**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: 13.07.2020 | | |
| Дата защиты отчета: 13.07.2020 | | |
|  | | |
| 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 | 15 |
| 4.1 | Тестирование интерфейса и обработки исключительных ситуаций | 15 |
| 4.2 | Тестирование алгоритма | 16 |
|  | Заключение | 18 |
|  | Список использованных источников | 19 |
|  | Приложение А. Исходный код – только в электронном виде | 20 |

**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:

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.

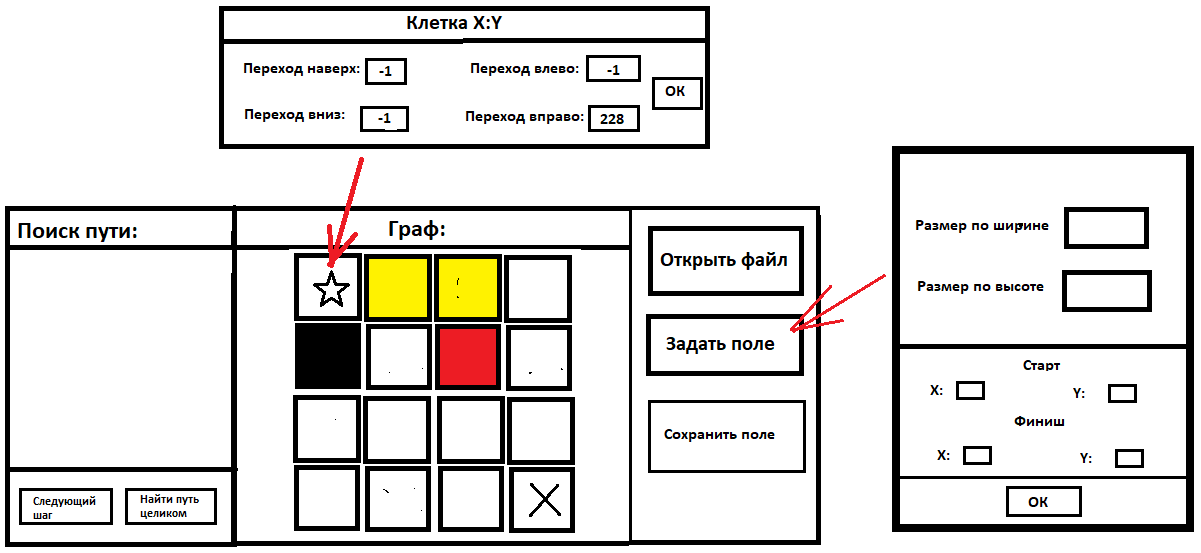


Рисунок 1 – Макет приложения

**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.

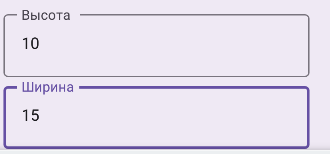
*fun PathFindingUi(state: State, cells: List<List<CellData>>, onClick: (Position) -> Unit, height: MutableState<Int>, width: MutableState<Int>, startPos : StatePosition, finPos: StatePosition, alg:Alg, log: MutableState<String>, context: Context) -* осуществляет вызов вышеописанных функций в таких компонентах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

**4. ТЕСТИРОВАНИЕ**

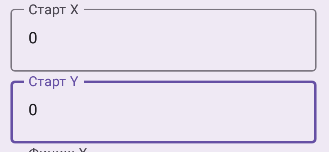
**4.1. Тестирование интерфейса и обработки исключительных ситуаций.**

Рассмотрим набор исключительных ситуаций и реакцию программы на них:

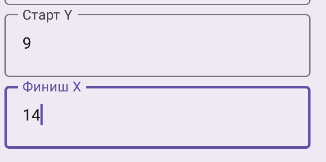
Некорректное задание размеров поля или стартовой/конечной клеток.



При попытке ввести отрицательные или нулевые значения размеров поля выставляются стандартные значения



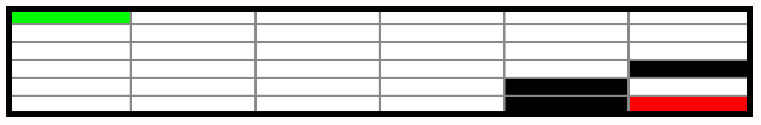
При попытке ввести отрицательные значения координат старта или финиша выставляются нулевые координаты.



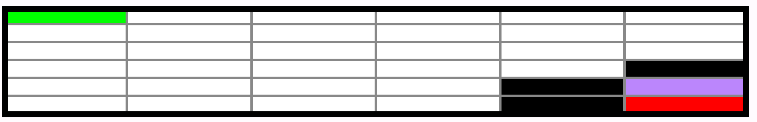
При попытке ввести значения, большие, чем размеры поля, выставляются крайние значения размера.

**4.2. Тестирование алгоритма**

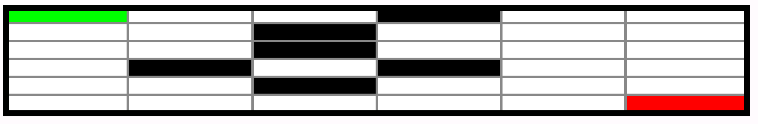
1.Работа алгоритма, когда финиш недостижим



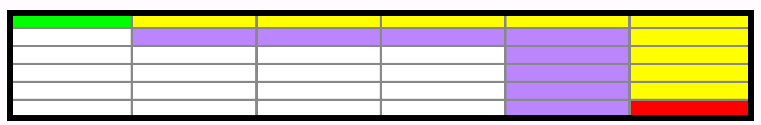
Итог работы на таком поле и вывод:



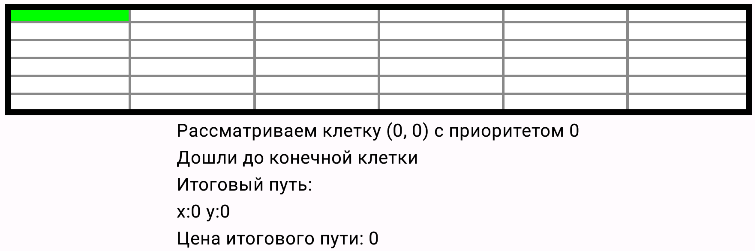


2. Обычная карта с препятствиями

Результат работы:

3. Карта без препятствий

4.Результат работы на поле, где старт и финиш совпадают

**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

**ПРИЛОЖЕНИЕ А**

**НАЗВАНИЕ ПРИЛОЖЕНИЯ**

**Alg:**

import android.util.Log

import com.example.path\_finding\_viz.CellData

import com.example.path\_finding\_viz.CellType

import com.example.path\_finding\_viz.Position

import com.example.path\_finding\_viz.State

import kotlinx.coroutines.delay

import kotlin.math.abs

class Alg(var field: State) {

var nextX = field.startPosition.column.value

var nextY = field.startPosition.row.value

var endedOnX = field.finishPosition.column.value

var endedOnY = field.finishPosition.row.value

//val cells = field.getCells()

var queue = Heap()

var res: MutableMap<CellData, CellData?> = mutableMapOf(field.getCells()[nextY][nextX] to null)

var finSingle:Boolean = false

var log: String = ""

var smallLog: String = ""

var stepPath: String = ""

var way: Int = 0

var processing = true

private var startSingle = true

var singlePath: String = ""

fun clear(){

nextX = field.startPosition.column.value

nextY = field.startPosition.row.value

endedOnX = field.finishPosition.column.value

endedOnY = field.finishPosition.row.value

queue = Heap()

res = mutableMapOf(field.getCells()[nextY][nextX] to null)

finSingle = false

log = ""

smallLog = ""

way = 0

processing = true

startSingle = true

}

private fun heuristic(x:Int, y:Int):Int{

return abs(x-field.finishPosition.column.value)+ abs(y-field.finishPosition.row.value)

}

private fun refreshStart(){

nextX = field.startPosition.column.value

nextY = field.startPosition.row.value

}

suspend fun AStarWhole():Pair<MutableMap<CellData, CellData?>, String>{

processing = true

Log.d("ves", field.getCells()[0][0].rightJump.toString())

refreshStart()

var x = nextX

var y = nextY

val finishX = field.finishPosition.column.value

val finishY = field.finishPosition.row.value

field.setCellVisitedAtPosition(field.getCells()[y][x].position)

queue.put(field.getCells()[y][x])

while(queue.size() != 0){

val cur = queue.extractMin()

if (cur.distance != -1)

way = cur.distance

//println("${cur.position.column} ${cur.position.row}")

//field.setCellShortestAtPosition(cur.position)

Log.d("tired", "${cur.position.row} ------- ${cur.position.column}")

delay(10.toLong())

x = cur.position.column

y = cur.position.row

log += "Рассматриваем клетку ($x, $y) с приоритетом ${cur.priority}\n"

if(x == finishX && y == finishY) {

log += "Дошли до конечной клетки\n"

finSingle = true

break

}

//queue = Heap()

addNextCell(x+1, y, queue, res, field.getCells()[y][x], field.getCells()[y][x].rightJump)

addNextCell(x-1, y, queue, res, field.getCells()[y][x], field.getCells()[y][x].leftJump)

addNextCell(x, y+1, queue, res, field.getCells()[y][x], field.getCells()[y][x].downJump)

addNextCell(x, y-1, queue, res, field.getCells()[y][x], field.getCells()[y][x].uppJump)

}

val shortestPath = retrievePathWhole(res)

Log.d("chego", "${res.size}")

shortestPath.forEach{

field.setCellShortestAtPosition(it.position)

}

if(!finSingle){

log += "Пути от старта до финиша не существует\n"

}

//retrievePathWhole(res)

return Pair (res,log)

}

suspend fun AStarSingle(): Pair<MutableMap<CellData, CellData?>, String>{

if (startSingle)

{

refreshStart()

startSingle = false

}

if(!finSingle){

processing = true

var x = nextX

var y = nextY

val finishX = field.finishPosition.column.value

val finishY = field.finishPosition.row.value

field.setCellVisitedAtPosition(field.getCells()[y][x].position)

queue.put(field.getCells()[y][x])

val cur = queue.extractMin()

way = cur.distance

field.setCellShortestAtPosition(cur.position)

x = cur.position.column

y = cur.position.row

log += "Рассматриваем клетку ($x, $y) с приоритетом ${cur.priority}\n"

smallLog = "Рассматриваем клетку ($x, $y) с приоритетом ${cur.priority}\n"

if (x == finishX && y == finishY) {

log += "Дошли до конечной клетки\n"

smallLog += "Дошли до конечной клетки\n"

finSingle = true

retrievePathSingle(res)

return Pair (res,smallLog)

}

//queue = Heap()

addNextCell(x + 1, y, queue, res, field.getCells()[y][x], field.getCells()[y][x].rightJump)

addNextCell(x - 1, y, queue, res, field.getCells()[y][x], field.getCells()[y][x].leftJump)

addNextCell(x, y + 1, queue, res, field.getCells()[y][x], field.getCells()[y][x].downJump)

addNextCell(x, y - 1, queue, res, field.getCells()[y][x], field.getCells()[y][x].uppJump)

endedOnX = x

endedOnY = y

singlePath += "x:${x} y:${y}\n"

}

// val shortestPath = retrievePathSingle(res)

// Log.d("gde log", smallLog)

//

// shortestPath.forEach{

// field.setCellShortestAtPosition(it.position)

// }

if(!finSingle && queue.size() == 0){

log += "Пути от старта до финиша не существует\n"

smallLog += "Пути от старта до финиша не существует\n"

}

retrievePathSingle(res)

return Pair(res, smallLog)

}

private suspend fun addNextCell(x: Int, y: Int, queue:Heap, res: MutableMap<CellData, CellData?>, previousCell:CellData, roadToNew:Int){

if( x < 0 || x >= field.width || y < 0 || y >= field.height) {

log += "Не можем добавить в очередь клетку ($x, $y), т.к. ее не существует\n"

smallLog += "Не можем добавить в очередь клетку ($x, $y), т.к. ее не существует\n"

return

}

if(field.getCells()[y][x].type == CellType.WALL) {

log += "Не можем добавить в очередь клетку ($x, $y), т.к.она непроходима\n"

smallLog += "Не можем добавить в очередь клетку ($x, $y), т.к.она непроходима\n"

return

}

if(field.getCells()[y][x].isVisited) {

log += "Не можем добавить в очередь клетку ($x, $y), т.к.она уже рассмотрена\n"

smallLog += "Не можем добавить в очередь клетку ($x, $y), т.к.она уже рассмотрена\n"

return

}

field.setCellVisitedAtPosition(Position(y,x))

val newCell = field.getCells()[y][x]

val newDistance = previousCell.distance+roadToNew

if(newCell.distance == -1 || newCell.distance > newDistance){

res[newCell] = previousCell

newCell.distance = newDistance

newCell.priority = heuristic(x, y) + newCell.distance

log += "Добавляем в очередь клетку (${newCell.position.column}, ${newCell.position.row}) с приоритетом ${newCell.priority}\n"

smallLog += "Добавляем в очередь клетку (${newCell.position.column}, ${newCell.position.row}) с приоритетом ${newCell.priority}\n"

queue.put(newCell)

delay(10.toLong())

}

val cur = queue.extractMin()

nextX = cur.position.column

nextY = cur.position.row

queue.put(cur)

}

fun retrievePathWhole(res:MutableMap<CellData, CellData?>): MutableList<CellData>{

val path = emptyList<CellData>().toMutableList()

var curr: CellData? = field.getCells()[field.finishPosition.row.value][field.finishPosition.column.value]

while(curr != null){

path.add(curr)

curr = res[curr]

}

path.reverse()

log += "Итоговый путь:\n"

for (elem in path)

log += "x:${elem.position.column} y:${elem.position.row}\n"

log += "Цена итогового пути: ${way}\n"

return path

}

fun retrievePathSingle(res:MutableMap<CellData, CellData?>): MutableList<CellData>{

val path = emptyList<CellData>().toMutableList()

var curr: CellData? = field.getCells()[endedOnY][endedOnX]

if(finSingle)

curr = field.getCells()[field.finishPosition.row.value][field.finishPosition.column.value]

while(curr != null){

path.add(curr)

curr = res[curr]

}

path.reverse()

if(finSingle){

smallLog += "Итоговый путь:\n"

smallLog += singlePath

smallLog += "Цена итогового пути: ${way}\n"

processing = false

}

return path

}

fun refresh(force:Boolean = false){

if(finSingle || force){

nextX = field.startPosition.column.value

nextY = field.startPosition.row.value

endedOnX = field.finishPosition.column.value

endedOnY = field.finishPosition.row.value

queue = Heap()

res = mutableMapOf(field.getCells()[nextY][nextX] to null)

finSingle = false

log = ""

smallLog = ""

way = 0

processing = true

}

}

}

**FieldReader:**

import android.content.Context

import androidx.compose.runtime.Composable

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import com.example.path\_finding\_viz.CellType

import com.example.path\_finding\_viz.StatePosition

import java.io.File

import com.example.path\_finding\_viz.State

import Alg

class FieldReader(private val context: Context) {

fun readField(filename: String, field: State, alg: Alg){

//val file = File(filename)

context.openFileInput(filename).bufferedReader().useLines { data ->

//val lines = lines.first()

val lines = data.toList()

val sizeX = lines[0].split(" ")[0].toInt()

val sizeY = lines[0].split(" ")[1].toInt()

val startX = lines[1].split(" ")[0].toInt()

val startY = lines[1].split(" ")[1].toInt()

val finishX = lines[2].split(" ")[0].toInt()

val finishY = lines[2].split(" ")[1].toInt()

field.width = sizeX

field.height = sizeY

field.startPosition.column.value = startX

field.startPosition.row.value = startY

field.finishPosition.column.value = finishX

field.finishPosition.row.value = finishY

field.custom\_init()

//val startPos = StatePosition(remember {mutableStateOf(startX)}, remember {mutableStateOf(startY)})

//val finPos = StatePosition(remember {mutableStateOf(finishX)}, remember {mutableStateOf(finishY)})

var i = 0

var j = 0

for(k in 3 until lines.size){

if(k % 2 != 0){

field.gridState[i][j].uppJump = lines[k].split(" ")[0].toInt()

field.gridState[i][j].rightJump = lines[k].split(" ")[1].toInt()

field.gridState[i][j].downJump = lines[k].split(" ")[2].toInt()

field.gridState[i][j].leftJump = lines[k].split(" ")[3].toInt()

field.gridState[i][j].isVisited = false

if(i == startY && j == startX)

field.gridState[i][j].type = CellType.START

else

if (i == finishY && j == finishX)

field.gridState[i][j].type = CellType.FINISH

else

field.gridState[i][j].type = CellType.BACKGROUND

}else{

if(lines[k] == "Unpassable") {

field.gridState[i][j].type = CellType.WALL

}

if(j == sizeX-1){

j = 0

i+=1

}else{

j+=1

}

}

alg.nextX = startX

alg.nextY = startY

alg.endedOnX = finishX

alg.endedOnY = finishY

alg.finSingle = false

alg.log = ""

alg.smallLog = ""

}

}}

}

**FieldWriter:**

import android.content.Context

import androidx.compose.runtime.Composable

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import com.example.path\_finding\_viz.CellType

import com.example.path\_finding\_viz.StatePosition

import java.io.File

import com.example.path\_finding\_viz.State

class FieldWriter (private val context: Context){

fun writeField(field:State, filename: String){

context.openFileOutput(filename, Context.MODE\_PRIVATE).use { out ->

out.write("${field.width} ${field.height}\n".toByteArray())

out.write("${field.startPosition.column.value} ${field.startPosition.row.value}\n".toByteArray())

out.write("${field.finishPosition.column.value} ${field.finishPosition.row.value}\n".toByteArray())

for(i in 0 until field.height)

for(j in 0 until field.width){

out.write("${field.getCells()[i][j].uppJump} ${field.getCells()[i][j].leftJump} ${field.getCells()[i][j].downJump} ${field.getCells()[i][j].rightJump}\n".toByteArray())

if(field.getCells()[i][j].type != CellType.WALL){

if(i == field.height-1 && j == field.width-1)

out.write("Normal".toByteArray())

else

out.write("Normal\n".toByteArray())

}

else{

if(i == field.height-1 && j == field.width-1)

out.write("Unpassable".toByteArray())

else

out.write("Unpassable\n".toByteArray())

}

}

}

}

}

**Heap:**

import com.example.path\_finding\_viz.CellData

class Heap(){

var queue = emptyList<CellData>().toMutableList()

fun siftUp(index: Int){

if(index < 0 || index >= this.queue.size){

return

}

var tmpIndex = index

var parent = (index - 1)/2

while(tmpIndex > 0 && this.queue[parent].priority >= this.queue[tmpIndex].priority){

val tmp = this.queue[parent]

this.queue[parent] = this.queue[tmpIndex]

this.queue[tmpIndex] = tmp

val buf = tmpIndex

tmpIndex = parent

parent = (buf-1)/2

}

}

fun siftDown(index: Int){

if (index < 0 || index >= this.queue.size) {

return

}

var minIndex = index

var tmpIndex = index

var left: Int

var right: Int

while(true) {

left = 2\*tmpIndex+1

right = 2\*tmpIndex+2

if(right < this.queue.size && this.queue[right].priority < this.queue[minIndex].priority)

minIndex = right

if(left < this.queue.size && this.queue[left].priority < this.queue[minIndex].priority)

minIndex = left

if(minIndex == tmpIndex)

return

else{

this.queue[tmpIndex]; this.queue[minIndex] = this.queue[minIndex]; this.queue[tmpIndex]

tmpIndex = minIndex

}

}

}

fun extractMin(): CellData{

val min\_element = this.queue[0]

this.queue[0] = this.queue[this.queue.size-1]

this.queue.removeAt(this.queue.size - 1)

this.siftDown(0)

return min\_element

}

fun put(element: CellData){

this.queue.add(element)

this.siftUp(this.size() - 1)

}

fun size(): Int{

return this.queue.size

}

}

**a\_star.kt:**

import android.util.Log

import com.example.path\_finding\_viz.CellData

import com.example.path\_finding\_viz.StatePosition

import com.example.path\_finding\_viz.Position

import com.example.path\_finding\_viz.State

suspend fun startA\_star(gridState: State, alg:Alg): Pair<List<CellData>, String> {

//val alg = Alg(gridState)

val value = alg.AStarWhole()

val map = value.first

val path = alg.retrievePathWhole(map)

return Pair (path,value.second)

//animatedDijkstra(gridState)

//return getShortestPathOrder(gridState.getFinishCell())

}

suspend fun startA\_star\_single(gridState: State, alg: Alg): Pair<List<CellData>, String> {

val value = alg.AStarSingle()

val map = value.first

val path = alg.retrievePathSingle(map)

Log.d("checknu", "${alg.smallLog} -------- ${value.second}")

return Pair (path,value.second)

//animatedDijkstra(gridState)

//return getShortestPathOrder(gridState.getFinishCell())

}

fun CellData.isAtPosition(position: Position) =

this.position.row == position.row && this.position.column == position.column

fun CellData.isAtPosition(position: StatePosition) =

this.position.row == position.row.value && this.position.column == position.column.value

**Buttons.kt:**

package com.example.path\_finding\_viz

import androidx.compose.foundation.ExperimentalFoundationApi

import androidx.compose.material3.Button

import androidx.compose.material3.ButtonColors

import androidx.compose.material3.ButtonDefaults

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

@ExperimentalFoundationApi

@Composable

fun PathFind(modifier: Modifier = Modifier, onClick: () -> (Unit), enabled: Boolean = true) {

ButtonWithText(

modifier,

onClick = onClick,

label = "PathFind",

enabled = enabled,

colors = ButtonDefaults.buttonColors(containerColor = Color.Blue)

)

}

@ExperimentalFoundationApi

@Composable

fun StepPathFind(modifier: Modifier = Modifier, onClick: () -> (Unit), enabled: Boolean = true) {

ButtonWithText(

modifier,

onClick = onClick,

label = "StepFind",

enabled = enabled,

colors = ButtonDefaults.buttonColors(containerColor = Color.Magenta)

)

}

@ExperimentalFoundationApi

@Composable

fun ClearButton(modifier: Modifier = Modifier, onClick: () -> (Unit)) {

ButtonWithText(

modifier,

onClick = onClick,

label = "Clear",

colors = ButtonDefaults.buttonColors(containerColor = Color.DarkGray)

)

}

@ExperimentalFoundationApi

@Composable

fun OpenFile(

modifier: Modifier = Modifier,

onClick: () -> (Unit),

enabled: Boolean = true

) {

ButtonWithText(

modifier,

onClick = onClick,

label = "Open file",

enabled = enabled,

colors = ButtonDefaults.buttonColors(containerColor = Color.Black)

)

}

@ExperimentalFoundationApi

@Composable

fun SaveMap(

modifier: Modifier = Modifier,

onClick: () -> (Unit),

enabled: Boolean = true

) {

ButtonWithText(

modifier,

onClick = onClick,

label = "Save map",

enabled = enabled,

colors = ButtonDefaults.buttonColors(containerColor = Color.Green)

)

}

@ExperimentalFoundationApi

@Composable

private fun ButtonWithText(

modifier: Modifier = Modifier,

enabled: Boolean = true,

label: String,

onClick: () -> (Unit),

colors: ButtonColors

) {

Button(onClick = onClick, modifier = modifier, colors = colors, enabled = enabled) {

Text(text = label, color = Color.White)

}

}

**CellData.kt:**

package com.example.path\_finding\_viz

import android.util.Log

import androidx.compose.animation.animateColorAsState

import androidx.compose.animation.core.tween

import androidx.compose.foundation.BorderStroke

import androidx.compose.foundation.background

import androidx.compose.foundation.border

import androidx.compose.foundation.clickable

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.rememberScrollState

import androidx.compose.foundation.text.KeyboardOptions

import androidx.compose.foundation.verticalScroll

import androidx.compose.material3.AlertDialog

import androidx.compose.material3.Button

import androidx.compose.material3.Checkbox

import androidx.compose.material3.ExperimentalMaterial3Api

import androidx.compose.material3.Text

import androidx.compose.material3.TextField

import androidx.compose.runtime.Composable

import androidx.compose.runtime.MutableState

import androidx.compose.runtime.getValue

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import androidx.compose.runtime.setValue

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.text.input.KeyboardType

import androidx.compose.ui.unit.dp

data class CellData(

var type: CellType,

val position: Position,

var isVisited: Boolean = false,

var isShortestPath: Boolean = false,

var distance: Int = -1,

var previousShortestCell: CellData? = null,

var id: Int = (0..Int.MAX\_VALUE).random(),

var leftJump: Int = 1,

var rightJump: Int = 1,

var downJump: Int = 1,

var uppJump: Int = 1,

var priority: Int = 0

){

override fun hashCode(): Int {

return id

}

fun print () : String{

if (isShortestPath)

return "--- shortPath"

if (isVisited)

return "--- visited"

if (type == CellType.START)

return "--- start"

if (type == CellType.FINISH)

return "--- finish"

return "no"

}

override fun equals(other: Any?): Boolean {

if (this === other) return true

if (javaClass != other?.javaClass) return false

other as CellData

if (type != other.type) return false

if (position != other.position) return false

if (isVisited != other.isVisited) return false

if (isShortestPath != other.isShortestPath) return false

if (distance != other.distance) return false

if (previousShortestCell != other.previousShortestCell) return false

if (id != other.id) return false

if (leftJump != other.leftJump) return false

if (rightJump != other.rightJump) return false

if (downJump != other.downJump) return false

if (uppJump != other.uppJump) return false

if (priority != other.priority) return false

return true

}

}

enum class CellType {

START,

FINISH,

WALL,

BACKGROUND,

}

data class Position(

var row: Int,

var column: Int

)

data class StatePosition(

var row: MutableState<Int>,

var column: MutableState<Int>

)

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun Cell(cellData: CellData) {

val bgColor = animateColorAsState(

targetValue = getBackgroundByType(cellData),

animationSpec = tween(durationMillis = 700)

)

val showDialog = remember { mutableStateOf(false) }

val leftJump = remember { mutableStateOf(cellData.leftJump) }

val rightJump = remember { mutableStateOf(cellData.rightJump) }

val downJump = remember { mutableStateOf(cellData.downJump) }

val uppJump = remember { mutableStateOf(cellData.uppJump) }

val passability = remember { mutableStateOf(cellData.type) }

val boxModifier = Modifier

.padding(0.dp)

.border(BorderStroke(1.dp, Color.Gray))

.height(16.dp)

.background(bgColor.value)

.fillMaxWidth()

.clickable { showDialog.value = true }

if (showDialog.value) {

AlertDialog(

onDismissRequest = { showDialog.value = false },

title = { Text("Введите параметры") },

text = {

Column (

modifier = Modifier.verticalScroll(rememberScrollState())

){

TextField(value = leftJump.value.toString(), onValueChange = { leftJump.value = it.toIntOrNull() ?: 0 }, label = { Text("Шаг влево") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

)

)

TextField(value = rightJump.value.toString(), onValueChange = { rightJump.value = it.toIntOrNull() ?: 0 }, label = { Text("Шаг вправо") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

))

TextField(value = downJump.value.toString(), onValueChange = { downJump.value = it.toIntOrNull() ?: 0}, label = { Text("Шаг вниз") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

))

TextField(value = uppJump.value.toString(), onValueChange = { uppJump.value = it.toIntOrNull() ?: 0 }, label = { Text("Шаг вверх") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

))

Row {

Text("Проходимость (не): ")

Checkbox(checked = (passability.value == CellType.WALL), onCheckedChange = {

passability.value = if (it) CellType.WALL else CellType.BACKGROUND

cellData.type = passability.value

})

}

}

},

confirmButton = {

Button(onClick = {

cellData.leftJump = leftJump.value

cellData.rightJump = rightJump.value

cellData.downJump = downJump.value

cellData.uppJump = uppJump.value

showDialog.value = false

}) {

Text("OK")

}

}

)

}

Box(modifier = boxModifier)

}

val Purple200 = Color(0xFFBB86FC)

val CELL\_BACKGROUND = Color.White

val CELL\_START = Color.Red

val CELL\_FINISH = Color.Green

val CELL\_VISITED = Purple200

val CELL\_PATH = Color.Yellow

val CELL\_WALL = Color.Black

private fun getBackgroundByType(cellData: CellData): Color {

if (cellData.isShortestPath && cellData.type != CellType.START && cellData.type != CellType.FINISH)

{

Log.d("shortColor", "here")

return CELL\_PATH

}

if (cellData.isVisited && cellData.type != CellType.START && cellData.type != CellType.FINISH) {

Log.d("visittColor", "here")

return CELL\_VISITED

}

return when (cellData.type) {

CellType.BACKGROUND -> CELL\_BACKGROUND

CellType.WALL -> CELL\_WALL

CellType.START -> CELL\_START

CellType.FINISH -> CELL\_FINISH

}

}

**Grid.kt:**

package com.example.path\_finding\_viz

import android.util.Log

import androidx.compose.foundation.BorderStroke

import androidx.compose.foundation.ExperimentalFoundationApi

import androidx.compose.foundation.border

import androidx.compose.foundation.horizontalScroll

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.lazy.grid.GridCells

import androidx.compose.foundation.lazy.grid.LazyVerticalGrid

import androidx.compose.foundation.lazy.grid.items

import androidx.compose.foundation.lazy.items

import androidx.compose.foundation.rememberScrollState

import androidx.compose.runtime.Composable

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.unit.dp

@ExperimentalFoundationApi

@Composable

fun PathFindingGrid(

height : Int,

width : Int,

cellData: List<CellData>

) {

val fix\_wid = if (width != 0) width else 1

LazyVerticalGrid(

columns = GridCells.Fixed(fix\_wid),

modifier = Modifier

.height((16 \* (height) + 2+5).dp)

.padding(2.dp)

.border(BorderStroke(5.dp, Color.Black))

) {

items(cellData) {

Column(horizontalAlignment = Alignment.CenterHorizontally) {

Cell(it)

}

}

}

}

**Legend.kt:**

package com.example.path\_finding\_viz

import androidx.compose.foundation.BorderStroke

import androidx.compose.foundation.ExperimentalFoundationApi

import androidx.compose.foundation.background

import androidx.compose.foundation.border

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.layout.width

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.unit.dp

@ExperimentalFoundationApi

@Composable

fun Legend(

label: String,

color: Color,

hasBorder: Boolean = false

) {

val boxModifier = Modifier

.border(BorderStroke(if (hasBorder) 0.5.dp else 0.dp, Color.Gray))

.padding(4.dp)

.height(if (hasBorder) 10.dp else 16.dp)

.background(color)

.width(if (hasBorder) 10.dp else 16.dp)

Box(modifier = boxModifier)

Text(text = label, color = Color.Black)

}

**MainActivity.kt:**

package com.example.path\_finding\_viz

import Alg

import FieldReader

import FieldWriter

import android.content.Context

import android.os.Bundle

import android.util.Log

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.ExperimentalFoundationApi

import androidx.compose.foundation.layout.Arrangement

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.Surface

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.runtime.LaunchedEffect

import androidx.compose.runtime.MutableState

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.unit.dp

import com.example.path\_finding\_viz.ui.theme.Path\_finding\_vizTheme

import kotlinx.coroutines.CoroutineScope

import kotlinx.coroutines.Dispatchers

import kotlinx.coroutines.coroutineScope

import kotlinx.coroutines.delay

import kotlinx.coroutines.launch

import java.io.FileReader

private val scope = CoroutineScope(Dispatchers.Default)

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

val context: Context = applicationContext

super.onCreate(savedInstanceState)

setContent {

Path\_finding\_vizTheme {

// A surface container using the 'background' color from the theme

Surface(

//modifier = Modifier.fillMaxSize(),

color = MaterialTheme.colorScheme.background

) {

PathFindingApp(context)

}

}

}

}

}

@Composable

fun PathFindingApp(context :Context){

val height = remember { mutableStateOf(6) }

val width = remember { mutableStateOf(6) }

val log = remember {

mutableStateOf("")

}

val startPos = StatePosition(remember {

mutableStateOf(5)

}, remember {

mutableStateOf(5)

})

val finPos = StatePosition(remember {

mutableStateOf(0)

}, remember {

mutableStateOf(0)

})

val state = remember(height.value, width.value, startPos, finPos, log) { State(height.value, width.value, startPos, finPos, log) }

val currentGridState = remember(state, startPos, finPos) { mutableStateOf(state.drawCurrentGridState()) }

val alg = remember (state,height.value, width.value, startPos, finPos, log){

(Alg(state))

}

val onCellClicked = { p: Position -> // пока не используется, перерисовка клеток при изменении перехоов в них не предусмотрена

if (state.isPositionNotAtStartOrFinish(p) && !state.isVisualizing) {

currentGridState.value = state.drawCurrentGridState()

}

}

PathFindingUi(state, currentGridState.value, onCellClicked, height, width, startPos, finPos, alg, log, context)

LaunchedEffect(Unit) {

while (true) {

delay(10.toLong())

currentGridState.value = state.drawCurrentGridState()

}

}

}

@OptIn(ExperimentalFoundationApi::class)

@Composable

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) {

val isVisualizeEnabled = remember { mutableStateOf(true) }

val onPathfind: () -> Unit = {

Log.d("cell", state.printCell(0,0))

Log.d("startStata", "${alg.field.startPosition.column.value},----- ${alg.field.startPosition.row.value}")

refreshCells(cells, state, true)

Log.d("cell", state.printCell(0,0))

scope.launch {coroutineScope{ state.animatedShortestPath(alg)}

refreshCells(cells, state, false)

height.value -=1

height.value +=1

isVisualizeEnabled.value = false

}

//state.animatedShortestPath(alg)

//cells[1][0].isShortestPath = state.getCell(0,1).isShortestPath

}

val onStepPathfind: () -> Unit = {

refreshCells(cells, state, true)

alg.refresh()

scope.launch { state.animatedShortestPath\_single(alg, cells, height) }

isVisualizeEnabled.value = true

}

val onCleared: () -> Unit = {

state.clear()

alg.clear()

refreshCells(cells, state, reverse = false)

alg.refresh()

height.value -= 1

height.value += 1

isVisualizeEnabled.value = true

}

val onOpenFile: () -> Unit = {

Log.d("shock1", "${state.height} ---- ${state.width}\n ${state.finishPosition.column.value} && ${state.finishPosition.row.value} ===============================")

val loader = FieldReader(context)

loader.readField(filename = "field.txt", state, alg)

height.value = state.height

width.value = state.width

alg.refresh(force = true)

Log.d("shock2", "${state.height} ---- ${state.width}\n ${state.finishPosition.column.value} && ${state.finishPosition.row.value} ===============================")

}

val onSaveMap: () -> Unit = {

val saver = FieldWriter(context)

saver.writeField(state, "field.txt")

}

LazyColumn(

modifier = Modifier

.padding(8.dp),

verticalArrangement = Arrangement.Center,

horizontalAlignment = Alignment.CenterHorizontally,

) {

item {

PathFindingGrid(height.value, width.value, cells.toLinearGrid()) // рисует поле

}

Log.d("mypainnew", "-${state.log}")

item {

Text(text = log.value, color = Color.Black)

}

item{

Row(modifier = Modifier.padding(8.dp)) {

PathFind(

modifier = Modifier.padding(start = 16.dp),

onClick = onPathfind,

enabled = isVisualizeEnabled.value

)

StepPathFind(

modifier = Modifier.padding(start = 16.dp),

onClick = onStepPathfind,

enabled = isVisualizeEnabled.value

)

OpenFile(

modifier = Modifier.padding(start = 16.dp),

onClick = onOpenFile,

enabled = isVisualizeEnabled.value

)

SaveMap(

modifier = Modifier.padding(start = 16.dp),

onClick = onSaveMap,

enabled = isVisualizeEnabled.value

)

ClearButton(modifier = Modifier.padding(horizontal = 16.dp), onCleared)

SetField(

height = height.value,

width = width.value,

onSubmit = { n1 :Int, n2:Int ->

val n1\_checked = if (n1 != 0 ) n1 else 10

val n2\_checked = if (n2!= 0 ) n2 else 15

if (startPos.column.value > n2\_checked){

cells[startPos.row.value][startPos.column.value].type = CellType.BACKGROUND

startPos.column.value = n2\_checked-1

cells[startPos.row.value][startPos.column.value].type = CellType.START

}

if (startPos.row.value > n1\_checked){

cells[startPos.row.value][startPos.column.value].type = CellType.BACKGROUND

startPos.row.value = n1\_checked-1

cells[startPos.row.value][startPos.column.value].type = CellType.START

}

if (finPos.column.value > n2\_checked){

cells[finPos.row.value][finPos.column.value].type = CellType.BACKGROUND

finPos.column.value = n2\_checked-1

cells[finPos.row.value][finPos.column.value].type = CellType.START

}

if (finPos.row.value > n1\_checked){

cells[finPos.row.value][finPos.column.value].type = CellType.BACKGROUND

finPos.row.value = n1\_checked-1

cells[finPos.row.value][finPos.column.value].type = CellType.START

}

height.value = n1\_checked

width.value = n2\_checked

}, startPos.column.value,startPos.row.value, finPos.column.value,finPos.row.value,

{ startPosX :Int->

cells[startPos.row.value][startPos.column.value].type = CellType.BACKGROUND

//state.updateCellTypeAtPosition(Position(startPos.row.value, startPos.column.value), CellType.BACKGROUND)

if (startPosX < width.value)

startPos.column.value = startPosX

else

startPos.column.value = width.value-1

onCleared()

//state.updateCellTypeAtPosition(Position(startPos.row.value, startPos.column.value), CellType.START)

cells[startPos.row.value][startPos.column.value].type = CellType.START

height.value -= 1

height.value += 1

},

{ startPosY :Int->

cells[startPos.row.value][startPos.column.value].type = CellType.BACKGROUND

if (startPosY <height.value)

startPos.row.value = startPosY

else

startPos.row.value = height.value -1

onCleared()

cells[startPos.row.value][startPos.column.value].type = CellType.START

height.value -= 1

height.value += 1

},

{ finishPosX :Int->

cells[finPos.row.value][finPos.column.value].type = CellType.BACKGROUND

if (finishPosX < width.value)

finPos.column.value = finishPosX

else

finPos.column.value = width.value -1

onCleared()

cells[finPos.row.value][finPos.column.value].type = CellType.FINISH

height.value -= 1

height.value += 1

},

{ finishPosY :Int->

cells[finPos.row.value][finPos.column.value].type = CellType.BACKGROUND

if (finishPosY < height.value)

finPos.row.value = finishPosY

else

finPos.row.value = height.value-1

onCleared()

cells[finPos.row.value][finPos.column.value].type = CellType.FINISH

height.value -= 1

height.value += 1

}

)

}

Row(modifier = Modifier.padding(4.dp)) {

Legend("Start", CELL\_START)

Legend("Finish", CELL\_FINISH)

Legend("Visited", CELL\_VISITED)

Legend("Wall", CELL\_WALL)}

}

}

}

**SetField.kt:**

package com.example.path\_finding\_viz

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.rememberScrollState

import androidx.compose.foundation.text.KeyboardOptions

import androidx.compose.foundation.verticalScroll

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.filled.Edit

import androidx.compose.material3.AlertDialog

import androidx.compose.material3.Button

import androidx.compose.material3.ExperimentalMaterial3Api

import androidx.compose.material3.FloatingActionButton

import androidx.compose.material3.Icon

import androidx.compose.material3.OutlinedTextField

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.text.input.KeyboardType

@OptIn(ExperimentalMaterial3Api::class)

@Composable

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){ // состояние, которое будет хранить значения, введенные пользователем в диалоговом окне

val showDialog = remember { mutableStateOf(false) } // состояние, которое будет хранить флаг, показывающий, нужно ли отображать диалоговое окно

val onClick = {

showDialog.value = true

}

// Определяем диалоговое окно

if (showDialog.value) {

AlertDialog(

onDismissRequest = { showDialog.value = false },

title = { Text("Введите значения") },

text = {

// Отображаем поля ввода для двух целочисленных значений

Column (

modifier = Modifier.verticalScroll(rememberScrollState())

){

OutlinedTextField(

value = height.toString(),

onValueChange = { onSubmit(it.toIntOrNull() ?: 0, width) },

label = { Text("Высота ") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

)

)

OutlinedTextField(

value = width.toString(),

onValueChange = { onSubmit(height, it.toIntOrNull() ?: 0) },

label = { Text("Ширина") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

)

)

OutlinedTextField(

value = startPositionX.toString(),

onValueChange = { startSubmitX(it.toIntOrNull()?:0) },

label = { Text("Старт X") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

)

)

OutlinedTextField(

value = startPositionY.toString(),

onValueChange = { startSubmitY(it.toIntOrNull()?:0) },

label = { Text("Старт Y") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

)

)

OutlinedTextField(

value = finishPositionX.toString(),

onValueChange = { finSubmitX(it.toIntOrNull()?:0) },

label = { Text("Финиш X") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

)

)

OutlinedTextField(

value = finishPositionY.toString(),

onValueChange = { finSubmitY(it.toIntOrNull()?:0) },

label = { Text("Финиш Y") },

keyboardOptions = KeyboardOptions(

keyboardType = KeyboardType.Number

)

)

}

},

confirmButton = {

// Определяем кнопку подтверждения, которая сохраняет введенные значения

Button(

onClick = { showDialog.value = false

},

content = { Text("OK") }

)

}

)

}

// Отображаем кнопку на правой стороне экрана

Box(

Modifier.fillMaxSize(),

contentAlignment = Alignment.BottomEnd

) {

FloatingActionButton(

onClick = onClick,

content = { Icon(Icons.Default.Edit, null) }

)

}

}

**State.kt:**

package com.example.path\_finding\_viz

import Alg

import android.util.Log

import androidx.compose.runtime.MutableState

import startA\_star

import startA\_star\_single

class State (var height: Int, var width: Int, var startPosition: StatePosition, var finishPosition: StatePosition, var log: MutableState<String>) {

var gridState: MutableList<MutableList<CellData>> = mutableListOf()

var isVisualizing = false

private set

init {

if (height == 0)

height = 1

if (width == 0)

width = 1

clear()

}

fun clear() {

gridState = getInitGridState()

Log.d("beda", "refresh\_grid")

isVisualizing = false

log.value = ""

addStartAndFinishGrids()

Log.d("bedaExtra", "${this.startPosition.column.value} ------- ${this.startPosition.row.value}")

}

fun custom\_init(){

gridState = getInitGridState()

addStartAndFinishGrids()

}

fun drawCurrentGridState(): List<List<CellData>> {

val updatedGrid = getInitGridState()

for (i in 0 until updatedGrid.size) {

for (j in 0 until updatedGrid[i].size) {

updatedGrid[i][j] = gridState[i][j]

}

}

return updatedGrid

}

private fun getInitGridState() = getGridWithClearBackground(height, width)

private fun addStartAndFinishGrids() {

Log.d("bedad", "start ${this.startPosition.column.value} ------- ${this.startPosition.row.value}")

gridState[startPosition.row.value][startPosition.column.value] =

CellData(CellType.START, Position(startPosition.row.value, startPosition.column.value), distance = 0)

gridState[finishPosition.row.value][finishPosition.column.value] =

CellData(CellType.FINISH, Position(finishPosition.row.value, finishPosition.column.value))

Log.d("bedad", "finish ${this.finishPosition.column.value} ------- ${this.finishPosition.row.value}")

}

fun getCellAtPosition(p: Position) = gridState[p.row][p.column]

private fun getCellAtPosition(p: StatePosition) = gridState[p.row.value][p.column.value]

fun getCells() = gridState

fun getCurrentGrid(): List<List<CellData>> = gridState

fun setCellVisitedAtPosition(p: Position) {

gridState[p.row][p.column] = getCellAtPosition(p).copy(isVisited = true)

}

fun setCellShortestAtPosition(p: Position) {

gridState[p.row][p.column] = getCellAtPosition(p).copy(isShortestPath = true)

}

fun isPositionNotAtStartOrFinish(p: Position) =

getCellAtPosition(p).type != CellType.START &&

getCellAtPosition(p).type != CellType.FINISH

fun toggleCellTypeToWall(p: Position) {

if (getCellAtPosition(p).type == CellType.WALL) {

updateCellTypeAtPosition(p, CellType.BACKGROUND)

} else {

updateCellTypeAtPosition(p, CellType.WALL)

}

}

fun printCell(x:Int, y: Int):String {

if (gridState[y][x].isShortestPath)

return "[$y][$x]--- shortPath"

if (gridState[y][x].isVisited)

return "[$y][$x]--- visited"

if (gridState[y][x].type == CellType.START)

return "[$y][$x]--- start"

if (gridState[y][x].type == CellType.FINISH)

return "[$y][$x]--- finish"

if (gridState[y][x].type == CellType.WALL)

return "[$y][$x]--- wall"

return "no"

}

fun getCell(x:Int, y: Int): CellData {

return gridState[y][x]

}

fun getFinishCell() = getCellAtPosition(finishPosition)

suspend fun animatedShortestPath(alg: Alg) {

isVisualizing = true

val value = startA\_star(this, alg)

val shortestPath = value.first

log.value = value.second

// shortestPath.forEach {

// val p = it.position

// Log.d("detonator", shortestPath.size.toString())

// this.setCellShortestAtPosition(p)

// //delay(10.toLong())

// }

}

suspend fun animatedShortestPath\_single(alg : Alg, cells:List<List<CellData>>,height: MutableState<Int>) {

isVisualizing = true

val value = startA\_star\_single(this, alg)

refreshCells(cells, this, false, )

log.value = value.second

height.value -= 1

height.value += 1

Log.d("pochemu net", log.value)

}

@JvmName("getFinishPositionMethod")

fun getFinishPosition() = finishPosition

fun updateCellTypeAtPosition(p: Position, cellType: CellType) {

gridState[p.row][p.column] = getCellAtPosition(p).copy(type = cellType)

}

fun refreshCellAtPosition(p: Position, cellType: CellType, isVisited:Boolean, isShortestPath:Boolean, distance: Int,

prevShortest: CellData?, leftJump: Int,rightJump: Int,downJump: Int,uppJump: Int,priority:Int ) {

gridState[p.row][p.column] = getCellAtPosition(p).copy(type = cellType, isVisited = isVisited, isShortestPath = isShortestPath, distance = distance, previousShortestCell = prevShortest, leftJump = leftJump, rightJump = rightJump, uppJump = uppJump, downJump = downJump, priority = priority)

}

}

fun getGridWithClearBackground(height: Int, width: Int): MutableList<MutableList<CellData>> {

val mutableGrid = MutableList(height) {

MutableList(width) {

CellData(CellType.BACKGROUND, Position(0, 0))

}

}

for (i in 0 until height) {

for (j in 0 until width) {

mutableGrid[i][j] = CellData(CellType.BACKGROUND, Position(i, j))

}

}

return mutableGrid

}

**Utils.kt:**

package com.example.path\_finding\_viz

import android.util.Log

fun List<List<CellData>>.toLinearGrid(): MutableList<CellData> {

val mutableList = mutableListOf<CellData>()

for (i in this.indices) {

for (j in this[i].indices) {

mutableList.add(this[i][j])

}

}

return mutableList

}

fun refreshCells(cells: List<List<CellData>>, state:State, reverse :Boolean){

if (!reverse){

for (i in 0 until state.height){

for (j in 0 until state.width){

refreshCell(cells[i][j], state.getCell(j,i))

}

}}

else {

for (i in 0 until state.height){

for (j in 0 until state.width){

Log.d("typeChange?", state.printCell(j,i))

refreshStateCell(state, cells[i][j], i, j)

Log.d("typeChange?", state.printCell(j,i))

}

}

}

Log.d("abaldet", " start kletka v itoge ${ state.printCell(0, 0) }")

}

fun refreshCell (cell1 :CellData, cell2:CellData){

cell1.type = cell2.type

cell1.isShortestPath = cell2.isShortestPath

cell1.distance = cell2.distance

cell1.isVisited = cell2.isVisited

cell1.previousShortestCell = cell2.previousShortestCell

cell1.priority = cell2.priority

cell1.leftJump = cell2.leftJump

cell1.rightJump = cell2.rightJump

cell1.downJump = cell2.downJump

cell1.uppJump = cell2.uppJump

}

fun refreshStateCell (state:State, cell2:CellData, y: Int, x:Int){

state.refreshCellAtPosition(Position(y,x), cell2.type, cell2.isVisited, cell2.isShortestPath, cell2.distance, cell2.previousShortestCell, cell2.leftJump, cell2.rightJump, cell2.downJump, cell2.uppJump, cell2.priority)

}