**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по учебной практике**

**Тема: Реализация алгоритма A\* на языке Kotlin с визуализацией**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
|  |  |  |
| Студент гр. 1303 |  | Попандопуло А.Г. |
|  |  |  |
| Руководитель |  | Шестопалов Р.П. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**НА УЧЕБНУЮ ПРАКТИКУ**

| Студент Чубан Д.В. группы 1303 | | |
| --- | --- | --- |
| Студент Попандопуло А.Г. группы 1303 | | |
| Тема практики: Командная итеративная разработка визуализатора алгоритма на Kotlin с графическим интерфейсом | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.  Алгоритм: А\* | | |
| Сроки прохождения практики: 30.06.2023 – 13.07.2023 | | |
| Дата сдачи отчета: 13.07.2023 | | |
| Дата защиты отчета: 13.07.2023 | | |
|  | | |
| Студент |  | Чубан Д.В. |
|  |  |  |
| Студент |  | Попандопуло А.Г. |
|  |  |  |
| Руководитель |  | Шестопалов Р.П. |

**АННОТАЦИЯ**

Целью проекта является получение навыков программирования на Kotlin и создание программы по поиску кратчайшего пути во взвешенном графе, визуализирующей работу алгоритма А\*

**SUMMARY**

The aim of the project is to acquire programming skills in Kotlin and create a program to find the shortest path in a weighted graph, visualizing the operation of the algorithm A\*

**СОДЕРЖАНИЕ**

|  | Введение | 5 |
| --- | --- | --- |
| 1. | Требования к программе | 6 |
| 1.1. | Исходные требования к программе | 6 |
| 1.2. | Уточнение требований после сдачи 1-ой версии | 7 |
| 2. | План разработки и распределение ролей в бригаде | 8 |
| 2.1. | План разработки | 8 |
| 2.2. | Распределение ролей в бригаде | 8 |
| 3. | Особенности реализации | 9 |
| 3.1. | Основные структуры данных | 9 |
| 3.2. | Структуры данных, отвечающие за алгоритм | 11 |
| 3.3 | Структуры данных, отвечающие за визуализацию | 12 |
| 4. | Тестирование | 14 |
| 4.1 | Тестирование интерфейса и обработки исключительных ситуаций | 14 |
| 4.2 | Тестирование алгоритма | 15 |
|  | Заключение | 17 |
|  | Список использованных источников | 18 |
|  |  |  |

**ВВЕДЕНИЕ**

Главной целью работы было реализовать алгоритм A\* для поиска кратчайших путей на карте и представить его в виде приложения с графическим интерфейсом. Для корректной работы алгоритма реализуем очередь с приоритетом, в которой будут храниться клетки-кандидаты для перехода.

В реализованную очередь с приоритетом добавляем стартовую вершину. До тех пор, пока очередь не пуста, достаем из нее вершину с наименьшим значением эвристической функции и рассчитываем аналогичное значение для смежных вершин. Если очередная вершина ещё не была посещена, или существующая оценка больше только что вычисленной, значение для данной вершины обновляется. После этого вершина и её приоритет помещаются в очередь. Если достигнута конечная вершина, поиск прекращается.

**1. ТРЕБОВАНИЯ К ПРОГРАММЕ**

**1.1. Исходные Требования к программе**

**1.1.1 – Требования к вводу исходных данных**

Задать поле можно как через текстовый файл, так и используя интерфейс программы.

Текстовый файл должен иметь вид

\*X\* \*Y\* - Размеры поля

\*X\* \*Y\* – Координаты старта

\*X\* \*Y\* – Координаты финиша

Далее идет описание каждой клетки:

\*Up\* \*Right\* \*Down\* \*Left\* – Стоимость перехода в соответствующую соседнюю клетку

\*Type\* – Тип клетки (например, непроходимая или обычная)

Если ввод нужно сделать используя интерфейс, то сначала нужно нажать кнопку “Задать поле”, выбрать его размеры и точки старта и финиша. Затем нажимая на каждую созданную клетку задать ее характеристики. Долгое нажатие будет изменять тип клетки.

**1.1.2 – Требования к визуализации**

Окно разделено на три части (Рисунок 1):

Левая – записываются действия алгоритма по выбору следующей вершины, промежуточные выводы. Также присутствуют кнопки для перехода к следующему шагу либо моментальному нахождению пути.

Средняя – визуализация поля. Спецсимволами выделяются старт и финиш, цветами выделяются пройденные клетки, текущая рассматриваемая клетка и непроходимые клетки.

Правая - функциональная. Имеет кнопки “Открыть файл”, чтобы прочитать поле из файла, “Задать поле”, чтобы задать поле вручную через интерфейс, и “Сохранить поле”, чтобы создать текстовый файл с данными о текущем поле.

Дополнительные окна вызываются при нажатии на клетку и кнопку “Задать поле”. В окне клетки можно вручную задать веса путей в соседние клетки. В окне “Задать поле” вводится размер поля и координаты старта и финиша.

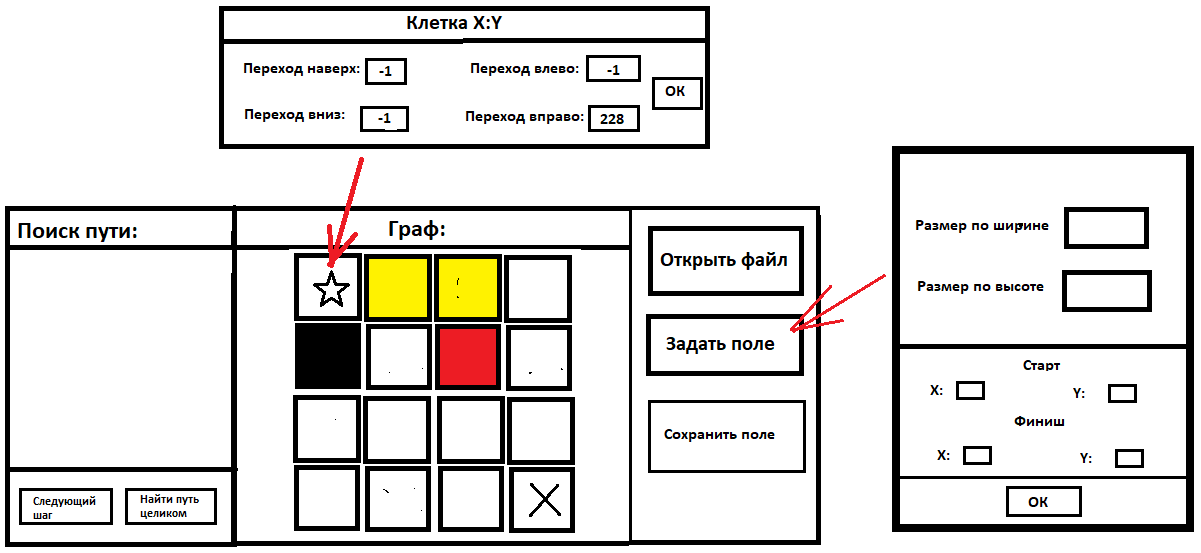


Рисунок 1 – Макет приложения

**1.1. Уточнение требований после сдачи 1-й версии**

Добавить в вывод итоговый путь и суммарный вес полного пути.

**2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ**

**2.1. План разработки**

Приблизительный план разработки:

5 июля – Согласование спецификации и плана разработки

7 июля – Сдача прототипа: разработка диалогового окна, обработка нажатий

10 июля – Сдача 1-й версии: написание алгоритма, визуализация пошагового выполнения

12 июля – Сдача 2-й версии: исправление недочетов

13 июля – Сдача финальной версии и отчета

**2.2. Распределение ролей в бригаде**

Попандопуло А. – интерфейс, классы, отвечающие за визуализацию работы

Чубан Д. – реализация алгоритма и классов карты и клетки

**3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ**

**3.1 Основные структуры данных**

*Position () –* класс, соответствующий координатам клетки. Имеет поля row: Int и column: Int, предназначенные для хранения координаты по вертикали и горизонтали соответсвенно.

*StatePosition –* класс, аналогичный *Position()* с поправкой на то, что типом полей row и column является MutableState<Int>. Этот тип представляет изменяемое состояние переменной. Использование класса *StatePosition* предполагается для хранения позиций стартовой и финишной клеток, а указанный тип полей позволяет отслеживать изменения состояний координат, что необходимо для преобразования элементов графического интерфейса при использовании инструментария Jetpack Compose для построения графического интерфейса.

*enum CellType –* класс, отвечающий за тип клетки. Предусмотрены типы: START, FINISH, WALL, BACKGROUND – тип для стартовой клетки, финишной, тип отражающий наличие стены, и тип обычной клетки соответственно.

*CellData() –* класс соответствующий клетке. Его полями являются:

*type: CellType –* хранитвышеописанный тип клетки.

*position: Position* – хранит позицию клетки.

*isVisited: Boolean* – хранит информацию о том, была ли посещена (просмотрена) клетке в ходе работы алгоритма поиска пути.

*isShortestPath: Boolean* - хранит информацию о том, входит ли клетка в итоговый путь, найденный алгоритмом A\*.

*distance:Int* – хранит расстояние от старта до клетки, используемое в ходе работы алгоритма поиска пути.

*previousShortestCell: CellData?* – хранит информацию о предыдущей клетке в формируемом пути.

*id : Int* – хранит идентификатор клетки

*leftJump: Int* – хранит информацию о стоимости перехода влево относительно данной клетки.

*rightJump: Int* - хранит информацию о стоимости перехода вправо относительно данной клетки.

*downJump: Int* - хранит информацию о стоимости перехода вниз относительно данной клетки.

*uppJump: Int* - хранит информацию о стоимости перехода вверх относительно данной клетки.

*priority: Int* - хранит информацию о приоритете данной клетки (сумме эвристики и расстояния).

*State() –* класс, отвечающий за текущее состояние поля.

Имеет поля:

*gridState: MutableList<MutableList<CellData>>* - непосредственно клетки, имеет приватный доступ.

*height: Int* – высота поля

*width : Int* – ширина поля

*startPosition: StatePosition* – позиция стартовой клетки

*finishPosition: StatePosition* – позиция финишной клетки

*log: MutableState<String>* - отвечает за отслеживание работы алгоритма поиска пути, хранит информацию о производимых алгоритмом действиях.

А также методы:

*clear()* – выполняет очистку рабочего поля.

*addStartAndFinishGrids()* – приватный метод, отвечающий за размещение стартовой и финишной клеток.

*animatedShortestPath (alg :Alg)* – отвечает за запуск полного алгоритма поиска пути посредством экземпляра класса Alg.

*animatedShortestPath\_single (alg : Alg, cells:List<List<CellData>>)* – отвечает за запуск пошагового алгоритма поиска пути.

*FieldReader() –* класс, отвечающий за чтение поля из файла.

*FileWriter() –* класс, отвечающий за запись поля в файл

**3.2 Структуры данных, отвечающие за алгоритм**

*Alg() –* Класс, отвечающий за реализацию алгоритма A\*.

Основные методы:

*AStarWhole() –* метод, реализующий полное выполнение алгоритма(без разбиения на отдельные итерации). Из очереди извлекается клетка с наименьшим приоритетом, помечается как рассмотренная, после чего ее клетки-соседи помещаются в очередь, если они могут быть продолжением пути. Возвращаемый результат – словарь переходов *res*.

*AStarSingle() –* метод, реализующий одну итерацию алгоритма. Так же из очереди извлекается клетка с наименьшим приоритетом и рассматриваются её соседи. В результате работы метод возвращает словарь переходов *res.*

*AddNextCell(x: Int, y: Int, queue:Heap, res: MutableMap<CellData, CellData?>, previousCell:CellData, roadToNew:Int) –* Метод обработки клетки для добавления ее в очередь. В случае если клетка уже рассмотрена или является непроходимой, она не добавляется в очередь. Если же вершина ранее не была добавлена в очередь, или записанное значение приоритета меньше только что вычисленного, в очередь помещается новое значение. В словарь переходов *res* добавляется пара из клетка - родитель.

*retrievePathWhole(res:MutableMap<CellData, CellData?>) –* метод восстанавливает путь от старта до финиша на основе переданного словаря переходов.

*retrievePathSingle(res:MutableMap<CellData, CellData?>) –* метод восстанавливает путь от старта до точки, на которой остановилась очередная итерация на основе переданного словаря переходов.

Для реализации очереди с приоритетом, представленной в виде минимальной двоичной кучи, был написан класс *Heap*:

Основные методы:

*siftUp(index) –* метод, осуществляющий просеивание элемента с индексом *index* вверх.

*siftDown(index) –* метод, осуществляющий просеивание элемента с индексом *index* вниз.

*extractMin() –* метод, извлекающий минимальный элемент из кучи. Первый и последний элементы меняются местами, после чего последний (бывший первый) удаляется из кучи, а первый (бывший последний) просеивается вниз.

*put(element) –* метод, помещает элемент в кучу. Изначально элемент добавляется в конец, после чего просеивается вверх.

*size() –* метод возвращает длину списка, формирующего кучу.

**3.3 Функции, отвечающие за визуализацию**

*fun Cell(cellData: CellData)* – отвечает за визуализацию клетки. Принимает экземпляр вышеописанного класса *CellData*. При нажатии на клетку вызывается отрисовка диалогового окна для ввода значений таких параметров как переход влево, переход вправо, переход вниз и переход вверх, а также значение проходимости клетки, реализованного в формате флажка (чекбокса). Соответствующие поля клетки принимают данные значения.

*fun PathFindingGrid(height : Int, width : Int, cellData: List<CellData>)* - отвечает за визуализацию поля из клеток. Вызывает отрисовку каждой клетки вышеописанной функцией внутри посредством такого компонента *Compose* как *LazyVerticalGrid*.

*fun Legend( label: String,color: Color, hasBorder: Boolean = false)* – отвечает за визуализацию легенды – пояснений значений использумых цветов. Принимает аргументы – строку-пояснение, цвет, и опционально границу.

*PathFind(modifier: Modifier = Modifier, onClick: () -> (Unit), enabled: Boolean = true)* – отвечает за визуализацию кнопки полного поиска пути. Принимает аргумент типа Modifier с указанными настройками визуализации, функцию, выполняющую операции соответствующие нажатию и Boolean значения, отражающее активность кнопки.

Аналогичные функции, отвечающие за визуализцию других кнопок:

* *fun StepPathFind(modifier: Modifier = Modifier, onClick: () -> (Unit), enabled: Boolean = true)* - запуск полного поиска пути;
* *fun ClearButton(modifier: Modifier = Modifier, onClick: () -> (Unit))* – запуска очистки поля;
* *fun OpenFile(modifier: Modifier = Modifier,onClick: () -> (Unit),enabled: Boolean = true)* – осуществляет открытие файла;
* *fun SaveMap(modifier: Modifier = Modifier,onClick: () -> (Unit), enabled: Boolean = true)* – осуществляет сохранение.

*fun SetField(height: Int, width: Int, onSubmit: (Int, Int) -> Unit, startPositionX: Int,startPositionY: Int, finishPositionX: Int, finishPositionY: Int, startSubmitX:(Int) -> Unit, startSubmitY:(Int) -> Unit, finSubmitX:(Int) -> Unit, finSubmitY:(Int) -> Unit)* – отвечает за визуализацию кнопки для изменения поля. Принимает аргументы: высота поля, ширина, функция-действие при изменении размеров поля, координаты стартовой и финишной клеток и функции-действия, соответствующие изменению одной из них. При нажатии на кнопку отрисовывает диалоговое окно для ввода значений параметров: высоты поля, ширины, координат стартовой и финишной клеток.

*fun PathFindingUi(state: State, cells: List<List<CellData>>, onClick: (Position) -> Unit, height: MutableState<Int>, width: MutableState<Int>, startPos : StatePosition, finPos: StatePosition, alg:Alg, log: MutableState<String>, context: Context) -* осуществляет вызов вышеописанных функций в таких компонентах *Jetpack* *Compose*, как *LazyColumn*, и *Row*.

**4. ТЕСТИРОВАНИЕ**

**4.1. Тестирование интерфейса и обработки исключительных ситуаций.**

Рассмотрим набор исключительных ситуаций и реакцию программы на них:

Некорректное задание размеров поля или стартовой/конечной клеток.

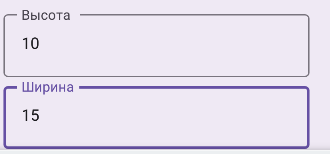


Рисунок 2 – Некорректное задание размера поля

При попытке ввести отрицательные или нулевые значения размеров поля выставляются стандартные значения (Рисунок 2)

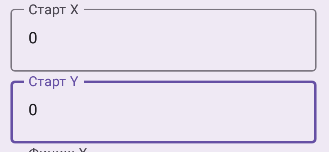


Рисунок 3 – Ввод отрицательных значений

При попытке ввести отрицательные значения координат старта или финиша выставляются нулевые координаты. (Рисунок 3)

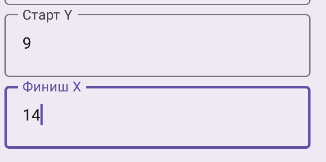


Рисунок 4 – Ввод нулевых значений

При попытке ввести значения, большие, чем размеры поля, выставляются крайние значения размера.(Рисунок 4)

**4.2. Тестирование алгоритма**

1.Работа алгоритма, когда финиш недостижим

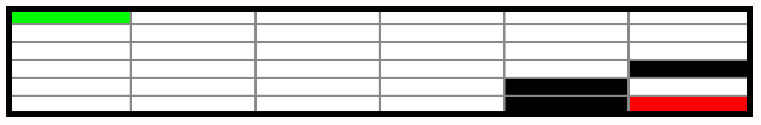


Рисунок 5 – Недостижимый финиш на поле

Итог работы на таком поле и вывод (Рисунок 5 и 6):

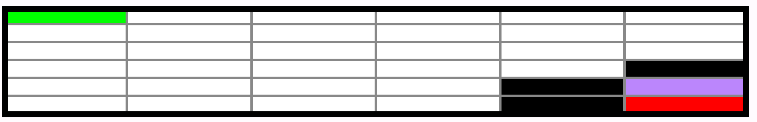




Рисунок 6 – Пример вывода при недостижимом финише

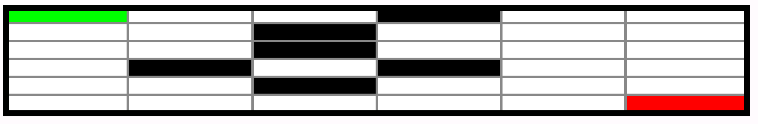
2. Обычная карта с препятствиями (Рисунок 7)

Рисунок 7 – Обычная карта с препятствиями

Результат работы на обычной карте (Рисунок 8):

Рисунок 8 – Результат работы на обычной карте с препятствиями

3. Карта без препятствий (Рисунок 9)

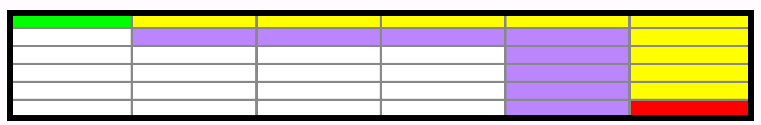


Рисунок 9 – Результат работы на поле без препятствий

4.Результат работы на поле, где старт и финиш совпадают (Рисунок 10)

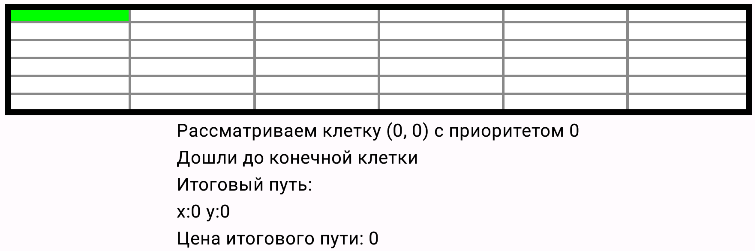


Рисунок 10 – Результат работы при совпадающем старте и финише**ЗАКЛЮЧЕНИЕ**

В ходе выполнения практической работы было реализовано приложение с графическим интерфейсом, демонстрирующее пошаговое выполнение алгоритма А\*. Закреплены навыки программирования на языке Kotlin.

Для написания GUI была изучена библиотека Compose Jetpack.

Итоговая программа соответствует требованиям, предъявленным в начале работы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Репозиторий бригады:

URL: <https://github.com/defrozentruth/summer_practise>

1. Сайт Alexanderklimov.

URL: https://developer.alexanderklimov.ru/android/simplepaint.php

1. Сайт kotlinlang.

URL: https://kotlinlang.ru/docs/reflection.html

1. Сайт metanit.

URL: https://metanit.com/kotlin/jetpack

1. Сайт android.

URL: https://developer.android.com/jetpack/compose