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

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритма A*

Студентка гр. 5382 _____ Васильева Л. Ю.

Студент гр. 5382 _____ Смоляков И. Ю.

Студент гр. 5382 _____ Павлов Д. С.

Руководитель _____ Томша А. Э.

Санкт-Петербург
2017

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студентка Васильева Л. Ю. группы 5382

Студент Смоляков И. Ю. группы 5382

Студент Павлов Д. С. группы 5382

Тема практики: визуализация алгоритма A*

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: A*.

Сроки прохождения практики: 21.06.2017 – 04.07.2017

Дата сдачи отчета: 30.06.2017

Дата защиты отчета: 30.06.2017

Студентка _____ Васильева Л. Ю.

Студент _____ Смоляков И. Ю.

Студент _____ Павлов Д. С.

Руководитель _____ Томша А. Э.

АННОТАЦИЯ

Работа заключалась в написании програмы на Java, демонстрирующей алгоритм A*. Он состоит в поиске минимального пути в графе, является эвристическим алгоритмом. В отчете представлено описание алгоритма, спецификация программы и тесты, проверяющие правильность работы программы.

SUMMARY

The work consisted in the writing program in Java, showing the algorithm A *. It consists in finding the minimum path in the graph, is a heuristic algorithm. The report describes the algorithm, the specification of the program and the tests that verify the correctness of the program.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель учебной практики: получение основных сведений о языке программирования Java.

Для выполнения указанной цели поставлены следующие задачи:

1. Изучение литературы по Java.
2. Изучение библиотек Java, позволяющих реализовать графический интерфейс.
3. Написание программы, демонстрирующей алгоритм A* .

Алгоритм A* относится к эвристическим алгоритмам поиска по первому лучшему совпадению на графе с положительными весами ребер, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Краткое описание работы алгоритма:

Создается очередь вершин, которые нужно обработать (openset), и список вершин, которые были рассмотрены (closeset), а также карта маршрута пройденных вершин. Очередь будет строиться по возрастанию, относительно оценочной функции f(x) , которая является суммой стоимости пути от начальной вершины g(x) и эвристической оценки расстояния до цели h(x).

Шаг 1. Начальная точка start заносится в очередь обрабатываемых вершин, происходит вычисление f(start) (g(x) будет равен 0, а h(x) = h(start, end) , где end - конечная вершина).

Пока список обрабатываемых вершин не пуст выполняются шаги 2 и 3.

Шаг 2. Рассматривается вершина, у которой оценка f(x) наименьшая. Если это точка является конечной, то алгоритм завершает работу и по карте маршрута восстанавливает путь, пройденный алгоритмом. Текущая вершина перемещается в список рассмотренных вершин.

Шаг 3: Рассматривается каждый сосед текущей вершины. Если его нет в списке пройденных, то вычисляется текущая оценка, которая будет равна

сумме g(cur) и h(cur, neighbour) , где cur - рассматриваемая вершина, а neighbour - текущий сосед. Если соседа не было в очереди обратываемых вершин, то его туда добавляют. Если его добавили в список или его g(neighbour) меньше, чем его текущий, то происходит перевычисление f(neighbour)) и g(neighbour)), а также в карту заносится информация, что было совершено перемещение от соседа через текущую вершину. Переход на шаг 2.

Если список оказался пустым, то это означает , что алгоритм не нашел маршрут.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

Программа должна выполнять следующие функции:

1. Генерировать карту по запросу пользователя.

   Карта должна представлять собой поверхность, покрытую координатной сеткой. Клетки могут быть двух типов: препятствие и участок, по которому можно идти.

2. Давать пользователю возможность выбора клеток старта и финиша.
3. Запускать алгоритм по запросу пользователя.
4. Отображать найденный путь, а также процесс работы алгоритма (добавление клеток в openset и closet). В случае отсутствия пути между выбранными стартом и финишем сообщать, что его нет.
5. Очищать уже нарисованный путь по запросу пользователя.
6. Выводить соответствующие сообщения об ошибках при недопустимых действиях со стороны пользователя.

   Возможные сообщения об ошибках:

   1. Установка старта и/или финиша на препятствие.
   2. Запуск алгоритма до выбора старта и финиша.

Программа должна быть написана на Java и иметь графический интерфейс.

В программе используется алгоритм A* с эвристической функцией, вычисляющей манхэттенское расстояние между текущей точкой и точкой финиша (h(u) = |u.x — goal.x| + |u.y — goal.y|).

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

23.06 Разработка спецификации

26.06 Создание прототипа

- разработка интерфейса
- программирование алгоритма

28.06 Создание первой версии

- создание генератора карт
- сборка в один проект

30.06 Создание финальной версии

- устранение ошибок и багов

2.2. Распределение ролей в бригаде

Разработка алгоритма - Павлов Данила

Разработка интерфейса - Васильева Людмила

Генерация карты - Смоляков Иван

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры, используемые в программе

Классы, используемые в программе, представлены в таблице 1.

Таблица 1. Классы, используемые в программе.

Имя класса	**Назначение**
Main	Основной класс. Инициализирует окно приложения.
Window	Отвечает за графический интерфейс. Осуществляет действия для инициализации окна, обрабатывает нажатия на кнопки и клик по карте.
Canvas	Отвечает за рисование карты.
Map	Класс карты. Осуществляет генерацию карты. Содержит функции, необходимые для работы с ней.
Algorithm	Класс, реализующий алгоритм A*.

Диаграмма классов представлена в Приложении В.

3.2. Описание класса Main

Описание методов представлено в таблице 2.

Таблица 2. Описание методов класса Main.

Имя метода	main			
Назначение	Инициализирует окно приложения			
Модификатор доступа и тип	Public static void			
Возвращаемое значение	Ничего не возвращает			
Переменные	**Вид**	**Имя**	**Тип**	**Назначение**
	Формальная переменная	args	String[]	Параметры командной строки
	Локальная	window	Window	Содержит главное окно приложения

3.3. Описание класса Window

Описание полей представлено в таблицах 3 и 4.

Таблица 3 Описание полей класса Window.

Тип	Модификатор	Имя	Назначение
int	private static final	WINDOWS_WIDTH	Ширина окна приложения
int	private static final	WINDOWS_HEIGHT	Высота окна приложения
double	private static final	PROPOTION_RIGHT	Доля, занимаемая содержимым в правой части окна
double	private static final	PROPOTION_TOP	Доля, занимаемая содержимым в верхней правой части окна
int	private static final	BUTTON_SIZE	Высота кнопки
long	private static final	DELAY	Задержка для анимации работы алгоритма
Canvas	Private	canvas	Переменная для изображения карты
Point	Private	saveStart	Координата точки старта
Point	Private	saveFinish	Координата точки финиша
Frame	Private	frame	Переменная для главного окна приложения
Algorithm	private	alg	Переменная для переопределения некоторых методов класса алгоритм

Содержит следущие классы (представлены в таблице 4), интерфейс которых отнаследован от класса MouseListener.

Таблица 4. Поля-классы класса Window

<table>
<tr><th>Имя класса</th><th colspan="2">Описание переопределенных методов</th></tr>
<tr><td rowspan="4">GenerateListener</td><td>Имя</td><td>mouseClicked</td></tr>
<tr><td>Назначение</td><td>Обработка нажатия на клавишу</td></tr>
<tr><td>Модификатор доступа и тип</td><td>public void</td></tr>
<tr><td>Переменные</td><td>
<table>
<tr><th>Вид</th><th>Имя</th><th>Тип</th><th>Назначение</th></tr>
<tr><td>Формальная переменная</td><td>e</td><td>MouseEvent</td><td>Переменная для получения свойств события</td></tr>
</table>
</td></tr>
<tr><td rowspan="4">RunListener</td><td>Имя</td><td>mouseClicked</td></tr>
<tr><td>Назначение</td><td>Обработка нажатия на клавишу</td></tr>
<tr><td>Модификатор доступа и тип</td><td>public void</td></tr>
<tr><td>Переменные</td><td>
<table>
<tr><th>Вид</th><th>Имя</th><th>Тип</th><th>Назначение</th></tr>
<tr><td>Формальная переменная</td><td>e</td><td>MouseEvent</td><td>Переменная для получения свойств события</td></tr>
</table>
</td></tr>
<tr><td>ClearListener</td><td>Имя</td><td>mouseClicked</td></tr>
</table>

	Назначение	Обработка нажатия на клавишу
	Модификатор доступа и тип	public void
	Переменные	**Вид** | **Имя** | **Тип** | **Назначение** Формальная переменная | e | MouseEvent | Переменная для получения свойств события

Описание методов представлено в таблице 5

Таблица 5. Описание методов класса Window.

Имя метода	Window			
Назначение	Конструктор класса			
Имя метода	Init			
Назначение	Действия по инициализации главного окна			
Модификатор доступа и тип	Public void			
Возвращаемое значение	Ничего не возвращает			
Переменные	**Вид**	**Имя**	**Тип**	**Назначение**
	Локальные переменные	frame	JFrame	Главное окно приложения
		generateButton	JButton	Кнопка для генерации карты
		runButton	JButton	Кнопка для запуска алгоритма
		clearButton	JButton	Кнопка для очищения уже нарисованных путей
		c	GridBagConstraints	Переменная для менеджера расположения
		insets	Insets	Переменная для хранения значений отступов для содержимого правой части главного окна
Имя метода	includeOpenSet			
Назначение	Включение точки в openset (предстоит обработать)			
Модификатор доступа и тип	protected Queue<Point>			
Возвращаемое значение	Измененный openset			
Переменные	**Вид**	**Имя**	**Тип**	**Назначение**

	Формальные переменные	openset	Queue<Point>	Текущий openset
		current	Point	Включаемый элемент

Имя метода	includeCloseSet
Назначение	Включение точки в closeset (уже были обработаны)
Модификатор доступа и тип	protected Queue<Point>
Возвращаемое значение	Измененный closeset

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	closeset	Vector<Point>	Текущий closeset
	current	Point	Включаемый элемент

Имя метода	printWay
Назначение	Изображение кратчайшего пути от старта до финиша
Модификатор доступа и тип	private void
Возвращаемое значение	Ничего не возвращает

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальная переменная	way	Vector<Point>	Последовательность элементов пути от старта к финишу

Имя метода	correctCoordtoMap
Назначение	Перевод координат из координат изображения карты в номер строки и столбца для матрицы Map
Модификатор доступа и тип	private void
Возвращаемое значение	Переведенное значение координат

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальная переменная	x	int	Координата в формате изображения карты

3.4. Описание класса Canvas

Описание полей представлено в таблице 6.

Таблица 6. Описание полей класса Canvas.

Тип	Модификатор	Имя	Назначение
int	public static final	CANVS_WIDTH	Ширина изображения карты
int	public static final	CANVAS_HEIGHT	Высотаа изображения карты
int	public static final	POINT_SIZE	Размер элемента карты
Graphics2D	Private	g2d	Переменная для использования графики
Point	Private	coordStart	Координата точки старта
Point	Private	coordFinish	Координата точки финиша
Algorithm	Private	alg	Переменная для вызова методов алгоритма

Описание методов представлено в таблице 7

Таблица 7. Описание методов класса Canvas

Имя метода	drawMap			
Назначение	Рисование карты			
Модификатор доступа и тип	public void			
Возвращаемое значение	Ничего не возвращает			
Переменные	**Вид**	**Имя**	**Тип**	**Назначение**
	Формальная переменная	Map	_map	Карта, которую необходимо нарисовать
	Локальные переменные	I	int	Переменная для управления циклом
		J	int	Переменная для управления циклом
Имя метода	drawPoint			
Назначение	Рисование элемента карты			
Модификатор доступа и тип	public void			
Возвращаемое значение	Ничего не возвращает			
Переменные	**Вид**	**Имя**	**Тип**	**Назначение**
	Формальные переменные	newcoord	Point	Координаты элемента карты, который надо нарисовать
		Color	Color	Цвет этого элемента
	Локальные переменные	g2d	Graphics2D	Переменная для использования графики

Имя метода	setStartFinish
Назначение	Рисование элементов, соответствующих старту или финишу и стирание старых, если они были
Модификатор доступа и тип	public Point
Возвращаемое значение	Координаты старта или финиша

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	newcoord	Point	Координаты элемента, который надо нарисовать
	Type	boolean	Тип элемента (1 — старт, 0 - финиш)
	Coord	Point	Старые координаты, которые нужно стереть

Имя метода	setStart
Назначение	Вызов функции для рисования элемента карты, соответствующего старту
Модификатор доступа и тип	public void
Возвращаемое значение	Ничего не возвращает

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	X	int	X координата элемента старта
	Y	int	Y координата элемента старта

Имя метода	setFinish
Назначение	Вызов функции для рисования элемента карты, соответствующего финишу
Модификатор доступа и тип	public void
Возвращаемое значение	Ничего не возвращает

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	X	int	X координата элемента финиша
	Y	int	Y координата элемента финиша

Имя метода	paintComponent
Назначение	Рисование координатной сетки при запуске программы
Модификатор доступа и тип	public void

Возвращаемое значение	Ничего не возвращает
Переменные	

Вид	**Имя**	**Тип**	**Назначение**
Формальная переменная	g	Graphics	Переменная, необходимая для использования графики
Локальные переменные	g2d	Graphics2D	Переменная, необходимая для использования 2D графики (приведенная g)
	i	int	Переменная для управления циклом

3.5. Описание класса Map

Описание полей представлены в таблицах 8 и 9.

Таблица 8. Описание полей класса Map.

Тип	**Модификатор**	**Имя**	**Назначение**
int	public static final	DEF_SIZE	Размер карты по умолчанию
int	public static final	N_SEED	Количество стартовых точек в генераторе
int	public static final	N_UPDATE	Количество шагов генератора
Node[][]	private	_map	Карта
int	private	height	Высота карты
int	private	width	Ширина карты

Описание класса Node представлено в таблице 9.

Таблица 9. Описание класса Node.

Назначение	Свойства вершин карты
Поля	

Тип	**Модификатор**	**Имя**	**Назначение**
int	public	exist	Флаг существования вершины

Методы	public Node() - конструктор класса

Описание методов представлено в таблице 10.

Таблица 10. Описание методов класса Map

Имя метода	Map()
Назначение	Конструктор класса по умолчанию
Имя метода	Map(int, int)
Назначение	Конструктор класса
Переменные	

Вид	**Имя**	**Тип**	**Назначение**

	Формальная переменная	width	int	Ширина карты
		height	int	Высота карты

Имя метода	isExist
Назначение	Проверка вершины на существование
Модификатор доступа и тип	public boolean
Возвращаемое значение	Если существует то true, иначе false

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	point	Point	Координаты проверяемой вершины

Имя метода	setExist
Назначение	Установка модификатора существования вершины
Модификатор доступа и тип	public void
Возвращаемое значение	Ничего не возвращает

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	point	Point	Координаты изменяемой вершины
	exist	boolean	Новое значение

Имя метода	defMap
Назначение	Очистка карты
Модификатор доступа и тип	private void
Возвращаемое значение	Ничего не возвращает

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Локальные переменные	a	Node[]	Переменная для управления циклом
	i	int	Переменная для управления циклом

Имя метода	getHeight
Назначение	Получение высоты карты
Модификатор	public int

доступа и тип	
Возвращаемое значение	Высота карты
Имя метода	getWidth
Назначение	Получение ширины карты
Модификатор доступа и тип	public int
Возвращаемое значение	Высота карты
Имя метода	getHeight
Назначение	Получение высоты вершины
Модификатор доступа и тип	public int
Возвращаемое значение	Высота вершины

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальная переменная	point	Point	Координаты вершины

Имя метода	getWay
Назначение	Получение расстояния между соседними вершинами
Модификатор доступа и тип	public int
Возвращаемое значение	Расстояние между соседними вершинами

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные Переменные	point1	Point	Первая вершина
	point2	Point	Вторая вершина

Имя метода	Generate
Назначение	Генерация карты
Модификатор доступа и тип	public void
Возвращаемое значение	Ничего не возвращает

Переменные

Вид	**Имя**	**Тип**	**Назначение**

	Локальные переменные	rand	Random	Генератор случайных значений
		numUpdates	int	Количество обновлений «семян»
		seedPos	int[]	Массив позиций «семян»
		i, j	int	Переменные для управления циклами

Имя метода	setHeight
Назначение	Изменение высоты вершины
Модификатор доступа и тип	public void
Возвращаемое значение	Ничего не возвращает

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	point	Point	Координаты вершины
	height	int	Новая высота вершины

3.6. Описание класса Algorithm

Описание полей представлено в таблице 11.

Таблица 11. Описание полей класса Algorithm.

Тип	**Модификатор**	**Имя**	**Назначение**
Map	protected	_map	Карта, на которой будет работать алгоритм
int [][]	private static	G	Минимальное путь от начальной вершины до конкретной точки
int [][]	private static	F	Функция оценки для каждой вершины

Описание методов представлено в таблице 12.

Таблица 12. Описание методов класса Algorithm

Имя метода	Algorithm
Назначение	Конструктор класса

Переменные

Вид	**Имя**	**Тип**	**Назначение**

	Формальная переменная	map	Map	Карта, на которой будет работать алгоритм

Имя метода	generateMap
Назначение	Вызов внутреннего генератора карты
Модификатор доступа и тип	public Map
Возвращаемое значение	Сгенерированную карту

Имя метода	getMap
Назначение	Получение карты
Модификатор доступа и тип	public Map
Возвращаемое значение	карта

Имя метода	aStar
Назначение	Воспроизведение работы алгоритма A*
Модификатор доступа и тип	public Vector<Point>
Возвращаемое значение	Vector<Point> -маршрут от конечной до начальной точки

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	start	Point	Начальная точка
	end	Point	Конечная точка
Локальные переменные	openset	PriorityQueue<Point>	Очередь с приоритетом , служащая для сортировки нерассмотренных вершин по оценке F(x)
	closeset	Vector<Point>	Список рассмотренных вершин
	fromset	[] [] Point	Карта маршрута
	curr	Point	Текущая точка
	neighbor	Point	Сосед текущей точки
	better_result	boolean	Проверка на необходимость изменения свойств точки
	tentativeScore	int	Текущая оценка соседа и текущей вершины

Имя метода	reconstructPath
Назначение	Воспроизведение пути маршрута от начальной до конечной точки
Модификатор доступа и тип	protected Vector<Point>
Возвращаемое значение	Vector<Point>-маршрут от начальной до конечной точки

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	start	Point	Начальная точка
	end	Point	Конечная точка
	fromset	[] [] Point	Карта маршрута

Имя метода	setHeuristicFunction
Назначение	Вызов расчета эвристической функции
Модификатор доступа и тип	protected Integer
Возвращаемое значение	Результат эвристической функции

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	cell	Point	Текущая точка
	end	Point	Другая точка

Имя метода	includeCloseSet
Назначение	Включание элемента в closeset
Модификатор доступа и тип	protected Vector<Point>
Возвращаемое значение	Vector<Point> с добавленным элементом

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	closeset	Vector< Point>	Список обработанных вершин
	current	Point	Текущая вершина, которую нужно записать в список

Имя метода	removeOpenSet
Назначение	Удаление из списка openset элемента с минимальной оценкой f(x)
Модификатор доступа и тип	protected Queue<Point>
Возвращаемое значение	Queue<Point> без первого элемента

<table>
<tr><td>Переменные</td><td>
<table>
<tr><th>Вид</th><th>Имя</th><th>Тип</th><th>Назначение</th></tr>
<tr><td rowspan="2">Формальные переменные</td><td>openset</td><td>PriorityQueue <Point></td><td>Очередь приоритетов , в которой хранятся необработанные элементы</td></tr>
<tr><td>current</td><td>Point</td><td>Текущая точка , которую нужно удалить</td></tr>
</table>
</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Имя метода</td><td>includeOpenSet</td></tr>
<tr><td>Назначение</td><td>Включение в очередь openset элемента</td></tr>
<tr><td>Модификатор доступа и тип</td><td>protected Queue<Point></td></tr>
<tr><td>Возвращаемое значение</td><td>Очередь с добавленным элементом</td></tr>
<tr><td>Переменные</td><td>
<table>
<tr><th>Вид</th><th>Имя</th><th>Тип</th><th>Назначение</th></tr>
<tr><td rowspan="2">Формальная переменная</td><td>openset</td><td>PriorityQueue<Point></td><td>Очередь приоритетов , в которой хранятся необработанные элементы</td></tr>
<tr><td>current</td><td>Point</td><td>Текущая точка , которую нужно удалить</td></tr>
</table>
</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Имя метода</td><td>includeFromSet</td></tr>
<tr><td>Назначение</td><td>Включение элемента для создания маршрута</td></tr>
<tr><td>Модификатор доступа и тип</td><td>protected Point</td></tr>
<tr><td>Возвращаемое значение</td><td>Point</td></tr>
<tr><td>Переменные</td><td>
<table>
<tr><th>Вид</th><th>Имя</th><th>Тип</th><th>Назначение</th></tr>
<tr><td>Формальная переменная</td><td>current</td><td>Point</td><td>Текущая точка, которую записывают в fromset</td></tr>
</table>
</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Имя метода</td><td>takeMin</td></tr>
<tr><td>Назначение</td><td>Взятие минимального значения в очереди openset</td></tr>
<tr><td>Модификатор доступа и тип</td><td>protected Point</td></tr>
<tr><td>Возвращаемое значение</td><td>Point</td></tr>
<tr><td>Переменные</td><td>
<table>
<tr><th>Вид</th><th>Имя</th><th>Тип</th><th>Назначение</th></tr>
</table>
</td></tr>
</table>

	Формальная переменная	openset	PriorityQueue<Point>	Очередь приоритетов , из которой берут первый элемент

Имя метода	findNeighbours
Назначение	Создание вектора соседей текущей вершины
Модификатор доступа и тип	protected Vector<Point>
Возвращаемое значение	Vector<Point>- вектор соседей.

Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные переменные	current	Point	Точка , от которой находятся соседи
Локальные переменные	parametr1	boolean	Определение существования правого соседа
	parametr2	boolean	Определение существования нижнего соседа
	parametr3	boolean	Определение существования левого соседа
	parametr4	boolean	Определение существования верхнего соседа

Имя метода	Compare
Назначение	Компаратор, с помощью которого составляется очередь openset
Модификатор доступа и тип	public int
Возвращаемое значение	В зависимости от оценки F(x) , перставляет точки Point

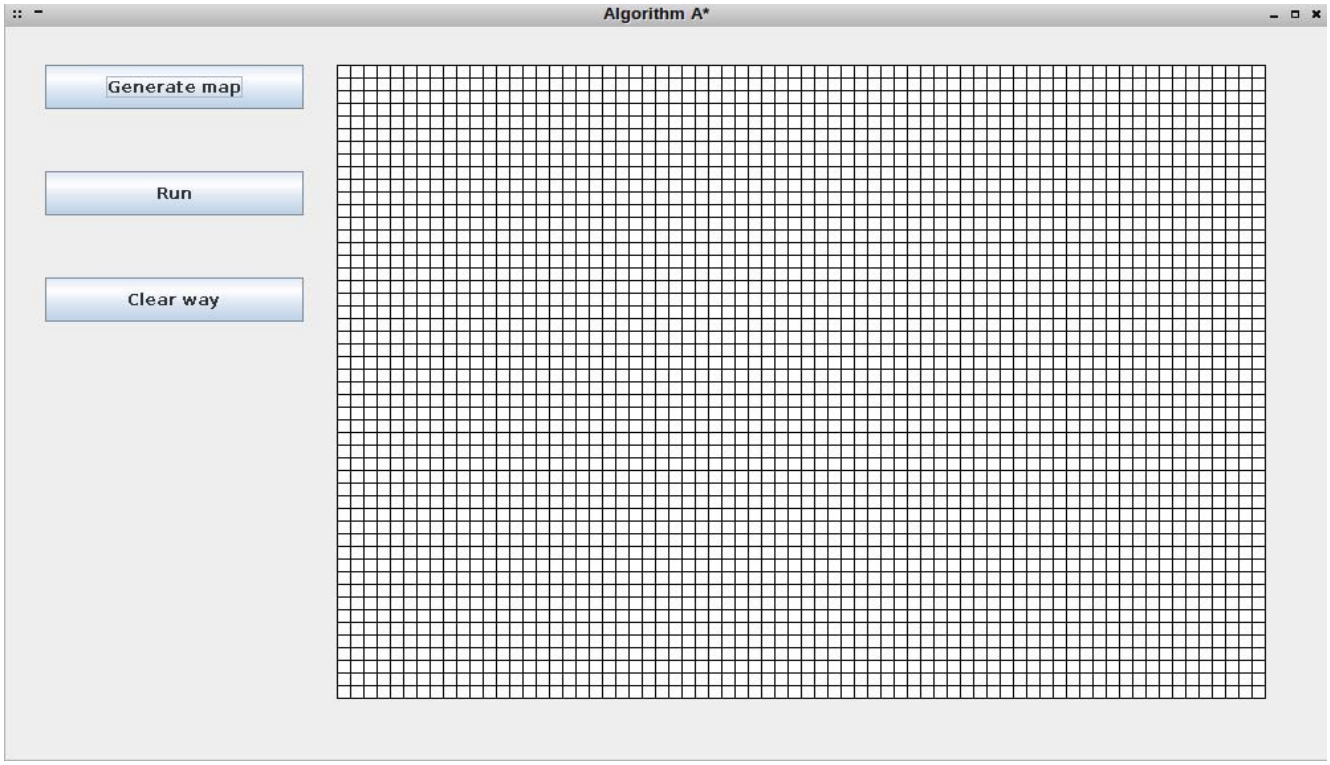
Переменные

Вид	**Имя**	**Тип**	**Назначение**
Формальные	P1	Point	Первая точка
	P2	Point	Вторая точка

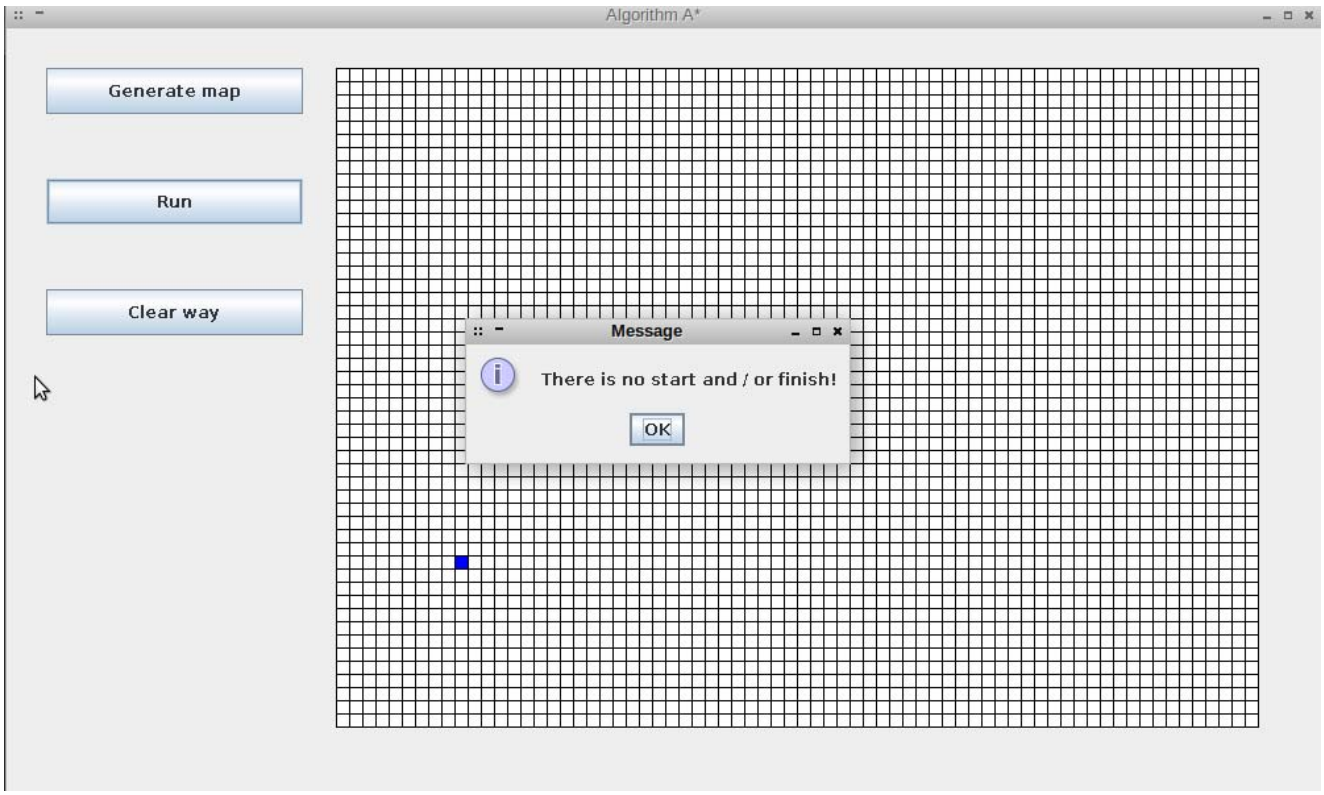
4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

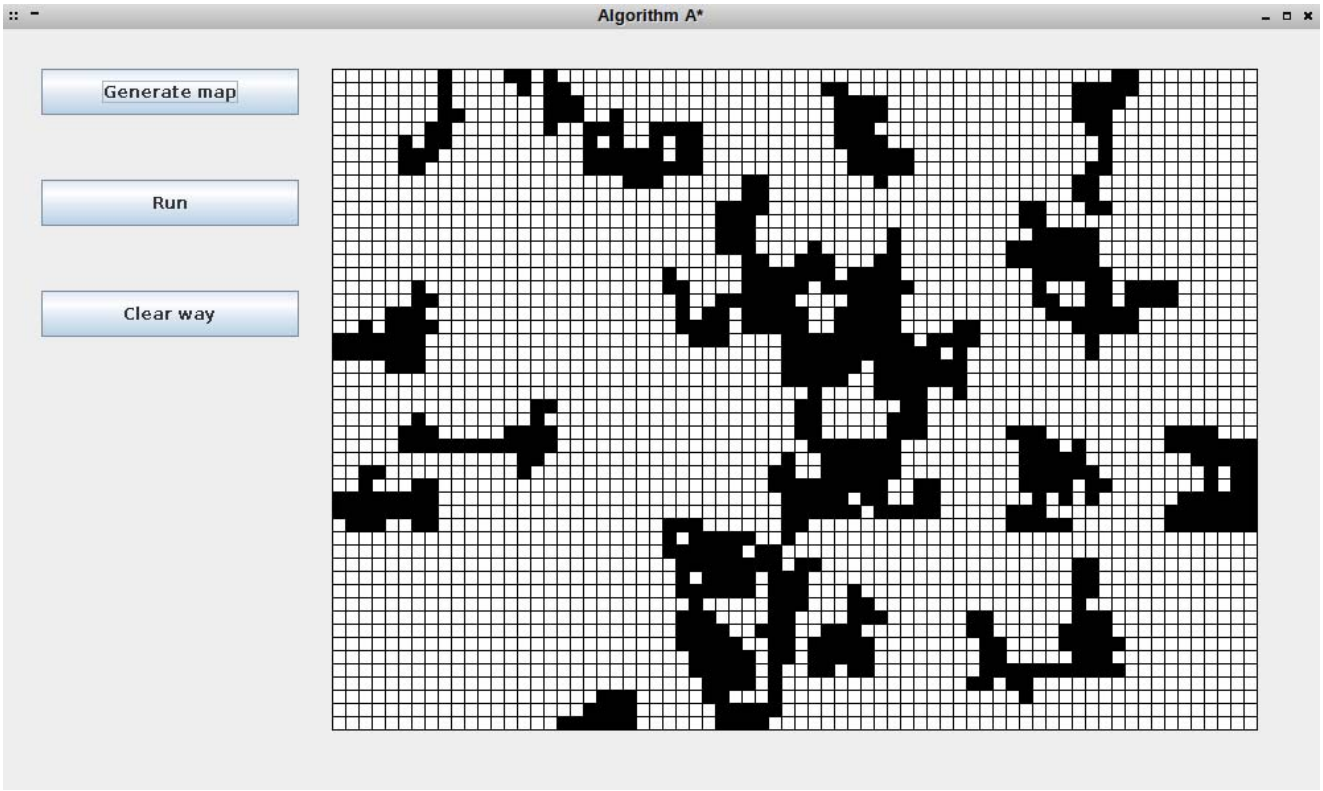
Тест 1. Запуск программы.



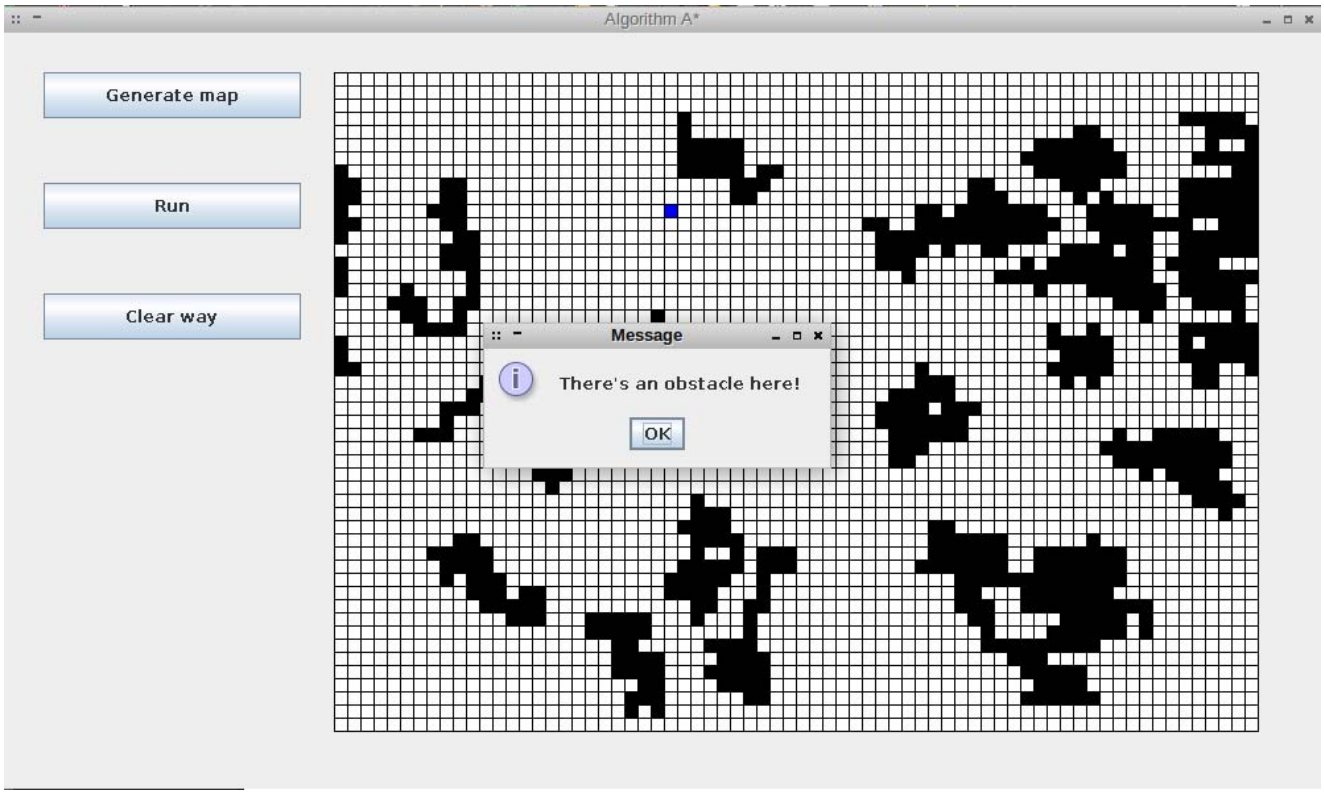
Тест 2. Нажатие кнопки Run до выбора старта и/или финиша приводит к появлению окна об ошибке.



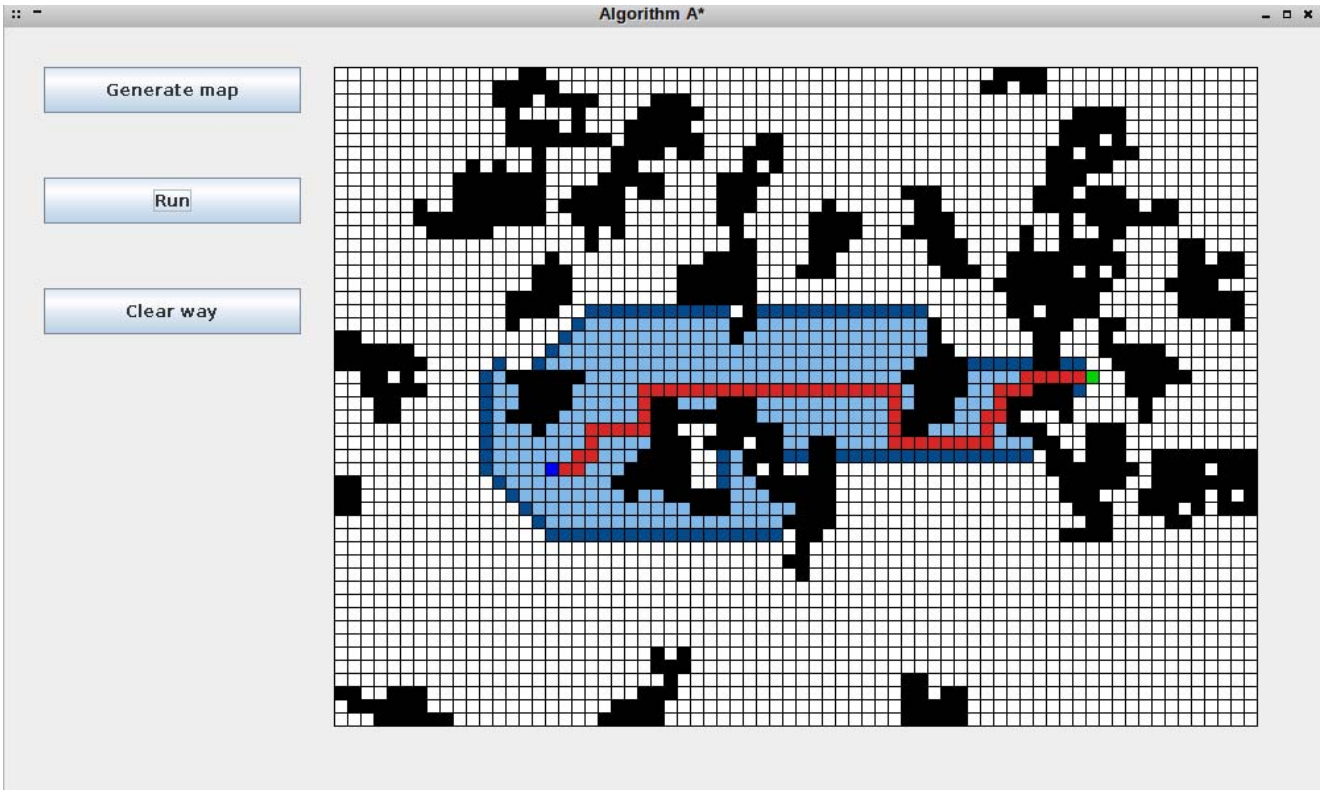
Тест 3. Нажатие кнопки Generaete map приводит к появлению изображения карты.



Тест 4. Установка старта или финиша на препятствие приводит к появлению окна об ошибке.



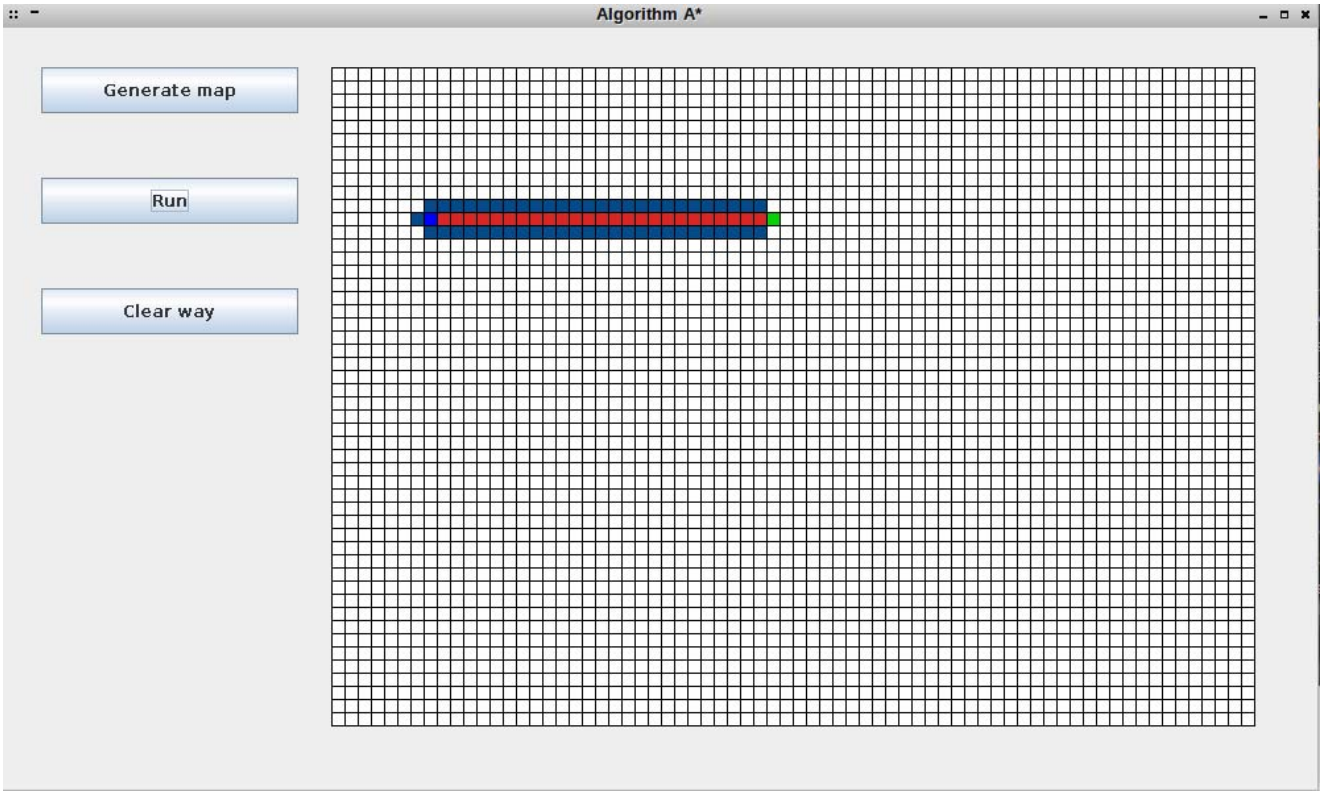
Тест 5. Нажатие кнопки Run приводит к выполнению алгоритма.



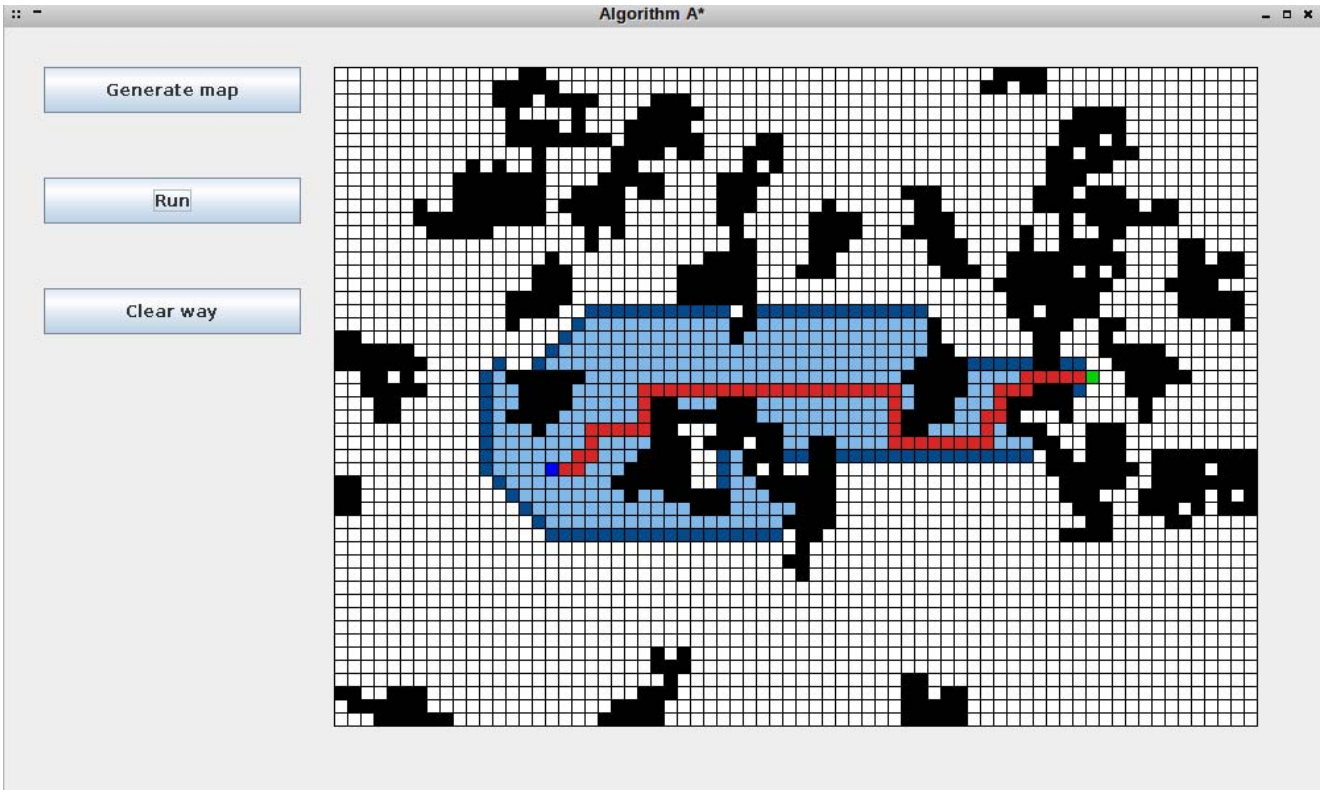
Тест 6. Нажатие кнопки Clear way приводит к стиранию нарисованного пути и сохранению изображения карты.

4.2. Тестирование алгоритма

Тест 1. При нахождении старта и финиша на одной прямой кратчайший путь представляет собой прямую линию.



Тест 2. Алгоритм находит кратчайший путь из возможных, проверяемые алгоритмом клетки выделяются в соответствии с ним.



ЗАКЛЮЧЕНИЕ

При выполнении данной работы были изучени основы языка программирования Java, ее библиотеки для работы с графикой (swing и awt), запрограммирован алгоритм A*.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Брюс Э. Философия Java.
2. https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_A*
3. https://stepik.org/course/Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81-187/syllabus
4. http://java-online.ru

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Main.java

```
public class Main {

	/**
	 * @param args
	 */
	public static void main(String[] args) {

		Window window = new Window();
		window.init();

	}

}
```

Файл Window.java

```
import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.Queue;
import java.util.Vector;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Window {

	private static final int WINDOW_WIDTH = 1000;
	private static final int WINDOW_HEIGHT = 600;
	private static final double PROPOTION_RIGHT = 0.08;
	private static final double PROPOTION_TOP = 0.05;
	private static final int BUTTON_SIZE = 10;
	private static final long DELAY = 20;

	private Canvas canvas;
	private Point saveStart, saveFinish;
	private JFrame frame;

	private Algorithm alg = new Algorithm(new Map(Canvas.CANVAS_WIDTH / 10,
			Canvas.CANVAS_HEIGHT / 10)) {
		@Override
		protected Queue<Point> includeOpenSet(Queue<Point> openset,
				Point current) {
			if (!(current.equals(saveStart)) && !
		(current.equals(saveFinish)))
				canvas.drawPoint(new Point(current.x * Canvas.POINT_SIZE,
						current.y * Canvas.POINT_SIZE), new Color(5,
				75, 140));

			return super.includeOpenSet(openset, current);
```

```
            }

            @Override
            protected Vector<Point> includeCloseSet(Vector<Point> closeset,
                                    Point current) {
                        if (!(current.equals(saveStart)) && !
      (current.equals(saveFinish)))
                                    canvas.drawPoint(new Point(current.x * Canvas.POINT_SIZE,
                                                current.y * Canvas.POINT_SIZE),
                                                new Color(129, 184, 234));
                        try {
                                    Thread.sleep(DELAY);
                        } catch (Exception e) {

                        }
                        return super.includeCloseSet(closeset, current);
            }

};

public void init() {
            frame = new JFrame("Algorithm A*");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
            frame.setResizable(false);
            frame.setLocationRelativeTo(null);
            frame.setLayout(new GridBagLayout());

            JButton generateButton = new JButton("Generate map");
            JButton runButton = new JButton("Run");
            JButton clearButton = new JButton("Clear way");
            canvas = new Canvas();

            GridBagConstraints c = new GridBagConstraints();
            Insets insets = new Insets(30, 30, 0, 15);

            c.gridx = 0;
            c.gridy = 0;
            c.gridheight = 1;
            c.gridwidth = 1;
            c.weightx = PROPOTION_RIGHT;
            c.weighty = PROPOTION_TOP;
            c.anchor = GridBagConstraints.NORTH;
            c.fill = GridBagConstraints.HORIZONTAL;
            c.insets = insets;
            c.ipady = BUTTON_SIZE;
            frame.add(generateButton, c);
            generateButton.addMouseListener(new GenerateListener());

            c.gridx = 0;
            c.gridy = 1;
            frame.add(runButton, c);
            runButton.addMouseListener(new RunListener());

            c.gridx = 0;
            c.gridy = 2;
            c.weighty = 1 - 2 * PROPOTION_TOP;
            frame.add(clearButton, c);
            clearButton.addMouseListener(new ClearListener());

            c.gridx = 1;
            c.gridy = 0;
```

```
		c.gridheight = 3;
		c.weightx = 1 - PROPOTION_RIGHT;
		c.fill = GridBagConstraints.BOTH;
		c.insets = new Insets(30, 10, 45, 45);
		frame.add(canvas, c);
		canvas.addMouseListener(new SelectStartEndListener());

		frame.setVisible(true);
	}

	private void printWay(Vector<Point> way) {
		if (way == null) {
			JOptionPane.showMessageDialog(frame, "There is not way!");
			return;
		}

		for (Point p : way) {
			if (!(p.equals(saveStart)) && !(p.equals(saveFinish)))
				canvas.drawPoint(new Point(p.x * Canvas.POINT_SIZE, p.y
						* Canvas.POINT_SIZE), new Color(217, 39, 39));
		}
	}

	private int correctCoordtoMap(int x) {
		return (x - x % Canvas.POINT_SIZE) / Canvas.POINT_SIZE;
	}

	public class GenerateListener implements MouseListener {

		public void mouseClicked(MouseEvent e) {
			saveStart = null;
			saveFinish = null;
			alg.generateMap();
			canvas.drawMap(alg.getMap());
		}

		public void mouseEntered(MouseEvent e) {
		}

		public void mouseExited(MouseEvent e) {
		}

		public void mousePressed(MouseEvent e) {
		}

		public void mouseReleased(MouseEvent e) {
		}
	}

	public class RunListener implements MouseListener {

		public void mouseClicked(MouseEvent e) {
			if (saveStart != null && saveFinish != null) {
				printWay(alg.aStar(saveStart, saveFinish));
			} else
				JOptionPane.showMessageDialog(frame,
						"There is no start and / or finish!");
		}

		public void mouseEntered(MouseEvent e) {
		}
```

```
	public void mouseExited(MouseEvent e) {
	}

	public void mousePressed(MouseEvent e) {
	}

	public void mouseReleased(MouseEvent e) {
	}
}

public class ClearListener implements MouseListener {

	public void mouseClicked(MouseEvent e) {
		saveStart = null;
		saveFinish = null;
		canvas.drawMap(alg.getMap());
	}

	public void mouseEntered(MouseEvent e) {
	}

	public void mouseExited(MouseEvent e) {
	}

	public void mousePressed(MouseEvent e) {
	}

	public void mouseReleased(MouseEvent e) {
	}
}

public class SelectStartEndListener implements MouseListener {

	@Override
	public void mouseClicked(MouseEvent e) {

		Integer x = e.getX();
		Integer y = e.getY();

		if ((alg.getMap()).isExist(new Point(correctCoordtoMap(x),
				correctCoordtoMap(y)))) {
			if (e.getButton() == MouseEvent.BUTTON1) {
				saveStart = new Point(correctCoordtoMap(x),
						correctCoordtoMap(y));
				canvas.setStart(x, y);
			} else {
				saveFinish = new Point(correctCoordtoMap(x),
						correctCoordtoMap(y));
				canvas.setFinish(x, y);
			}
		} else
			JOptionPane.showMessageDialog(frame,
					"There's an obstacle here!");

	}

	public void mouseEntered(MouseEvent e) {
	}

	public void mouseExited(MouseEvent e) {
	}
```

```
		public void mousePressed(MouseEvent e) {
		}

		public void mouseReleased(MouseEvent e) {
		}

	}

}
```

Файл Canvas.java

```
import java.awt.*;
import javax.swing.JComponent;


public class Canvas extends JComponent {

	private static final long serialVersionUID = 1L;
	private Graphics2D g2d;
	private Point coordStart, coordFinish;
	Algorithm alg;

	public static final int CANVAS_WIDTH = 700;
	public static final int CANVAS_HEIGHT = 500;
	public static final int POINT_SIZE = 10;

	public void drawMap(Map _map) {
		coordStart = null;
		coordFinish = null;
		for (int i = 0; i < _map.getHeight(); i++)
			for (int j = 0; j < _map.getWidth(); j++) {
				if (!_map.isExist(new Point(j, i))) {
					drawPoint(new Point(j * POINT_SIZE, i * POINT_SIZE),
							Color.black);
				} else {
					drawPoint(new Point(j * POINT_SIZE, i * POINT_SIZE),
							Color.white);
				}
			}

	}

	public void drawPoint(Point newcoord, Color color) {
		Graphics g = super.getGraphics();
		g2d = (Graphics2D) g;
		g2d.setPaint(color);
		g2d.fillRect(newcoord.x + 1, newcoord.y + 1, POINT_SIZE - 1,
				POINT_SIZE - 1);
	}

	public Point setStartFinish(Point newcoord, boolean Type, Point coord) {

		if (coord == null) {
			coord = new Point();
		} else {
			drawPoint(coord, Color.white);
		}

		coord.x = newcoord.x - (newcoord.x % POINT_SIZE);
		coord.y = newcoord.y - (newcoord.y % POINT_SIZE);
		drawPoint(coord, (Type) ? Color.blue : new Color(7, 210, 7));
```

```
            return coord;
        }

        public void setStart(int x, int y) {
            coordStart = setStartFinish(new Point(x, y), true, coordStart);
        }

        public void setFinish(int x, int y) {
            coordFinish = setStartFinish(new Point(x, y), false, coordFinish);
        }

        public void paintComponent(Graphics g) {

            super.paintComponents(g);
            g2d = (Graphics2D) g;

            g2d.setPaint(Color.white);
            g2d.fillRect(0, 0, CANVAS_WIDTH, CANVAS_HEIGHT);
            g2d.setPaint(Color.black);
            g2d.drawRect(0, 0, CANVAS_WIDTH, CANVAS_HEIGHT);

            for (int i = 0; i < CANVAS_WIDTH; i += POINT_SIZE) {
                g2d.setPaint(Color.black);
                g2d.drawLine(i, 0, i, CANVAS_HEIGHT);
            }

            for (int i = 0; i < CANVAS_HEIGHT; i += POINT_SIZE) {
                g2d.setPaint(Color.black);
                g2d.drawLine(0, i, CANVAS_WIDTH, i);
            }
        }

}
```

Файл Map.java

```
import java.util.Random;

import java.awt.Point;

public class Map {

        public final static int DEF_SIZE = 10;
        public final static int N_SEED = 30;
        public final static int N_UPDATE = 50;

        public Map() {
            this(DEF_SIZE, DEF_SIZE);
        }

        public Map(int width, int height) {
            _height = height;
            _width = width;
            defMap();
        }

        public boolean isExist(Point point) throws NullPointerException {
            return _map[point.x][point.y].exist;
        }

        public void setExist(Point point, boolean exist) {
```

```
		_map[point.x][point.y].exist = exist;
}

private void defMap() {
		_map = new Node[_width][_height];
		for (Node[] a : _map) {
			for (int i = 0; i < a.length; i++) {
				a[i] = new Node();
			}
		}

}

public int getHeight() {
		return _height;
}

public int getWidth() {
		return _width;
}

public int getHeight(Point point) {
		return _map[point.x][point.y].height;
}

public void setHeight(Point point, int height) {
		_map[point.x][point.y].height = height;
}

public int getWay(Point point1, Point point2) {
		return Math.abs(getHeight(point1) - getHeight(point2)) + 1;
}

public void generate() {
		defMap();
		Random rand = new Random();
		int[] seedPos = new int[N_SEED];
		int numUpdates = N_UPDATE;

		for (int i = 0; i < seedPos.length; i++) {
			seedPos[i] = Math.abs(rand.nextInt() % (_height * _width));
			setExist(new Point(seedPos[i] % _width, seedPos[i] / _width),
  false);
		}

		for (int i = 0; i < numUpdates; i++) {
			for (int j = 0; j < seedPos.length; j++) {
				switch (Math.abs(rand.nextInt()) % 4) {

				case 0:
					seedPos[j] -= _width;
					break;
				case 1:
					seedPos[j]++;
					break;
				case 2:
					seedPos[j] += _width;
					break;
				case 3:
					seedPos[j]--;
					break;
				}
```

```
                                        if (!(seedPos[j] < 0 || seedPos[j] >= (_height * _width)))
        {
                                                setExist(new Point(seedPos[j] % _width, seedPos[j] /
                                          _width),          false);
                                        }
                                }
                        }
        }

        private class Node {
                public Node() {
                        exist = true;
                        height = 1;
                }

                public boolean exist;
                public int height;
        }

        private Node[][] _map;
        private int _height, _width;
}
```

Файл Algorithm.java

```
import java.util.*;
import java.awt.Point;

public class Algorithm {

        /**
         * aStar Algorithm
         *
         */
        public Algorithm(Map map) {
                _map = map;
                G = new int[_map.getWidth()][_map.getHeight()];
                F = new int[_map.getWidth()][_map.getHeight()];
        }

        public void generateMap() {
                _map.generate();
        }

        public Map getMap() {
                return _map;
        }

        public Vector<Point> aStar(Point start, Point end) {
                Vector<Point> closeset = new Vector<>();
                Point[][] fromset = new Point[_map.getWidth()][_map.getHeight()];
                Queue<Point> openset = new PriorityQueue<>(_map.getWidth()
                                * _map.getHeight(), fieldComparator);
                G[start.x][start.y] = 0;
                F[start.x][start.y] = G[start.x][start.y]
                                + setHeuristicFunction(start, end);
                openset = includeOpenSet(openset, start);
                while (!openset.isEmpty()) {

                        boolean better_result = false;
```

```
                Point curr = takeMin(openset);
                if (curr.equals(end)) {
                        return reconstructPath(fromset, start, end);
                }

                openset = removeOpenSet(openset, curr);
                closeset = includeCloseSet(closeset, curr);
                Vector<Point> neighbours = findNeighbours(curr);

                for (Point neighbour : neighbours) {

                        if (closeset.contains(neighbour))
                                continue;

                        int tentativeScore = G[curr.x][curr.y]
                                        + _map.getWay(curr, neighbour);
                        if (!openset.contains(neighbour)) {

                                better_result = true;
                        } else {
                                if (tentativeScore < G[neighbour.x][neighbour.y])
                                        better_result = true;
                                else
                                        better_result = false;

                        }
                        if (better_result) {
                                fromset[neighbour.x][neighbour.y] =
                includeFromSet(curr);
                                G[neighbour.x][neighbour.y] = tentativeScore;
                                F[neighbour.x][neighbour.y] = G[neighbour.x]
                [neighbour.y]
                                                + setHeuristicFunction(neighbour, end);
                                if (!openset.contains(neighbour))
                                        openset = includeOpenSet(openset, neighbour);
                                else {
                                        openset = removeOpenSet(openset, neighbour);
                                        openset = includeOpenSet(openset, neighbour);
                                }
                        }
                }

        }
        return null;
}

protected Vector<Point> reconstructPath(Point[][] fromset, Point start,
                Point end) {
        Vector<Point> pathset = new Vector<>();

        Point curr = end;
        pathset.add(curr);
        while (curr != start) {
                curr = fromset[curr.x][curr.y];
                pathset.add(curr);

        }
        Collections.reverse(pathset);
        return pathset;

}
```

```
/**
 * Setting HeuristicFunction
 *
 * @param cell
 * @param end
 */
protected Integer setHeuristicFunction(Point cell, Point end) {
	return Math.abs(cell.x - end.x) + Math.abs(cell.y - end.y);
}

/**
 * Include in closeset elemets
 *
 * @param closeset
 * @param current
 * @return
 */
protected Vector<Point> includeCloseSet(Vector<Point> closeset,
			Point current) {
	closeset.add(current);
	return closeset;
}

/**
 * remove element from openset
 *
 * @param openset
 * @param current
 * @return
 */

protected Queue<Point> removeOpenSet(Queue<Point> openset, Point current) {
	openset.remove(current);
	return openset;
}

protected Queue<Point> includeOpenSet(Queue<Point> openset, Point current) {
	openset.add(current);
	return openset;
}

protected Point includeFromSet(Point current) {
	return current;
}

protected Point takeMin(Queue<Point> openset) {
	Point curr = openset.peek();
	return curr;
}

/**
 * Creation vector of Neighbours
 *
 * @param current
 * @return
 */
protected Vector<Point> findNeighbours(Point current) {
	Vector<Point> finded = new Vector<>();
	boolean parametr1 = (current.x + 1 <= _map.getWidth() - 1);
	boolean parametr2 = (current.y + 1 <= _map.getHeight() - 1);
	boolean parametr3 = (current.x - 1 >= 0);
	boolean parametr4 = (current.y - 1 >= 0);
```

```
            if (parametr1)
                if (_map.isExist(new Point(current.x + 1, current.y)))
                    finded.add(new Point(current.x + 1, current.y));
            if (parametr2)
                if (_map.isExist(new Point(current.x, current.y + 1)))
                    finded.add(new Point(current.x, current.y + 1));
            if (parametr3)
                if (_map.isExist(new Point(current.x - 1, current.y)))
                    finded.add(new Point(current.x - 1, current.y));
            if (parametr4)
                if (_map.isExist(new Point(current.x, current.y - 1)))
                    finded.add(new Point(current.x, current.y - 1));
            return finded;
        }

        protected Map _map;
        private static int[][] G;
        private static int[][] F;

        public static Comparator<Point> fieldComparator = new Comparator<Point>() {

            @Override
            public int compare(Point p1, Point p2) {

                return (int) (F[p1.x][p1.y] - F[p2.x][p2.y]);

            }
        };


}
```

ПРИЛОЖЕНИЕ В
ДИАГРАММА КЛАССОВ