【奇妙森林的光颖传说】

森林火灾监测系统设计文档

1 系统开发与运营环境

◆ **本森林火灾监测系统开发环境**

操作系统：Windows10

开发语言：基于.NET Framework 框架，在 Visual Studio 2019 下采用 C#语言开发

开发平台：基于 ArcGIS Engine

◆ **本森林火灾监测系统开发者信息**

黄　妙 北京师范大学 地理科学学部 201811051142@mail.bnu.edu.cn

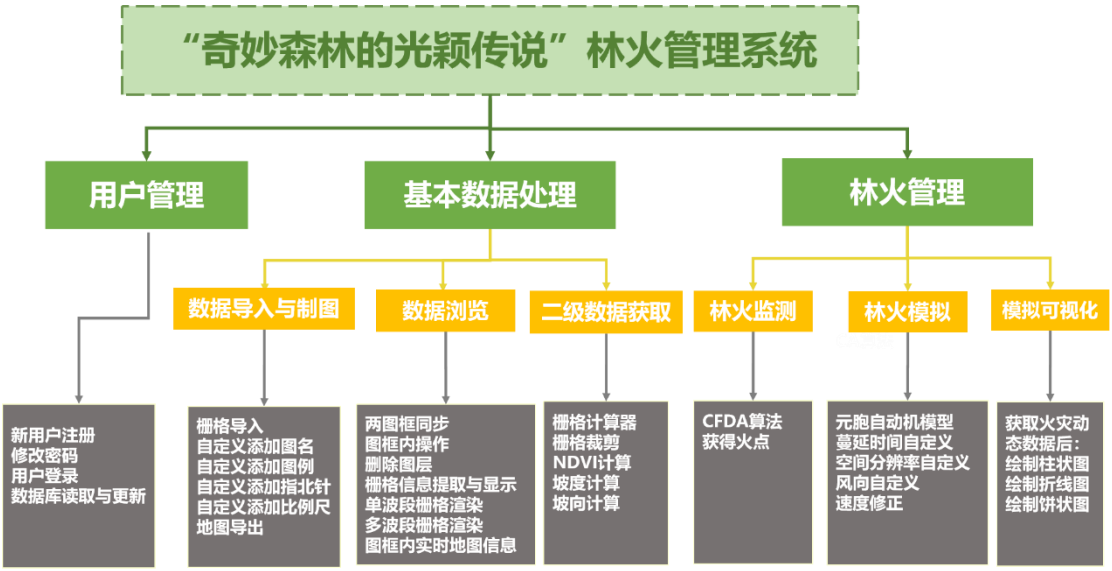
张　颖 北京师范大学 地理科学学部 201811051125@mail.bnu.edu.cn

李森洋 北京师范大学 地理科学学部 201811051123@mail.bnu.edu.cn

2 系统研究背景与目的

森林生态系统是重要的陆地生态系统类型，森林覆盖着全球陆地面积的 1/3，森林年光合产量约占陆地生态系统的 2/3[1]。全球变暖等一系列的全球变化对地球上的植物尤其是对森林产生了巨大的影响，影响森林生态系统物候、促进森林生态系统生物入侵……但是在诸多危害森林的因素中，森林火灾的破坏性最大[2]。森林火灾后，烧伤林木，生长衰退，为大量病虫害发生造成有利环境。林分极易受到森林害虫的再次侵袭，损失严重，形成火灾、虫害、火灾的恶性循环。因此，及时实现对于森林火灾灾情的监测、预测以及评估对于政府管控火情并做好应急防控、对森林生态系统的保护和管理起到至关重要的作用。

本森林火灾监测系统以森林火灾监测和预测为基点，其功能主要分为用户管理、基本数据处理和林火管理三大模块，分别实现系统用户的注册与管理，数据导入、浏览和数据再处理，林火监测、模拟、可视化三大模块的功能，林火管理系统的具体功能设计见图 1。



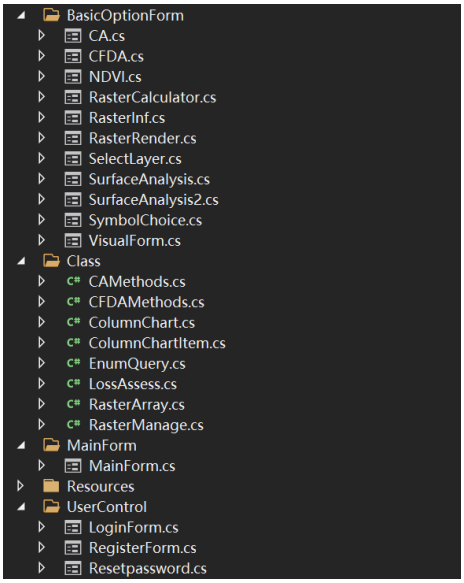
图表 1 林火管理系统框架设计

2.1 数据存储与组织

数据是一个 GIS 系统的前提和基础，为实现火灾监测、火势预测与损害评估，需要多种数据的支持。本系统中用于火灾监测和模拟的文件地理数据库中均为火灾监测和模拟所需的气象要素与遥感**栅格**的**面数据**，此外用户账户管理系统数据保存于本地文件中。

2.2 代码组织与说明

在代码实现方面，系统的代码设计以树形图格式展示在程序的【解决方案资源管理器】中，解决方案包含一个或多个项目，每个项目有其自身的文件夹，用户还可以添加新的文件夹或项目以满足程序功能拓展的需要。



图表 2 解决方案资源管理器模块界面

通常，基于 ArcGIS Engine 开发的程序集主要包括 Properties 文件、引用文件、主窗体及其他窗体、Program.cs 文件, bin 文件与 obj 文件共六个部分，该系统除了基本的组成之外,还包含其他部分(如数据部分,图标部分等)，下面对系统的整个程序集构成做一个详细的介绍。

◆ Properties 文件

该主要包括程序集的版本信息、项目的资源数据以及项目的属性设置等。AssemblyInfo.cs 文件存储的是程序的信息，如名称、版本等，“项目资源”中则包含应用程序所需的二进制数据、文本文件、音频或视频文件、字符串表、图标图像、XML 文件或其他类型的数据。项目资源数据以 XML 格式存储在.resx 文件中(默认文件名为 Resources.resx)，Settings.settings 文件则用来存储程序的属性信息及其他信息,允许用户修改并保存。

图表 3 Resources.resx 中存储的图标库

◆ 引用文件

存放着用于实现各种功能的类库,类库主要包括接口、抽象类和具体类等,本林火监测系统中引入了 MathNet 开源程序包。

◆ Appcode 文件

Appcode 文件中存放着一些自定义类：

自定义类	功能实现
CAMethod.cs	CA 元胞计算算法
CFDAMethods.cs	CFDA 计算算法
ColumnChart.cs	3D 柱状图绘制函数包
ColumnChartItem.cs	3D 图表构造函数包
EnumQuery.cs	制图要素标记枚举值
LossAssess.cs	林火损失评估算法
RasterArray.cs	栅格与数组交互函数包
RasterManage.cs	栅格数据处理函数包

Program.cs 是主程序入口，其中的 Application 和 Run 函数用于控制窗体的运行。

◆ bin 文件

bin 文件夹用于存放编译生成的二进制文件,其根据文件的不同用途，又建立了 Debug 和 Release 两个文件夹，分别存放用于调试和最终发布的文件。我们的账户管理系统的测试 txt 文件放在 Debug 文件下的 UserData.txt 文件中。

◆ obj 文件

obj 文件用来保存每个模块的编译结果。

◆ 图标文件

图标文件(MouseIcons 和图标)存放的是系统各个功能按钮的外观、鼠标样式及效果渲染图片。

◆ 数据文件

火灾数据文件存放的是基于 CA 算法需要的火灾监测数据，这些数据基于一些免费的遥感或者气象数据获取网站或平台获取，通过加载栅格数据来实现火势扩散模型的模拟。

◆ 主窗体与其他窗体包含：

窗体名称	功能实现
LoginForm.cs	用户登录
RegisterForm.cs	用户注册
Retpassword.cs	用户修改密码
MainForm.cs	程序主窗体
CA.cs	林火模拟窗体
CFDA.cs	林火监测窗体
NDVI.cs	NDVI 计算窗体
RasterCalculator.cs	栅格计算器窗体
RasterInf.cs	栅格信息窗体（栅格的属性信息表）
RasterRender.cs	栅格渲染窗体
SelectLayer.cs	栅格裁剪的图层选择窗体
SurfaceAnalysis.cs	计算坡度窗体
SurfaceAnalysis2.cs	计算坡向窗体
SymbolChoice.cs	样式选择器窗体
VisualForm.cs	图表可视化窗体

3 软件设计指南

3.1 总览

单击 bighm.exe 启动程序。

软件导航界面如下：

森林火灾监测系统的导航界面由【注册】、【登录】、【修改密码】三大功能组成，用户登录成功后将进入软件主界面。

图表 4 软件登录界面

软件主界面如下：

主界面主要包含主菜单、图层控制区、工具条以及数据浏览区域（数据视图与布局视图）。

图表 5 软件主界面

3.2 用户账号管理部分

该部分实现了用户的注册、登录和密码修改部分，主要代码分别存放于 RegisterForm.cs、LoginForm.cs 和 Resetpassword.cs。

3.2.1 用户注册

账号：用户输入账号名，且不可以输入汉字，用户名区分大小写。

密码：用户输入密码，且密码需大于 6 位数。

确认密码：用户再次输入相同密码，若两次密码不同则无法成功注册。

验证码：用户查看验证码图片中的数字并输入验证码，点击图片可以更换验证码，若验证码不同则无法成功注册。

是否为管理员：用户标记用户权限。

全部信息完整且符合要求则可以成功**注册**。

图表 6 用户注册功能

3.2.2 用户登录

用户名：用户输入用户名，若用户账号管理系统数据库中不存在该用户名，则提示"用户名不存在"且无法成功登录。

密码：用户输入密码，若用户累计输入 3 次错误密码，则无法进入系统，且每次输入错误密码后会弹窗提示剩余尝试次数。

全部信息完整且负责要求则可以成功**登录**。

图表 7 用户登录功能

3.2.3 用户修改密码

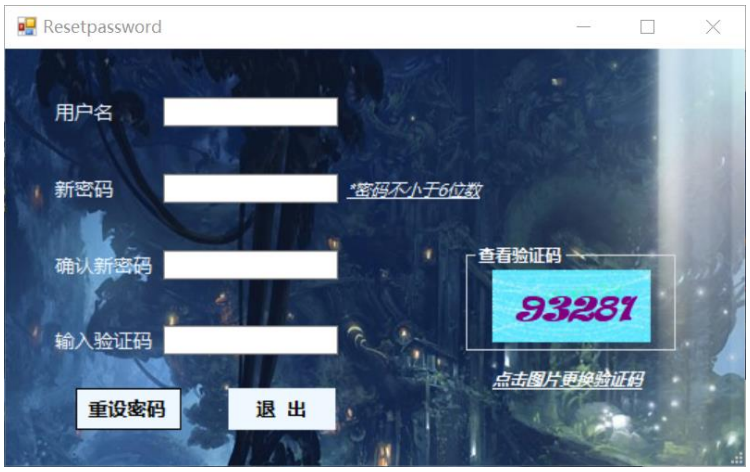
用户名：用户输入用户名，若用户账号管理系统数据库中不存在该用户名，则提示“用户名不存在”且无法成功修改。

新密码：用户输入新密码，密码需大于 6 位数。

确认新密码：用户名再次输入密码，两次输入密码需相同，若不同则无法成功修改密码。

输入验证码：用户查看验证码图片中的数字并输入验证码，点击图片可以更换验证码，若验证码不同则无法成功修改。

全部信息完整且负责要求则可以成功**修改密码**。



图表 8 用户修改密码功能

**设计优点与痛点

设计优点：

1、仿真实现了现实生活中用户数据管理系统需要的基本功能包括查看验证码、错误密码输入次数监测等。

设计痛点：

1、 没有调用本地或云端的 SQL Sever 数据库实现对于用户信息的数据管理。

2、 在用户信息存储和读取的过程中未进行数据加密和解密处理。

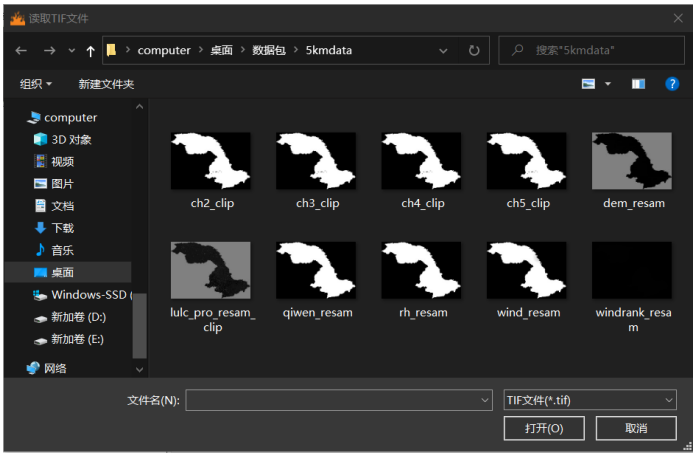
3.3 数据导入与制图部分

该部分实现了数据的导入和制图的功能，主要代码分别存放于 MainForm.cs、SymbolChoice.cs。

图表 9 数据的导入和制图的功能

3.3.1 数据导入

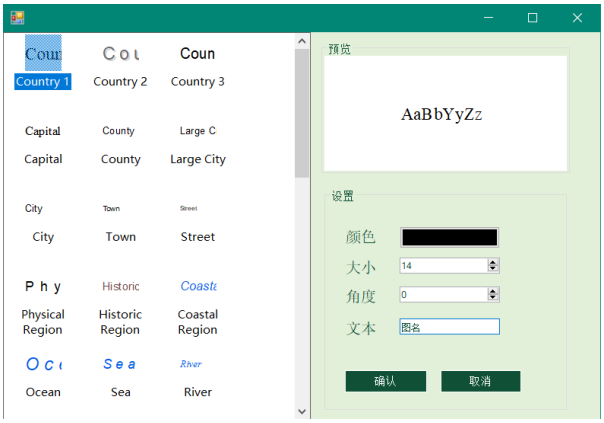
点击【导入栅格】按钮，会弹出文件选择新窗口，选择 tif 文件即可导入成功。



图表 10 添加数据功能

3.3.2 添加图名

样式：用户在弹出的样式选择器内选择文字样式。
颜色：用户点击选择文字颜色
大小：用户设置文字大小，不超过 100
角度：用户设置文字角度
文本：用户设置文字内容
选择后样式预览在右上方。在布局视图内按住鼠标右键拖动可以确定添加的位置，添加到地图后可以选择该制图要素进行移动。



图表 11 添加图名功能

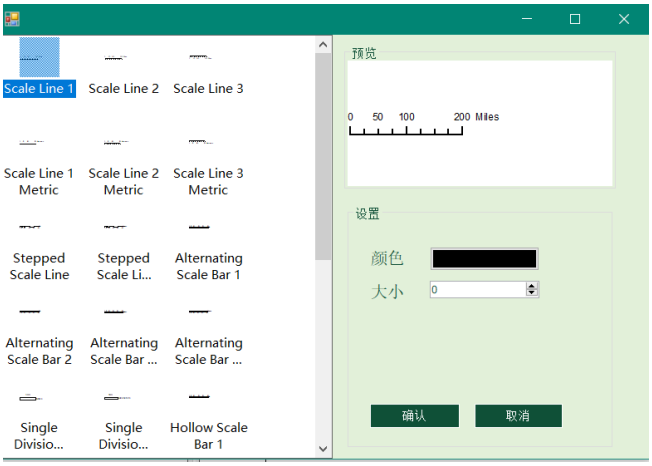
3.3.3 添加图例

点击【添加图例】后直接在在布局视图内按住鼠标右键拖动就可以添加图例，添加到地图后可以选择该制图要素进行移动和大小调整。

3.3.4 添加比例尺

样式：用户在弹出的样式选择器内选择比例尺样式。
颜色：用户点击选择比例尺颜色
大小：用户设置文字比例尺
选择后样式预览在右上方。在布局视图内按住鼠标右键拖动可以确定添加的位置，添加到地

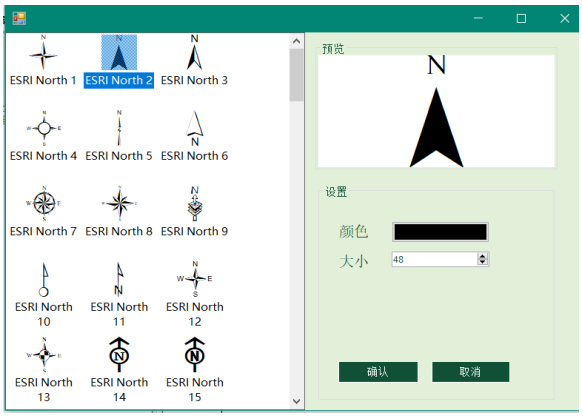
图后可以选择该制图要素进行移动，并调整大小。



图表 12 添加比例尺功能

3.3.5 添加指北针

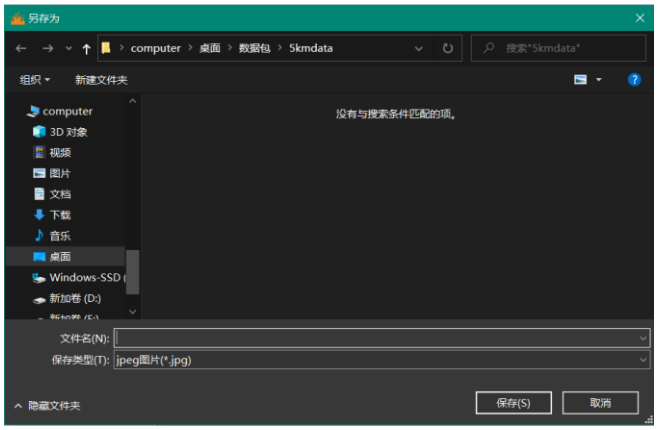
样式：用户在弹出的样式选择器内选择指北针样式。
颜色：用户点击选择指北针颜色
大小：用户设置指北针大小
选择后样式预览在右上方。在布局视图内按住鼠标右键拖动可以确定添加的位置，添加到地图后可以选择该制图要素进行移动，并确定大小。



图表 13 添加指北针功能

3.3.6 地图导出

文件名：用户输入导出地图的文件名，支持数字、中文、英文、部分特殊字符。
格式：用户下拉选择格式格式，可以在 jpg、png、bmp、emf、tif、gif 中任选一种。
选择后样式预览在右上方。在布局视图内按住鼠标右键拖动可以确定添加的位置，添加到地图后可以选择该制图要素进行移动。



图表 14 导出地图功能

**设计优点与痛点

设计优点：

1、实现了样式选择器，可以自由选择制图要素的样式，并调整属性。

2、添加制图要素后可以在图框内自由对其调整，包括移动、放大、缩小、删除等。

设计痛点：

1、 导出地图本质上是图框快照，没有实现高分辨输出。

2、 没有设计导入自定义样式的功能。

3.4 数据浏览部分

该部分实现了数据浏览功能，包括图像在图框里的浏览和数据的基本信息浏览和渲染。主要代码分别存放于 MainForm.cs、 RasterInf.cs、RasterRenderer.cs。

3.4.1 图框图像同步

在数据视图内对图像进行操作，布局视图内会复制整个数据视图内的图像。实现两个图框内容即时、高度的一致性。

3.4.2 图像基本浏览

图像操作工具栏调用了大量实用的图像操作工具，可以实现放大、缩小、展示全图、移动图像、添加元素、选择元素等等功能；

图表 15 图像基本浏览功能

放大：用户可以放大数据视图和布局视图内的图像。

缩小：用户可以缩小数据视图和布局视图内的图像。

展示全图：用户可以在数据视图和布局视图视窗内展示全范围的图像。

移动元素：用户可以在数据视图和布局视图内移动图像。

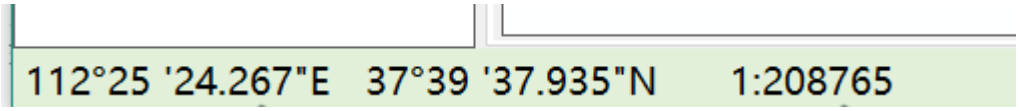
添加元素：用户可以在放大数据视图和布局视图内添加元素。

选择元素：用户可以在放大数据视图和布局视图内选择元素。

移动元素：用户可以在放大数据视图和布局视图内选择元素后进行移动。

3.4.3 显示实时地理位置和比例尺信息

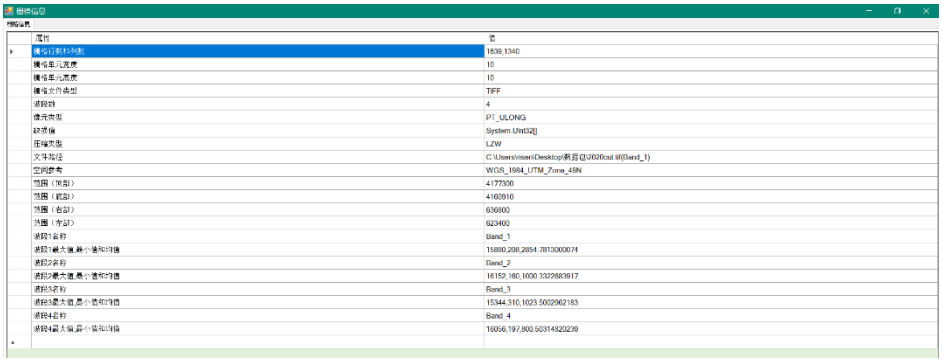
无论是在数据视图还是布局视图，移动鼠标都可以显示当前所在地理位置和比例尺信息。



图表 16 显示实时地理位置和比例尺信息功能

3.4.4 显示栅格属性

在 TOCcontrol 内右击图层后点击【显示属性】后可以浏览该栅格图层的基本属性信息，包括栅格的行列数、栅格的单元像元信息、栅格的地理范围、栅格各波段的统计信息等等。



图表 17 显示栅格属性功能

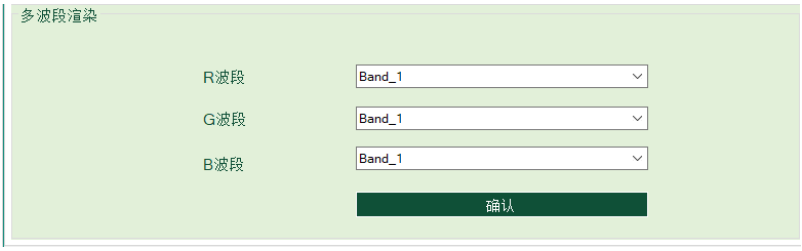
3.4.5 单波段渲染

渲染波段：用户需要选择渲染的波段
渲染方式：用户需要确定渲染方式是拉伸、分级还是唯一值。
分级数目：渲染方式是分级的话，需要确定分级数目，最小为 2，最大为 10。
渲染色带：用户需要选择渲染的色带
点击确认后图窗可以查看渲染结果。

图表 18 单波段渲染功能

3.4.6 多波段渲染

R 波段：用户需要选择渲染为红色的波段。
G 波段：用户需要选择渲染为绿色的波段。
B 波段：用户需要选择渲染为蓝色的波段。
点击确认后图窗可以查看渲染结果。



图表 19 多波段功能

3.4.7 2.3.7 删除图层

在 TOCcontrol 内右击图层后点击【删除图层】后可以删除该栅格图层。

**设计优点与痛点

设计优点：
1、兼顾了多波段数据和单波段数据的可能，可以实现多波段数据对应波段的栅格渲染。
2、栅格信息表信息采集于栅格各接口的不同属性，比较全面。
设计痛点：
1、窗体样式设置存在瑕疵。
2、多波段渲染只实现了 RGB 渲染。

3.5 数据基本处理部分

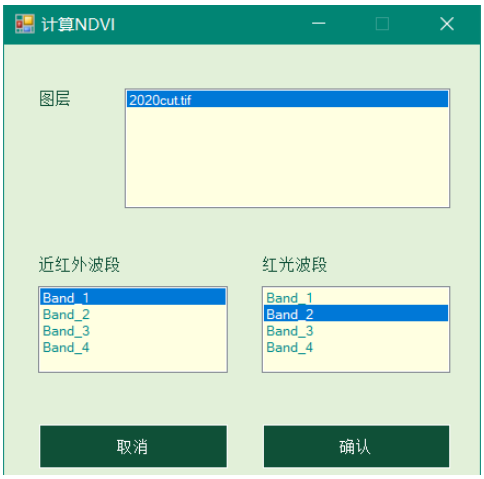
该部分实现了数据基本处理功能，包括栅格计算器和基于其的 NDVI 计算，以及坡度坡向发计算和栅格裁剪。主要代码分别存放于 MainForm.cs、RasterCalculator.cs、SurfaceAnalysis.cs、SurfaceAnalysis2.cs、SelectLayer.cs、RasterManage.cs。

图表 20 数据基本处理功能

3.5.1 NDVI 计算

图层：用户选择操作图层。

近红外波段：用户从操作图层中选择近红外波段。
红光波段：用户从操作图层中选择红光波段。
NDVI 计算结果将以新图层的形式呈现。



图表 21 计算 NDVI 功能

3.5.2 2.4.2 栅格计算器功能

计算公式：用户自定义计算公式。需要是四则运算公式，且包含变量。变量需要是字母开头的字母数字组合。运算公式的符号和变量间需要使用空格隔开。
变量图层：系统会自动识别输入变量，每一个变量需要选择图层。
变量波段：每一个变量需要选择图层后，需要进一步选择该图层的某一波段。
例如，输入计算公式 b1 + b2 为例，系统会识别变量 b1 和 b2，然后需要以此选择它们对应的波段，可以是任一图层的任一波段，然后进行计算获得结果。
栅格计算结果将以新图层的形式呈现。

图表 22 栅格计算器功能

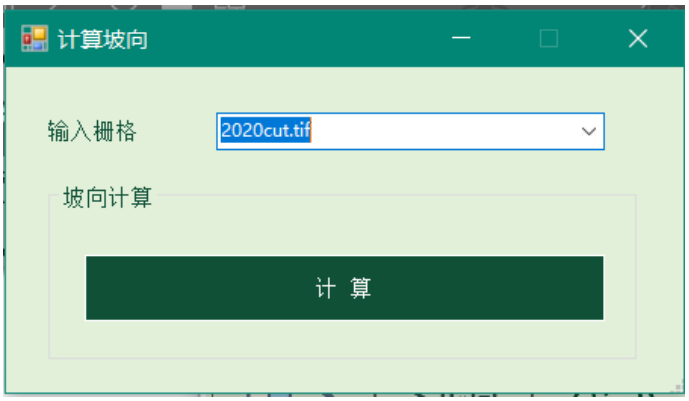
3.5.3 坡度计算

输入栅格：用户确定操作图层。
输出测量单位：确定输入单位的格式是 DEGREE 还是高度增量百分比。
计算结果将以新图层的形式呈现。

图表 23 坡度计算功能

3.5.4 坡向计算

输入栅格：用户确定操作图层。
计算结果将以新图层的形式呈现。

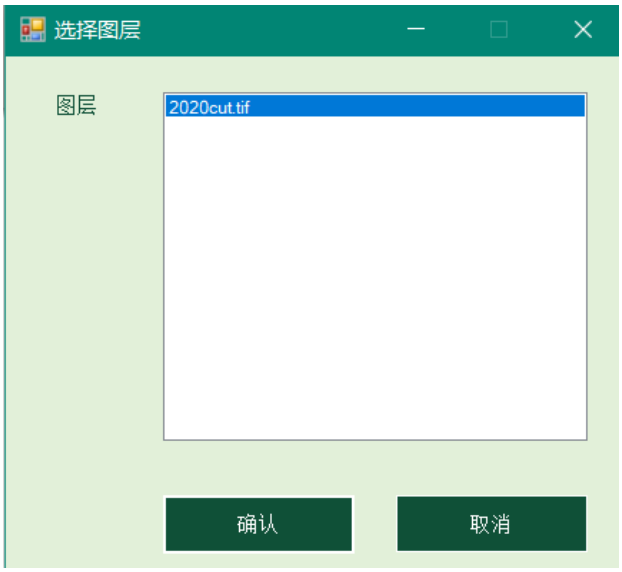


图表 24 坡向计算功能

3.5.5 栅格裁剪

选择图层：选择要裁剪的图层，可以选择多个图层

在数据视图框内绘制一个元素，点击【栅格裁剪】，选择裁剪图层后可以根据该元素范围实现对多选图层的快速裁剪，结果输出为新图层。



图表 25 栅格裁剪功能

3.5.6 **设计优点与痛点

设计优点：

1、栅格计算器利用正则表达式实现变量识别，智能程度提高，用户发挥空间更大。

2、栅格裁剪功能实现了快速的批量处理。

设计痛点：

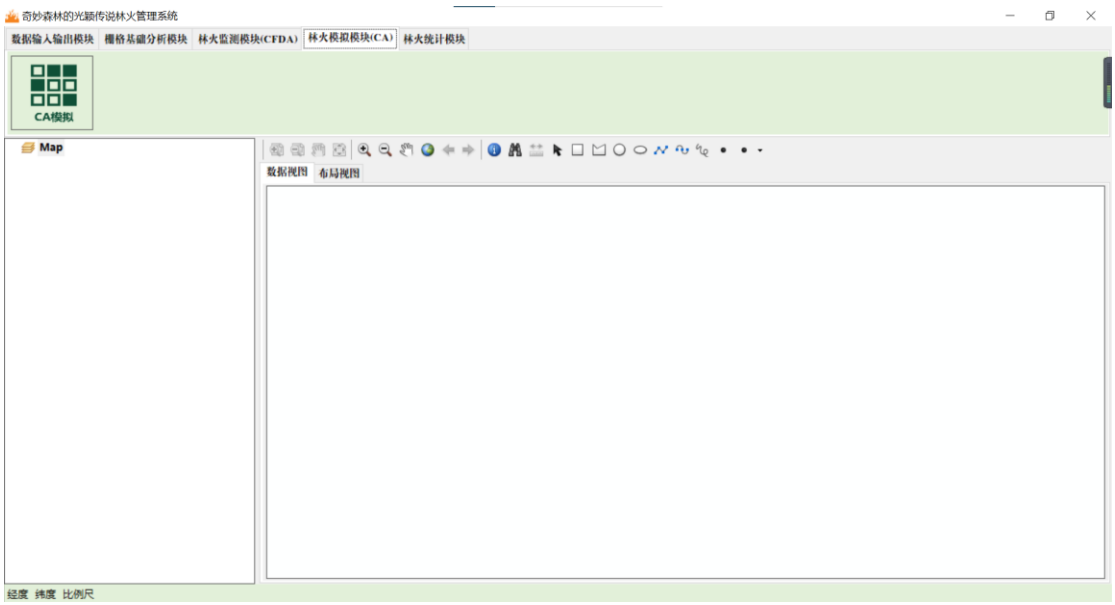
1、功能数目仍有不足。

3.6 林火监测部分

3.7 林火模拟部分

该基于元胞自动机的林火模拟模块，基于.NET 框架，在 VisualStudio2019 中基于 C#语言，参考修正的王正非毛贤敏林火蔓延模型搭建而成，可基于林火监测结果、气象数据、地形数据等，实现林火蔓延的模拟。

3.7.1 模型架构



图表 26 模型架构示意图

该林火模拟模块，目前包含一种基于修正的王正非毛贤敏林火蔓延模型林火蔓延模拟方法，未来有待进一步开发，实现多种林火扩张模型的搭建。

3.7.2 模型启动

1、单击“林火模拟模块”页面，实现功能模块跳转
2、双击“CA 模拟”按钮，
3、弹出图层选择窗口，实现林火模拟功能启动

3.7.2.1 模型界面

林火模拟模型主界面如下：

图表 27 林火模拟界面

主界面包括以下部分：

1、 图层选择。包括林火监测结果、温度、风速、风级、湿度、dem 图层的选择与输入
2、 参数设置。包括风向、空间分辨率、模拟时间的选择与输入
3、 开始运算按钮，用于启动元胞模拟

3.7.2.2 示例数据介绍

在目录下的 testdata 文件夹中，能找到用于模拟大兴安岭地区、黑河市 2006 年 5·21 特大火灾蔓延情况的示例数据，用于模型的操作与运行演示。

一、数据说明

示例数据来源及数据说明如下：

数据类型	数据名称	分辨率	获取时间	数据源
气象数据	日最高温度	0.002248°	2006.5.21	全国温室数据系统 http://data.sheshiyuanyi.com
	日最大湿度	0.002248°	2006.5.21	
	日平均风速	0.002248°	2006.5.21	
	日平均风级	0.002248°	2006.5.21	
	风向	-	-	-
地形数据	DEM 高程数据	0.002248°	-	遥感数据共享 (ceode.ac.cn)
遥感数据	NOAA-AVHRR	0.002248°	2006.5.21	Google Earth Engine 谷歌云平台 NOAA/CDR/AVHRR/SR/V5 数据集

其中，日最高温度、日最大湿度、日平均风速等气象数据，主要利用研究区内气象站点数据，基于 Kriging 插值获取，插值数据的分辨率、大小、行列数，必须保持一致。而日平均风级（0-12）数据，通过对插值后的日平均风速(m/s)数据进行重分类得到，其分级标准参考为：http://blog.sina.com.cn/s/blog_6d3ec1db0101cq0h.html。而 NOAA-AVHRR 遥感数据，主要需要获取 BT_CH3、BT_CH4、BT_CH5、RF_2 波段，主要用于林火的识别，基于 CFDA 算法完成林火的监测。

二、数据用途

1、 对于 NOAA 遥感数据，主要用于
2、 对于日最高温度(℃)、日最大湿度(%)、日平均风级等气象数据，主要用于初始林火蔓延速度的运算。
3、 对于日平均风速(m/s)、风向等气象数据，以及 DEM 高程等地形数据，主要用于 8 方向林火蔓延速度的修正。

三、数据要求

1、 模型所需的林火监测结果数据必须为 01 数据（基于 CFDA 算法监测结果的 01 数据）。
2、 模型所需的输入数据投影、像元大小和行列数需要保持一致，各栅格间完全对齐。
3、 日最高温度、日最大湿度、日平均风速等气象数据，以及 DEM 高程数据，不需要进行数据预处理，但需要保证数据单位与要求一致，即：

数据类型	数据名称	数据描述
气象数据	日最高温度	摄氏度℃
	日最大湿度	百分比%
	日平均风速	m/s 米/秒
	日平均风级	0-12，分级数据

	风向	0-360，角度数据
地形数据	DEM 高程数据	m

案例数据旨在引导模型使用者了解模型的基本操作过程。在实际城市扩展模拟过程中，可以考虑更多的因素的影响。比如：土地利用数据、NDVI 数据等。由于数据获取受限，且模拟区域相对较小，出于大兴安岭地区，地表覆盖相对单一，本模块并未使用其他数据进行林火蔓延速度的修正，在未来的功能扩展中可以进一步拓展、优化。

3.7.2.3 林火模拟模块介绍

林火模拟模块具体操作步骤如下：

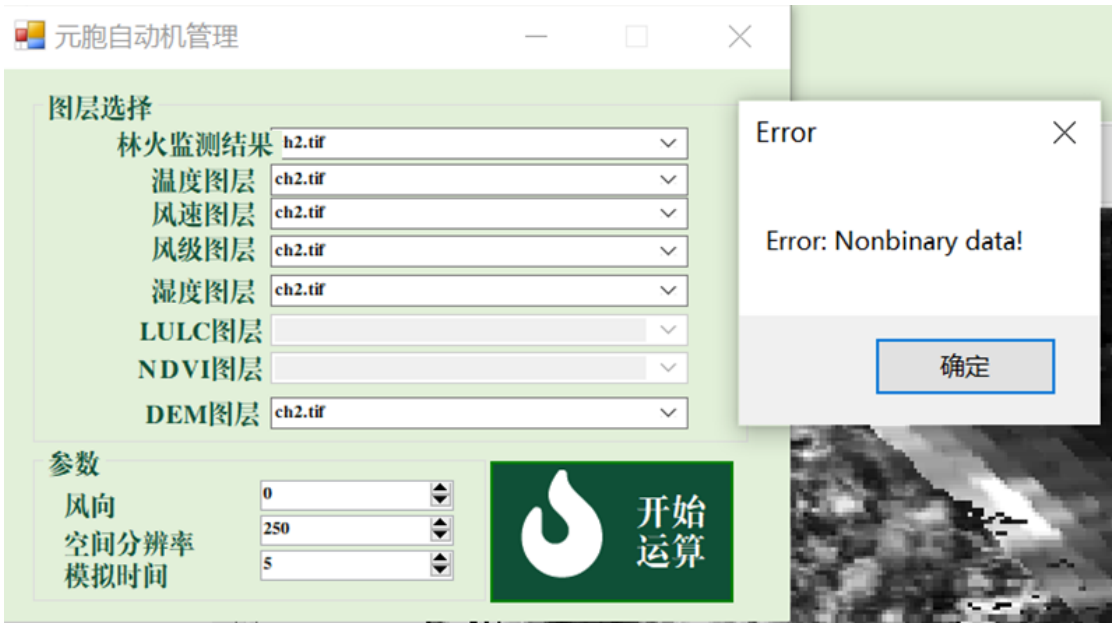
1、 单击“林火模拟模块”页面，实现功能模块跳转
2、 双击“CA 模拟”按钮，弹出图层选择窗口，实现林火模拟功能启动
3、 点击林火监测结果下选框，选择 01 数据格式的林火监测结果
4、 点击温度图层下选框，选择温度插值数据
5、 点击风速图层下选框，选择风速插值数据
6、 点击分级图层下选框，选择风级插值数据
7、 点击湿度图层下选框，选择湿度插值数据
8、 点击 DEM 图层下选框，选择 DEM 数据
9、 输入风向，取值范围为 0-360° 当风向为 0°，表示为南风；当风向为 90°，表示为西风；当风向为 180°，表示为北风；当风向为 270°，表示为东风。
10、输入数据空间分辨率，取值范围为[1,∞]，单位为 m。
11、输入模拟时间，单位为 min。
12、单击开始运算

数据输入结果如下：

图表 28 输出结果

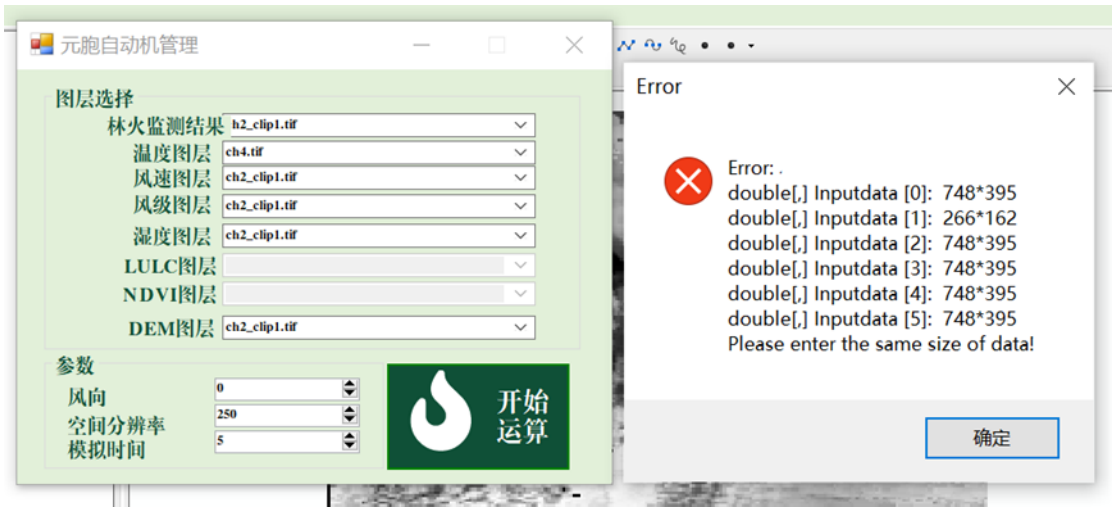
3.7.2.4 模型操作注意事项

1、 必须保证输入的林火监测结果图层为 01 类型数据。当输入非 01 数据时，将出现：



图表 29 报错提示

2、必须保证输入的气象数据图层、地形数据图层、遥感数据图层，具有相同的投影、相同的行列号，同时，必须保证各栅格间完全对齐。当数据行列数不一致时，将出现：



图表 30 报错提示

3、由于该模型主要为经验模型，基于修正的王正非毛敏贤模型而搭建，因此必须保证输入数据单位如上要求，否则不能根据经验公式准确计算林火蔓延速度，从而进行准确模拟。

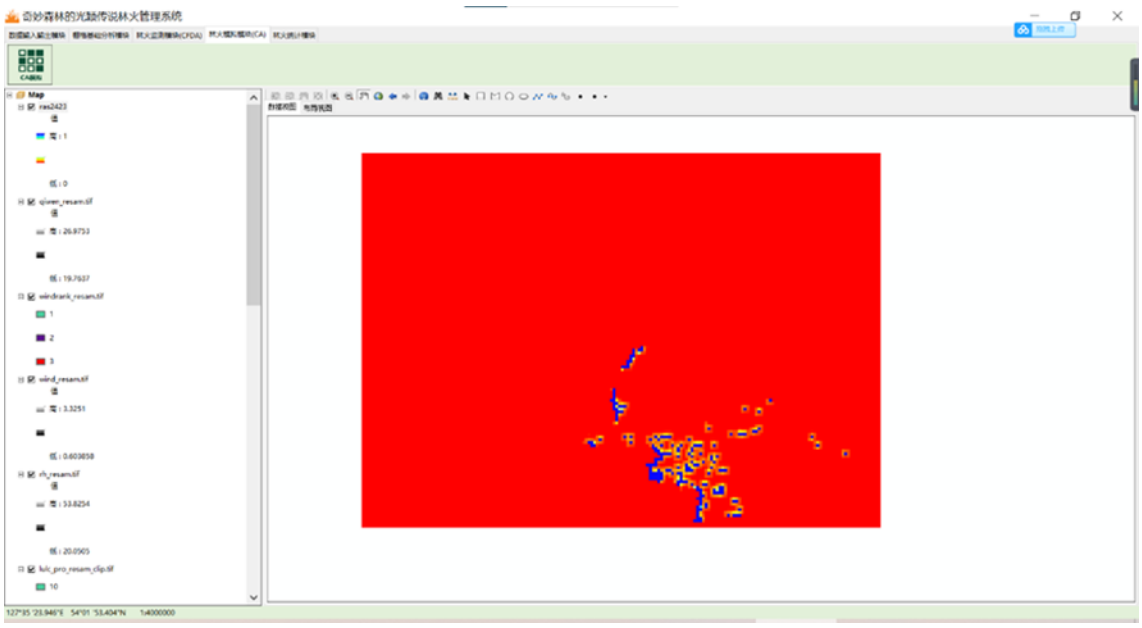
3.7.2.5 预期结果

1、林火模拟结果

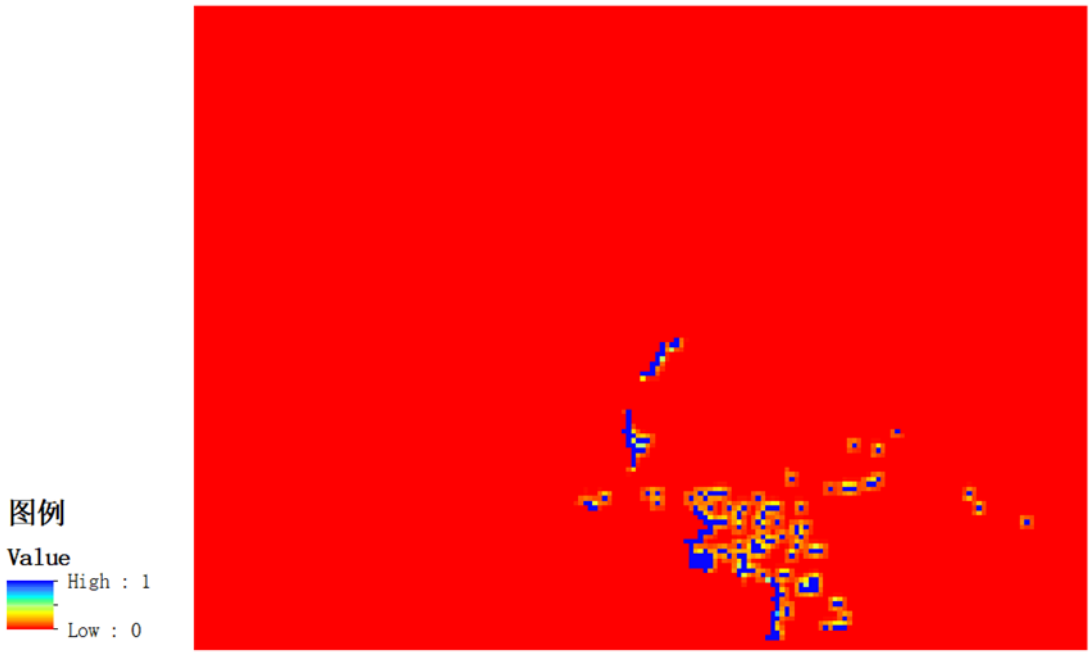
输入参数

参数名称	参数取值
风向	0
空间分辨率	250
模拟时间	5

输出模拟结果：



2、输出林火模拟结果



3.7.3 基本原理

3.7.3.1 元胞基本原理

元胞自动机(CA，cellular automata) 由元胞、状态、邻域和局部规则 4 个部分组成，其形式定义为[1]：

$$CA = (N, S, NC, R)$$

其中，N代表一个规则划分的网络空间，每个网络空间单元就是一个元胞。

S是有限集合，用来表示元胞的状态。

NC表示邻域。

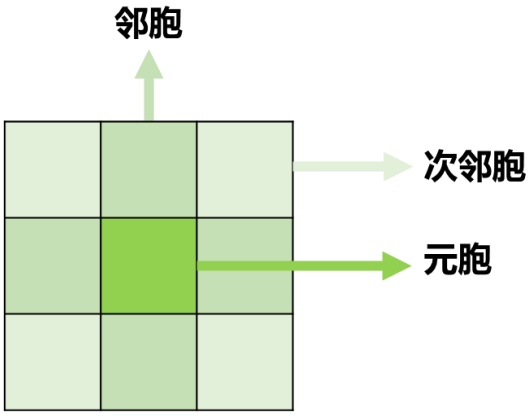
R表示局部规则，即根据 t 时刻某个元胞的所有邻居的状态组合来确定 $t+1$ 时刻该元胞的状态值。因此，转换规则是元胞状态和邻域元胞关系的函数，决定了元胞自动机动态演化的过程和结果。

由此得出，元胞自动机是一种时间和空间都离散的动力系统，系统的整体行为完全靠大量简单的个体行为的总和实现。元胞就是构成系统的基本单元，散布在空间上的一系列元胞根据确定的局部规则在离散的时间维上演化，用于模拟和分析几何空间内的各种现象[2]，具有较强的模拟能力。林火蔓延是林火行为的主要表现形式，是一个多相、多组分可燃物在各种气象条件和地形影响下燃烧和运动极其复杂的物理现象[3]。因此，可以利用元胞机来实现林火蔓延的模拟。

根据我国的林火蔓延特点，以王正非和毛贤敏的林火蔓延模型为基础结合元胞自动机原理进行林火蔓延模拟的研究，将林火蔓延模型进行改进，求解出在地形因子、气象因子影响下的林火蔓延速度公式，最后将其转化为计算机语言，显示林火蔓延的全过程。

3.7.3.2 元胞空间划分

本林火模拟模型采用二维元胞空间的 Moore 型邻域，该领域为 3*3 矩形，一个元胞的邻居包括上、下、左、右 4 个相邻的元胞，以及对角线方向上的 4 个次相邻的元胞。如下图所示，相邻元胞是与中心元胞(i ，j) 有公共边的元胞，分别用$(i-1,j)$，$(i+1,j)$，$(i,j-1)$，$(i,j+1)$表示。次相邻元胞分别用$(i-1,j-1)$，$(i-1,j+1)$，$(i+1,j-1)$，$(i+1,j+1)$表示。



图表 31 元胞空间划分示意图

3.7.3.3 2.8.3.3 元胞状态定义

在本林火模拟模型中，定义元胞状态是一个 0~1 的 double 类型的数据，表明某一时刻燃烧状态。在t时刻元胞(i,j)的状态定义为[4]：

$$a_{i,j}^{t}=\frac{\text{元胞}(i,j)\text{已经燃烧的面积}}{\text{整个元胞}(i,j)\text{面积}}$$

当$a_{i,j}^{t}=1$，表明在t时刻，元胞(i,j)已经完全燃烧
当$a_{i,j}^{t}=0$，表明在t时刻，元胞(i,j)未燃烧
当$0<a_{i,j}^{t}<1$，表明在t时刻，元胞(i,j)部分燃烧

当$a_{i,j}^{t}<0$，表明元胞(i,j)为不可燃烧区域，在本模型中我们定义为背景区域。

3.7.3.4 初始林火蔓延速度

根据经典王正非林火蔓延模型，给出火的初始蔓延速度回归式[5](初始蔓延速度单位为m/min)。该回归式是王正非根据大兴安岭和四川省的数百次火烧实验数据，加上符合预报理论和物理机制分析，再加上根据毛敏贤所作的修正，最终获得。其计算公式如下：

$$R_0 = aT + bV + ch - D$$

$$a = 0.03, b = 0.05, c = 0.0, D = 0.3$$

其中，R_0表示初始林火蔓延速度，

T表示元胞对应的当日温度（℃）

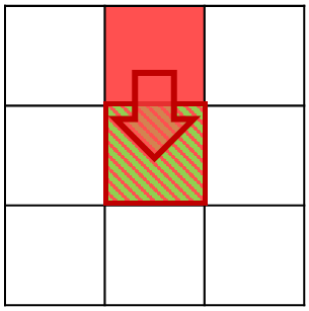
V表示元胞对应的当日中午平均风级（0-12）

h表示为日最小湿度 RH%，

林火蔓延素的含义为：每个元胞都有一个初始蔓延速率R_0，它决定了元胞(i,j)林火蔓延的时间。可以认为，当经过步长时间$T=\frac{L}{R_0}$，元胞(i,j)将从未燃烧状态到完全燃烧状态。

1） **情形 1：**当有且仅有一个邻胞为完全燃烧（如$a_{i-1,j}^{t}=1$），而中心元胞处于未燃烧状态，$a_{i,j}^{t}=0$，

当经过$T=\frac{L}{R_0}$，将实现$a_{i,j}^{t+1}=1$

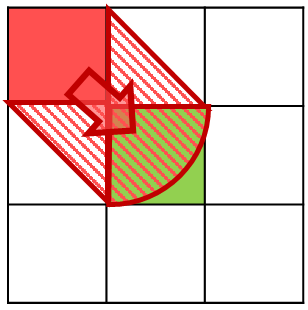


图表 32 邻胞影响示意图

2） **情形 2：**当有且仅有一个次邻胞为完全燃烧（如$a_{i-1,j-1}^{t}=1$），而中心元胞处于未燃烧状态，$a_{i,j}^{t}=0$，

当经过$T=\frac{\sqrt{2}L}{R_0}$，将实现$a_{i,j}^{t+1}=1$

或者经过$T=\frac{L}{R_0}$，将实现$a_{i,j}^{t+1}=\frac{\pi L^2}{4L^2}=\frac{\pi}{4}=0.785$



图表 33 次邻胞影响示意图

3.7.3.5 方向林火蔓延速度及修正

除了根据经验公式，基于湿度、温度、风级等气象因子估算得到的初始林火蔓延速度，还存在其他因子，影响因子的林火蔓延速度。影响林火蔓延速度的因子可以分为 4 类：土壤特性、可燃物类型、气象因子及地形因子。受到土壤特性、可燃物类型的影响，可能导致各栅格林火蔓延速度偏离域初始蔓延速度受到风向、风速等气象因子以及海拔地形因子的影响，可能导致邻胞、次邻胞对元胞的影响发生增强或者减弱

因此，需要基于各影响因子，对一个 3*3 元胞组的 8 方向林火蔓延速度进行修正。王正非提出的林火蔓延速度模型计算方程如下：

$$R = R_0 K_w K_\varphi K_s$$

其中，R表示修正后元胞林火蔓延速度

R_0表示元胞初始蔓延速度

K_w表示风作用系数，根据风向、蔓延方向、风速大小、距离修正蔓延速度

K_φ表示地形坡度调整系数，根据上坡、下坡修正蔓延速度

K_s表示可燃物配置格局系数，根据元胞(i,j)地表覆盖物类别修正蔓延速度修正

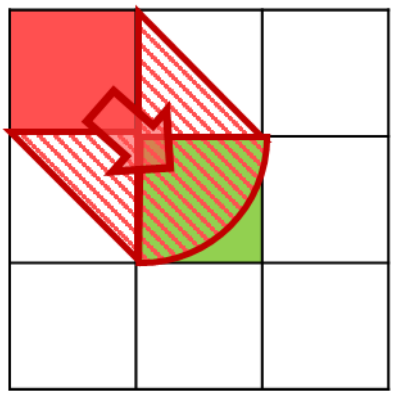
该修正速度的具体含义为：在邻胞、次邻胞状态影响下，对中心元胞(i,j)林火蔓延速度发生改变，影响了中心元胞(i,j)林火蔓延的时间，如对于次邻胞$R^t_{i-1,j-1}$，表示 t 时刻，次邻胞$C_{i-1,j-1}$向元胞$C_{i,j}$产生的林火蔓延的速度，该速度经过修正公式处理。可以认为：

1） **情形 1：**当有且仅有一个次邻胞为完全燃烧（如$a^t_{i-1,j-1}=1$），$a^t_{i,j}=0$

经过$T=\frac{L}{R^t_{i-1,j-1}}$，将实现$a^{t+1}_{i,j}=\frac{\pi L^2}{4L^2}=\frac{\pi}{4}=0.785$

3） **情形 2：**当有且仅有一个次邻胞为部分燃烧（如$a^t_{i-1,j-1}=0.5$），$a^t_{i,j}=0$

经过$T=\frac{L}{R^t_{i-1,j-1}}$，将实现$a^{t+1}_{i,j}=a^t_{i-1,j-1}\times\frac{\pi L^2}{4L^2}=a^t_{i-1,j-1}\times\frac{\pi}{4}=0.5\times0.785$



图表 34 修正正蔓延速度

据此原理，可以进一步得到 **8 方向林火蔓延速度**，邻胞蔓延速度$R^t_{i,j-1}$，$R^t_{i,j+1}$，$R^t_{i+1,j}$，$R^t_{i-1,j}$；次邻胞林火蔓延速度$R^t_{i-1,j-1}$，$R^t_{i-1,j+1}$，$R^t_{i+1,j-1}$，$R^t_{i+1,j\mp1}$。

3.7.3.6 修正系数基本原理

一、K_φ地形坡度调整系数计算原理

在毛敏贤模型中，$K_\varphi=e^{3.533(\tan(\varphi))^{1.2}}$。其中，$\tan(\varphi)$表示林火没拿眼区域上坡方向坡度，值大于 0；下坡方向为$\tan(\varphi)$。而在元胞空间中，任何一个邻胞或者次邻胞(k,l)，都有各自相对中心燃烧元胞(i,j)的$(K_\varphi)_{k,l}$值和坡度值。因此：

2） 邻胞相对中心燃烧元胞的K_φ为：$K_\varphi = e^{3.533(-1)^G\left|\frac{h_{k,l}-h_{i,j}}{L}\right|^{1.2}}$

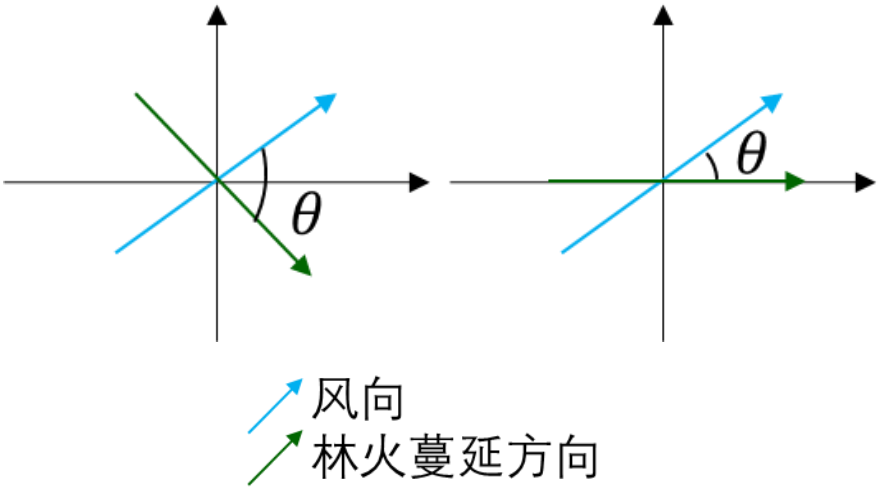
3） 次邻胞相对中心燃烧元胞的K_φ为$K_\varphi = e^{3.533(-1)^G\left|\frac{h_{k,l}-h_{i,j}}{\sqrt{2}L}\right|^{1.2}}$

其中，L 表示栅格大小，即数据空间分辨率；

$h_{k,l}$表示邻胞、次邻胞(k,l)相较于中心元胞(i,j)的相对高度。当$\frac{h_{k,l}-h_{i,j}}{L} > 0$时，$G = 0$，表示上坡对蔓延速度的增强作用；当$\frac{h_{k,l}-h_{i,j}}{L} < 0$时，$G = 1$，表示虾皮对蔓延速度的一直作用。

二、K_w风作用系数计算原理

在毛敏贤模型中，$K_w = e^{0.1783V}$，表示在风方向上的K_w与风速V的关系。当蔓延方向雨凤香间存在夹角时，应当对风按照蔓延方向进行分解。



图表 35 按照蔓延方向风的分解

在标准的 Moore 邻域中，存在 8 个蔓延方向，定义蔓延方向与风向之间的夹角为θ为子蔓延方向向风方向顺时针所夹的夹角。其修正公式为：

$$K_w = e^{0.1783Vcos(\theta)}$$

通过三角函数将风可以分解直 8 个蔓延方向，各蔓延方向具体的计算公式为：

1） 对于邻胞$(i-1,j)$，有：$K_w = e^{0.1783Vcos(\varphi-180)}$

2） 对于邻胞$(i+1,j)$，有：$K_w = e^{0.1783Vcos(\varphi-0)}$

3） 对于邻胞$(i,j-1)$，有：$K_w = e^{0.1783Vcos(\varphi-90)}$

4） 对于邻胞$(i,j+1)$表示，有：$K_w = e^{0.1783Vcos(\varphi-270)}$

5） 对于次邻胞 $(i-1,j-1)$，有：$K_w = e^{0.1783Vcos(\varphi-135)}$

6） 对于次邻胞$(i-1,j+1)$，有：$K_w = e^{0.1783Vcos(\varphi-225)}$

7） 对于次邻胞$(i+1,j-1)$，有：$K_w = e^{0.1783Vcos(\varphi-45)}$

8） 对于次邻胞$(i+1,j+1)$，有：$K_w = e^{0.1783Vcos(\varphi-315)}$

其中，K_w表示对各蔓延方向的修正速度，φ表示风向，取值为 0-360。

但上述风作用系数无法衡量“风”对位于相同方位但距中心元胞距离不同的邻胞的影响。因此，应当进一步引入距离系数。但考虑到本林火模拟模型运算速度较慢，模拟范围较小，风速随距离的变化并不显著，因此在模型中并未考虑距离对于风速的影响，进一步简化模型，提升运算效率。

三、K_s可燃物配置格局系数计算原理

可燃物配置格局系数主要用来表示可燃物的易燃程度(化学特性) 及是否有利于燃烧的配置格局（物理特性）的一个更正系数，在整个燃烧过程中，K_S可以假定为常数。王正非按照野外实地可燃物配置类型，把它予以参数化，如下表所示。

表 可燃物的配置格局更正系数

可燃物类型	更正系数
平铺松针	0.8
枯枝落叶	1.2
茅草杂草	1.6
莎草杂草	1.8
牧场草原	2.0
红松、华山松等林地	1.0

由于数据获取有限，无法获得相关地表覆盖数据，同时考虑到模拟范围较小，因此对林火模拟模型进一步简化，不考虑采用可燃物配置格局系数对林火蔓延速度进行修正。但本模型提供了利用土地利用数据以及 NDVI 数据图层的输入窗口，未来可以进一步考虑如何采用土地利用数据以及 NDVI 数据对林火蔓延速度进行修正。

3.7.3.7 元胞状态转换机制

元胞自动机演化规则是局部的，对指定元胞的状态进行更新时只需要知道其邻胞的状态。对于通常状态下的元胞，不考虑特殊附加属性的影响，它在 $t+1$ 时刻的状态由元胞 t 时刻自身的状态以及 t 时刻元胞周围 4 个邻胞和 4 个次邻胞状态共同决定。

有：

元胞(i,j)在 $t+1$ 时刻的燃烧状态$a_{i,j}^{t+1}$是由其邻域元胞在t时刻向其蔓延的速度$R_{k,l}^{t}$和元胞(i,j)在 t 时刻的燃烧状态$a_{i,j}^{t}$共同决定的[6]

$$a_{i,j}^{t+1}=f(a_{i,j}^{t},R_{i-1,j-1}^{t}+R_{i-1,j}^{t}+R_{i-1,j+1}^{t}+R_{i,j-1}^{t}+R_{i,j+1}^{t}+R_{i+1,j}^{t}+R_{i+1,j-1}^{t}+R_{i+1,j+1}^{t})$$

具体表达式为：

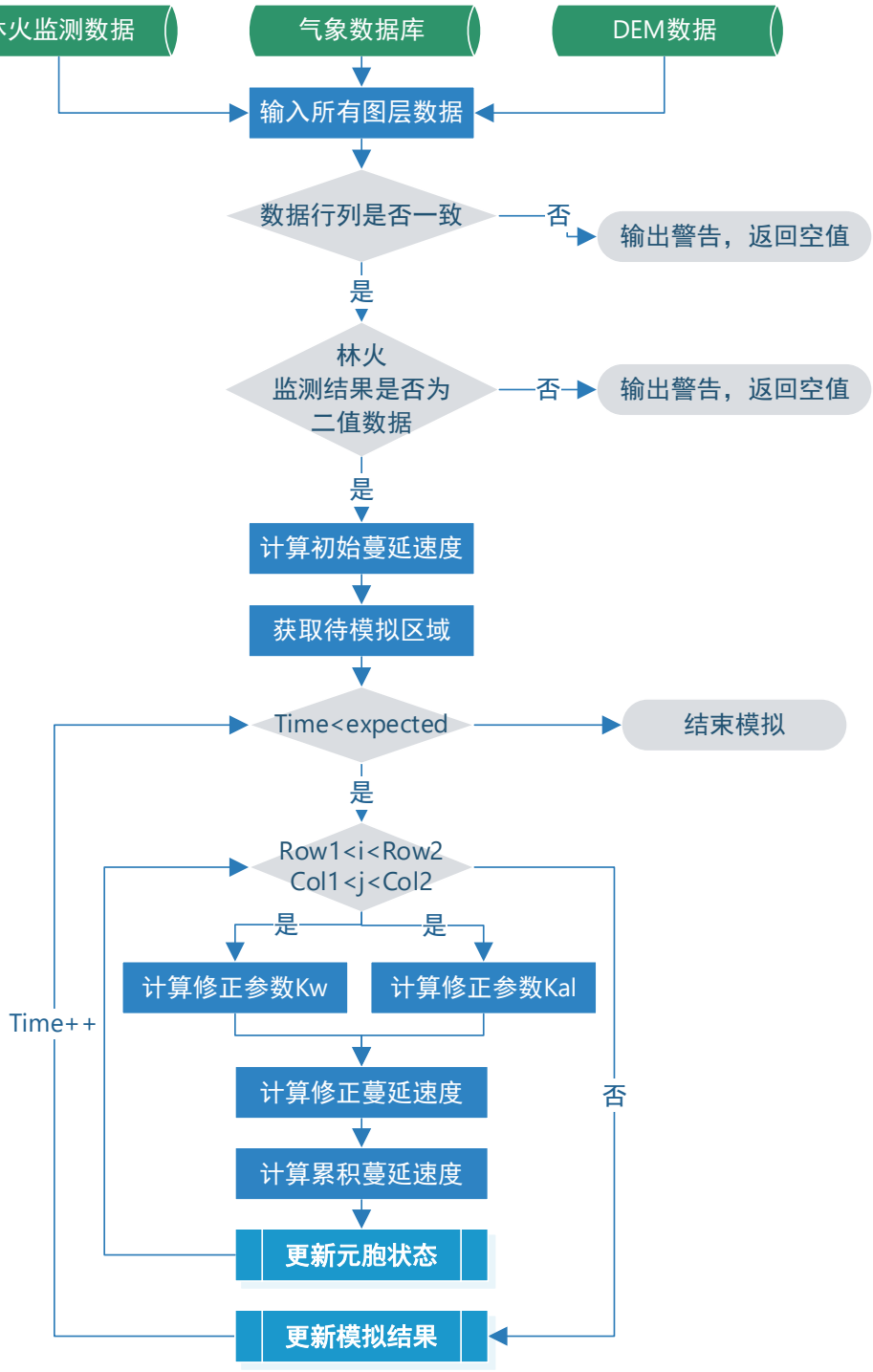
$$a_{i,j}^{t+1}=a_{i,j}^{t}+\left(a_{i-1,j-1}^{t}R_{i-1,j-1}^{t}+a_{i-1,j}^{t}R_{i-1,j}^{t}+a_{i-1,j+1}^{t}R_{i-1,j+1}^{t}+a_{i,j-1}^{t}R_{i,j-1}^{t}\right)/L$$

$$+0.785\times(a_{i,j+1}^{t}R_{i,j+1}^{t}+a_{i+1,j}^{t}R_{i+1,j}^{t}+a_{i+1,j-1}^{t}R_{i+1,j-1}^{t}+a_{i+1,j+1}^{t}R_{i+1,j+1}^{t})/\sqrt{L}$$

核心模型：$a_{i,j}^{t+1}=a_{i,j}^{t}+\sum_{m=1}^{4}a_{m}^{t}R_{m}^{t}/L+0.785\times\sum_{n=1}^{4}a_{n}^{t}R_{n}^{t}/\sqrt{L}$
其中，下标 m 表示邻胞，下标 n 表示次邻胞

该转换公式的含义为：t时刻未燃烧的或者即将熄灭的元胞的$a_{k,l}^{t}=0$，因此，它们对$t+1$时刻元胞(i,j)的状态没有影响，只有完全燃烧或正在燃烧的的元胞才会有向各个方向蔓延的速度。即：元胞在 t+1 时刻的状态=邻胞状态*邻胞对元胞的蔓延速度+元胞状态邻胞状态*邻胞对元胞的蔓延速度即为元胞在 t~t+1 时刻发生的变化。当邻胞状态=1，说明可以完全按照蔓延速度进行扩张；当元胞状态=0，说明邻域不对其产生影响。

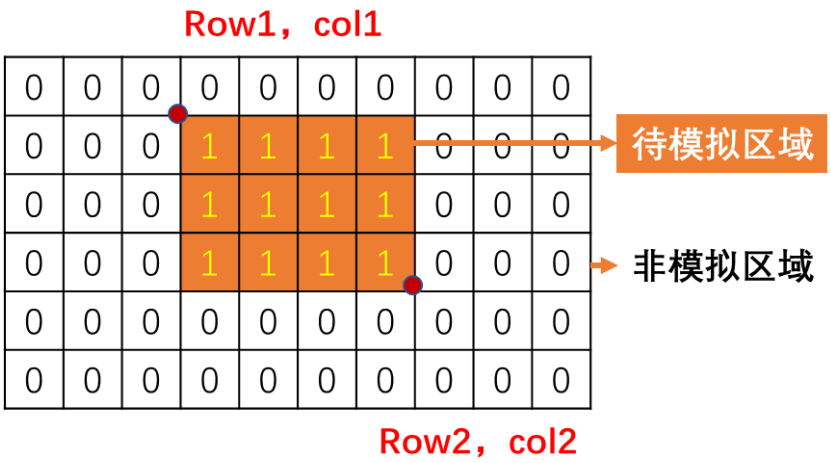
3.7.4 技术路线



图表 36 技术路线

3.7.5 算法优化

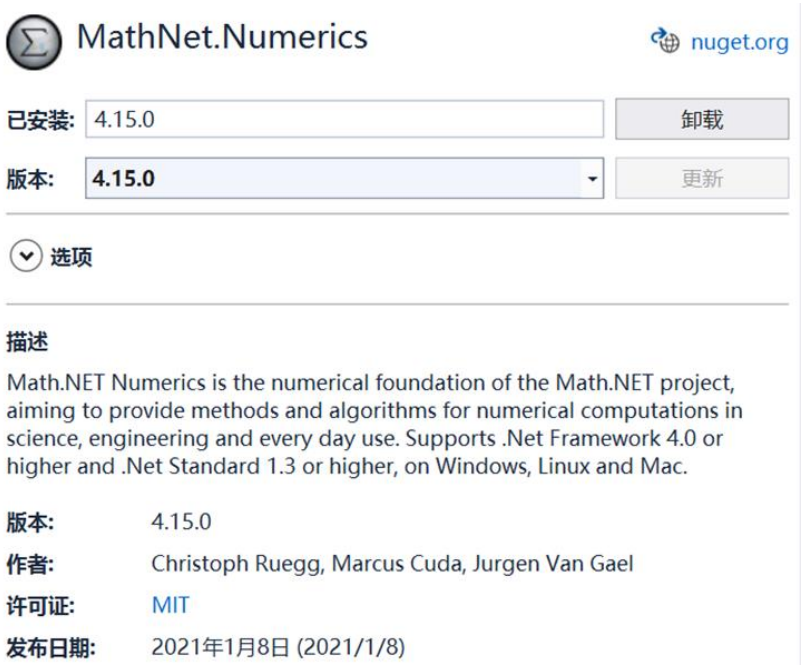
1、 算法优化



图表 37 循环中的算法优化

当存在数据掩膜时，根据掩膜范围确定 for 循环具体范围，避免从 0~最大行、列的无效循环，造成模拟速度过慢。利用该方法能够显著提升运算速度，高效完成林火模拟。

2、 工具优化



图表 38 利用开源包简化代码

为提升运算速率，本模型使用开源包，简化数组运算代码过程：利用开源 Math.net，可以实现数组转矩阵，并利用矩阵实现快速的点乘运算，减少循环，简化代码，加快运算效率。

3.7.6 模型缺陷

1、 模型比较简单，耦合多个经验公式而成，不具有广泛适用性及可拓展性

2、 数据获取受限，难以对模型进行参数校正及精度验证，模型有效性有待进一步考证

3、 该模拟模型对于输入数据具有较高要求，数据预处理难度高，这也是众多元胞自动机模型均存在的痛点。未来希望能扩展该林火监测 GIS 系统功能，实现数据预处理自动化，高效、实时完成林火的监测模拟功能。

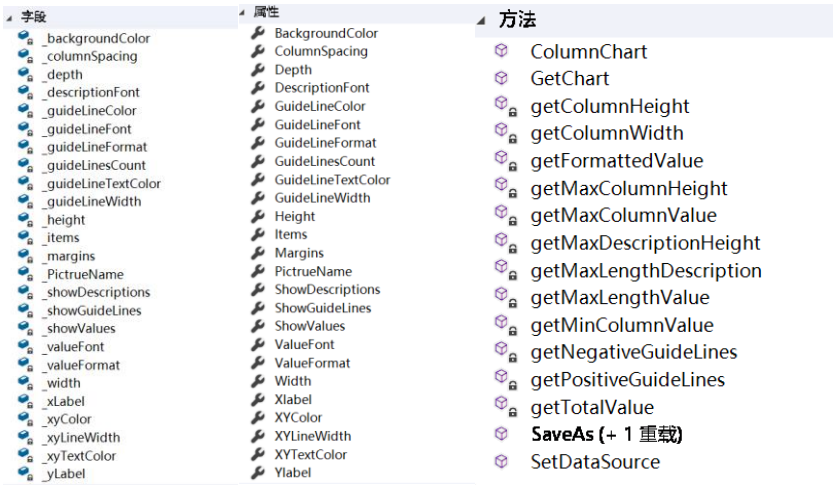
3.7.7 模型使用说明

1、 本模拟模型核心算法以函数、变量的形式存储在该类库中，完全独立于窗体，具有广泛的可扩展性。
2、 并且对于相关函数，输入输出参数、输出结果的数据类型都为更广泛使用的 double[,]类型，即二维数组，而不是 DenseMatrix，具有更广阔的使用空间，能够被直接应用。
3、 在使用时，一方面可以直接添加类库，另一方面可以添加方法类的已有项，满足不同的设计需求

3.8 火灾预测信息可视化部分

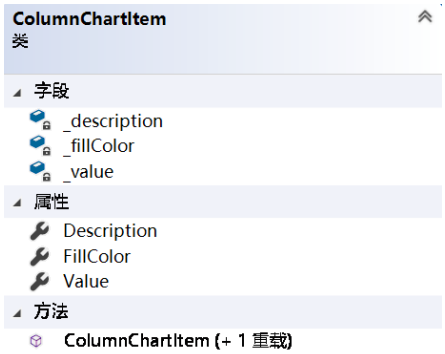
该部分实现了对于 CA 算法中不同模拟时间长度下的森林过火面积实时监测，其对应窗体为 VisualForm.cs，折线图和饼状图的绘制调用了 Chart 图标控件，3D 立体柱状图的绘制调用了自设计的柱状图构造类 ColumnChart.cs 和 3D 柱状图绘制函数类 ColumnChartItem.cs，专门用于实现自定义的柱状图图表内容绘制。

对于 ColumnChart.cs 类，其内容包含：



图表 39　ColumnChart.cs 类内容

对于 ColumnChartItem.cs，其内容包括：



图表 9 ColumnChartItem.cs 类内容

3.8.1 绘制柱状图

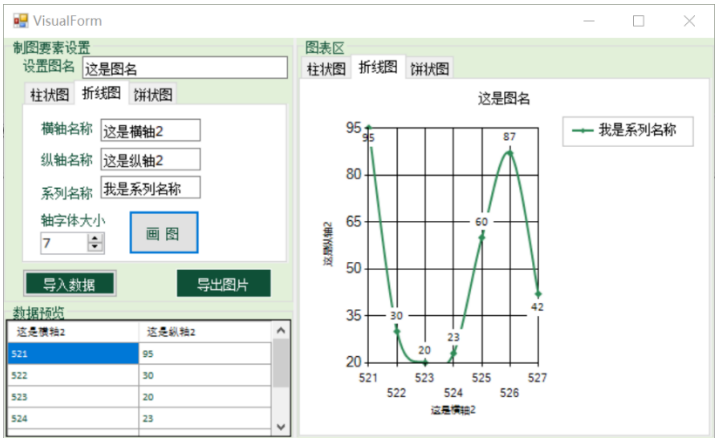
导入数据：在 CA 元胞自动机模拟后，将模拟过程中产生的“时间-过火面积”序列数据导入可

视化模块中。
在柱状图制图要素设置内容中：
设置图名：输入柱状图的图名，是必填项。
横轴名称：输入柱状图的横轴名称，是必填项。
纵轴名称：输入柱状图的纵轴名称，是必填项。纵轴名称和横轴名称不可以相同，若相同则无法成功绘制柱状图。
显示横坐标：可选项，勾选后输出内容框中出现横坐标。
显示数据标签：可选项，勾选后输出内容框中出现数据标签。
显示纵坐标：可选项，勾选后输出内容框中出现纵坐标。
导出图片：能够以*.png 的数据存储格式导出该柱状图。
点击“画图”按钮则在右侧图表输出框中输出图表模样并在数据预览窗口查看制图数据。

图表 10 柱状图绘制界面

3.8.2 绘制折线图

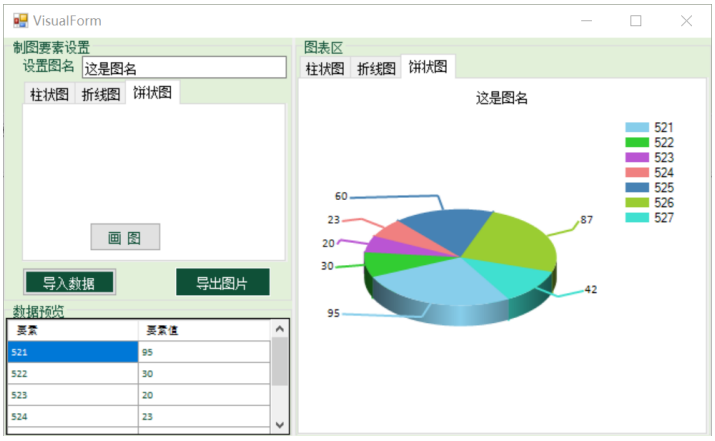
在折线图制图要素设置内容中：
设置图名：输入折线图的图名，是必填项。
横轴名称：输入折线图的横轴名称，是必填项。
纵轴名称：输入折线图的纵轴名称，是必填项。纵轴名称和横轴名称不可以相同，若相同则无法成功绘制折线图。
轴字体大小：可用于调节 Chart 控件中可视化折线图图表的轴标签字体的大小。
点击“画图”按钮则在右侧图表输出框中输出图表模样并在数据预览窗口查看制图数据。



图表 11 折线图绘制界面

3.8.3 绘制饼状图

在饼状图制图要素设置内容中：
设置图名：输入折线图的图名，是必填项。
点击“画图”按钮则在右侧图表输出框中输出图表模样并在数据预览窗口查看制图数据。



图表 12 饼状图绘制界面

**设计优点与痛点

设计优点：
1、柱状图绘制过程中实现了对于制图要素的交互控制。
2、能够浏览详细的制图数据，且保留统一小数单位的数据格式
设计痛点：
1、柱状图导出图片时图片分辨率略低。

附录-1 ARCGIS ENGINE 相关程序代码实现简述

附录 1.1 数据导入代码实现

附 1.1.1 导入功能：用户可以实现遥感数据的导入（包括一次性导入多项数据）

首先是确定工作空间，然后设置 OpenFileDialog。

```
#region 读取栅格
1 个引用
private void pictureBox1_Click(object sender, EventArgs e)
{
    //打开栅格
    IRasterWorkspace workspace;
    IWorkspaceFactory wFactory = new RasterWorkspaceFactoryClass();
    //选定工作空间和文件
    OpenFileDialog of = new OpenFileDialog();
    of.Filter = "TIF文件(*.tif)|*.tif";
    of.Title = "读取TIF文件";
    of.Multiselect = true;
```

然后创建字符串数组存入所有选择路径，并拆分出母路径和文件名，母路径成为栅格工作空间，在此基础上打开栅格文件。

```
if (of.ShowDialog() == DialogResult.OK)
{
    string[] files = of.FileNames;
    foreach(string file in files)
    {
        int index = file.LastIndexOf("\\");
        string path = file.Substring(0, index);
        string name = file.Substring(index + 1, file.Length - index - 1);
        workspace = (IRasterWorkspace)wFactory.OpenFromFile(path, 0);
        IRasterDataset rasterDS = workspace.OpenRasterDataset(name);
```

最后 IRasterDataset 转化为 IrasterDataset2 和 IRaster，通过 CreateFullRaster 方法创建 IRasterLayer，添加到图层中去。

```
if (rasterDS != null)
{
    //栅格波段数
    IRasterBandCollection rasterBandCollection = rasterDS as IRasterBandCollection;
    //栅格图层创建
    IRasterDataset2 rasterDataset2 = rasterDS as IRasterDataset2;
    IRaster raster = rasterDataset2.CreateFullRaster();
    IRasterLayer pRasterLayer = new RasterLayer();
    pRasterLayer.CreateFromRaster(raster);
    //添加图层
    axMapControl1.AddLayer(pRasterLayer);
    axPageLayoutControl1.ActiveView.Refresh();
    axMapControl1.Refresh();
```

附录 1.2 数据浏览代码实现

附 1.2.1 图框基本操作：包括放大、缩小、展示全图、选择元素、添加元素等

导入相关工具到 axToolBarControl

```
//ToolBar工具设置
axToolbarControl1.AddItem("esriControls.ControlsPageZoomInTool", -1, -1,
    true, 0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsPageZoomOutTool", -1, -1,
    false, 0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsPagePanTool", -1, -1, false,
    0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsPageZoomWholePageCommand", -1,
    -1, false, 0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapZoomInTool", -1, -1, true,
    0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapZoomOutTool", -1, -1,
    false, 0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapPanTool", -1, -1, false,
    0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapFullExtentCommand", -1, -
    1, false, 0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapZoomToLastExtentBackCommand",
    -1, -1, false, 0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapZoomToLastExtentForwardCommand",
    -1, -1, false,0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapIdentifyTool", -1, -1,
    true, 0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapFindCommand", -1, -1,
    false, 0, esriCommandStyles.esriCommandStyleIconOnly);
axToolbarControl1.AddItem("esriControls.ControlsMapMeasureTool", -1, -1,
    false, 0, esriCommandStyles.esriCommandStyleIconOnly);
ControlsGraphicElementToolbar command = new ControlsGraphicElementToolbar();
axToolbarControl1.AddItem(command, -1, -1,
    true, 0, esriCommandStyles.esriCommandStyleIconOnly);
```

关联 TOCcontrol 和 ToolBarControl 到 MapControl

```
//关联控件
axTOCControl1.SetBuddyControl(axMapControl1);
axToolbarControl1.SetBuddyControl(axMapControl1);
```

附 1.2.2 图框同步：更改 MapControl 显示 PageLayoutControl 同样会复制

创建一个复制函数，生成一个 IObjectCopy 对象复制 MapControl 到 PageLayoutControl 的 FocusMap 里

```
8 个引用
private void copy(AxMapControl axMapControl1,AxPageLayoutControl axPageLayoutControl1)
{
    IObjectCopy pobjectcopy = new ObjectCopyClass();
    object from = pobjectcopy.Copy(axMapControl1.Map);
    object to = axPageLayoutControl1.ActiveView.FocusMap;
    pobjectcopy.Overwrite(from, ref to);
    axPageLayoutControl1.ActiveView.Refresh();
}
```

在 MapControl 边界变化、内容变化时调用该函数即可。

附 1.2.3 基本信息：用户可以获得遥感图像的基本信息：

（1）遥感图像操作媒介：设置 TOCControl，右击图层名弹出窗口
设计 TOCconrol 的点击事件，通过 HitTest 捕捉所点击的图层，并在点击原地建立 contextMenuStrip，点击该菜单的【基本信息】选项可以将图层传入构建信息函数

```
#region 同步两个窗口
1 个引用
private void axMapControl1_OnExtentUpdated_1(object sender, IMapControlEvents2_OnExtentUpdatedEvent e)
{
    copy(axMapControl1, axPageLayoutControl1);

}


1 个引用
private void axMapControl1_OnMapReplaced(object sender, IMapControlEvents2_OnMapReplacedEvent e)
{
    copy(axMapControl1, axPageLayoutControl1);
}
```

```
#region 与TOCControl交互（以及右键菜单）
1 个引用
private void axTOCControl1_OnMouseDown(object sender, ITOCControlEvents_OnMouseDownEvent e)
{
    //TOCcontrol的所有要素声明
    esriTOCControlItem toccItem = esriTOCControlItem.esriTOCControlItemNone;
    ILayer iLayer = null;
    IBasicMap iBasicMap = null;
    object unk = null;
    object data = null;
    if (e.button == 2)
    {
        axTOCControl1.HitTest(e.x, e.y, ref toccItem, ref iBasicMap, ref iLayer, ref unk, ref data);
        //判断鼠标单击的事件源是否为图层（也可能是数据框或空白）
        if (toccItem == esriTOCControlItem.esriTOCControlItemLayer)
        {
            if (iLayer is IRasterLayer)
            {
                HitLayer = iLayer;
                this.TOCcontextMenuStrip1.Show(axTOCControl1, e.x, e.y);
            }
        }
    }

}
```

（2）遥感图像信息获取：后台获取遥感文件基本信息，通过建立信息获取函数获得遥感文件基本信息，并返回一个 DataTable。
首先是根据传入图层获得 IRaster、IGeoDataset、IRasterProps 等各种接口的获得遥感信息的数据。并创建一个空的 DataTable。

```
#region 构造栅格信息表

1 个引用
private DataTable FL_BuildTable(RasterLayer layer)
{
    //创建DataTable，以图层名命名
    DataTable Table = new DataTable(layer.Name);
    //获取图层的栅格信息
    IRaster raster = layer.Raster;
    IGeoDataset geoDS = raster as IGeoDataset;
    IRasterProps rasterProps = (IRasterProps)raster;
    IRasterBandCollection rasterBandCollection = (IRasterBandCollection)raster;
    IRasterBand rasterBand = rasterBandCollection.Item(0);
    IRasterDataset rasterDS = rasterBand as IRasterDataset;
    //创建DataTable
    DataColumn namefield = new DataColumn();
    namefield.ColumnName = "属性";
    namefield.DataType = System.Type.GetType("System.String");
    DataColumn valuefield = new DataColumn();
    valuefield.ColumnName = "值";
    valuefield.DataType = System.Type.GetType("System.String");
    Table.Columns.Add(namefield);
    Table.Columns.Add(valuefield);
```

然后根据要获取的属性遍历填充这个 DataTable。包括根据 IRasterProps 获得的栅格行列数、单元宽度、单元高度、像元类型、缺损值、栅格范围；根据 IRasterDataset 获得的栅格文件类型、压缩类型、文件路径；根据 IGeoDataset 获得的空间参考；根据 IRasterBandCollection 获得的波段数；根据 IRasterBand 和 IRasterStatistics 获得的栅格各波段统计值。

```
  //填充DataTable
  Table = AddRowValue(Table, "栅格行数和列数", rasterProps.Height + "," + rasterProps.Width);
  Table = AddRowValue(Table, "栅格单元宽度", rasterProps.MeanCellSize().X.ToString());
  Table = AddRowValue(Table, "栅格单元高度", rasterProps.MeanCellSize().Y.ToString());
  Table = AddRowValue(Table, "栅格文件类型", rasterDS.Format);
  Table = AddRowValue(Table, "波段数", rasterBandCollection.Count.ToString());
  Table = AddRowValue(Table, "像元类型", rasterProps.PixelType.ToString());
  Table = AddRowValue(Table, "缺损值", rasterProps.NoDataValue.ToString());
  Table = AddRowValue(Table, "压缩类型", rasterDS.CompressionType);
  Table = AddRowValue(Table, "文件路径", rasterDS.CompleteName);
  Table = AddRowValue(Table, "空间参考", geoDS.SpatialReference.Name);
  Table = AddRowValue(Table, "范围（顶部）",rasterProps.Extent.YMax.ToString());
  Table = AddRowValue(Table, "范围（底部）", rasterProps.Extent.YMin.ToString());
  Table = AddRowValue(Table, "范围（右部）", rasterProps.Extent.XMax.ToString());
  Table = AddRowValue(Table, "范围（左部）", rasterProps.Extent.XMin.ToString());
  for (int i = 0; i < rasterBandCollection.Count; i++)
  {
      int number = i + 1;
      IRasterBand rBand = rasterBandCollection.Item(i);
      IRasterStatistics sta = rBand.Statistics;
      //bool hasStatistics;
      //rasterBand.HasStatistics(out hasStatistics);
      if (sta==null)
      {
          //计算栅格统计值
          rBand.ComputeStatsAndHist();
      }
      sta = rBand.Statistics;

      Table = AddRowValue(Table, "波段"+number+"名称", rasterBandCollection.Item(i).Bandname);
      Table = AddRowValue(Table, "波段" + number + "最大值,最小值和均值", sta.Maximum+","+ sta.Minimum+","+sta.Mean);
  }
  //返回表
  return Table;
```

（3）遥感图像信息呈现：在窗口展示获得的信息。新建立一个窗口，生成一个 DataGridView，以传入的 DataTable 为数据源即可。

```
4 个引用
public partial class RasterInf : Form
{
    1 个引用
    public RasterInf(DataTable FT)
    {
        InitializeComponent();
        this.dataGridView1.DataSource = FT;
    }

}
```

附 1.2.4 删除图层：用户可以右键点击并删除一个图层。

首先仍然是设计 TOCconrol 的点击事件，通过 HitTest 捕捉所点击的图层，并在点击原地建立 contextMenuStrip，点击该菜单的【删除图层】选项可以删除图层。
确定了操作图层后遍历 MapContorl 的所有图层，找到对应图层移除即可。并将 MapControl 的状态复制到 PageLayoutControl 里面。

```
1 个引用
private void 删除图层ToolStripMenuItem_Click(object sender, EventArgs e)
{
    for (int i = 0; i < axMapControl1.LayerCount; i++)
    {
        if (axMapControl1.get_Layer(i) == HitLayer)
        {
            axMapControl1.DeleteLayer(i);
            copy(axMapControl1, axPageLayoutControl1);
            axTOCControl1.Update();
        }
    }

}
```

附 1.2.5 单色渲染：用户可以选定某一波段进行灰度值渲染。渲染分为拉伸渲染、分级渲染、唯一值渲染三种类型。

（1）遥感图像操作媒介：设置 TOCControl。右击图层名弹出窗口，点击栅格【渲染】,传入图层到新窗口中。具体原来上述功能已经涉及，不再赘述。
（2）遥感图像渲染窗口：设置点击 TOCControl 后的板块，提供渲染功能
（3）选择渲染的波段：导入图层到新窗口后，需要确定我们要渲染的波段。
我们需要先通过 IRasterBandCollection 确定波段数目，然后轮流将各波段填入 combobox 中以待用户选择。根据选择的索引确定操作的波段。

```
5 个引用
public partial class RasterRender : Form
{
    RasterLayer iLayer;
    ISymbologyStyleClass pSymbologyStyleClass;
    IRasterBandCollection rasterBandCollection;
    1 个引用
    public RasterRender(RasterLayer iLayer)
    {
        InitializeComponent();
        InitSymbologyControl();
        InitColorRampCombobox();
        this.iLayer = iLayer;
        rasterBandCollection = (IRasterBandCollection)iLayer.Raster;
        //将栅格的各波段名字预设combobox中以供选择
        for (int i = 0; i < rasterBandCollection.Count; i++)
        {
            cb_band.Items.Add(rasterBandCollection.Item(i).Bandname);
            RBox.Items.Add(rasterBandCollection.Item(i).Bandname);
            GBox.Items.Add(rasterBandCollection.Item(i).Bandname);
            BBox.Items.Add(rasterBandCollection.Item(i).Bandname);
        }
        this.cb_band.SelectedIndex = 0;
        this.RBox.SelectedIndex = 0;
        this.GBox.SelectedIndex = 0;
        this.BBox.SelectedIndex = 0;
        this.cb_method.SelectedIndex = 0;
        this.numericUpDown1.Visible = false;

    }
```

（4）选择渲染的色带：渲染后的颜色分布依据确定。

这里我直接导入 ESRI 样式库的色带：构建 SymbologyControl,读取 ESRI 色带样式库并另存。

```
// 初始化色带下拉框
1 个引用
private void InitColorRampCombobox()
{
    this.cb_colormap.DrawMode = DrawMode.OwnerDrawFixed;
    this.cb_colormap.DropDownStyle = ComboBoxStyle.DropDownList;
    //绘制色带
    for (int i = 0; i < pSymbologyStyleClass.ItemCount; i++)
    {
        //从另存中读取色带
        IStyleGalleryItem pStyleGalleryItem = pSymbologyStyleClass.GetItem(i);
        IPictureDisp pPictureDisp = pSymbologyStyleClass.PreviewItem(pStyleGalleryItem, cb_colormap.Width, cb_colormap.Height);
        Image image = Image.FromHbitmap(new IntPtr(pPictureDisp.Handle));
        cb_colormap.Items.Add(image);
    }
    cb_colormap.SelectedIndex = 20;
}
```

```
// 初始化符号库
1 个引用
private void InitSymbologyControl()
{
    //读取ESRI样式库
    string sInstall = ESRI.ArcGIS.RuntimeManager.ActiveRuntime.Path;
    axSymbologyControl1.LoadStyleFile(sInstall + "\\Styles\\ESRI.ServerStyle");
    this.axSymbologyControl1.StyleClass = esriSymbologyStyleClass.esriStyleClassColorRamps;
    //色带另存
    this.pSymbologyStyleClass = axSymbologyControl1.GetStyleClass(esriSymbologyStyleClass.esriStyleClassColorRamps);
}
```

然后将存入的色带填入 combobox 中，并设置预览色，默认选择黑白灰色带。

（5）实现拉伸渲染：根据 combobox 确定渲染方式后就可以进行渲染了。

拉伸渲染需要声明 IRasterStretchColorRampRenderer，设置渲染的波段。再转化为 IRasterRenderer，设置其 Raster 属性和色带并更新。最后导入到 IRasterLayer 的 Renderer 属性中。

```
//拉伸渲染
1 个引用
private void RasterStretchRender(IColorRamp pColorRamp, IRasterLayer pRasterLayer)
{
    //渲染器定义
    IRasterStretchColorRampRenderer pSRRender = new RasterStretchColorRampRendererClass();
    pSRRender.BandIndex = cb_band.SelectedIndex;
    IRasterRenderer pRRender = pSRRender as IRasterRenderer;

    //渲染器设置
    pRRender.Raster = pRasterLayer.Raster;
    pRRender.Update();
    pSRRender.ColorRamp = pColorRamp;

    //栅格渲染
    pRasterLayer.Renderer = pRRender;
}
```

（6）实现分级渲染：

分级渲染需要声明 IRasterClassifyColorRampRenderer，并设置渲染的波段（该波段需要生成统计数据）。再转化为 IRasterRenderer，设置其 Raster 属性和分级数。

```
//分级渲染
1 个引用
private void RasterClassifyRender(IColorRamp pColorRamp, IRasterLayer pRasterLayer, int cls_num)
{
    //渲染器定义
    IRasterClassifyColorRampRenderer pRCRender = new RasterClassifyColorRampRenderer() as IRasterClassifyColorRampRenderer;
    IRasterRenderer pRRend = pRCRender as IRasterRenderer;

    IRaster pRaster = pRasterLayer.Raster;
    IRasterBandCollection pRBandCol = pRaster as IRasterBandCollection;
    IRasterBand pRBand = pRBandCol.Item(cb_band.SelectedIndex);
    if (pRBand.Histogram == null)
    {
        pRBand.ComputeStatsAndHist();
    }
    pRRend.Raster = pRaster;
    pRCRender.ClassCount = cls_num;
    pRRend.Update();
```

色带方面，根据色带大小和分级数，等间隔为每一个分类分配颜色，并更新。最后导入到 IRasterLayer 的 Renderer 属性中

```
//渲染映射
double gap = pColorRamp.Size / (cls_num - 1);
IFillSymbol fillSymbol = new SimpleFillSymbol() as IFillSymbol;
for (int i = 0; i < pRCRender.ClassCount; i++)
{
    int index;
    if (i < pRCRender.ClassCount - 1)
    {
        index = Convert.ToInt32(i * gap);
    }
    else
    {
        index = pColorRamp.Size - 1;
    }
    fillSymbol.Color = pColorRamp.get_Color(index);
    pRCRender.set_Symbol(i, fillSymbol as ISymbol);
    pRCRender.set_Label(i, pRCRender.get_Break(i).ToString("0.00") + "-" + pRCRender.get_Break(i + 1).ToString("0.00"));
}
//栅格渲染
pRasterLayer.Renderer = pRRend;
```

（7）实现唯一值渲染：

拉伸渲染需要声明 IRasterUniqueRenderer，再转化为 IRasterRenderer，设置其 Raster 属性和波段并更新。

```
//唯一值渲染
1 个引用
private void RasterUniqueRender(IColorRamp pColorRamp, IRasterLayer pRasterLayer)
{
    //渲染器定义
    IRasterUniqueValueRenderer pRURender = new RasterUniqueValueRendererClass();
    IRasterRenderer pRRend = pRURender as IRasterRenderer;

    //渲染器设置
    pRRend.Raster = pRasterLayer.Raster;
    pRRend.Update();

    IUniqueValues uniqueValues = new UniqueValuesClass();
    IRasterCalcUniqueValues calcUniqueValues = new RasterCalcUniqueValuesClass();
```

然后开始使用 IUniqueValues 和 IRasterCalcUniqueValues 计算波段内唯一值。并根据色带大小和唯一值数数，等间隔为每一个唯一值分配颜色，并更新。最后导入到 IRasterLayer 的 Renderer 属性

```
try
{
    calcUniqueValues.AddFromRaster(pRasterLayer.Raster, cb_band.SelectedIndex, uniqueValues);//计算第n通道的唯一值
    //渲染映射
    double gap = pColorRamp.Size / (uniqueValues.Count - 1);
    IFillSymbol fillSymbol = new SimpleFillSymbol() as IFillSymbol;
    for (int i = 0; i < uniqueValues.Count; i++)
    {
        int index;
        if (i < uniqueValues.Count - 1)
        {
            index = Convert.ToInt32(i * gap);
        }
        else
        {
            index = pColorRamp.Size - 1;
        }
        fillSymbol.Color = pColorRamp.get_Color(index);
        pRURender.AddValue(0, i, uniqueValues.get_UniqueValue(i));
        pRURender.set_Label(0, i, uniqueValues.get_UniqueValue(i).ToString());
        pRURender.set_Symbol(0, i, fillSymbol as ISymbol);
    }
    //栅格渲染
    pRasterLayer.Renderer = pRRend;
}
catch(Exception ex)
{
    MessageBox.Show("唯一值过多！");
}
```

附 1.2.6 RGB 渲染：用户可以选定三个波段对应三个渲染域，实现真彩色合成、假彩色合成等等功能：

（1）遥感图像操作媒介：设置 TOCControl。和单波段渲染操作一致。

（2）遥感图像渲染窗口：设置点击 TOCControl 后的板块，提供渲染功能

（3）选择渲染图层:

与单波段渲染类似，我们需要先通过 IRasterBandCollection 确定波段数目，然后轮流将各波段填入代表 R/G/B 的 combobox 中以待用户选择。根据选择的索引确定操作的波段。

（4）实现 RGB 渲染：

RGB 渲染需要声明 IRasterRGBRenderer2，再转化为 IRasterRenderer，设置其 Raster 属性。并根据 combobox 选择的 R/G/B 光的索引确定 R/G/B 渲染索引进行渲染并更新。最后导入到 IRasterLayer 的 Renderer 属性中

```
#region 多波段栅格渲染
//RGB渲染
1 个引用
private void button2_Click(object sender, EventArgs e)
{
    //渲染器
    IRasterRGBRenderer2 RasterRGBRenderer2 = new RasterRGBRendererClass();
    IRasterRenderer rasterRenderer = RasterRGBRenderer2 as IRasterRenderer;
    rasterRenderer.Raster = iLayer.Raster;
    rasterRenderer.Update();
    //渲染
    RasterRGBRenderer2.RedBandIndex = RBox.SelectedIndex;
    RasterRGBRenderer2.GreenBandIndex = GBox.SelectedIndex;
    RasterRGBRenderer2.BlueBandIndex = BBox.SelectedIndex;
    rasterRenderer.Update();
    iLayer.Renderer = rasterRenderer;
    this.DialogResult = DialogResult.OK;
}
#endregion
```

附 1.2.7 实时图框地理信息：用户想要移动鼠标获得视窗内鼠标位置的地理位置和比例尺

（1）信息显示: 将获取的信息在窗体底部。在窗体下建立 StatusStrip 显示信息：

（2）MapControl 鼠标位置获取：将鼠标移动到 MapControl 某处，获得在图窗内位置：设置 MapControl 的 OnMouseMove 事件，获取鼠标的平面坐标

```
#region 实时显示地理信息
//显示鼠标点地理位置
1 个引用
private void axMapControl1_OnMouseMove(object sender, IMapControlEvents2_OnMouseMoveEvent e)
{
```

（3）MapControl 比例尺获取：获得图窗内位置比例尺信息。利用 MapControl 的 MapScale 属性即可。

（4）MapControl 位置转化：将 MapControl 内的平面坐标转化为地理坐标

利用 map 属性直接获取地理坐标，经判断仍为平面坐标时则先利用 IProjectCoordinateSystem 获取 Mapcontrol 的空间参考，再建立 WKSPoint 将其转译为地理坐标。最后将坐标形式转为度分秒格式。

```
if (axMapControl1.LayerCount > 0)
{
    //当前比例尺
    toolStripStatusLabel1.Text = "   1:" + ((long)axMapControl1.MapScale).ToString() + "  ";
    IProjectedCoordinateSystem pcs = this.axMapControl1.SpatialReference as IProjectedCoordinateSystem;
    //定义点
    double d1, n1, m1, L1, d2, n2, m2, L2;
    WKSPoint pt = new WKSPoint(); //不能用IPoint pt = new PointClass();因为后面的方法只支持WKSPoint
    pt.X = e.mapX;
    pt.Y = e.mapY;
    if (pt.X > 180 || pt.X < -180 || pt.Y > 180 || pt.Y < -180)
        pcs.Inverse(1, ref pt); //将平面坐标转换为地理坐标
    d1 = pt.X / 1;//度的整数部分
    n1 = pt.X % 1;//度的小数部分
    m1 = (n1 * 60) / 1;//分
    L1 = ((n1 * 60) % 1) * 60;//秒
    d2 = pt.Y / 1;//纬度的整数部分
    n2 = pt.Y % 1;//度的小数部分
    m2 = (n2 * 60) / 1;//分
    string m22;
    if (m2 < 10)
        m22 = "0" + m2.ToString();
    else
        m22 = m2.ToString();
    L2 = ((n2 * 60) % 1) * 60;//秒
```

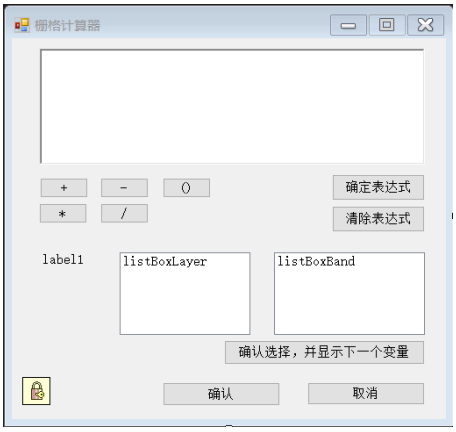
（5）PageLayoutControl 的实时地理位置显示与 MapControl 思路一致。

附录 1.3 数据处理

附 1.3.1 栅格计算器：用户想要自定义公式计算二级数据

（1）界面设计：

用户在上方输入一个公式，点击【确定表达式】后系统会自动检测变量，之后会轮流显示每一个变量，需要用户在下方选择对应图层的波段，最后点击【确认】即可计算。



（2）输入公式：用户输入计算公式

设计一个 richtextbox，用户可以自行键入公式，也可以点击系统提供的加号、减号等按键辅助输入，最后我们需要提取键入公式的字符串。

（3）提取变量：利用正则表达式提取输入变量

建立正则表达式，应用于用户输入的公式文本，提取对应变量（变量要求以英文字母为开头，由英文字母或数字或下划线组成）。将变量存入列表，并准备遍历。

```
#region 处理集构建

//确认公式
1 个引用
private void ConfirmEquation_Click(object sender, EventArgs e)
{
    //读取变量项
    bool flag = true;
    var pattern = new Regex(@"( [^\W_\d] ( [^\W] )* )",RegexOptions.IgnorePatternWhitespace);
    var input = richTextBox1.Text;
    Variable = new List<string>();
    foreach (Match m in pattern.Matches(input))
    {
        if (m.Groups[1].Value == "float")
            continue;
        flag = true;
        foreach (string s in Variable)
            if (m.Groups[1].Value == s)
                flag = false;
        if (flag)
        {
            Variable.Add(m.Groups[1].Value);
        }
    }
```

（4）选择变量：根据提取的变量选择对应波段

首先将所有图层导入 Combobox 供选择。

```
//可选图层函数
1 个引用
private void ChoiceBand(ListBox listbox)
{
    for (int i = 0; i < Layers.Count; i++)
    {
        listBoxLayer.Items.Add(Layers[i].Name);
    }
    listbox.SelectedIndex = 0;
}
```

当选择了操作图层后，会触发 listBox 的索引改变事件，这个时候另一个 listBox 会存入该图层的所有波段以供选择。

```
//可选波段函数
1 个引用
private void listBoxLayer_SelectedIndexChanged(object sender, EventArgs e)
{
    listBoxBand.Items.Clear();
    IRasterBandCollection rasterBandCollection = (IRasterBandCollection)Layers[listBoxLayer.SelectedIndex].Raster;
    for (int i = 0; i < rasterBandCollection.Count; i++)
    {
        listBoxBand.Items.Add(rasterBandCollection.Item(i).Bandname);
    }
    listBoxBand.SelectedIndex = 0;
}
```

对于系统的所有变量，系统生成一个游标 cursor 来进行遍历，每遍历到一个变量，会要求用户选择对于的图层和波段，并存为 IGeodataset 列表。

```
    //进行下一步操作
    if (Variable.Count>0)
    {
        cursor = 0;
        labelVariable.Text = Variable[0];
        ConfirmEquation.Enabled = false;
        ClearEquation.Enabled = false;
        button1.Enabled = true;
        ChoiceBand(listBoxLayer);
    }

}

//所有变量对应
1 个引用
private void button1_Click(object sender, EventArgs e)
{
    if (cursor<Variable.Count)
    {
        IRasterBandCollection rb = (IRasterBandCollection)Layers[listBoxLayer.SelectedIndex].Raster;
        geodataset.Add(rb.Item(listBoxBand.SelectedIndex) as IGeoDataset);
        cursor++;
        if(cursor<Variable.Count)
        {
            labelVariable.Text = Variable[cursor];
        }
    }
    if (cursor == Variable.Count)
    {
        button1.Enabled = false;
        ConfirmValue.Enabled = true;
    }
```

（5）计算结果：根据选择的变量计算对应结果

首先声明 IMapAlgebraOp 计算器，并将变量与选择的波段相匹配，再根据选择的变量结合输入公式文本建立栅格计算器计算要求的文本。

```
#region 栅格计算器计算
//确认计算
1 个引用
private void ConfirmValue_Click(object sender, EventArgs e)
{
    // 栅格计算器
    IMapAlgebraOp pRSalgebra = new RasterMapAlgebraOpClass();
    // 从波段集中获取单个波段，即该波段的矩阵GeoDataSet（Item的下标从0开始）
    for (int i = 0; i < geodataset.Count; i++)
    {
        pRSalgebra.BindRaster(geodataset[i],Variable[i]);
    }
    String strOut = richTextBox1.Text;
    for (int i = 0; i < Variable.Count; i++)
    {
        strOut = strOut.Replace(Variable[i], "[" + Variable[i] + "]");
    }
```

调用栅格计算器的 Execute 函数，代入计算文本，获得结果地理数据集，将其转化为栅格数据集，并通过 CreateFromRaster 生成图层。

```
try
{
    // 执行
    IGeoDataset pOutGeo = pRSalgebra.Execute(strOut);
    IGeoDataset pGeo1 = RasterArray.ProNoDataRaster(pOutGeo);
    // 将矩阵GeoDataset转成栅格集
    IRasterBandCollection irbc = (IRasterBandCollection)pGeo1;
    IRasterDataset pRD = irbc.Item(0).RasterDataset;
    //rastDataset_savi = pRD;

    outLayer = new RasterLayerClass();
    outLayer.CreateFromRaster(pOutGeo as IRaster);
    this.DialogResult = DialogResult.OK;
}
catch(Exception ex)
{
    MessageBox.Show("输入情况出错");
}
```

附 1.3.2 NDVI 计算：用户想要获取遥感数据的 NDVI 信息

（1）获取图层和波段数据

将所有图层导入到 listBox 中

```
1 个引用
public NDVI(AxMapControl mapcontrol)//构造函数
{
    for (int i = 0; i < mapcontrol.LayerCount; i++)
    {
        ILayer layer = mapcontrol.get_Layer(i);
        if (layer is IRasterLayer)
        {
            IRasterLayer newLayer = layer as IRasterLayer;
            Layers.Add(newLayer);
        }
    }
    InitializeComponent();
    ConfirmValue.Enabled = false;

    for (int i = 0; i < Layers.Count; i++)
    {
        listBoxLayer.Items.Add(Layers[i].Name);
        listBoxLayer.SelectedIndex = 0;
    }

}
```

当选择图层变化时，将对应图层的所有波段导入代表红光和代表近红外的 listBox 中供用户选择。

```
//可选波段函数
1 个引用
private void listBoxLayer_SelectedIndexChanged(object sender, EventArgs e)
{
    listBoxBandNIR.Items.Clear();
    listBoxBandR.Items.Clear();
    IRasterBandCollection rasterBandCollection = (IRasterBandCollection)Layers[listBoxLayer.SelectedIndex].Raster;
    for (int i = 0; i < rasterBandCollection.Count; i++)
    {
        listBoxBandNIR.Items.Add(rasterBandCollection.Item(i).Bandname);
        listBoxBandR.Items.Add(rasterBandCollection.Item(i).Bandname);
    }
    listBoxBandNIR.SelectedIndex = 0;
    listBoxBandR.SelectedIndex = 0;
    ConfirmValue.Enabled = true;
}
```

（2）NDVI 计算：根据图层数据获得 NDVI

计算原理本质是调用栅格计算器，将红光和近红外波段与我设置的 R 和 NIR 变量绑定，载入计算公式"float([NIR] - [R]) / ([NIR] + [R])"，导入栅格计算器进行计算，获得结果地理数据集，将其转化为栅格数据集，并通过 CreateFromRaster 生成图层。

```
//确认计算
1 个引用
private void ConfirmValue_Click(object sender, EventArgs e)
{
    // 栅格计算器
    IMapAlgebraOp pRSalgebra = new RasterMapAlgebraOpClass();
    // 从波段集中获取单个波段
    IRasterBandCollection rb = (IRasterBandCollection)Layers[listBoxLayer.SelectedIndex].Raster;
    IGeoDataset NIR = rb.Item(listBoxBandNIR.SelectedIndex) as IGeoDataset;
    IGeoDataset R = rb.Item(listBoxBandR.SelectedIndex) as IGeoDataset;
    pRSalgebra.BindRaster(NIR, "NIR");
    pRSalgebra.BindRaster(R, "R");
    String strOut = "float([NIR] - [R]) / ([NIR] + [R])";
    try
    {
        // 执行
        IGeoDataset pOutGeo = pRSalgebra.Execute(strOut);
        // 将矩阵GeoDataset转成栅格集
        IRasterBandCollection irbc = (IRasterBandCollection)pOutGeo;
        IRasterDataset pRD = irbc.Item(0).RasterDataset;
        //rastDataset_savi = pRD;

        outLayer = new RasterLayerClass();
        outLayer.CreateFromRaster(pOutGeo as IRaster);
        this.DialogResult = DialogResult.OK;
    }
    catch(Exception ex)
    {
        MessageBox.Show("输入情况出错");
    }
}
#endregion
```

附 1.3.3 坡度计算：用户想要获取遥感数据的坡度信息

坡度的计算分为图层选取、坡度计算和图层导出，图层选取和图层导出跟 NDVI 计算差别不大，关键在于坡度计算:

调用 ISurfaceOp 的 Slope 方法，输入地理数据集和输出单位即可。

```
#region 计算坡度
1 个引用
private void SlopeCountbtn_Click(object sender, EventArgs e)
{
    try
    {
        outGeodataset = surfaceOp.Slope(inGeodataset, slopeEnum, ref Missing);//坡度计算方法
        Sloperesult = ShowRasterResult(outGeodataset, "Slope(坡度)");
        this.Close();
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

附 1.3.4 坡向计算：用户想要获取遥感数据的坡向信息

坡向的计算也分为图层选取、坡度计算和图层导出，图层选取和图层导出跟 NDVI 计算差别不大，关键在于坡向计算:

调用 ISurfaceOp 的 Aspect 方法，输入地理数据集和输出单位即可。

```
#region 计算坡向
1 个引用
private void AspectCountbtn_Click(object sender, EventArgs e)
{
    try
    {
        outGeodataset = surfaceOp.Aspect(inGeodataset);
        Aspectresult = ShowRasterResult(outGeodataset, "Aspect(坡向)");
        this.Close();

    }
    catch { }
}
```

附 1.3.5 栅格裁剪：用户想要根据某个范围裁剪栅格

（1）图框元素侦测：确定 MapControl 被选择的元素作为裁剪范围

将 MapControl 的 Map 转化为一个 IGraphicsContainerSelect，其 ElementSelectionCount 属性可以反映图框中选择的元素是，当不存在选择选择元素时，弹出提醒。当存在是，获取第一个元素，并将其范围作为裁剪范围。

```
//裁剪框
IGraphicsContainerSelect gcs = axMapControl1.Map as IGraphicsContainerSelect;
if (gcs.ElementSelectionCount == 0)
{
    MessageBox.Show("当前没有选择任何Element！请先选取。");
    return;
}
IElement el = gcs.SelectedElement(0);
IEnvelope geo = el.Geometry.Envelope;
```

（2）选择图层：确定被裁剪的图层

弹出一个新窗体，选择被裁剪的图层。选择原理和栅格计算器部分涉及原理相似。将所有图层导入 listBox，供用户选择，返回被选择图册的列表。

```
//裁剪图层
SelectLayer sl = new SelectLayer(axMapControl1);
```

（3）裁剪结果生成：点击裁剪后，图框内出现被裁剪的图层
循环被选择的图层列表，调用 RasterClip 函数进行裁剪，返回裁剪后的图层。

```
if (sl.ShowDialog() == DialogResult.OK)
{
    for (int i = 0; i < sl.outLayers.Count; i++)
    {
        IRasterLayer outLayer = RasterManage.RasterClip(sl.outLayers[i], ge
        outLayer.Name = sl.outLayers[i].Name + "[裁剪后]";
        axMapControl1.AddLayer(outLayer);
    }
}
```

裁剪函数 RasterClip 要求输入栅格图层和裁剪范围，声明 IExtractionOp，利用其 Rectangle 方法进行裁剪。本质上是种掩膜手段。效率相对也较高。

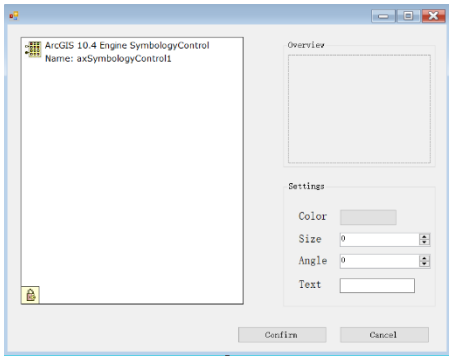
```
1 个引用
public static IRasterLayer RasterClip(IRasterLayer pRasterLayer, IEnvelope clipGeo)
{
    // IExtractionOp
    IExtractionOp extraction = new RasterExtractionOpClass();
    try
    {

        IGeoDataset geoDataset = extraction.Rectangle((IGeoDataset)pRasterLayer.Raster, clipGeo, true);
        IRaster raster = geoDataset as IRaster;
        if (raster != null)
        {
            IRasterLayer outLayer = new RasterLayerClass();
            outLayer.CreateFromRaster(raster);
            return outLayer;
        }
        else
        {
            MessageBox.Show("没有栅格值");
            return null;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return null;
    }
}
```

附录 1.4 地图导出

附 1.4.1 样式选择器：用户可以自主选择样式填充到制图要素中。

功能实现：
（1）设计一个新窗口用来选择样式：



该窗口左半部分为 symbology control 相关内容，负责展示并点选样式；右半部分上方的 PictureBox 可以预览选择样式，下方可以预览并修改样式的颜色、大小、角度和文本。

（2）该样式选择器基本工作原理主要是通过下方 Get_Item 函数和 SymbologyControl 的选择改变事件配合抓取所选要素。

```
/// <summary> 获取样式
8 个引用
public IStyleGalleryItem GetItem(ESRI.ArcGIS.Controls.esriSymbologyStyleClass styleClass)
{
    m_StyleGalleryItem = null;
    button1.Enabled = false;
    axSymbologyControl1.StyleClass = styleClass;//读取传入样式
    axSymbologyControl1.GetStyleClass(styleClass).UnselectItem();
    Visible();
    this.ShowDialog();
    return m_StyleGalleryItem;
}
```

```
/// <summary> 抓取样式
1 个引用
private void axSymbologyControl1_OnItemSelected(object sender, ESRI.ArcGIS.Controls.ISymbologyControlEvents_OnItemSelectedEvent e)
{
    textBox1.Enabled = true;
    m_StyleGalleryItem = axSymbologyControl1.GetStyleClass(axSymbologyControl1.StyleClass).GetSelectedItem();
    ShowAttributes();
    button1.Enabled = true;
}
```

（3）而在最开始，程序会调用 Visible 函数根据传入的 styleclass 类型来使大小、角度、文本等调整框可见或不可见（如下，节选，当 styleclass 为点符号时，角度可见，文本不可见），并调用 PrevieImage 函数在预览框内展示所选要素：

```
///
///设置不同情况的可见性
///
1 个引用
private void Visible()
{
    switch (this.axSymbologyControl1.StyleClass)
    {
        case esriSymbologyStyleClass.esriStyleClassMarkerSymbols:
            this.lblAngle.Visible = true;
            this.Angle.Visible = true;
            this.textBox1.Visible = false;
            this.Text.Visible = false;
            break;

        case esriSymbologyStyleClass.esriStyleClassLineSymbols:
            this.lblAngle.Visible = false;
            this.Angle.Visible = false;
            this.textBox1.Visible = false;
            this.Text.Visible = false;
            break;
```

```
#region 展示格相关
/// ...
5 个引用
private void PreviewImage()

{
    stdole.IPictureDisp picture = this.axSymbologyControl1.GetStyleClass(this.axSymbologyControl1.StyleClass).
        PreviewItem(m_StyleGalleryItem, this.ptbPreview.Width, this.ptbPreview.Height);
    System.Drawing.Image image = System.Drawing.Image.FromHbitmap(new System.IntPtr(picture.Handle));
    this.ptbPreview.Image = image;
}
```

（4）当我们修改样式属性时，会触发各种空间的值改变事件，以角度值改变事件为例，程序首先会判断 styleClass 类型，然后根据不同类型修改对应样式的值，最后调用预览函数。大小、颜色、文本等的修改同理。

```
1 个引用
private void lblAngle_ValueChanged(object sender, EventArgs e)
{
    switch (this.axSymbologyControl1.StyleClass)
    {
        //点符号
        case esriSymbologyStyleClass.esriStyleClassMarkerSymbols:
            ((IMarkerSymbol)this.m_StyleGalleryItem.Item).Angle = (double)this.lblSize.Value;
            break;
        //文字
        case esriSymbologyStyleClass.esriStyleClassTextSymbols:
            ((ITextSymbol)this.m_StyleGalleryItem.Item).Angle = (double)this.lblSize.Value;
            break;
    }
    this.PreviewImage();
}
```

附 1.4.2 添加图名：用户想要给地图添加图名

（1）用户触发【添加图名】事件&文字样式选择

触发事件后标记枚举类型为 Title，触发样式选择器窗口，选择相关样式存入后台。

```
#region 添加标题
1 个引用
private void pictureBox3_Click(object sender, EventArgs e)
{
    try
    {
        _enumMapSurType = EnumMapSurroundType.Title;
        SymbolChoice SymbolChoicePanel = new SymbolChoice();
        IStyleGalleryItem styleGalleryItem = SymbolChoicePanel.GetItem(esriSymbologyStyleClass.esriStyleClassTextSymbols);
        if (styleGalleryItem == null) return;
        pStyleGalleryItem = styleGalleryItem;
        SymbolChoicePanel.Dispose();
    }
    catch (Exception ex)
    {
    }
}
```

（2）确定添加位置

首先生成 PageLayoutControl 的 OnMouseDown 事件，当鼠点击 PageLayoutControl 且标记枚举类型不为 None（比如添加标题时该值为 Title），确定点击点为起始点，之后鼠标滑动触发 OnMouseMove 事件，每次移动后或最后再次点击时相应点成为结束点，根据起始点和结束点生成 INewEnvelopeFeedback 值。

```
#region 确定添加框架

//通过OnMouseDown事件，产生矩形框的第一个点
1 个引用
private void axPageLayoutControl1_OnMouseDown(object sender, IPageLayoutControlEvents_OnMouseDownEvent e)
{
    try
    {
        if (_enumMapSurType != EnumMapSurroundType.None)
        {
            IActiveView pActiveView = null;
            pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
            m_PointPt = pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(e.x, e.y);
            if (pNewEnvelopeFeedback == null)
            {
                pNewEnvelopeFeedback = new NewEnvelopeFeedbackClass();
                pNewEnvelopeFeedback.Display = pActiveView.ScreenDisplay;
                pNewEnvelopeFeedback.Start(m_PointPt);
            }
            else
            {
                pNewEnvelopeFeedback.MoveTo(m_PointPt);
            }

        }
    }
    catch
    {
    }
}
```

```
1 个引用
private void axPageLayoutControl1_OnMouseMove(object sender, IPageLayoutControlEvents_OnMouseMoveEvent e)
{
    try
    {
        if (_enumMapSurType != EnumMapSurroundType.None)
        {
            if (pNewEnvelopeFeedback != null)
            {
                m_MovePt = (axPageLayoutControl1.PageLayout as IActiveView).ScreenDisplay.DisplayTransformation.ToMapPoint(e.x, e.y);
                pNewEnvelopeFeedback.MoveTo(m_MovePt);
            }
        }
    }
```

```
//通过OnMouseUp事件，产生矩形框的第一个点的对焦点，返回一个矩形，并将制图要素添加到该矩形中
1 个引用
private void axPageLayoutControl1_OnMouseUp(object sender, IPageLayoutControlEvents_OnMouseUpEvent e)
{
    if (_enumMapSurType != EnumMapSurroundType.None)
    {
        if (pNewEnvelopeFeedback != null)
        {
            IActiveView pActiveView = null;
            pActiveView = axPageLayoutControl1.PageLayout as IActiveView;
            IEnvelope pEnvelope = pNewEnvelopeFeedback.Stop();
            AddMapSurround(pActiveView, _enumMapSurType, pEnvelope);
            pNewEnvelopeFeedback = null;
            _enumMapSurType = EnumMapSurroundType.None;
        }
    }
}
```

当不再点击鼠标时，触发 OnMouseUp 事件，根据 INewEnvelopeFeedback 生成 IEnvelop 作为添加制图要素的地理范围。接着根据标记枚举类型调用对应的添加制图要素函数（这里标记枚举类型为 Title，因此调用添加标题函数）。最后还原标记枚举类型到 None。

（3）实现图名添加

先声明一个 TextElement，导入之前选择的样式，text 为添加文本。转化为 Element，确定其位置，作为元素添加到 GraphicContainer 上。可以通过元素选择器移动、删除或者改变尺寸。

```
1 个引用
private void MakeTitle(IPageLayout pPageLayout, IEnvelope pEnv, IActiveView pActiveView)
{
    IMap pMap = pActiveView.FocusMap;
    IGraphicsContainer pGraphicsContainer = pPageLayout as IGraphicsContainer;
    IMapFrame pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;
    if (pStyleGalleryItem == null) return;
    IMapSurroundFrame pMapSurroundFrame = new MapSurroundFrameClass();
    pMapSurroundFrame.MapFrame = pMapFrame;
    ITextElement txtElement = new TextElementClass();
    ITextSymbol pTextSymbol = (ITextSymbol)pStyleGalleryItem.Item;//Text的符号样式
    txtElement.Text = pTextSymbol.Text;
    txtElement.Symbol = pTextSymbol;
    //pMapSurroundFrame.MapSurround = (IMapSurround)pTextSymbol;//根据用户的选取，获取相应的MapSurround

    //IElementProperties pElePro = null;
    //pElement = (IElement)pMapSurroundFrame;
    //pElement.Geometry = (IGeometry)pEnv;
    //pElePro = pElement as IElementProperties;



    IElement pElement = txtElement as IElement;
    pElement.Geometry = (IGeometry)pEnv;
    pGraphicsContainer.AddElement(pElement, 0);
    pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
}
```

附 1.4.3 添加指北针：用户想要给地图添加指北针

（1）用户触发【添加指北针】事件&选择样式

此处与添加图名时思路一致，区别在于其标记枚举类型为 NorthArrow。

（2）确定添加位置

此处与添加图名时思路一致，不赘述。

（3）实现指北针添加

包括指北针、比例尺、图例在内这一类的制图要素需要围绕 IMapSurround 接口展开，首先依次进行 IMap->GraphicContainer->FrameElement-> MapFrame-> MapSurroundFrame 的转化，然后将选择的指北针样式赋值到 MapSurroundFrame 的 MapSurround 内，转化为 Element，导入之前选择的范围，进行添加。

```
1 个引用
private void addNorthArrow(IPageLayout pPageLayout, IEnvelope pEnv, IActiveView pActiveView)
{
    IMap pMap = pActiveView.FocusMap;
    IGraphicsContainer pGraphicsContainer = pPageLayout as IGraphicsContainer;
    IMapFrame pMapFrame = pGraphicsContainer.FindFrame(pMap) as IMapFrame;
    if (pStyleGalleryItem == null) return;
    IMapSurroundFrame pMapSurroundFrame = new MapSurroundFrameClass();
    pMapSurroundFrame.MapFrame = pMapFrame;
    INorthArrow pNorthArrow = new MarkerNorthArrowClass();
    pNorthArrow = pStyleGalleryItem.Item as INorthArrow;
    //pNorthArrow.Size = pEnv.Width * 50;
    pMapSurroundFrame.MapSurround = (IMapSurround)pNorthArrow;//根据用户的选取，获取相应的MapSurround
    IElement pElement = axPageLayoutControl1.FindElementByName("NorthArrows");//获取PageLayout中的指北针元素
    if (pElement != null)
    {
        pGraphicsContainer.DeleteElement(pElement);  //如果存在指北针，删除已经存在的指北针
    }
    IElementProperties pElePro = null;
    pElement = (IElement)pMapSurroundFrame;
    pElement.Geometry = (IGeometry)pEnv;
    pElePro = pElement as IElementProperties;
    pElePro.Name = "NorthArrows";
    pGraphicsContainer.AddElement(pElement, 0);
    pActiveView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null, null);
}
```

附 1.4.4 添加比例尺：用户想要给地图添加比例尺

（1）用户触发【添加比例尺】事件&选择样式

此处与添加指北针时思路一致，区别在于其标记枚举类型为 ScaleBar。

（2）确定添加位置

此处与添加指北针时思路一致，不赘述。

（3）实现指北针添加

添加思路与指北针一致，不赘述。

附 1.4.5 添加图例：用户想要给地图添加图例

（1）用户触发【添加图例】事件&选择样式

此处与添加指北针时思路一致，区别在于其标记枚举类型为 Legend。且我们不再通过样式选择器选择图例样式。

（2）确定添加位置

此处与添加指北针时思路一致，不赘述。

（3）实现指北针添加

添加思路与指北针、图例大致一致，唯一的区别在于通过 UID 确定样式，然后通过 MapFrame 的 CreateSurroundFrame 方法导入样式。

附 1.4.6 导出地图：用户想要将处理过后的数据导出为地图

（1）选定导出格式

选定格式直接用 SaveFileDialog 的 Filter 配合判断输出文件名是否包含某个选项实现，例如输出文件名包含“jpg”，则进行 jpg 输出模式。

```
#region 制图-导出地图
1 个引用
private void exportMapToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog m_save = new SaveFileDialog();
    m_save.Filter = "jpeg图片(*.jpg)|*.jpg|tiff图片(*.tif)|*.tif|bmp图片(*.bmp)|*.bmp|emf图片(*.emf)|*.emf|png图片(
    m_save.ShowDialog();
    string Outpath = m_save.FileName;
    if (Outpath != "")
    {
        //分辨率
        double resolution = axPageLayoutControl1.ActiveView.ScreenDisplay.DisplayTransformation.Resolution;
        IExport m_export = null;
        if (Outpath.EndsWith(".jpg"))
        {
            m_export = new ExportJPEG() as IExport;

        }
        else if (Outpath.EndsWith(".tig"))
        {
            m_export = new ExportTIFF() as IExport;

        }
```

（2）实现导出

输出地图需要先确定路径、分辨率，再获得 PageLayout 是 tagRECT 框数据，根据该数据确定 Envelop 范围，最后根据该范围、分辨率等信息调用 PageLayoutControl 的 ActiveView 的 Output 函数输出地图

```
//设置输出的路径
m_export.ExportFileName = Outpath;
//设置输出的分辨率
m_export.Resolution = 300;
IActiveView activeView = this.axPageLayoutControl1.PageLayout as IActiveView;
tagRECT piexPound = activeView.ScreenDisplay.DisplayTransformation.get_DeviceFrame();
//设置输出的IEnvelope
IEnvelope m_envelope = new Envelope() as IEnvelope;
m_envelope.PutCoords(piexPound.left, piexPound.bottom, piexPound.right, piexPound.top);
m_export.PixelBounds = m_envelope;

ITrackCancel m_trackCancel = new CancelTracker();
//输出的方法
axPageLayoutControl1.ActiveView.Output(m_export.StartExporting(), (short)resolution, ref piexPound, activeView.Extent, m_trackCancel);
m_export.FinishExporting();
```

附录 1.5 数据转化（CA 算法辅助函数）

附 1.5.1 栅格复制函数：复制一个栅格图层到新图层

实现该函数主要是为了满足两类需求，第一类纯粹是复制一个栅格图层；而第二类，不论是 CA 还是 CFDA 算法都会根据数组生成新图层，而需要该图层跟初始图层地理信息一致，此时往往复制初始图层并修改数组复制信息会更加省事。(不复制的话也可以直接修改原图层，但这样会对原图层造成不可逆的改变)

实现：

调用栅格计算器，设计变量 Raster，绑定输入图层到 Raster，计算公式“[Raster]”，获得地理数据集。

```
//新建栅格
2 个引用
public static IGeoDataset CopyGeoDataset(IGeoDataset pGeo)
{
    IMapAlgebraOp pRSalgebra = new RasterMapAlgebraOpClass();
    pRSalgebra.BindRaster(pGeo, "Raster");
    String strOut = "[Raster]";
    IGeoDataset pOutGeo = pRSalgebra.Execute(strOut);
    return pOutGeo;
}
```

附 1.5.2 栅格无效值处理函数：改变栅格的无效值。

在进行 CFDA 或元胞自动机计算时，我们需要遍历栅格的所有数据，但这些数据包含了大量的背景值，即无效值，尤其是栅格具有多个无效值，或无效值与正常值冲撞时，我们难以进行辨别，因此要提前统一改变其无效值。

实现：

将导入的地理数据集转化为 IRasterProps，将其无效值转化为-99。这样可以有效规避无效值干扰情况。

```
5 个引用
public static IGeoDataset ProNoDataRaster(IGeoDataset pGeoDataset)
{
    IGeoDataset my_GeoDataset = pGeoDataset;
    //IRasterBandCollection pRBandCollection = my_GeoDataset as IRasterBandCollection;
    //IRasterBand pRBand = pRBandCollection.Item(0);
    IRasterProps pRasterprops = my_GeoDataset as IRasterProps;
    pRasterprops.NoDataValue = -99;
    //pRBand = pRasterprops as IRasterBand;
    //pRBand.ComputeStatsAndHist();
    //pRBandCollection = pRBand.RasterDataset as IRasterBandCollection;
    return pGeoDataset as IGeoDataset;
}
```

附 1.5.3 获得背景图层函数：根据栅格每个像元是背景值与否获得二值背景图层。

在元胞自动机算法执行时，我们需要循环整个栅格的所有像元，但我们要处理并计算的只是它的有效值，这些有效值往往只占小部分。以此需要一个二值的背景图层来判断我们是否遇到了有效值，起到类似于掩膜的作用，提高运算效率。

实现：

输入已进行无效值处理后的栅格图层（无效值为-99）转化后的二维数组，建立等尺寸大小的新数组，遍历该数组，是无效值新数组对应位置存入 0，反之存入 1。

```
//获取背景值
1 个引用
private double[,] GetBackGround(double[,] arr)
{
    // 获取double的array
    int x = arr.GetLength(0);
    int y = arr.GetLength(1);
    double[,] newarr = new double[x, y];
    for (int i = 0; i < arr.GetLength(0); i++)
    {
        for (int j = 0; j < arr.GetLength(1); j++)
        {
            if (arr[i, j] == -99)
                newarr[i, j] = 0;
            else
                newarr[i, j] = 1;
        }
    }
    return newarr;
}
```

附 1.5.4 栅格转 double 数组函数：将栅格图层数据提取为 double 数组。

元胞自动机算法要求输入 double 数组，因此必须要将栅格数据转化为 double 数组来进行运算。

实现:

首先，提取栅格的 float 数据到数组中。利用 IPnt 接口设置坐标系，利用 IPixelBlock 接口实现读取。通过 IPixelBlock3 接口实现数据转化和导出。

```
//栅格转数组
2 个引用
public static System.Array RasterToArray(IGeoDataset pGD)
{
    //, ref object novalue
    IRaster raster = pGD as IRaster;
    IRasterProps props = (IRasterProps)raster;
    //novalue = props.NoDataValue;
    IPnt pBlockSize = new PntClass();
    pBlockSize.SetCoords((double)props.Width, (double)props.Height);
    IRaster2 raster2 = (IRaster2)raster;
    IPixelBlock pixelBlock = raster2.CreateCursorEx(pBlockSize).PixelBlock;
    pBlockSize.SetCoords(0.0, 0.0);
    raster.Read(pBlockSize, pixelBlock);
    IPixelBlock3 block3 = (IPixelBlock3)pixelBlock;
    return (System.Array)block3.get_PixelData(0);
}
```

然后循环遍历 float 数组，将其转化为 double。

附 1.5.5 根据 double 数组生成栅格函数：利用计算得到的 double 函数生成结果栅格图层。

元胞自动机算法结果会生成 double 数组，我们需要对其进行可视化，因此必须要将其转化为栅格图层，且其地理信息要和输入栅格图层一致。

实现:

首先复制初始图层的地理数据集到一个新的地理数据集，对该新图层无效值进行处理。

```
//拷贝新图层
IGeoDataset modelset = model.Raster as IGeoDataset;
IGeoDataset newset = RasterArray.CopyGeoDataset(modelset);
newset = RasterArray.ProNoDataRaster(newset);
```

由于新图层数据为 float，因此我们需要把结果 double 数组转化为 float

```
//获取double数组
1 个引用
private double[,] Raster2Double(IRasterLayer layer)
{
    //摘除无效（为0）-转化为array
    IGeoDataset geodataset= layer.Raster as IGeoDataset;
    geodataset = RasterArray.ProNoDataRaster(geodataset);
    System.Array arr = RasterArray.RasterToArray(geodataset);

    // 获取double的array
    int x = arr.GetLength(0);
    int y = arr.GetLength(1);
    double[,] newarr = new double[x, y];
    for (int i = 0; i < arr.GetLength(0); i++)
    {
        for (int j = 0; j < arr.GetLength(1); j++)
        {
            // 这一点的像素值
            double pixelvalue = Convert.ToDouble(arr.GetValue(i, j));
            newarr[i, j] = Convert.ToDouble(pixelvalue);
        }
    }
    return newarr;
}
```

```
// 转化double到float
int x = arr.GetLength(0);
int y = arr.GetLength(1);
float[,] newarr = new float[x, y];
for (int i = 0; i < arr.GetLength(0); i++)
{
    for (int j = 0; j < arr.GetLength(1); j++)
    {
        // 这一点的像素值
        float pixelvalue = Convert.ToSingle(arr.GetValue(i, j));
        newarr[i, j] = (pixelvalue);
    }
}
```

最后调用一个修改图层数据的函数将新图层数据替换为结果数据，获得地理数据集，根据该数据集生成图层。

```
// 让二维数组arr生成栅格数据(单波段、同类型)
IGeoDataset aod = RasterArray.AlterRasterArray(newset,newarr); //修改pGeo中的矩阵数组
IRasterLayer outLayer = new RasterLayerClass();
outLayer.CreateFromRaster(aod as IRaster);
return outLayer;
```

对于这个修改图层数据的函数，需要传入地理数据集和跟该数据同尺寸、同数据类型的数组。利用 IPnt 接口设置坐标系和游标，利用 IPixelBlock 接口实现读取和替换。通过 IPixelBlock3 接口实现数据转化和导出。

```
//根据数组新值构造栅格
2 个引用
public static IGeoDataset AlterRasterArray(IGeoDataset pGeo, Array arr)
{
    try
    {
        IRaster raster = pGeo as IRaster;
        IRasterProps props = (IRasterProps)raster;
        IPnt pBlockSize = new PntClass();
        pBlockSize.SetCoords((double)props.Width, (double)props.Height);
        IRaster2 raster2 = (IRaster2)raster;
        IPixelBlock pixelBlock = raster2.CreateCursorEx(pBlockSize).PixelBlock;
        pBlockSize.SetCoords(0.0, 0.0);
        raster.Read(pBlockSize, pixelBlock);
        pixelBlock.set_SafeArray(0, arr);
```

最后利用 IRasterEdit 将更新的值写入 raster 中，返回新的地理数据集。

```
//编辑raster，将更新的值写入raster中
IRasterEdit rasterEdit = raster as IRasterEdit;
//左上点坐标
IPnt tlp = new Pnt();
tlp.SetCoords(0, 0);
rasterEdit.Write(tlp, pixelBlock);
rasterEdit.Refresh();
return pGeo;
```

附录-2 小组分工

李森洋:数据导入与制图、数据浏览、数据基本处理代码与窗体编写；程序整合；程序修改与优化

张颖:用户管理系统、CFDA 算法、林火检测结果可视化代码与窗体编写；实例数据下载与处理；图标制作；损失评估窗体编写。

黄妙:CA 原理理解、讲解文档制作、源代码编写；损失评估原代码编写；实例数据处理；程序美化

参考文献

[1] Eric Innocenti，Xavier Silvani，Alexandre Muzy，et al． A software framework for fine grain parallelization of cellular models with OpenMP: Application to fire spread ［J］． Environmental Modelling ＆ Software，2009 (24) : 819 ～ 831．

[2] 周成虎，孙战利，谢一春． 地理元胞自动机研究［M］． 北京:科学出版社，1999．

[3] 李艳杰,解新路,张菲菲.基于改进的元胞自动机林火蔓延模拟研究与实现[J].绿色科技,2012(08):109-112.

[4] Hernandez Encinas A． ，Hoya White S． ． Simulation of forest fire fronts using cellular automata ［J］． Advances in Engineering Software，2007(38) : 372 ～ 378．

[5] 张菲菲．基于地理元胞自动机的林火蔓延模型与模拟研究［D］． 汕头: 汕头大学，2011

[6] Stephen G Berjak，John W Hearne． An improved cellular automaton model for simulation fire in a spatially heterogeneous Savanna system ［J］． Ecological Modeling，2002(148) : 135 ～ 140．

[7] Kramer P J. Carbon dioxide concentration，photosynthesis, and dry matter production[J]. Bioscience,1981:31: 29-33.

[8] 舒立福,田晓瑞,寇晓军.林火研究综红光 1 近红外是 2 述(Ⅰ):研究热点与进展[J].世界林业研究,2003,16(3):37-40.