**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 wir es 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. 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. Dies wurde schon in der Entwurfsphase spezifiziert.

**Live-Demo**

Kommen wir nun zur Demonstration unseres Programmes.

1. **Standard-config mit weniger Runden laufen lassen**

Wir öffnen das Programm und sehen, dass die Ausgabe noch leer ist. Es ist eine Standard-Konfiguration eingestellt, die wir mit weniger Runden laufen lassen wollen. Also drücken wir auf das Zahnrad und gelangen zum Konfigurationsfenster, wo wir alle Einstellungen treffen können. Zum Bestätigen drücken wir auf den grünen Haken und sehen das die Einstellung übernommen wurde. Jetzt starten wir unserer Simulation 2mal.

1. **CPU-Auslastung**

Mit einem Klick auf das rote Kreuz, können wir die Simulation abbrechen.

Wir sehen im Hintergrund, dass alle 4 Kerne voll ausgelastet werden

1. **Resultate veranschaulichen**

Ist die Simulation fertig, so können wir uns die gewünschten Ergebnisse anschauen. Neu in unserer Ausgabe ist dieses Diagramm, das die relativen Anteile der Strategien im Verlauf der Simulation anzeigt. (Beschreiben)

Der Rest der Ausgabe ist unverändert und so wie im Entwurf spezifiziert.

1. **Spiel erstellen**

Dies waren die Standard-Funktionen. Kommen wir nun zu den erweiterten Möglichkeiten.

Zuerst wollen wir unser eigenes Spiel erstellen. Dazu gehen wir im Menü punkt „Extensions“ auf „Create new game“. Hier geben wir dem Spiel Auszahlungswerte, eine Beschreibung und einen Namen. Mit einem Klick auf den Haken wird das Spiel nur ins Repository eingefügt und ist beim nächsten Öffnen des Programms nicht mehr da. Um dies zu verhindern, speichern wir das Spiel in einer Datei ab. Alle Spiele die in diesem Ordner liegen werden beim nächsten Öffnen automatisch geladen.

1. **Strategie erstellen**

Nach der Erstellung eines eigenen Spiels, wollen wir nun eine eigene Strategie erstellen. Wir gehen wieder unter „Extensions“ und dieses mal auf „Create new Strategy“. Hier können wir uns unsere eignen Strategien intuitiv zusammenklicken. Wir erstellen die Strategie „Wenn der andere Agent in der gleichen Gruppe ist wie ich, dann mach ich Tit-For-Tat, ansonsten Nie kooperieren.

Wir nenne sie Tit-for-tat with group und fügen sie ins Repository ein.

1. **Gruppe erstellen**

Jetzt erstellen wir uns eine Gruppe die nur unsere eben erstellte Strategie verwendet. Das Fenster finden wir wieder unter „Extensions“. Ich füge der Gruppe zwei Segmente hinzu mit gleichem Anteil an Agenten. Beide erhalten nur unsere eben erstellte Strategie, jedoch verschiedene Kaptialverteilungen. Ich wähle die Poissonverteilung mit Parameter 0.5 beim ersten Segment und 0.3 beim zweiten.

Diese Gruppe nenne ich Egozentriker und füge sie ins Repository ein.

1. **Population erstellen**

Bekanntlich werden die Gruppen zu Populationen zusammengesetzt. Dies geschieht in unserem Populationsfenster. Hier kann man eine Gruppe wählen mit einer Anzahl an Agenten und diese dann der Population hinzufügen.

Ich will 20 Agenten der Gruppe Egozentriker und 30 aus der vorgefertigten Standard-Gruppe.

Ich gebe der Population den Namen und speichere sie als externe Datei ab

1. **Simulation mit erstellten Objekten**

Spiele, Gruppen und Populationen können nach dem abspeichern wieder in den respektiven Fenster geöffnet und editiert werden. Dies passiert mit einem Klick auf den Pfeil nach oben.

Nach dem wir uns nun eigene Objekte erstellt haben, wollen wir eine Simulation mit diesen starten.

Im Konfigurationsfenster können wir nun unser erstelltes Stufenspiel und unsere neue Population auswählen.

1. **Multi-Konf-Resultat laden**

Weitere Erweiterungen, die ich ihnen bis jetzt vorenthalten habe, finden sie unter „File“. Sie sehen, sie können auch die aktuelle Konfiguration speichern und abgespeichert Konfigurationen laden. So wie Simulationsresultate laden. Letzteres will ich ihnen noch zeigen anhand einer interessanten Multikonfigurations-Simulation die wir im Vorfeld erstellt haben.

Das Speichern der ausgewählten Simulations-Ergebnissen erfolgt übrigens über diesen Knopf

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