Взаимодействие между интерфейсом и программой реализовано через слушателей (mouseEventHandler, для каждой кнопки свой). Т.е. остается как есть в принципе.

Соответственно, у интерфейса есть доступ к алгоритму, а у алгоритма есть доступ к визуалайзеру. За всеми классами наблюдает история. Ответственность за генерацию операций за самими классами.

* При нажатии на чекербокс смены эвристики:

Отправка в историю события смены чекербокса

Вызов Alg.changeHeuristic()

* При нажатии на кнопку Next

Вызов Alg.nextStep()

Вызов Vis.repaint() для всех ребер/вершин, которые нужно поменять(ответственность внутреннего класса NextStep)

* При нажатии на кнопку Prev

Вызов His.prevStep()

* При нажатии на кнопку Reset

Вызов His.Reset()

Переименование кнопки Pause/Continue в Start

* При нажатии на кнопку Start

Запуск таймера, тот в свою очередь вызывает Alg.nextStep();

Переименование кнопки на Pause

* При нажатии на кнопку Pause

Остановка таймера.

Переименование кнопки на Continue

* При выборе стартовой/конечной вершины

Вызов Alg.setSource()/setSink()

При этом блокируется и восстанавливается функционал интерфейса:

|  |  |
| --- | --- |
| Если нажали на | То нельзя взаимодействовать с |
| Editing mode | Prev/Next/Start/Reset |
| Animation mode | Graph component (block listeners) |
| Start /Continue | Всем, кроме Stop |
| -\-\-\- | То восстанавливается взаимодействие с |
| Editing mode | Heuristic/GraphComponent/Edit vertex mode |
| Animation mode | Prev/Next/Start/Reset |
| Pause/Reset | Prev/Next |
| Желательно запретить редактирование уже обработанных вершин/ребер. Если научимся, то добавим. | |

**Поля:**

**Algorithm:**

IHeuristic heuristic;

Queue q

Object source, sink.

Visualizer vis

**Функции:**

**Algorithm:**

changeHeuristic(IHeuristic h): создает экземпляр Algorithm.HeuristicChange(h) и отдает его на обработку History.

nextStep(): создает экземпляр Algorithm.StepNext и отдает его на обработку History.

setSource/setSink (Object v): создает экземпляр Algorithm.SourceSet/SinkSet (v) и отдает его на обработку History.

**Interface:**

Функции для обработчиков событий. Для edit vertex mode, Mode, Animation manager при взаимодействии создаются и отправляются истории операции смены/изменения/переименования в виде обратимых операций\* и вызов функций алгоритма. Для Step by step manager происходит вызов функций алгоритма и истории. Где, что, как, в каком порядке указано выше.

**History:**

prevStep(): вытаскивает из истории блок операций между последними двумя операциями алгоритма. Т.е. находит ближайшую к вершине стека операцию [o1], созданную алгоритмом, после находит следующую ближайшую к вершине стека операцию [o2], созданную алгоритмом, и удаляет из истории блок (o1 … o2] вызывая для каждой из операций в этом блоке undo() сверху вниз.

Reset(): вытаскивает из истории все вплоть до первой операции, созданной алгоритмом включительно.

**Visualizer:**

repaint(Object v, Color c): создает экземпляр Visualizer.Repaint (v, c) и отдает его на обработку History.

**Обратимые операции:**

**Algorithm:**

**HeuristicChange(h):**

Поля: IHeuristic prev = heuristic;

Методы:

do() -> heuristic = h;

undo() -> heuristic = prev;

**StepNext: //TODO добавить что нужно, помимо очереди**

Поля: Queue prev;

Методы:

do() -> сделать шаг вперед алгоритма, отправить визуалайзеру запросы на смену цветов графа.;

undo() -> q = prev;

**SourceSet/SinkSet (v):**

Поля: Object prev = source/sink;

Методы:

do() -> source/sink = v;

undo() -> source/sink = prev;

**Visualizer:**

**Visualizer.Repaint (v, c):**

Поля: Color prev = v.color;

Методы:

do() -> v.color = c;

undo() -> v.color = prev;

**Interface:**

**ChangeEditVertexMode(selection) [selection prev, do(), undo()]**

**И т.д**

\*Обратимая операция: встроенный класс, реализующий функции do() и undo(). Каждая из них по сути работает с объектом внешнего для них класса.

Можно добавлять любые дополнительные методы/поля. Желательно оставить их приватными.