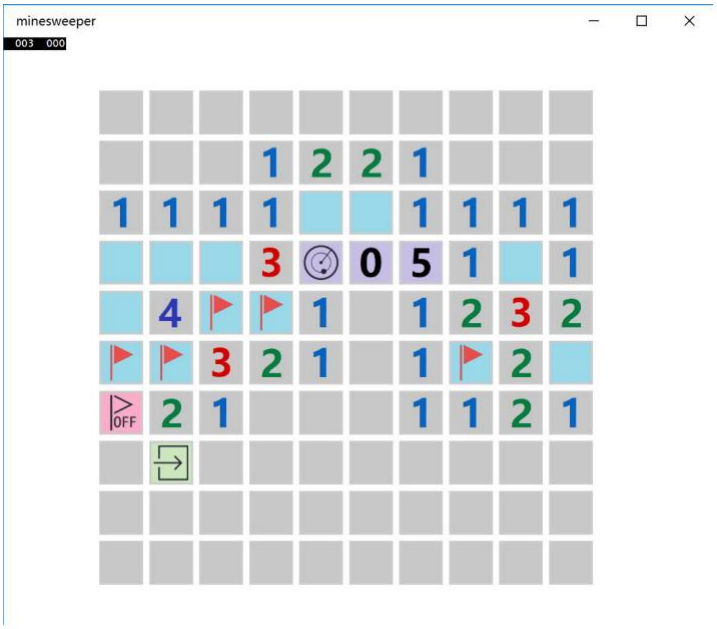
1) MainPage(开始界面)
2) LevelPage(难度选择界面)
3) HelpPage(帮助界面)
4) WinPage(游戏胜利界面)
5) OverPage(游戏失败界面)
6) SimplePage(简单难度的游戏界面)
7) MediumPage(中等难度的游戏界面)
8) HardPage(困难难度的游戏界面)

各页面之间的跳转设计：



5. UI 设计：

UI 采用 UWP 扁平化设计理念，菜单界面以黑白灰色调为基础，游戏界面以浅色调及自主设计的扁平化多彩图标为主，使得游戏界面简洁又时尚。



6. 游戏性设计

采用经典的扫雷玩法及三种梯度的难度等级设计，不失传统扫雷的味道。

创新性地将标记按钮、地雷计数器、退出按钮以随机位置嵌入到雷区中，使得界面更加简约统一，也使得游戏更加有趣。

每次新开始的游戏都是随机布雷及随机分布控件按钮的，并且人性化地设计了第一次点开的格子必定能打开一块空白区域，避免了出现一开始需要猜雷的情况。

三、软件设计技术

Design Patterns(设计模式)

采用了微软带来的一种新型框架 MVVM，其宗旨在于，视图(View)部分只对 ViewModel 进行操作，而一切需要的数据也是由 ViewModel 返回，Model 不直接受到 View 的控制

选取代码中的一例来看，在这段代码中，SimplePage 页面只是创建了一个 ViewModel 而已，对 Model 操作的部分其实都是交由 ViewModel 完成

```
public sealed partial class SimplePage : Page
{
    public SimplePage()
    {
        InitializeComponent();
        DataContext = new GamePageViewModel(10, 10, 10);
        var viewTitleBar = Windows.UI.ViewManagement.ApplicationView.
              GetForCurrentView().TitleBar;
        viewTitleBar.BackgroundColor = Windows.UI.Colors.CornflowerBlue;
        viewTitleBar.ButtonBackgroundColor = Windows.UI.Colors.CornflowerBlue;
```

```
    }
}
```

关于 Model 的设计见如下代码

```
public class Cell
{
    private int x;
    public int X { get { return x; } set { x = value; RaisePropertyChanged("X"); } }
    private int y;
    public int Y { get { return y; } set { y = value; RaisePropertyChanged("Y"); } }
    private string imgsrc;
    public string ImgSrc { get { return imgsrc; } set { imgsrc = value;
         RaisePropertyChanged("ImgSrc"); } }
    private int minenum;
    public int MineNum { get { return minenum; } set { minenum = value;
         RaisePropertyChanged("MineNum"); } }
    public Cell(int x, int y)
    {
        X = x;          //行坐标
        Y = y;          //列坐标
        ImgSrc = "../Img/方块.png";  //显示图像路径
        MineNum = 0;  //-1代表格子为地雷，-2代表格子为控件，其他非负数代表格子周围含有的地雷数
    }
    //实现格子点击命令
    private DelegateCommand cellclick;
    public DelegateCommand CellClick
    {
        get{
            return cellclick ?? (cellclick = new DelegateCommand((Object obj) => {
                CellArea.ClickCell(this);
            }, null));
        }
    }
}
```

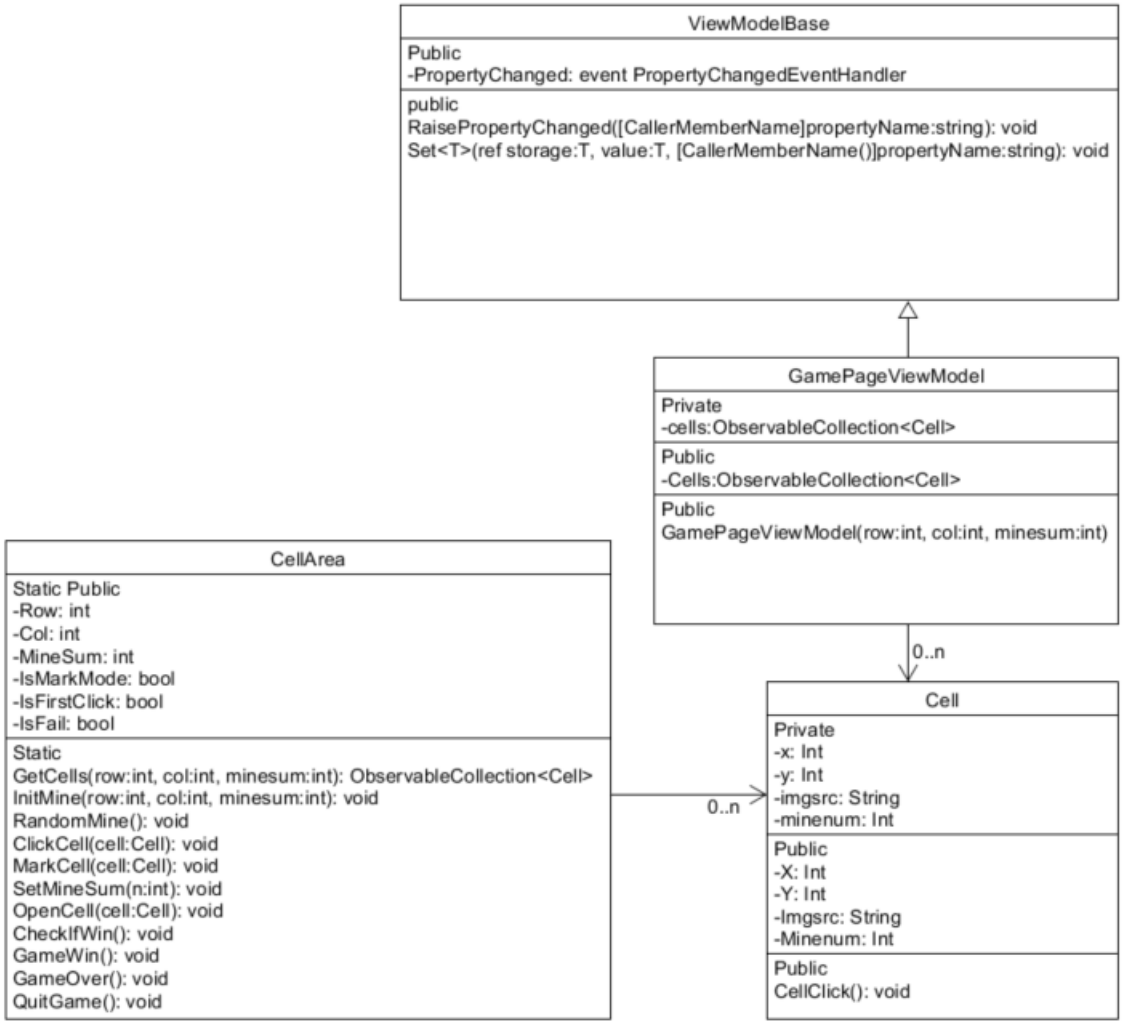
而 ViewModel 则是作为一个中间件

```
public class GamePageViewModel : ViewModelBase
{
    private ObservableCollection<Cell> cells;
    public ObservableCollection<Cell> Cells { get { return cells; } set { cells = value;
         RaisePropertyChanged("Cells"); } }
    public GamePageViewModel(int row, int col, int minenum)
    {
        //获取雷区的集合数据
        Cells = CellArea.GetCells(row, col, minenum);
    }
```

}

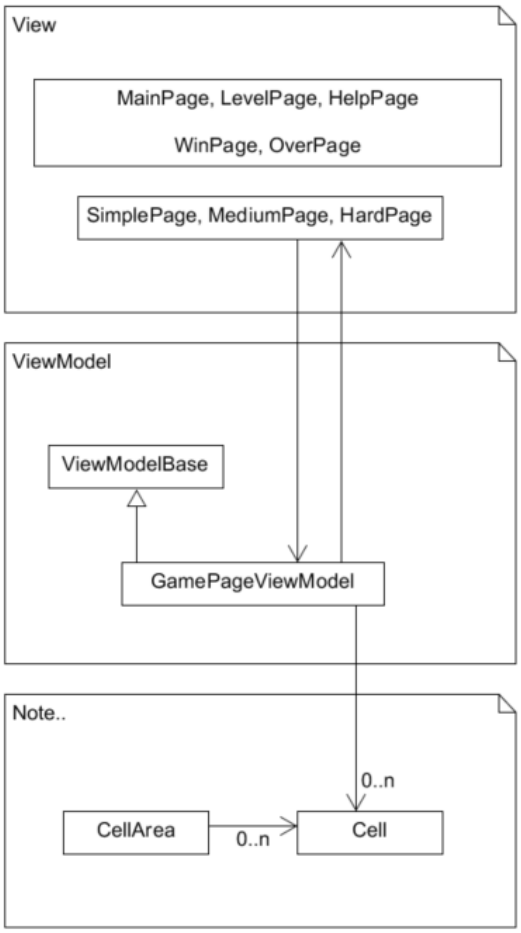
Object-Oriented Programming(面向对象设计)

面向对象设计要求我们把处理的物体抽象成一个个类，并且在类之间只使用接口相互访问. 基于这个思想我们为程序设计的类图如下



Structured programming(结构程序设计)

结构程序设计在于使模块之间的访问尽可能的少，这也是我们为了降低耦合度的追求. 基于这个思想我们设计出了如下的结构



将页面的设计和模型，逻辑的设计完全分开，这样修改页面时就不会触碰到底层游戏逻辑，而游戏逻辑的修改也可以同时应用到多个页面

Event-driven(事件驱动)

软件的一切行为都是事件驱动的，即用户触发一个事件(在这里是指点击某个按钮)，程序进行相应并返回相应的结果. 以 MainPage 为例，其相应代码如下

```
public sealed partial class MainPage : Page
{
        public MainPage()
        {
            this.InitializeComponent();
            TileContent content = createTile();
            UpdateTile(content);
        }
        private void NewGameClick(object sender, RoutedEventArgs e)
        {
            Frame.Navigate(typeof(LevelPage));
        }
        private void GameHelp(object sender, RoutedEventArgs e)
        {
```

```
            Frame.Navigate(typeof(HelpPage));
        }
        private void EXIT(object sender, RoutedEventArgs e)
        {
            Application.Current.Exit();
        }
}
```

在用户点击开始游戏或者帮助按钮时，会跳转到对应的页面，而如果用户点击退出，那么显而易见的，会让游戏退出。