ПРИДНЕСТРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

им. Т.Г. Шевченко

Физико-математический факультет

Кафедра прикладной математики и информатики

|  |  |
| --- | --- |
|  | ПРОВЕРЕНО  преподаватель кафедры ПМиИ  Горб Евгений Александрович    «\_\_»\_\_\_\_\_\_\_\_ 20\_\_\_г. |

О Т Ч Ё Т

ПО УЧЕБНОЙ ПРАКТИКЕ

СТУДЕНТКИ Липчанской Анны Владимировны

|  |  |
| --- | --- |
| Факультет: | физико-математический |
| Направление подготовки: | 01.03.02 Прикладная математика и информатика |
| Профиль: | Системное программирование и компьютерные технологии |
| Квалификация: | Бакалавр |
| Курс: | III |
| Группа: | ФМ17ДР62ПФ1 |
| Форма обучения: | очная |

|  |  |
| --- | --- |
| Приказ о направлении на практику: | № 45-пр от «15» января 2020г. |
| Срок практики: | с «10» февраля 2020г.  по «16» июня 2020г. |

Тирасполь, 2020 г.

Оглавление

[Цель и задачи практики 3](#_Toc54346478)

[1. Задание “Менеджер рецептов” 4](#_Toc54346479)

[1.1. Постановка задачи 4](#_Toc54346480)

[1.2. Используемые средства и технологии 4](#_Toc54346481)

[1.3. Ход выполнения задания 4](#_Toc54346482)

[1.3.1. Модель предметной области. 4](#_Toc54346483)

[1.3.2. Структура проекта. 5](#_Toc54346484)

[1.4. Скриншоты приложения 9](#_Toc54346485)

[2. Задание “Лабиринт” 11](#_Toc54346486)

[2.1. Постановка задачи 11](#_Toc54346487)

[2.2. Используемые средства и технологии 11](#_Toc54346488)

[2.3. Структура проекта 12](#_Toc54346489)

[2.4. Скриншоты приложения 16](#_Toc54346490)

[3. Задание “Лесной пожар” 19](#_Toc54346491)

[3.1. Постановка задачи 19](#_Toc54346492)

[3.2. Используемые средства и технологии 19](#_Toc54346493)

[3.3. Структура проекта 19](#_Toc54346494)

[3.4. Скриншоты приложения 21](#_Toc54346495)

[Вывод 22](#_Toc54346496)

**ЦЕЛЬ И ЗАДАЧИ ПРАКТИКИ**

На период практики от руководителя практики от кафедры было получено следующее задание:

— Изучение среды программирования Microsoft Visual Studio 2019

— Изучение языка программирования C#

— разработка программы, реализующей операции с данными.

— разработка блок-схем алгоритмов программ и их реализация.

— разработка блок-схем алгоритмов обработки списков с последующей их программной реализацией

— изучение базовых приемов обработки строк в языке программирования C#.

— изучение приемов работы с внешними данными, а также особенностей их использования

**Задание “Менеджер рецептов”**

Постановка задачи

Создайте класс рецептов с ингредиентами и поместите их в программу управления рецептами, которая организует их по категориям, таким как десерты, основные блюда или ингредиенты, такие как курица, говядина, лук и т.д. Напишите проверочные тесты. Реализуйте приложение для создания и управления рецептами.

Используемые средства и технологии

Данное приложение разрабатывалось на объектно-ориентированном языке программирования C#. Для создания графических интерфейсов с помощью платформы .NET применялась технология – Window Forms. При реализации следовали принципам ООП и SOLID.

Ход выполнения задания

Модель предметной области.

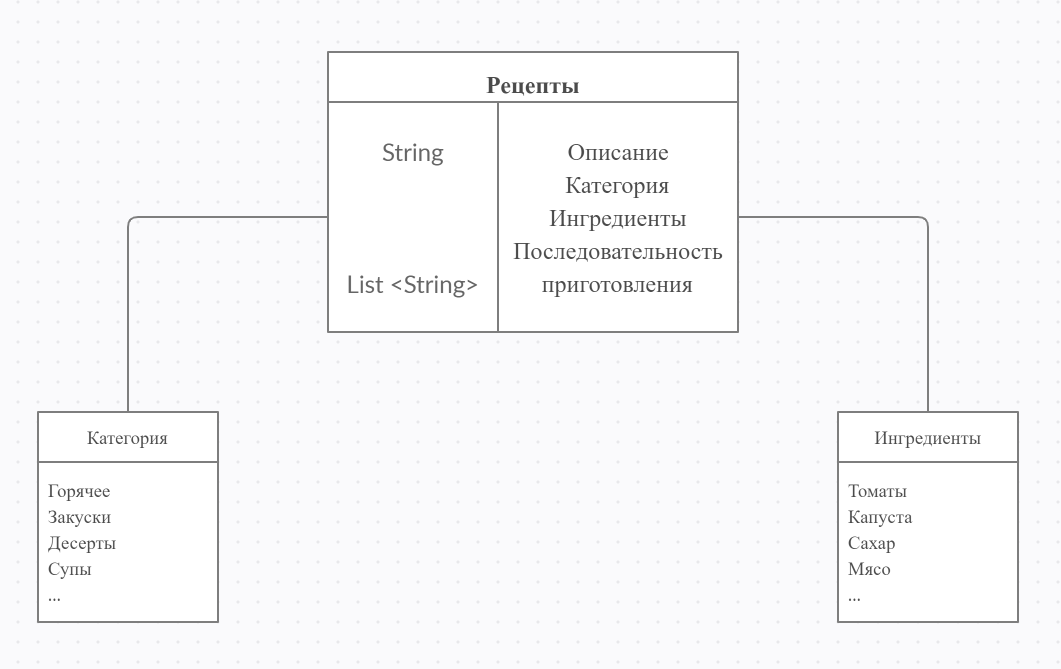


Рисунок 1. Блок-схема

Выделено 3 объекта:

Рецепты

Категории блюд

Ингредиенты

Структура проекта.

Классы:

DishCategory

Класс конструктор, ролью которой выступает создание категорий блюд. Предназначен для представления выбора категорий блюд в рецепте.

DishCategoryComposition

Класс, помогающий создать коллекцию категорий блюд и использовать её в классе Recipe

Ingredient

Класс конструктор, ролью которой выступает создание ингредиентов. Предназначен для представления выбора и создания списка ингредиентов в рецепте.

IngredientStorage

Класс, дает возможность создать коллекцию ингредиентов и взаимодействовать с ней остальным классам.

ObjectStorage

Класс использует паттерн Singleton и содержит в себе данные всех контейнеров. Предназначен для взаимодействия данных (таких как “Group”, ”Ingredient”, “Recipe”) между собой.

Recipe

Класс конструктор, ролью которой выступает создание рецептов. Предназначен для создания рецептов, в которые входят классы “Group”, “Ingredient”.

RecipeContainer

Класс, выступающий в качестве контейнера для класса “Recipe”. Дает возможность создать коллекцию рецептов и взаимодействовать с ней.

Result

Класс инициализирующий другие классы, предназначен для обработки событий ошибок.

Паттерн Singleton.

Serializable/Deserializable

public static class RecipeDataManager

{

public static void SaveData(string path, RecipeManagerData storage)

{

using (var stream = new FileStream(path, FileMode.Create))

{

var serializer = new XmlSerializer(typeof(RecipeManagerData));

serializer.Serialize(stream, storage);

}

}

public static RecipeManagerData LoadData(string path)

{

using (var fileStream = new FileStream(path, FileMode.Open, FileAccess.Read, FileShare.Read))

{

var serializer = new XmlSerializer(typeof(RecipeManagerData));

var storage = (RecipeManagerData)serializer.Deserialize(fileStream);

List<Recipe> tmpRecipes = new List<Recipe>();

foreach (var itemRecipe in storage.Recipes)

{

IngredientStorage ingredients = new IngredientStorage();

Ingredient tmpIngredient;

foreach (var itemIngredient in itemRecipe.IngredientStorage)

{

tmpIngredient = storage.Ingredients.FindLast(t => t.Name == itemIngredient.Name);

ingredients.Add(tmpIngredient);

}

DishCategory group = storage.DishCategory.FindLast(t => t.Name == itemRecipe.DishCategory.Name);

Recipe recipe = Recipe.Create(itemRecipe.Description, group, ingredients, itemRecipe.RecipeSteps).Value;

tmpRecipes.Add(recipe);

}

storage.Recipes = tmpRecipes;

return storage;

}

}

}

}

Работа с файлами конфигурации App.config

Класс Form1

private string filePath

{

get

{

return ConfigurationManager.AppSettings["recipesFileName"];

}

set

{

ConfigurationManager.AppSettings["recipesFileName"] = value;

}

}

App.config

<appSettings>

<add key="recipesFileName" value="T:\Recipes.dat" />

</appSettings>

Скриншоты приложения

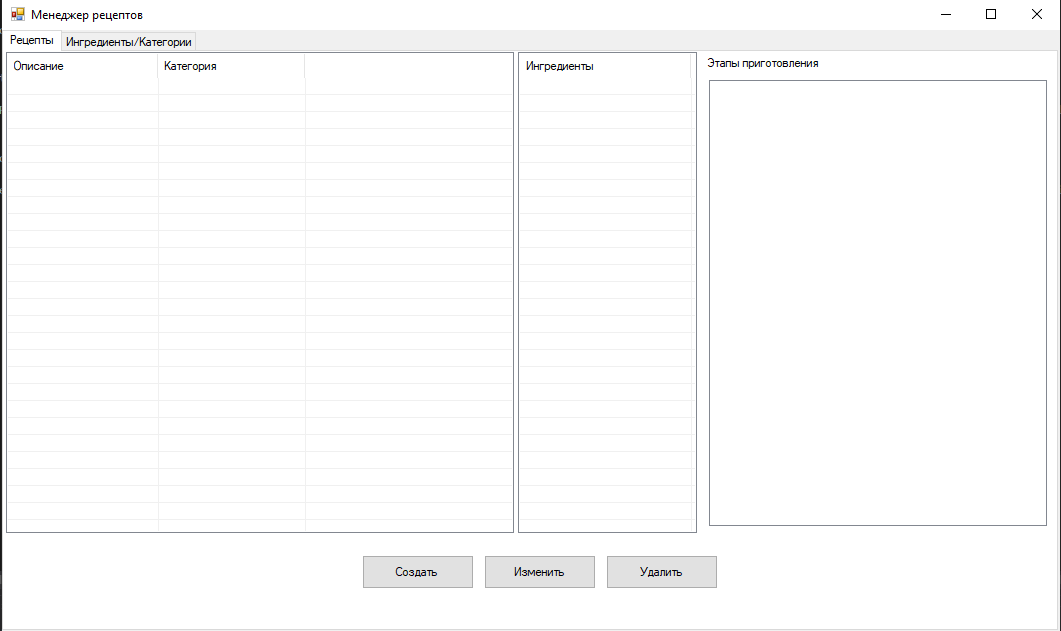


Рисунок 1.1 – Главное меню приложения

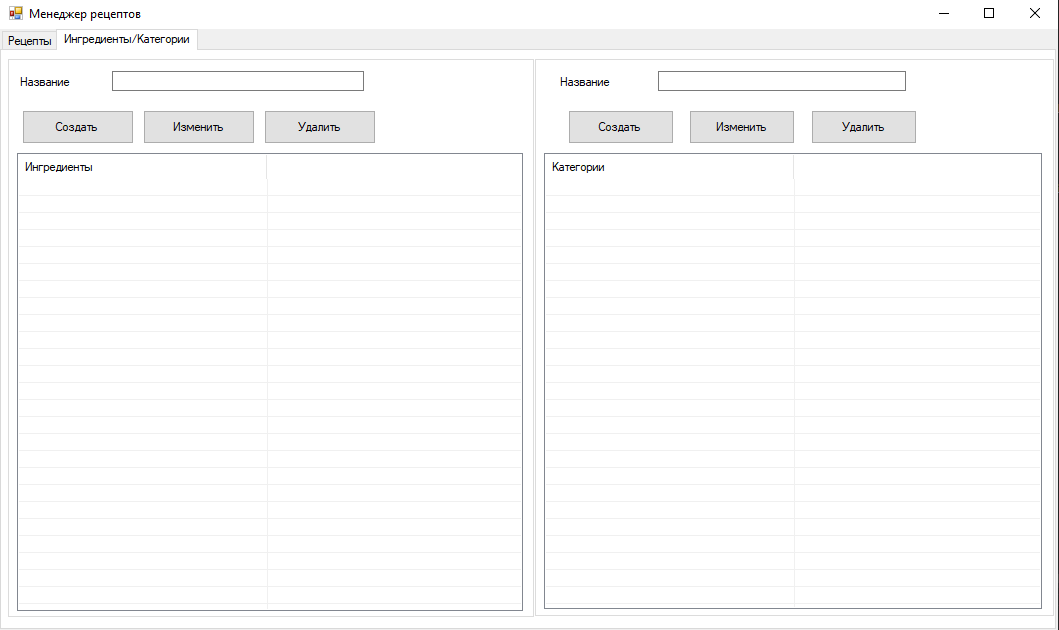


Рисунок 1.2 – Вкладка создания ингредиентов/категорий.

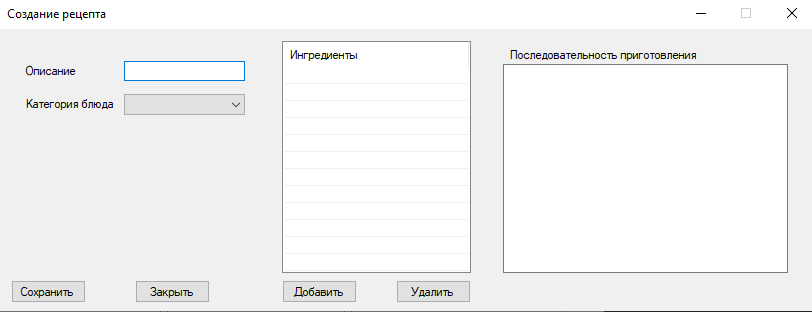


Рисунок 1.3 – Редактор создания рецептов.

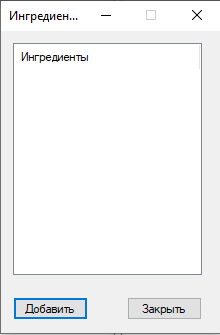


Рисунок 1.4 – Окно выбора ингредиентов в редакторе рецептов.

**Задание “Лабиринт”**

Постановка задачи

Сгенерировать и показать лабиринт, используя рандомизированную версию алгоритма Прима:

Начните с поля, все клетки которого разделены стенами.

Выберите клетку, отметьте её как часть лабиринта. Добавьте стены клетки в список стен

Пока в списке есть стены:

Выберите случайную стену из списка. Если посещена только одна из двух клеток, которые разделяет стена, то

Сделайте стену проходом и отметьте непосещенную клетку как часть лабиринта.

Добавьте соседние стены клетки в список стен.

Уберите стену из списка.

Модифицированная версия. Хотя классический алгоритм Прима использует список стен, для создания лабиринта можно использовать список соседних клеток. Если случайно выбранная клетка имеет несколько стен, которые соединяют её с существующим лабиринтом, выберите одну из стен случайным образом. Такой лабиринт будет ветвиться немного больше, чем основная на стенах версия.

Для сгенерированного лабиринта предусмотреть возможность сохранения в файл в текстовом и графическом виде. Также предусмотреть возможность пошаговой генерации.

Используемые средства и технологии

Данное приложение разрабатывалось на объектно-ориентированном языке программирования C#. Для создания графических интерфейсов с помощью платформы .NET применялась технология – Command line Interface. При реализации следовали принципам ООП и SOLID.

Структура проекта

Классы:

Cell

Структура, ролью которой выступает создания ячейки. Предназначен для взаимодействия с классом “Maze”

Maze

Класс лабиринта.

class Maze

{

public readonly Cell[,] \_cells;

public int \_width;

public int \_height;

public Stack<Cell> \_path = new Stack<Cell>();

public List<Cell> \_solve = new List<Cell>();

public List<Cell> \_visited = new List<Cell>();

public List<Cell> \_neighbours = new List<Cell>();

public Random rnd = new Random();

public Cell start;

public Cell finish;

private Maze(int width, int height)

{

Width = width;

Height = height;

Start = new Cell(1, 1, true, true);

Finish = new Cell(Width - 3, Height - 3, true, true);

Cells = new Cell[Width, Height];

}

//если ячейка нечетная по х и по у и не выходит за пределы лабиринта

**Функция создания лабиринта.**

public void MazeCreate()

{

Cells[Start.X, Start.Y] = Start;

while (\_path.Count != 0) //пока в стеке есть клетки ищем соседей и строим путь

{

\_neighbours.Clear();

GetNeighbours(\_path.Peek());

if (\_neighbours.Count != 0)

{

Cell nextCell = ChooseNeighbour(\_neighbours);

RemoveWall(\_path.Peek(), nextCell);

nextCell.IsVisited = true; //делаем текущую клетку посещенной

Cells[nextCell.X, nextCell.Y].IsVisited = true; //и в общем массиве

\_path.Push(nextCell); //затем добавляем её в стек

}

else

{

\_path.Pop();

}

}

Скриншоты приложения

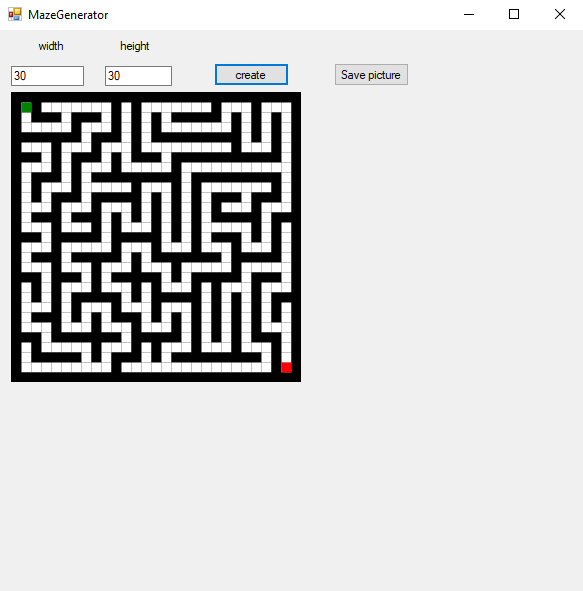


Рисунок 2.1 – Графическое отображение лабиринта.

**Задание “Лесной пожар”**

Постановка задачи

Реализовать модель лесных пожаров по Дросселю и Шваблю.

По сути, это двумерный клеточный автомат в котором каждая клетка находится в трех разных состояниях (пусто, дерево и пожар) и развивается в соответствии со следующими правилами:

Горящая клетка превращается в пустую клетку.

Дерево сгорит, если горит хотя бы один сосед.

Дерево загорается с вероятностью , даже если сосед не горит.

Пустое пространство заполняется деревом с вероятностью .

Сосед определяется из окрестности Мура для данной клетки. Граничные условия таковы, что на границе клетки всегда пусты (“фиксированные” граничные условия).

Сначала заполните поле пустыми и древовидными клетками в соответствии с определенной вероятностью (например, ячейка имеет вероятность 0.5 быть деревом). Затем позвольте системе развиваться. Реализация модель должна иметь графический интерфейс, в котором будет отображаться развитие системы.

**Используемые средства и технологии**

Данное приложение разрабатывалось на объектно-ориентированном языке программирования C#. Для создания графических интерфейсов с помощью платформы .NET применялась технология – Command line Interface. При реализации следовали принципам ООП и SOLID.

**Структура проекта**

Классы:

ForestModel

Класс с пошаговой реализацией модели лесных пожаров по Дросселю и Шваблю.

Result

Обобщенный класс обработчик. Позволяет безопасно создавать методы с выводом полученных ошибок. Предназначен для инициализации классов.

Например:

Класс Program

public static Result<bool> ParserValue()

{

var errors = new List<string>();

if (!IsDigitsOnlyAndPoint(ConfigurationManager.AppSettings["WoodFireRisk"])) { errors.Add("Поле WoodFireRisk не удалось преобразовать в float"); }

if (!IsDigitsOnlyAndPoint(ConfigurationManager.AppSettings["FillingWithWood"])) { errors.Add("Поле FillingWithWood не удалось преобразовать в float"); }

if (!IsDigitsOnly(ConfigurationManager.AppSettings["HeightMap"])) { errors.Add("Поле HeightMap не удалось преобразовать в int"); }

if (!IsDigitsOnly(ConfigurationManager.AppSettings["WidthMap"])) { errors.Add("Поле WidthMap не удалось преобразовать в int"); }

if (errors.Any())

{

return Result<bool>.Fail(errors);

}

CultureInfo ci = (CultureInfo)CultureInfo.CurrentCulture.Clone();

ci.NumberFormat.CurrencyDecimalSeparator = ".";

f = float.Parse(ConfigurationManager.AppSettings["WoodFireRisk"], NumberStyles.Any, ci);

p = float.Parse(ConfigurationManager.AppSettings["FillingWithWood"], NumberStyles.Any, ci);

height = int.Parse(ConfigurationManager.AppSettings["HeightMap"]);

width = int.Parse(ConfigurationManager.AppSettings["WidthMap"]);

return Result<bool>.Success(true);

}

Скриншоты приложения

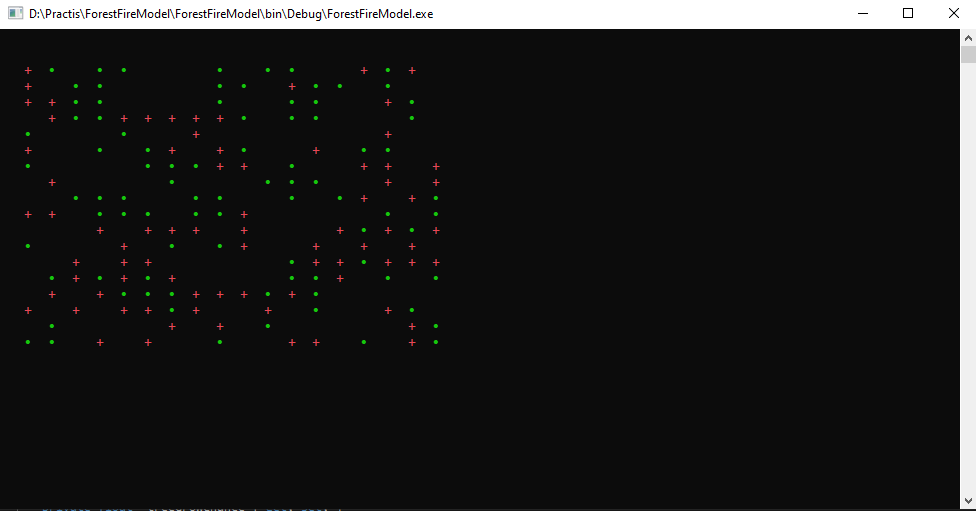


Рисунок 3.1 – Консольное окно с выводом псевдо-графики

**Вывод**

В данной практической работе были реализованы следующие задачи “Менеджер рецептов”, “Лабиринт”, “Лесной пожар”. В ходе работы был выполнен план по данным задачам, а также реализована их документация.

В итоге, за это время был получен важнейший практический опыт и рекомендации, которые помогли справиться с трудностями в задачах.

«\_\_» \_\_\_\_\_\_\_\_ 20\_\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Липчанская А.В./