МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритма A-star

Студентка гр. 8383 _____ Ишанина Л.Н.

Студент гр. 8383 _____ Ларин А.

Студентка гр. 8383 _____ Сырцова Е.А.

Руководитель _____ Размочаева Н.В.

Санкт-Петербург

2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Ишанина Л.Н. группы 8383

Студент Ларин А. группы 8383

Студентка Сырцова Е.А. группы 8383

Тема практики: визуализация алгоритма A-star

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: A-star

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 12.07.2020

Дата защиты отчета: 12.07.2020

Студентка	_____	Ишанина Л.Н.
Студент	_____	Ларин А.
Студентка	_____	Сырцова Е.А.
Руководитель	_____	Размочаева Н.В.

АННОТАЦИЯ

В процессе выполнения задания учебной практики был реализован алгоритм, предназначенный для нахождения кратчайшего пути в графе, представленного в виде сетки. Поиск кратчайшего пути осуществляется с использованием алгоритма A-star. Разработанная программа детально показывает этапы работы алгоритма при построении кратчайшего пути.

Программа была написана на языке программирования Java, в среде разработки IntelliJ Idea.

SUMMARY

During the educational practice task, an algorithm was implemented to find the shortest path in a graph represented as a grid. The shortest path is searched using the A-star algorithm. The developed program shows in detail the steps of the algorithm when constructing the shortest path.

The program was written in the Java programming language, in the IntelliJ Idea development environment.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью выполнения данной работы является закрепление изученного материала программы второго курса, изучение нового языка программирования и выработка практических навыков командной работы.

Задачами данной практической работы являются: реализовать алгоритм поиска A* кратчайшего пути в графе, визуализировать его и предоставить удобный пользовательский интерфейс.

Поиск A* – это алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной). [3]

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость». Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины из начальной и функции эвристической оценки расстояния от рассматриваемой вершины к конечной. [3]

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к вводу исходных данных

На вход алгоритму должен подаваться граф, представленный в виде сетки с установленными стенами, начальной и конечной точкой.

1.1.2. Требования к визуализации

Пользовательский интерфейс должен представлять собой диалоговое окно, содержащее набор кнопок, предназначенных для управления состоянием программы.

Диалоговое окно должно состоять из:

- Рабочей области – сетки для построения графа.
- Набора кнопок для размещения объектов на поле: «BlankCell», «WallCell», «StartCell», «EndCell».
- Набора кнопок для отображения шагов алгоритма: «Next step» и «Prev step». А также кнопка «Reset», устанавливающая первоначальное поле для дальнейшего его изменения.
- Для оптимизации отображения работы алгоритма реализовать кнопку «Play», которая воспроизводит шаги алгоритма с установленной скоростью в выбранном направлении.
- Меню работы с файлом, где можно сохранить текущее состояние алгоритма, восстановить алгоритм из файла, создать новое поле заданного размера или очистить текущее.
- Информационное меню, содержащее информацию о проекте и описание рабочей области.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

№	Наименование работ	Срок выполнения
1.	Проектирование прототипа и реализация структуры данных снимок	06.07.2020
2.	Написание кода алгоритма, реализация функционала и интерфейса визуализатора, контроллера и модели	08.07.2020
3.	Исправление недочетов в работе алгоритма, расширение пользовательского интерфейса	10.07.2020
4.	Построение UML диаграмм используемых паттернов	10.07.2020
5.	Проведение тестирования программы	12.07.2020

2.2. Распределение ролей в бригаде

Ишанина Л.Н. – реализация алгоритма.

Ларин А. – визуализация алгоритма.

Сырцова Е.А. – архитектура программы.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Архитектура программы

- Архитектура данного приложения построена на основе шаблона проектирования MVC (Model View Controller) (см. рис. 1). Суть данного шаблона заключается в том, что любое действие пользователя обрабатывается в контроллере, который имеет доступ к модели. Контроллер, в свою очередь, изменяет состояние модели. View отображает текущее состояние модели. Таким образом, выделив три интерфейса (Visualisator, Controller, Model) можно разрабатывать данное приложение независимо. [2]

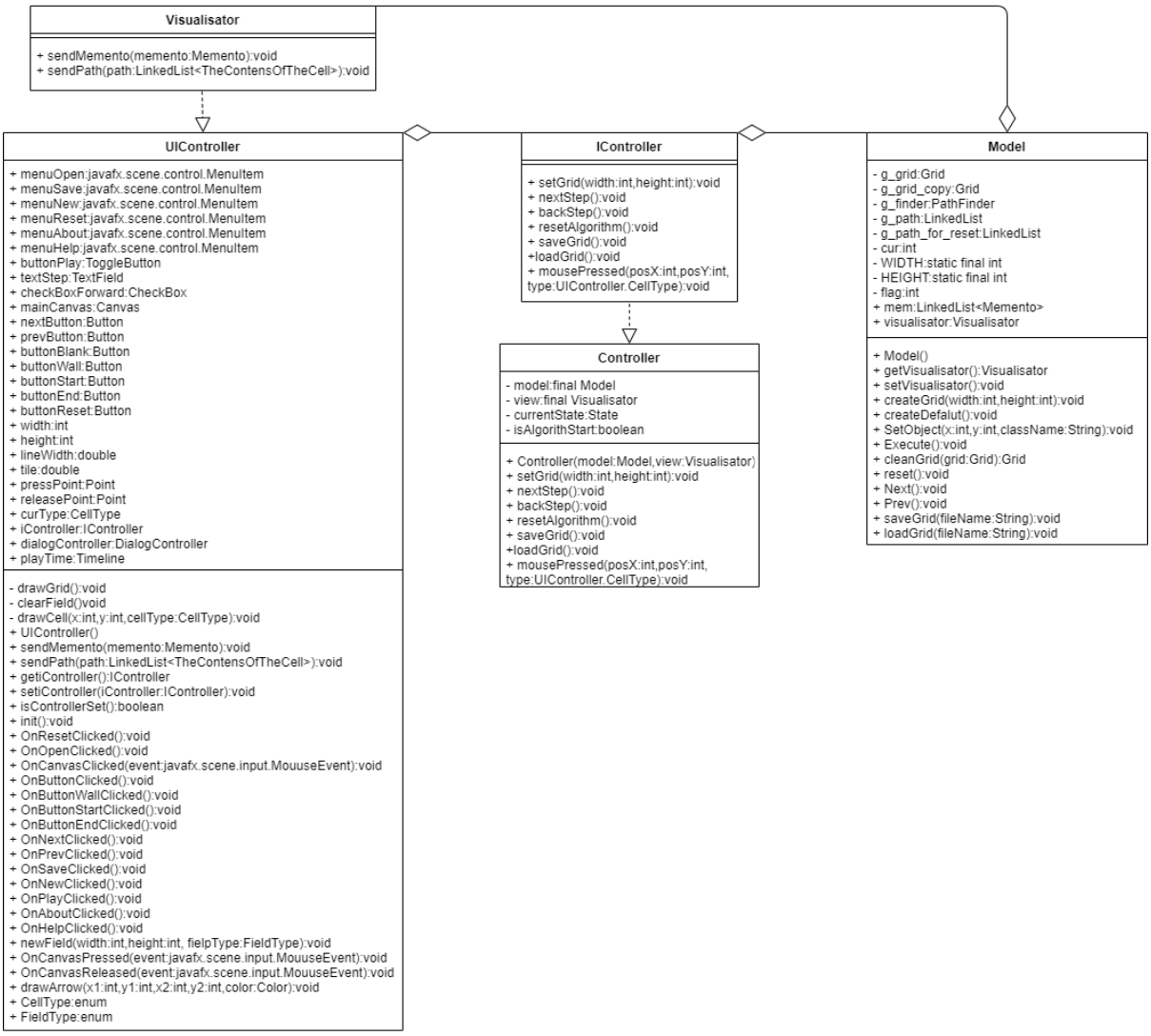

Рисунок 1 – UML диаграмма MVC данного приложения.

- Для того, чтобы пользователь мог добавлять объекты на поле, а также управлять работой алгоритма был применен шаблон проектирования State (Состояние) (см. рис. 2).

Пользователь имеет рычаги управления – выбрать объект нажатием на соответствующие кнопки и установить его на поле,

Чтобы не писать большие условные конструкции был применен шаблон проектирования State. Были выделены следующие состояния: AddingState, AlgorithmState. Контекстом для данных состояний стал контроллер, который имеет поле currentState и обрабатывает пользовательский ввод. Смена состояний происходит путем обработки нажатий на соответствующие кнопки.

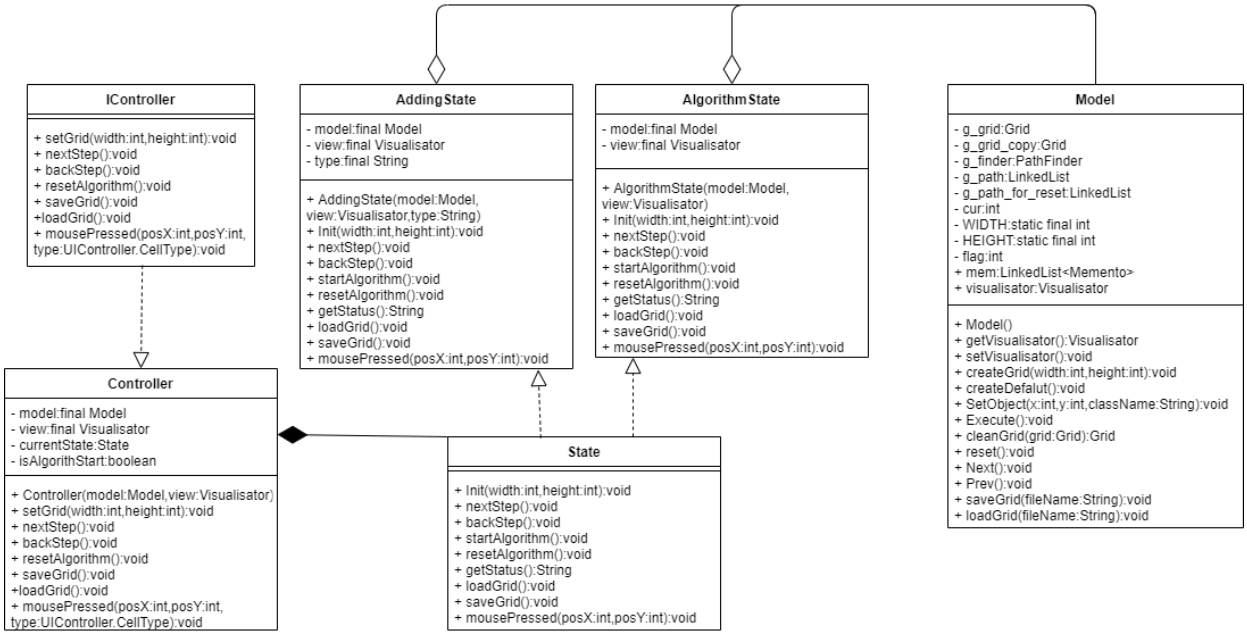

Рисунок 2 – UML диаграмма работы с использованием шаблона State.

- Следующая проблема заключалась в реализации пошаговой визуализации работы алгоритма. Решить эту задачу помогла идея шаблона проектирования Снимок. Был создан класс Memento, который хранит в себе размеры и текущее состояние поля (стартовая и конечная вершины, открытые и закрытые вершины, стены и путь). В определенные моменты работы алгоритма в контейнер сохраняются снимки текущего состояния. Затем, когда необходимо сделать шаг вперед просто отображается i+1 снимок (см. рис. 3). Также снимок используется для сохранения текущего состояния алгоритма в файл. [1]

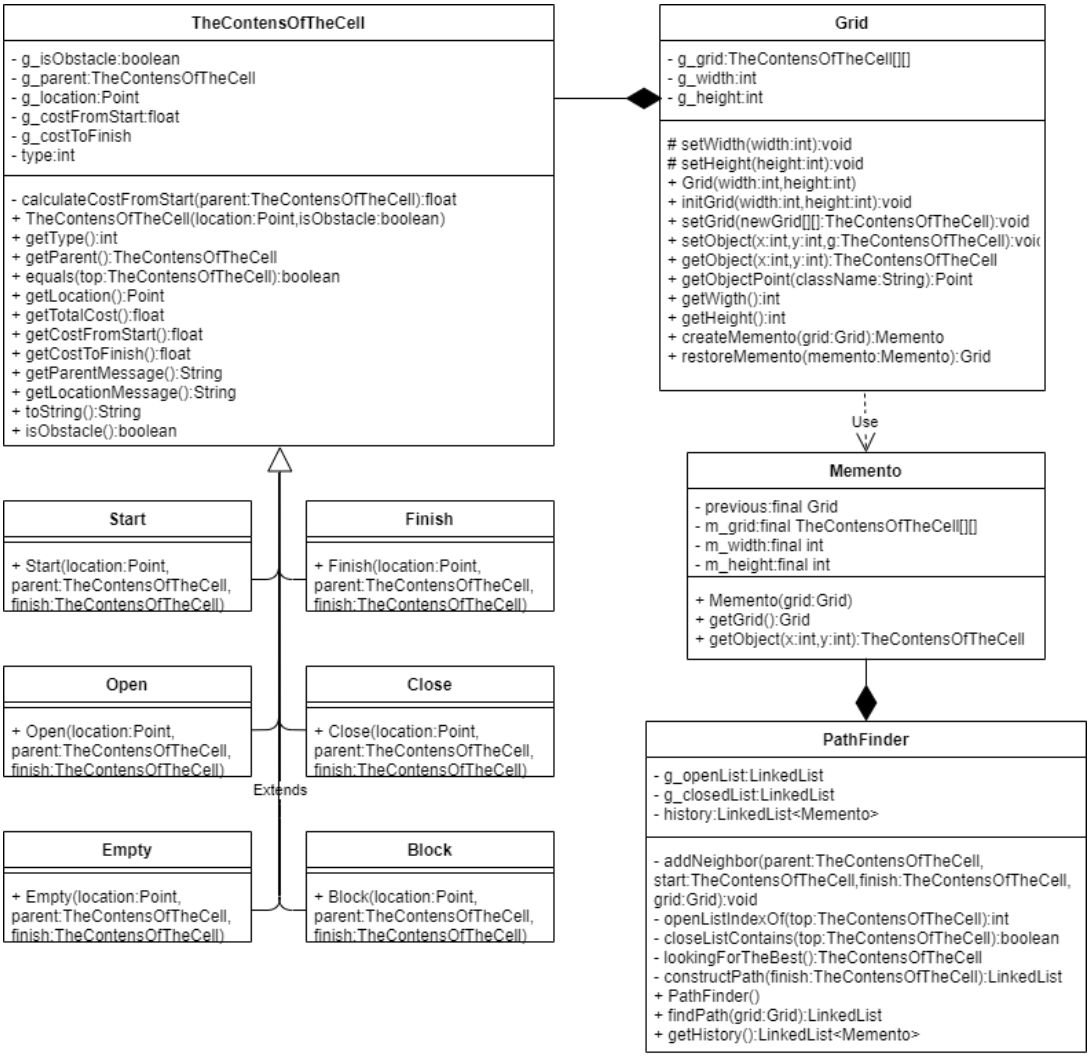

Рисунок 3 – UML диаграмма паттерна Memento

- Был продуман и реализован пользовательский интерфейс (см. рис. 4).

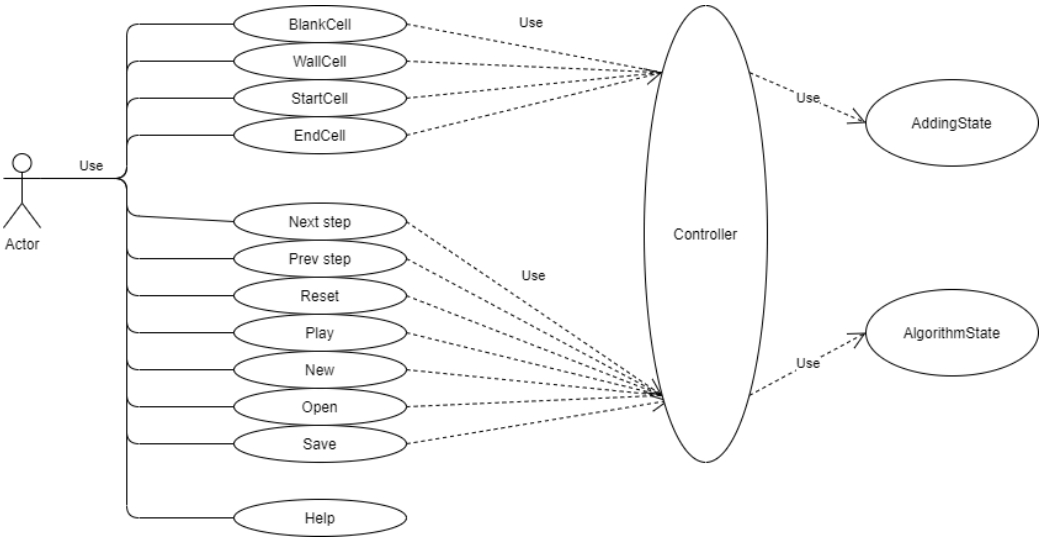

Рисунок 4 – use-case диаграмма программы.

3.2. Реализация визуализации

Для реализации графического интерфейса программы был выбран инструментарий JavaFX. С его помощью были написаны окна интерфейса на языке FXML семейства xml, а также реализован контроллер к ним на языке Java, осуществляющий работу с пользовательским вводом и компонентами окон. [4]

Всё взаимодействие пользователя с интерфейсом в контексте алгоритма не ведет к изменению интерфейса непосредственно. Все нажатия отсылаются контролеру, где обрабатываются, переадресуются модели, и наконец после этого модель по интерфейсу визуализатора посылает данные для отрисовки (см. рис. 5).

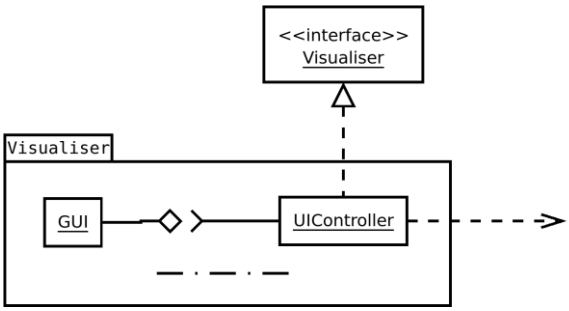

Рисунок 5 – Взаимодействие пользователя с интерфейсом

Передача управления из GUI в соответствующий контроллер происходит по событиям (Event), которые и обрабатываются контроллером.

Для отрисовки использован компонент Canvas, позволяющий рисовать растровые изображения с использованием примитивных и не очень фигур. Так поле, представляющее из себя сетку размера m*n, рисуется в виде квадратов разного цвета и одинакового размера, размещаемых по сетке. Поверх отрисованного клеточного поля наносятся стрелки, необходимые для демонстрации алгоритма (см. рис. 6).

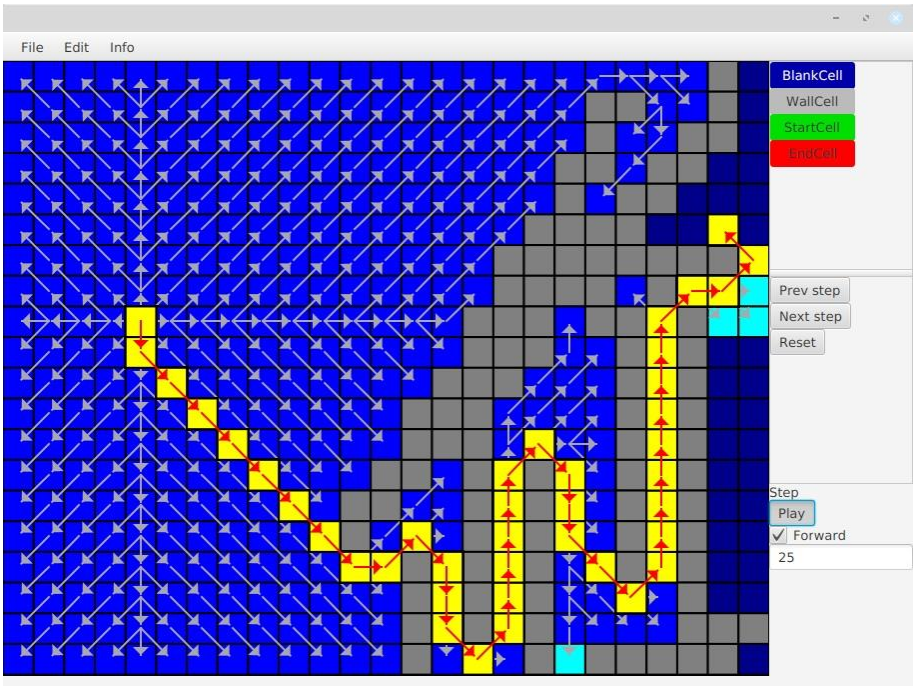

Рисунок 6 – Пример визуализации алгоритма

В область ответственности визуализатора так же входят вспомогательные окна (Help, About), диалоги (диалог создания поля), различные элементы управления, а также таймер, необходимый для функции автоматического исполнения алгоритма (анимация с фиксированным интервалом, вместо проигрывания каждого шага по нажатию). Для диалога, в силу присутствия собственных элементов управления, реализован свой контролер.

Так как визуализация непосредственно не отвечает за функционал программы, а лишь является триггером для его активации, тестирование корректности его работы в работе программы затруднительна из-за зависимости результата от других частей алгоритма. Потому тестирование таким способом происходит лишь после отладки всех частей по отдельности и, по сути, является интеграционным.

Тестирование же непосредственно визуализатора производилось преимущественно при помощи отладчика, где по точкам останова можно убедиться в посыле правильной команды по правильному действию пользователя, не заботясь на данном этапе о валидном исполнении этой

команды другими частями программы, а также убедится в правильном приеме команд на отрисовку поля, без поправки на правильность данного вида поля в контексте исполнения алгоритма.

3.3. Реализация контроллера

Интерфейс контроллера IController содержит методы для передачи действий пользователя методамам модели:

void setGrid(int width, int height) – задаёт размер поля;

void nextStep() – отображает следующий шаг работы алгоритма;

void backStep() – отображает предыдущий шаг работы алгоритма;

void resetAlgorithm() – сбрасывает работу алгоритма, устанавливает первоначальное поле;

void mousePressed(int posX, int posY, UIController.CellType type) – размещает выбранные объекты на поле;

void saveGrid() – сохраняет текущее состояние поля в файл;

void loadGrid() – загружает поле из файла.

Класс контроллера Controller наследуется от интерфейса и реализует его методы. Класс содержит приватные поля:

private final Model model – модель;

private final Visualisator view – визуализация;

private State currentState – текущее состояние;

private boolean isAlgorithmStart – метка начала работы алгоритма, нужна для ограничения размещения объектов на поле во время работы алгоритма и отображения шагов алгоритма, только если он начался.

Методы контролера устанавливают текущее состояние в зависимости от пользовательского ввода. Так метод mousePressed(int posX, int posY, UIController.CellType type) устанавливает состояние добавления объектов, а методы setGrid(int width, int height), nextStep и loadGrid() устанавливают состояние работы алгоритма. Методы backStep(), resetAlgorithm() и saveGrid()

не изменяют текущее состояние, но выполняются только при состоянии работы алгоритма.

Интерфейс состояний State:

void Init(int width, int height) – задаёт размеры поля;

void nextStep() – переходит к следующему шагу алгоритма;

void backStep() – переходит к предыдущему шагу алгоритма;

void startAlgorithm() – начинает работу алгоритма;

void resetAlgorithm() – сбрасывает работу алгоритма;

void loadGrid() – сохраняет граф в файл «save.txt»;

void saveGrid() – загружает граф из файла «save.txt»;

void mousePressed(int posX, int posY) – размещает объекты на поле.

Класс AddingState наследуется от интерфейса состояний, содержит приватные поля, устанавливаемые в конструкторе:

private final Model model – модель;

private final Visualisator view – визуализация;

private final String type – тип размещаемого объекта.

Метод класса mousePressed(int posX, int posY) вызывает метод модели SetObject(int x, int y, String className), размещающий объект на поле.

Класс AlgorithmState наследуется от интерфейса состояний, содержит приватные поля, устанавливаемые в конструкторе:

private final Model model – модель;

private final Visualisator view – визуализация.

Методы класса AlgorithmState вызывают соответствующие им методы модели: nextStep() вызывает Next(), метод backStep() - Prev(), метод Init(int width, int height) вызывает метод createGrid(int width, int height), метод startAlgorithm() вызывает Execute(), resetAlgorithm() - Reset(), а методы

saveGrid() и loadGrid() вызывают методы saveGrid(String fileName) и loadGrid(String fileName) соответственно.

3.4. Реализация модели

Описание класса Model

Класс Model состоит из следующих полей:

- Grid g_grid – объект класса поле, с которым и будет происходить работа;
- PathFinder g_finder – объект класса поиска пути. Используется в дальнейшем для вызова конструктора этого класса и последующим поиском пути;
- LinkedList g_path – список для записи в него найденного пути;
- LinkedList<Memento> mem – список снимков, необходим для предоставления работы алгоритма по шагам;
- int cur – индекс текущего снимка;
- static final int WIDTH и HEIGHT – переменные, которое задают поле стандартного размера.
- int flag_start – переменная-флаг, сообщает был ли уже установлен start или нет, используется для обеспечения только одной клетки start на поле;
- int flag_finish – переменная-флаг, аналогично сообщает был ли уже установлен finish или нет, используется для обеспечения только одной клетки finish на поле;
- int prohibition_flag – переменная-флаг, сообщает установлены ли и start, и finish на поле или нет, применяется для контролирования начала алгоритма, то есть алгоритм не будет начат, если нет какой-то из двух типов клеток;
- Visualisator visualisator – объект класса Visualisator, используется непосредственно для связанной работы с визуализатором.

Описание методов класса Model:

- Model() – конструктор, при его вызове будет автоматически вызван метод создания поля по умолчанию.
- Visualisator getVisualisator() – геттер, возвращающий объект класса Visualisator;
- void setVisualisator(Visualisator visualisator) – сеттер, устанавливает визуализатор в классе Model;
- void createDefault() – метод создания поля по умолчанию. Он вызывает конструктор класса Grid и тем самым создает поле высотой 20, а шириной 25 клеток. Также в этом методе устанавливаются начальные значения флагов (flag_start, flag_finish, prohibition_flag) и индекса снимков(cur).
- void createGrid(int width, int height) – метод создания поля по заданному размеру. Аналогично вызывается конструктор класса Grid и тем самым создает поле заданной высотой и шириной. И также в этом методе устанавливаются начальные значения флагов (flag_start, flag_finish, prohibition_flag) и индекса снимков(cur).
- void SetObject(int x, int y, String className) – метод вставки объекта на поле, принимает координаты ячейки поля и строку, обозначающую имя необходимого класса.
- void Execute() – метод, который сначала вызывает конструктор класса PathFinder, а затем запускает алгоритм, записывает найденный путь и все снимки работы алгоритма.
- boolean checkingTheStartAndFinish() – метод проверки нахождения на поле клеток start и finish. В соответствии возвращает либо true, либо false.
- void removeStart() – метод удаления «старой» ячейки start, вызывается из метода SetObject, в случае, когда пользователь пытается поставить несколько точек start. Таким образом предыдущая точка start будет удалена, а новая установлена в методе SetObject.

- void removeFinish() – аналогично работает метод удаления «старой» ячейки finish.
- Grid cleanGrid(Grid grid) – метод очистки поля от ячеек открытых вершин, закрытых и пути, то есть всех клеток, которые появлялись во время работы алгоритма. Используется в методе Reset, когда надо сбросить работу алгоритма.
- void Reset() – метод сброса работы алгоритма. Очищает поле, очищает снимки и найденный список пути. После вызова данного метода можно изменять поле (менять расположения start/finish, а также расставлять/убирать блоки).
- void Next() – метод для вывода на поле только одного следующего шага алгоритма. Данный метод отправляет следующий снимок визуализатору, который выводит его на поле. Это позволяет продемонстрировать пошаговую работу алгоритма вперед.
- void Prev() – метод для вывода на поле только одного предыдущего шага алгоритма. Данный метод отправляет предыдущий снимок визуализатору, который выводит его на поле. Это позволяет продемонстрировать пошаговую работу алгоритма не только вперед, но и назад.
- void saveGraph(String fileName) – метод для сохранения графа в файл fileName. Сохраняет в файл размеры поля и текущий шаг алгоритма для корректоного восстановления с файла, после сохраняет поле в виде символов (' ' – пустая клетка, '+' – открытая клетка, '-' – закрытая клетка, 's' – стартовая клетка, 'f' – конечная клетка, '#' – клетка со стеной) для удобного понимания (см. рис. 7).
- void loadGraph(String fileName) – функция для восстановления графа из файла fileName. Устанавливает текущее состояние модели согласно графу, сохраненному в файле.

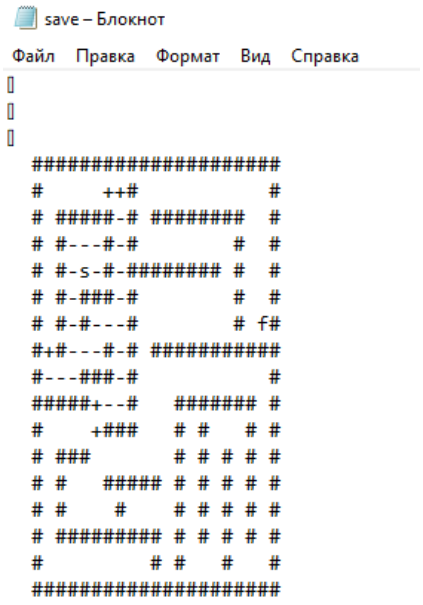

Рисунок 7 – Пример сохранения графа в файл

Описание класса Grid

Класс Grid состоит из следующих полей:

- TheContentsOfTheCell g_grid[][] – поле, состоящее из двумерного массива объектов TheContentsOfTheCell;
- int g_width – ширина поля;
- g_height – высота поля;

Описание методов класса Grid:

- Grid(int width, int height) – конструктор, вызывает метод создания и инициализации поля.
- void initGrid(int width, int height) – метод создания и инициализации поля.
- void setObject(int x, int y, TheContentsOfTheCell g) – метод замены объекта но поле, принимает координаты ячейки и новый объект, который будет помещен на поле.
- TheContentsOfTheCell getObject(int x, int y) – метод получения объекта из поля, принимает его местоположение и возвращает сам объект.
- void setWidth(int width) – метод присваивания ширине.
- void setHeight(int height) – метод присваивания высоты.

- int getWidth() – метод получения значения ширины поля.
- int getHeight() – метод получения значения высоты поля.
- Memento createMemento(Grid grid)
- Grid restoreMemento(Memento memento)

Описание класса PathFinder

Класс PathFinder состоит из следующих свойств:

- LinkedList g_openList – список открытых вершин;
- LinkedList g_closedList – список закрытых вершин;
- LinkedList<Memento> history – список снимков;

Описание методов класса PathFinder:

- PathFinder() – конструктор класса, создает и выделяет память для списков(g_openList, g_closedList, history).
- LinkedList findPath(Grid grid1) – метод поиска пути принимает на вход поле. Метод изначально находит и создает начальную и стартовую вершины поля, затем в цикле с помощью методов lookingForBestTop и addNeighbor находит оптимальный путь до финишной ячейки и вызывает метод построения пути.
- public LinkedList<Memento> getHistory() - метод, возвращающий список снимков поля.
- void addNeighbor(TheContentsOfTheCell parent, TheContentsOfTheCell start, TheContentsOfTheCell finish, Grid grid) – метод поиска соседних узлов текущей вершины принимает на вход родителя вершины(она же и есть текущая temp в методе findPath), финишную ячейку поля и само поле. Метод находит всех её соседей(включая диагональные), если она блок или закрытая, то пропускается, иначе она будет добавлена в список открытых вершин.

- int openListIndexOf(TheContentsOfTheCell top) – метод принимает на вход объект поля(иначе вершину). Данный метод возвращает индекс вершины из открытого списка, если её там нет, то возвращает -1.
- boolean closedListContains(TheContentsOfTheCell top) – метод принимает на вход объект поля(вершину). Данный метод возвращает true, если вершина есть в закрытом списке, если её там нет, то возвращает false.
- TheContentsOfTheCell lookingForBestTop() – метод выбора лучшей вершины из открытого списка. Лучшая вершина из доступных выбирается путем сравнения значений функций f(x) = g(x) + h(x), то есть функции, которая состоит из суммы уже пройденного пути и эвристической оценки.
- LinkedList constructPath(TheContentsOfTheCell finish) – метод построения пути, вызывается из метода findPath, когда была достигнута ячейка finish. Данный метод принимает финишную вершину, добавляет её в список path и переходит к её родителю. Таким образом и будет выстроен путь от страта до финиша.

Описание класса TheContentsOfTheCell

Класс содержимого ячейки поля - TheContensOfTheCell состоит из следующих свойств:

- boolean g_isObstacle – переменна, которая обозначает наличие препятствие в ячейке поля.
- TheContentsOfTheCell g_parent – родитель ячейки.
- Point g_location – расположение x и y ячейки.
- float g_costFromStart – значение функции g(x), которая считает расстояние от начальной вершины до текущей.
- float g_costToFinish – значение функции h(x), которая считает эвристическую оценку от текущей вершины до финальной.
- int type – тип объекта поля(Start, Finish, Empty, Block, Open, Close, Path).

Описание методов класса TheContensOfTheCell:

- TheContentsOfTheCell(Point location, boolean isObstacle, TheContentsOfTheCell parent, TheContentsOfTheCell finish, int type) – конструктор класса содержимого ячейки. Принимает на вход, расположение объекта, является ли она блоком, её родителя, ячейку финиша и её тип. Присваивает её все значения и считает расстояние от начальной вершины до неё и эвристическую оценку до финиша.
- int getType() – функция для получения типа объекта.
- void getType(int newType) – функция установки нового типа для вершины, принимает на вход новый номер типа.
- float calculateCostFromStart(TheContentsOfTheCell parent) – метод, вычисляющий расстояние от начальной вершины до текущей. Считает по евклидову расстоянию.
- TheContentsOfTheCell getParent() – метод для получения родителя вершины, возвращает её родителя.
- boolean equals(TheContentsOfTheCell top) – метод для сравнения объектов поля.
- Point getLocation() – метод для получения расположения объекта на поле.
- float getTotalCost() – метод, который складывает расстояние от начальной вершины до текущей и эвристическую оценку до финиша.
- float getCostFromStart() – метод для получения расстояния от начальной вершины до текущей.
- float getCostToFinish() – метод для получения эвристической оценки до финиша.
- boolean isObstacle() – метод, который возвращает true или false в зависимости от наличия блока на ячейке.

Для удобства были созданы 7 классов, наследуемых от TheContensOfTheCell, чтобы каждый раз не прописывать большое количество аргументов при создании объекта TheContensOfTheCell:

- Класс Empty, конструктор которого Empty(Point location, TheContentsOfTheCell parent, TheContentsOfTheCell finish) – принимает всего 3 аргумента: расположения объекта, родителя и конечную ячейку. Его конструктор вызывает конструктор супер-класса, добавляя параметры в вызов его конструктора. Таким образом, создается пустой объект или же пустая ячейка на поле.
- Класс Open, конструктор которого Open(Point location, TheContentsOfTheCell parent, TheContentsOfTheCell finish) – принимает 3 аргумента: расположения объекта, родителя и конечную ячейку. Его конструктор вызывает конструктор супер-класса, добавляя параметры в вызов его конструктора. Таким образом, создается объект, олицетворяющий открытую вершину на поле.
- Класс Close, конструктор которого Close(Point location, TheContentsOfTheCell parent, TheContentsOfTheCell finish) – принимает 3 аргумента: расположения объекта, родителя и конечную ячейку. Его конструктор вызывает конструктор супер-класса, добавляя параметры в вызов его конструктора. Таким образом, создается объект, олицетворяющий закрытую вершину на поле.
- Класс Block, конструктор которого Block(Point location, TheContentsOfTheCell parent, TheContentsOfTheCell finish) – принимает 3 аргумента: расположения объекта, родителя и конечную ячейку. Его конструктор вызывает конструктор супер-класса, добавляя параметры в вызов его конструктора. Таким образом, создается объект-блок, на который нельзя “наступать” на поле.
- Класс Start, конструктор которого Start(Point location, TheContentsOfTheCell parent, TheContentsOfTheCell finish) – принимает 3 аргумента: расположения объекта, родителя и конечную ячейку. Его

конструктор вызывает конструктор супер-класса, добавляя параметры в вызов его конструктора. Таким образом, создается стартовая вершина на поле.

- Класс Finish, конструктор которого Finish(Point location, TheContentsOfTheCell parent, TheContentsOfTheCell finish) – принимает 3 аргумента: расположения объекта, родителя и конечную ячейку. Его конструктор вызывает конструктор супер-класса, добавляя параметры в вызов его конструктора. Таким образом, создается финальная вершина на поле.
- Класс Path, конструктор которого Path(Point location, TheContentsOfTheCell parent, TheContentsOfTheCell finish) – принимает 3 аргумента: расположения объекта, родителя и конечную ячейку. Его конструктор вызывает конструктор супер-класса, добавляя параметры в вызов его конструктора. Таким образом, создается объект на поле, который является частью пути, построенного алгоритмом.

Тестирование

В ходе написания вышеперечисленных классов, постоянно проводилось ручное тестирование, которое помогло выявить уязвимые места в коде и исправить их.

1. Тест. Если пользователь не введет на поле стартовую или финишную вершину, либо и ту, и другую (пустое поле), то при нажатии любой из кнопок интерфейса не будет ничего происходить, программа будет ждать, когда на поле будут обе ячейки. Демонстрация представлена на рис. 8.

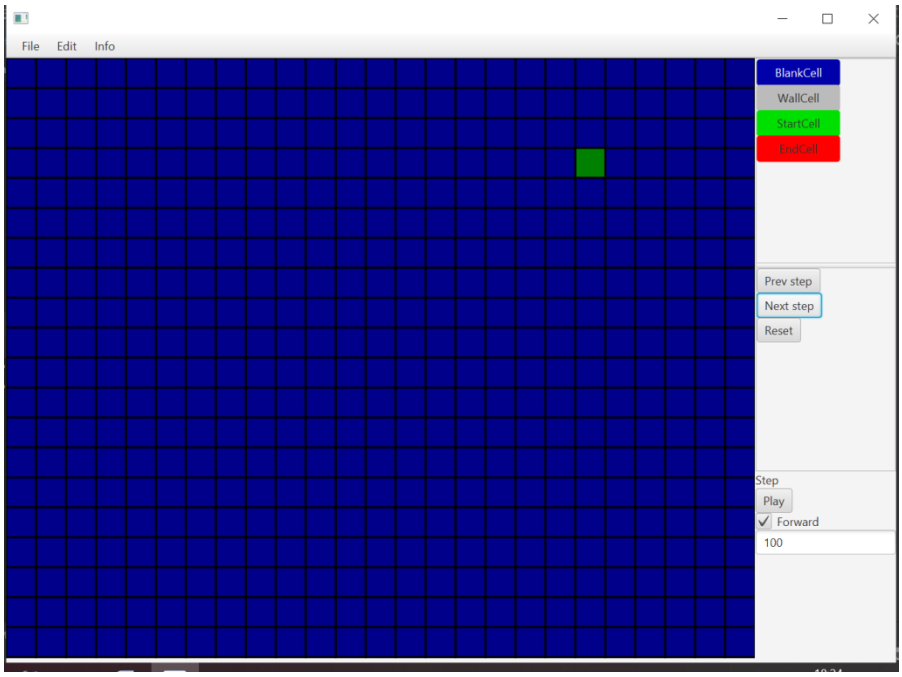

Рисунок 8 – Демонстрация работы программы в случае отсутствия начальной/финальной вершины на поле

2. Тест. Также в ходе работы была отлажена передача снимков визуализатору. Программа никак не будет реагировать на ситуацию, когда был выведен на экран финальный снимок (показан весь алгоритм), а пользователь снова будет нажимать кнопку Next step, то есть попытается вызвать несуществующий снимок. В таком случае программа не будет никак реагировать и не сломается. Аналогично работает и с ситуацией, когда показан на экране нулевой снимок, а пользователь захочет сделать откат назад, то есть опять попытается выйти за граница списка снимков.

   Демонстрация работы программы представлена на рис. 9.

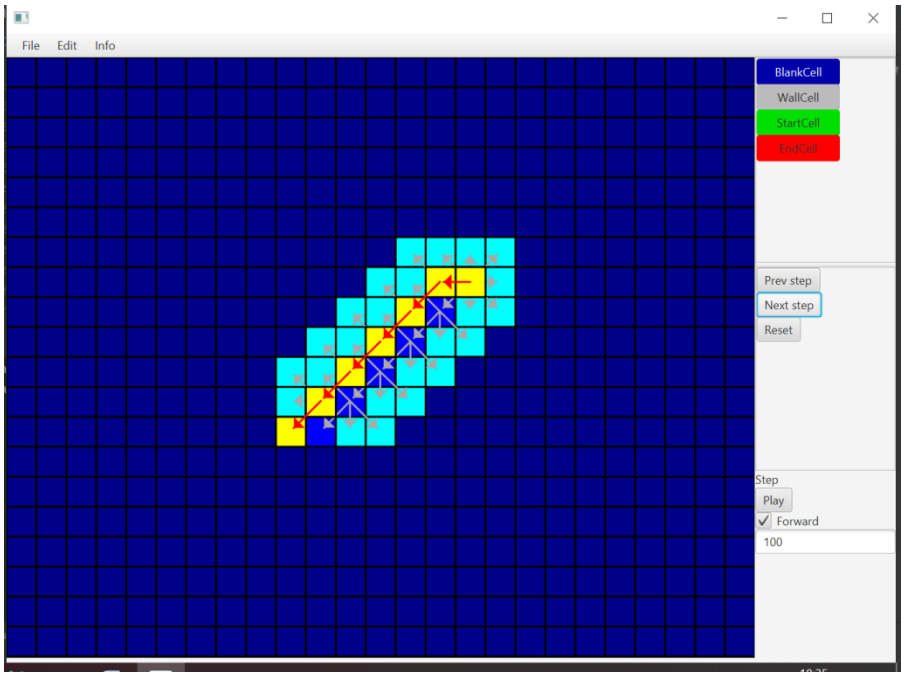

Рисунок 9 – Демонстрация работы программы в случае многократного нажатия кнопки NextStep, после вывода последнего снимка на экран

3. Тест. В случае, когда нет пути, изначально программа выдавала ошибки, но затем была изменена так, что если путь невозможно найти, то программа просто будет выводить по-прежнему снимки поля, до момента пока алгоритм не встанет в тупик. Демонстрация представлена на рис. 10.

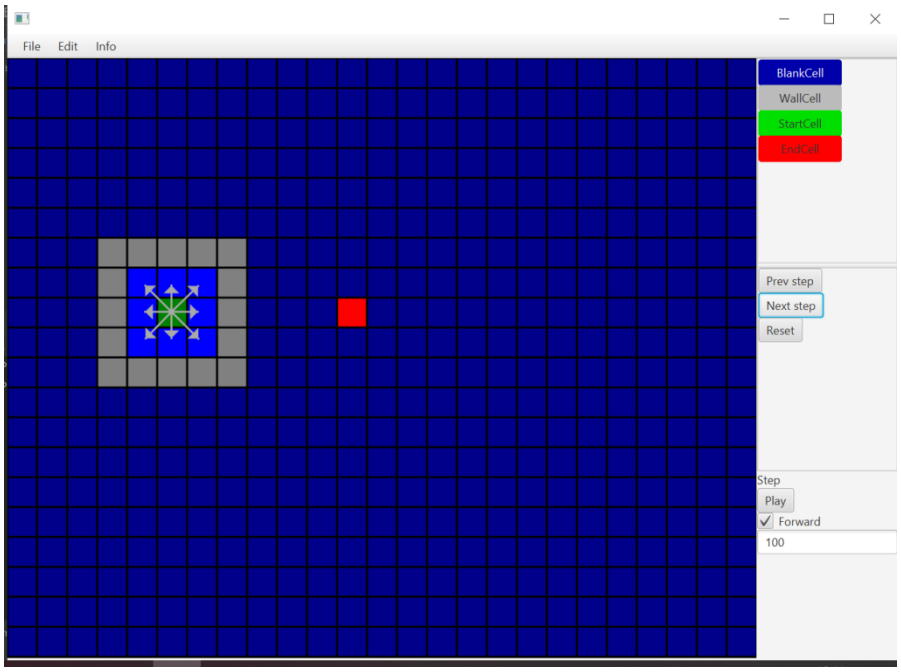

Рисунок 10 – Демонстрация работы программы в случае невозможного построения пути

4. Тест. Также были тесты, проверяющие корректность работы алгоритма, действительно ли найден самый оптимальный путь. Демонстрация представлена на рис. 11 и 12.

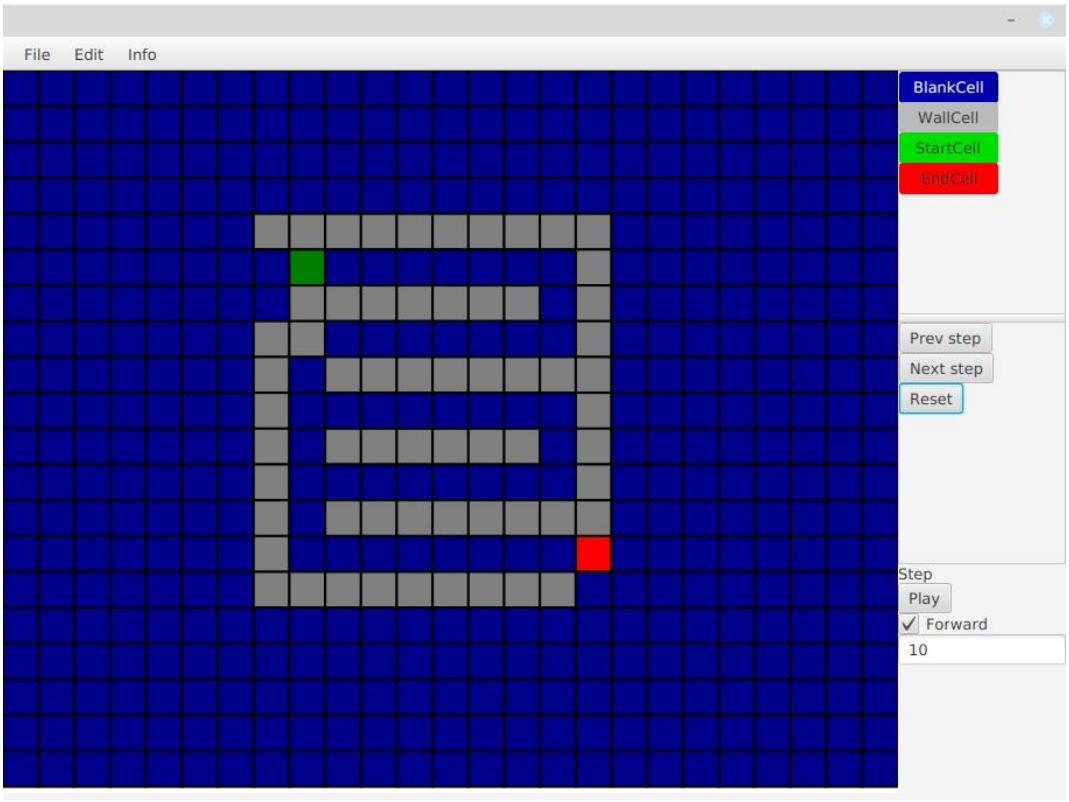

Рисунок 11 – Начальное заданное поле

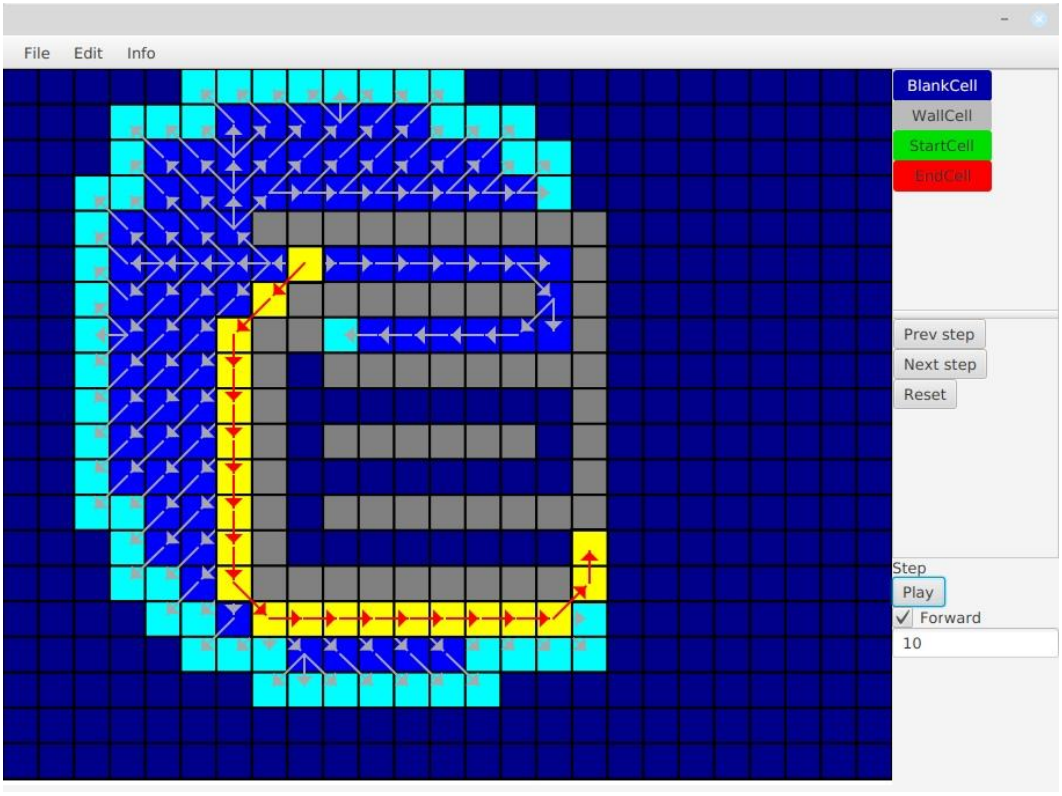

Рисунок 12 – Корректная работа алгоритма

Таким образом, путь действительно найден кратчайший, алгоритм работает корректно.

Дополнительные результаты тестирования программы представленны на рис. 13-14.

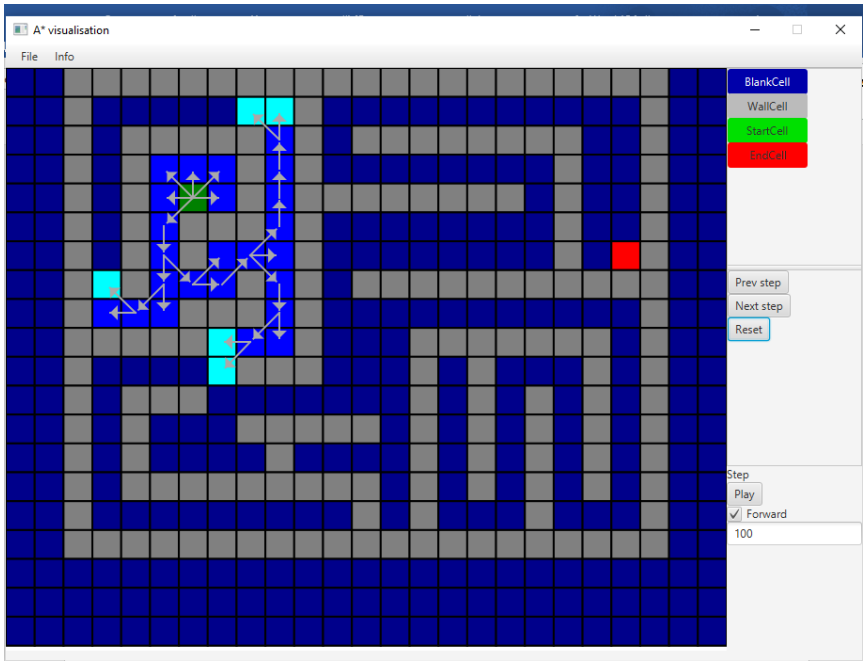

Рисунок 13 – Пример загрузки графа из файла

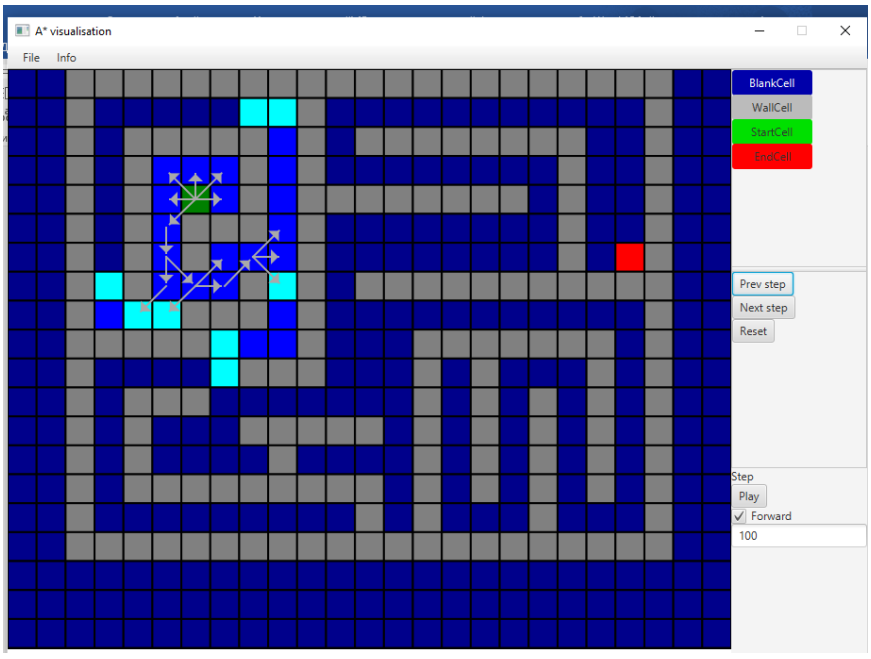

Рисунок 14 – Пример отображения предыдущих шагов из загруженного графа

ЗАКЛЮЧЕНИЕ

В процессе выполнения задания учебной практики были изучены основы программирования на языке Java. Для изучения данного языка был пройден интерактивный курс «Java.Базовый курс» на платформе Stepik. После чего была разработана программа, которая находит кратчайшие пути в графе с помощью алгоритма A*. Разработанная программа соответствует всем заявленным требованиям спецификации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Р.Седжвик, К. Уэйн книга Алгоритмы на Java. Изд-во ”Вильямс”, 2013.– 846 с.

2. Java Базовый курс. [Электронный ресурс]

URL: stepik.org/course/Java-Базовый-курс-187/syllabus (дата обращения 3.07.2020).

3. Википедия. [Электронный ресурс]

URL: wikipedia.org/wiki/A* (дата обращения 3.07.2020).

4. Учебник по JavaFX. [Электронный ресурс]

URL: https://habr.com/ru/post/474292/ (дата обращения 6.07.2020).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
package groupid1;

import Controller.Controller;
import Model.Model;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;


/**
 * JavaFX App
 */
public class App extends Application {

    private static UIController uiController;
    private static Model model;
    private static Controller controller;


    @Override
    public void start(Stage stage) throws IOException {
        uiController = new UIController();
        FXMLLoader loader;
        URL xmlUrl;
        Parent root;

        loader = new FXMLLoader();

        xmlUrl = getClass().getResource("/UI.FXML");
        System.out.println(getClass().getResource("/UI.FXML"));
        System.out.println(xmlUrl);
        loader.setLocation(xmlUrl);
        loader.setController(uiController);
        root = loader.load();
        stage.setScene(new Scene(root));

        model = new Model();
```

```
        controller = new Controller(model,uiController);
        uiController.setiController(controller);
        uiController.init();
        model.setVisualisator(uiController);
        stage.setResizable(false);
        stage.setTitle("A* visualisation");

        InputStream iconStream;
        Image image;
/*
        iconStream =
getClass().getResourceAsStream("/Astern512.png");
        image = new Image(iconStream);
        stage.getIcons().add(image);
        iconStream =
getClass().getResourceAsStream("/Astern256.png");
        image = new Image(iconStream);
        stage.getIcons().add(image);
        iconStream =
getClass().getResourceAsStream("/Astern128.png");
        image = new Image(iconStream);
        stage.getIcons().add(image);
        iconStream =
getClass().getResourceAsStream("/Astern64.png");
        image = new Image(iconStream);
        stage.getIcons().add(image);
*/

        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }

}


package groupid1;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
```

```
import javafx.scene.control.Button;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

import java.io.IOException;
import java.net.URL;

public class DialogController {
    public UIController uiController;
    @FXML
    public TextField textWidth;
    @FXML
    public TextField textHeight;
    @FXML
    public RadioButton radioBlank;
    @FXML
    public RadioButton radioFilled;
    @FXML
    public RadioButton radioRandom;
    @FXML
    public RadioButton radioLabyrinth;
    @FXML
    public Button buttonOk;

    public DialogController(UIController uiController1) {
        uiController = uiController1;
        Parent root;
        URL xmlUrl;
        Stage stage = new Stage();
        FXMLLoader loader = new FXMLLoader();

        try {
            xmlUrl = getClass().getResource("/NewDialog.fxml");
            loader.setLocation(xmlUrl);
            loader.setController(this);
            root = loader.load();
            stage.setScene(new Scene(root));
            stage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    @FXML
```

```
    public void OnOkClicked(ActionEvent event) {
        int width =
Integer.parseInt(textWidth.getCharacters().toString());
        int height =
Integer.parseInt(textHeight.getCharacters().toString());
        UIController.FieldType fieldType;
        if (radioBlank.isSelected()) {
            fieldType = UIController.FieldType.BLANK;
        } else if (radioFilled.isSelected()) {
            fieldType = UIController.FieldType.FILLED;
        } else if (radioLabyrinth.isSelected()) {
            fieldType = UIController.FieldType.LABYRINTH;
        } else if (radioRandom.isSelected()) {
            fieldType = UIController.FieldType.RANDOM;
        } else {
            fieldType = UIController.FieldType.BLANK;
        }
        uiController.newField(width, height, fieldType);
        // Hide this current window (if this is what you want)
        ((Node)
(event.getSource())).getScene().getWindow().hide();
    }

}
```

```
package groupid1;

public class SystemInfo {

    public static String javaVersion() {
        return System.getProperty("java.version");
    }

    public static String javafxVersion() {
        return System.getProperty("javafx.version");
    }

}
```

```
package groupid1;

import Controller.IController;
import Model.Grid;
import Model.Memento;
```

```
import Model.States.Visualisator;
import Model.TheContentsOfTheCell;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleButton;
import javafx.scene.paint.Color;
import javafx.scene.transform.Affine;
import javafx.scene.transform.Transform;
import javafx.stage.Stage;
import javafx.util.Duration;

import java.awt.*;
import java.io.IOException;
import java.net.URL;
import java.util.HashSet;
import java.util.LinkedList;


public class UIController implements Visualisator {

    //--------GUI elements
    @FXML
    public javafx.scene.control.MenuItem menuOpen;
    @FXML
    public javafx.scene.control.MenuItem menuSave;
    @FXML
    public javafx.scene.control.MenuItem menuNew;
    @FXML
    public javafx.scene.control.MenuItem menuReset;
    @FXML
    public javafx.scene.control.MenuItem menuAbout;
    @FXML
    public javafx.scene.control.MenuItem menuHelp;
    @FXML
    public ToggleButton buttonPlay;
```

```
    @FXML
    public TextField textStep;
    @FXML
    public CheckBox checkBoxForward;
    @FXML
    public Canvas mainCanvas;
    @FXML
    public Button nextButton;
    @FXML
    public Button prevButton;
    @FXML
    public Button buttonBlank;
    @FXML
    public Button buttonWall;
    @FXML
    public Button buttonStart;
    @FXML
    public Button buttonEnd;
    @FXML
    public Button buttonReset;
    //--------Package-private fields
    int width;
    int height;
    double lineWidth;
    double tile;
    Point pressPoint;
    Point releasePoint;
    CellType curType;
    IController iController;

    DialogController dialogController;
    Timeline playTimer;

    public UIController() {
        width = 10;
        height = 10;
        tile = 0;
    }

    //--------Interface implementation
    @Override
    public void sendPath(LinkedList<TheContentsOfTheCell> path) {
        for (TheContentsOfTheCell cell : path) {
            drawCell((int) cell.getLocation().getX(), (int)
cell.getLocation().getY(), CellType.PATH);
```

```
            TheContentsOfTheCell parent = cell.getParent();
            if (parent != null) {
                Point loc = cell.getLocation();
                Point ploc = parent.getLocation();
                drawArrow((int) ((ploc.getX() + 1) * tile - tile /
2), (int) ((ploc.getY() + 1) * tile - tile / 2), (int)
((loc.getX() + 1) * tile - tile / 2), (int) ((loc.getY() + 1) *
tile - tile / 2), Color.RED);
            }
        }
    }

    @Override
    public void sendMemento(Memento memento) {
        assert (memento.getGrid().getHeight() == height);
        assert (memento.getGrid().getWidth() == width);
        Grid grid = memento.getGrid();
        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                TheContentsOfTheCell cell = memento.getObject(i,
j);
                if (cell.isObstacle()) {
                    drawCell((int) i, (int) j, CellType.WALL);
                } else if (cell.getType() == 6) {
                    drawCell((int) i, (int) j, CellType.CLOSE);
                } else if (cell.getType() == 3) {
                    drawCell((int) i, (int) j, CellType.START);
                } else if (cell.getType() == 4) {
                    drawCell((int) i, (int) j, CellType.END);
                } else if (cell.getType() == 5) {
                    drawCell((int) i, (int) j, CellType.OPEN);
                } else {
                    drawCell((int) i, (int) j, CellType.BLANK);
                }
            }
        }

        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                TheContentsOfTheCell cell = memento.getObject(i,
j);
                if (cell.getParent() != null) {
                    TheContentsOfTheCell parent =
cell.getParent();
                    Point loc = cell.getLocation();
                    Point ploc = parent.getLocation();
```

```
                    drawArrow((int) ((ploc.getX() + 1) * tile -
tile / 2), (int) ((ploc.getY() + 1) * tile - tile / 2), (int) ((i
+ 1) * tile - tile / 2), (int) ((j + 1) * tile - tile / 2),
Color.DARKGRAY);
                }
            }
        }
    }

//--------Package-used methods

    public IController getiController() {
        return iController;
    }

    public void setiController(IController iController) {
        this.iController = iController;
    }

    boolean isControllerSet() {
        if (iController == null) {
            System.err.println("Visualiser: Controller Not Set!");
            return false;
        }
        return true;
    }

    public void init() {
        curType = CellType.WALL;
        width = 25;
        height = 20;
        lineWidth = 2;
        tile = Math.min(mainCanvas.getWidth() / width,
mainCanvas.getHeight() / height);
        drawGrid();

        if (isControllerSet()) {
            iController.setGrid(width, height);
        }
    }


    //--------Event handlers
    @FXML
    public void OnResetClicked() {
```

```
        if (isControllerSet()) {
            clearField();
            iController.resetAlgorithm();
        }
    }

    @FXML
    public void OnOpenClicked() {
        if (isControllerSet()) iController.loadGrid();
    }

    @FXML
    public void OnCanvasClicked(javafx.scene.input.MouseEvent
event) {
        if (isControllerSet())
            iController.mousePressed((int) (event.getX() / tile),
(int) (event.getY() / tile), curType);
    }

    @FXML
    public void OnButtonBlankClicked() {
        curType = CellType.BLANK;
    }

    @FXML
    public void OnButtonWallClicked() {
        curType = CellType.WALL;
    }

    @FXML
    public void OnButtonStartClicked() {
        curType = CellType.START;
    }

    @FXML
    public void OnButtonEndClicked() {
        curType = CellType.END;
    }

    @FXML
    public void OnNextClicked() {
        if (isControllerSet()) {
            iController.nextStep();
        }
    }
```

```
    @FXML
    public void OnPrevClicked() {
        if (isControllerSet()) iController.backStep();
    }

    @FXML
    public void OnSaveClicked() {
        if (isControllerSet()) iController.saveGrid();
    }

    @FXML
    public void OnNewClicked() {
        dialogController = new DialogController(this);
    }

    @FXML
    public void OnPlayClicked() {
        if (buttonPlay.isSelected()) {
            try {
                playTimer.stop();
            } catch (Exception e) {
            }
            playTimer = new Timeline(new
KeyFrame(Duration.millis(Integer.parseInt(textStep.getCharacters()
.toString())), new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    if (checkBoxForward.isSelected()) {
                        OnNextClicked();
                    } else {
                        OnPrevClicked();
                    }
                }
            }));
            playTimer.setCycleCount(Timeline.INDEFINITE);
            playTimer.play();
        } else {
            try {
                playTimer.stop();
            } catch (Exception e) {
            }
        }
    }

    @FXML
    public void OnAboutClicked() {
```

```
        Parent root;
        URL xmlUrl;
        Stage stage = new Stage();
        FXMLLoader loader = new FXMLLoader();
        try {
            xmlUrl = getClass().getResource("/About.fxml");
            loader.setLocation(xmlUrl);
            root = loader.load();
            stage.setScene(new Scene(root));
            stage.setTitle("About");
            stage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @FXML
    public void OnHelpClicked() {
        Parent root;
        URL xmlUrl;
        Stage stage = new Stage();
        FXMLLoader loader = new FXMLLoader();
        try {
            xmlUrl = getClass().getResource("/Help.fxml");
            loader.setLocation(xmlUrl);
            root = loader.load();
            stage.setScene(new Scene(root));
            stage.setTitle("Help");
            stage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void newField(int width, int height, FieldType
fieldType) {
        clearField();
        this.width = width;
        this.height = height;
        tile = Math.min(mainCanvas.getWidth() / width,
mainCanvas.getHeight() / height);
        drawGrid();

        if (isControllerSet()) {
            iController.setGrid(width, height);
        }
```

```
    }

    @FXML
    public void OnCanvasPressed(javafx.scene.input.MouseEvent
event) {
        if (curType != CellType.END && curType != CellType.START)
            pressPoint = new Point((int) event.getX(), (int)
event.getY());
    }

    @FXML
    public void OnCanvasReleased(javafx.scene.input.MouseEvent
event) {
        if (pressPoint != null) {
            releasePoint = new Point((int) event.getX(), (int)
event.getY());
            /*
            pressPoint=new
Point((int)(pressPoint.x),(int)(pressPoint.y));
            releasePoint=new
Point((int)(releasePoint.x),(int)(releasePoint.y));
            */

            int steps = (int) (mainCanvas.getHeight() +
mainCanvas.getWidth());
            double r = 1;
            HashSet<Point> s = new HashSet<Point>();

            for (int i = 0; i <= steps; i++) {
                double ratio = ((double) i) / steps;
                double dx = releasePoint.getX() -
pressPoint.getX();
                double dy = releasePoint.getY() -
pressPoint.getY();

                Point cur = new Point((int) pressPoint.getX(),
(int) pressPoint.getY());
                cur = new Point((int) (cur.getX() + (dx * ratio)),
(int) (cur.getY() + (dy * ratio)));
                Point pt = new Point((int) (cur.getX() / tile),
(int) (cur.getY() / tile));
                if (!s.contains(pt)) {
                    s.add(pt);
                    iController.mousePressed((int) pt.getX(),
(int) pt.getY(), curType);
                }
```

```
                //iController.mousePressed((int) (cur.getX() /
tile), (int) (cur.getY() / tile), curType);

                /*
                for(int xx=(int)(cur.getX()-
r);xx<=cur.getX()+r;xx++){
                    for(int yy=(int)(cur.getY()-
r);yy<=cur.getY()+r;yy++){

                    }
                }

                 */
            }


            pressPoint = releasePoint = null;
        }

    }

    private void drawGrid() {
        GraphicsContext gc = mainCanvas.getGraphicsContext2D();
        gc.setLineWidth(lineWidth);
        gc.stroke();
        gc.setStroke(Color.BLACK);

        for (int i = 0; i < width + 1; i++) {
            gc.beginPath();
            gc.moveTo(i * tile, 0);
            gc.lineTo(i * tile, height * tile);
            gc.stroke();

        }
        for (int i = 0; i < height + 1; i++) {
            gc.beginPath();
            gc.moveTo(0, i * tile);
            gc.lineTo(width * tile, i * tile);
            gc.stroke();
        }
    }

    private void clearField() {
        GraphicsContext gc = mainCanvas.getGraphicsContext2D();
        gc.setLineWidth(0);
        gc.setFill(Color.WHITE);
```

```
        gc.fillRect(0, 0, mainCanvas.getWidth(),
mainCanvas.getHeight());

    }

    private void drawArrow(int x1, int y1, int x2, int y2, Color
color) {

        GraphicsContext gc = mainCanvas.getGraphicsContext2D();
        var tr = gc.getTransform();
        int ARR_SIZE = 8;
        gc.setFill(color);
        gc.setStroke(color);

        double dx = x2 - x1, dy = y2 - y1;
        double angle = Math.atan2(dy, dx);
        int len = (int) (Math.sqrt(dx * dx + dy * dy) * 0.90);

        Transform transform = Transform.translate(x1, y1);
        transform =
transform.createConcatenation(Transform.rotate(Math.toDegrees(angl
e), 0, 0));
        gc.setTransform(new Affine(transform));

        gc.strokeLine(0, 0, len, 0);
        gc.fillPolygon(new double[]{len, len - ARR_SIZE, len -
ARR_SIZE, len}, new double[]{0, -ARR_SIZE, ARR_SIZE, 0},
                4);
        gc.setTransform(tr);
    }

    private void drawCell(int x, int y, CellType cellType) {
        GraphicsContext gc = mainCanvas.getGraphicsContext2D();
        gc.setLineWidth(lineWidth);
        gc.setStroke(Color.BLACK);
        switch (cellType) {
            case BLANK:
                gc.setFill(Color.DARKBLUE);
                break;
            case WALL:
                gc.setFill(Color.GRAY);
                break;
            case START:
                gc.setFill(Color.GREEN);
                break;
            case END:
```

```
                gc.setFill(Color.RED);
                break;
            case OPEN:
                gc.setFill(Color.CYAN);
                break;
            case CLOSE:
                gc.setFill(Color.BLUE);
                break;
            case PATH:
                gc.setFill(Color.YELLOW);
                break;
        }
        gc.fillRect(x * tile, y * tile, tile, tile);
        gc.strokeRect(x * tile, y * tile, tile, tile);
    }

    public enum CellType {
        BLANK,
        WALL,
        START,
        END,
        OPEN,
        CLOSE,
        PATH
    }

    public enum FieldType {
        BLANK,
        FILLED,
        RANDOM,
        LABYRINTH
    }
}


package Model.States;

import Model.Memento;
import Model.TheContentsOfTheCell;

import java.util.LinkedList;

public interface Visualisator {
    void sendMemento(Memento memento);
    void sendPath(LinkedList<TheContentsOfTheCell> path);
}
```

```
package Controller;

import groupid1.UIController;

/**
 * Публичный интерфейс контроллера(Выступает контекстом для
состояний)
 * void setGrid(int width, int height) - задать поле
 * void nextStep() - следующий шаг алгоримта
 * void backStep() - предыдущий шаг алгоритма
 * void resetAlgorithm();
 * void mousePressed(int posX, int posY, UIController.CellType
type);
 * void saveGrid();
 * void loadGrid();
 */

public interface IController {
    void setGrid(int width, int height);
    void nextStep();
    void backStep();
    void resetAlgorithm();
    void mousePressed(int posX, int posY, UIController.CellType
type);
    void saveGrid();
    void loadGrid();
}


package Controller;
import Model.Model;
import Model.States.*;
import groupid1.UIController;
import java.io.File;

public class Controller implements IController{
    private final Model model;
    private final Visualisator view;
    private State currentState;
    private boolean isAlgorithmStart = false;

    public Controller(Model model,Visualisator view) {
        this.model = model;
        this.view = view;
```

```
    }

    @Override
    public void setGrid(int width, int height){
        if (!isAlgorithmStart){
            currentState = new AlgorithmState(model,view);
            currentState.Init(width,height);
        }
    }

    @Override
    public void nextStep(){
        if (!isAlgorithmStart){
            currentState = new AlgorithmState(model,view);
            currentState.startAlgorithm();
        }
        isAlgorithmStart = true;
        currentState.nextStep();
    }

    @Override
    public void backStep() {
        if (isAlgorithmStart)
            currentState.backStep();
    }

    @Override
    public void resetAlgorithm() {
        if (isAlgorithmStart) {
            isAlgorithmStart = false;
            currentState.resetAlgorithm();
        }
    }

    @Override
    public void mousePressed(int posX, int posY,
UIController.CellType type){
        if (!isAlgorithmStart){
            switch (type){
                case END:
                    currentState = new
AddingState(model,view,"Finish");
                    break;
                case WALL:
                    currentState = new
AddingState(model,view,"Block");
```

```
                    break;
                case BLANK:
                    currentState = new
AddingState(model,view,"Empty");
                    break;
                case START:
                    currentState = new
AddingState(model,view,"Start");
                    break;
            }
            currentState.mousePressed(posX,posY);
        }
    }

    @Override
    public void saveGrid() {
        if (isAlgorithmStart)
            currentState.saveGrid();
    }

    @Override
    public void loadGrid(){
        File f = new File("save.txt");
        if (f.canRead() && f.length()!=0) {
            currentState = new AlgorithmState(model,view);
            currentState.loadGrid();
            isAlgorithmStart = true;
        }
    }
}


package Model.States;

import java.io.FileNotFoundException;

/**
 * Интерфейс состояний
 *  void Init(int width, int height)
 *  void nextStep() - следующий шаг алгоритма
 *  void backStep() - предыдущий шаг алгоритма
 *  void startAlgorithm()
 *  void resetAlgorithm()
 *  void loadGrid();
 *  void saveGrid();
 *  void mousePressed(int posX, int posY) - реакция на нажатие
```

```
кнопки мыши
 */

public interface State {
    void Init(int width, int height);
    void nextStep();
    void backStep();
    void startAlgorithm();
    void resetAlgorithm();
    void loadGrid();
    void saveGrid() ;
    void mousePressed(int posX, int posY);
}


package Model.States;

import Model.Model;

public class AddingState implements State{
    private final Model model;
    private final Visualisator view;
    private final String type;

    public AddingState(Model model, Visualisator view, String
type) {
        this.model = model;
        this.view = view;
        this.type = type;
    }

    @Override
    public void Init(int width, int height){}

    @Override
    public void nextStep() {}

    @Override
    public void backStep(){}

    @Override
    public void startAlgorithm(){}

    @Override
    public void resetAlgorithm(){}
```

```
    @Override
    public void saveGrid(){ model.saveGrid("save.txt"); }

    @Override
    public void loadGrid() { model.saveGrid("save.txt"); }

    @Override
    public void mousePressed(int posX, int posY) {
        model.SetObject(posX,posY,type);
    }
}


package Model.States;

import Model.*;

public class AlgorithmState implements State{
    private final Model model;
    private final Visualisator view;

    public AlgorithmState(Model model, Visualisator view) {
        this.model = model;
        this.view = view;
    }

    @Override
    public void nextStep() {
        model.Next();
    }

    @Override
    public void backStep() {
        model.Prev();
    }

    @Override
    public void Init(int width, int height){
        model.createGrid(width, height);
    }

    @Override
    public void startAlgorithm(){
        try {
            model.Execute();
        } catch (ClassNotFoundException e) {
```

```
            e.printStackTrace();
        }
    }

    @Override
    public void resetAlgorithm() {
        model.Reset();
    }

    @Override
    public void saveGrid(){
        model.saveGrid("save.txt");
    }

    @Override
    public void loadGrid() {
        model.loadGrid("save.txt");
    }

    @Override
    public void mousePressed(int posX, int posY) { }
}


package Model;

public class Memento {
    private final Grid previous;
    private final TheContentsOfTheCell m_grid[][];
    private final int m_width;//ширина
    private final int m_height;//высота
    //private LinkedList open;

    public Memento(Grid grid){//конструктор
        this.previous = grid;
        this.m_width = grid.getWidth();
        this.m_height = grid.getHeight();
        this.m_grid = new
TheContentsOfTheCell[grid.getHeight()][grid.getWidth()];
        for(int y = 0; y < grid.getHeight(); y++)
            for(int x = 0; x < grid.getWidth(); x++)
                this.m_grid[y][x] = grid.getObject(x,y);
    }

    public Grid getGrid(){
        return this.previous;
```

```
    }

    public TheContentsOfTheCell getObject(int x, int y){
        return this.m_grid[y][x];
    }
}


package Model;

/*
 * это абстрактный класс, в котором 4 класса:
 * Пустая клетка, клетка-блок, начало, финиш,
 *  то есть каждая из них будет иметь свой цвет
 *
 * */

/*
* ТИПЫ:
* 1 - empty
* 2 - block
* 3 - start
* 4 - finish
* 5 - open
* 6 - close
* ????????7 - path
* */

import java.awt.*;

//----------------------------- Класс содержимого клетки ---------
--------------------
public class TheContentsOfTheCell {
    private boolean g_isObstacle;//есть препятствие или нет
    private TheContentsOfTheCell g_parent;
    private Point g_location;//её расположение на сетке( X и Y )
    private float g_costFromStart;//функция(g) - расстояние от
начальной вершины
    private float g_costToFinish;//функция(h) - оценка расстояния
до финиша
    private int type;

    public TheContentsOfTheCell(Point location, boolean
isObstacle, TheContentsOfTheCell parent, TheContentsOfTheCell
finish, int type){
        //this.setObstacle(location, isObstacle, parent, finish);
```

```
        this.g_isObstacle = isObstacle;
        this.g_parent = parent;
        this.g_location = location;
        this.type = type;
        float x;
        float y;
        if(parent != null){//если это НЕ стратовая вершина, то
            //вычисляем расстояние от начальной вершины до текщей
            this.g_costFromStart =
this.calculateCostFromStart(parent);//устанавливаем функцию g(v)
        }
        if(finish != null){//если это НЕ стратовая вершина, то
            //вычисляем функцию оценки до финиша!
            x = location.x - finish.getLocation().x;
            y = location.y - finish.getLocation().y;
            this.g_costToFinish = (float) Math.sqrt(Math.pow(x, 2)
+ Math.pow(y, 2));
            //эвристическая оценка выбрана как евклидово
расстояние
            //может в дальнейшем изменить..?
        }

    }

    /*public void setObstacle(Point location, boolean isObstacle,
TheContentsOfTheCell parent, TheContentsOfTheCell finish){
        this.g_isObstacle = isObstacle;
        this.g_location = location;
        this.g_parent = parent;
    }

    public void set(TheContentsOfTheCell parent,
TheContentsOfTheCell finish) {
        float x;
        float y;
        if(parent != null){//если это НЕ стратовая вершина, то
            //вычисляем расстояние от начальной вершины до текщей
            this.g_costFromStart =
this.calculateCostFromStart(parent);//устанавливаем функцию g(v)
        }
        if(finish != null){//если это НЕ стратовая вершина, то
            //вычисляем функцию оценки до финиша!
            x = this.g_location.x - finish.g_location.x;
            y = this.g_location.y - finish.g_location.x;
            this.g_costToFinish = (float) Math.sqrt(Math.pow(x, 2)
+ Math.pow(y, 2));
```

```
            //эвристическая оценка выбрана как евклидово
расстояние
            //может в дальнейшем изменить..?
        }
        this.g_parent = parent;
    }*/

    public int getType(){
        return this.type;
    }

    public void getType(int newType){
        this.type = newType;
    }

    private float calculateCostFromStart(TheContentsOfTheCell
parent){
        float x = this.g_location.x - parent.g_location.x;
        float y = this.g_location.y - parent.g_location.y;
        float sum = (float) Math.sqrt(Math.pow(x, 2) + Math.pow(y,
2));
        //if(this.getParent()!=null) {
        sum += this.getParent().getCostFromStart();
        //}
        return sum;
    }

    public TheContentsOfTheCell getParent(){
        return this.g_parent;
    }

    public boolean equals(TheContentsOfTheCell top){
        return this.g_location.equals(top.getLocation());
    }

    public Point getLocation(){
        return this.g_location;
    }

    public float getTotalCost(){
        return this.g_costFromStart + this.g_costToFinish;
    }

    public float getCostFromStart(){
        return this.g_costFromStart;
    }
```

```
    public float getCostToFinish(){
        return this.g_costToFinish;
    }

    public String getParentMessage(){
        if(this.g_parent == null){
            return null;
        }
        return "[" + this.g_parent.getLocation().x + "," +
this.g_parent.getLocation().y + "]";
    }

    public String getLocationMessage(){
        return "[" + this.g_location.x + "," + this.g_location.y +
"]";
    }

    public String toString(){
        return "[" + this.g_location.x + "," + this.g_location.y +
"]" + "\tparent: " + this.g_parent;
    }

    public boolean isObstacle(){
        return this.g_isObstacle;
    }

    public static class Empty extends TheContentsOfTheCell {

        public Empty(Point location, TheContentsOfTheCell parent,
TheContentsOfTheCell finish){
            super(location, false, parent, finish, 1);//вызывается
конструктор родительского класса
            //МЫ ГОВОРИМ ЧТО КЛЕТКА ПУСТАЯ
        }

    }

    public static class Open extends TheContentsOfTheCell {
        public Open(Point location, TheContentsOfTheCell parent,
TheContentsOfTheCell finish){
            super(location,false, parent, finish, 5);

        }
    }
```

```
    public static class Close extends TheContentsOfTheCell {

        public Close(Point location, TheContentsOfTheCell parent,
TheContentsOfTheCell finish){
            super(location,false, parent, finish, 6);//вызывается
конструктор родительского класса
            //МЫ ГОВОРИМ ЧТО КЛЕТКА close
        }

    }

    public static class Block extends TheContentsOfTheCell {

        public Block(Point location, TheContentsOfTheCell parent,
TheContentsOfTheCell finish){
            super(location,true, parent, finish, 2);

            // МЫ ГОВОРИМ ЧТО КЛЕТКА !НЕ! ПУСТАЯ - блок
        }

    }

    public static class Start extends TheContentsOfTheCell {

        public Start(Point location, TheContentsOfTheCell parent,
TheContentsOfTheCell finish){
            super(location,false, parent, finish, 3);

        }
    }

    public static class Finish extends TheContentsOfTheCell {

        public Finish(Point location, TheContentsOfTheCell parent,
TheContentsOfTheCell finish){
            super(location,false, parent, finish, 4);

        }
    }

    public static class Path extends TheContentsOfTheCell {

        public Path(Point location, TheContentsOfTheCell parent,
TheContentsOfTheCell finish){
            super(location,false, parent, finish, 7);
```

```
        }
    }

}


package Model;

import java.awt.Point;
//----------------------------- Класс сетки ----------------------
-------
public class Grid {
    private TheContentsOfTheCell g_grid[][];//сама сетка, сост. из
объектов
    private int g_width;//ширина
    private int g_height;//высота

    public Grid(int width, int height){
        this.initGrid(width, height);
    }

    public void initGrid(int width, int height){
        this.setWidth(width);
        this.setHeight(height);
        this.g_grid = new
TheContentsOfTheCell[this.getHeight()][this.getWidth()];
        //создали сетку и ниже инициализируем каждую клетку как
пустую
        for(int y = 0; y < this.getHeight(); y++){
            for(int x = 0; x < this.getWidth(); x++){
                this.g_grid[y][x] = new
TheContentsOfTheCell.Empty(new Point(x,y),null, null);
            }
        }
    }



    public void setGrid(TheContentsOfTheCell[][] newGrid){

        this.g_grid = newGrid;
        this.setWidth(newGrid.length);
        this.setHeight(newGrid.length);
    }

    public void setObject(int x, int y, TheContentsOfTheCell
```

```
g){//меняем значение клетки с одного объекта на другой
        this.g_grid[y][x] = g;
    }


    public TheContentsOfTheCell getObject(int x, int y){//ПОЛУЧИТЬ
объект клетки
        return this.g_grid[y][x];
    }

    public Point getObjectPoint(String className) throws
ClassNotFoundException{
        //Этот метод принимает параметр className,
        //который является классом, для которого требуется его
экземпляр
        Point p = null;
        for(int y = 0; y < this.getHeight(); y++){
            for(int x = 0; x < this.getWidth(); x++){
                //оператор isInstance позволяет проверять,
                //является ли объект g_grid[y][x] экземпляром
класса, на который ссылается className

if(Class.forName(className).isInstance(this.getObject(x, y))){
                    return p = new Point(x, y);
                }
            }
        }
        return p;
    }
    /*
    то есть пояснение к вышеизложенному методу
    он используется в PathFinder
    Например, создается точка start, то есть в метод передается
имя класса Start,
    и далее начинается проход по всему полю
    и если объект клетки равен классу Start, то бинго! мы создаем
точку старта Point
    * */

    protected void setWidth(int width){
        this.g_width = width;
    }

    protected void setHeight(int height){
        this.g_height = height;
    }
```

```
    public int getWidth(){
        return this.g_width;
    }

    public int getHeight(){
        return this.g_height;
    }

    // Снимок — неизменяемый объект, поэтому "Создатель"=сетка
передаёт
    // все своё состояние через параметры конструктора.
    public Memento createMemento(Grid grid) {
        return new Memento(grid);
    }


    // В нужный момент владелец снимка может восстановить
состояние сетки
    public Grid restoreMemento(Memento memento){
        return memento.getGrid();
    }
}


package Model;

import java.awt.Point;
import java.util.LinkedList;

//----------------------------- Класс поиска пути ----------------
-------------
public class PathFinder {
    //в классе поиска пути хранятся список открытых и закрытх
вершин!
    private LinkedList g_openList;
    private LinkedList g_closedList;
    private LinkedList<Memento> history;

    public PathFinder(){
        this.g_openList = new LinkedList();
        this.g_closedList = new LinkedList();
        this.history = new LinkedList();
    }

    public LinkedList findPath(Grid grid1) throws
```

```
NullPointerException, ClassNotFoundException{
        Grid grid = grid1;


if(!this.g_openList.isEmpty())this.g_openList.clear();//очищаем
списки
        if(!this.g_closedList.isEmpty())this.g_closedList.clear();
        if(!this.history.isEmpty())history.clear();

        //небольшое уточнение
        //TheContentsOfTheCell.Start.class.getName() -
        //возвращает имя класса, представленной этим объектом
класса, в виде строки.
        //далее с помощью метода getObjectPoint создастся точка
strat a затем и finish
        Point start = new
Point(grid.getObjectPoint(TheContentsOfTheCell.Start.class.getName
()));
        Point finish = new
Point(grid.getObjectPoint(TheContentsOfTheCell.Finish.class.getNam
e()));
        TheContentsOfTheCell finishTop = new
TheContentsOfTheCell.Finish(finish, null, null);
        TheContentsOfTheCell startTop = new
TheContentsOfTheCell.Start(start, null, null);
        TheContentsOfTheCell temp = new
TheContentsOfTheCell.Open(start, null, finishTop);


        this.g_openList.add(temp);//добавляем в список откр вершин
начальную вершину
        while(!this.g_openList.isEmpty()){//пока список откр
вершин не пуст...
            if(temp.getLocation().equals(finish)){//если
расположение текущей точки = finish
                return this.constructPath(temp, grid);//то строим
путь!!!
            }
            //иначе
            temp = this.lookingForBestTop();//текущая вершина =
лучшей из доступный
            ////////////////////
            if(!temp.getLocation().equals(start)) {
                TheContentsOfTheCell close = new
TheContentsOfTheCell.Close(temp.getLocation(), temp.getParent(),
null);
```

```
                this.g_closedList.addLast(temp);//добавляем её в
закрытый список
                grid.setObject(temp.getLocation().x,
temp.getLocation().y, close);
                grid.setObject(start.x, start.y, startTop);
                grid.setObject(finish.x, finish.y, finishTop);
                history.add(new Memento(grid));
            }
            else if (temp.getLocation().equals(start)) {
                history.add(new Memento(grid));
                //grid.setObject(start.x, start.y, startTop);
            }
            ////////////////////
            this.addNeighbor(temp, startTop, finishTop, grid);
            //вызываем метод добавляения соседней вершины
        }

        return null;
    }

    public LinkedList<Memento> getHistory(){
        return history;
    }

    private void addNeighbor(TheContentsOfTheCell parent,
TheContentsOfTheCell start, TheContentsOfTheCell finish, Grid
grid){
        int x = parent.getLocation().x;//получаем координаты x и y
родителя
        int y = parent.getLocation().y;
        //выполняем проход:
        // -----------
        //| 1 | 2 | 3 |
        // -----------
        //| 4 | T | 5 |
        // -----------
        //| 6 | 7 | 8 |
        // -----------
        for(int newY = y - 1; newY < y + 2; newY++){
            for(int newX = x - 1; newX < x + 2; newX++){
                try{
                    if(newY == y && newX == x){
                        //пропускаем самого себя
                    }
                    else if(!grid.getObject(newX,
newY).isObstacle()){//если в ячейке нет БЛОКА
```

```
                        TheContentsOfTheCell top = new
TheContentsOfTheCell.Open(new Point(newX, newY),parent,
finish);//создаем новую вершину,
                        //top.set(parent, finish);
                        // которую мы рассматриваем

                            int index =
this.openListIndexOf(top);//получаем её индекс в открытом списке
                            if (this.closedListContains(top) ) {
                                //если она в закрытом списке, то
пропускаем
                            } else if (index != -1) {//если она
есть в открытом списке, то
                                TheContentsOfTheCell old =
(TheContentsOfTheCell) this.g_openList.get(index);//создаем
вершину, точнее достаем её из открытого списка
                                //если предыдущий родитель
находится дальше от старта, чем новый, то...
                                if
(old.getParent().getCostFromStart() >
top.getParent().getCostFromStart()) {
                                    //old.setType()
                                    this.g_openList.set(index,
top);//меняем старое значение на новое
                                }
                            }
                            else {

                                TheContentsOfTheCell open =
new TheContentsOfTheCell.Open(top.getLocation(), parent, finish);

grid.setObject(top.getLocation().x, top.getLocation().y, open);

this.g_openList.add(top);//если вершины не было, то добавляем в
откр список её


                            }

                    }
                }
                catch(ArrayIndexOutOfBoundsException
ae){//учитываем ошибку:
                    //когда мы пытаемся обратиться к элементу
массива
                    //по отрицательному или превышающему размер
```

```
массива индексу
                    //System.err.println(ae);
                }
            }
        }
    }

    private int openListIndexOf(TheContentsOfTheCell
top){//возвращает индекс из открытого списка вершин
        for(int index = 0; index < this.g_openList.size();
index++){
            TheContentsOfTheCell anotherTop =
(TheContentsOfTheCell) this.g_openList.get(index);
            if(anotherTop.equals(top)){
                return index;
            }
        }
        return -1;
    }

    private boolean closedListContains(TheContentsOfTheCell
top){//проверка находится ли вершина в закрытом списке!!!
        for(int index = 0; index < this.g_closedList.size();
index++){//проход по закрытым вершинам
            TheContentsOfTheCell anotherTop =
(TheContentsOfTheCell) this.g_closedList.get(index);
            if(anotherTop.equals(top)){
                return true;
            }
        }
        return false;
    }

    private TheContentsOfTheCell lookingForBestTop(){//поиск
лучшего узла сетки
        if(this.g_openList.isEmpty()){//проверка что список
открытых вершин не пуст
            return null;
        }
        int lowestCostIndex = 0;// индекс наименьшей стоимости
        float cost =
((TheContentsOfTheCell)this.g_openList.get(0)).getTotalCost();//из
начально оценка =...
        for(int index = 1; index < this.g_openList.size();
index++){//проходимся по вершинам из открытого списка
            TheContentsOfTheCell top = (TheContentsOfTheCell)
```

```
this.g_openList.get(index);
            if(top.getTotalCost() < cost){//если оценка из вершины
меньше чем была, то

                cost = top.getTotalCost();//изменяем оценку на
наименьшую(она же наилучшая)
                lowestCostIndex = index;//меняем индекс
            }
        }
        TheContentsOfTheCell top = (TheContentsOfTheCell)
this.g_openList.remove(lowestCostIndex);
        //после прохода:
        // удаляем вершину которая оказалась лучшей из списка
открытых вершин
        return top;
    }

    private LinkedList constructPath(TheContentsOfTheCell finish,
Grid grid){//метод построения пути!
        LinkedList path = new LinkedList();//создаем новый список
        while(finish != null){
            //TheContentsOfTheCell pa = new
TheContentsOfTheCell.Path(finish.getLocation(),finish.getParent(),
finish);
            //history.add(new Memento(grid));
            path.addFirst(finish);//добавляем в путь вершину
            finish = finish.getParent();//и получаем её родителя
        }
        return path;
    }

}



package Model;

import java.awt.*;
import java.io.*;
import java.util.LinkedList;
import Model.States.Visualisator;

public class Model {
    private Grid g_grid;
    private Grid g_grid_copy;
    private PathFinder g_finder;
```

```
private LinkedList g_path;
private LinkedList g_path_for_reset;
public LinkedList<Memento> mem; //это будет массив снимков
private int cur;//индекс снимка
private static final int WIDTH = 25;
private static final int HEIGHT = 20;
private int flag_start;
private int flag_finish;
private int counter_reset;
private int prohibition_flag;

public Visualisator getVisualisator() {
    return visualisator;
}

public void setVisualisator(Visualisator visualisator) {
    this.visualisator = visualisator;
}

public Visualisator visualisator;

public Model() {
    createDefault();
}


public void createDefault() {//создается поле по умолчанию
    this.g_grid = new Grid(Model.WIDTH, Model.HEIGHT);
    this.cur = 0;
    this.g_grid_copy = new Grid(Model.WIDTH, Model.HEIGHT);
    flag_start = 0;
    flag_finish = 0;
    counter_reset = 0;
    prohibition_flag = 0;

}

public void createGrid(int width, int height){
    this.g_grid = new Grid(width, height);
    this.cur = 0;
    flag_start = 0;
    flag_finish = 0;
    prohibition_flag = 0;
}
```

```
    public void SetObject(int x, int y, String className) {

        Point location = new Point(x,y);

        if(className.equals("Start")){
            if(flag_start == 0) {
                TheContentsOfTheCell start = new
TheContentsOfTheCell.Start(location, null, null);
                this.g_grid.setObject(x, y, start);
                flag_start = 1;
            }
            else if(flag_start == 1){
                removeStart();
                TheContentsOfTheCell start = new
TheContentsOfTheCell.Start(location, null, null);
                this.g_grid.setObject(x, y, start);
            }
        }
        else if(className.equals("Finish")){
            if(flag_finish == 0) {
                TheContentsOfTheCell finish = new
TheContentsOfTheCell.Finish(location, null, null);
                this.g_grid.setObject(x, y, finish);
                flag_finish = 1;
            }
            else if(flag_finish == 1)
            {
                removeFinish();
                TheContentsOfTheCell finish = new
TheContentsOfTheCell.Finish(location, null, null);
                this.g_grid.setObject(x, y, finish);
            }
        }
        else if(className.equals("Block")){
            TheContentsOfTheCell block = new
TheContentsOfTheCell.Block(location, null, null);
            this.g_grid.setObject(x,y, block);
        }
        else {
            TheContentsOfTheCell empty = new
TheContentsOfTheCell.Empty(location, null, null);
            this.g_grid.setObject(x,y, empty);
        }


        visualisator.sendMemento(new Memento(g_grid));
```

```
        //this.flag = 0;
        //g_grid_copy = g_grid;

    }

    public void Execute() throws ClassNotFoundException {

        if(checkingTheStartAndFinish()) {
            this.g_finder = new PathFinder();//вызвали конструктор
            this.g_path =
this.g_finder.findPath(this.g_grid);//вызвали метод и нашли путь
            this.mem = this.g_finder.getHistory();//получаем
mementos
            //visualisator.sendPath(this.g_path);
            //this.mem.add(new Memento(this.g_grid));
        }


    }

    public boolean checkingTheStartAndFinish(){

        for(int y = 0; y < g_grid.getHeight(); y++){
            for(int x = 0; x < g_grid.getWidth(); x++){
                if(g_grid.getObject(x,y).getType() == 3 ) {
                    prohibition_flag = 1;
                }
                if(g_grid.getObject(x,y).getType() == 4){
                    prohibition_flag = 2;
                }

            }
        }
        if(prohibition_flag == 2){
            return true;
        }
        else
            return false;
    }

    public void removeStart(){
        for(int y = 0; y < g_grid.getHeight(); y++){
            for(int x = 0; x < g_grid.getWidth(); x++){
                if(g_grid.getObject(x,y).getType() == 3 ) {
                    TheContentsOfTheCell empty = new
TheContentsOfTheCell.Empty(new Point(x,y),null, null);
```

```
                    g_grid.setObject(x,y, empty);
                }

            }
        }
    }

    public void removeFinish(){
        for(int y = 0; y < g_grid.getHeight(); y++){
            for(int x = 0; x < g_grid.getWidth(); x++){
                if(g_grid.getObject(x,y).getType() == 4 ) {
                    TheContentsOfTheCell empty = new
TheContentsOfTheCell.Empty(new Point(x,y),null, null);
                    g_grid.setObject(x,y, empty);
                }

            }
        }
    }

    public Grid cleanGrid(Grid grid){
        for(int y = 0; y < grid.getHeight(); y++){
            for(int x = 0; x < grid.getWidth(); x++){
                if(grid.getObject(x,y).getType() == 5 ) {
                    TheContentsOfTheCell empty = new
TheContentsOfTheCell.Empty(new Point(x,y),null, null);
                    grid.setObject(x,y, empty);
                }
                if(grid.getObject(x,y).getType() == 6 ) {
                    TheContentsOfTheCell empty = new
TheContentsOfTheCell.Empty(new Point(x,y),null, null);
                    grid.setObject(x,y, empty);
                }
                if(grid.getObject(x,y).getType() == 7 ) {
                    TheContentsOfTheCell empty = new
TheContentsOfTheCell.Empty(new Point(x,y),null, null);
                    grid.setObject(x,y, empty);
                }
            }
        }
        return grid;
    }

    public void Reset(){

        if(prohibition_flag == 2) {
```

```
            cur = 0;
            this.g_grid = cleanGrid(this.g_grid);

            visualisator.sendMemento(mem.get(0));

            this.mem.clear();
            if(g_path!=null) {
                g_path.clear();
                visualisator.sendPath(g_path);
            }
        }

        //потом: возвращает поле изначальное Антону
    }

    public void Next() {

        if(prohibition_flag == 2) {

            if (this.cur < this.mem.size() - 1) {
                this.cur++;
            }
            if ((this.cur == this.mem.size() - 1) & this.cur != 0)
{

                if(g_path != null)
                    visualisator.sendPath(this.g_path);

            } else if (this.cur < this.mem.size() - 1) {
                visualisator.sendMemento(mem.get(this.cur));
            }
        }



    }

    public void Prev(){

        if(prohibition_flag == 2) {

            if ((this.cur <= this.mem.size() - 1) && this.cur > 0)
{
                --this.cur;
            }
```

```
            if ((this.cur <= this.mem.size() - 1) && this.cur >=
0) {
                visualisator.sendMemento(mem.get(this.cur));

            }
        }

    }

    public void saveGrid(String fileName){
        try {
            FileWriter writer = new FileWriter(fileName, false);
            writer.write(mem.get(cur).getGrid().getWidth());
            writer.append('\n');
            writer.write(mem.get(cur).getGrid().getHeight());
            writer.append('\n');
            writer.write(this.cur);
            writer.append('\n');
            for(int y = 0; y < mem.get(cur).getGrid().getHeight();
y++) {
                for (int x = 0; x <
mem.get(cur).getGrid().getWidth(); x++) {
                    switch (mem.get(cur).getObject(x,y).getType())
{
                        case 1:
                            writer.append(' ');
                            break;
                        case 2:
                            writer.append('#');
                            break;
                        case 3:
                            writer.append('s');
                            break;
                        case 4:
                            writer.append('f');
                            break;
                        case 5:
                            writer.append('+');
                            break;
                        case 6:
                            writer.append('-');
                            break;
                        case 7:
                            writer.append('x');
                            break;
                    }
```

```
                }
                writer.append('\n');
            }
            //writer.flush();
            writer.close();
        } catch (IOException e){

        }
    }

    public void loadGrid(String fileName) {
        try {
            if (this.cur != 0) {
                this.g_grid = cleanGrid(g_grid);
                this.mem.clear();
                this.g_path.clear();
                this.cur = 0;
            }
            FileReader reader = new FileReader(fileName);
            char c;
            int width = reader.read();
            c = (char) reader.read();
            int height = reader.read();
            c = (char) reader.read();
            int current = reader.read();
            c = (char) reader.read();
            createGrid(width, height);
            for (int y = 0; y < height; y++) {
                for (int x = 0; x < width; x++) {
                    Point location = new Point(x, y);
                    c = (char) reader.read();
                    switch (c) {
                        case ' ':
                            TheContentsOfTheCell empty = new
TheContentsOfTheCell.Empty(location, null, null);
                            g_grid.setObject(x, y, empty);
                            break;
                        case '+':
                            TheContentsOfTheCell open = new
TheContentsOfTheCell.Open(location, null, null);
                            g_grid.setObject(x, y, open);
                            break;
                        case '#':
                            TheContentsOfTheCell block = new
TheContentsOfTheCell.Block(location, null, null);
                            g_grid.setObject(x, y, block);
```

```
                            break;
                        case 's':
                            TheContentsOfTheCell start = new
TheContentsOfTheCell.Start(location, null, null);
                            g_grid.setObject(x, y, start);
                            break;
                        case 'f':
                            TheContentsOfTheCell finish = new
TheContentsOfTheCell.Finish(location, null, null);
                            g_grid.setObject(x, y, finish);
                            break;
                        case '-':
                            TheContentsOfTheCell close = new
TheContentsOfTheCell.Close(location, null, null);
                            g_grid.setObject(x, y, close);
                            break;
                        case 'x':
                            TheContentsOfTheCell path = new
TheContentsOfTheCell.Path(location, null, null);
                            g_grid.setObject(x, y, path);
                            break;
                    }
                }
                c = (char) reader.read();
            }
            reader.close();
            if (current != 0) {
                try {
                    Execute();
                } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                }
                this.cur = current;
                visualisator.sendMemento(mem.get(cur));
            }
        } catch (IOException e) {

        }
    }
}
```