Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

по курсу «Логика и основы алгоритмизации

в инженерных задачах»

на тему: «Разработка игрового агента для игры “Лабиринт”»

Выполнил:

Студент группы 22ВВВ3

Бормотов А.А.

Приняла

Юрова О.В.

**Содержание**

Реферат ............................................................................................................... 5

Введение............................................................................................................. 6

Постановка задачи............................................................................................. 7

Теоретическая часть задания ........................................................................... 8

Описание алгоритма программы...................................................................... 9

Описание программы....................................................................................... 16

Тестирование .................................................................................................... 28

Ручной расчѐт задачи........................................................................................ 29

Заключение ....................................................................................................... 30

Приложение A. Листинг программы ............................................................. 31

**Реферат**

Отчет 60 стр, 17 рисунков.

РАЗРАБОТКА ИГРОВОГО АГЕНТА ДЛЯ ИГРЫ “ЛАБИРИНТ”.

Цель исследования – разработка программы, способной находить путь к выходу из них, используя алгоритм поиска в глубину.

В работе рассмотрены правила поиска в глубину, на основе которых находится путь к выходу из лабиринта. Установлено, что с помощью данного алгоритма можно исследовать лабиринт, к структуре которого нет доступа, каким-то автоматизированным устройством.

**Введение**

Алгоритм поиска (или обхода) в глубину (англ. depth-firstsearch, DFS)

позволяет построить обход ориентированного или неориентированного графа, при котором посещаются все вершины, доступные из начальной вершины.

Отличие поиска в глубину от поиска в ширину заключается в том, что (в случае неориентированного графа) результатом алгоритма поиска в глубину является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины. Этим он принципиально отличается от поиска в ширину, где одновременно обрабатывается множество вершин, в поиске в глубину в каждый момент исполнения алгоритма обрабатывается только одна вершина.

С другой стороны, поиск в глубину не находит кратчайших путей, зато он применим в ситуациях, когда граф неизвестен целиком, а исследуется каким-то автоматизированным устройством.

Если же граф ориентированный, то поиск в глубину строит дерево путей из начальной вершины во все доступные из нее.

В качестве среды разработки мною была выбрана среда

MicrosoftVisualStudio2022, язык программирования – C#.

Целью данной курсовой работы является разработка программы на языке C#, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска в глубину, осуществляющий поиск пути к выходу из лабиринта.

**Постановка задачи**

Требуется разработать программу, которая будет представлять собой игру “Лабиринт” с реализованным в ней игровым агентом, который найдет путь к выходу из лабиринта, анализируя входные данные.

Лабиринт в программе должен представлять из себя множество, причем при его генерации должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь мог выбрать режим игры, применяющих разные способы загрузки лабиринта. После обработки его выбора на экран должен выводиться лабиринт, который игровой агент должен пройти. После, этот же лабиринт должен пройти и пользователь. После победы пользователя, данные о прошедшей игре сохраняются. Необходимо предусмотреть различные исходы действий игрока, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура.

**Теоретическая часть**

**Игровой агент** — программа или часть программы, имитирующая действия игрока в компьютерных играх: сетевых поединках, командных сражениях и т. д. В основе ботов лежит модуль искусственного интеллекта, адаптированный к особенностям данной игры: карте, правилам, а также другим тонкостям геймплея.

Игровые агенты могут помочь игроку потренироваться в точности своих действий, добавив ему мастерства в игре перед тем, как он будет играть в многопользовательскую игру. Некоторые PC игроки предпочитают играть исключительно с игровыми агентами. Из-за маленькой скорости соединения с интернетом (к примеру, dial-up соединение) они не могут играть онлайн. Игровые агенты не заставляют игрока беспокоиться о честности соперника, который может использовать чит-коды или баги игры. В этом отношении, боты продлевают интерес геймера к игре. Также для некоторых игроков игровые агенты могут стать целью, которую нужно превзойти, например, пройти уровень игры против игрового агента на самом высоком уровне сложности.

**Описание алгоритма программы**

Для программной реализации алгоритма понадобиться набор исходных данных о структуре лабиринта. В нашем случае это массив, содержащий в себе данные о стенах и выходе из лабиринта. Массив генерируется с помощью алгоритма Эллера, дублируется, после чего происходит поиск выхода из лабиринта игровым агентом, с помощью алгоритма поиска в глубину, и подсчет времени его выполнения. Затем тот же лабиринт проходит игрок. В случае, если время прохождения лабиринта игроком больше чем время прохождения лабиринта игровым агентом, массив сохраняется в файл в формате .txt со случайным именем. Иначе вся информация о прошедшей игре стирается.

Ниже представлен псевдокод функций Maze\_Generate() и Next\_Step():

**Maze\_Generate(Width, Height)**

1. если Width < 1 или Height < 1

2. вернуть null

1. конец условия
2. Max\_Limit = максимально возможное значение целочисленного числа без знака
3. если (Max\_Limit – 1) / 2 <= Width или (Max\_Limit – 1) / 2 <= Height

6. вернуть null

7. конец условия

8. Output\_Height = Height \* 2 + 1

9. Output\_Width = Width \* 2 + 1

10. выделить память для динамического массива Maze\_Ptr

11. размер Maze\_Ptr = Output\_Height

12. для I\_Index = 0, пока I\_Index < Output\_Height

13. выделить память для динамического массива Row

14. размер Row = Output\_Width

15. для J\_Index = 0, пока J\_Index < Output\_Width

16. если I\_Index % 2 == 1 и J\_Index % 2 == 1

17. добавить в Row пробел

18. конец условия

19. иначе

20. если (I\_Index % 2 == 1 и J\_Index % 2 == 0 и J\_Index != 0 и J\_Index != Output\_Width - 1) или (J\_Index % 2 == 1 и I\_Index % 2 == 0 и I\_Index != 0 и I\_Index != Output\_Height - 1)

21. добавить в Row пробел

22. конец условия

23. иначе

24. добавить в Row пробел

25. конец условия

26. конец условия

27. конец цикла

28. добавить Row в Maze\_Ptr

29. конец цикла

30. выделить память под динамический массив Row\_Set

31. размер Row\_Set = Width

32. для Index = 0, пока Index < Width

33. добавить в Row\_Set ноль

34. конец цикла

35. Set = 1

36. инициализировать RNG

37. для I\_Index = 0, пока I\_Index < Height

38. для J\_Index = 0, пока J\_Index < Width

39. если Row\_Set[J\_Index] == 0

40. Row\_Set[J\_Index] = Set++;

41. конец условия

42. конец цикла

43. для J\_Index = 0, пока J\_Index < Width – 1

44. Right\_Wall = случайное число от 0 до 1

45. если Right\_Wall == 1 или Row\_Set[J\_Index] == Row\_Set[J\_Index + 1]

46. Maze\_Ptr[I\_Index \* 2 + 1][J\_Index \* 2 + 2] = '#'

47. конец условия

48. иначе

49. Changing\_Set = Row\_Set[J\_Index + 1]

50. для L\_Index = 0, пока L\_Index < Width

51. если Row\_Set[L\_Index] == Changing\_Set

52. Row\_Set[L\_Index] == Row\_Set[J\_Index]

53. конец условия

54. конец цикла

55. конец условия

56. для J\_Index = 0, пока J\_Index < Width

57. Bottom\_Wall = случайное число от 0 до 1

58. Current\_Set\_Count = 0

59. для L\_Index = 0, пока L\_Index < Width

60. если Row\_Set[J\_Index] == Row\_Set[L\_Index]

61. Current\_Set\_Count++

62. конец условия

63. конец цикла

64. если Bottom\_Wall == 1 и Current\_Set\_Count != 1

65. Maze\_Ptr[I\_Index \* 2 + 2][J\_Index \* 2 + 1] = '#'

66. конец условия

67. конец цикла

68. если I\_Index != Height – 1

69. для J\_Index = 0, пока J\_Index < Width

70. Passage\_Count = 0

71. для L\_Index = 0, пока L\_Index < Width

72. если Row\_Set[L\_Index] == Row\_Set[J\_Index] и

Maze\_Ptr[I\_Index \* 2 + 2][L\_Index \* 2 + 1] == ' '

73. Passage\_Count++

74. конец условия

75. конец цикла

76. если Passage\_Count == 0

77. Maze\_Ptr[I\_Index \* 2 + 2][J\_Index \* 2 + 1] = ' '

78. конец условия

79. конец цикла

80. для J\_Index = 0, пока J\_Index < Width

81. если Maze\_Ptr[I\_Index \* 2 + 2][J\_Index \* 2 + 1] == '#'

82. Row\_Set[J\_Index] = 0

83. конец условия

84. конец цикла

85. конец условия

86. конец цикла

87. для J\_Index = 0, пока J\_Index < Width – 1

88. если Row\_Set[J\_Index] != Row\_Set[J\_Index + 1]

89. Maze\_Ptr[Output\_Height - 2][J\_Index \* 2 + 2] = ' '

90. конец условия

91. конец цикла

92. Maze\_Ptr[1][Width \* 2 - 1] = 'X'

93. вернуть Maze\_Ptr

**Next\_Step()**

1. если Is\_Win\_

2. вернуть

3. конец условия

4. если Current\_Direction == UP

5. Current\_Direction = LEFT

6. конец условия

7. иначе если Current\_Direction == LEFT

8. Current\_Direction = DOWN

9. конец условия

10. иначе если Current\_Direction == DOWN

11. Current\_Direction = RIGHT

12. конец условия

13. иначе

14. Current\_Direction = UP

15. конец условия

16. инициализировать Forward\_Cell

17. делать, пока Maze\_Ptr\_[Forward\_Cell.Y][Forward\_Cell.X] == '#'

18. Forward\_Cell = Current\_Position

19. если Current\_Direction == RIGHT

Forward\_Cell.X++

20. конец условия

21. если Current\_Direction == DOWN

22. Forward\_Cell.Y++

23. конец условия

24. если Current\_Direction == LEFT

25. Forward\_Cell.X—

26. конец условия

27. если Current\_Direction == UP

28. Forward\_Cell.Y--

29. конец условия

30. если Current\_Direction == UP

31. Current\_Direction = RIGHT

32. конец условия

33. иначе если Current\_Direction == LEFT

34. Current\_Direction = UP

35. конец условия

36. иначе если Current\_Direction == DOWN

37. Current\_Direction = LEFT

38. конец условия

39. иначе

40. Current\_Direction = DOWN

41. конец условия

42. конец цикла

43. если Current\_Direction == UP

44. Current\_Direction = LEFT

45. конец условия

46. иначе если Current\_Direction == LEFT

47. Current\_Direction = DOWN

48. конец условия

49. иначе если Current\_Direction == DOWN

50. Current\_Direction = RIGHT

51. конец условия

52. иначе

53. Current\_Direction = UP

54. конец условия

55. Previous\_Position = Current\_Position

56. Current\_Position = Forward\_Cell

57. Is\_Win\_ = Maze\_Ptr\_[Current\_Position.Y][Current\_Position.X] == 'X'

58. Maze\_Ptr\_[Current\_Position.Y][Current\_Position.X] = 'B';

59. Maze\_Ptr\_[Previous\_Position.Y][Previous\_Position.X] = 'a';

**Описание программы**

Для написания данной программы использован язык C#. Данный язык был выбран из-за его универсальности, а также большого количества библиотек, позволяющих реализовать разнообразный функционал.

Проект был создан в виде консольного приложения, имеет графический интерфейс.

Работа программы начинается c экрана заставки, где пользователь может выйти нажав Esc или продолжить нажав Enter.

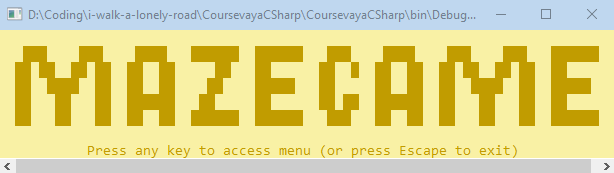


Рисунок 1 – экран заставки

Если пользователь решил продолжить, открывается экран главного меню, на котором пользователю предлагается выбор из 4 опций.

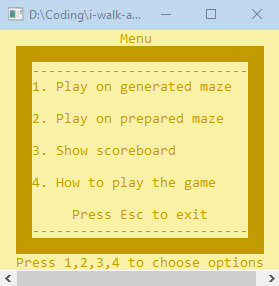


Рисунок 2 – экран главного меню

Если пользователь выбрал 1 опцию, запускается режим игры, в котором лабиринт генерируется случайно, первым его начинает проходить игрок, поэтому открывается экран готовности, на котором игроку нужно нажать Enter чтобы игра началась.

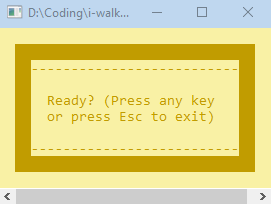


Рисунок 3 – экран готовности

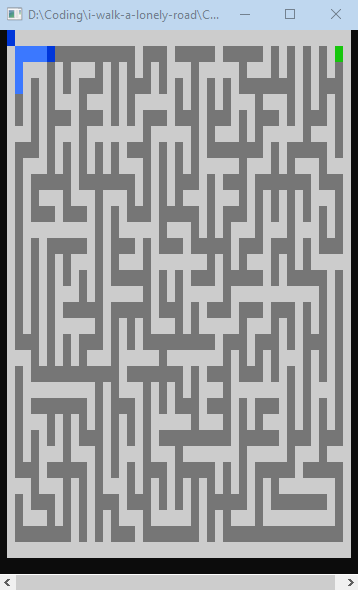


Рисунок 4 – лабиринт с игроком

После того как пользователь дошел до выхода очередь проходить лабиринт настает для игрового агента.

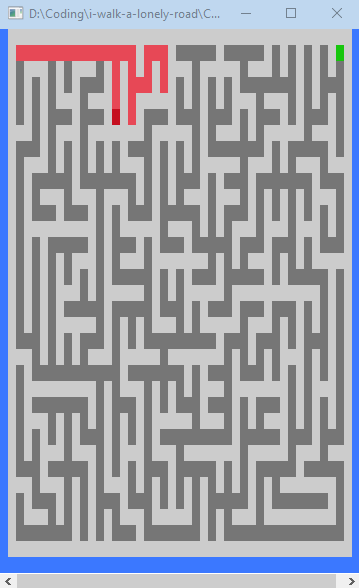


Рисунок 5 – лабиринт с игровым агентом



Рисунок 6 – экран затраченного времени

Когда игровой агент доходит до выхода из лабиринта, игра заканчивается, затем показывается экран, отображающий затраченное на прохождение время, и если пользователь прошел лабиринт быстрее, то он переходит на следующий уровень. В этом режиме игры всего 3 уровня, они различаются по размеру лабиринта.

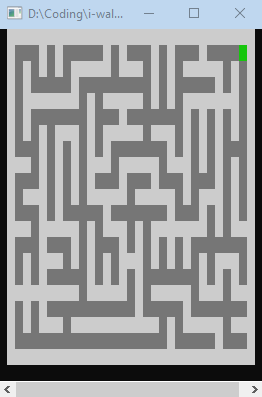


Рисунок 7 – лабиринт 1 уровня

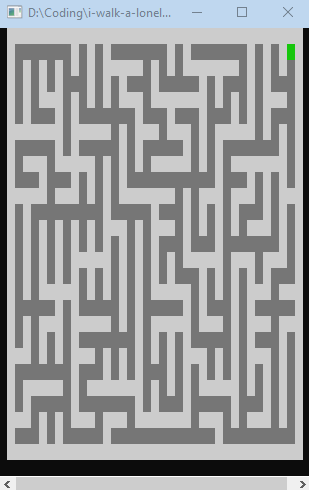


Рисунок 8 – лабиринт 2 уровня

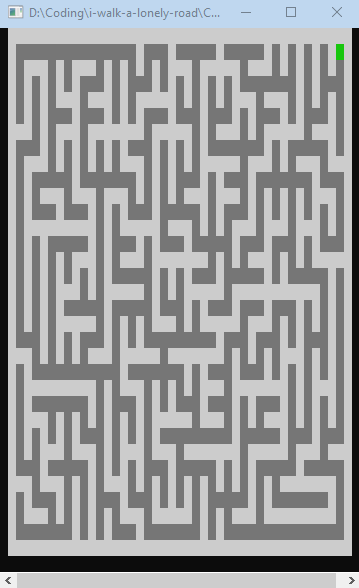


Рисунок 9 – лабиринт 3 уровня

Если пользователь прошел все три уровня, то открывается экран сохранения результата, на котором ему предлагается записать свой никнейм. Затем открывается экран статистики, на котором отображены 10 последних результатов.

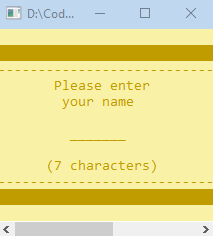


Рисунок 10 – экран сохранения результата

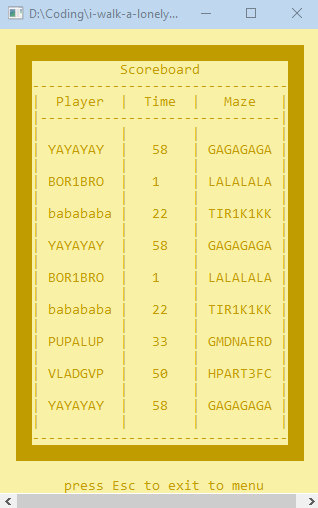


Рисунок 11 – экран статистики

Если пользователь не смог пройти лабиринт быстрее игрового агента на любом из уровней, то выводится экран проигрыша, после чего пользователь может вернуться в главное меню нажав Enter.



Рисунок 12 – экран проигрыша

Если в главном меню пользователь выбрал 2 опцию, открывается экран выбора лабиринтов для прохождения, они отображаются списком по 4 названия на страницу. При выборе лабиринта открывается экран подтверждения выбора, где пользователь может нажать Enter для продолжения и Esc для отмены.



Рисунок 13 – экран выбора лабиринтов



Рисунок 14 – экран подтверждения выбора

При окончательном выборе карты, открывается экран готовности, при подтверждении проходить лабиринт начинает пользователь. Когда пользователь доходит до выхода, открывается экран затраченного времени, и после нажатия пользователем клавиши Enter, этот же лабиринт проходит игровой агент. Если пользователь прошел лабиринт быстрее, то открывается экран сохранения результата, если игровой агент прошел быстрее – экран проигрыша, после чего пользователь возвращается в главное меню.

Если пользователь в главном меню выбрал 3 опцию, открывается экран статистики.

Если пользователь в главном меню выбрал 4 опцию, открывается экран информации об игре, из которого пользователь может выйти нажатием на клавишу Esc.

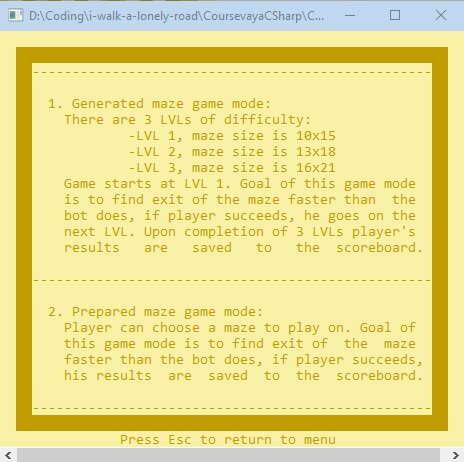


Рисунок 15 – экран информации об игре

Если пользователь в главном меню нажал на Esc, то открывается предупреждение, и если пользователь выбрал продолжить, то программа закончит свою работу.

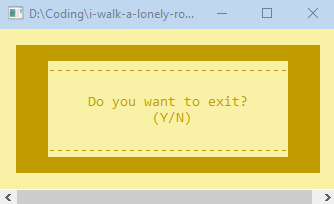


Рисунок 16 – экран предупреждения о выходе

**Тестирование**

Среда разработки MicrosoftVisualStudio2022 предоставляет все средства необходимые при разработке и отладке консольной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Корректный запуск основного окна | Верно |
| Выбор пунктов меню | Выбранные пункты выводят соответствующие им окна | Верно |
| Управление игроком | Игрок не должен проходить сквозь стены лабиринта | Верно |
| Сохранение лабиринта в файл | Корректное сохранение лабиринта в файл | Верно |
| Загрузка лабиринта из файла | Корректная загрузка лабиринта из файла | Верно |
| Сохранение имени игрока при его победе | Сохранение имени длиной до 7 знаков включительно | Верно |
| Вывод окна статистики | Корректное отображение всех сохраненных данных статистики | Верно |

В результате тестирования было выявлено, что программа успешно выполняет свои функции.

**Ручной расчет**

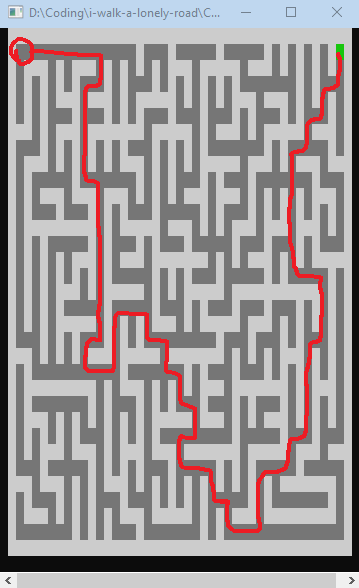


Рисунок 17 – ручной расчет

Таким образом можно сделать вывод, что программа работает верно.

**Заключение**

Таким образом, в процессе создания данного проекта разработана

программа, реализующая нахождение выхода из лабиринта.

При выполнении данной курсовой работы были получены навыки

разработки программ и освоены приемы анализа и обработки данных.

Углублены знания языка программирования C#.

**Приложение А.**

**Листинг программы.**

**Класс MazeFileManagement.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

namespace CoursevayaCSharp

{

internal class MazeFileManagement

{

/// <summary>

/// Function to load maze from file

/// </summary>

public static List<List<char>> Maze\_File\_Load(string File\_Name)

{

List<List<char>> Maze\_Ptr = new();

string Input = File.ReadAllText(@"Saved-Mazes/" + File\_Name + ".txt");

Maze\_Ptr.Capacity = 33;

int K\_Index = 0;

for (int I\_Index = 0; I\_Index < Maze\_Ptr.Capacity; ++I\_Index)

{

//if (K\_Index < 1419)

//{

List<char> Row = new()

{

Capacity = 43

};

for (int J\_Index = 0; J\_Index < Row.Capacity; ++J\_Index)

{

Row.Add(Input[K\_Index]);

K\_Index++;

}

Maze\_Ptr.Add(Row);

//}

}

return Maze\_Ptr;

}

/// <summary>

/// Function to save maze on which player was victorious

/// </summary>

public static string Maze\_File\_Save(List<List<char>> Maze\_Ptr, bool Option)

{

//Generating random file name

var PathName = Random\_File\_Name(8);

int K\_Index = 0;

char[] Maze\_Ptr\_String = new char[Maze\_Ptr.Count\*Maze\_Ptr[0].Count];

for (int I\_Index = 0; I\_Index < Maze\_Ptr.Count; ++I\_Index)

{

for (int J\_Index = 0; J\_Index < Maze\_Ptr[0].Count; ++J\_Index)

{

Maze\_Ptr\_String[K\_Index] = Maze\_Ptr[I\_Index][J\_Index];

K\_Index++;

}

}

if (Option)

{

using FileStream FileStream = File.OpenWrite(@"Saved-Mazes/" + PathName + ".txt");

Byte[] Info = new UTF8Encoding(true).GetBytes(Maze\_Ptr\_String);

//Writing maze into the file

FileStream.Write(Info, 0, Info.Length);

}

return PathName;

}

/// <summary>

/// Function to generate random file numbers (to save mazes)

/// </summary>

/// <returns>Random file name(without .txt)</returns>

public static string Random\_File\_Name(int Length)

{

const string Chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

return new string(Enumerable.Repeat(Chars, Length)

.Select(String => String[Random\_Name.Next(String.Length)]).ToArray());

}

/// <summary>

/// Function to get names (without extension) of all files from Saved-Mazes directory

/// </summary>

/// <returns>Array of names</returns>

public static string[] Load\_Maze\_File\_Names()

{

string[] Files = Directory.GetFiles(@"Saved-Mazes");

string[] File\_Names = new string[Files.Length];

for (int File\_Index = 0; File\_Index < Files.Length; File\_Index++)

{

File\_Names[File\_Index] = Path.GetFileNameWithoutExtension(Files[File\_Index]);

}

return File\_Names;

}

protected static Random Random\_Name = new();

}

}

**Класс MazeGenerator.cs**

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

namespace CoursevayaCSharp

{

public static class MazeGenerator

{

/// <summary>

/// Function to generate maze structure

/// </summary>

/// <param name="Width">Width of the maze</param>

/// <param name="Height">Height of the maze</param>

/// <returns>Generated maze</returns>

public static List<List<char>> Maze\_Generate(int Width, int Height)

{

//Restriction for parameters by 0

if ((Width < 1) || (Height < 1))

{

return null;

}

var Max\_Limit = uint.MaxValue;

//Restriction for parameters by max size

if (((Max\_Limit - 1) / 2 <= Width) || ((Max\_Limit - 1) / 2 <= Height))

{

return null;

}

//Initialize size of output maze matrix

//Every cell are fragments 2x2, + 1 up and left for walls

int Output\_Height = Height \* 2 + 1;

int Output\_Width = Width \* 2 + 1;

//Initialize ptr to maze

var Maze\_Ptr = new List<List<char>>();

//Reserve size of maze by height

Maze\_Ptr.Capacity = Output\_Height;

//Loop for initializing empty maze row by row

for (int I\_Index = 0; I\_Index < Output\_Height; ++I\_Index)

{

var Row = new List<char>();

Row.Capacity = Output\_Width;

for (int J\_Index = 0; J\_Index < Output\_Width; ++J\_Index)

{

if ((I\_Index % 2 == 1) && (J\_Index % 2 == 1))

{

Row.Add(' ');

}

else

{

if (((I\_Index % 2 == 1) && (J\_Index % 2 == 0) && (J\_Index != 0) && (J\_Index != Output\_Width - 1)) ||

((J\_Index % 2 == 1) && (I\_Index % 2 == 0) && (I\_Index != 0) && (I\_Index != Output\_Height - 1)))

{

Row.Add(' ');

}

else

{

Row.Add('#');

}

}

}

Maze\_Ptr.Add(Row);

}

//I. Creating first row of a maze, any cell is not in any set

//Initialize extra string which will save attachment to any set

var Row\_Set = new List<uint>();

Row\_Set.Capacity = Width;

//"0" means that cell is not in any set

for (int Index = 0; Index < Width; ++Index)

{

Row\_Set.Add(0);

}

//Counter for sets

uint Set = 1;

//Initialize RNG

Random Random = new Random();

//Loop for Eller's algorithm

for (int I\_Index = 0; I\_Index < Height; ++I\_Index)

{

//II. for each cell that is not in a set, create individual set

for (int J\_Index = 0; J\_Index < Width; ++J\_Index)

{

if (Row\_Set[J\_Index] == 0)

{

Row\_Set[J\_Index] = Set++;

}

}

//III. Create right walls for cells, moving from left to right

for (int J\_Index = 0; J\_Index < Width - 1; ++J\_Index)

{

//"0" - no wall, "1" - place wall, choosing randomly

var Right\_Wall = Random.Next(2);

//If current cell and cell to the right are in the same set, place wall between (excludes loops)

if ((Right\_Wall == 1) || (Row\_Set[J\_Index] == Row\_Set[J\_Index + 1]))

{

Maze\_Ptr[I\_Index \* 2 + 1][J\_Index \* 2 + 2] = '#';

}

else

{

//If no wall, merge sets of current cell and cell to the right

var Changing\_Set = Row\_Set[J\_Index + 1];

for (int L\_Index = 0; L\_Index < Width; ++L\_Index)

{

if (Row\_Set[L\_Index] == Changing\_Set)

{

Row\_Set[L\_Index] = Row\_Set[J\_Index];

}

}

}

}

//IV. Create bottom walls, moving from left to right

for (int J\_Index = 0; J\_Index < Width; ++J\_Index)

{

//"0" - no wall, "1" - place wall, choosing randomly

var Bottom\_Wall = Random.Next(2);

//If cell is single in it's set, don't place wall

uint Current\_Set\_Count = 0;

for (int L\_Index = 0; L\_Index < Width; ++L\_Index)

{

//Current set cell count

if (Row\_Set[J\_Index] == Row\_Set[L\_Index])

{

Current\_Set\_Count++;

}

}

//If cell is single in it's set and don't have bottom wall, don't place wall

if ((Bottom\_Wall == 1) && (Current\_Set\_Count != 1))

{

Maze\_Ptr[I\_Index \* 2 + 2][J\_Index \* 2 + 1] = '#';

}

}

//V. Decide, whether to add more rows or stop and finish maze

//Check that every set have atleast one cell without a wall (unless it's last row)

if (I\_Index != Height - 1)

{

for (int J\_Index = 0; J\_Index < Width; ++J\_Index)

{

uint Passage\_Count = 0;

for (int L\_Index = 0; L\_Index < Width; ++L\_Index)

{

if ((Row\_Set[L\_Index] == Row\_Set[J\_Index]) && (Maze\_Ptr[I\_Index \* 2 + 2][L\_Index \* 2 + 1]) == ' ')

{

Passage\_Count++;

}

}

if (Passage\_Count == 0)

{

Maze\_Ptr[I\_Index \* 2 + 2][J\_Index \* 2 + 1] = ' ';

}

}

for (int J\_Index = 0; J\_Index < Width; ++J\_Index)

{

//Check if there's wall at the bottom of current row

if (Maze\_Ptr[I\_Index \* 2 + 2][J\_Index \* 2 + 1] == '#')

{

//If wall, delete cell from set

Row\_Set[J\_Index] = 0;

}

}

}

}

//If finishing maze, add bottom wall for every cell, from left to right

for (int J\_Index = 0; J\_Index < Width - 1; ++J\_Index)

{

//If current cell and cell to the right are in the different sets, then

if (Row\_Set[J\_Index] != Row\_Set[J\_Index + 1])

{

//Delete right wall

Maze\_Ptr[Output\_Height - 2][J\_Index \* 2 + 2] = ' ';

}

}

//Place exit

Maze\_Ptr[1][Width \* 2 - 1] = 'X';

//Output ptr on result maze

return Maze\_Ptr;

}

/// <summary>

/// Serialization method to make clone of the maze

/// </summary>

/// <returns>Deep copy of desired object</returns>

public static List<List<char>> Maze\_Copy(this List<List<char>> Maze\_Ptr)

{

BinaryFormatter Formatter = new BinaryFormatter();

MemoryStream Stream = new MemoryStream();

Formatter.Serialize(Stream, Maze\_Ptr);

Stream.Position = 0;

return (List<List<char>>)Formatter.Deserialize(Stream);

}

/// <summary>

/// Function to draw maze in console

/// </summary>

public static void Maze\_Print(List<List<char>> Maze\_Ptr, int Cursor\_Position)

{

//Check ptr for null

if (Maze\_Ptr == null)

{

return;

}

//Scanning maze row by row and outputting to the console

for (int I\_Index = 0; I\_Index < Maze\_Ptr.Count(); ++I\_Index)

{

Console.SetCursorPosition(Cursor\_Position, I\_Index);

for (int J\_Index = 0; J\_Index < Maze\_Ptr[0].Count(); ++J\_Index)

{

Symbol\_Check(Maze\_Ptr[I\_Index][J\_Index]);

Console.Write(Maze\_Ptr[I\_Index][J\_Index]);

Console.ResetColor();

}

Console.WriteLine();

}

}

/// <summary>

/// Function to draw player's trail in console

/// </summary>

/// <param name="Prev\_Buf\_Maze\_Ptr">Maze before changes</param>

/// <param name="Buf\_Maze\_Ptr">Maze after changes</param>

/// <returns>Maze after changes</returns>

public static List<List<char>> Maze\_Print(List<List<char>> Prev\_Buf\_Maze\_Ptr, List<List<char>> Buf\_Maze\_Ptr)

{

//Check ptr for null

if (Buf\_Maze\_Ptr == null)

{

return null;

}

//Scanning maze row by row and outputting to the console only differences

for (int I\_Index = 0; I\_Index < Buf\_Maze\_Ptr.Count(); ++I\_Index)

{

for (int J\_Index = 0; J\_Index < Buf\_Maze\_Ptr[0].Count(); ++J\_Index)

{

if (Buf\_Maze\_Ptr[I\_Index][J\_Index] != Prev\_Buf\_Maze\_Ptr[I\_Index][J\_Index])

{

Console.SetCursorPosition(J\_Index + 1, I\_Index);

Symbol\_Check(Buf\_Maze\_Ptr[I\_Index][J\_Index]);

Console.Write(Buf\_Maze\_Ptr[I\_Index][J\_Index]);

//Is needed to evade erasing console char's next to one that is currently drawn

Console.SetCursorPosition(J\_Index + 1, I\_Index);

}

}

}

Prev\_Buf\_Maze\_Ptr = Maze\_Copy(Buf\_Maze\_Ptr);

return Prev\_Buf\_Maze\_Ptr;

}

/// <summary>

/// Function to visualize game

/// </summary>

/// <param name="Symbol">Checks symbol type to apply rendering on it</param>

public static void Symbol\_Check(char Symbol)

{

switch (Symbol)

{

case '#':

{

Console.BackgroundColor = ConsoleColor.Gray;

Console.ForegroundColor = ConsoleColor.Gray;

break;

}

case ' ':

{

Console.BackgroundColor = ConsoleColor.DarkGray;

break;

}

case 'P':

{

Console.BackgroundColor = ConsoleColor.DarkBlue;

Console.ForegroundColor = ConsoleColor.DarkBlue;

break;

}

case 'o':

{

Console.BackgroundColor = ConsoleColor.Blue;

Console.ForegroundColor = ConsoleColor.Blue;

break;

}

case 'B':

{

Console.BackgroundColor = ConsoleColor.DarkRed;

Console.ForegroundColor = ConsoleColor.DarkRed;

break;

}

case 'a':

{

Console.BackgroundColor = ConsoleColor.Red;

Console.ForegroundColor = ConsoleColor.Red;

break;

}

case 'X':

{

Console.BackgroundColor = ConsoleColor.Green;

Console.ForegroundColor = ConsoleColor.Green;

break;

}

case '|':

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.Yellow;

break;

}

case 'Y':

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

break;

}

}

}

}

}

**Класс PathFinder.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace CoursevayaCSharp

{

//Initializing directions enum

enum Direction2D

{

RIGHT,

DOWN,

LEFT,

UP,

};

// Initialize position struct

struct Position2D

{

public int X;

public int Y;

//Position2D constructor to initialize starting position

public Position2D(int X\_Pos, int Y\_Pos)

{

X = X\_Pos;

Y = Y\_Pos;

}

}

internal class PathFinder

{

/// <summary>

/// Function to initialize the player

/// </summary>

/// <param name="Start\_Position"></param>

/// <param name="Maze\_Ptr"></param>

/// <returns>Maze with player at start position</returns>

public List<List<char>> Init\_Player(Position2D Start\_Position, List<List<char>> Maze\_Ptr)

{

//Maze data to check every step

Maze\_Ptr\_ = Maze\_Ptr;

//At the start we are not at the exit

Is\_Win\_ = false;

//1. Initializing player's start position and direction

Current\_Position = Start\_Position;

Current\_Direction = Direction2D.RIGHT;

//Draw player on the screen

Maze\_Ptr\_[Current\_Position.Y][Current\_Position.X] = 'P';

return Maze\_Ptr\_;

}

/// <summary>

/// Function that determines bot movement in the maze

/// </summary>

public void Next\_Step()

{

//If at the exit, don't do more steps

if (Is\_Win\_)

{

return;

}

//2. Player turns 90 degrees, counter-clockwise

Current\_Direction = Current\_Direction == Direction2D.UP ? Direction2D.LEFT :

Current\_Direction == Direction2D.LEFT ? Direction2D.DOWN :

Current\_Direction == Direction2D.DOWN ? Direction2D.RIGHT :

Direction2D.UP;

//Initialize temporary variable to check forward cell

Position2D Forward\_Cell;

do

{

//3. Check cell that the player is facing

Forward\_Cell = Current\_Position;

switch (Current\_Direction)

{

case Direction2D.RIGHT:

{

Forward\_Cell.X++;

break;

}

case Direction2D.DOWN:

{

Forward\_Cell.Y++;

break;

}

case Direction2D.LEFT:

{

Forward\_Cell.X--;

break;

}

case Direction2D.UP:

{

Forward\_Cell.Y--;

break;

}

}

//3.1. If wall:

//Player turns 90 degrees, clockwise

Current\_Direction = Current\_Direction == Direction2D.UP ? Direction2D.RIGHT :

Current\_Direction == Direction2D.LEFT ? Direction2D.UP :

Current\_Direction == Direction2D.DOWN ? Direction2D.LEFT :

Direction2D.DOWN;

//Do step 3

} while (Maze\_Ptr\_[Forward\_Cell.Y][Forward\_Cell.X] == '#');

//3.2. If empty:

//Do step 4

//Restore correct dirrection

Current\_Direction = Current\_Direction == Direction2D.UP ? Direction2D.LEFT :

Current\_Direction == Direction2D.LEFT ? Direction2D.DOWN :

Current\_Direction == Direction2D.DOWN ? Direction2D.RIGHT :

Direction2D.UP;

//4. Player steps on next empty cell

Position2D Previous\_Position = Current\_Position; //Saving previous position to mark player's trail

Current\_Position = Forward\_Cell;

//5. Check if current cell is the exit:

//5.1. If it isn't do step 2 etc

//5.2. If it is the exit - end

Is\_Win\_ = Maze\_Ptr\_[Current\_Position.Y][Current\_Position.X] == 'X';

//Marking player position

Maze\_Ptr\_[Current\_Position.Y][Current\_Position.X] = 'B';

//Marking cells investigated by player

Maze\_Ptr\_[Previous\_Position.Y][Previous\_Position.X] = 'a';

}

/// <summary>

/// Function to control player's movement across the maze

/// </summary>

public void Player\_Controller()

{

//If at the exit, don't do more steps

if (Is\_Win\_)

{

return;

}

//Saving previous position to mark player's trail

Position2D Previous\_Position = Current\_Position;

Position2D Forward\_Cell;

//Wait for input

ConsoleKeyInfo Key\_Info = Console.ReadKey();

switch (Key\_Info.Key)

{

case ConsoleKey.UpArrow:

{

Forward\_Cell = Current\_Position;

Forward\_Cell.Y--;

if (Maze\_Ptr\_[Forward\_Cell.Y][Forward\_Cell.X] != '#')

{

Current\_Position = Forward\_Cell;

}

break;

}

case ConsoleKey.DownArrow:

{

Forward\_Cell = Current\_Position;

Forward\_Cell.Y++;

if (Maze\_Ptr\_[Forward\_Cell.Y][Forward\_Cell.X] != '#')

{

Current\_Position = Forward\_Cell;

}

break;

}

case ConsoleKey.LeftArrow:

{

Forward\_Cell = Current\_Position;

Forward\_Cell.X--;

if (Maze\_Ptr\_[Forward\_Cell.Y][Forward\_Cell.X] != '#')

{

Current\_Position = Forward\_Cell;

}

break;

}

case ConsoleKey.RightArrow:

{

Forward\_Cell = Current\_Position;

Forward\_Cell.X++;

if (Maze\_Ptr\_[Forward\_Cell.Y][Forward\_Cell.X] != '#')

{

Current\_Position = Forward\_Cell;

}

break;

}

}

Is\_Win\_ = Maze\_Ptr\_[Current\_Position.Y][Current\_Position.X] == 'X';

//Marking cells investigated by player

Maze\_Ptr\_[Previous\_Position.Y][Previous\_Position.X] = 'o';

//Marking player position

Maze\_Ptr\_[Current\_Position.Y][Current\_Position.X] = 'P';

}

/// <summary>

/// Function to check for victory

/// </summary>

/// <returns>Bool value</returns>

public bool isWin()

{

return Is\_Win\_;

}

//Protected fields

protected bool Is\_Win\_;

protected Position2D Current\_Position;

protected Direction2D Current\_Direction;

//Ptr on maze where we go

protected List<List<char>> Maze\_Ptr\_;

}

}

**Класс Scoreboard.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Media;

using System.Text;

using System.Threading.Tasks;

using static CoursevayaCSharp.ScreenVisual;

namespace CoursevayaCSharp

{

public struct PlayerInfo

{

public string Name;

public long Time;

public string Maze\_Name;

public PlayerInfo()

{

}

}

internal class Scoreboard

{

public static void Scoreboard\_Save(PlayerInfo Player\_Info)

{

if (!File.Exists(@"Scoreboard.txt"))

{

File.Create(@"Scoreboard.txt");

StreamWriter Scoreboard\_File = new StreamWriter(@"Scoreboard.txt", true);

Scoreboard\_File.Write(Player\_Info.Name + " ");

Scoreboard\_File.Write(Player\_Info.Time + " ");

Scoreboard\_File.Write(Player\_Info.Maze\_Name + " ");

Scoreboard\_File.Close();

}

else

{

StreamWriter Scoreboard\_File = new StreamWriter(@"Scoreboard.txt", true);

Scoreboard\_File.Write(Player\_Info.Name + " ");

Scoreboard\_File.Write(Player\_Info.Time + " ");

Scoreboard\_File.Write(Player\_Info.Maze\_Name + " ");

Scoreboard\_File.Close();

}

}

public static string[] Scoreboard\_Load()

{

if (!File.Exists(@"Scoreboard.txt"))

{

return null;

}

string All\_Scoreboard\_Info = File.ReadAllText(@"Scoreboard.txt");

string[] Splitted\_Scoreboard\_Info = All\_Scoreboard\_Info.Split(' ');

return Splitted\_Scoreboard\_Info;

}

public static string Scoreboard\_Player\_Input(string Player\_Name)

{

string Buf\_Name = Console.ReadLine();

Player\_Name = Buf\_Name.Substring(0, 7);

return Player\_Name;

}

public static string[] Get\_Scoreboard\_Ten(int Ten, string[] Info)

{

int Counter = 0;

string[] Ten\_Info = new string[30];

do

{

if (Ten == Info.Length)

{

return Ten\_Info;

}

Ten\_Info[Counter] = Info[Ten];

Ten++;

Counter++;

} while (Counter != 30 && Ten < Info.Length);

return Ten\_Info;

}

public static void Leaf\_Through\_Scoreboard()

{

bool Exit = false;

int Scoreboard\_Ten = 0;

string[] Scoreboard\_Info = Scoreboard\_Load();

string[] Scoreboard\_Ten\_Info = Get\_Scoreboard\_Ten(Scoreboard\_Ten, Scoreboard\_Info);

while (!Exit)

{

//Display prepared maze screen with ten of player info's

Scoreboard\_Screen(Scoreboard\_Ten\_Info);

//Wait for input

Key\_Info = Console.ReadKey();

switch (Key\_Info.Key)

{

//if -> show next quad (if remaining files don't make a quad, display only the remainings)

case ConsoleKey.RightArrow:

{

Scoreboard\_Ten = Scoreboard\_Ten + 30;

//Check if quad goes beyond name array's range, if does fit it in this range

if (Scoreboard\_Ten > Scoreboard\_Info.Length)

{

Scoreboard\_Ten = Scoreboard\_Ten - 30;

}

Scoreboard\_Ten\_Info = Get\_Scoreboard\_Ten(Scoreboard\_Ten, Scoreboard\_Info);

break;

}

//if <- show prev quad (if it's going negative, show first quad)

case ConsoleKey.LeftArrow:

{

Scoreboard\_Ten = Scoreboard\_Ten - 30;

//Check if quad goes beyond name array's range, if does fit it in this range

if (Scoreboard\_Ten < 0)

{

Scoreboard\_Ten = Scoreboard\_Ten + 30;

}

Scoreboard\_Ten\_Info = Get\_Scoreboard\_Ten(Scoreboard\_Ten, Scoreboard\_Info);

break;

}

//If player pressed Esc - go back to main menu

case ConsoleKey.Escape:

{

Click\_Sound.Play();

Exit = true;

break;

}

}

}

}

public static SoundPlayer Click\_Sound = new(@"Sounds/CLICK.wav");

}

}

**Класс ScreenVisual.cs**

using System;

using System.Media;

using System.Collections.Generic;

using System.Diagnostics;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Windows.Input;

using System.Security.Cryptography.X509Certificates;

using System.Threading;

using System.Linq;

using static CoursevayaCSharp.MazeGenerator;

using System.Reflection;

namespace CoursevayaCSharp

{

internal class ScreenVisual

{

/// <summary>

/// Function to display title screen

/// </summary>

public static void Title\_Screen()

{

Console.SetWindowSize(77, 8);

Console.SetBufferSize(78, 8);

Console.SetWindowPosition(0, 0);

string Game\_Title = File.ReadAllText(@"Screens/MAZEGAME.txt");

for (int Index = 0; Index < Game\_Title.Length; Index++)

{

Symbol\_Check(Game\_Title[Index]);

Console.Write(Game\_Title[Index]);

}

Console.ResetColor();

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.Write("\n Press any key to access menu (or press Escape to exit) ");

Console.ResetColor();

Key\_Info = Console.ReadKey();

if (Key\_Info.Key == ConsoleKey.Escape)

{

Environment.Exit(0);

}

Console.Clear();

}

/// <summary>

/// Function to display ready screen

/// </summary>

public static void Ready\_Screen(List<List<char>> Buf\_Maze\_Ptr)

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(34, 10);

Console.SetBufferSize(35, 10);

Console.SetWindowPosition(0, 0);

string Ready = File.ReadAllText(@"Screens/READY.txt");

for (int Index = 0; Index < Ready.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Ready[Index] == 'Y')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Ready[Index]);

}

}

/// <summary>

/// Function to draw menu

/// </summary>

public static void Game\_Menu\_Screen()

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(35, 15);

Console.SetBufferSize(36, 15);

Console.SetWindowPosition(0, 0);

string Game\_Menu = File.ReadAllText(@"Screens/MAZEGAME\_MENU.txt");

for (int Index = 0; Index < Game\_Menu.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Game\_Menu[Index] == 'Y')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Game\_Menu[Index]);

}

Console.ResetColor();

}

/// <summary>

/// Function to display warning when exiting from menu

/// </summary>

public static void Exit\_Warning\_Screen()

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(42, 10);

Console.SetBufferSize(43, 10);

Console.SetWindowPosition(0, 0);

string Warning = File.ReadAllText(@"Screens/EXIT\_WARNING.txt");

for (int Index = 0; Index < Warning.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Warning[Index] == 'T')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Warning[Index]);

}

Console.ResetColor();

}

/// <summary>

/// Function to display information screen

/// </summary>

public static void Game\_Info\_Screen()

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(58, 26);

Console.SetBufferSize(59, 26);

Console.SetWindowPosition(0, 0);

string Game\_Info = File.ReadAllText(@"Screens/INFO.txt");

for (int Index = 0; Index < Game\_Info.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Game\_Info[Index] == 'Y')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Game\_Info[Index]);

}

Console.ResetColor();

}

/// <summary>

/// Function to display time elapsed on maze completion

/// </summary>

public static void Elapsed\_Time\_Screen(List<long> Elapsed\_Time\_Array)

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(28, 12);

Console.SetBufferSize(29, 12);

Console.SetWindowPosition(0, 0);

string Elapsed\_Time = File.ReadAllText(@"Screens/ELAPSED.txt");

for (int Index = 0; Index < Elapsed\_Time.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Elapsed\_Time[Index] == '.')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Elapsed\_Time[Index]);

}

Console.SetCursorPosition(15, 5);

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.Write(Elapsed\_Time\_Array[0]);

Console.SetCursorPosition(15, 7);

Console.Write(Elapsed\_Time\_Array[1]);

}

public static void Lose\_Screen()

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(28, 17);

Console.SetBufferSize(29, 17);

Console.SetWindowPosition(0, 0);

string Game\_Info = File.ReadAllText(@"Screens/LOSE.txt");

for (int Index = 0; Index < Game\_Info.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Game\_Info[Index] == '.')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Game\_Info[Index]);

}

Console.ResetColor();

}

public static void Player\_Input\_Screen()

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(27, 12);

Console.SetBufferSize(50, 12);

Console.SetWindowPosition(0, 0);

string Game\_Info = File.ReadAllText(@"Screens/INPUT.txt");

for (int Index = 0; Index < Game\_Info.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Game\_Info[Index] == '.')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Game\_Info[Index]);

}

//Console.ResetColor();

}

/// <summary>

/// Function to display choose menu

/// </summary>

public static void Maze\_Choose\_Screen(string[] Names)

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(24, 17);

Console.SetBufferSize(25, 17);

Console.SetWindowPosition(0, 0);

string Game\_Info = File.ReadAllText(@"Screens/CHOOSE.txt");

for (int Index = 0; Index < Game\_Info.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Game\_Info[Index] == ',')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Game\_Info[Index]);

}

//TODO: Make this universal for lists of any size

// -> put it into separate function

Console.SetCursorPosition(7, 5);

Console.Write(Names[0]);

Console.SetCursorPosition(7, 7);

Console.Write(Names[1]);

Console.SetCursorPosition(7, 9);

Console.Write(Names[2]);

Console.SetCursorPosition(7, 11);

Console.Write(Names[3]);

Console.ResetColor();

}

public static void Maze\_Choose\_Warning\_Screen()

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(24, 17);

Console.SetBufferSize(25, 17);

Console.SetWindowPosition(0, 0);

string Game\_Info = File.ReadAllText(@"Screens/CHOOSE\_WARNING.txt");

for (int Index = 0; Index < Game\_Info.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Game\_Info[Index] == ',')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Game\_Info[Index]);

}

}

public static void Scoreboard\_Screen(string[] Scoreboard\_Info)

{

Console.SetCursorPosition(0, 0);

Console.SetWindowSize(40, 29);

Console.SetBufferSize(41, 29);

Console.SetWindowPosition(0, 0);

string Game\_Info = File.ReadAllText(@"Screens/SCOREBOARD.txt");

for (int Index = 0; Index < Game\_Info.Length; Index++)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.DarkYellow;

if (Game\_Info[Index] == '.')

{

Console.BackgroundColor = ConsoleColor.DarkYellow;

}

Console.Write(Game\_Info[Index]);

}

Console.SetCursorPosition(6, 7);

Console.Write(Scoreboard\_Info[0]);

Console.SetCursorPosition(19, 7);

Console.Write(Scoreboard\_Info[1]);

Console.SetCursorPosition(26, 7);

Console.Write(Scoreboard\_Info[2]);

Console.SetCursorPosition(6, 9);

Console.Write(Scoreboard\_Info[3]);

Console.SetCursorPosition(19, 9);

Console.Write(Scoreboard\_Info[4]);

Console.SetCursorPosition(26, 9);

Console.Write(Scoreboard\_Info[5]);

Console.SetCursorPosition(6, 11);

Console.Write(Scoreboard\_Info[6]);

Console.SetCursorPosition(19, 11);

Console.Write(Scoreboard\_Info[7]);

Console.SetCursorPosition(26, 11);

Console.Write(Scoreboard\_Info[8]);

Console.SetCursorPosition(6, 13);

Console.Write(Scoreboard\_Info[9]);

Console.SetCursorPosition(19, 13);

Console.Write(Scoreboard\_Info[10]);

Console.SetCursorPosition(26, 13);

Console.Write(Scoreboard\_Info[11]);

Console.SetCursorPosition(6, 15);

Console.Write(Scoreboard\_Info[12]);

Console.SetCursorPosition(19, 15);

Console.Write(Scoreboard\_Info[13]);

Console.SetCursorPosition(26, 15);

Console.Write(Scoreboard\_Info[14]);

Console.SetCursorPosition(6, 17);

Console.Write(Scoreboard\_Info[15]);

Console.SetCursorPosition(19, 17);

Console.Write(Scoreboard\_Info[16]);

Console.SetCursorPosition(26, 17);

Console.Write(Scoreboard\_Info[17]);

Console.SetCursorPosition(6, 19);

Console.Write(Scoreboard\_Info[18]);

Console.SetCursorPosition(19, 19);

Console.Write(Scoreboard\_Info[19]);

Console.SetCursorPosition(26, 19);

Console.Write(Scoreboard\_Info[20]);

Console.SetCursorPosition(6, 21);

Console.Write(Scoreboard\_Info[21]);

Console.SetCursorPosition(19, 21);

Console.Write(Scoreboard\_Info[22]);

Console.SetCursorPosition(26, 21);

Console.Write(Scoreboard\_Info[23]);

Console.SetCursorPosition(6, 23);

Console.Write(Scoreboard\_Info[24]);

Console.SetCursorPosition(19, 23);

Console.Write(Scoreboard\_Info[25]);

Console.SetCursorPosition(26, 23);

Console.Write(Scoreboard\_Info[26]);

Console.ResetColor();

}

//public SoundPlayer MusicPlayer = new SoundPlayer(@"MENU.wav");

//MusicPlayer.PlayLooping();

public static ConsoleKeyInfo Key\_Info;

}

}

**Класс Program.cs**

using System;

using System.Media;

using System.Collections.Generic;

using System.Diagnostics;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Windows.Input;

using System.Security.Cryptography.X509Certificates;

using System.Threading;

using System.Linq;

using static CoursevayaCSharp.MazeGenerator;

using static CoursevayaCSharp.PathFinder;

using static CoursevayaCSharp.ScreenVisual;

using static CoursevayaCSharp.MazeFileManagement;

using static CoursevayaCSharp.Scoreboard;

namespace CoursevayaCSharp

{

internal class Program

{

/// <summary>

/// Function to play generated maze game mode

/// </summary>

/// <param name="Maze\_Ptr"></param>

public static void Play\_Generated(List<List<char>> Maze\_Ptr)

{

PlayerInfo PlayerWin = new PlayerInfo {

Name = new string(new char[7]),

Maze\_Name = new string(new char[8]),

Time = 0,

};

int Height = 10; //LVL1

int Width = 15;

//int Height = 13; //LVL2

//int Width = 18;

//int Height = 16; //LVL3

//int Width = 21;

//Generate LVL 1 maze

Maze\_Ptr = Maze\_Generate(Width, Height);

//Play maze

Victory = Maze\_Play(Maze\_Ptr);

if (Victory)

{

//Generate LVL 2 maze

Maze\_Ptr = Maze\_Generate(Width + 3, Height + 3);

//Play maze

Victory = Maze\_Play(Maze\_Ptr);

if (Victory)

{

//Generate LVL 3 maze

Maze\_Ptr = Maze\_Generate(Width + 6, Height + 6);

//Play maze

Victory = Maze\_Play(Maze\_Ptr);

if (Victory)

{

//TODO: let player enter his name -> save it to structure

// -> save all data of player to file (name[7], time, maze)

// -> output scoreboard with all victorious attempts

Console.Clear();

Player\_Input\_Screen();

Console.SetCursorPosition(9, 6);

PlayerWin.Name = Scoreboard\_Player\_Input(PlayerWin.Name);

PlayerWin.Maze\_Name = Maze\_File\_Save(Maze\_Ptr, true);

Scoreboard\_Save(PlayerWin);

Leaf\_Through\_Scoreboard();

return;

}

}

}

if (!Victory)

{

//Display lose screen, let player try again or go back to the menu

Console.Clear();

Lose\_Screen();

Key\_Info = Console.ReadKey();

}

}

/// <summary>

/// Function to play prepared maze game mode

/// </summary>

public static void Play\_Prepared(List<List<char>> Maze\_Ptr, string File\_Name)

{

PlayerInfo PlayerWin = new PlayerInfo

{

Name = new string(new char[7]),

Maze\_Name = new string(new char[8]),

Time = 0,

};

Console.ResetColor();

Console.Clear();

//Get maze from file

Maze\_Ptr = Maze\_File\_Load(File\_Name);

//Play maze

Victory = Maze\_Play(Maze\_Ptr);

//If player has beaten the bot, display scoreboard + add result

if (Victory)

{

//TODO: let player enter his name -> save it to structure

// -> save all data of player to file (name[7], time, maze)

// -> output scoreboard with all victorious attempts

Console.Clear();

Player\_Input\_Screen();

Console.SetCursorPosition(9, 6);

PlayerWin.Name = Scoreboard\_Player\_Input(PlayerWin.Name);

PlayerWin.Maze\_Name = Maze\_File\_Save(Maze\_Ptr, false);

PlayerWin.Time = WinTime;

Scoreboard\_Save(PlayerWin);

Leaf\_Through\_Scoreboard();

return;

}

else {

//Display lose screen, let player try again or go back to the menu

Console.Clear();

Lose\_Screen();

Key\_Info = Console.ReadKey();

return;

}

}

/// <summary>

/// Function that enables player to choose from saved mazes and play them

/// </summary>

public static void Choose\_Maze()

{

List<List<char>> Maze\_Ptr = new();

bool Exit = false;

int File\_Quad = 0;

string[] File\_Names = Load\_Maze\_File\_Names();

string[] File\_Quad\_Names = Get\_Maze\_Quad(File\_Quad, File\_Names);

while (!Exit)

{

//Display prepared maze screen with quad of file names

Maze\_Choose\_Screen(File\_Quad\_Names);

//Wait for input

Key\_Info = Console.ReadKey();

Click\_Sound.Play();

switch (Key\_Info.Key)

{

//if -> show next quad (if remaining files don't make a quad, display only the remainings)

case ConsoleKey.RightArrow:

{

File\_Quad = File\_Quad + 4;

//Check if quad goes beyond name array's range, if does fit it in this range

if (File\_Quad > File\_Names.Length)

{

File\_Quad = File\_Quad - 4;

}

File\_Quad\_Names = Get\_Maze\_Quad(File\_Quad, File\_Names);

break;

}

//if <- show prev quad (if it's going negative, show first quad)

case ConsoleKey.LeftArrow:

{

File\_Quad = File\_Quad - 4;

//Check if quad goes beyond name array's range, if does fit it in this range

if (File\_Quad < 0)

{

File\_Quad = File\_Quad + 4;

}

File\_Quad\_Names = Get\_Maze\_Quad(File\_Quad, File\_Names);

break;

}

//If player inputs "1" - "4", show warning

// //if proceed - start game,

// //if not - let player choose different maze

// //TODO: Add screen to validate player's decision

case ConsoleKey.D1:

{

Maze\_Choose\_Warning\_Screen();

//Wait for input

Key\_Info = Console.ReadKey();

Click\_Sound.Play();

if (Key\_Info.Key == ConsoleKey.Enter)

{

Play\_Prepared(Maze\_Ptr, File\_Quad\_Names[0]);

}

if (Key\_Info.Key == ConsoleKey.Escape)

{

break;

}

break;

}

case ConsoleKey.D2:

{

//Check if this quad member has any name value, if not - go back

if (File\_Quad\_Names[1] == null)

{

break;

}

Maze\_Choose\_Warning\_Screen();

//Wait for input

Key\_Info = Console.ReadKey();

Click\_Sound.Play();

if (Key\_Info.Key == ConsoleKey.Enter)

{

Play\_Prepared(Maze\_Ptr, File\_Quad\_Names[1]);

}

if (Key\_Info.Key == ConsoleKey.Escape)

{

break;

}

break;

}

case ConsoleKey.D3:

{

//Check if this quad member has any name value, if not - go back

if (File\_Quad\_Names[2] == null)

{

break;

}

Maze\_Choose\_Warning\_Screen();

//Wait for input

Key\_Info = Console.ReadKey();

Click\_Sound.Play();

if (Key\_Info.Key == ConsoleKey.Enter)

{

Play\_Prepared(Maze\_Ptr, File\_Quad\_Names[2]);

}

if (Key\_Info.Key == ConsoleKey.Escape)

{

break;

}

break;

}

case ConsoleKey.D4:

{

//Check if this quad member has any name value, if not - go back

if (File\_Quad\_Names[3] == null)

{

break;

}

Maze\_Choose\_Warning\_Screen();

//Wait for input:

Key\_Info = Console.ReadKey();

Click\_Sound.Play();

if (Key\_Info.Key == ConsoleKey.Enter)

{

Play\_Prepared(Maze\_Ptr, File\_Quad\_Names[3]);

}

if (Key\_Info.Key == ConsoleKey.Escape)

{

break;

}

break;

}

//If player pressed Esc - go back to main menu

case ConsoleKey.Escape:

{

Click\_Sound.Play();

Exit = true;

break;

}

}

}

}

/// <summary>

/// Function to iterate through names of maze files

/// </summary>

/// <returns>Quad of names</returns>

public static string[] Get\_Maze\_Quad(int Quad, string[] Names)

{

int Counter = 0;

string[] Quad\_Names = new string[4];

do

{

if (Quad == Names.Length)

{

return Quad\_Names;

}

Quad\_Names[Counter] = Names[Quad];

Quad++;

Counter++;

} while (Counter != 4 && Quad < Names.Length);

return Quad\_Names;

}

/// <summary>

/// Function to display menu

/// </summary>

public static void Game()

{

//Initialize values for maze storage

List<List<char>> Maze\_Ptr = new();

//Boolean variable for game loop condition

bool Exit = false;

//In fact this is the core loop of the game

while (!Exit)

{

//Draw menu

Game\_Menu\_Screen();

//Wait for input of desired option (1,2,3,4)

Key\_Info = Console.ReadKey();

Click\_Sound.Play();

Console.Clear();

switch (Key\_Info.Key)

{

//Play on generated maze

case ConsoleKey.D1:

{

Play\_Generated(Maze\_Ptr);

break;

}

//Play on prepared maze

case ConsoleKey.D2:

{

Choose\_Maze();

break;

}

//Show scoreboard

case ConsoleKey.D3:

{

//Display scoreboard

Leaf\_Through\_Scoreboard();

break;

}

//Display game info

case ConsoleKey.D4:

{

while (true)

{

//Display info from file

Game\_Info\_Screen();

//Wait for player's input

Key\_Info = Console.ReadKey();

//Let player leave this screen

if (Key\_Info.Key == ConsoleKey.Escape)

{

Click\_Sound.Play();

break;

}

}

break;

}

//Exit game

case ConsoleKey.Escape:

{

while (true)

{

//Display warning

Exit\_Warning\_Screen();

//Wait for input (Y/N)

Key\_Info = Console.ReadKey();

//If Y exit

if (Key\_Info.Key == ConsoleKey.Y)

{

Exit = true;

break;

}

//If N go to menu

if (Key\_Info.Key == ConsoleKey.N || Key\_Info.Key == ConsoleKey.Escape)

{

Click\_Sound.Play();

break;

}

}

break;

}

}

}

}

/// <summary>

/// Function that starts gameplay

/// </summary>

public static bool Maze\_Play(List<List<char>> Maze\_Ptr)

{

//Array to save elapsed time data

List<long> Elapsed\_Time = new();

//Initialize position of player

PathFinder Player = new();

Position2D Player\_Pos = new(1, 1);

//Loading already generated maze

List<List<char>> Buf\_Maze\_Ptr = Maze\_Ptr.Maze\_Copy();

List<List<char>> Prev\_Buf\_Maze\_Ptr = Maze\_Ptr.Maze\_Copy();

Player.Init\_Player(Player\_Pos, Buf\_Maze\_Ptr);

//Ready screen

Ready\_Screen(Maze\_Ptr);

Key\_Info = Console.ReadKey();

Console.ResetColor();

Console.Clear();

Console.SetWindowPosition(0, 0);

//Returning console window to maze size

Console.SetWindowSize(Buf\_Maze\_Ptr[0].Count + 2, Buf\_Maze\_Ptr.Count + 1);

Console.SetBufferSize(Buf\_Maze\_Ptr[0].Count + 3, Buf\_Maze\_Ptr.Count + 1);

Click\_Sound.Play();

//Marking the time of maze completion

var Time = Stopwatch.StartNew();

//Displaying first frame of maze

Console.SetWindowSize(Maze\_Ptr[0].Count + 2, Maze\_Ptr.Count + 1);

Console.SetBufferSize(Maze\_Ptr[0].Count + 3, Maze\_Ptr.Count + 1);

Console.SetCursorPosition(1, 0);

Maze\_Print(Maze\_Ptr, 1);

Console.SetCursorPosition(1, 0);

//Player goes first

while (!Player.isWin())

{

Player.Player\_Controller();

Prev\_Buf\_Maze\_Ptr = Maze\_Print(Prev\_Buf\_Maze\_Ptr, Buf\_Maze\_Ptr);

Console.SetCursorPosition(1, 0);

}

//Stopping the timer and saving elapsed time data

Time.Stop();

Elapsed\_Time.Add(Time.ElapsedMilliseconds / 1000);

//Clearing the console to hide maze

Console.Clear();

//Bot goes next

//Loading already generated maze

Buf\_Maze\_Ptr = Maze\_Ptr.Maze\_Copy();

//Return player to the start

Player.Init\_Player(Player\_Pos, Buf\_Maze\_Ptr);

Maze\_Print(Maze\_Ptr, 1);

Console.SetCursorPosition(1, 0);

//Marking the time of maze completion

Time = Stopwatch.StartNew();

Maze\_Print(Maze\_Ptr, 1);

while (!Player.isWin())

{

Player.Next\_Step();

Prev\_Buf\_Maze\_Ptr = Maze\_Print(Prev\_Buf\_Maze\_Ptr, Buf\_Maze\_Ptr);

Console.SetCursorPosition(1, 0);

//Add delay in bots movement, because it's moving every tick

Thread.Sleep(75);

}

//Stopping the timer and saving elapsed time data

Time.Stop();

Elapsed\_Time.Add(Time.ElapsedMilliseconds / 1000);

//Clearing the console to hide maze

Console.ResetColor();

Console.Clear();

//Display elapsed time for Player and Bot\_Player

Elapsed\_Time\_Screen(Elapsed\_Time);

Console.ReadKey();

Click\_Sound.Play();

Console.ResetColor();

Console.Clear();

if (Elapsed\_Time[0] == Elapsed\_Time[1])

{

//Maybe add tie screen

return false;

}

WinTime = Elapsed\_Time[0];

return (Elapsed\_Time[0] < Elapsed\_Time[1]);

}

public static long WinTime;

public static PlayerInfo Player\_Info;

public static ConsoleKeyInfo Key\_Info;

public static bool Victory;

public static SoundPlayer Click\_Sound = new(@"Sounds/CLICK.wav");

static void Main(string[] args)

{

//Display title screen

Title\_Screen();

Click\_Sound.Play();

Game();

//int c = 0;

//while (c != 14)

//{

// Maze\_File\_Save(Maze\_Generate(21, 16), true);

// c++;

//}

}

}

}