**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по учебной практике**

Тема: Визуализация алгоритма A\* на языке java

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Лящевская А.П. |
| Студент гр. 7381 |  | Минуллин М.А. |
| Студент гр. 7381 |  | Лукашев Р.С. |
| Руководитель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

**ЗАДАНИЕ**

**на учебную практику**

|  |  |  |
| --- | --- | --- |
| Студентка Лящевская А.П. группы 7382 | | |
| Студент Минуллин М.А. группы 7381 | | |
| Студент Лукашев Р.С. группы 7381  Тема практики: Визуализация алгоритма А\* на языке java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.  Алгоритм: A\*. | | |
| Сроки прохождения практики: 01.07.2019 – 14.07.2019 | | |
| Дата сдачи отчета: 14.07.2019 | | |
| Дата защиты отчета: 14.07.2019 | | |
|  | | |
| Студентка |  | Лящевская А.П. |
| Студент |  | Минуллин М.А. |
| Студент |  | Лукашев Р.С. |
| Руководитель |  | Жангиров Т.Р. |

**Аннотация**

Цель учебной практики – получить практические навыки в визуализации алгоритмов, в разработке графического интерфейса, изучить и получить навыки использования языка программирования Java, получить навыки работы в команде. В работе представлена визуализация алгоритма A\*. Работа представляет собой командную итеративную разработку визуализатора алгоритма на языке программирования java. Для сборки проекта и организации юнит тестирования был использован фреймвоорк Intellij IDEA.

**Summary**

The purpose of educational practice is to gain practical skills in the visualization of algorithms, in the development of a graphical interface, to learn and acquire skills in using the Java programming language, to gain teamwork skills. The paper presents the visualization of the algorithm A \*. The work is a team iterative development of an algorithm visualizer in the java programming language. To build the project and organize the unit testing, the Intellij IDEA framework was used.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5-6 |
| 1. | Требования к программе | 7 |
| 1.1.1. | Требования к вводу исходных данных | 7 |
| 1.1.2. | Требования к визуализации | 9 |
| 1.1.3. | Требования к архитектуре | 10 |
| 1.2. | Уточнение требований после сдачи прототипа | 15 |
| 1.3. | Уточнение требований после сдачи 1-ой версии | 15 |
| 2. | План разработки и распределение ролей в бригаде | 16 |
| 2.1. | План разработки | 16 |
| 2.2. | Распределение ролей в бригаде | 16 |
| 3. | Особенности реализации | 17 |
| 3.1. | Использованные структуры данных | 17 |
| 3.2. | Использованные библиотеки | 18 |
| 4. | Тестирование | 0 |
| 4.1 | Тестирование графического интерфейса | 0 |
| 4.2 | Тестирование кода алгоритма | 0 |
|  | Заключение | 19 |
|  | Список использованных источников | 0 |
|  | Приложение А. Исходный код – только в электронном виде | 0 |

**введение**

Цель учебной практики – получить практические навыки в визуализации алгоритмов, в разработке графического интерфейса, изучить и получить навыки использования языка программирования Java.

Задача практики итеративная командная разработка визуализатора алгоритма A\*.

Реализуемый алгоритм применяется везде, где имеет место быть оптимальная оценка нахождения кратчайшего пути.

Алгоритм А\* — алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной). Алгоритм можно описать следующим образом:

1. Создается 2 списка вершин — ожидающие рассмотрения и уже рассмотренные. В ожидающие добавляется точка старта, список рассмотренных пока пуст.
2. Для каждой точки рассчитывается F = G + H. G — расстояние от старта до точки, H — примерное расстояние от точки до цели. О расчете этой величины будет сказано позднее. Так же каждая точка хранит ссылку на точку, из которой в нее пришли.
3. Из списка точек на рассмотрение выбирается точка с наименьшим F. Обозначим ее X.
4. Если X — цель, то мы нашли маршрут.
5. Переносим X из списка ожидающих рассмотрения в список уже рассмотренных.
6. Для каждой из точек, соседних для X (обозначим эту соседнюю точку Y), делаем следующее:
7. Если Y уже находится в рассмотренных — пропускаем ее.
8. Если Y еще нет в списке на ожидание — добавляем ее туда, запомнив ссылку на X и рассчитав Y.G (это X.G + расстояние от X до Y) и Y.H.
9. Если же Y в списке на рассмотрение — проверяем, если X.G + расстояние от X до Y < Y.G, значит мы пришли в точку Y более коротким путем, заменяем Y.G на X.G + расстояние от X до Y, а точку, из которой пришли в Y на X.
10. Если список точек на рассмотрение пуст, а до цели мы так и не дошли — значит маршрут не существует.

**Функция примерной оценки расстояния до цели.**

Эта функция должна выполнять несколько условий:

* Функция никогда не переоценивает расстояние до цели.
* Для это функции расстояния выполняется неравенство треугольника. Поясню подробнее: предположим у нас есть три точки — A, B и C. Для путей A-B B-C и A-C должно быть верно следующее неравенство: A-B + B-C >= A-C.

**1. требования к программе**

**1.1. Исходные Требования к программе**

**1.1.1 Требования к вводу исходных данных**

Вершины исходного графа задаются строками, а ребра целыми числами. Есть две возможности загрузить исходный граф в программу: через окно графического интерфейса и через файл.

Для загрузки графа через файл нужно иметь уже сохраненные файлы с графами. Поэтому требуется нажать на кнопку *Graph*, находящуюся в меню сверху и выбрать опцию *Load*. Далее откроется окно для выбора файла. Остается только выбрать нужный.

Для загрузки графа через экран графического интерфейса пользователю необходимо производить клики по специальному полю для ввода графа, формируя тем самым граф.

Для корректного отображения графа следует руководствоваться правилами:

Клик левой кнопкой мыши два раза — появляется одна вершина.

Клик правой кнопкой мыши по любому объекту — удаление этого объекта.

Для добавления ребра следует навести курсор на центр вершины, из которой будет выходить ребро, нажать и протянуть ребро до вершины окончания ребра, а затем отпустить. Ребро добавлено. По умолчанию веса ребер задаются их длинной на экране.

Для перемещения вершины следует навести курсор на край вершины дважды кликнуть, второй раз из которых следует удержать кнопку мыши. Далее передвинуть курсор мыши и отпустить в месте, куда требуется переместить вершину.

Все вышесказанное относится к редактированию графа в режиме добавления/удаления вершин.

Также есть режим добавления/удаления старта и финиша для алгоритма. В нем для добавления старта нужно кликнуть левой кнопкой мыши на вершине, а для финиша – правой.

Режимы переключаются на панели “Edit modes”.

**1.1.2 Требования к визуализации**

При запуске программы пользователь видит главное окно, в котором представлен весь интерфейс программы:

Панель меню. Его представляют следующие компоненты: кнопка *Graph*, отвечающая за загрузку(Load) и сохранение графа(Save as..) и кнопка помощи в которой находится справка по редактированию графа.

Две связанные кнопки редактирования графа, отвечающие за режимы добавления-удаления вершин и добавления-удаления старта и финиша. Добавлять ребра, к слову, можно в обоих режимах.

Связанные кнопки выбора эвристики – Манхеттенское расстояние, расстояние Чебышева и Евклидово расстояние.

Кнопки отката и вызова следующего шага алгоритма. Откат работает также с изменением старта и финиша, эвристики, режимами редактирования графа.

Трекер выбора задержки времени между шагами во время воспроизведения алгоритма.

Кнопки управления воспроизведением алгоритма – сброс, стоп и воспроизведение.

Панель логов программы.

И отдельное пространство для представления и редактирования графа.

Вид окна представлен на рис. 1.

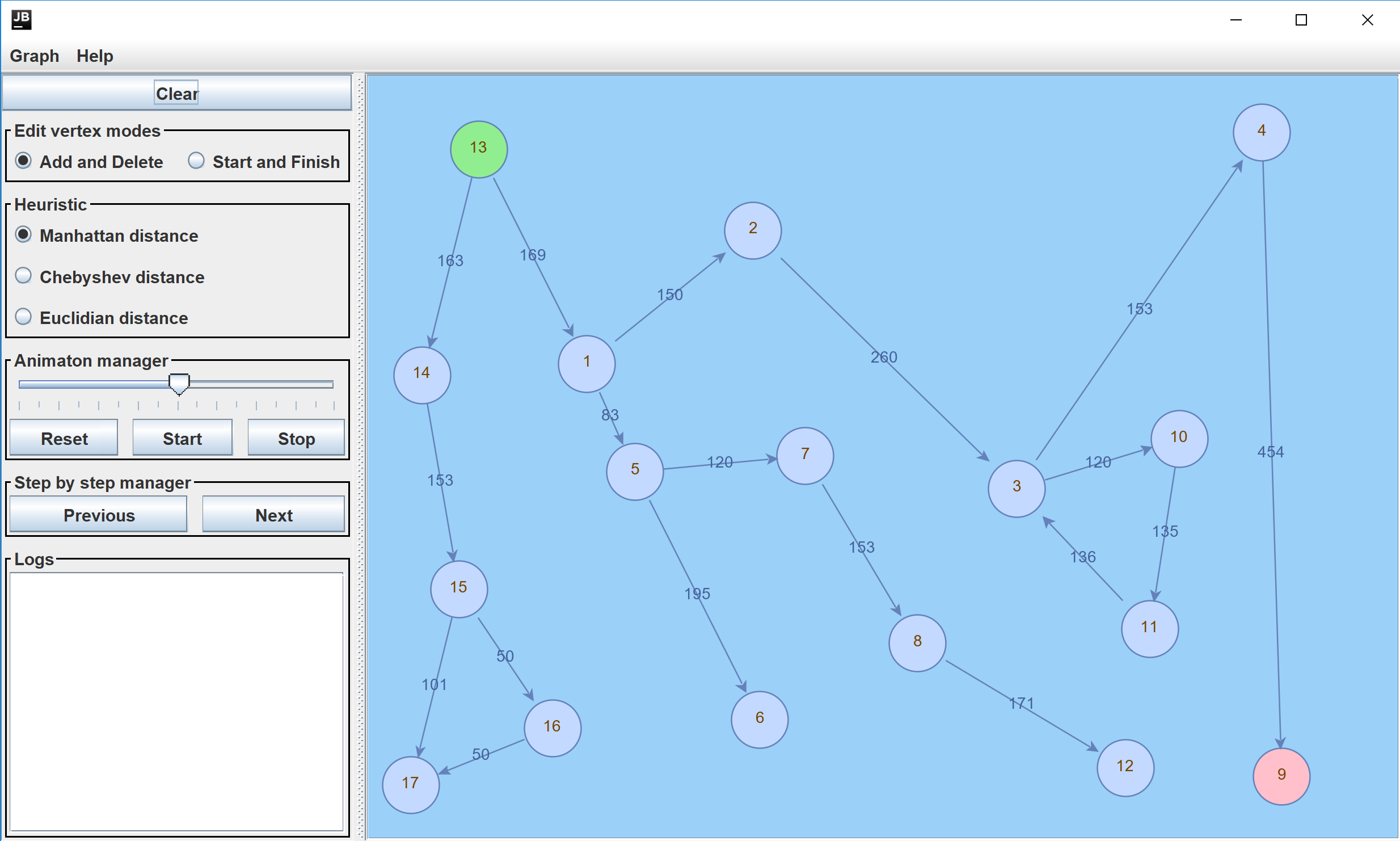


Рисунок 1 – Представление главного окна

**1.1.3 Требования к архитектуре**

Для сохранения состояний алгоритма для последующего отката назад (состояние графа, т.е. цвета ребер/вершин (для визуализации) + состояние алгоритма, т.е. текущие дистанции для вершин, текущая очередь вершин) реализован паттерн *Снимок*.

Используется классическая реализация через вложенный класс и опекуна.

Опекун: Класс, которым будем пользоваться в главной программе. Содержит экземпляр класса алгоритма, и стек (массив) снимков. Методы: *stepForward*(), *undo*().

Возможно добавление функционала восстановления состояния по снимку: добавить методы *getState*(step: *int*): *Memento*, *setState*(*m*: *Memento*). Используется для прыжка на определенный шаг назад или вперед (если существует, можно кидать исключение и обрабатывать его позже, например);

Снимок: Внутренний класс класса алгоритма. Содержит поля состояния (достаточные для восстановления состояния алгоритма, но не избыточные), и методы *Memento*(*state*), *getState*();

Класс алгоритма: поскольку нужно восстанавливать состояния, полное состояние алгоритма будет содержаться в его полях условно эту совокупность будем называть state. Методы: *nextStep*(), *save*(): *Memento*, *restore*(*m*: *Memento*).

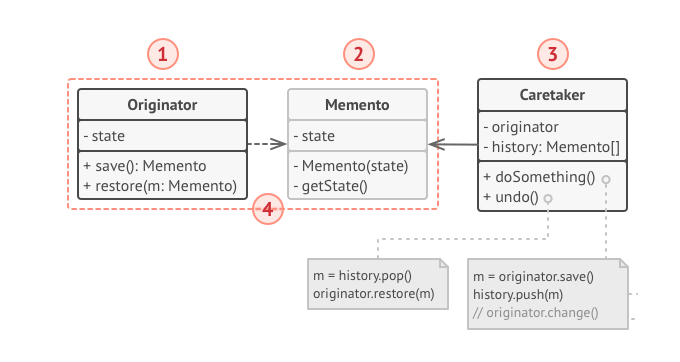


Рисунок 2 – диаграмма паттерна *Memento*.

Эвристика: один общий класс делегирует работу внутреннему объекту класса конкретной эвристической функции:

Интерфейс *Heuristic*: метод *get*(*obj1*: *Object*, *obj2*: *Object*) : *double*

Класс *Heuristics*: реализует *Heuristic*, доп. метод *changeHeuristic*(*func*: *Heuristic*).

Классы конкретных эвристических функций реализуют *Heuristic*.

Пример представлен на рис. 3.

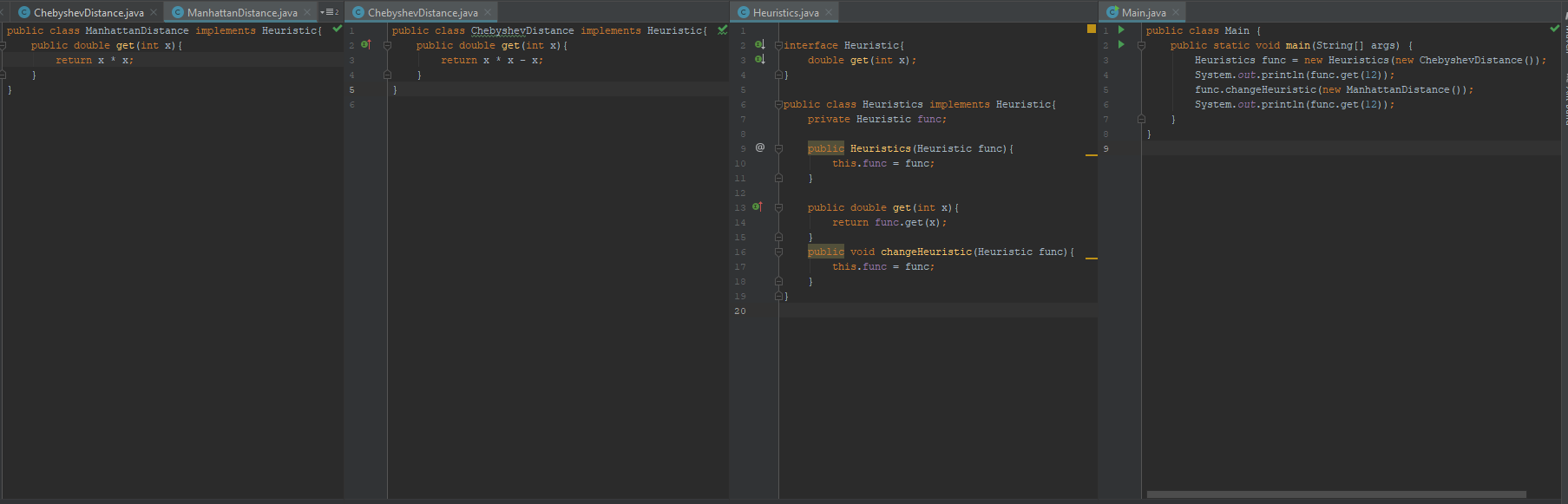


Рисунок 3 – пример делегирования по общему интерфейсу.

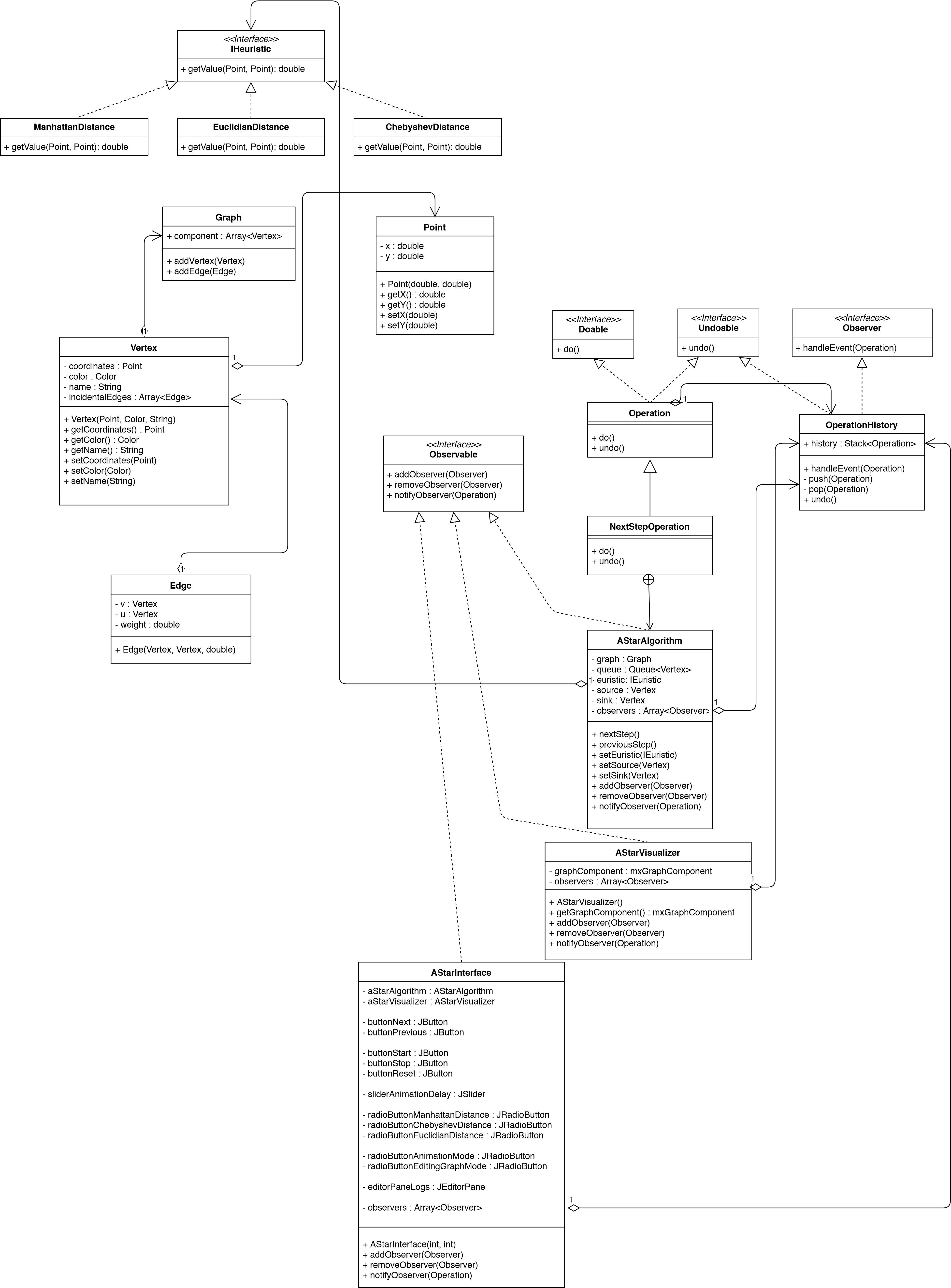


Рисунок 4 – Диаграмма классов.

Рисунок 5 – Диаграмма состояний

Режим

редактирования

графа

Сохранение графа

Загрузка графа из

файла

Режим анимации

(

воспроизведения

алгоритма)

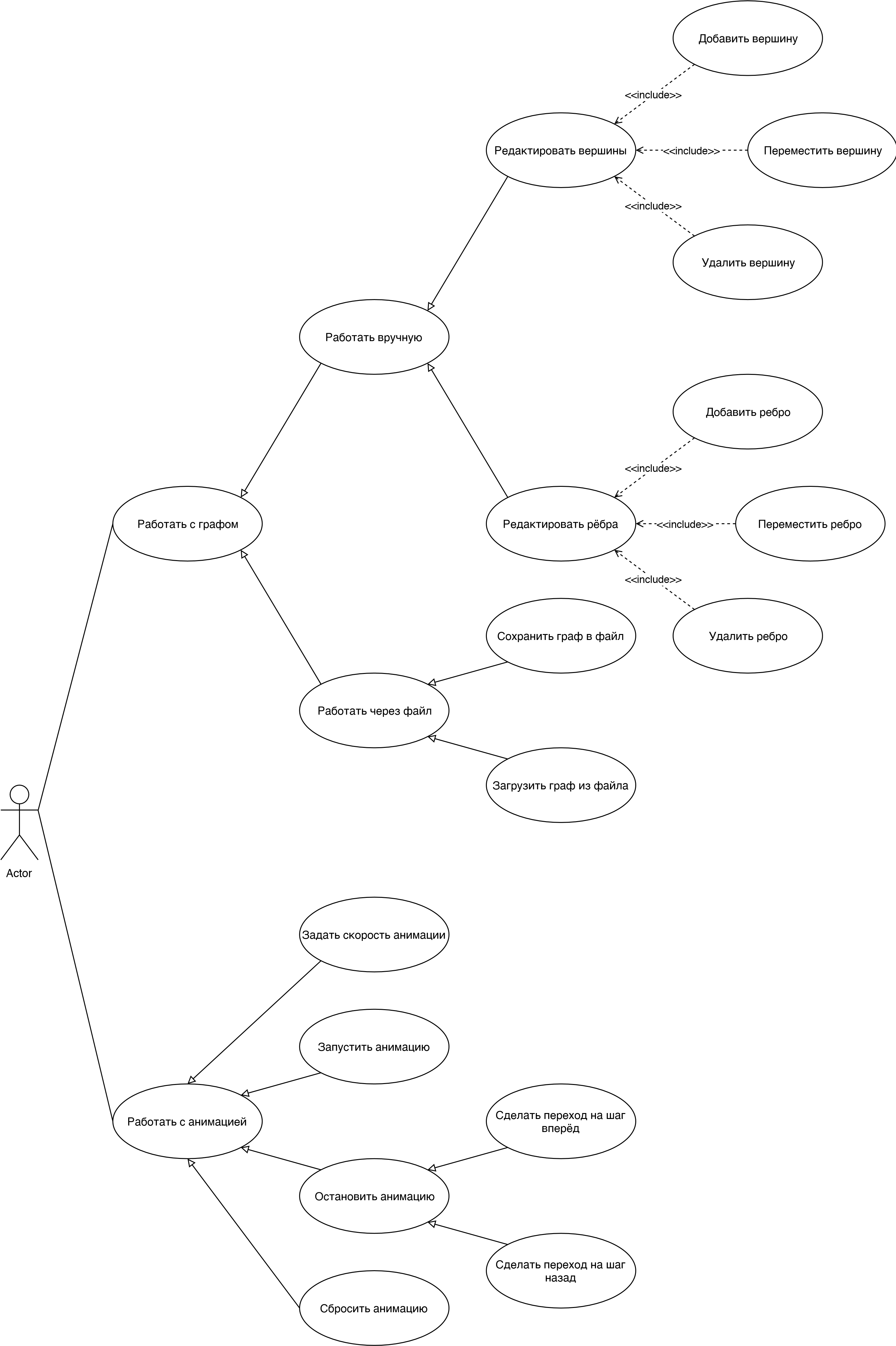


Рисунок 6 – Use Case Диаграмма

**1.2. Уточнение требований после сдачи прототипа**

После сдачи прототипа 04.07.2019 было принято решение отказаться от разделения режимов редактирования графа и работы с анимацией на разных вкладках. Теперь вкладки отсутствуют. Необходимый режим выбирается с помощью нажатия на соответствующую кнопку режима.

**1.3 Уточнения требований после сдачи 1-ой версии.**

Выявилось требование добавления uml диаграмм в отчет. Было принято решение о добавлении кнопки помощи, содержащей информацию о требованиях к исходным данным.

**2. План разработки и распределение ролей в бригаде**

**2.1. План разработки**

02.07.2019 — 04.07.2019 — разработка спецификации, согласование спецификации с руководителем, реализация некоторых отдельных частей программы (представление графа).

04.07.2019 — 08.07.2019 — разработка части визуализации, ответственной за представление графа и редактирование графа; разработка структуры проекта и разделение процесса работы по разным классам.

08.07.2019 — 11.07.2019 — разработка части реализации, ответственной за считывание графа из файла, сохранение графа в файл; разработка графической части визуализации, ответственной за отрисовку шагов алгоритма.

12.07.2019 – 14.07.2019 – удаление багов программы, построение логики деталей при использовании интерфейса, написание юнит-тестов.

**2.2. Распределение ролей в бригаде**

1. Минуллин — реализация.

2. Лящевская — визуализация.

3. Лукашев — архитектура.

**3. Особенности реализации**

**3.1. Использованные структуры данных**

Для реализации логики программы были созданы следующий классы:

1. **Point** – представление точки. Содержит сеттеры и геттеры для Х и Y.
2. **AStarAlgorithm** – реализует работу алгоритма. Содержит подклассы *NextStep, SetHeuristic, SetSink, SetSource, MyPair* (для реализации паттерна *Снимок*), являющийся наследником *UndoableOperation*, то есть действие, производимое им, можно отменить; а также сеттеры для корректного взаимодействия с другими классами.
3. **ASratVisualizer –** реализует визуальное представление графа, редактирование графа, раскраску, корректное сохранение в файл и загрузку.
4. **HeuristicFactory** – по полученной информации возвращает нужную эвристику.
5. **ChebyshevHeuristic** – одна из эвристик.
6. **ManhattanHeuristic** – одна из эвристик.
7. **EuclideanHeuristic** – одна из эвристик.
8. **OperationHistory** – хранилище истории операций. Реализовано на основе стека для UndoableOperation. Шаг вперед – добавление операции в стек. Шаг назад – удаление.
9. **UndoableOperation** – абстрактный класс реализующий *IDoable*, *IUndoable.*
10. **IDoable** – интерфейс представляющий собой идею продвижения вперед по истории.
11. **IUndoable** – интерфейс представляющий собой идею продвижения назад по истории.
12. **IObserver** – интейфейс представляющий собой идею наблюдателя.
13. **IObservable** –интейфейс представляющий собой идею наблюдаемого.
14. **IHeuristic** –интерфейсэвристик.

**3.2. Использованные библиотеки**

На протяжении всей работы над проектом велась работа с библиотекой *JGrapht*. Эта библиотека использует контейнер графа mxGraph, его визуализацию и предоставляет возможности для его ручного редактирования.

Таким образом, из библиотеки JGapht были использованы следующие классы:

1. DefaultDirectedGraph – для представления ориентированного графа.

2. JGraphXAdapter – Адаптер для рисования графика JGraphT с помощью библиотеки чертежей JGraphX.

3. mxGraphComponent – визуальная компонента графа.

**4. Тестирование**

**4.1. Тестирование визуализации**

**заключение**

Таким образом, был реализован алгоритм А\*. Был разработан интерфейс удобный и понятный для пользователя. Так же в ходе учебной практики были получены знания по языку программирования Java, получены навыки в визуализации алгоритмов и работе в команде.

**приложение А**

**Название приложения**

полный код программы должен быть в приложении, печатать его не надо