**Einleitung**

Nach der Spezifikation des Programms in der ersten und dem Entwurf einer Programmarchitekur in der zweiten wurde das Programm in dieser Phase nun implementiert. Der Ablauf und die Ergebnisse der Implementierungsphase sollen hier kurz vorgestellt werden.

Innerhalb der letzten vier Wochen wurden durch insgesamt über X Commits im Versionskontrollsystem git X Klassen und Schnittstellen in X Paketen implementiert, welche insgesamt X Java-Quellcode-Zeilen umfassen. Des Weiteren wurden zahlreiche Komponententests erstellt.

Im Laufe der Implementierung sind viele verschiedene Bibliotheken und Programme zum Einsatz gekommen, die hier kurz aufgelistet werden sollen.

**Werkzeuge**

**Gradle:** Gradle ist ein Build-Management-Automatisierungs-Tool zum automatischen Erstellen ausführbarer Anwendungsprogramme aus Java-Quellcode

**Eclipse und IntelliJ:** Eclipse sowie IntelliJ sind Open-Source-Code-Editoren.

**JUnit:** JUnit ist ein Test-Framework für Java.

**JavaFX:** JavaFX ist ein Framework zur Erstellung von Benutzeroberflächen für Java-Applikationen.

**ControlsFx:** : ControlsFX ist eine GUI-Bibliothek, die JavaFX ergänzt.

**JGraphT:** JGraphT ist eine Open-Source-Graphbibliothek für Java, die in einem unserer Paarbildungsalgorithmen zum Einsatz kam

**Gson:** Gson ist eine Open-Source-Java-Bibliothek zum Serialisieren und Deserialisieren von Java-Objekten

**Apache Commons Math:** Apache Commons Math ist eine Open-Source-Java-Bibliothek, die grundlegende mathematische und statistische Werkzeuge zur Verfügung stellt. Diese kam bei der Generierung spezifiziert verteilter Zufallszahlen zum einsatz

**Zeitplan**

Zu Beginn der Implementierungsphase wurde vereinbart, die Aufteilung anhand der Paketstruktur vorzunehmen. Weiterhin erachteten es die Entwickler als sinnvoll, zunächst das Model zu implementieren und zu testen, und erst danach mit Controller und View zu beginnen. Grund ist eine Vereinfachung der Fehlersuche bei der Implementierung des Controllers. Ist dann nämlich das Model bereits fertig implementiert und getestet und es entsteht ein Fehler in einem Ablauf, in dem sowohl Controller als auch Model beteiligt sind, so liegt der Fehler höchstwahrscheinlich im Controller. Diese Vorgehensweise hat sich bewährt. Bei der Implementierung des Models sind wir so vorgegangen, dass wir das aus fünf Entwicklern bestehende Team aufgeteilt haben in drei, die mit der Implementierung der 8 Model-Pakete beginnen und zwei, die parallel Komponententests dazu schreiben. Die Implementierung des Models verlief reibungslos und war bereits nach etwa eineinhalb Woche abgeschlossen. Während die zwei Tester noch weiter an Komponententests für das Model arbeiteten, begonnen die anderen drei Entwickler mit der Implementierung des Controllers und dem Anfertigen der fxml-Dateien für die View. Da die meisten Entwickler keine Erfahrung im Umgang mit JavaFX hatten, stellte sich diese Phase als etwas zeitaufwendiger heraus, verlief aber auch ohne Probleme.

**Was wurde implementiert**

Alle Kann- sowie Musskriterien wurden erfolgreich implementiert. Es wurden 2 kleine Änderungen veranlasst:

1) Die Form der Ausgabe der Strategieverteilung der Agenten wurde angepasst. Dies werden Sie nachher in der Live-Demo veranschaulicht bekommen

2) Neben Konfigurationen und Simulationsergebnissen können auch Gruppen, Populationen, Strategien und Stufenspiele als Dateien exportiert und importiert werden.

3) Die Struktur von Gruppen und Segmenten hat sich in der Entwurfsphase durch die Einführung von Populationen geändert (siehe Entwurfsdokument)

**Live-Demo**

Kommen wir nun zur Demonstration unseres Programmes.

1. **Standard-config mit weniger Runden laufen lassen**
2. **CPU-Auslastung**
3. **Resultate veranschaulichen**
4. **Spiel erstellen**
5. **Strategy erstellen**
6. **Gruppe erstellen**
7. **Population erstellen**
8. **Population auswählen**
9. **Simulation mit erstellten Objekten**
10. **Multi-Konf-Resultat laden**

Vielen Dank für ihre Aufmerksamkeit. Wenn sie wollen, können sie unser Programm auch selbst mal ausprobieren