1 Anmerkungen zum Pflichtenheft

1.1 Klarstellungen

1.2 Änderungen

1.2.1 GUI

Graphen-Vorschau In der Graph-Preview Ansicht in der GUI werden die einzelnen Graphen, seien sie generiert oder importiert, unter einem neuen Tab angezeigt.

Diese Anzeige war bisher so gestaltet, dass die Graphen in einer Grid-View gesetzt werden. Dies würde in einer tabellenartigen Darstellung resultieren, bei der Beispielsweise 2 Spalten und 3 Reihen für die Graphen gleichzeitig dargestellt werden.

Diese Ansicht hatte den Nachteil, dass der User immer gezeichnete Graphen vor sich sieht. Dies führt zu deutlich geringerer Übersichtlichkeit. Außerdem bestand kein großes Interesse des Kunden daran, dass man die zuvor generierten Graphen sofort betrachten kann. Das graphische Darstellen der Graphen wurde eher an anderer Stelle gewünscht. Darüber hinaus ist diese Art der Ansicht nicht besonders gut skalierbar, wenn der User die Fenstergröße anpassen möchte, besteht die Gefahr, dass die Graphen-Bilder zu klein werden, um anschaulich zu sein.

Aus diesem Grund haben wir die Ansicht zu einer DropDownBox-Ansicht geändert. Dies bedeutet, dass man nun eine Liste an ausklappbaren DropDownBox-Elementen mit den jeweiligen Graphen-Namen vor sich sieht. Demzufolge kann man bei Interesse die einzelnen DropDownBoxen der jeweiligen Graphen ausklappen. Beim Ausklappen wird dann genau dieser zu betrachtende Graph gezeichnet. Daraus folgt, dass man nicht mehr mit Graphen-Zeichnungen überschüttet wird.

Durch diese Änderung entsteht ein weiterer Vorteil. Die Performance des Programms wird verbessert, da das Programm nicht sofort alle Graphen zeichnen muss, sondern diesen Task erst bei Bedarf starten muss.

Graphen-Generierung Möchte man die Heuristiken anwenden, benötigt man selbstverständlich hierfür erst einmal Graphen. Unser Programm stellt zu diesem Zweck mehrere Beschaffungsmöglichkeiten zur Verfügung: Automatische zufällige Generierung mit zuvor getätigten Einstellungen. Import bereits generierter Graphen. Im Graph-Editor von Grund auf neue Graphen von Hand erstellen.

Unter dem Tab "Graphen Generieren"der GUI war es bisher so gehalten, dass man als erstes die möglichen Einstellungsmöglichkeiten zur Generierung hat und sich darunter dann die verschiedenen Knöpfe befinden, welche die Generierung, den Import, oder das Zeichnen von Hand starten.

Diese Anordnung macht nur wenig Sinn, da man im Falle eines Imports oder auch des Editors keine Einstellungsmöglichkeiten benötigt.

Aus diesem Grund befinden sich nun die Buttons, welche die einzelnen möglichen Aktionen (Starten der Generierung, des Zeichnens oder Imports) ausführen, an oberster Stelle. Außerdem werden die Einstellungsmöglichkeiten zur zufälligen Generierung so lange vor dem User verborgen, bis er/sie aktiv auswählt diese Funktionalität wirklich zu benutzen.

Graphen-Editor Beim Graph-Editor kann man standardmäßig sowohl einen „Siple-Undirected-Graph“, als auch „Simple-Hyper-Graph“ editieren oder auch erstellen. Dabei gibt es unterschiedliche Funktionen, die dem User geboten werden um dies zu tun.

Bisher wurden diese nicht auf spezielle Graphentypen eingeschränkt.

Allerdings entsteht bei einigen der angebotenen Funktionen die Gefahr, dass der User den Graphentyp durch die gemachten Änderungen verändert, oder gar den gesamten Graphen ungültig für die weitere Bearbeitung macht.

Die daraus von uns getroffene Anpassung war es die Funktionen auf den Graphen-Typ einzuschränken und den Graph-Editor den Typ des editierten Graphen überprüfen zu lassen.

2 Übersicht

Allgemein Das von uns als Grundstein des gesamten Projekts gewählte Entwurfsmuster ist das „Model-View-Controller" (MVC) Entwurfsmuster. Es gliedert das Programm, wie der Name schon sehr stark vermuten lässt in die folgenden drei grundlegenden Bestandteile

[Modell] Das Model beschäftigt sich mit der gesamten Logik des Programms Hierunter zählen in unserem Fall hauptsächlich die verschiedenen Heuristiken.

[View / Präsentation] Die View stellt die Graphische Schnittstelle zum User dar.

[Controller / Steuerung] Der Controller bildet das Zwischenstück zu den bisherigen Teilen und verteilt die Aufgaben.

3 Model

4 View

4.1 Allgemein

Kommen wir nun zum nächsten Großen Abschnitt des Programm-Entwurfs. Nachdem wir im letzten Abschnitt über das Model gesprochen haben folgt nun der View-Teil des Model-View-ControllerEntwurfsmusters. Die View beschäftigt sich, wie der Name andeutet mit dem Aussehen des Programms und somit mit der graphischen Repräsentation.

Wie im Pflichtenheft beschreiben haben wir uns für die Entwicklung mit Java entschieden. Unter Java gibt es mehrere Möglichkeiten eine GUI zu erstellen.

1. Standart Widget Toolkit (SWT)
2. Abstract Widget Toolkit (AWT)
3. Swing
4. JavaFx

Uns war allerdings relativ schnell klar, dass die Wahl auf JavaFx fallen wird. Dies lag nicht zuletzt an FXML und der bisherigen Entwicklungs-Erfahrung. Dazu gleich mehr.

4.1.1 JavaFX

- JavaFX ist eine Abkürzung für Java Graphics.
- JavaFX ist eine Möglichkeit unter Java eine graphische Oberfläche zu erstellen.
- JavaFX ist eine komplette Neuentwicklung von Oracle.
- Es ist unabhängig von den bisherigen Methoden AWT und Swing.
- JavaFX wurde 2014 veröffentlicht.
- Es ist seit Version 7.6 in x86 Java Standard Edition (JavaSE) Runtime Installation enthalten.
- Da wir mit Java 8 arbeiten werden ist dies somit kein Problem.

JavaFX arbeitet mit einem Szenengraphen (engl. scene graph), der die einzelnen Bestandteile einer GUI verwaltet. Auf diesen werden dann alle weiteren Bestandteile gesetzt.

4.1.2 FXML

Wie auch bei den alternativen kann man natürlich auch mit JavaFx über zu schreibenden Code GUI-Objekte erstellen und diese auf den Scenen-Graphen aufbringen. Allerdings besteht mit JavaFx erstmals die Möglichkeit eine neue Form der GUI Entwicklung zu beschreiten. Diese erfolgt in Form von FXML.

FXML ist eine deklarative Beschreibung der grafischen Oberflächen auf XML-Basis. Dies bietet einige Vorteile gegenüber der konventionellen GUI-Entwicklung. Zum einen ist durch diese Technologie die Trennung des Designs der GUI und deren Funktionalität strikt getrennt. Zum anderen ist das Einfügen von GUI-Bestandteilen, die an mehreren Stellen der Benutzeroberfläche zum Einsatz kommen sehr einfach möglich. Dies Ermöglicht, dass der mehrfachverwendbare Code nur einmal in einem Separatem FXML-Dokument abgespeichert werden muss und dann über den „include-Tag“ an allen Stellen verwendet werden kann. Darüber hinaus können für die Gestaltung auch Web-Technologien wie CSS eingesetzt werden. Dies sorgt zusätzlich für eine Trennung von Layout auf der einen und Style und Design auf der anderen Seite, da separate CSS-Dateien erstellt werden können. Diese können dann in den FXML-Code eingebettet werden, sodass die GUI das Design übernehmen kann.

Die Entwicklung der FXML-Dateien erfolgt zuerst über den SceneBuilder. Dieser ist ein grafisches Tool, das die Erstellung von FXML-Dateien vereinfacht. Der daraus generierte Code wird bei Bedarf dann nochmals per Hand nachbearbeitet. Zur Nachbereitung zählen unter anderem auch das Einfügen der „include-Tags“ (wie oben beschrieben).

4.2 Entwurf

Der Entwurf der View gliedert sich prinzipiell in folgende Pakete auf:

1. Graphic
2. Drawer
3. Sound

Diese sind Sup-Pakete des "View-Packagesünd werden im folgenden genauestens unter die Lupe genommen.

Paket Graphic

The Graphic-Package is a Package for some adaptations and expansions with the JavaFx Stuff.

Paket Graphic.UIElements

The UI-Elements-Package contains new created UI-Elements that expand the JavaFx-UI.

Klasse ZoomableScrollPane

Beschreibung
This is an expansion to the JavaFx-ScrollPane. This adds the ScrollPane that it can be zoomed.

This is used so that the drawn Graph could be zoomed in/out so that the user can easily look for some Edges.

This Class is not made by ourself. @author https://www.pixelduke.com/2012/09/16/zooming-inside-a-scrollpane/

Dokumentation
Because this Class is already fully implemented by the creator, there will be no Documentation from our side.

Paket Drawer

The Drawer-Package. This Package contains everything that belongs to the Drawing of the Graphs. It is a upper-Package, therefore no further Documentation for this.

Paket Drawer.GraphDrawer

The GraphDrawer-Package contains like the Name suggested the GraphDrawer that visualizes the Graph and "drawsït to a JavaFx-Node for the User.

Klasse GraphDrawer

Beschreibung
The Drawer that draws the given Graph to the given JavaFx-Node. **Dokumentation**

- **graph : VisualGraph**
  The Graph that should be drawn.
- **fxNode : javafx.scene.Node**
  The JavaFx-Node where the Graph should be drawn on.
- **colourManager : ColourManager**
  The ColourManager of this Drawer to Map the ColourID's to the actual Colours of the to drawn Objects.

  This Object is created at the Constructor as new ColourManager and before the Drawing the ColourID's are added.
- **selectedVertices : List<Integer>**
  The List of Vertices-ID's that the user selected the Vertices at the GUI.

+ **GraphDrawer(graph : VisualGraph, fxNode : javafx.scene.Node)**
  The Constructor of this Class.

  Sets the given Graph and fxNode. Also initializes the ColourManager.
  @param graph The Graph that should be set as the Graph of this Drawer.
  @param fxNode The JavaFx-Node that should be set as the fxNode of this Class.

+ **printGraphTextual()**
  This Method prints the textual Representation onto the given JavaFx-Node.

+ **drawGraph(layout : GraphLayoutEnum)**
  This Method draws the Graph to the given JavaFx-Node by using the given Layout to position it's Vertices.
  @param layout The Enum that indicates which Layout the Drawer should use. If it is null the Drawer will use the Circle Layout.

- **drawVertex(vertexID : Integer)**
  Draw the given Vertex.

  Get the Vertex by searching for the given VertexID at the vertices-List of the given Graph. Use the GraphicLayout to get the correct Position of this Vertex.
  @param vertexID The ID of the to drawn Vertex.

- **drawVertexSelected(vertexID : Integer)**
  Draw the given Vertex as a selected Vertex.

  This Method is called if the to drawn Vertex of the drawVertex-Method is in the selectedVertices-List.

  The Vertex is drawn as selected by adding the corresponding Picture into the Vertex-JavaFx-Shape. Then the standard draw-Method is used to do the rest.
  @param vertexID The ID of the Vertex that should be drawn as a selected Vertex.

- **drawEdge(edgePosition : Integer)**
  Draw the Edge that is on the given Position at the Edge-List of the Graph of this Drawer.

  This Method only draws one Edge so that the Editor can show specific Edges. This Method is also called multiple times to draw all Edges.
  @param edgePosition The Position of the to drawn Edge at the List of Edges of the Graph.

+ **addToSelectedVertices(vertexID : Integer)**
  Add the given VertexID to the List of selected ones.
  @param vertexID The Vertex-ID that should be added to the List of selected Vertices.

+ **clearSelectedVerticesList()**
  Clear the List of selected Vertices-ID's.

+ **getGraph() : VisualGraph**
  Get the VisualGraph of this Drawer.
  @return returns The VisualGraph of this Drawer.

+ **setGraph(graph : VisualGraph)**
  Set the VisualGraph of this Drawer.
  @param graph The VisualGraph that should be set.

+ **setFxNode(fxNode : javafx.scene.Node)**
  Set the JavaFx-Node where the Graph should be drawn on.
  @param fxNode The Node that should be set as the JavaFx-Node to draw on.

Paket Drawer.ColourManager

The ColourManager-Package. This Package only contains the ColourManager which maps the abstract ColourID's that are given by the calculation into a real Colour-Value that could be drawn. This Class is separately because it provides a relatively general task, that easily can be (re)used elsewhere.

Klasse ColourManager

Beschreibung
The ColourManager manages the different Colours by Mapping the ColourID's to an actual Colour-Value, so that the Drawer can draw the coloured Graph by these ColourID's. **Dokumentation**

- **colourMap : Hashmap<IntegerString>**
  The HashMap of every ColourID to the actual Colour-Value that is represented as a String.

+ **ColourManager()**
  The Empty-Constructor of this Class. The Colours are added step by step at a later point.

+ **ColourManager(colourIDs : List<Integer>)**
  The Constructor of this Class. It adds the given ColourID's of the List and puts them into the Hashmap. Then the initColours-Method is called so that the mapping is completed for the given ColourId's.
  @param colourIDs The List of colourID's that should be mapped to real Colour-Values.

+ **initColours()**
  This Method has to be called when every ColourID is put into the HashMap. Then this Method calculates a Assignment of real Colours to the ColourID's and writes them into the HashMap, where it can be read out at a later Time.

+ **addColourID(colourIDs : Integer)**
  Add a new ColourID to the HashMap, where later the real Colour is mapped to.

  It is checked if the given ColourID is already at the HashMap.
  @param colourIDs The ColourID that should be added.

+ **getColourFromID(colourID : Integer) : javafx.scene.paint.Color**
  Get the real Colour-Object from the given ColourID. This Colour is then used to draw the Vertex/Edge to the screen to represent the Colouring-Solution.

  The initColour-Method has to be called first so that the ColourManager has already mapped the Colour-Values at the HashMap.
  @param colourID The colourID from what the colour should be.
  @return returns The actual Colour of the Object.

Paket Drawer.Layouts

This Package contains the implemented Layouts for the GraphDrawer and the Enum that Lists all of them.

Klasse GraphLayoutEnum

Beschreibung
This Enum Contains all implemented GraphLayout's that can be used by the graphDrawer to position the VisualVertices. This Enum is needed because the Drawer needs to know whitch Layout to use for the drawing of the Graph and this is done via this Enumeration. In our case there is only one Layout, because we well always draw the Graphs in a Circle. If someone wants to Expand this Drawer by adding a new Layout he/she/it has to update this Enum as well. This is not against ObjectOrienting Programming because the Programmer that would add this new Layout already needs to recompile the Program and therefore can

expand the Enum as well. **Dokumentation**

+ **circleLayout**
  The Enum for the possible Layouts.

  There will be only one Value in it because we will only use the Circle-Layout. But this is needed for possible extensions by other Programmers.

Klasse GraphLayout

Beschreibung
This is the Layout of the Drawing of the Graph. It is an abstract class so that there could be multiple Layouts for the Representation that implements this. **Dokumentation**

+ **GraphLayout()**
  The Constructor of this abstract Class. This is used at the Childs if they do not have an separate Constructor because they do not need parameters to set as well.

+ **executeLayout(graph : VisualGraph) : VisualGraph**
  This is an abstract Method and has to be implemented at the Sub-Classes.

  This Method set's the given Graph to the implemented Layout of the particular Child-Class. Therefore it sets the Positions of the Vertices of the given Graph.
  @param graph The Graph that gets the layout set on it. Therfore all Elements of this given Graph will be relocated to the calculated Position this Method calculates.
  @return returns The given Graph with the calculated Layout.

Klasse GraphLayoutCircle

Beschreibung
This is the Circle Layout of the Graph. Therefore this Layout orders the Graph-Nodes into a Circle.

It is an Child-Class of the abstract GraphLayout-Class. **Dokumentation**

- **radius : Double**
  The Radius of the Circle where the Elements should be positioned at.

+ **GraphLayoutCircle(radius : Double)**
  The Constructor of this Class.

  Sets the given Radius as radius of this Layout.
  @param NAME The Radius to set.

+ **executeLayout()**
  This is the overwritten Method from the abstract-Parent-Class.

+ **setRadius(radius : Double)**
  The Setter for the Radius.
  @param radius The Radius to set.

Paket Drawer.Visualization.VisualizationGraph

Klasse VisualVertex

Beschreibung
The Vertex of an Visual-Graph. It is the Child of the JavaFx-Circle Object so this Vertex can be drawn.
Dokumentation

- **ID : Integer**
  The Identification-Number (ID) of this Node. This Variable is Final.

+ **VisualVertex(id : Integer)**
  The Constructor of this Class.

  It contains only the final-ID as Parameter to set. The Parameters of the JavaFx-Node will be set by the Layout if it calculates the Position of this Vertex.
  @param id The ID that will be set to this Vertex.

+ **getID() : Integer**
  Get the ID of this Vertex.
  @return returns The Integer-Value of the ID of this Vertex.

+ **toString() : String**
  This Method overwrites the standard toString-Method.
  @return returns It returns a String-Representation of this VisualVertex. «ID>"

Klasse VisualVertexColoured

Beschreibung
Extends the VisualVertex Class.

This Vertex also contains a Colour-ID so that the Vertex can be coloured. **Dokumentation**

- **colourID : Integer**
  The ID of the Colour used by the Heuristic. This is like a Foreign-Key of the Colour.

  Remember: The actual colour of the specific Elements are not important because the User wants to see if the calculation of the Heuristic found a solution not what colour the Elements have. The Colour-ID can be associated with different drawing-colours for different draws without changing the statement of the Program.

+ **VisualVertexColoured(id : Integer, colourId : Integer)**
  The Constructor of this Class.

  It contains only the final-ID as Parameter to set. The Parameters of the JavaFx-Node will be set by the Layout if it calculates the Position of this Vertex.
  @param id The ID that will be set to this Vertex.
  @param coulorID The ID that will be set to this Vertex.

  If this Vertex is not coloured jet set the colour to null or use the other constructor.

+ **isColoured() : Boolean**
  Checks if this Vertex is Coloured.

Therefore this Method checks if the ColourID is null or an actual Integer-Value.
@return returns True if the ColourID-Varialbe is set and false if not.

+ **getColourID() : Integer**
  Get the ColourID of this Vertex.
  @return returns The Integer-Value of the ColourID of this Vertex.

+ **setColourID(colourID : Integer)**
  Set the ColourID of this Vertex.
  @param The Colour-ID this Vertex should be coloured with.

+ **toString() : String**
  This Method overwrites the standard toString-Method.
  @return returns It returns a String-Representation of this VisualVertexColoured. «ID>:<ColourID>"

Klasse VisualEdge

Beschreibung
The Edge of an Visual-Graph. It is the Child of the JavaFx-Polygon Object so this Edge can be drawn. **Dokumentation**

- **connectedVerticesID : List<Integer>**
  This List contains all Vertices-ID's from the Vertices this Edge connects.

+ **VisualEdge(connectedVertices : List<VisualVertex>)**
  The Constructor of this Class.

  Set's the given List of by this Edge connected Vertices to the List of this Object.
  @param connectedVertices The List of by this Edge connected Vertices. This given List will be set to the List of this Edge-Object.

+ **connectsSame(compareEdge : VisualEdge) : Boolean**
  Checks if the given VisualEdge is an edge between the Same Vertices as this Edge.
  @param compareEdge The Edge of which the connected-Vertices should be checked with.
  @return returns If the two Edges are conections between the same Vertices it returns true, else false.

+ **getConnectedVertricesIDList() : List<Integer>**
  Get the List of the connected VerticesIDs.
  @return returns The List of the Vertices-ID's that this Edge connects.

+ **toString() : String**
  This Method overwrites the standard toString-Method.
  @return returns It returns a String-Representation of this VisualEdge. "<VertexID1>, ..."

Klasse VisualEdgeColour

Beschreibung
Extends the VisualEdge Class. This Edge also contains a Colour-ID so that the Edge can be coloured. **Do-**

kumentation

- **colourID : Integer**
  The ID of the Colour used by the Heuristic. This is like a Foreign-Key of the Colour.

  Remember: The actual colour of the specific Elements are not important because the User wants to see if the calculation of the Heuristic found a solution not what colour the Elements have. The Colour-ID can be associated with different drawing-colours for different draws without changing the statement of the Program.

+ **VisualEdgeColoured(connectedVertices : List<VisualVertex>, colourId : Integer)**
  The Constructor of this Class.

  Set's the given List of by this Edge connected Vertices to the List of this Object.
  @param connectedVertices The List of by this Edge connected Vertices. This given List will be set to the List of this Edge-Object.
  @param coulorID The ColourID that will be set to this Edge.

  If this Edge is not coloured jet set the colour to null or use the other constructor.

+ **isColoured() : Boolean**
  Checks if this Vertex is Coloured.

  Therefore this Method checks if the ColourID is null or an actual Integer-Value.
  @return returns True if the ColourID-Varialbe is set and false if not.

+ **getColourID() : Integer**
  Get the ColourID of this Edge.
  @return returns The Integer-Value of the ColourID of this Edge.

+ **setColourID(colourID : Integer)**
  Set the ColourID of this Edge.
  @param The Colour-ID this Edge should be coloured with.

+ **toString() : String**
  This Method overwrites the standard toString-Method.
  @return returns It returns a String-Representation of this VisualEdge-Coloured. "<VertexID1>, ...:<ColourID>"

Klasse VisualGraph

Beschreibung
This is the VisualGraph. It is the Graph-Construct that is used for the Drawing.

Remember: VisualGraph is Generic VisualGaph<V extends VisualVertex, E extends VisualEdge> This is necessary so that the Graph can differentiate between the uncoloured and the coloured Elements.

This separate Graph-Representation for the View is necessary because the Model and the View of the Rage-Program should be strictly separated and therefore the View could not use the same Graph-Object. As well this Graph-Representation uses special Nodes and Edges as Elements that could be drawn. **Dokumentation**

- **vertices : List<VisualVertex>**
  This is a List of all Vertices (=Node's) of this Graph.

Remenber: At any further Point the Nodes"will be named Vertex/Vertices because of the confusion with JavaFx-Nodes that would otherwise occur.

- **edges : List<VisualEdge>** This is a List of all Edge's of this Graph.

+ **VisualGraph()**
The Empty-Constructor of this Class.

+ **isColoured()**
Checks if the Graph is made out of VisualVertexColoured and VisualEdgeColoured and if so if the ColouredID's of all Objects are set.
@return returns If they are set it returns true, and if not false.

+ **addVertex()**
Add a new Vertex to the List of Vertices of this Graph.

If the List is not instanciated yet this will be done.

To add a new Vertex this Method searches for the next unused Integer-ID that could be used for a new Node and created the VisualVertex-Object with this Parameter. This created Object will be added to the List.

+ **addVertex(vertex : VisualVertex) : Boolean**
Add the given Vertex to the List of Vertices.

If the List is not instanciated yet this will be done.

Also it is checked that the Vertex-ID is not already used by another Vertex. If so the given Vertex will not be added.
@param vertex The Vertex that should be added to this Graph.
@return returns If the Vertex-ID was added this Method returns true, otherwise false.

+ **addVertex(vertices : List<VisualVertex>)**
Add a whole List of Vertices to this Graph.

This is done by calling the addVertex-Method multiple times.
@param vertices The List of Vertices that should be added to the List.

+ **addVertex(amount : Integer)**
Add the given amount of Vertices to the Graph.

This is done by calling the addVertex-Method multiple times.
@param amount The amount of Vertices the user wants to add to this Graph.

+ **addEdge(edges : VisualEdges)**
Add the given Edge to the Graph.

If the List is not instanciated yet this will be done.

Also it is checked if this Edge has the exact same connected Vertices as any other Edge of this Graph. This is done by calling the connectSame-Method of the given Edge.

Also it is checked that the given Edge is valid. That means that this method checks if all connected-Vertices that are given by ID are Vertices of this Graph. If there is an unexisting Vertex this Vertex will be created and added to the Graph by calling the addVertex(VisualVertex)-Method.
@param edge The Edge that should be added to this Graph.

Check if this Edge contains valid VertexID's and if it only connects Vertices that are not currently connected.

+ **addEdge(vertices : List<VisualEdges>)**
Add a whole List of Edges to this Graph.

This is done by calling the addEdge-Method multiple times.
@param edges A List of Edges that should be added.

+ **addEdge(verticeIDs : List<Integer>)**
Add the Edge, that is given by the List of Vertice-ID's, to the graph.

This is done by creating an new VisualEdge-Object with the given List as Parameter and then calling the addEdge-Method.
@param verticeIDs The List of Vertice-ID's that should be connected by Edge that should be added.

+ **removeVertex(vertex : VisualVertex)**
Remove the given Vertex from the Graph.

If an Edge was connected to this Vertex and it only contains one other Vertex after the deletion, the Edge will be removed too.
@param vertex The Vertex that should be removed.

+ **removeVertex(vertexID : Integer)**
Remove the Vertex, by the given ID, from the Graph.

This is done by calling the removeVertex-Method. (The Vertex that should be deleted can be found at the Vertices-List by the given ID).
@param vertexID The Vertex-ID from the Vertex that should be removed from the Graph.

+ **removeEdge(edge : VisualEdge)**
Remove the given Edge from the Graph.
@param edge The Edge of the VisualGraph that should be removed.

+ **removeEdge(verticesIDs : List<Integer>)**
Remove the Edge between the given Vertrice.
@param verticesIDs The List of the Vertices-ID's that the Edge is between, that should be removed.

+ **duplicateVertex(vertexID : Integer)**
Duplicate the given Vertex so that a new Vertex is at the Graph with exactly the same neighbourhood.
@param vertexID The Vertex-ID of the Vertex that should be duplicated.

+ **contractVertices(verticesIDs : Integer)**
Contract the given Vertices to one Vertex.

Multiple Edges between the same destinations will be removed, so that only one of these Edges is in the Graph. Edge-Loops will be removed.
@param verticesIDs The List of the given VertricesID's.

+ **setVertexOrder(vertexID : Integer, order : Integer)**
Set the Vertex to the given Order.

The Vertex that was at this Position of the List earlier will be put behind the set Vertex.
@param vertexID The ID of the Vertex that should be moved to a different Order.
@param vertexID The order the Vertex should be set to.

Paket Sound

The Sound-Package contains everything that has to do with the Sounds. It separates the SoundHandler from the other parts.

Klasse SoundHandler

Beschreibung
The Sound Handler that manages the different Sounds the Program can make. Including the Error and finish Sound.

Dokumentation

- **soundList : List<String>**
  The List of all paths to the Audio-Files.

- **player : javafx.scene.media.MediaPlayer**
  The MediaPlayer that plays the given Music.

+ **SoundHandler()**
The Constructor of this Class. Has no parameters so it only sets the List to an Empty List so that the User can add File-paths to the playable Sounds later.

+ **SoundHandler(sounds : List<String>)**
The Constructor of this Class. The List of Strings should contain path to the Sound-Files the Player should play. The given List will be set at the soundList of this Class.
@param sounds The Path-List that the SoundHandler should use as soundList.

+ **addSound(filepath : String)**
Add a new Sound-Filepath to the soundList.
@param filepath The Filepath that should be added.

+ **playSound()**
Starts the MediaPlayer with a random Sound of the given List.

Therefore it calls the playSound(listPosition)-Method with an randomly choosen Value.

+ **playSound(listPosition : Integer)**
Starts the MediaPlayer with the Sound at the given position of the soundList of this Class.

Therefore it checs the given position if it is valid. Then it loads the File from the path that is stored at the soundList at the given Position. If the File could not be loaded the Method stops. Else the loaded File will be passed on to the MediaPlayer of this class. The MediaPlayer will be started, so that the Sound is played.
@param listPosition The position of the Sound at the soundList that should be played.

+ **stopSound()**
  Stops the playing of the MediaPlayer.

5 Controller

Dieser Abschnitt beschäftigt sich wie der Titel andeuten lässt mit dem Controller des Projektes. Dieser ist wiederum in zwei Hauptbestandteile unterteilt. Zum einen natürlich den üblichen Controller, zum anderen aber auch einem Graphic-Controller, der sich spezifisch mit dem Controlling der View beschäftigt.

5.1 Super-Controller

5.2 View-Controller

5.2.1 Allgemein

Der Graphic-Controller oder unter JavaFx üblicherweise auch FxController ist der Teil eines JavaFx-Programms der direkt mit dem von der FXML-Datei bereitgestellten GUI verknüpft ist. Der FxController ist somit ein separater Teil des Controllers, der sich lediglich mit der GUI beschäftigt und die getätigten Eingaben an die richtigen Stellen im allgemeinen Controller weitergibt. Dies bringt den Vorteil, dass der allgemeine Controller keine Kenntnisse über die GUI benötigt und losgelöst von dieser funktionieren kann. Dadurch ist auch die Modularität in diesem Teil des Entwurfs gewährleistet.

5.2.2 Entwurf

Paket ViewController

This Package contains the View Controller. It contains the FxController and the User-Input-Controller for the different GUIs. It works like a Fassade (Interface-like) between the actual Controller of the Program and the GUI.

Paket ViewController.UIController

The UI-Controller-Package contains the Input-Controller/Handlers. They catches the Mouse or Key Events and managing the things to perform the correct actions.

Paket ViewController.UIController.MouseController

The MouseController is the Part where the different Mouse-Events are catched, so that the Program can react to them.

Klasse MouseController

Beschreibung The MouseController so that the Program could detect Mouse Clicks, Drag and Drop. **Dokumentation**

- **xPos : Double**
  The X-Position of the Mouse at the time of the Event.

- **yPos : Double**
  The Y-Position of the Mouse at the time of the Event.
- **onMousePressedEventHandler : javafx.event.EventHandler<MouseHandler>**
  This is the MouseHandler for the onPressed-Action. This is an Implementation of the EventHandler<MouseHandler>, that overwrites the handle-Method.
- **onMouseDraggedEventHandler : javafx.event.EventHandler<MouseHandler>**
  This is the MouseHandler for the onDragged-Action. This is an Implementation of the EventHandler<MouseHandler>, that overwrites the handle-Method.
- **onMouseReleasedEventHandler : javafx.event.EventHandler<MouseHandler>**
  This is the MouseHandler for the onRelease-Action. This is an Implementation of the EventHandler<MouseHandler>, that overwrites the handle-Method.

+ **MouseController()**
The Empty-Constructor of this Class. The MouseHandler of this Class had to be set at a later time.

+ **MouseController(onMousePressedEventHandler : javafx.event.EventHandler<MouseHandler>, onMouseDraggedEventHandler : javafx.event.EventHandler<MouseHandler>, onMouseReleasedEventHandler : javafx.event.EventHandler<MouseHandler>)**
The Constructor of this Class. It gets the differentEventHandlers as Parameters and sets them.
@param onMousePressedEventHandler The MouseHandler for the onPressed-Action that should be set.
@param onMouseDraggedEventHandler The MouseHandler for the onDragged-Action that should be set.
@param onMouseReleasedEventHandler The MouseHandler for the onReleased-Action that should be set.

+ **setOnMousePressedEventHandler(onMousePressedEventHandler : javafx.event.EventHandler<MouseH**
Set's the onMousePressedEventHandler of the Class.
@param onMousePressedEventHandler The EventHandler for the onMousePressed-Event to set.

+ **setOnMouseDraggedEventHandler(onMouseDraggedEventHandler : javafx.event.EventHandler<Mouse**
Set's the onMouseDraggedEventHandler of the Class.
@param onMouseDraggedEventHandler The EventHandler for the onMouseDragged-Event to set.

+ **setOnMouseReleasedEventHandler(onMouseReleasedEventHandler : javafx.event.EventHandler<Mous**
Set's the onMouseReleasedEventHandler of the Class.
@param onMouseReleasedEventHandler The EventHandler for the onMouseReleased-Event to set.

Klasse MouseControllerGraphDrawer

Beschreibung This is a Child-Class of the MouseController. It is used for the Mouse-Controlling at the Graph-Drawer, to notify the modifying.

For Example: It registers when a Vertex is pressed so that it can be added to the selected Vertices. Or if the Vertex-Order should be changed so the User draggs the Vertex to the new Position. **Dokumentation**

- **graph : VisualGraph**
  The Graph the User wants to modify by using the Mouse.

+ **MouseControllerGraphDrawer(graph : VisualGraph)**
  This is the Constructor of this Class. It uses the Empty-SuperConstructor. It get's an Graph as a Parameter and sets them.
  @param graph The VisualGraph that should be set to modified with the Mouse.

+ **MouseControllerGraphDrawer(onMousePressedEventHandler : javafx.event.EventHandler<MouseHand onMouseDraggedEventHandler : javafx.event.EventHandler<MouseHandler>, onMouseReleasedEventHandler : javafx.event.EventHandler<MouseHandler>, graph : VisualGraph)**
  The Constructor of this Class. It gets the different EventHandlers as Parameters and sets them. Also it sets the given Graph.
  @param onMousePressedEventHandler The MouseHandler for the onPressed-Action that should be set.
  @param onMouseDraggedEventHandler The MouseHandler for the onDragged-Action that should be set.
  @param onMouseReleasedEventHandler The MouseHandler for the onReleased-Action that should be set.
  @param graph The VisualGraph that should be set to modified with the Mouse.

+ **setGraph(graph : VisualGraph)**
  Set the VisualGraph of this MouseControllerGraphDrawer.
  @param graph The VisualGraph that should be set.

Paket ViewController.UIController.KeyController

The KeyController is the Part where the different Keyboard-Events are catched, so that the Program can react to them.

Klasse KeyController

Beschreibung The Abstract KeyController that should be implemented at it's childs.

Because for our purpose it is not important if the KeyEvent was a KeyPress or a KeyRelease this Class does not contain other Handler that Handle these different Events. (unlike at the MouseController) Instead this Class is the only KeyEventHandler. **Dokumentation**

- **pressed : List<String>**
  The List of pressed Keys. This is used if multiple Keys are pressed. Then the Program puffers them in here.

  This List will be managed by this Class completely and does not need to be seen by outer Classes. Therefore there is not getter/setter-Method.

+ **handle(keyEvent : KeyEvent)**
  This is the actual Handler of the Key-Event wich is overwritten at the Child's to react to the different Events.
  @param keyEvent The actual Event that was triggered by the KeyBoard

Paket ViewController.FxController

This Package Fontains all the FxController for the Program. This is the Interface Between the actual FXML-Scene GUI and the Controller where the work will be passed on to the Model-Part.

Klasse FxController

Beschreibung This is the FxController. It is the abstract Class from which every other FxController of this Program gets passed on. **Dokumentation**

- **bundle : ResourceBundle**
  This is the Resource-Bundle of the Object. This is a kind of Key-Value storage-Unit.

  It contains the Text, ToolTip, ... of a specific Language for every GUI-Element that is identified by its fx-id.

  The ResourceBundle will be changed to the correct Language if needed.

- **languages : list<String>**
  A List of all possible Languages. This list will be dynamically filled with the correct String-ID's of the implemented Languages.

  This is needed for every controller-Class because of the ResourceBundle switching for different Languages. Every time the GUI-Language is switched a new ReourceBundle has to be loaded. Therefore the Program loads a new File from the Disk where the Language-ID will be the Ending-String of the Filename by which the program identifies the correct ResourceBundle for the wished Language.

+ **init()**
  Initializes the whole Controller by calling all necessary Methods.

  Remember: The Constructor is called first. Then all Fields annotated with @FXML will be loaded. After that the init-Method will run. That means that the Constructor does not have access to the FXML-GUI-Elements but the init-method does.

  This is an abstract Method so this will be Overwriten by the Childs. Then the Child can also overload this Method with different Parameters to give, and so on.

+ **changeLanguageTo(language : String)**
  Set the currently active Language of the GUI of this Controller to the given Language. Sets the correct ResourceBundle.
  @param language The Language that should be set.

Da es zu jedem FXML-Dokument und somit zu jeder GUI einen eingenen FxController gibt, benötigt man eine zentrale Einheit für das Programm, das als Manager für die einzelnen FxController fungiert. Diese Aufgabe übernimmt die Folgende Klasse FxRageController".

Klasse FxRageController

Beschreibung This is the FxController for the general Program-Window. It functions as the Manager of the calls of the other FxController. It is the only FxController that communicates with any SuperController and passes on the needed Values to the other FxController. **Dokumentation**

- **tpRage : javafx.scene.control.TabPane**
  The Main-TabPane of the Program.

- **fxGraphEditorController : FxGraphEditorController**
  The FxGraphEditorController of this Class. There should be only one active Editor at a time.

- **fxGraphGenerationController : FxGraphGenerationController**
  The GraphGenerationController of this Class. It is no list because there can only be one Generation-Tab per Rage-Program.

- **fxTabControllerList : List<FxTabController>**
  The List of all FxTabController that are needed. Remember: Every Super-Preview-Tab needs it's own FxTabController. Therefore there are multiple FxTabController possible.

- **fxMenuBarController : FxMenuBarController**
  The FxController for the MenuBar. This Attribute will be set at this Main-Controller and passed on to the other Tab-Controller where the MenuBar is needed as well.

- **superController : SuperController**
  This is the SuperController of the Program. It contains all other Controller. These other Controllers will be read out by this Class (the FxRageController) and then passed on to the Sup-FxController so that the whole Program can operate like it should.

+ **showMessage(messageText : String, messageType : EnumMessageBoxType)**
  Show a MessageBox at the Screen. This Method is Used at the ExceptionHandler to show the Error Log.
  @param messageText The text that should be shown at the MessageBox.
  @param messageType The Type of Message you want to Show. (Information, Warning, Error, ...)

+ **initFxGraphEditor()**
  This Method starts the GraphEditor by calling the FXMLLoader and passing the correct FxGraphEditorController.

+ **initNewTab()**
  This Method adds a new FxTabController to the List and by calling the FXMLLoader and then passing it to the FXML. This Tab will be added to the GUI.

  Also it calls the SuperController so that it can a new Tab for its own.

+ **initLanguages()**
  Is called at the init-Method.

  Calls the IOHandler to search for ResourceBundles. Adds all found Languages to the List so that the user can select them.

Nach dieser Betrachtung der oberen Struktur des View-Controllers benötigt man natürlich noch die einzelnen dort aufgeführten FxController. Diese werden im Anschluss beschrieben.

Klasse FxFilterController

Beschreibung This is the FxController for the Filter-Window. This is a separate Window that will pop up if the User wants to Filter the shown Graphs at the Preview-Tab. **Dokumentation**

- **filterController : FilterController**
  The FilterController for the passing of the User-Filter to the Model.

- **init()**
  Create the Needed GUI-Components. Therefore the FilterController is called to get the List of all Heuristics. Then for every Heuristic the ToggleButtonGroup to let the User select if the Heuristic should be correctly or not or if it does not matter will be added.

- **filter()**
  Parse the selected Filter into the correct form and give them to the FilterController by calling the

corresponding Method. Everything further will be done by the FilterController.

- **cancel()**
  Cancel the Filtration and close the Filter-Window.

+ **setFilterController(filterController : FilterController)**
  Set the FilterController of this Class.
  @param filterController The FilterController to set the attribute of this Class to.

Klasse FxHeuristicSettingsController

Beschreibung The FxController for the HeurisitcSetting-Screen. This is a separate Window that will pop up if the User wants to start a Heuristic that needs some Settings. The GUI-Elements are dynamically added using the HeuristicController to get the HeuristicProperties to read the possible settings the user can/have to take. **Dokumentation**

- **heuristicController : HeuristicController**
  This is the HeuristicController of this Class. It will be passed on from the FxTabController via the TabController that has the HeuristicController instance.
- **init()**
  Create the Needed GUI-Components. The Elements are created from the HeuristicProperties-List that it gets from the HeuristicController of this Class. These Elements are used so that the User can set his/her Properties of the Heuristic. These Values will be passed on the the HeuristicController so that the DataPool can be updated and the Model calculates the correct stuff.

- **updateSettings()**
  This Method calls the HeuristicController and calls the addToHeuristicsMethod with the set Properties, that are given by the dynamically added TextBoxes.

- **cancel()**
  Cancel the HeuristicSetting and close the Settings-Window.

+ **setHeuristicController(heuristicController : HeuristicController)**
  Set the HeuristiCController to the given.
  @param heuristicController The HeuristicController to set.

Klasse FxGraphEditorController

Beschreibung The FxController of the GraphEditor Window. **Dokumentation**

- **graphController : GraphController**
  This is the GraphEditorController of this Class.
- **graphDrawer : GraphDrawer**
  The GraphDrawer of this Editor. This contains the Graph that will be modified and drawn.
- **mouseController : MouseControllerGraphDrawer**
  This is the MouseController for this Editor. It will be created dynamically whenever the graph is changed and it was not jet created, so that an simple update of the graph via the Setter would be enough. So there is no need for a Setter-Method.

The Graph of the Drawer of this Class will be passed on as the Graph of this MouseController. Then the MouseEvent's can be managed as needed and the Vertices can be dragged.

The MouseHandler does not need to be accessed by any other Object so there is no need of Getter Methods.

- **cbGraphType : javafx.scene.control.ComboBox**
  This is the ComboBox where the GraphType will be set.

  The Value of this Box will be read-out and then the function of the Editor will be adapted to the Graph-Type.

  If the Editor is opened for an already existing Graph the ComboBox will be set to the type of the given Graph. If the Editor is opened for creating a new Graph the User has to set this ComboBox to the wished Type.

- **cmdCirlce : javafx.scene.control.Button**
  The Button for the AddCircle-Operation. This Buttons is needed because this Function has to be hidden if the Editor works on an VisualGraph of an SimpleHyperGraph.

- **cmdCirlce : javafx.scene.control.Button**
  The Button for the AddCircle-Operation. This Buttons is needed because this Function has to be hidden if the Editor works on an VisualGraph of an SimpleHyperGraph.

- **cmdClique : javafx.scene.control.Button**
  The Button for the addClique-Operation. This Buttons is needed because this Function has to be hidden if the Editor works on an VisualGraph of an SimpleHyperGraph.

- **cmdMergeVertices : javafx.scene.control.Button**
  The Button for the Merge-Vertices-Operation. This Buttons is needed because this Function has to be hidden if the Editor works on an VisualGraph of an SimpleHyperGraph.

- **cmdDuplicateVertices : javafx.scene.control.Button**
  The Button for the Duplicate-Vertices-Operation. This Buttons is needed because this Function has to be hidden if the Editor works on an VisualGraph of an SimpleHyperGraph.

+ **FxGraphEditorController()**
  This is the Empty-Constructor of this Class.

- **checkGraphType()**
  This Method is called at the init-Method and whenever a new GraphDrawer is set.

  Checks the Properties of the currently edited Graph and checks the Type.

  Then the corresponding ComboBox for the Graph-Type will be set to the correct Value.

  If the Editor works on an SimpleHyperGraph some Buttons will be hidden.

- **drag()**
  This Method is accessed if the cmdDrag-Button is pressed.

  Therefore the user wants to relocate an selected Node. Then this Method calls the corresponding Method at the GraphDrawer.

- **addEdge()**
  This Method is accessed if the cmdAddEdge-Button is pressed.

  Therefore the user wants to add an Edge between the selected Vertices. It checks the graph Type of the given Graph and if the Edge is between only two Vertices for an SimpleGraph. Then this Method calls the corresponding Method at the GraphDrawer.

- **vertriceAmountChange()**
  This Method is accessed if the VerticesAmount-Spinner changes it's Value.

  If this happens this can mean one of two different things the User wants to Delete or Add Vertices from the Graph. Therefore check if the Amount was changed to a smaller or bigger Value and then call the corresponding Method at the Graph from the GraphDrawer. In case the User wants to delete Vertices this is only possible while there are unconnected Nodes at the End of the List. If there are none left, the User has to delete the Vertices manually.

- **removeVertex()**
  This Method is accessed if the cmdEraser-Button is pressed.

  Therefore the user wants to delete the selected Vertex. Then this Method calls the corresponding Method at the GraphDrawer.

- **mergeVertices()**
  This Method is accessed if the cmdMergeVertices-Button is pressed.

  Therefore the user wants to merge the selected Vertices. Then this Method calls the corresponding Method at the GraphDrawer.

- **duplicateVertices()**
  This Method is accessed if the cmdDuplicateVertices-Button is pressed.

  Therefore the user wants to duplicate the selected Vertices. Then this Method calls the corresponding Method at the GraphDrawer.

- **deleteEdgesBetweenSelectedVertrices()**
  This Method is accessed if the cmdDeleteEdgesBetweenSelectedVertrices-Button is pressed.

  Therefore the user wants to delete the Edge between the selected Vertices. Then this Method calls the corresponding Method at the GraphDrawer.

- **addPath()**
  This Method is accessed if the cmdAddPath-Button is pressed.

  Therefore the user wants to add an Path between the selected Vertices. Then this Method calls the corresponding Method at the GraphDrawer.

- **addCircle()**
  This Method is accessed if the cmdAddCircle-Button is pressed.

  Therefore the user wants to add an Circle between the selected Vertices. Then this Method calls the corresponding Method at the GraphDrawer.

- **addClique()**
  This Method is accessed if the cmdAddClique-Button is pressed.

  Therefore the user wants to add an Clique. Then this Method calls the corresponding Method at the GraphDrawer.

- **undo()**
  This Method is accessed if the cmdUndo-Button is pressed.

  Therefore the user wants to revert the last done Change. Then this Method calls the corresponding Method at the GraphDrawer.

- **save()**
  This Method is accessed if the cmdSave-Button is pressed.

  Therefore the user wants to save the modified Graph. Then this Method calls the corresponding Method so that the VisualGraph of the Drawer can be converted to an Graph of the Model-Structure by the Adapter Class and be passed on to the Model by the general Controller.

- **cancel()**
  This Method is accessed if the cmdCancel-Button is pressed.

  Therefore the user wants to cancel the Modification of the Graph. So the Editor closes and the Modifications are deleted.

- **ok()**
  This Method is accessed if the cmdOk-Button is pressed.

  Therefore the user wants to save the modified Graph and go on.

+ **setGraphEditorController(graphEditorController : GraphEditorController)**
  Set the GraphEditorController.
  @param graphEditorController The GraphEditorController so set.

+ **setGraphDrawer(graphDrawer : GraphDrawer)**
  Set the GraphDrawer of this Class. Also calls the checkGraphType-Method.
  @param graphDrawer The GraphDrawer to set.

Paket ViewController.FxController.FxPreviewTabController

This Package orders the FxController a little bit so that the FxController for the Preview Tab are inside this Package.

Klasse FxTabController

Beschreibung This is the FxControler for the upper Preview-Tab. It manages the whole thing including the Sub Tabs like: Preview, DetailView and Statistics **Dokumentation**

- **tabController : TabController**
  This is the TabController which Contains the DataPool of this Tab as well as the other sub-Controller like Statistic, Heuristic and DetailView.
- **fxPreviewController : FxPreviewController**
  The FxController for this Preview-Tab.
- **detailViewList : List<FxDetailViewController>**
  A List of all FxDetailViews that are currently needed because these DetailViews are opened.

- **fxStatisticController : FxStatisticController**
  The FxStatisticController for this whole Preview-Tab.
- **fxRageController : FxRageController**
  This is the Upper-Controller. It is needed in this Class so that this Controller can access the needed Super-Controller parts. (Because the FxRageController is the only direct connection between the ViewController and the SuperController)

+ **FxTabController()**
The Empty-Constructor of this Class.

+ **setTabController(tabController : TabController)**
This Method sets the TabController of this Class. It is called from FxRageController that gets the needed TabController from the SuperController.
@param tabController The given TabController to set the Attribute to.

getFxRageController() : FxRageController
Get the FxRageController. This Method is protected. This is used so that the startEditor-Method can do what it should and starts the Editor.
@return returns The FxRageController of this Class.

Klasse FxPreviewController

Beschreibung This is the FxController for the lower Preview-Tab, where all Graphs are shown as a List of Tabs. **Dokumentation**

- **graphDrawer : GraphDrawer**
  The GraphDrawer of this Tab. The Graph of this Tab will be passed on to the Drawer whenever the User opens the DropDown-Box of an Graph, so it can be drawn.
- **filterController : FilterController**
  The FilterController for the passing of the Filter, selected by the User at the GUI via the FxFilterController, to the Model.
- **scrollPaneGraphList : javafx.scene.control.ScrollPane**
  The ScrollPane where all the DropDowns of the Graphs are in.

+ **FxPreviewController()**
The Empty-Constructor of this Class. It creates a new GraphDrawer, where the needed Graph will be put in if the User opens the Tab of a Graph.

+ **startGlobalHeuristics()**
This Method will start the calculation of the GlobalHeuristics. This will be done by calling the given TabController with the Method "heuristicApplyToDatapool()".

+ **stopGlobalHeuristics()**
This Method will start the calculation of the GlobalHeuristics. This will be done by calling the given DetailViewController.

+ **searchForPlugins()**
Try to add new Plugins to the List. This Method opens an FileChooser which will return a Folder that will be passed on to the IOController to perform the PlugIn-Loading.

+ **filter()**
This Method will show the Filter-Window, so that the User can select the Filters. This is done by Loading the Filter-FXML-File which creates the new Window. This already creates the needed FxFilterController which is read out, so that the FilterController at this Class can be set. Then the User-Input will be registers by this passes FxFilterController and will be passed on the given FilterController via the FxFilterController directly.

+ **setGraphDrawer(graphDrawer : graphDrawer)**
Set the Graph Drawer of this Class.
@param graphDrawer Set the Graph Drawer of this Class.

+ **setFilterController(filterController : FxFilterController)**
Set the FilterController of this Class.
@param graphDrawer The FilterController to set the attribute of this Class to.

Klasse FxDetailViewController

Beschreibung The FxController for the DetailView. **Dokumentation**

- **detailViewController : DetailViewController**
The DetailVewController of this Class. It Contains the Lists of all Local and Global Heuristics and the Graph of this DetailView.

- **fxTabController : FxTabController**
The FxTabController that contains this DetailView in the List of DetailViews. This is passed on by the FxTabController itself.

+ **FxDetailViewController()**
The Empty-Constructor of this Class.

+ **init(fxTabController : FxTabController)**
This method overrides the init-Method from the abstract-Class FxController. It sets the FxRageController by the given Parameter. This is possible because the FxRageController calls this Method and gives itselv as parameter with it.
@param fxTabController The FxTabController that will be set.

+ **searchForPlugins()**
Try to add new Plugins to the List. This Method opens an FileCooser which will return a Folder that will be passed on to the IOController to perform the PlugIn-Loading.

+ **startGlobalHeuristics()**
This Method will start the calculation of the GlobalHeuristics. This will be done by calling the given DetailViewController.

+ **stopGlobalHeuristics()**
This Method will stop the calculation of the GlobalHeuristics. This will be done by calling the given DetailViewController.

+ **startLocalHeuristics()**
This Method will start the calculation of the Local Heuristics for the Graph of this Detail View. This

will be done by calling the given DetailViewController.

+ **stopLocalHeuristics()**
  This Method will stop the calculation of the Local Heuristics for the Graph of this Detail View. This will be done by calling the given DetailViewController.

+ **stertStepByStepMode()**
  This Method will start the Step-By-Step-Mode of the Heuristic. This will be done by calling the given SuperDetailViewController.

+ **startGraphEditor()**
  Beschreibung

+ **setDetailViewController(detailViewController : DetailViewController)**
  Set the DetailViewController of this Class.
  @param detailViewController The DetailViewController to set.

+ **setGraphDrawer(graphDrawer : GraphDrawer)**
  Set the Graph Drawer of this Class.
  @param graphDrawer The GraphDrawer to set.

Klasse FxStatisticController

Beschreibung The FxConntroller for the Statistic-Tab.

Only fills in the Table with the Collected Data from the DataPool provided from the Model via the given HeristicController. **Dokumentation**

- **statisticController : StatisticController**
  This is the Controller that manages the DataPool which contains all the Statistics.

- **tableStatistics : javafx.scene.control.TableView**
  This is the Table view where the Statistics will be showed in.

+ **FxStatisticController()**
  The Empty-Constructor of this Class.

+ **updateStatistics()**
  This Method updates the Table-View with new Statistics, that should be shown to the User. It gets the Statistics from the HeuristicController.

+ **setStatisticController(statisticController : StatisticController)**
  This is the setter-Method for the StatisticController. It only sets the StatisticCotroller of this Class to the given Controller.

  This Method is called form the FxTabController which gets the StatisticController via the TabController.
  @param statisticController The StatisticController to set.

Paket ViewController.FxController.FxGraphGenerationController

This Package orders the FxController a little bit so that the FxController for the Graph-Generation Tab is inside this Package.

Klasse FxGraphGenerationController

Beschreibung The FxController for the Graph-Generation-Tab. **Dokumentation**

- **fxRageController : FxRageController**
  This is the FxRageController. This is needed so that the GraphGenerationTab can start an Editor by calling this Super-Class. It is passed on by the init-Method.
- **graphGeneratorController : GraphGeneratorController**
  The GraphGeneratorController for this Class.
- **graphProperties : GraphProperties**
  The GraphProperties of this Class. It among other things contains the differentGraphTypes that are implemented in this Program and the User can choose.

  This is only a copy that is get via the GeneratorController, so that the access to this Object is easier for the FxGraphGeneratorController and the GraphGenerationController is not flooded with requests. Therefore there is no need for a getter-Method.
- **cbGraphType : javafx.scene.control.ComboBox**
  The Combobox where the User can choose the Type of the generated Graph.

  The Possible Choices are given via ...??

  The other Settings will be based on this Choice, to provide the user with the correct Settings for the different Graph-Types.

+ **init(fxRageController : FxRageController)**
  This method overrides the init-Method from the abstract-Class FxController. It sets the FxRageController by the given Parameter. This is possible because the FxRageController calls this Method and gives itself as parameter with it.
  @param fxRageController The FxRageController that will be set.

- **updateGUIElementFromProperties()**
  This Method is called by the GraphProperties-Setter. It changes the GUI-Elements to the needed ones. They will be dynamically created.

- **startGeneration()**
  This Method starts the generation by calling the correct Method at the GraphGenerationController.

- **startImport()**
  This Method starts the import of an previously saved Tab by calling the correct Method at the GraphGenerationController.

- **startEditor()**
  This Method starts the GraphEditor by calling the correct Method at the GraphGenerationController. Also it init's the GraphEditor by calling the FxRageController.

+ **setGraphGeneratorController(graphGeneratorController : GraphGeneratorController)**
  Set the GraphGeneratorController of this Class. Then set the graphProperties of this Class from the

Value out of the GraphGenerationController. Then call the updateGUIElementFromProperties-Method to update the GUI to the given Properties.
@param graphGeneratorController The GraphGeneratorController to set.

+ **setGraphGeneratorController(graphProperties : GraphProperties)**
The Setter for the GraphProperties of this Class. It also calls the updateGUIElement-Method.
@param graphProperties The GraphProperties to set.

Paket ViewController.FxController.FxMenuBarController

This Package orders the FxController a little bit, so that the FxController for the MenuBar is inside this Package.

Klasse FxMenuBarController

Beschreibung This is the FxController for the MenuBar. **Dokumentation**

- **fxRageController : FxRageController**
  This is the FxRageController of this Class.

  This is needed for several Reasons: - So that the MenuBar can change some Properties of all Tabs,Windows,... (like the Language). - For the Tab-Switching. For example: at the showGraphGeneration-Method.

- **menuLanguages : javafx.scene.control.Menu**
  The Menu for the Language-Choosing.

  This is the Menu where where the MenuItems for the available Languages will be put into. By these added MenuItems the User can change the Language of the GUI.

  The MenuItems for the different Languages will be added dynamically. That's why this JavaFx-Element is an Object at this Controller. (unlike the others of this Scene).

- **toggleGroupLanguage : javafx.scene.control.ToggleGroup**
  The ToggleGroup for the different MenuItems for the Language-Choosing.

  The MenuItems for the different Languages will be added dynamically to the Menu. Therefore the dynamically generated Objects will be set into this ToggleGroup. So that the User only can select one of the available Languages to set the GUI to.

- **importHeuristics()**
  This Method calls the corresponding Controller to perform the Plugin-Import.

  Therefore a new FileChooser will be opened so that the User can choose the Folder where the Plugin is stored in.

- **importTab()**
  This Method calls the corresponding Controller to perform the Tab-Import.

  Therefore a new FileChooser will be opened so that the User can choose the Folder where the Graph is stored in.

- **exportTab()**
  This Method calls the corresponding Controller to perform the Tab-Export. Therefore a new FileChooser will be opened so that the User can choose the Folder where the Tab ahould be exported to.

- **showFunctionalHelp()**
  This Method opens the Help Dialogue for the Functional-Things of this Program.

  This Dialogue is shown as a new Window over the current Screen.

- **showUsageHelp()**
  This Method opens the Help Dialogue for the Usage-Things of this Program. It explains the GUI and the steps the User has to do to accomplish the different functional things the Program provides him/her with. This Dialogue is shown as a new Window over the current Screen.

- **showImportHelp()**
  This Method opens the Help Dialogue for the Import. It explains what can be Imported and how the user has to do it. This Dialogue is shown as a new Window over the current Screen.

- **setLanguage()**
  This Method sets the Language of the GUI. Therefore this class calls the changeLanguageMethod of the FxRageController so that all GUI-Elements that contain Text will be shown correct Language.

- **showAbout()**
  This Method opens the About Dialogue from this Program. This Dialogue is shown as a new Window over the current Screen.

- **showGraphGeneration()**
  This Method will switch the GUI to the GraphGeneration-Tab, so that the User can start set the properties and so on. Therefore uses the FxRageController.

- **showGraphEditor()**
  This Method will open the Graph-Editor, so that the User can start working on his/her own Graph. Therefore uses the FxGraphEditorController.

+ **setFxRageController(fxRageController : FxRageController)**
  Set the FxRageController. **@param fxRageController** The FxRageController to set.

6 Resources

Allgemein Die Ressourcen sind alle Dateien, die nicht in direktem Zusammenhang mit der Funktionalität und des Programms stehen und keinen Einfluss auf den Ablauf haben. Hierunter fallen meist Bilder, wie Icons, oder auch andere Mediendateien und vieles mehr. Diese Dateien muss unser Programm aus externen Stellen ziehen.

Entwurf Diese Daten werden getrennt vom Programmcode abgelegt und dann bei Bedarf aus der vordefinierten Stelle vom Programm eingeladen.

Paket Resources

This contains all the Resources that are needed for the Project.

* **FXML**
  This contains all the FXML files for the GUI. They are arranged in different Sub-Folders to separate.

    Main

      StartTab

      Preview

      GraphGeneration

    MenuBar

    Editor

    Popups

* **Pictrues**
  This contains all the Pictures used at the GUI organized by sub-Folders.

      Icons
  This contains all Icons for the Buttons, ... of the GUI.

      Logo
  This contains all Logos used at the GUI.

* **Sound**
  This contains all the Sounds that can be played by default.
* **StyleSheets**
  This Contains all the CSS-Files for the GUI.
* **Plugins**
  This Contains all the Plugins the User could add to the Rage-Program. By Default, there are the Plugins for the TC and EFL that we should implement.
* **Log**
  Contains the Log-Files.