

IT-Studienprojekt am Institut für Intelligente Informationssysteme (M.Sc.)

VISAB: Visualizing Agent Behaviour



Dokumentation

Jobst-Julius Bartels, 235499
Philipp Yasrebi-Soppa, 224949
Sebastian Viefhaus, 263430

Prüfer:
Prof. Dr. Klaus-Dieter Althoff
Dr. Pascal Reuss

Inhalt

1 Dokumentation.....	1
1.1 Architektur.....	1
1.1.1 Softwarearchitektur	1
1.1.2 File-Struktur.....	2
1.2 Programmcode.....	3
1.2.1 Controller.....	3
1.2.2 Views	19
1.2.3 CSS-Datei	20
1.2.4 Tabellen-Modelle.....	23
1.2.5 Util-Klasse	23
1.3 FPS-Shooter Schnittstelle	24
1.4 Datenbank	26
1.4.1 VISAB-Dateien	26
1.4.2 Externe Schnittstelle.....	27
1.5 Bedienung der Benutzeroberfläche	28
1.5.1 Perspektiven.....	28
1.5.2 Dialogbeispiele	29
1.6 Skalierbarkeit.....	33
1.6.1 StatisticsWindowController: Anpassungen.....	33
1.6.2 PathViewerWindowController: Anpassungen.....	33
1.6.3 Auswahlmenü mit Legende: Anpassungen	35
1.6.4 Sonstige Anpassungen.....	35

1 Dokumentation

Das vorliegende Programm ist das Ergebnis des IT-Studienprojekts der Universität Hildesheim von Jobst-Julius Bartels, Philipp Yasrebi-Soppa und Sebastian Viefhaus. Zur Entwicklung einer Benutzeroberfläche wurde das Framework JavaFX genutzt, welches Logik über Java-Code und Oberflächenelemente über FXML-Dateien implementiert. Dies kann durch Erstellung einer CSS-Datei und das Stylen aller genutzten Elemente abgerundet werden. Im Folgenden sollen Softwarearchitektur und Implementierung anhand von Code-Beispielen beleuchtet werden. Daraufhin wird die entstandene Oberfläche erläutert und dies anhand verschiedener Interaktionsbeispiele vertieft.

1.1 Architektur

Die Architektur des Projekts besteht im Wesentlichen aus der durch den Programmcode gegebenen Softwarearchitektur und der dabei implementierten Dateistruktur. Beides wird zunächst näher erläutert.

1.1.1 Softwarearchitektur

Bei der Softwarearchitektur wurde ein klassisches 3-Schichten-Modell verwendet, welches sich aus der Präsentationsschicht, der Logikschicht und der Datenhaltungsschicht zusammensetzt.

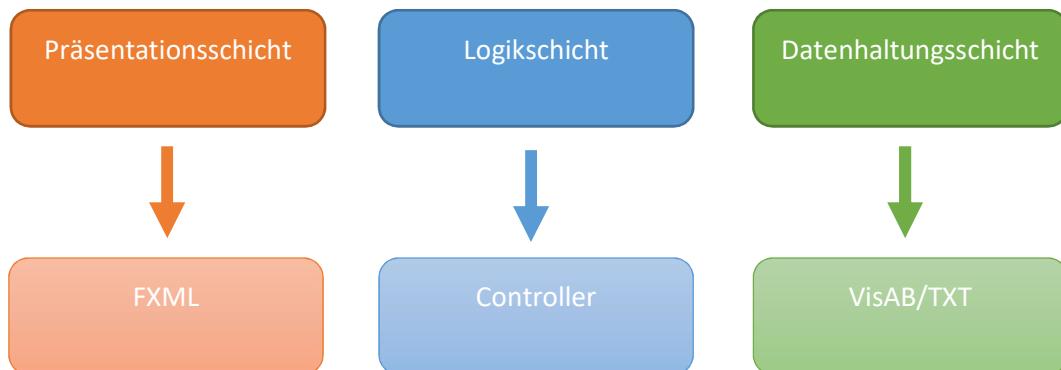


Abbildung 1: Schichtenmodell

Der Grafik zu entnehmen, wird die Präsentationsschicht durch FXML-Dateien implementiert. Dies bietet den Vorteil, die Gestaltung der Benutzeroberfläche strikt von der Logikschicht zu trennen. Außerdem kann zum Design der sogenannte Scene-Builder, ein Tool zur visuellen Erstellung von Oberflächen, genutzt werden. Dieser wird im weiteren Verlauf der Dokumentation noch näher beleuchtet. Die Logikschicht setzt sich aus verschiedenen Controllern zusammen, welche wiederum für die Behandlung der Benutzereingaben auf der Präsentationsschicht zuständig sind. Dabei werden die Ansichten der Präsentationsschicht jeweils mit einem Controller verknüpft. Die Logik wird durch

die Programmiersprache Java implementiert. Zuletzt wurde zur Vereinfachung für die Datenhaltungsschicht ein primitives Textformat verwendet. Hierbei ist es möglich, TXT-Dateien oder ein eigens definiertes Format, sogenannte VISAB-Dateien, zu nutzen. Letztere werden durch einen FPS-Shooter, der die Hauptschnittstelle für das Programm bildet, generiert.

1.1.2 File-Struktur

Die Erläuterung der File-Struktur beschränkt sich auf die für die Implementierung des Projekts relevanten Dateien. Alle weiteren Dateien sind beispielsweise dem genutzten Versionskontrollsystem und Standard-Bibliotheken beim Anlegen eines Java-Projekts geschuldet. Im Rahmen der Implementierung ergab sich folgende Struktur:

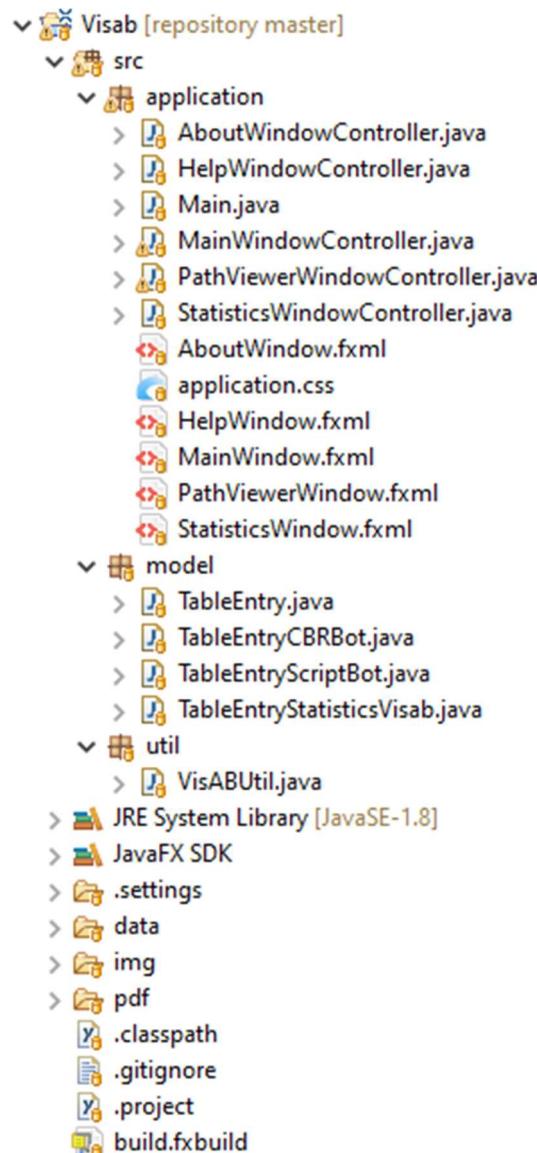


Abbildung 2: File-Struktur

Aus der Abbildung wird sichtbar, dass sich sämtliche Controller- und FXML-Dateien im Verzeichnis „applications“ befinden. Jede FXML-Datei referenziert dabei eine der Java-Dateien. Die Klasse

Main.java ist für das Laden der Controller und das Starten des Programms zuständig. Mit der Datei application.css werden einzelne Elemente der Benutzeroberfläche angesprochen und optisch verändert. Im Ordner „model“ befinden sich Klassen, die die Struktur der im Projekt genutzten Tabellen definieren und in „util“ ist eine Klasse mit statischen Methoden zur Vereinfachung diverser Vorgänge vorhanden. Letztlich wurden die Ordner „data“, „img“ und „pdf“ erstellt, die sowohl TXT- und VISAB-Dateien der Datenhaltungsschicht, als auch referenzierte Bilder und eine PDF-Datei mit dieser Dokumentation enthalten. Auf die Struktur der Datenbank wird im Folgenden noch näher eingegangen.

1.2 Programmcode

Bei der Beschreibung des Codes wird sich an der Reihenfolge der File-Struktur orientiert. Zunächst werden Controller, Views und deren Gestaltung durch die CSS-Datei beschrieben. Daraufhin werden die Klassen der Tabellen und die Util-Klasse beschrieben.

1.2.1 Controller

Bis auf die Main-Klasse wird jeder Controller immer von einer FXML-Datei referenziert. Im Folgenden werden einzelne Funktionen durch Code-Ausschnitte beleuchtet und textuell näher beschrieben.

1.2.1.1 Main

Die Klasse Main.java enthält eine Methode für jede View, in der das zu spawnende Fenster definiert wird und der entsprechende Controller geladen wird. In folgender Abbildung wird dies beispielhaft für das MainWindow sichtbar.

```
40⊕ public void mainWindow( ) {
41     try {
42         FXMLLoader loader = new FXMLLoader(Main.class.getResource("MainWindow.fxml"));
43         AnchorPane pane = loader.load();
44
45         primaryStage.setHeight(1000.00);
46         primaryStage.setWidth(1200.00);
47         primaryStage.getIcons().add((new Image("file:img/visabLogo.png")));
48         primaryStage.setTitle("VisAB");
49
50         MainWindowController mainWindowController = loader.getController();
51         mainWindowController.setMain(this);
52
53         Scene scene = new Scene(pane);
54         scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
55
56         primaryStage.setScene(scene);
57         primaryStage.show();
58
59     } catch (IOException e) {
60         e.printStackTrace();
61     }
62 }
63 }
```

Abbildung 3: Code-Snippet

Ab Zeile 42 wird zunächst die entsprechende FXML-Datei geladen, deren Fenster auf eine Höhe und Weite von mindestens 1000 bzw. 1200 Pixel festgelegt wird. Dies wird durch das Laden eines Logos für die Kopfzeile und einen Titel ergänzt. Des Weiteren wird der Controller geladen und in Zeile 54 der Pfad für die CSS-Datei zum Styling der auf der Seite befindlichen Elemente definiert. Letztlich werden mit den Methoden `setScene()` und `show()` alle Einstellungen übernommen und das Fenster generiert. Da sich die Funktionen für die anderen Controller kaum unterscheiden, wird hier auf eine genaue Beschreibung verzichtet.

Zur Verwendung der Datenbank wird innerhalb einer weiteren Methode das entsprechende Verzeichnis geladen und die Namen der sich darin befindenden Dateien als Liste zurückgegeben.

```

181⊕  private ObservableList<String> loadFilesFromDatabase(){
182      // Read database for Combobox
183      File folder = new File("data");
184      File[] listOfFiles = folder.listFiles();
185
186      ObservableList<String> filesComboBox = FXCollections.observableArrayList();
187
188      for (int i = 0; i < listOfFiles.length; i++) {
189          if (listOfFiles[i].isFile()) {
190              filesComboBox.add(listOfFiles[i].getName());
191          }
192      }
193
194      return filesComboBox;
}

```

Abbildung 4: Code-Snippet

Von Zeile 183 an wird der Name des Verzeichnisses definiert und dessen Inhalt aufgelistet. In der Schleife ab Zeile 188 wird das Ergebnis in der zuvor definierten Liste gespeichert. In diesem Fall war eine `ObservableList` notwendig, da der Inhalt für eine Combo Box verwendet wird, die ausschließlich diesen Datentyp akzeptiert.

Letztlich wird das Programm über eine von JavaFX vorgegebene Methode gestartet. Wie in folgender Abbildung sichtbar, wird in Zeile 35 lediglich das entsprechende Fenster angegeben.

```

32⊕  @Override
33  public void start(Stage primaryStage) {
34      this.primaryStage = primaryStage;
35      mainWindow();
36  }
37

```

Abbildung 5: Code-Snippet

1.2.1.2 AboutWindowController

Der Controller für das About-Fenster ist inhaltlich als am trivialsten zu bezeichnen, da er lediglich ein Menü zur Navigation auf alle anderen Ansichten und eine textuelle Darstellung, deren Beschreibung

später in der Dokumentation folgen wird, enthält. Dennoch bietet dieses Beispiel einen guten Einstieg in die Erklärung der allgemeinen Struktur der Controller.

```
8@   @FXML  
9    private MenuItem browseFileMenu;  
10@  @FXML  
11    private MenuItem pathViewerMenu;
```

Abbildung 6: Code-Snippet

Der AboutWindowController enthält, wie jeder andere Controller auch, Elemente der verknüpften View, deren Verbindung durch die Annotation @FXML gekennzeichnet werden.

```
25@  @FXML  
26    public void handleBrowseFileMenu() {  
27        main.mainWindow();  
28    }  
29  
30@  @FXML  
31    public void handlePathViewerMenu() {  
32        main.pathViewerWindow();  
33    }
```

Abbildung 7: Code-Snippet

Analog dazu gibt es Methoden, die die Logik für das Interagieren mit dem entsprechenden Element implementieren. Im Beispiel der beiden vorhergehenden Abbildungen wird bei der Interaktion mit einem MenuItem das entsprechende Fenster geladen.

```
19  public Main main;  
20  
21@  public void setMain(Main main) {  
22      this.main = main;  
23  }
```

Abbildung 8: Code-Snippet

Der Zugriff auf das Fenster findet über eine Setter-Funktion eines Klassenattributes statt, welche in der Main-Klasse aufgerufen wird.

1.2.1.3 HelpWindowController

Für die Logik des Hilfe-Fensters wurde der HelpWindowController implementiert. Dieser unterscheidet sich vom Controller des About-Fensters lediglich durch einen hinzugefügten Button, der diese Dokumentation öffnen soll.

```
33@  @FXML  
34    public void handleLoadButton() {  
35        File file = new File("../pdf/visab_documentation.pdf");  
36        HostServices hostServices = getHostServices();  
37        hostServices.showDocument(file.getAbsolutePath());  
38    }
```

Abbildung 9: Code-Snippet

In Zeile 35 wird eine Klasse File der java.io-Bibliothek in Abhängigkeit eines relativen Pfades, der auf die PDF-Datei der Dokumentation zeigt, erstellt. Letztlich wird das Dokument durch eine Methode der durch JavaFX implementierten HostServices geöffnet.

1.2.1.4 MainWindowController

Der Controller des Main-Fensters wird beim Start des Programms aufgerufen und bietet daher den Einstieg in alle vorhandenen Funktionen. Hierbei ist es möglich, eine Datei durch ein Popup des Betriebssystems auszuwählen und in der Datenbank zu speichern oder direkt auf die Ansicht „Path Viewer“ oder „Statistics“ zu wechseln. Letzteres wird durch das Klicken zweier Buttons ermöglicht, die jeweils das zugehörige Fenster aufrufen. Bei der Auswahl der abzuspeichernden Datei wurde folgende Fallunterscheidung implementiert.

```
45     FileChooser fileChooser = new FileChooser();
46     fileChooser.setTitle("Open Resource File");
47
48     file = fileChooser.showOpenDialog(null);
49
50     File folder = new File("data");
51     File[] listOfFiles = folder.listFiles();
52
53     // If file is selected
54     if (file != null) {
55
56         // Get Current Filename
57         Path currentFileName = Paths.get("", file.getName());
58
59         // Check if file exists
60         boolean fileExists = false;
61
62         for (int i = 0; i < listOfFiles.length; i++) {
63             if (listOfFiles[i].isFile()) {
64                 if (listOfFiles[i].getName().equals(currentFileName.toString())) {
65                     fileExists = true;
66                 }
67             }
68         }
69     }
70 }
```

Abbildung 10: Code-Snippet

Von Zeile 45 bis 48 wird zunächst das besagte Popup aufgerufen, um dem Nutzer die Auswahl der entsprechenden Datei zu ermöglichen. In Zeile 50 und 51 wird ein neues File instanziert, welches den Pfad darstellt, in dem die Datei gespeichert werden soll sowie alle bereits bestehenden Files aufgelistet, um diese in einem Array zu speichern. Ab Zeile 54 wird dann innerhalb der Schleife überprüft, ob die Datei bereits vorhanden ist. Ist dies der Fall, wird das Flag fileExists auf true gesetzt.

```

70     if (!fileExists) {
71
72         String loadedFilePath = file.getAbsolutePath();
73         String content = VisABUtil.readFile(loadedFilePath.toString());
74
75         boolean externalFileAccepted = false;
76         boolean visabFileAccepted = false;
77
78         if (currentFileName.toString().endsWith(".visab")) {
79             // show success message & write to Database
80             VisABUtil.writeFileToDatabase(currentFileName.toString(), content);
81
82             visabFileAccepted = true;
83
84         } else {
85             for (int i = 0; i < VisABUtil.getAcceptedExternalDataEndings().length; i++) {
86                 if (currentFileName.toString().endsWith(VisABUtil.getAcceptedExternalDataEndings()[i])) {
87                     externalFileAccepted = true;
88                 }
89             }
90         }

```

Abbildung 11: Code-Snippet

Sollte die Datei noch nicht existieren, wird ab Zeile 70 überprüft, ob diese den Vorgaben des Programmes entspricht. Hierfür wird von Zeile 78 bis 82 erfragt, ob das File die Dateiendung .visab besitzt. Sollte dies der Fall sein, wird es direkt in der Datenbank abgespeichert und ein entsprechendes Flag auf true gesetzt. Ab Zeile 84 wird getestet, ob die Dateiendung einem Eintrag eines Arrays entspricht, welches weitere akzeptierte Dateiformate enthält, woraufhin ebenfalls ein Flag gesetzt wird. Da im Rahmen der Implementierung zwischen internen und externen Dateien unterschieden wird, ist diese Differenzierung notwendig.

```

91     if (externalFileAccepted) {
92         VisABUtil.writeFileToDatabase(currentFileName.toString(), content);
93         warningMessage.setText("The file is not a visab-file!\nTherefore PathViewer won't be available, "
94             + file.getName() + " was saved anyway.");
95         warningMessage.setStyle("-fx-text-fill: orange;");
96     } else {
97         warningMessage.setText("This file ending is not accepted!\nThe following ending/s is/are accepted: "
98             + VisABUtil.getAcceptedExternalDataEndingsAsString() + ", .visab");
99         warningMessage.setStyle("-fx-text-fill: red;");
100    }
101    if (visabFileAccepted) {
102        warningMessage.setStyle("-fx-text-fill: green;");
103        warningMessage.setText(file.getName() + " successfully saved");
104    }

```

Abbildung 12: Code-Snippet

Sollte das Flag externalFileAccepted gesetzt sein, wird die Datei ebenfalls in der Datenbank gespeichert und eine Warnmeldung eines auf der Seite vorhandenen Labels gesetzt, wie von Zeile 92 bis 95 sichtbar wird. Für visabFileAccepted wird von Zeile 102 bis 103 eine Erfolgsmeldung ausgegeben. Sollte keines der beiden Flags gesetzt sein, erscheint eine Fehlermeldung mit Informationen darüber, dass die Datei nicht akzeptiert wurde und welche Dateiformate möglich sind.

```

106
107     } else {
108         warningMessage
109             .setText("File already exists! Therefore it was not saved.\nPlease change the file name.");
}

```

Abbildung 13: Code-Snippet

Sollte keine der bisherigen Bedingungen eingetreten sein, wird in Zeile 107 und 108 darüber informiert, dass die Datei bereits existiert und daher der Dateiname angepasst werden sollte.

1.2.1.5 StatisticsWindowController

Zur Darstellung der Statistiken wird ebenfalls wieder zwischen VISAB-Dateien und externen Dateien unterschieden. Zunächst wurde eine Methode implementiert, die beim Klicken eines Buttons zum Laden der Statistiken ausgeführt wird.

```
99@  @FXML  
100 public void handleLoadStatistics() {  
101     String fileNameFromComboBox = comboBox.getValue();  
102  
103     boolean externalFileAccepted = false;  
104  
105     // Read file  
106     Path filePath = Paths.get("", "data\\" + fileNameFromComboBox);  
107     String content = VisABUtil.readFile(filePath.toString());  
108  
109     if (fileNameFromComboBox == null) {  
110         // Set InfoLabel  
111         infoLabel.setText("Please select a file name first!");  
112     } else if (fileNameFromComboBox.endsWith(".visab")) {  
113         // If file is visab file  
114         try {  
115             loadVisabStatistics(content);  
116         } catch (Exception e) {  
117             infoLabel.setText("Visab file corrupted. Please check its content!");  
118         }  
119     }  
120  
121     } else {  
122         for (int i = 0; i < VisABUtil.getAcceptedExternalDataEndings().length; i++) {  
123             if (fileNameFromComboBox.endsWith(VisABUtil.getAcceptedExternalDataEndings()[i])) {  
124                 externalFileAccepted = true;  
125             }  
126         }  
127     }  
128  
129 }  
130 if (externalFileAccepted) {  
131     // If file is external  
132     loadExternalStatistics(content);  
133 }  
134 }  
135 }
```

Abbildung 14: Code-Snippet

Der aktive Dateiname wird in Zeile 101 einer Combobox entnommen, welche die Dateinamen der Datenbank beinhaltet. Ab Zeile 103 findet die beschriebene Fallunterscheidung statt, welche mit dem Lesen der Datei in Zeile 106 und 107 beginnt. Danach wird von Zeile 113 an überprüft, ob es sich um eine VISAB-Datei handelt. Wenn dies der Fall ist, wird eine Methode zum Laden der Inhalte aufgerufen. Wenn die Dateiendung als extern akzeptiert wird, wird eine andere Methode genutzt, wie in Zeile 132 sichtbar ist.

Bei Nutzung einer VISAB-Datei werden eine Tabelle mit Statistiken und zwei Balkendiagramme zur Darstellung der Häufigkeit genannter Pläne des CBR- und Script-Bots dargestellt.

```

137④    private void loadVisabStatistics(String content) {
138
139        // Plan Counters for Charts
140        int campCountCBR = 0;
141        int collectItemCountCBR = 0;
142        int moveToEnemyCountCBR = 0;
143        int reloadCountCBR = 0;
144        int seekCountCBR = 0;
145        int shootCountCBR = 0;
146        int switchWeaponCountCBR = 0;
147        int useCoverCountCBR = 0;
148
149        int campCountScript = 0;
150        int collectItemCountScript = 0;
151        int moveToEnemyCountScript = 0;
152        int reloadCountScript = 0;
153        int seekCountScript = 0;
154        int shootCountScript = 0;
155        int switchWeaponCountScript = 0;
156
157        List<Integer> calculatedCounters = createTableFromContentVisab(content, campCountCBR, collectItemCountCBR,
158                           moveToEnemyCountCBR, reloadCountCBR, seekCountCBR, shootCountCBR, switchWeaponCountCBR,
159                           useCoverCountCBR, campCountScript, collectItemCountScript, moveToEnemyCountScript, reloadCountScript,
160                           seekCountScript, shootCountScript, switchWeaponCountScript);
161
162        createPlanChartCBRBot(calculatedCounters);
163        createPlanChartScriptBot(calculatedCounters);
164
165        // Clear infoLabel
166        infoLabel.setText("Data successfully loaded!");
167        infoLabel.setStyle("-fx-text-fill: green;");
168    }

```

Abbildung 15: Code-Snippet

Die dazugehörigen Aufrufe sind von Zeile 157 bis 163 sichtbar. Die Funktion zur Erstellung der Tabelle bekommt dabei noch mit Null initialisierte Zählvariablen übergeben, um den Methoden das Ergebnis zur Darstellung der Balkendiagramme als Liste zu übergeben.

```

170④    private void loadExternalStatistics(String content) {
171
172        planChartCBRBot.getData().clear();
173        planChartScriptBot.getData().clear();
174
175        createTableFromContentExternal(content);
176
177        infoLabel.setText("No Visab file selected! Path Viewer Menu and Plan Chart is not available.");
178        infoLabel.setStyle("-fx-text-fill: orange;");
179    }

```

Abbildung 16: Code-Snippet

Sollte eine externe Datei ausgewählt worden sein, wird in Zeile 172 und 173 der Inhalt der Balkendiagramme gelöscht und in Zeile 175 eine separate Methode zur Erstellung der Statistik-Tabelle aufgerufen.

```

186@    private List<Integer> createTableFromContentVisab(String content, int campCount, int collectItemCount,
187        int moveToEnemyCount, int reloadCount, int seekCount, int shootCount, int switchWeaponCount,
188        int useCoverCount, int campCountScript, int collectItemCountScript, int moveToEnemyCountScript,
189        int reloadCountScript, int seekCountScript, int shootCountScript, int switchWeaponCountScript) {
190
191    createTableVisabStatistics();
192
193    List<Integer> counters = fillTableVisabStatistics(content, campCount, collectItemCount, moveToEnemyCount,
194        reloadCount, seekCount, shootCount, switchWeaponCount, useCoverCount, campCountScript,
195        collectItemCountScript, moveToEnemyCountScript, reloadCountScript, seekCountScript, shootCountScript,
196        switchWeaponCountScript);
197
198    return counters;

```

Abbildung 17: Code-Snippet

In der Methode zur Erstellung der Statistiken aus VISAB-Dateien werden von Zeile 191 bis 196 lediglich zwei weitere Funktionen aufgerufen, eine zur Erstellung der Tabellenstruktur, die andere zur Befüllung der Spalten.

```

202@    @SuppressWarnings("unchecked")
203    private void createTableFromContentExternal(String content) {
204
205        VisABUtil.clearTable(statisticsTable);
206        // Convert
207        List<List<String>> rawData = VisABUtil.convertStringToList(content);
208
209        // Create Table
210        @SuppressWarnings("rawtypes")
211        TableColumn col1 = new TableColumn("Name");
212        col1.setCellValueFactory(new PropertyValueFactory<>("Name"));
213        col1.prefWidthProperty().bind(statisticsTable.widthProperty().divide(2));
214
215        @SuppressWarnings("rawtypes")
216        TableColumn col2 = new TableColumn("Value");
217        col2.setCellValueFactory(new PropertyValueFactory<>("Value"));
218        col2.prefWidthProperty().bind(statisticsTable.widthProperty().divide(2));
219
220        statisticsTable.getColumns().addAll(col1, col2);
221
222        for (int i = 0; i < rawData.size(); i++) {
223            List<String> temp2 = rawData.get(i);
224            TableEntry tableEntry = new TableEntry();
225            for (int j = 0; j < temp2.size(); j += 2) {
226                tableEntry.setName(temp2.get(j));
227                tableEntry.setValue(temp2.get(j + 1));
228            }
229            statisticsTable.getItems().add(tableEntry);
230        }
231    }

```

Abbildung 18: Code-Snippet

Die Definition von Struktur und Inhalt der Tabelle mit externen Daten sind in obenstehender Abbildung sichtbar. Dabei werden in Zeile 210 bis 220 Tabellenspalten mit generischen Namen erstellt und einer Tabelle hinzugefügt. Letztlich wird in den darauffolgenden Zeilen über den Inhalt der Datei aus der Datenbank iteriert und für jede Zeile ein entsprechender Eintrag in der Tabelle gemacht.

Bezüglich der Erstellung von Spalten der durch VISAB-Dateien generierten Tabelle wird im Folgenden ein Ausschnitt sichtbar, der repräsentativ für alle weiteren Einträge ist.

```

233@    @SuppressWarnings("unchecked")
234    private void createTableVisabStatistics() {
235
236        VisABUtil.clearTable(statisticsTable);
237
238        // Create Table
239        @SuppressWarnings("rawtypes")
240        TableColumn frame = new TableColumn("frame");
241        frame.setCellValueFactory(new PropertyValueFactory<>("frame"));
242
243        @SuppressWarnings("rawtypes")
244        TableColumn coordinatesCBRBot = new TableColumn("coordinatesCBRBot");
245        coordinatesCBRBot.setCellValueFactory(new PropertyValueFactory<>("coordinatesCBRBot"));

```

Abbildung 19: Code-Snippet

Dabei ist von Zeile 239 bis 245 exemplarisch sichtbar, dass die Spalten mit entsprechender Benennung generiert werden. Schließlich werden der Tabelle wieder alle Spalten hinzugefügt, wie in folgender Abbildung sichtbar ist.

```

311    statisticsTable.getColumns().addAll(frame, coordinatesCBRBot, coordinatesScriptBot, healthCBRBot,
312                                         healthScriptBot, weaponCBRBot, weaponScriptBot, statisticsCBRBot, statisticScriptBot, nameCBRBot,
313                                         nameScriptBot, planCBRBot, weaponMagAmmuCBRBot, weaponMagAmmuScriptBot, healthPosition, weaponPosition,
314                                         ammuPosition, roundCounter);
315 }

```

Abbildung 20: Code-Snippet

Die Methode zur Befüllung der Tabelle beinhaltet zunächst Listen, die mit Werten aus den Dateien der Datenbank befüllt werden. Ein Beispiel kann der folgenden Abbildung entnommen werden.

```

317@    @SuppressWarnings("unchecked")
318    private List<Integer> fillTableVisabStatistics(String content, int campCountCBR, int collectItemCountCBR,
319                                                 int moveToEnemyCountCBR, int reloadCountCBR, int seekCountCBR, int shootCountCBR, int switchWeaponCountCBR,
320                                                 int useCoverCountCBR, int campCountScript, int collectItemCountScript, int moveToEnemyCountScript,
321                                                 int reloadCountScript, int seekCountScript, int shootCountScript, int switchWeaponCountScript) {
322
323        // Lists for Table
324        List<String> coordinatesCBRBotList = new ArrayList<String>();
325        List<String> coordinatesScriptBotList = new ArrayList<String>();
326
327        List<String> healthScriptBotList = new ArrayList<String>();
328        List<String> healthCBRBotList = new ArrayList<String>();

```

Abbildung 21: Code-Snippet

Beispielhaft aufgeführt, wird von Zeile 324 bis 328 für die Koordinaten der Bots und für Informationen über die Standorte der Lebenscontainer jeweils eine Liste initialisiert.

```

358     for (int i = 0; i < rawData.size(); i++) {
359         List<String> rawDataRow = rawData.get(i);
360
361         for (int j = 0; j < rawDataRow.size(); j += 2) {
362
363             // Coordinates
364             if (rawDataRow.get(j).contains("coordinatesCBRBot")) {
365                 String coordinatesCBRBot = rawDataRow.get(j + 1);
366                 coordinatesCBRBotList.add(coordinatesCBRBot);
367                 frameCount++;
368             }
369
370             if (rawDataRow.get(j).contains("coordinatesScriptBot")) {
371                 String coordinatesScriptBot = rawDataRow.get(j + 1);
372                 coordinatesScriptBotList.add(coordinatesScriptBot);
373             }
374
375             // Health
376             if (rawDataRow.get(j).contains("healthScriptBot")) {
377                 String healthScriptBot = rawDataRow.get(j + 1);
378                 healthScriptBotList.add(healthScriptBot);
379             }
380
381             if (rawDataRow.get(j).contains("healthCBRBot")) {
382                 String healthCBRBot = rawDataRow.get(j + 1);
383                 healthCBRBotList.add(healthCBRBot);
384             }

```

Abbildung 22: Code-Snippet

Diese Listen werden wiederum von Zeile 364 bis 384 mit Werten befüllt, wenn der Name des Eintrags den entsprechenden Wert enthält.

```

419     // Executed Plans of CBR-Bot & Script-Bot
420     if (rawDataRow.get(j).contains("planCBRBot")) {
421         String planCBRBot = rawDataRow.get(j + 1);
422         planCBRBotList.add(planCBRBot);
423
424         if (planCBRBot.contains("Camp")) {
425             campCountCBR++;
426         }

```

Abbildung 23: Code-Snippet

Sowohl für die Daten über die ausgeführten Pläne des CBR-Bots, als auch die des Script-Bots wird zusätzlich überprüft, welcher Plan ausgeführt wurde. Daraufhin wird die entsprechende Zählvariable, die der Methode als Parameter übergeben wurde, hochgezählt. Dies ist von Zeile 424 bis 426 exemplarisch erkennbar. Wie bereits zuvor beschrieben, werden alle Zählvariablen von der Methode als Liste zurückgegeben.

```

532     // set table entries
533     for (int k = 0; k < frameCount; k++) {
534         TableEntryStatisticsVisab tableEntryStatisticsVisab = new TableEntryStatisticsVisab();
535
536         tableEntryStatisticsVisab.setFrame(k);
537
538         tableEntryStatisticsVisab.setCoordinatesCBRBot(coordinatesCBRBotList.get(k));
539         tableEntryStatisticsVisab.setCoordinatesScriptBot(coordinatesScriptBotList.get(k));
540
541         tableEntryStatisticsVisab.setHealthCBRBot(healthCBRBotList.get(k));
542         tableEntryStatisticsVisab.setHealthScriptBot(healthScriptBotList.get(k));
543
544     statisticsTable.getItems().add(tableEntryStatisticsVisab);

```

Abbildung 24: Code-Snippet

Letztlich werden von Zeile 536 bis 542 die Tabelleneinträge gesetzt und in Zeile 564 der Tabelle hinzugefügt.

Bezüglich der Methoden zur Erstellung der Balkendiagramme werden im Folgenden lediglich die Inhalte für den CBR-Bot dargestellt, da sich der Programmcode beider Diagramme nur in der Benennung und Anzahl der Balken unterscheidet.

```

571@    @SuppressWarnings("unchecked")
572    private void createPlanChartCBRBot(List<Integer> calculatedCounters) {
573
574        planChartCBRBot.getData().clear();
575
576        xAxisPlanChartCBRBot.setCategories(FXCollections.observableArrayList(Arrays.asList("Camp",
577            "CollectItem", "MoveToEnemy", "Reload", "Seek", "Shoot", "SwitchWeapon", "UseCover")));
578        xAxisPlanChartCBRBot.setLabel("Plan");
579
580        yAxisPlanChartCBRBot.setLabel("Number of Executions");
581
582        planChartCBRBot.setTitle("Plan Chart CBR-Bot");
583
584        int sum = VisABUtil.sumIntegers(calculatedCounters.get(0), calculatedCounters.get(1), calculatedCounters.get(2),
585            calculatedCounters.get(3), calculatedCounters.get(4), calculatedCounters.get(5),
586            calculatedCounters.get(6), calculatedCounters.get(7));
587
588        XYChart.Series<String, Number> data = new XYChart.Series<>();
589        data.setName("Total Number of Executions " + sum);
590        data.getData().add(new XYChart.Data<>("Camp", calculatedCounters.get(0)));
591        data.getData().add(new XYChart.Data<>("CollectItem", calculatedCounters.get(1)));
592        data.getData().add(new XYChart.Data<>("MoveToEnemy", calculatedCounters.get(2)));
593        data.getData().add(new XYChart.Data<>("Reload", calculatedCounters.get(3)));
594        data.getData().add(new XYChart.Data<>("Seek", calculatedCounters.get(4)));
595        data.getData().add(new XYChart.Data<>("Shoot", calculatedCounters.get(5)));
596        data.getData().add(new XYChart.Data<>("SwitchWeapon", calculatedCounters.get(6)));
597        data.getData().add(new XYChart.Data<>("UseCover", calculatedCounters.get(7)));
598
599        planChartCBRBot.getData().addAll(data);
600    }

```

Abbildung 25: Code-Snippet

Von Zeile 574 bis 582 wird das Diagramm zunächst gelöscht, falls es zuvor schon befüllt wurde, um dann entsprechende Namen für die Kategorien, Labels für x- bzw. y-Achse und einen Titel festzulegen. Daraufhin werden in Zeile 584 bis 599 der Liste der zuvor berechneten Zählvariablen alle benötigten Werte entnommen, um diese dem Diagramm als Daten für die Länge der Balken zu übergeben.

1.2.1.6 PathViewerWindowController

Der PathViewerWindowController ist das Herzstück von VISAB und bietet die ausführliche Visualisierung sämtlicher Daten des FPS-Shooter. Die Abbildung 26 zeigt die entwickelte GUI mit ihren Funktionalitäten, welche an dieser Stelle durchnummeriert wurden.

Die Hauptkomponente des PathViewerWindowController ist die Spielkarte des FPS-Shooters, auf der alle Daten abgebildet werden. Die restlichen Funktionalitäten ergeben sich aus der Spielkarte und werden im folgenden anhand von Codebeispielen näher erläutert.

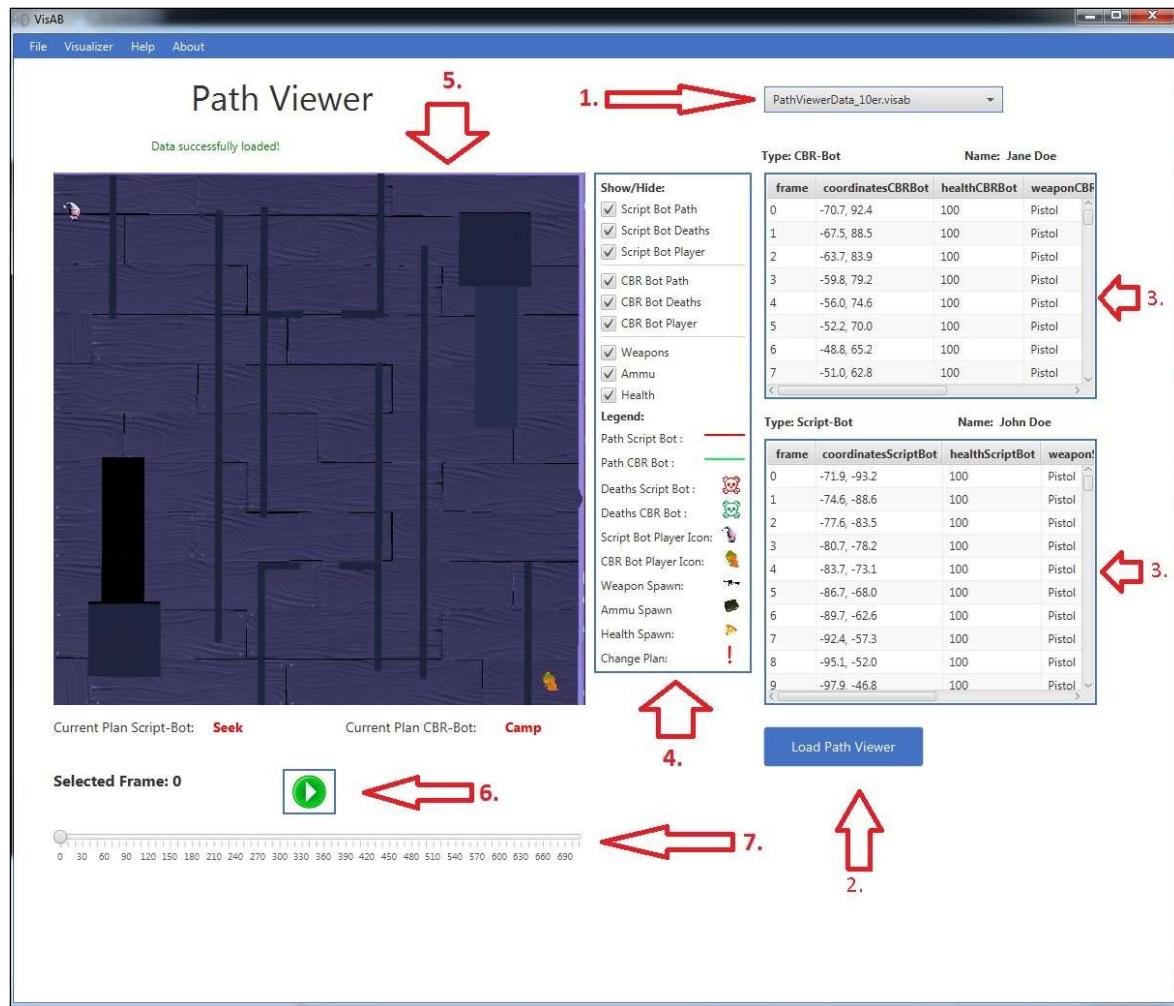


Abbildung 26: Klasse: GUI des Path Viewers

Zu Beginn jeder Sitzung muss eine entsprechende Visab-Datei geladen werden, welche zuvor in das System gespeichert wurde oder bereits im System hinterlegt ist (siehe 1. Punkt). Danach wird die ausgewählte Datei durch den "Load Path Viewer" Button in das System geladen und ausgeführt (siehe 2. Punkt). Sobald der Button gedrückt wurde, wird das System mit den Daten aus der Visab-Datei initialisiert und die Tabellen und die Spielkarte mit den Daten gefüllt. Im dritten Punkt sind jeweils die

Tabellen der Bots zu sehen. Eine Tabelle enthält die Daten des CBR-Bots und die Andere die des Skript-Bots. Dort werden für jeden einzelnen Frame des Spieldurchlaufs die aktuellen Werte der Parameter wie zum Beispiel Koordinaten, Name, Gesundheit oder momentane Waffe hinterlegt. Neben den Daten in den Tabellen wird durch das Betätigen des "Load Path Viewer" Button auch die Spielkarte (Punkt 5) mit den visualisierten Informationen befüllt. Dabei öffnet sich ein Menü mit Checkboxen und einer Legende (Punkt 4), welches das Ein- bzw. Ausblenden der relevanten Visualisierungen ermöglicht und die dargestellten Symbole und Inhalte definiert. Über die verschiedenen Checkboxen können die auf der Spielkarte dargestellten Informationen selektiert werden, um einzelne Aspekte hervorzuheben oder isoliert zu betrachten. Die Standardeinstellungen beim Start der Visualisierung sehen keine Selektion vor. Das heißt, auf der Spielkarte (Punkt 5) werden alle für den Spielkontext relevanten Daten dargestellt, sofern der Nutzer keine Auswahl über die Checkboxen vornimmt. Diese visualisierten Daten umfassen: Pfad des Scriptbots, Tode des Scriptbots, Spielfigur des Scriptbots, Pfad des CBR-Bots, Tode des CBR-Bots, Spielfigur des CBR-Bots, Waffen-Spawns, Ammu-Spawns und Healthcontainer-Spawns. Der sechste Punkt beinhaltet den "Play and Pause" Button. Sobald der Button gedrückt wird, startet der Path Viewer und die Daten werden Frame für Frame gezeichnet. Der Vorgang kann jederzeit pausiert und wieder aufgenommen werden. Während des Durchlaufens ist es möglich, die Geschwindigkeit des Abspielens zu variieren. Dieser Slider ist allerdings nur solange sichtbar, wie der "Play and Pause" Button aktiv ist. Sobald das Geschehen pausiert wird, verschwindet der Slider.

Die Abbildung 27 zeigt den Code, sobald der "Play and Pause" Button gedrückt wird. Zuerst wird in Zeile 638 bis 642 die Sichtbarkeit der entsprechenden Slider angepasst. Als nächstes wird das Zeichnen in einem eigenen Thread behandelt, damit die Zeichnungen simultan zur Laufzeit des Systems stattfinden können (siehe Zeile 644 bis 670).

```

631     // setOnAction method of the play and pause button
632     playPauseButton.setOnAction(new EventHandler<ActionEvent>() {
633         @Override
634         public void handle(ActionEvent event) {
635             if (playPauseButton.isSelected()) {
636                 //Sets visibility of UI components
637                 playPauseButton.setGraphic(pauseImageView);
638                 frameSlider.setVisible(false);
639                 veloLabel.setVisible(true);
640                 veloSlider.setVisible(true);
641
642                 // Starts a new Runnable task
643                 Runnable task = new Runnable()
644                 {
645                     public void run()
646                     {
647                         //Starts the frame loop and updates the frame label with the current frame position
648                         while(masterIndex < coordinatesScriptBotListPrep.size() / 2) {
649                             if(playPauseButton.isSelected()) {
650                                 showCoordinates.setDisable(true);
651                                 int i = masterIndex;
652                                 i++;
653                                 drawMap(coordinatesCBRBotListPrep,coordinatesScriptBotListPrep, statisticsCBRBotList, statisticsScriptBotList, i * 2, ammuP
654                                 masterIndex++;
655                                 Platform.runLater(new Runnable() {
656                                     @Override
657                                     public void run() {
658                                         int j = masterIndex;
659                                         j--;
660                                         frameLabel.setText("Selected Frame: " + j );
661                                     }
662                                 });
663                             } // Triggers sleep after each loop run
664                             try {
665                                 Thread.sleep(sleepTimer);
666                             } catch (InterruptedException e) {
667                                 System.out.println("First interrupted");
668                             }
669                         // Interrupts the loop if toggle button pressed again
670                         } else {
671                             break;
672                         }
673                     }
674                 });
675             }
676         }
677     });
678
679     //Updates UI componentens after hole loop
680     Platform.runLater(new Runnable() {
681         @Override
682         public void run() {
683             veloLabel.setVisible(false);
684             veloSlider.setVisible(false);
685             frameSlider.setValue(masterIndex);
686             playPauseButton.setGraphic(playImageView);
687             frameSlider.setVisible(true);
688             showCoordinates.setDisable(false);
689             veloSlider.setValue(0);
690         }
691     });
692
693     //Starts backgroundThread and activates daemon
694     Thread backgroundThread = new Thread(task);
695     backgroundThread.setDaemon(true);
696     backgroundThread.start();
697 }
698 });
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2697
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
275
```

Für den Slider der Geschwindigkeit wird der Wert des Sliders ausgelesen und die "sleepTimer" Variable entsprechend gesetzt. Je höher der Wert des Sliders ist, desto geringer ist der Wert des "sleepTimers" (siehe Abbildung 29).

```

611      //Listener of the velocity slider
612      veloSlider.valueProperty().addListener(new ChangeListener<Number>() {
613          @Override
614          public void changed(ObservableValue<? extends Number> observable, //
615              Number oldValue, Number newValue) {
616
617              // cases for different sleep values
618              if((int)Math.round(newValue.doubleValue()) == 0) {
619                  sleepTimer = 1000;
620              } else if ((int)Math.round(newValue.doubleValue()) == 2) {
621                  sleepTimer = 500;
622              } else if ((int)Math.round(newValue.doubleValue()) == 4){
623                  sleepTimer = 250;
624              } else if ((int)Math.round(newValue.doubleValue()) == 6) {
625                  sleepTimer = 125;
626              } else if ((int)Math.round(newValue.doubleValue()) == 8) {
627                  sleepTimer = 62;
628              }
629          }
630      });

```

Abbildung 29: Klasse: Velocity-Slider

Die Abbildung 30 zeigt einen Codeausschnitt aus der "drawMap()" Methode. Hier wird in Zeile 949-974 geprüft, ob das Zeichnen in einem eigenständigen Thread geschehen soll. Dies ist notwendig, damit die entsprechenden UI-Komponenten simultan zur Laufzeit aktualisiert werden.

```

947      /// Updates the drawPane in a thread or not
948      if(multithread) {
949          Platform.runLater(new Runnable()
950          {
951              @Override
952              public void run()
953              {
954                  int i = masterIndex;
955                  i--;
956
957                  // Clears current drawPane and adds new children
958                  drawPane.getChildren().clear();
959                  labelCurrentPlanScript.setText(planScriptBotList.get(i));
960                  labelCurrentCBR.setText(planCBRBotList.get(i));
961                  drawPane.getChildren().addAll(scriptDeathImageViews);
962                  drawPane.getChildren().addAll(cbrDeathImageViews);
963                  drawPane.getChildren().addAll(scriptPath, cbrPath,cbrbotImageView,scriptbotImageView,deathImageView, healthImage,deathImageViewCBR, ammuImage);
964
965              }
966          });
967      } else {
968          // Clears current drawPane and adds new children
969          drawPane.getChildren().clear();
970          labelCurrentPlanScript.setText(planScriptBotList.get(masterIndex));
971          labelCurrentCBR.setText(planCBRBotList.get(masterIndex));
972          drawPane.getChildren().addAll(scriptDeathImageViews);
973          drawPane.getChildren().addAll(cbrDeathImageViews);
974          drawPane.getChildren().addAll(scriptPath, cbrPath,cbrbotImageView,scriptbotImageView,deathImageView, healthImage,deathImageViewCBR, ammuImage, weapImage);
975
976      }
977  }

```

Abbildung 30: Klasse: drawMap()-Methode und Multithreadbehandlung

Der siebte und letzte Punkt behandelt den Frame-Slider. Der Frame-Slider ist sichtbar, solange der "Play and Pause" Button nicht aktiv ist und ist für das manuelle Wechseln zwischen den Frames verantwortlich. Die Abbildung 31 zeigt den "ChangeListener" des "FrameSlider". In Zeile 593 wird der Dezimalwert des Sliders in einen Integer gecastet. Danach wird in Zeile 602 bis 605 die "drawMap()"

Methode aufgerufen. Für den Methodenaufruf existieren an dieser Stelle zwei Möglichkeiten: die eines Initialstarts oder einer normalen Iteration.

```
585     frameSlider.setMax(coordinatesScriptBotListPrep.size() / 2 - 1);
586
587     //Listener of the Slider
588     frameSlider.valueProperty().addListener(new ChangeListener<Number>() {
589         @Override
590         public void changed(ObservableValue<? extends Number> observable, //
591             Number oldValue, Number newValue) {
592             //Initializes masterIndex
593             masterIndex = (int) Math.round(newValue.doubleValue());
594             // Sets text of the frame label
595             frameLabel.setText("Selected Frame: " + masterIndex);
596             if(statisticsScriptBotList.size() > masterIndex) {
597                 //calls drawMap method for first or i frame
598                 if(i == 0) {
599                     drawMap(coordinatesCBRBotListPrep,coordinatesScriptBotListPrep, statisticsCBRBotList, statisticsScriptBotList, 1, ammuPositionList, weap
600                 } else {
601                     i++;
602                     drawMap(coordinatesCBRBotListPrep,coordinatesScriptBotListPrep, statisticsCBRBotList, statisticsScriptBotList, i * 2, ammuPositionList,
603                 }
604             }
605         });
606     });
607
608
609
610
```

Abbildung 31: Klasse: Frame-Slider

1.2.2 Views

Für die Präsentationsschicht und die damit verbundenen Views wurde der sogenannte Scene Builder verwendet. Dadurch ist es möglich, Elemente der GUI visuell zu erstellen und ggf. per Drag and Drop zu verschieben und anzupassen. Schließlich werden daraus FXML-Dateien generiert, deren Inhalte durch die Controller angesprochen werden. In den beiden folgenden Abbildungen werden beispielhaft ein Ausschnitt des Scene Builders und die zugehörige FXML-Datei sichtbar.

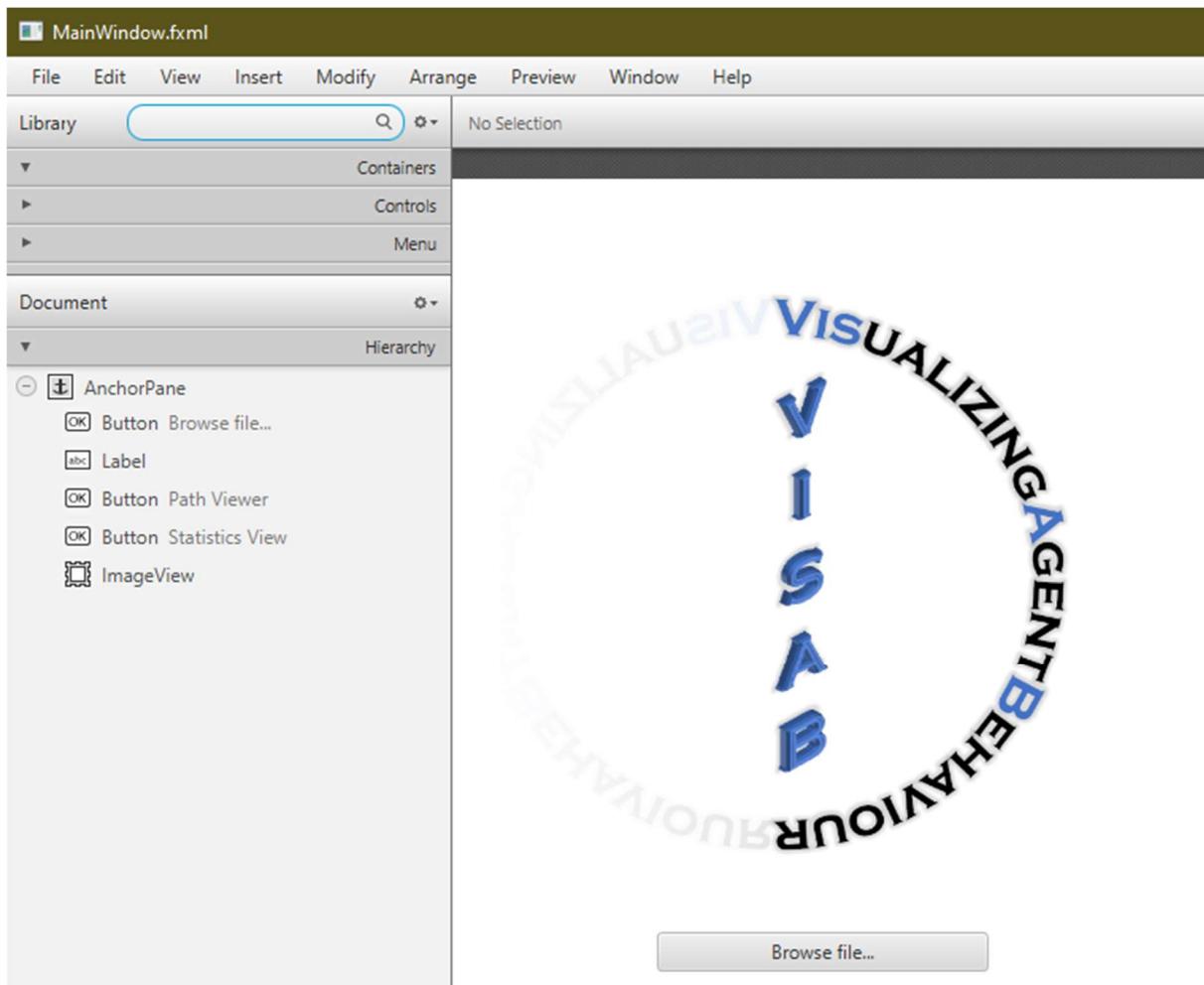


Abbildung 32: Scene Builder

Auf der linken Seite werden in der Hierarchy alle Elemente der aktiven Seite angezeigt. Durch Bedienung der oben links blau umrandeten Suche können weitere Komponenten gesucht und hinzugefügt werden. Am Beispiel des „Browse file...“-Buttons wird Folgendes in der zugehörigen FXML-Datei generiert:

```
14      <Button fx:id="browseFile" layoutX="490.0" layoutY="475.0" mnemonicParsing="false"
15          onAction="#handleBrowseFile" prefHeight="25.0" prefWidth="209.0" text="Browse file..." />
```

Abbildung 33: FXML-Ausschnitt

Um den Button im Controller zu verwenden, wird ihm eine ID und ein onAction-Parameter mit dem entsprechenden Methodennamen zugewiesen. Außerdem sind bereits Variablen zur Definition der Größe und Beschriftung des Buttons zu sehen. Auf diese Weise entsteht ein komfortabler Prozess zur Anpassung der Bedienelemente des Programms.

1.2.3 CSS-Datei

Nach der Implementierung aller relevanten Steuerungs- und Visualisierungselemente der GUI sowie der dazugehörigen Logik wurde für die Programmoberfläche ein ansprechendes und individuelles Design entwickelt. Dazu wurde, basierend auf dem entwickelten Logo, ein Farb- bzw. Designkonzept für die Benutzeroberfläche festgelegt. Dieses wurde mit Hilfe von CSS-Befehlen umgesetzt. Dabei wurden Designanpassungen mit geringer Komplexität oder für einzelne Objekte direkt im Scenebuilder implementiert. So wurde bspw. die Hintergrundfarbe aller Fenster per CSS-Befehl direkt im Scenebuilder für die betreffenden Objekte integriert. Hierzu wurde unter den „Properties“ eines Objektes im Abschnitt Style der CSS Befehl hinterlegt (siehe Abbildung 34).

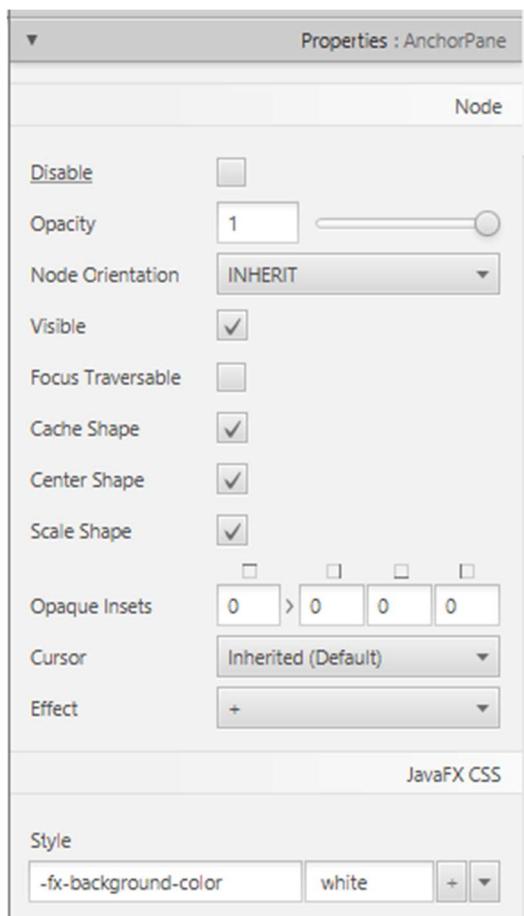


Abbildung 34: Scene Builder Properties

Designanpassungen mit höherer Komplexität oder für Objekttypen wurden im Gegensatz dazu im Quellcode des Java-Projektes per CSS-Befehl vorgenommen. Um die CSS-Befehle für die erstellte GUI

nutzen zu können, mussten diese zuerst integriert werden. Dazu wurde in der Main-Methode die application.css, welche die CSS-Befehle enthält, in die Scene hinzugefügt, sodass darauf zugegriffen werden kann (Zeile 131).

```

116⊕  public void aboutWindow( ) {
117      try {
118
119          FXMLLoader loader = new FXMLLoader(Main.class.getResource("AboutWindow.fxml"));
120          AnchorPane pane = loader.load();
121
122          primaryStage.setHeight(1000.00);
123          primaryStage.setWidth(1200.00);
124          primaryStage.getIcons().add((new Image("file:img/visabLogo.png")));
125          primaryStage.setTitle("VisAB");
126
127          AboutWindowController aboutWindowController = loader.getController();
128          aboutWindowController.setMain(this);
129
130          Scene scene = new Scene(pane);
131          scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
132

```

Abbildung 35: Code-Snippet

Die eigentlichen CSS-Befehle wurden dann in der application.css implementiert. Dabei wurden alle Buttons, welche in der GUI implementiert sind, über den Typelektor Button angepasst. Die Änderungen beziehen sich dabei auf Farbe, Schrift und Hover-Effekt der Buttons (siehe Abbildung 36). Das Farbschema ist dabei an das entwickelte Logo angepasst und weist unterschiedliche Intensitätsstufen der Grundfarbe auf.

```

2  /*** Styling Buttons ***/
3⊕ Button {
4      -fx-base: #4472C4;
5      -fx-background-color: derive(-fx-base, 0%);
6      -fx-color: black;
7      -fx-padding: 10px 28px;
8      -fx-font-size: 14px;
9      -fx-text-fill: white;
10 }
11
12⊕ Button:hover {
13     -fx-background-color: derive(-fx-base, -30%);
14 }

```

Abbildung 36: CSS-Datei

Bei der Anpassung der Menüleiste, welche am Rand der Fenster zu finden ist, wurden die Designanpassungen per Styleklasse vorgenommen. Dazu wurden, wie bei den Buttons, die Farbgebung, Schrift und die Effekte angepasst. Für ein einheitliches Erscheinungsbild wurde das gleiche Design- bzw. Farbschema wie bei den Buttons implementiert. Hier musste darauf geachtet werden, dass alle in der Menüleiste enthaltenen Elemente über eigene Styleklassen angesprochen werden, damit das angestrebte Gesamtdesign der Menüleiste korrekt umgesetzt wird (siehe Abbildung 37).

```

15  /*** Styling MenuBar ***/
16@ .menu-bar {
17      -fx-background-color: #4472C4;
18 }
19
20
21@ .menu-bar > .container > .menu-button {
22     -fx-background-color: #4472C4;
23 }
24
25@ .menu-bar > .container > .menu-button > .label {
26     -fx-text-fill: white;
27 }
28
29@ .menu-bar > .container > .menu-button > .label:disabled {
30     -fx-opacity: 1.0;
31 }
32
33@ .menu-bar > .container > .menu-button:hover,
34 .menu-bar > .container > .menu-button:focused,
35 .menu-bar > .container > .menu-button:showing {
36     -fx-background-color: derive(#4472C4, -30%);
37 }
38
39@ .menu-bar > .container > .menu-button:hover > .label,
40 .menu-bar > .container > .menu-button:focused > .label,
41 .menu-bar > .container > .menu-button:showing > .label {
42     -fx-text-fill: white;
43 }
44
45@ .menu-item {
46     -fx-background-color: #4472C4;
47 }
48
49@ .menu-item .label {
50     -fx-text-fill: white;
51 }
52
53@ .menu-item .label:disabled {
54     -fx-opacity: 1.0;
55 }
56
57@ .menu-item:focused, .menu-item:hovered {
58     -fx-background-color: derive(#4472C4, -30%);
59 }
60
61@ .menu-item:focused .label, .menu-item:hovered .label {
62     -fx-text-fill: white;
63 }
64@ .context-menu {
65     -fx-background-color: #4472C4;
66 }
67

```

Abbildung 37: CSS-Datei

1.2.4 Tabellen-Modelle

Zur Darstellung der Tabellen sind Klassen notwendig, die in ihrer Struktur jeweils einen einzelnen Eintrag repräsentieren. Hierbei werden die Werte der Einträge als Klassenattribute definiert, welche entweder durch den Konstruktor oder Setter initialisiert werden können. Dies ist notwendig, da die von JavaFX bereitgestellte TableView eine solche Struktur erwartet. Alle für das Projekt notwendigen Klassen befinden sich im model-Package. Letztlich wird dann für jeden Eintrag der Tabelle die Methode add() aufgerufen, die sich vorzugsweise innerhalb einer Schleife befindet und damit über eine Liste möglicher Tabelleneinträge iteriert. Ein Beispiel hierfür wurde bereits bei der Erklärung des StatisticsWindowControllers aufgeführt.

1.2.5 Util-Klasse

Im util-Package wird außerdem eine Klasse mit statischen Methoden bereitgestellt. Dabei sind folgende Funktionen vorhanden:

- acceptedExternalDataEndings: Array zur Definition akzeptierter Dateiendungen. Dies kann nach Belieben erweitert werden und beinhaltet im Auslieferungszustand lediglich die Endung .txt. Außerhalb der Klasse kann der Inhalt des Arrays über eine Getter-Methode als String abgefragt werden.
- readFile()-Methode: Gibt den Inhalt einer Datei als String wieder, deren Pfad ebenfalls als String angegeben wird.
- convertStringToList(): Konvertiert einen String in eine Liste aus zweidimensionalen Listen. Der String muss dabei den von VISAB gegebenen Konventionen entsprechen. Dies wird im weiteren Verlauf der Dokumentation noch näher erläutert.
- writeFileToDatabase(): Fügt der Datenbank eine durch Name und Inhalt definierte Datei hinzu.
- clearTable(): Löscht den Inhalt einer TableView aus JavaFX.
- sumIntegers(): Summiert eine unbestimmte Anzahl von Ganzzahlen. Dies wird bei der Erstellung der Balkendiagramme notwendig.

1.3 FPS-Shooter Schnittstelle

Die Schnittstelle innerhalb des FPS-Shooters wurde wie folgt umgesetzt. Innerhalb des Projekts befinden sich drei Klassen, welche hauptverantwortlich für den Datentransfer sind. In der Klasse "ConnectionToPathViewer" wird die TCP/IP Verbindung zu dem Java-Projekt hergestellt und geschlossen (siehe *Abbildung 38*).

```
15      * Diese Klasse stellt die Verbindung via TCP/IP zum Java Projekt her.
16      */
17  public class ConnectionToPathViewer
18  {
19
20      /**
21       * TCP-Client
22       */
23      private TcpClient mClient;
24
25      /**
26       * Data Stream.
27       */
28      private Stream mStream;
29
30      /**
31       * Diese Methode stellt konkret die Verbindung her.
32       */
33      private void InitiateConnection()
34      {
35          mClient = new TcpClient();
36          mClient.Connect(Constants.HOST_ADDRESS, 5558);
37          mStream = mClient.GetStream();
38      }
39
40      ~ConnectionToPathViewer()
41      {
42          Console.WriteLine("Connection closed");
43          CloseConnection();
44      }
45
46      /**
47       * Diese Methode schließt die Verbindung zwischen C# und Java.
48       */
49      private void CloseConnection()
50      {
51          if (mClient != null && mClient.Connected)
52          {
53              Console.WriteLine("Shutting down TCP/IP");
54              mClient.Close();
55          }
56      }
57  }
```

Abbildung 38: ConnectionToPathViewer

Als nächstes muss eine Objektklasse für die Parameter der Statistik existieren. Aus diesem Grund wurde die Klasse StatisticsForPathViewer erstellt. Diese Klasse entspricht den Vorgaben einer Standard Objektklasse und beinhaltet alle Getter und Setter für die Attribute, sowie den Konstruktor und eine `toString()` Methode (siehe *Abbildung 39*).

```

55
56     [DataMember]
57     public string ammuPosition { get; set; }
58
59     [DataMember]
60     public string roundCounter { get; set; }
61
62     /**
63      * Default-Konstruktor
64      */
65     public StatisticsForPathViewer() : this("", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "")
66     {
67     }
68
69     /**
70      * Konstruktor, der sämtliche Daten der Statistik erwartet.
71      */
72     public StatisticsForPathViewer(string coordinatesCBRBot, string coordinatesScriptBot, string healthCBRBot, string healt
73     {
74         this.coordinatesCBRBot = coordinatesCBRBot;
75         this.coordinatesScriptBot = coordinatesScriptBot;
76         this.healthCBRBot = healthCBRBot;
77         this.healthScriptBot = healthScriptBot;
78         this.weaponScriptBot = weaponScriptBot;
79         this.weaponCBRBot = weaponCBRBot;
80         this.statisticCBRBot = statisticCBRBot;
81         this.statisticScriptBot = statisticScriptBot;
82         this.nameCBRBot = nameCBRBot;
83         this.nameScriptBot = nameScriptBot;
84         this.planCBRBot = planCBRBot;
85         this.weaponMagAmmuCBRBot = weaponMagAmmuCBRBot;
86         this.weaponMagAmmuScriptBot = weaponMagAmmuScriptBot;
87         this.healthPosition = healthPosition;
88         this.ammuPosition = ammuPosition;
89         this.weaponPosition = weaponPosition;
90         this.roundCounter = roundCounter;
91         this.planScriptBot = planScriptBot;
92     }
93     /**
94      * ToString Methode der Klasse StatisticsForPathViewer
95      */
96     public override string ToString()
97     {
98         return "StatisticsForPathViewer [coordinatesCBRBot=" + coordinatesCBRBot + "]"
99     }

```

Abbildung 39: StatisticsForPathViewer

Zuletzt müssen die Attribute des Statistics-Objekts noch instanziert und initialisiert werden. Dies geschieht in der bereits vorhandenen Klasse "BotCBRBehaviourScript".

Dort werden allen Attributen entsprechende Werte zugewiesen. Für manche Attribute mussten extra Variablen innerhalb des Unity-Projekts angelegt werden, um die Daten zu erhalten. Zum Beispiel wurde ein Rundenzähler eingebaut, der nach Ablauen der Zeit oder eines Abschusses den Integer-Wert hochzählt. Für andere Attribute wie Gesundheit, Name oder Koordinaten der Spieler konnte auf die Getter-Methoden der Spielerklasse zugegriffen werden. In der Abbildung 40 ist ausschnittsweise die Zuweisung zu entnehmen.

```

196     // Aktuelle Rundenanzahl des Spiels.
197     String roundCounter = GameControllerScript.roundCounter.ToString();
198
199     // Aktuelle Koordinate der Spieler.
200     String cbrBotCoords = mPlayerWithCBR.GetPlayerPosition().ToString();
201     String scriptBotCoords = mEnemy.GetPlayerPosition().ToString();
202
203     // Aktuelle Gesundheit der Spieler.
204     String cbrBotHealth = mPlayerWithCBR.mPlayerHealth.ToString();
205     String scriptBotHealth = mEnemy.mPlayerHealth.ToString();
206
207     // Aktuelle Waffe der Spieler.
208     String cbrBotWeapon = mPlayerWithCBR.mEquippedWeapon.mName;
209     String scriptBotWeapon = mEnemy.mEquippedWeapon.mName;
210
211     // Aktuelle Munition der Spieler.
212     String cbrBotAmmu = mPlayerWithCBR.mEquippedWeapon.mCurrentMagazineAmmu.ToString();
213     String scriptBotAmmu = mEnemy.mEquippedWeapon.mCurrentMagazineAmmu.ToString();
214
215     // Aktuelle Statistik der Spieler
216     String cbrBotStatistic = mPlayerWithCBR.mStatistics.ToString();
217     String scriptBotStatistic = mEnemy.mStatistics.ToString();
218
219     // Aktueller Name der Spieler
220     String cbrBotName = mPlayerWithCBR.mName;
221     String scriptBotName = mEnemy.mName;
222
223     // Aktueller Plan der Spieler
224     String cbrBotPlan = mPlayerWithCBR.mPlan.actionsAsString;
225     String scriptBotPlan = BotBehaviourScript.ScriptBotPlan.ToString();
226
227     // Aktuelle Position der Items
228
229     // Gesundheit
230     String healthPosition = GameControllerScript.healthPositionRaw.ToString();
231     if(healthPosition.Equals("(0,9, 24,2, -145,8)"))
232     {
233         healthPosition = " healthSpawnPointA";
234     }
235     else if (healthPosition.Equals("(-2,8, 24,2, 8,1)"))
236     {
237         healthPosition = " healthSpawnPointB";
238     }

```

Abbildung 40: BotCBRBehaviourScript

Sämtliche Daten werden dann an das Java-Projekt geschickt und dort in der richtigen Formatierung für das VISAB Programm in einer Textdatei gespeichert. Diese Textdatei kann dem "CBRS" Ordner des FPS-Shooters entnommen werden und trägt die Bezeichnung "PathViewerData.visab".

1.4 Datenbank

Wie bereits beschrieben, besteht die Datenbank aus Dateien verschiedener Formate. Einerseits ist es möglich, VISAB-Dateien zu verwenden, andererseits werden auch externe Dateiformate akzeptiert. Beide Möglichkeiten werden im Folgenden näher erläutert.

1.4.1 VISAB-Dateien

Das VISAB-Format wird von einem FPS-Shooter bereitgestellt und enthält 18 verschiedene Attribute, die sich für jeden Frame des Spiels verändern können:

- coordinatesCBRBot: Koordinaten des CBR-Bots
- coordinatesScriptBot: Koordinaten des Script-Bots
- healthScriptBot: Leben des Script-Bots
- healthCBRBot: Leben des CBR-Bots
- weaponScriptBot: aktuelle Waffe des Script-Bots

- weaponCBRBot: aktuelle Waffe des CBR-Bots
- statisticScriptBot: Kills & Tode des Script-Bots
- statisticCBRBot: Kills & Tode des CBR-Bots
- nameScriptBot: Name des Script-Bots
- nameCBRBot: Name des CBR-Bots
- planCBRBot: ausgeführter Plan des CBR-Bots
- weaponMagAmmuCBRBot: aktueller Munitionsstand der getragenen Waffe des CBR-Bots
- weaponMagAmmuScriptBot: aktueller Munitionsstand der getragenen Waffe des Script-Bots
- healthPosition: Position des Lebenscontainers
- weaponPosition: Position der Waffe
- ammuPosition: Position der Munitionskiste
- roundCounter: Zähler der aktuellen Runde
- planScriptBot: ausgeführter Plan des Script-Bots

1.4.2 Externe Schnittstelle

Prinzipiell ist es möglich, auch externe Datenquellen zu nutzen. Dabei kann eine Datei genutzt werden, welche Attribute und entsprechende Werte enthält. Jedes Attribut-Wert-Paar muss durch ein Gleichheitszeichen getrennt werden und mit eckigen Klammern umrandet werden. Ein Beispiel dafür wären folgende Einträge:

- [Name = DonaldDuck]
- [City = Entenhausen]

Wenn sich diese Einträge in einem von VISAB akzeptierten Dateiformat befinden, können sie als Tabelle innerhalb des StatisticsWindows dargestellt werden. Die Verwendung des Path Viewers ist nicht verfügbar.

1.5 Bedienung der Benutzeroberfläche

Die Erklärung der Benutzeroberfläche besteht aus einer Übersicht aller möglichen Views und einzelner Dialogbeispiele mit verschiedenen Szenarios, in denen sich ein Benutzer befinden kann.

1.5.1 Perspektiven

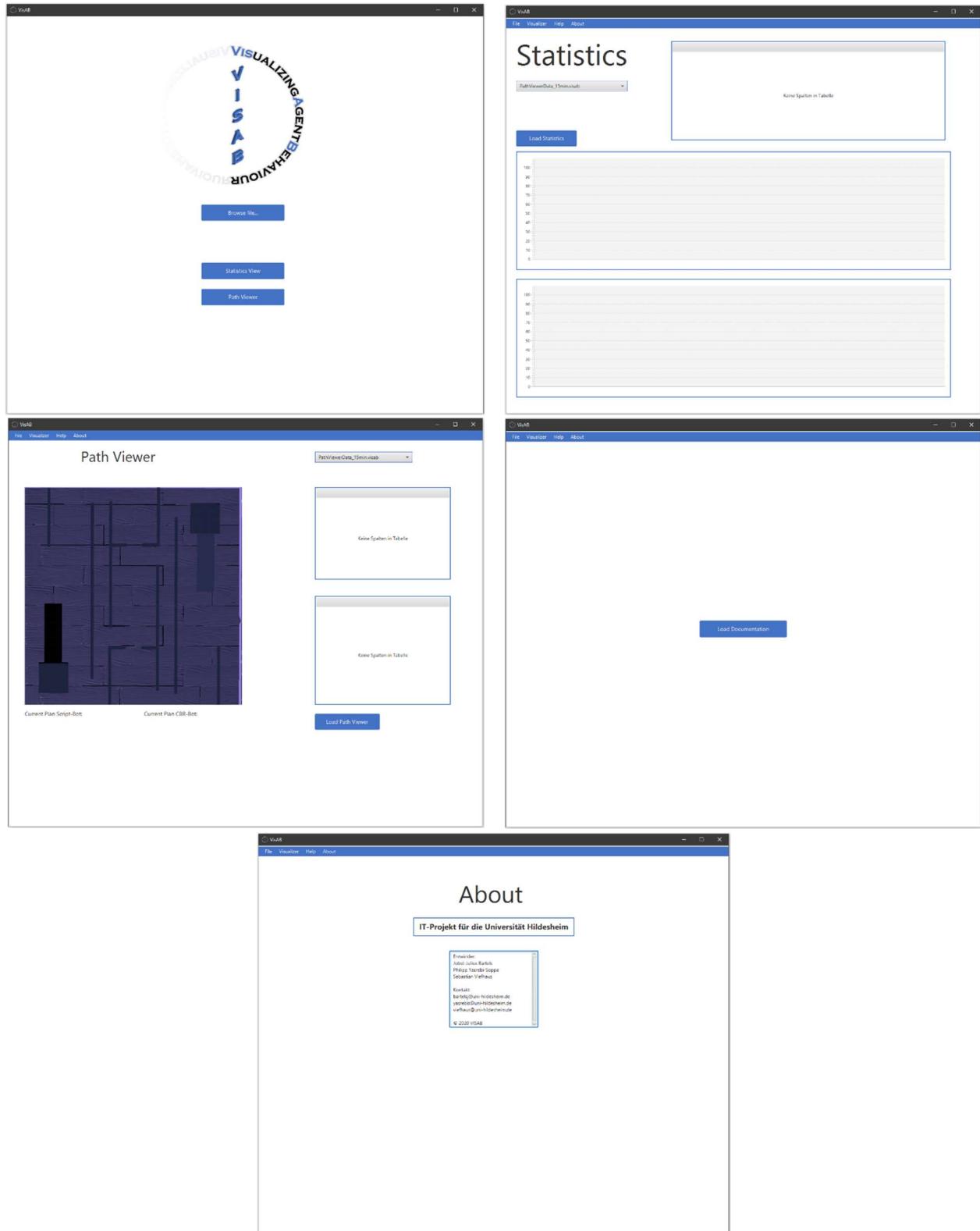


Abbildung 41: Perspektiven (1-5)

Analog zur Anzahl der beschriebenen FXML-Dateien gibt es fünf verschiedene Ansichten des Programms. Beginnend bei der oberen linken Abbildung, zeigt das Programm eine Startseite mit folgenden Interaktionsmöglichkeiten an:

- Browse file: Hochladen einer Datei in die Datenbank
- Statistics View: Wechsel auf die Ansicht zur Darstellung der Statistiken
- Path Viewer: Wechsel auf die Ansicht zur Darstellung des Bot-Verhaltens

Die Statistics View, welche oben rechts sichtbar ist, enthält eine Combobox zur Auswahl einer Datei der Datenbank und einen Button zum Laden der Datei. Außerdem sind eine Tabelle zur Darstellung der Attribute und zwei Diagramme zur Visualisierung der Anzahl ausgeführter Pläne der Bots vorhanden.

Mittig links wird der Path Viewer dargestellt, der erneut eine Combobox und einen Button zum Laden der ausgewählten Datei beinhaltet. Des Weiteren gibt es zwei Tabellen, jeweils für die Darstellung von Attributen des CBR- bzw. des Script-Bots und einer Übersicht der Spielkarte des FPS-Shooters. In Letzterer werden zur Laufzeit die Pfade der Bots und Gegenstände des Spiels eingezeichnet.

In den beiden letzten Ansichten, einem Help- und einem About-Fenster, ist es möglich, diese Dokumentation zu öffnen bzw. Informationen über die Ersteller des Projekts zu erfahren.

Außer der Hauptseite haben alle Ansichten auf der oberen Seite eine Menüleiste, um komfortabel zwischen den Seiten wechseln zu können.

1.5.2 Dialogbeispiele

Im Folgenden werden zwei verschiedene Dialogbeispiele bei der Nutzung des Programms näher beleuchtet. Hierbei werden zum einen das Laden einer VISAB-Datei, zum anderen die Nutzung einer externen Datequelle beschrieben und die damit einhergehenden Interaktionsmöglichkeiten aufgeführt.

1.5.2.1 Speichern & Laden einer VISAB-Datei

Nach Start des Programms wird zunächst der Button „Browse file“ geklickt, der ein Fenster des Betriebssystems zur Auswahl einer Datei öffnet. Daraufhin wird ein VISAB-konformes Dateiformat selektiert und der Benutzer erhält eine Erfolgsmeldung.

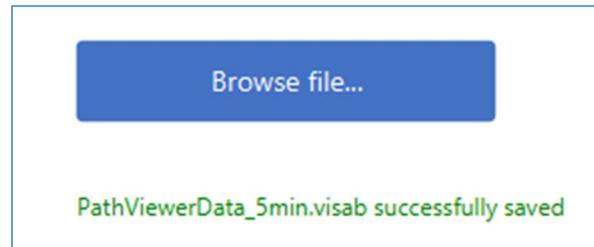


Abbildung 42: Browse File

Um auf die Ansicht zur Darstellung der Statistiken zu wechseln, wird der entsprechende Button geklickt. Dort angekommen, wird die zuvor gespeicherte Datei innerhalb der Combobox ausgewählt und der Button „Load Statistics“ angeklickt.

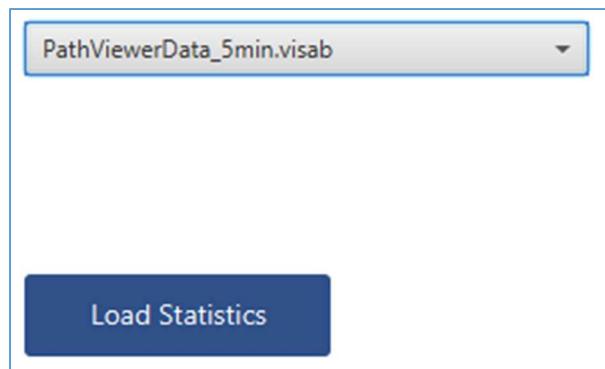


Abbildung 43: Load Statistics

Dadurch werden alle Informationen, die der Datei entnommen wurden, innerhalb der Tabelle dargestellt und eine Übersicht über die Anzahl ausgeführter Pläne innerhalb zweier Balkendiagramme geboten.

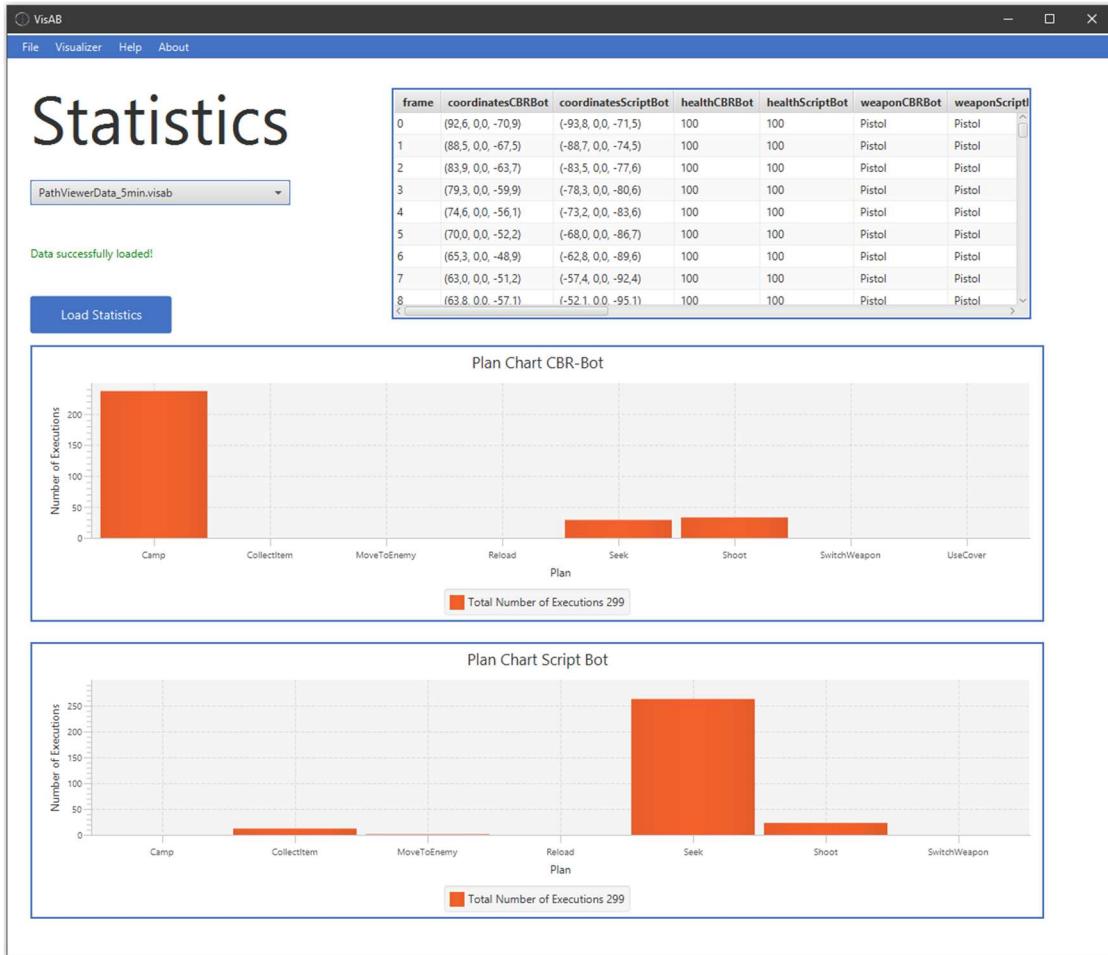


Abbildung 44: Statistics View

Daraufhin wählt der Benutzer den Menüpunkt Visualizer\Path Viewer aus der Menüleiste aus und wird auf die PathViewer-Ansicht weitergeleitet. Hier wird erneut das entsprechende File in der Combobox angewählt und der Button zum Laden des Path Viewers angeklickt. Jetzt werden die Daten der VISAB-Datei gefiltert und tabellarisch für beide Bots getrennt dargestellt. Außerdem kann der Benutzer durch Klicken des grünen Play-Buttons den Ablauf des Spiels visuell darstellen lassen und diese Darstellung durch An- bzw. Abwählen der Checkboxen modifizieren.

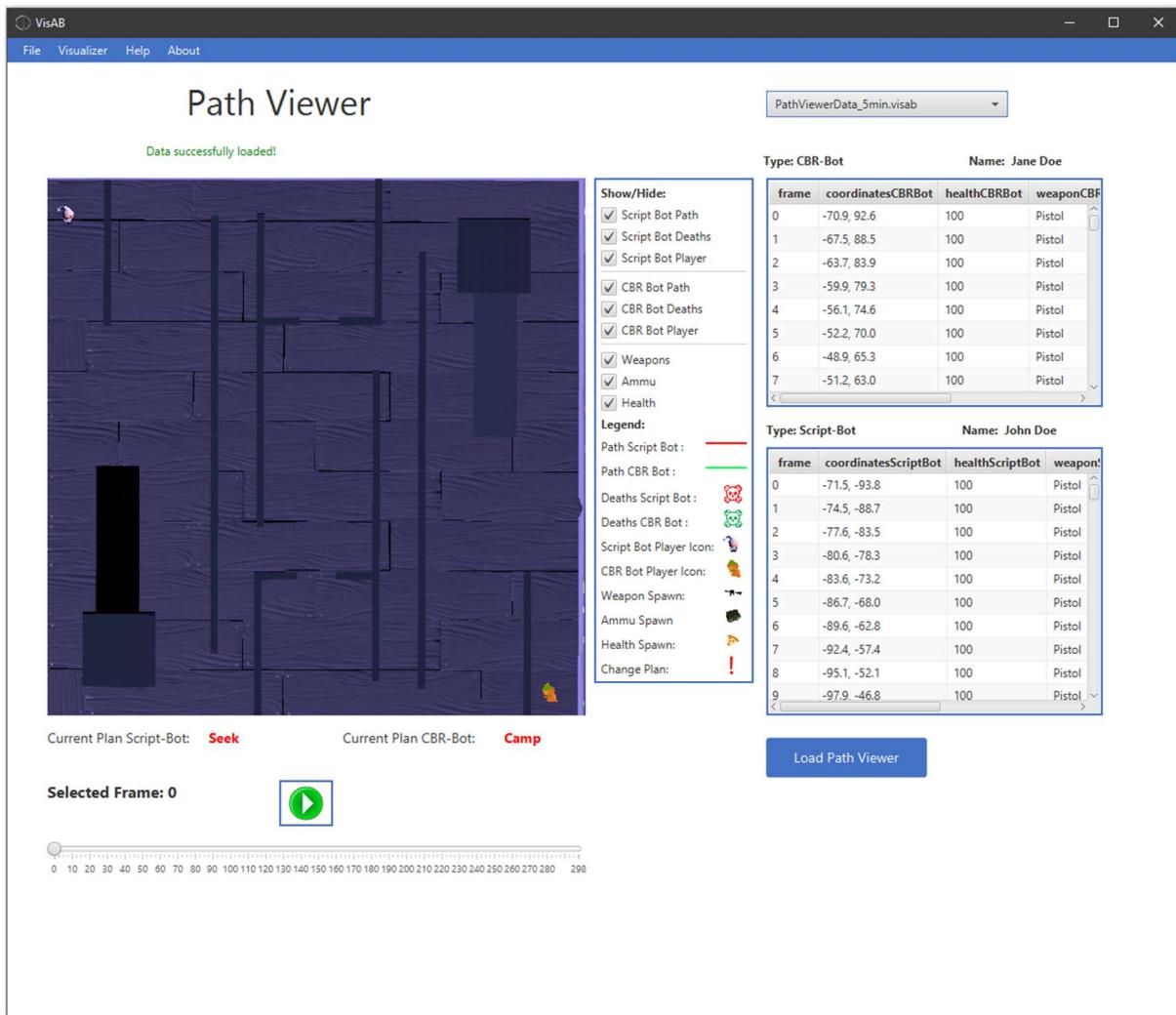


Abbildung 45: Path Viewer View

1.5.2.2 Speichern & Laden einer externen Datei

Da sich dieses Szenario mit dem vorherigen überschneidet, wird vorausgesetzt, dass bereits eine Datei gespeichert wurde, die dem Format einer externen Datei entspricht. Der Benutzer wird hierbei bereits darüber informiert, dass der Path Viewer mit dem gewählten Dateiformat nicht verfügbar sein wird. Wenn die Datei innerhalb der Statistik-Ansicht ausgewählt und geladen wird, wird der Inhalt innerhalb der Tabelle in einem generischen Format sichtbar. Des Weiteren wird hier wieder darüber informiert, welche Einschränkungen mit einem solchen Format wirksam werden.

Statistics

test.txt

No Visab file selected! Path Viewer Menu and Plan Chart is not available.

Name	Value
Name	DonaldDuck
City	Entenhausen

Abbildung 46: External Statistics

1.6 Skalierbarkeit

Um die Darstellung der Verhaltensmuster weiterer Bots zu ermöglichen, werden im Folgenden alle notwendigen Schritte zur Erweiterung des Programms aufgeführt. Dabei wurde der Code mit Kommentaren im Format „TODO: (Skalierbarkeit)“ versehen, um diese Stellen entweder durch eine Volltextsuche oder durch die Task-Funktion der genutzten Entwicklungsumgebung aufzufinden. Sollte sich „(s.o.)“ im Kommentar befinden, wird darüberstehender Code referenziert, mit „(s.u.)“ ist wiederum Code gemeint, der darunter steht.

1.6.1 StatisticsWindowController: Anpassungen

- Neue Spalten erstellen (s.o): Es müssen neue Spalten anhand eines Tabellenmodells erstellt werden.
- Neue Spalten hinzufügen (s.u.): Die Spalten müssen der Tabelle hinzugefügt werden.
- Analoge Listen erstellen (s.o.): Zur Extrahierung der Daten müssen Listen für die Einträge in der Datei der Datenbank erstellt werden.
- Analoge Schleifeneinträge erstellen (s.u.): Durch einen neuen Abgleich in der Schleife werden die zuvor erstellten Listen befüllt.

1.6.2 PathViewerWindowController: Anpassungen

- Analoge Listen erstellen (s.o.): Zur Extrahierung der Daten müssen Listen für die Einträge in der Datei der Datenbank erstellt werden.
- Analoge Schleifeneinträge erstellen (s.u.): Durch einen neuen Abgleich in der Schleife werden die zuvor erstellten Listen befüllt.
- Koordinaten extrahieren (s.u.): Für die Extrahierung der Koordinaten kann der untenstehende Schleifeneintrag kopiert und angepasst werden.
- Methode zur Erstellung einer neuen Tabelle hinzufügen (s.o.): Der Inhalt einer der beiden bestehenden Methoden kann kopiert und entsprechend angepasst werden.
- Analoge Vorbereitungsliste für die Koordinaten erstellen (s.o.): Zur Vorbereitung der Verarbeitung muss eine Liste nach dem Vorbild der bestehenden Listen erstellt werden.
- Vorbereitungsliste befüllen und Tabelleneinträge erstellen (s.u.): Die zuvor erstellte Liste muss befüllt werden und die Tabelleneinträge erstellt werden.
- Parameter für weitere Spieler(Bots) übergeben: Der Methode „drawMap“ müssen Parameter für weitere Spieler übergeben werden (bspw. Koordinaten oder Pläne). Diese richten sich nach den darzustellenden Informationen. Als Orientierung dienen die Parameter, die für CBR- und Scriptbot übergeben werden. Die Parameter müssen sowohl in der Methode selbst, als auch bei Methodenaufruf angepasst werden.

- Anpassen der Methode für weitere Spieler bzgl. Funktionen und Parametern: Methode „drawMap“ muss für zusätzliche Spieler angepasst werden. Dafür gibt es zwei Möglichkeiten:
 - 1. Möglichkeit: Methode muss um Parameter, Variablen und Funktionen für zusätzliche Spieler erweitert werden. Dieser Ansatz wird hier und im Code beschrieben und erläutert
 - 2. Möglichkeit: Die Methode selbst kann kopiert und mit analogen Parametern für weitere Spieler angepasst werden. Dafür müsste die Methode kopiert und an den entsprechenden Stellen aufgerufen werden. Dabei müssten dann die Parameter der „neuen“ zusätzlichen Spieler übergeben werden und zwecks Übersichtlichkeit die Variablennamen in der Methode angepasst werden.
- Hier wird nur auf die erste Möglichkeit eingegangen, um aufzuzeigen, an welchen Stellen in der Methode Änderungen bzw. Erweiterungen vorgenommen werden müssen.
- Erweiterungen „drawMap“ Methode:
 - Hinzufügen und Formatieren von Imageviews für neue Spieler (s.u.): Für neue Spieler müssen analog zu den vorhandenen Implementierungen neue Imageviews für die Visualisierung bestimmter Sachverhalte erstellt und formatiert werden. Die Formatierung richtet sich nach Größe und Ausrichtung der neu erstellten Imageviews. Die bestehenden Implementierungen können als Orientierung verwendet werden.
 - Analoge Listen für neue Spieler initialisieren (s.u.): Um bestimmte Spielaspekte darzustellen (z.B. Abschüsse), wurden innerhalb der Methode Listen erstellt, befüllt und verarbeitet. Diese müssen analog zu den implementierten Listen für neue Spieler erstellt werden.
 - Initialisieren und Formatieren von neuen Spielerpfaden (s.u.): Für neue Spieler müssen analog zu den bestehenden Implementierungen Pfade initialisiert werden. Diese sollten zur besseren Zuordnung für jeden neuen Spieler individuell formatiert werden. Zudem sollte ein Parameter für die Sichtbarkeit der Pfade übergeben werden, der vom Nutzer veränderbar ist (siehe Abschnitt Auswahlmenü mit Legende).
 - Schleife für Erstellung des Pfades und wesentlicher Spielaspekte für neue Spieler erstellen (s.u.): Analog zu den beiden existierenden Schleifen für den CBR- und den Scriptbot, müssen für neue Spieler die Schleife kopiert und die Parameter analog angepasst werden. Die Logik kann übernommen werden, lediglich die verwendeten Parameter und Listen müssen durch die zuvor für den neuen Spieler erstellten Parameter und Listen ersetzt werden.

- Changelistener für neue Checkboxen erstellen und analog einbauen für neue Spieler (s.u.): Um die Visualisierungen ein- bzw. ausblendbar zu machen, müssen für alle neu erstellten Checkboxen (siehe Abschnitt Auswahlmenü mit Legende) Changelistener erstellt und eingebunden werden. Die Logik kann von den bestehenden Changelistenern übernommen werden. Es müssen jedoch die Parameter und ggf. Listen für den jeweiligen Sachverhalt angepasst werden.
- Alle neuen Elemente der UI müssen dem drawPane hinzugefügt werden (s.u.).

1.6.3 Auswahlmenü mit Legende: Anpassungen

- Für neue Spieler müssen Symbole in der Legende und Checkboxen für die Sichtbarkeit angelegt werden. Dabei sollten wie für die vorhandenen Spieler alle relevanten Visualisierungsaspekte gesondert betrachtet werden. D.h., sie sollten individuell ein- bzw. ausblendbar sein und jeder sollte explizit definiert sein.
- Die Erstellung der Checkboxen und Elemente der Legende kann im Scenebuilder erfolgen und der bestehenden VBOX hinzugefügt werden.
- Für die Umsetzung der Auswählbarkeit bzgl. der Sichtbarkeit müssen Changelistener für alle erstellten Checkboxen implementiert werden (siehe Abschnitt PathViewerController: Anpassungen)

1.6.4 Sonstige Anpassungen

Zusätzlich sind folgende Schritte notwendig:

- Erstellung eines Tabellenmodells im model-Package
- Erstellung einer Tabelle mit dem Scene Builder auf dem PathViewerWindow