Санкт-Петербургский политехнический университет Петра Великого

Кафедра компьютерных систем и программных технологий

**Отчёт по проекту**

**Дисциплина**: Алгоритмы и структуры данных

**Вариант**: 30

Выполнил студент гр. 3530901/90003 М.С. Зотов

(подпись)

Преподаватель М.Х. Ахин

(подпись)

“ ” 2020 г.

Санкт-Петербург

2020

Оглавление

[**1. Техническое задание** 3](#_Toc57231376)

[**2. Метод решения** 5](#_Toc57231377)

[**3. Описание классов и методов** 7](#_Toc57231378)

[**4. Работа программы** 11](#_Toc57231379)

# **1. Техническое задание**

**Описание приложения**

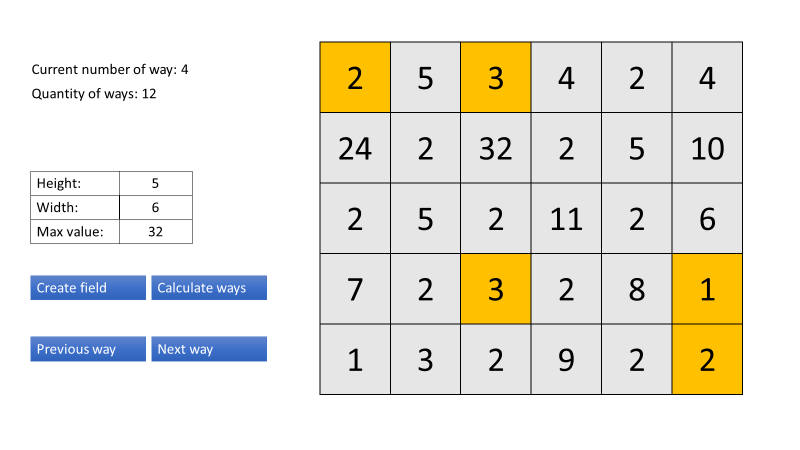
**Общие сведения**

Поле NxM заполняется целыми положительными числами. Целое число в каждой клетке указывает, какой длины должен быть шаг из нее. Все шаги могут быть только вправо или вниз. Найти все возможные пути из левого верхнего угла в правый нижний

**Более подробная информация**

* Приложение будет реализовано на JavaFX
* Размер поля определяется на основе входных данных пользователя. Визуализация поля зависит от высоты и ширины: чем они больше, тем меньшего размера ячейки поля
* На стартовом и единственном окне приложения есть следующие компоненты:
  + Контейнер, внутри которого будет отображаться поле – занимает большую часть экрана
  + Поля для ввода, указывающее высоту и ширину поля, а также максимальное значения в ячейках
  + Каждая ячейка поля при нажатии на себя позволяет пользователю изменить своё значение: ПКМ – уменьшить значение, ЛКМ - увеличить
  + Кнопка «Create field» - по нажатии создаётся новое поле
  + Поле для вывода «Current number of way» - отображает номер (номер = индекс + 1) текущего рассматриваемого пути
  + Поле для вывода «Quantity of ways» - отображает всё количество найденных путей
  + Кнопка «Calculate ways» - запускает алгоритм поиска путей, в результате которого обновляются данные таких полей для вывода, как «Current number of way» и «Quantity of ways», а также, если хотя бы один путь найден, специальным цветом выделяются ячейки, которые представляют собой путь с нулевым индексом, если же путей нет – выделений цветом не производиться
  + Кнопка «Next way» - если текущее значение «Current number of way» меньше длины всех наборов путей, увеличивает «Current number of way» на 1, иначе – не делает ничего
  + Кнопка «Previous way» - если текущее значение «Current number of way» больше 0, уменьшает «Current number of way» на 1, иначе – не делает ничего

**Предварительный вид приложения**



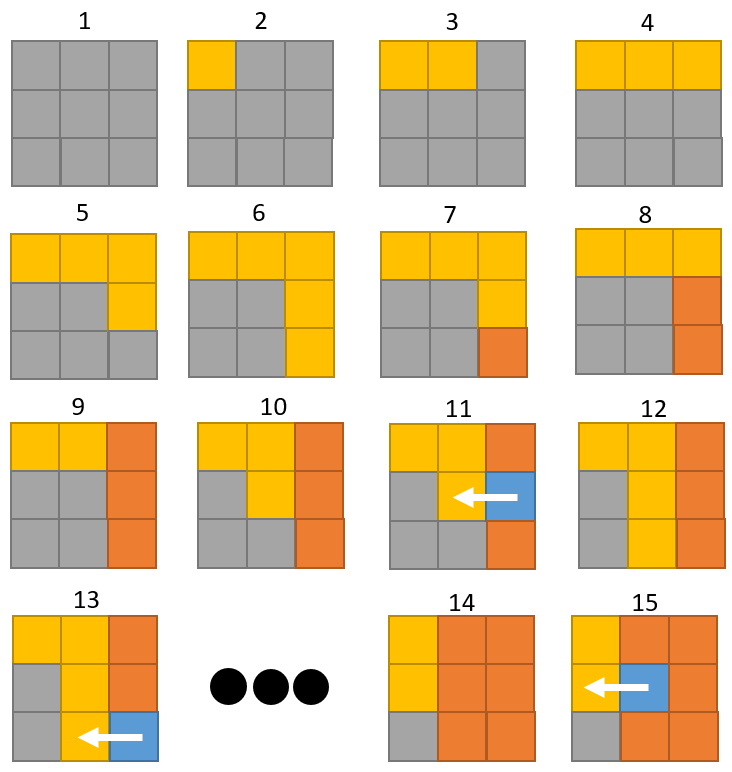
**GitHub**

<https://github.com/MaksimZotov/FindingWays>

# **2. Метод решения**

В основе решения лежит идея одноразового посещения ячеек, в которые мы можем попасть, за счет сохранения информации о пройденных ячейках.

Для наглядности рассмотрим пример:



В этом примере исходим из того, что каждая ячейка поля имеет значение 1 (то есть из каждой ячейки можно перейти в соседнюю), значение N = M = 3.

Алгоритм работает так:

У нас имеется класс вершины Vertex, главная функциональность которого – хранить ссылки на правые и нижние вершины, в которые мы можем сделать шаг из данной, а также содержать поле isThereWay, говорящее о том, что данная вершина является частью хотя бы одного из путей.

Первым делом создаем мапу map, в которую позже будем по индексам ячейки (key) заносить объекты типа Vertex (value), соответствующие этой ячейке. Далее:

Мы рекурсивно идем вправо, если вправо нельзя, идем вниз (шаги 1-6). Как дошли до финальной правой нижней ячейки, заносим в map (key = 2 to 2) объект типа Vertex, *передавая ему в конструктор индексы текущей ячейки (2 и 2)*, а также, так как это финиш, указываем, что данный Vertex.isThereWay = true – ***это шаг 7***. Аналогичные действия проделываем на 8-9 – так как у вершин на этих шагах у нижнего соседа isThereWay = true, то мы добавляем к текущим вершинам нижнего соседа в качестве потомка и также указываем, что у текущих вершин isThereWay = true. На 9-10 шагах, так как правого соседа рассмотрели, идем вниз. На 11 шаге поворачиваем вправо. Так как синяя вершина на шаге 8 была занесена в map (проверка этого условия делается через просмотр условия map.containsKey(1 to 2)), мы серединной вершине в качестве правого соседа просто присваиваем ссылку на синюю вершину (а, если бы, например, синяя вершина из 11 шага на 8 шаге не получила бы состояние isThereWay = true (допустим, ячейка с индексами (1 to 2) имела бы значение 2) то серединной вершине мы бы ничего не присвоили). Аналогично делается на 13 и 15 шагах. Как видим, вместо того, чтобы на 15 шаге проверять 2 возможных пути из синей ячейки в финальную, мы ограничиваемся всего лишь одним действием.

После того, как каждая вершина будет иметь ссылки на правую и нижнюю вершины, в которые можно сделать шаг из данной (или не будет иметь, если правая или нижняя вершина отсутствует или не ведёт к финальной вершине), мы, начиная со стартовой вершины, проходим по её потомкам рекурсивно в глубину. Когда достигаем финальной вершины в определённой ветке рекурсии, последовательность посещённых в этой ветке вершин принимаем за один из возможных путей.

Так как каждая вершина инициализирована соответствующими ей индексами ***(смотреть описание шага 7 – подчеркнутую часть)***, не составляет особой сложности по пути, состоящему из вершин, восстановить путь, состоящий из ячеек.

Причина, по которой я не стал объединять класс ячейки поля и класс вершины в один общий класс, состоит в желании абстрагировать работу алгоритма от остальной логики приложения. Если все же объединить, то алгоритм будет работать чуть быстрее за счет того, что по пути из вершин не надо будет строить путь из ячеек, но это мало даст выигрыша по времени. Так что решил оставить как есть – так архитектура приложения получается более гибкой к возможным изменениям.

# **3. Описание классов и методов**

**Класс Main:**

***Отвечает за визуализацию состояния модели, получает от пользователя входные данные****.*

**Методы класса Main:**

start():

Создаёт UI-элементы, привязывает методы кнопкам, при нажатии на которые данные методы будут вызываться.

getUpdatedDataAboutTheModel(Object data):

На основании состояния data строит обновленное состояние модели.

createField():

Отправляет модели запрос на создание поля из ячеек

setNumberOfCell(int row, int column, int number):

Отправляет модели запрос установить значение ячейки с индексами row, column, равное number.

calculateWays():

Отправляет модели запрос вычислить пути.

showNextCalculatedWay():

Отправляет модели запрос показать следующий вычисленный путь.

showPreviousCalculatedWay():

Отправляет модели запрос показать предыдущий вычисленный путь.

showHelp():

Открывает окно с инструкцией к программе.

updateStageForResolution(Stage stage, double resolutionY, double resolutionX):

Обновляет размеры и расположение UI-элементов в соответствии с новым разрешением.

getIndexOfNumber(Text number):

На основе объекта number определяет индекс ячейки, соответствующей number. Используется при изменении значения ячейки посредством клика по ней.

**Класс State:**

***Содержит ту информацию, которая может быть интересна пользователю.***

**Методы класса State:**

getQuantityOfWays():

Возвращает количество высчитанных путей.

getIndexOfCurrentWay():

Возвращает индекс показываемого в данный момент пользователю пути.

getFieldHeight():

Возвращает количество строк.

getFieldWidth():

Возвращает количество столбцов.

getNumberOfCell(int row, int column):

Возвращает значение ячейки с индексами row, column.

getCellIsItPartOfTheWay(int row, int column):

Возвращает true, если ячейка с индексами row, column является частью показываемого в данный момент пользователю пути.

***Также State содержит данные и методы для работы с ними, которые могут понадобиться алгоритму поиска путей.***

getField():

Возвращает объект текущего поля ячеек.

setNextCalculatedWay():

Устанавливает следующий высчитанный путь.

setPreviousCalculatedWay():

Устанавливает предыдущий высчитанный путь

setCalculatedWay(Consumer<Integer> consumer):

Вспомогательный метод для setPreviousCalculatedWay() и setNextCalculatedWay().

setWays(ArrayList<ArrayList<Cell>> ways):

После расчета путей класс с алгоритмом устанавливает ways через этот метод.

setField(Field field):

Устанавливает поле.

**Класс Model:**

***Обрабатывает входные данные от пользователя и возвращает ему информацию об изменениях.***

**Методы класса Model:**

createField():

Создаёт поле из ячеек.

setNumberOfCell(int row, int column, int number):

Устанавливает значение ячейки с индексами row, column, равное number.

calculateWays():

Вычисляет пути.

showNextCalculatedWay():

Изменяет в State текущий путь на следующий.

showPreviousCalculatedWay():

Изменяет в State текущий путь на предыдущий.

**Класс Field:**

***Отражает состояние поля.***

**Методы класса Field:**

getHeight():

Возвращает количество строк.

getWidth():

Возвращает количество столбцов.

getCell(int row, int column):

Возвращает Cell по индексу row, column.

setThatAllCellsAreNotPartOfTheWay():

Устанавливает, что ни одна ячейка поля не является частью пути.

**Класс Cell:**

***Отражает состояние ячейки поля.***

**Методы класса Cell:**

getNumber():

Возвращает номер ячейки

getIsItPartOfTheWay():

Возвращает true, если ячейка является частью показываемого в данный момент пользователю пути.

setIsItPartOfTheWay(boolean isItPartOfTheWay):

Устанавливает, является ли ячейка частью показываемого в данный момент пользователю пути.

setNumber(int number):

Устанавливает номер ячейки

**Класс CustomAlgorithm:**

***Содержит алгоритм поиска путей.***

***Есть внутренний класс Vertex для вершин, описание которого есть в разъяснении метода решения.***

**Методы класса CustomAlgorithm:**

getWays(Field field):

Возвращает список последовательностей вершин, где каждая последовательность – один из возможных путей.

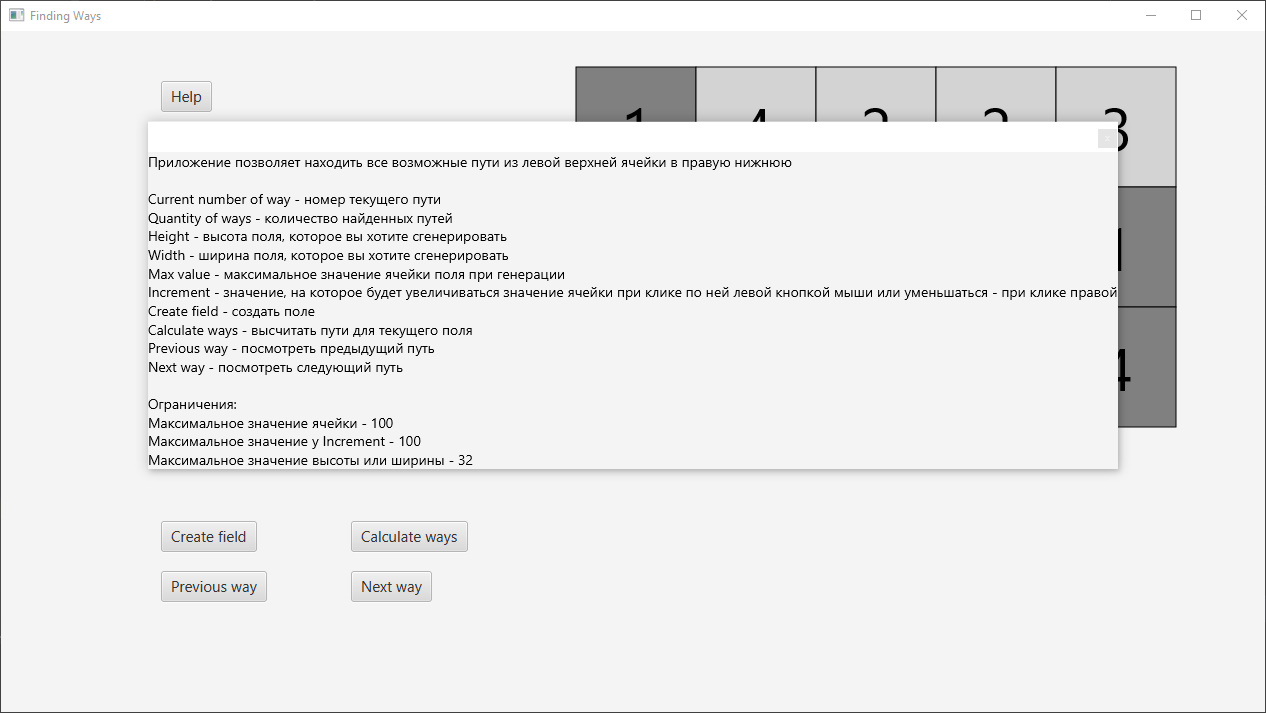
findChildren():

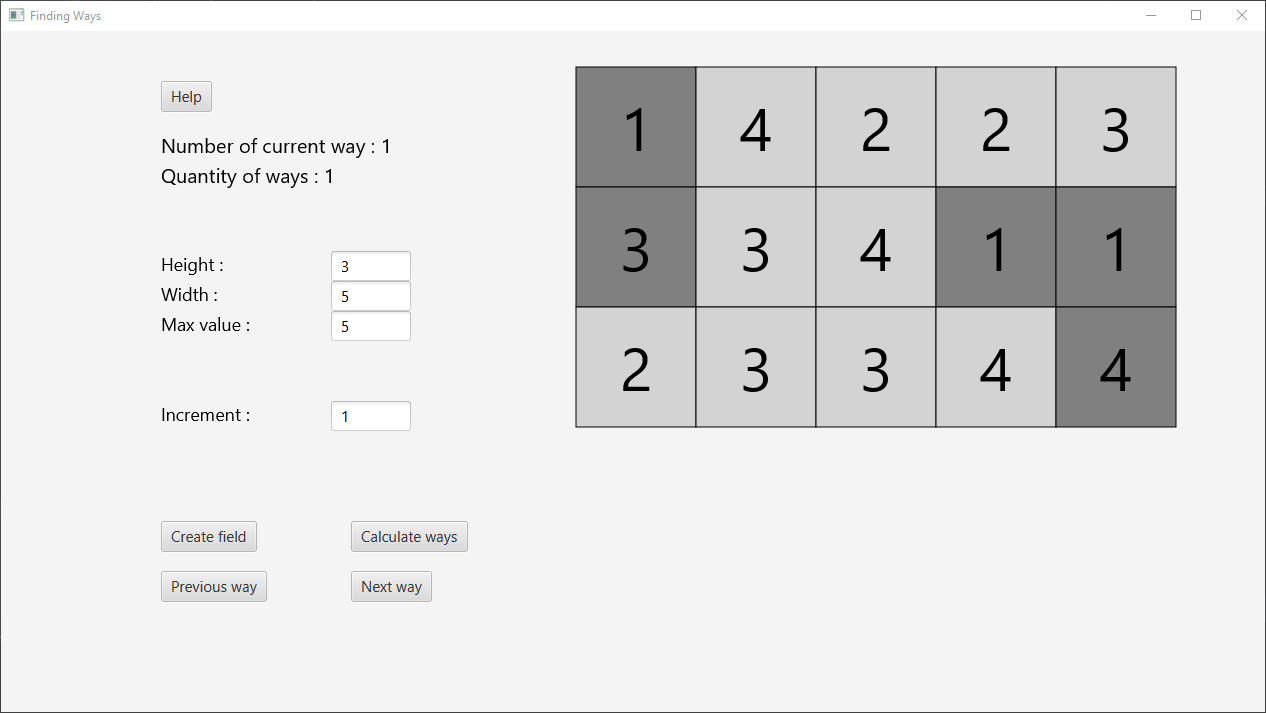
Строит дерево потомков для старшей вершины, где концом каждой ветки является финальная вершина.

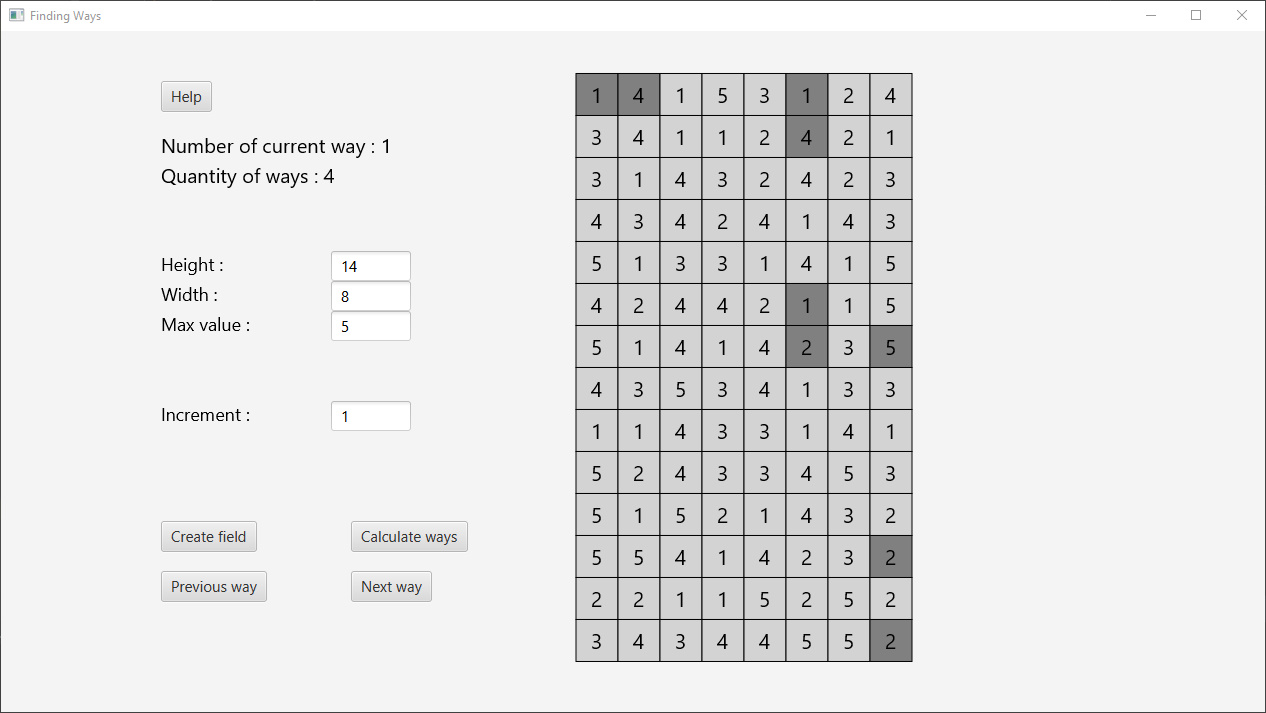
createWaysOnBaseOfVertices(Vertex vertex, ArrayList<Cell> way):

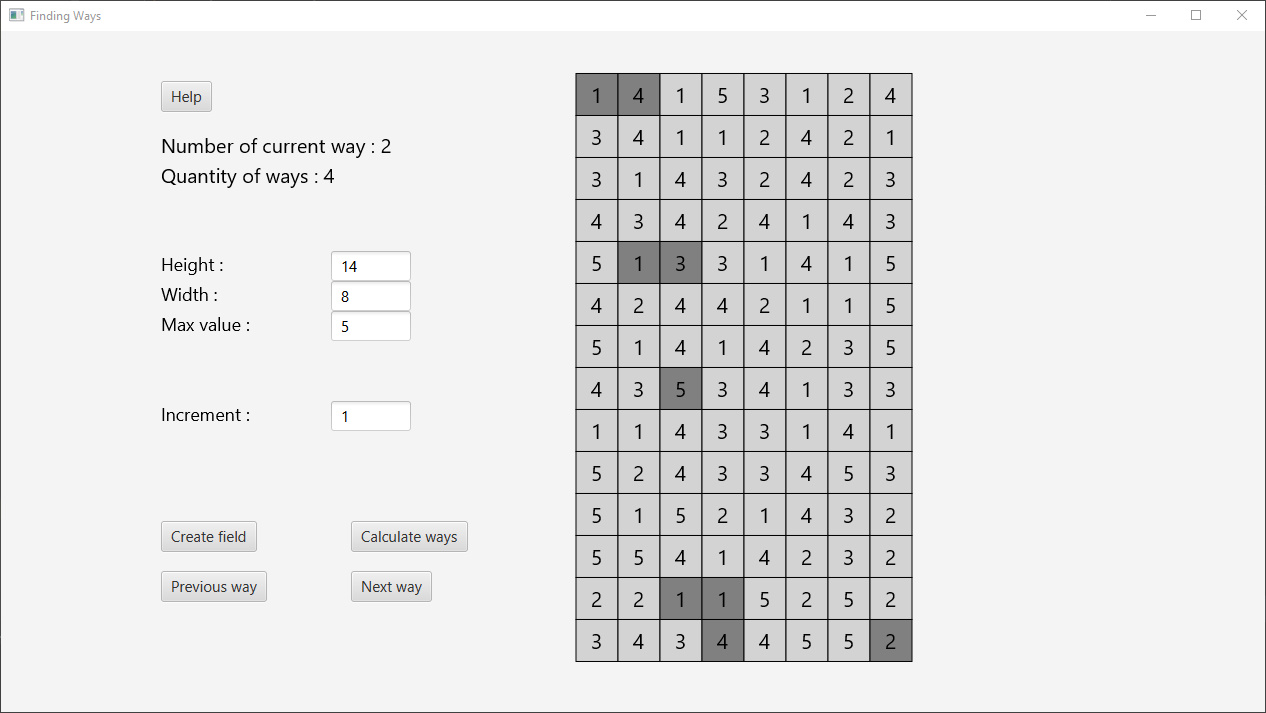
После исполнения findChildren() вызывается этот метод – тут на основе построенного дерева, где каждая ветка является путём из вершин, строится последовательность путей уже из ячеек, а затем эти пути возвращаются данным методом.

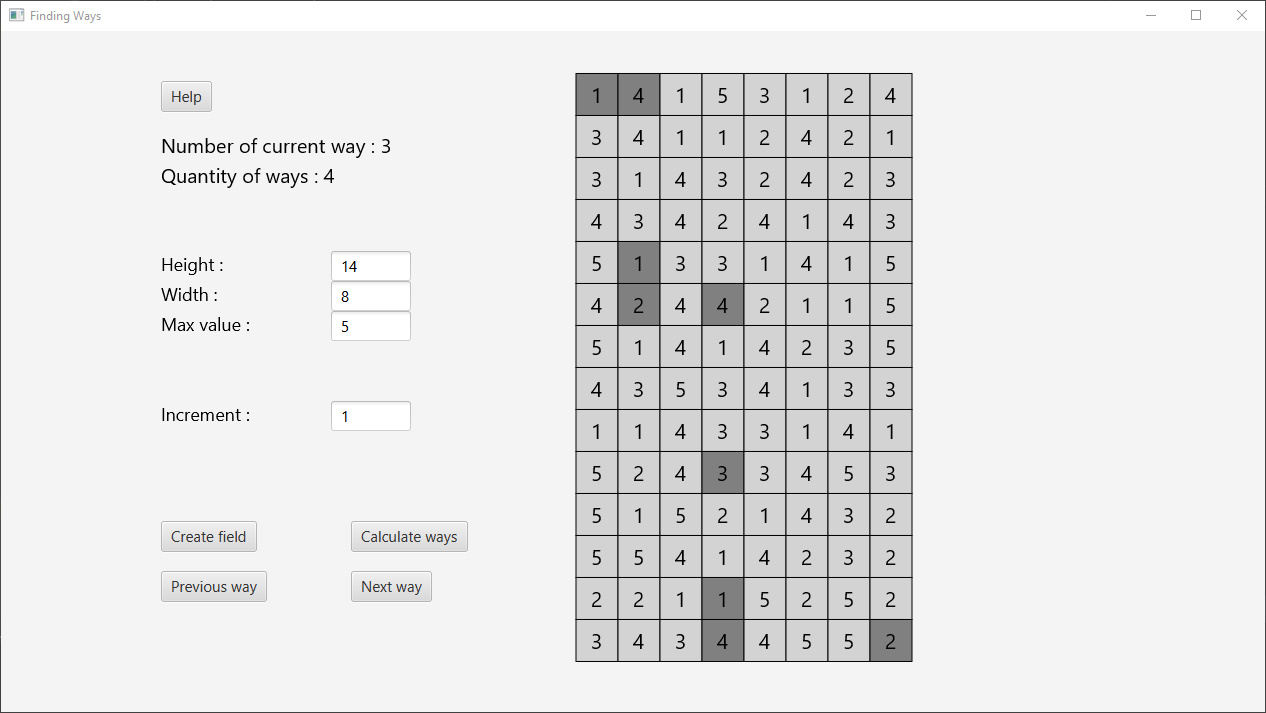
# **4. Работа программы**

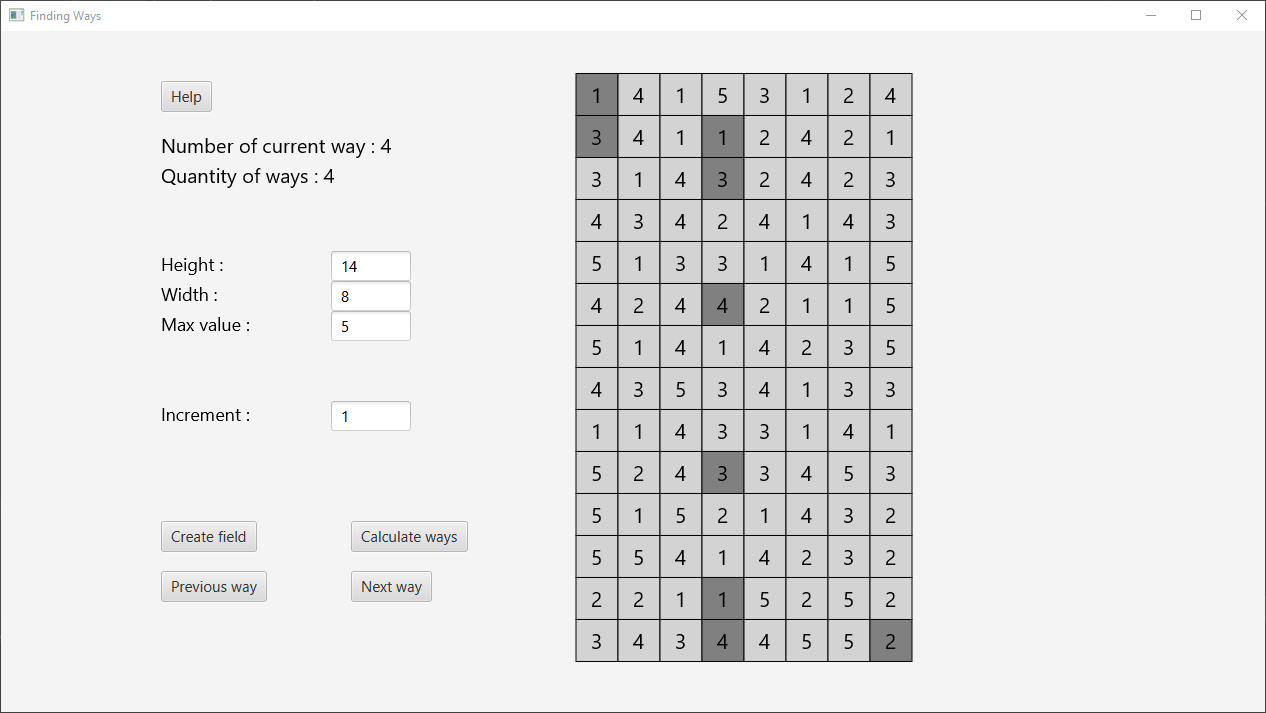


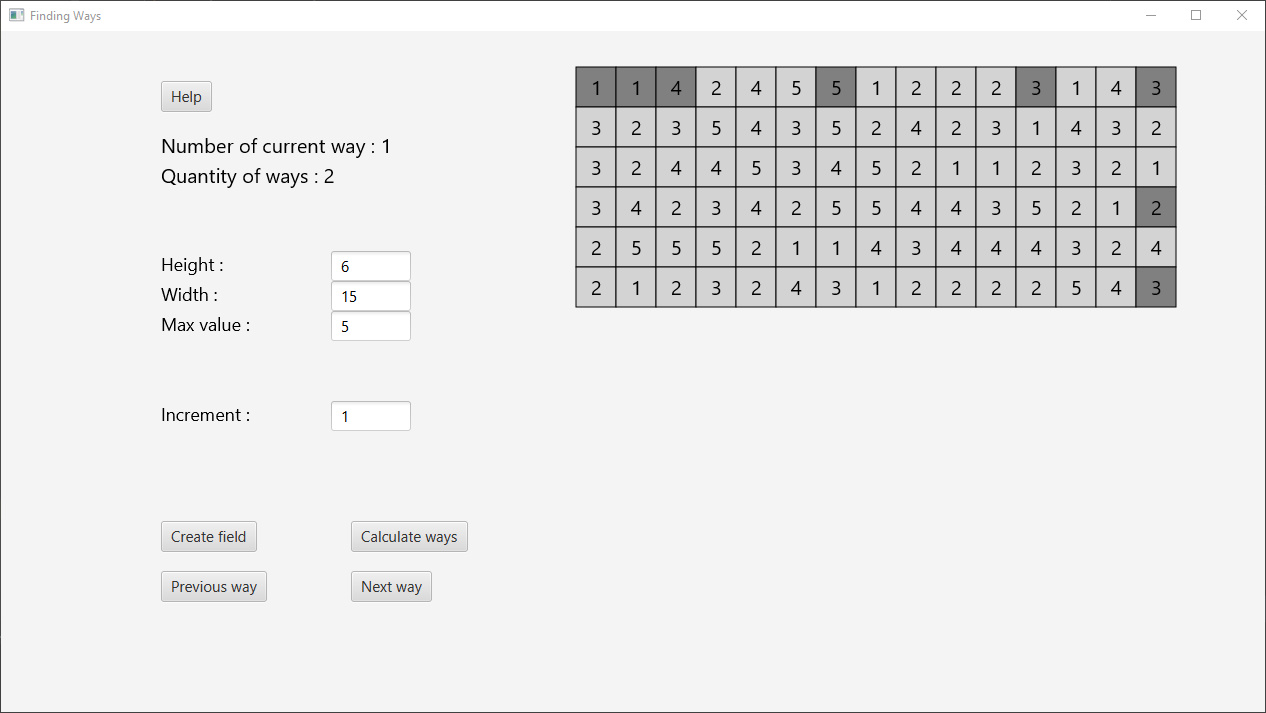
****

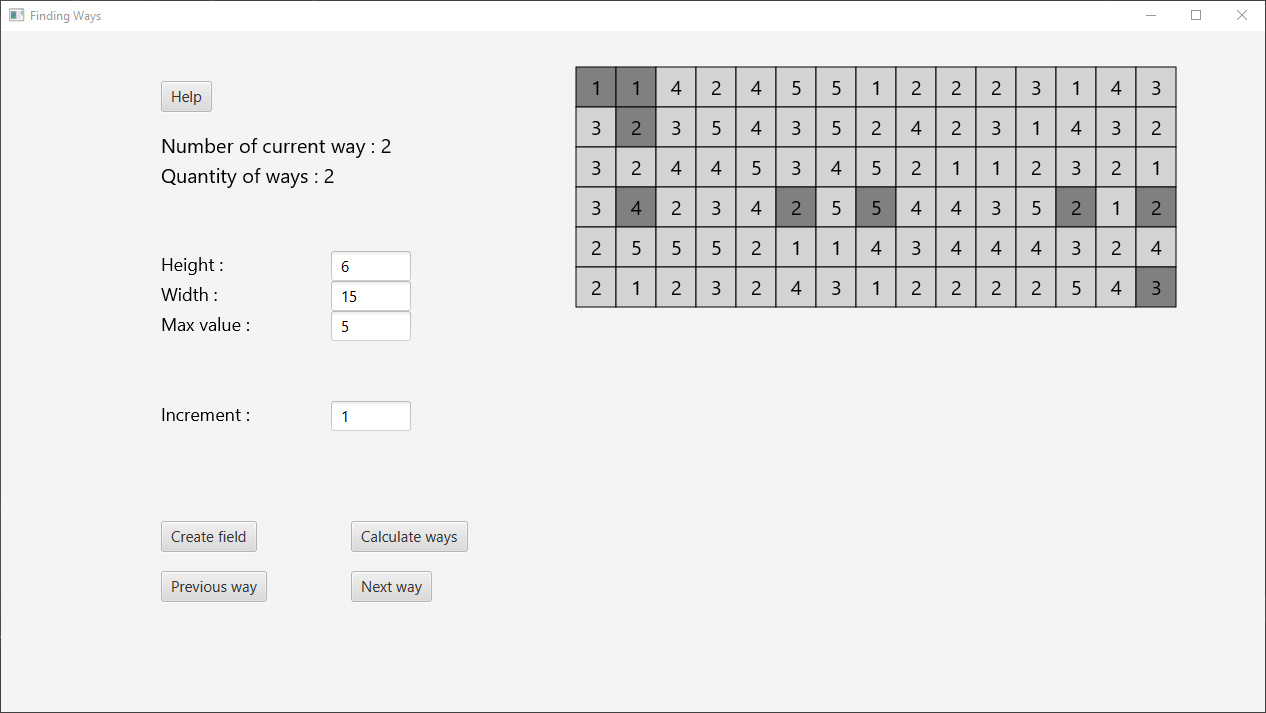


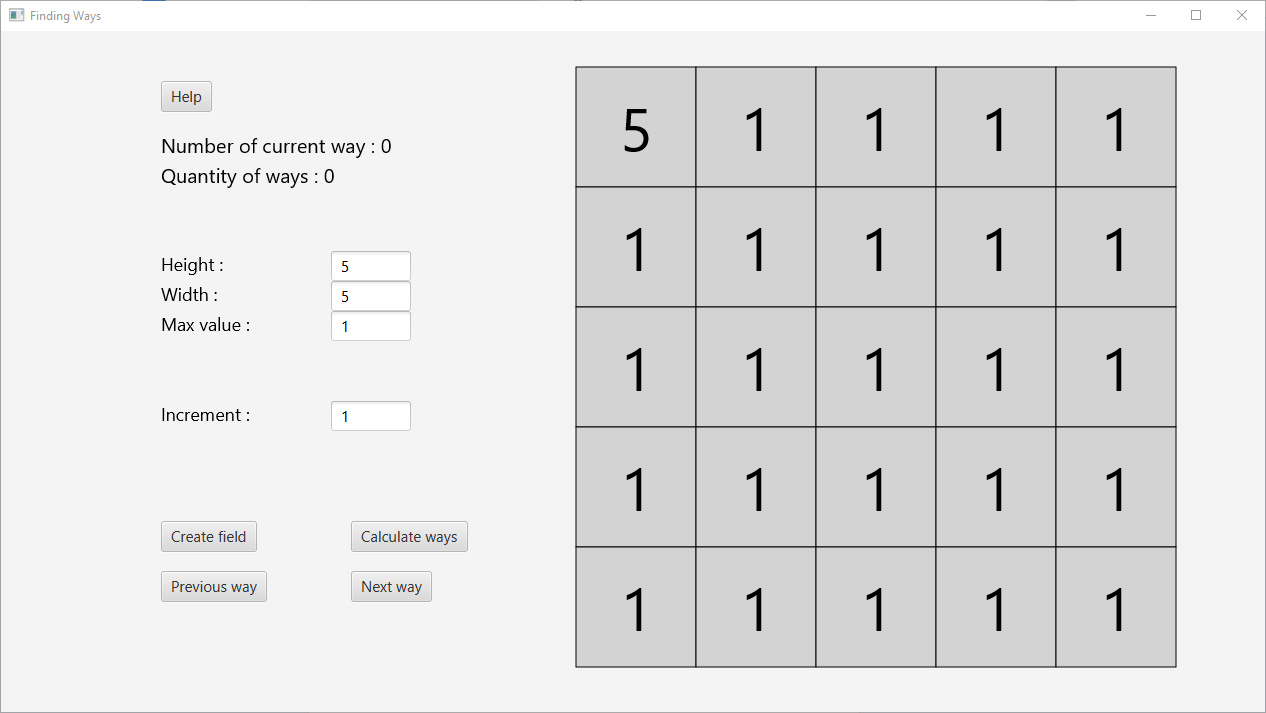












На этом скриншоте видно, что в случае, когда пути до финальной ячейки не найдены, Quantity of ways (количество путей) равно 0.