HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ADV Landscape Module

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2019

Autor(en): Yannic Schneiter, Marc Weber, Lukas Christel
Betreuer: Thomas Letsch

Datum: 19.12.2019
Dokumentversion: 1.2

Management Summary

Ausgangslage

Im Unterricht für Algorithmen und Datenstrukturen wird ein eigenentwickeltes Tool (ADV) zur Visualisierung der gelernten Algorithmen verwendet. Der ADV ist modular um neue Algorithmen erweiterbar und unterstützt bereits einige Algorithmen. Ziel dieser Arbeit ist es, ein zusätzliches Modul zur Visualisierung des Dijkstra Algorithmus zu entwickeln.

Vorgehen

Im bestehenden Tool zur Visualisierung von Algorithmen (ADV) wird ein einzelnes neues Modul zur Visualisierung des Dijkstra Algorithmus entwickelt.

In einem Prototyp wird ermittelt, wie gut JavaFX für die Darstellung von 3D Landschaften geeignet ist. Der Code aus diesem Prototyp wird anschliessend in ein neues Modul im ADV übernommen.

Technologien

Das bestehende Tool zur Visualisierung von Algorithmen (ADV) ist in zwei Komponenten aufgeteilt. Das UI ist mit JavaFX implementiert und wird von uns mit den 3D Bibliotheken von JavaFX erweitert, um eine Landschaft darstellen zu können. Die zweite Komponente stellt die Datenstrukturen bereit, die im Unterricht gelernt werden. Die Komponenten kommunizieren über eine Socket-Schnittstelle.

Ergebnisse

Durch die Entwicklung des Prototyps konnte festgestellt werden, dass JavaFX zur Darstellung von 3D Elementen ausreicht. Es werden keine zusätzlichen Grafik-Libraries benötigt.

Nebst der Darstellung des Dijkstra Pfades ist es jetzt im fertigen Produkt auch möglich, weitere Informationen darzustellen, u. A. Kreise und Punkte.

Inhaltsverzeichnis

Management Summary 1
Ausgangslage 1
Vorgehen 1
Technologien 1
Ergebnisse 1
Technischer Bericht 4
Ausgangslage & Problembeschreibung 4
Lösungskonzept 4
Vorgehensweise 5
Umsetzung 6
Ergebnisdiskussion 6
Eingetretene Risiken 6
Clipping-Bug 7
Performance Verbesserung 7
Ausblick 8
Projektorganisation 9
Externe Schnittstellen 9
Arbeitspakete 9
Zeitauswertung 9
Wöchentliche Besprechung 9
Arbeitszeit 9
Meilensteine 9
Risiken 10
Qualitätsmassnahmen 11
Dokumentenablage 11
Backup 11
Testing 11
Anforderungsspezifikationen 12
Functional Requirements 12
Non-Functional Requirements 12
Zwingende Anforderungen 12
Optionale Anforderungen 12
User Stories 13
US1: Grafische Darstellung 13
US2: Random Generated 13

Use Cases 13
Hauptszenario 13
Erweitert 13
Definition of Done 13
Code Guidelines 14
Version Control und Branches 14
Code Review 14
Architektur und Design 15
Literaturverzeichnis 17
Tabellenverzeichnis 17
Abbildungsverzeichnis 17
Administrative Anhänge 18
Projektplanung 18
Zeitabrechnung 19

Technischer Bericht

Ausgangslage & Problembeschreibung

In vorangegangenen Arbeiten ist für die Module Algorithmen und Datenstrukturen (AD1 und AD2) der ADV entwickelt worden. Der ADV soll die bisher eingesetzten Tools, wie GVS und jogl, ablösen. Damit müssen die Studenten nur noch eine Plattform bzw. Applikation erlernen und verwenden.

Der ADV (Algorithm & Data Structure Visualizer) ist modular aufgebaut und kann entsprechend unabhängig vom Kern erweitert werden. Diese Applikation ist grundsätzlich in zwei Teile aufgeteilt: Ein Teil ist für die Darstellung von Datenstrukturen verantwortlich und mit JavaFX implementiert. Der andere Teil stellt die Datenstrukturen bereit, welche von den Studenten verwendet werden. Diese beiden Artefakte kommunizieren über eine Socket-Schnittstelle, welche die Daten vom Client an das User Interface sendet.

Im Rahmen dieser Studienarbeit soll das vorläufig letzte Modul zum ADV hinzugefügt werden. Hier geht es darum, dass eine 3D-Landschaft angezeigt wird, in welcher ein Shortest-Path-Algorithmus angewendet werden kann. Damit soll der Student vor allem den Dijkstra-Algorithmus auf einem Graphen implementieren, welcher ein Gitternetz abbildet. Die Kanten dieses Graphen sind so gewichtet, dass sie die Höhenunterschiede in der Landschaft repräsentieren.

Der Hauptbestandteil dieser Arbeit ist, die Lösung so zu erstellen, dass die bisherigen Hilfsmittel abgelöst werden können. Zudem werden optionale Erweiterungen eingebaut, so dass das neue Tool generell, also auch für zukünftige, zusätzliche Algorithmen, eingesetzt werden kann. Sobald diese Arbeit umgesetzt ist, kann das letzte Third-Party-Hilfsmittel aus dem Modul AD2 entfernt werden.

Lösungskonzept

Die Lösung für die 3D-Landschaft besteht aus einem einzelnen, neuen Modul für den ADV. Die Verwendung dieses Moduls durch den Studenten geschieht über eine einzelne abstrakte Klasse. Diese stellt einerseits die Factory-Methoden bereit, um die Landschaft zu erstellen. Andererseits enthält diese Klasse auch die wichtigsten Methoden für den Studenten. Durch diese Abstraktion wird die Schnittstelle für den Studenten klein gehalten und ermöglicht es gleichzeitig auch, Implementationsdetails in Subklassen zu kapseln.

Da bereits die bestehenden Module in JavaFX implementiert wurden, ist auch dieses neue Modul mit demselben Framework umgesetzt. Diese Strategie ermöglicht uns, die neue Lösung so plattform-unabhängig wie möglich zu halten. Genau diese Unabhängigkeit von Betriebssystemen ist eine der wichtigsten Gründe, warum die bisherige Lösung neu implementiert werden sollte.

Vorgehensweise

Die Entwicklung des Moduls wurde in mehreren Schritten durchgeführt, angelehnt an das Phasen-Modell von Unified Process. Zunächst wurde die Projekt-Planung durchgeführt, die Entwicklungsumgebung eingerichtet und Verständnis für die Architektur des bestehenden Source Codes aufgebaut. Dazu gehörte auch, dass wir Gradle als Build-Tool kennen lernen mussten, da wir bisher nur mit anderen Build-Tools gearbeitet haben.

Anschliessend wurde ein Prototyp erstellt. Dabei ging es darum, herauszufinden, wie die Entwicklung von 3D-Objekten in JavaFX konzeptioniert ist und welche Techniken für eine 3D-Landschaft funktionieren. Wir haben schnell gemerkt, dass reines JavaFX für die Lösung der Aufgabe ausreicht. Es mussten keine weitere Libraries hinzugeschalten werden. Obwohl wir uns für diese Studienarbeit zuerst die Funktionsweise von JavaFX erarbeitet haben, benötigte dieser Schritt verhältnismässig wenig Zeit.

Der nächste Schritt war, den Code des Prototyps in das neue ADV-Modul zu übernehmen. Dafür musste zunächst ein neues Modul im bestehenden ADV erstellt werden. Danach wurde der Code aus dem Prototyp entsprechend migriert und die Klassen hinzugefügt, welche für die Kommunikation der beiden Artefakte miteinander verantwortlich sind. Anschliessend wurden weitere, zusätzliche Features hinzugefügt und vor allem diverse Fehler behoben. Gerade dieser letzte Schritt hat viel Zeit beansprucht, da sich einige Komponenten von JavaFX anders verhalten, wenn sie in anderen Komponenten verschachtelt werden. Zudem mussten diverse Schritte unternommen werden, um die Performance im ADV zu gewährleisten.

Die Dokumentation wurde fortlaufend nach Bedarf angepasst und erweitert. Dazu gehörte einerseits die gesamte Dokumentation der Studienarbeit, sowie auch das Benutzerhandbuch des neuen Landscape-Moduls. Letzteres umfasst neben einer Installationsanleitung auch insbesondere die wichtigsten Methoden, welche von den Benutzern aufgerufen werden können. Dadurch dass sich das ADV-Modul dauernd weiterentwickelt hat, mussten diese Erklärungen auch auf dem aktuellen Stand gehalten werden.

Umsetzung

Damit die Landschaft keine Ecken hat, haben wir aus dem Input Array, welches die Berge und Täler definiert ein grösseres Array erstellt, welches dazu gebraucht wird, Landschaft geschmeidig darzustellen. Die Kurve wird anhand eines Algorithmus berechnet, welcher dem Abstand zum Tal oder Berg eine Gewichtung zuteilt und anhand der Gewichtung die Höhe der jeweiligen Position berechnet. Bei dem Design, für das wir uns entschieden haben, werden jeweils vier Punkte des kleineren Arrays betrachtet, um die Punkte des Grösseren zu berechnen. Es gäbe auch Varianten 16 Punkte sodass die Steigung, nicht wie bei uns, an den vorgegebenen Punkte nicht immer null wäre.

Vom Input-Array weiss der Benutzer nichts, wenn er es zufällig generieren lässt. Da die Abstände, die wir zum Rendern der Grafik brauchen, zu fein und die Abstände auf der die Berge und Täler definiert sind zu grob sind, haben wir eine weitere Unterteilung implementiert, mit der gerechnet werden kann, und mit der Punkte sowie Kreise auf die Landschaft gezeichnet werden können und die dazu verwendet werden soll, mit dem Dijkstra-Algorithmus den kürzesten Pfad zu berechnen.

Wenn das Array zufällig generiert wird, werden wie oben beschrieben drei verschiedene Arrays verwendet. Da es nicht intuitiv wäre, ein drittes Array zu verwenden, wenn das Array vom Benutzer definiert wurde, wird in diesem Fall das vom Benutzer vorgegebene Array verwendet. Die Idee hinter dieser Entscheidung war, das selbst definierte Array nur zu Testzwecken zu verwenden, sodass der Benutzer eine bessere Übersicht darüber hat, was in seinem Programm passiert.

Ergebnisdiskussion

Damit das Programm eine einigermassen sinnvolle Antwortzeit hat, mussten die Anzahl Punkte zur Darstellung der Landschaft im Vergleich zu der des Prototyps verkleinert werden. Visuell gibt es kaum ein Unterschied, aber auf langsameren Laptops könnte die Performance zu einem Problem werden.

Eingetretene Risiken

Ein Risiko, welches wir im Voraus vermutet haben, betrifft das Thema fehlendes Know-How. Obwohl der Prototyp des Landscape-Modules in relativ kurzer Zeit fertiggestellt werden konnte, so beanspruchte die erfolgreiche und korrekte Integration in den bestehenden ADV viel Zeit.

Wir vermuten, dass uns mehr, sorgfältige Einarbeitungszeit in das Thema JavaFX in diesem Fall nicht weitergeholfen hätten. Die aufgetretenen Herausforderungen waren sehr spezifisch auf die Anwendung von 3D-Objekten.

Schlussendlich konnten wir die meisten Probleme mit Hilfe von Pair Programming lösen. Zum Beispiel gab es einige Details, welche der einen Person aufgefallen sind, die andere Person hingegen übersprungen und gar nicht ausprobiert hätte.

Clipping-Bug

Nachdem die Landschaft aus dem Prototyp in den ADV übernommen wurde, war die grösste Aufgabe, das Clipping der entsprechenden JavaFX-Pane richtig zu konfigurieren. Denn zunächst ragte die Landschaft über den Rand der ihr zugewiesenen Pane hinaus.

Die Lösung wäre eigentlich ganz einfach: Auf der betroffenen Pane könnte Clipping definiert werden, so dass die Landschaft am Rand abgeschnitten wird. Dies hat dann jedoch dazu geführt, dass die Landschaft falsch dargestellt wurde. Täler, welche eigentlich hinter Bergen versteckt sein sollte, waren plötzlich im Vordergrund sichtbar.

Nachträglich hat sich herausgestellt, dass Clipping in JavaFX als 2D-Operation konzipiert ist und daher auch mit Problemen im 3D-Bereich zu rechnen ist. Wir konnten dieses Problem schlussendlich umgehen, indem wir mit Property-Binding die Grössen und Positionen angepasst haben. So haben wir unter anderem die Höhe und Breite der Subscene mit denen der umliegenden Pane verbunden. Nachdem wir herausgefunden haben, dass dazu die Grössenänderung nicht mehr automatisch durch JavaFX verwaltet werden darf und dieses Verhalten abgeschaltet werden kann, wurde die Landschaft richtig im ADV dargestellt.

Performance Verbesserung

Da wir das System mit den Snapshots aus dem ADV übernommen haben, kam es zu Performance-Problemen. Die Landschaft beim Client auszurechnen und dem Host zu übermitteln war nicht optimal, da alle berechneten Punkte der Landschaft zu einem riesigen Array führen, welches zu einem String umgewandelt, übertragen und geparst werden musste. Es hat sich als performanter herausgestellt, sowohl beim Host als auch beim Client die Punkte der Landschaft zu berechnen.

Durch die Architektur des ADV wurde die Landschaft am Anfang pro Snapshot einmal geschickt. Die vorher beschriebene Massnahme wurde die Performance signifikant verbessert, aber die Landschaft wurde anfangs pro Snapshot einmal und zusätzlich einmal im Client berechnet. Um diese teure Operation zu verhindern haben wir die benötigten Daten im Server gecached. Da die verfeinerte Landschaft später dazu verwendet wird, um ein Triangle Mesh herzustellen, haben wir uns anstelle des Caching der Landschaft dazu entschlossen, das Triangle Mesh zu cachen, um die Performance weiter zu verbessern.

Ausblick

Mit dieser SA zum Landscape Module stehen die Grundlagen, um zum Beispiel einen Shortest-Path-Algorithmus in einer 3D-Landschaft zu visualisieren. Dennoch ist dieses Modul generisch gehalten, so dass auch weitere 3D-Visualisierungen denkbar wären.

Es bieten sich vor allem Features an, welche weitere Objekte in einer Landschaft platzieren. So könnten neben Pfaden, Kreisen und Punkten zum Beispiel auch Quader innerhalb dieser Landschaft dargestellt werden.

Wenn weitere 3D-Module geplant werden, welche nicht auf einer Landschaft basieren, kann sich möglicherweise auch herausstellen, ob und wie die Darstellungs-Komponenten des Landscape Modules extrahiert werden können. Denkbar ist, dass die Steuerung und Darstellung von MeshViews generisch wiederverwendet werden könnten. Ebenso könnte das Grundkonzept der Snapshots für 3D-Module überarbeitet werden, da ansonsten potenziell sehr viele Daten vom Client and das UI gesendet werden müssten.

Eine Abstraktions-Schicht für JavaFX einzubauen wird hingegen sehr aufwändig, auch wenn dies die Entwicklung von späteren 3D-Modulen erheblich vereinfachen würde. Einerseits sind die Darstellung bzw. das Einfärben von Objekten komplex, da dafür ein sogenanntes Material als Bild aufbereitet werden muss. Andererseits kann ein solcher Layer auch dazu führen, dass die spezifischen Implementationen keine Möglichkeit haben, die Objekte hinsichtlich Performance zu optimieren, indem einige Vertices weggelassen werden. Im Projekt werden genau 3 verschiedene Arten von 3D Modellen verwenden: Sphären, Zylinder und die Landschaft, die Sphäre sowie der Zylinder sind in JavaFX einfach zu verwenden sodass es keinen Sinn macht dafür einen Layer einzubauen. Einzig die Methode, die aus einem 2-dimensionalen Array eine MeshViews macht, wäre Sinnvoll in einem tieferen Layer zu implementieren. Einen Layer nur für eine Methode zu generieren wäre unserer Meinung nach übertrieben.

Projektorganisation

Externe Schnittstellen

Das Projekt wird von Thomas Letsch betreut. Kommuniziert wird per E-Mail und mit Besprechungen vor Ort.

Arbeitspakete

Das Projekt ist mit einem iterativen Entwicklungsprozess agil organisiert. Da das Projekt eine Studienarbeit ist, sind trotzdem einige Termine vorgegeben. Diese werden berücksichtigt.

Zur Organisation der Arbeit verwenden wir Azure DevOps mit einem angepassten Scrum Workflow. Bei der Planung eines Sprints werden die Arbeitspakete geschätzt. Wenn ein Arbeitspaket so wie vorgegeben nicht in den Sprint aufgenommen werden kann (z.B., weil es zu gross ist), wird es überarbeitet und später in einen Sprint aufgenommen.

Zeitauswertung

Zur Auswertung unserer benötigten Zeit verwenden wir 7pace Time Tracker in Azure DevOps.

Wöchentliche Besprechung

Wir besprechen während der gesamten Projektdauer wöchentlich am Donnerstag, 12:00 Uhr den Stand des Projektes mit unserem Betreuer. Dabei ist jeweils ein Rückblich auf die vergangene Woche, der aktuelle Projektstand und die Arbeit der nächsten Woche das Thema. Für diese Besprechungen wird ein Kurzprotokoll geführt.

Arbeitszeit

Wir arbeiten grösstenteils selbständig an unseren Aufgaben. Wenn eine Diskussion notwendig ist, sehen wir uns mehrmals täglich und können somit Unklarheiten sofort besprechen.

Meilensteine

14.11.2019 07:00	Erster Release	Erste Funktionalitäten implementiert
04.12.2019 07:00	Zweitletzter Release	Weitere Funktionalitäten sind implementiert. UML Modell mit Enterprise Architect
20.12.2019 17:00	Letzter Release	Abgabe der Studienarbeit

Tabelle 1 Übersicht über die Meilensteine in dieser SA

Risiken

Nr.	Titel	Beschreibung	max. Schaden (h)	Eintritts-wahrschein-lichkeit	Vorbeugung	Verhalten beim Eintreten
1	Fehlerhafte Leitung des Projekts	Der Projektleiter oder Scrum Master verliert den Überblick.	6	30%	Entscheidungen werden auf Azure DevOps bzw. OneDrive dokumentiert.	Teammeeting für Brainstorming.
2	Planungs-Fehler	Tasks benötigen länger als geplant.	12	50%	Über das gesamte Projekt werden Pufferzeiten definiert, welche lang-andauernde Tasks ausgleichen.	Es muss ein Meeting mit dem SA-Betreuer organisiert werden, um eine mögliche Lösung zu erarbeiten.
3	Fehlende Kompetenzen	Benötigtes Wissen für verwendete Frameworks fehlt.	24	60%	Für die Einarbeitung muss genügend Zeit eingeplant werden.	Das Thema sollte mit den anderen Team-Mitgliedern oder ggf. mit dem Betreuer besprochen werden.
4	Merge Conflicts	Merge Conflicts verzögern die geplante Arbeit.	6	20%	Feature Branches verkleinern den Umfang der betroffenen Codes.	Die Teammitglieder werden informiert, so dass keine Commits den Merge erschweren.
5	Datenverlust	Dokumente oder Source Code geht verloren.	24	10%	Tasks werden mind. täglich ins zentrale Repository gepusht, Dokumente werden in der Cloud gespeichert. Wichtige Daten sind zudem lokal gesichert. Somit beträgt ein Ausfall max. ein Arbeitstag (8 Stunden)	Es muss ein dringendes Meeting mit dem SA-Betreuer einberufen werden.

Tabelle 2 Beschreibung der Risiken

Qualitätsmassnahmen

Einige Werkzeuge sind bereits durch die Vorgängerarbeiten am ADV[1] vorgegeben. Zudem gilt die Anforderung, dass ein Ausfall einer Komponente einen Arbeitsverlust von höchstens 8 Stunden verursachen darf. Das heisst, nach Ablauf dieser Zeit muss die Arbeit wieder auf dem gleichen Stand sein, wie kurz vor dem Ausfall.

Dokumentenablage

Damit alle wichtigen Dokumente gesichert und von überall her zugreifbar sind, wird für diese Arbeit OneDrive verwendet. So sind alle Team-Mitglieder stets auf dem aktuellen Stand und das Backup der Dokumente ist gleichzeitig auch geregelt. Die relevanten Dokumente (z.B. Protokolle der Besprechungen) werden jeweils dem Betreuer per Mail zugestellt.

Backup

Alle Arbeits-Dokumente und der Source Code sind an mehreren Orten gespeichert. Einerseits befinden sich diese Dateien auf den Geräten der Entwickler, welche jeweils ihre Daten in regelmässigen Abständen sichern. Andererseits werden auch Ablagen in der Cloud verwendet, wie zum Beispiel OneDrive für die Dokumente und Azure DevOps für das Git-Repository.

Testing

Unit Tests werden mit den bestehenden Frameworks aus den Vorgängerarbeiten durchgeführt. Dazu werden vor allem Microtests auf Klassen-Ebene durchgeführt.

Da es sich bei diesem Projekt vor allem um ein 3D-Visualisierungs-Projekt handelt, werden die meisten Tests manuell von uns ausgeführt. Dazu gehört insbesondere die Ausführung der Klassen aus “ADV-User_Codebase”. Diese Klassen befinden sich im landscape-Package und beinhalten unterschiedliche Szenarien, wie dieses Modul verwendet werden kann.

Wir konnten die Entwicklung am Landscape Module schneller abschliessen als zunächst gedacht. Deshalb hatten wir die Möglichkeit, bereits früher abzugeben und damit den finalen Usability Test direkt mit den diesjährigen Studenten als Teil der AD2 Übung durchzuführen.

[1] (Winter & Meier, 2018)

Anforderungsspezifikationen

Functional Requirements

Die Aufgabe dieser Arbeit ist es eine 3-dimensionale Karte mit einem Pfad anzuzeigen, der vom User dem Programm übergeben wird.

Das Programm soll als weiteres Modul im ADV eingepflegt werden.

Die Benutzereingaben sollen gleich gehandhabt werden wie in der Version mit jogl.

Das Programm soll Schichten-mässig aufgebaut werde, sodass die unterste Schicht, die nur für die Darstellung zuständig ist, auch für andere Module verwendet werden kann.

Non-Functional Requirements

Zwingende Anforderungen

Einfache Bedienung für den User, optimal nur eine unserer Klassen soll verwendet werden müssen. Eine intuitive Bedienung der Schnittstelle hat aber Vorrang.

Das UI des ADV darf bei falschen Inputs vom Client nicht abstürzen.

Eine einzelne Karte soll innerhalb von 3 Sekunden gerendert werden können. Davon ausgeschlossen ist der Zeitraum, in welcher der Source Code beim Client kompiliert wird.

Optionale Anforderungen

Die CPU-Auslastung während dem Idle-Modus des Landscape-Modules (die Landschaft wird angezeigt und keine Berechnungen oder Rendering werden vorgenommen) soll die Auslastung ohne laufendes Landscape-Modul nicht um das Zweifache übersteigen.

User Stories

US1: Grafische Darstellung

Als Student möchte ich meine Karte und meinen Pfad übergeben und die dazugehörige Visualisierung angezeigt bekommen. Dadurch kann ich besser lernen wie mein Algorithmus funktioniert.

US2: Random Generated

Als Student möchte ich die Karte, die mein Algorithmus verwendet, zufällig generiert erhalten können, oder eine eigene Karte verwenden.

Use Cases

Weil die einzige Aufgabe des Landscape Modules das Anzeigen einer Landschaft und eines Pfades ist, gibt es eine beschränke Anzahl an Use Cases.

Hauptszenario

1. Der Student startet Eclipse.
2. Der Student bereitet den Code vor, welcher den Pfad ausrechnet.
3. Das System Startet ADV und übermittelt die Daten des Studenten.
4. Das System zeigt die Landschaft mit dem angegebenen Pfad an.

Erweitert

1a. Der Student Konfiguriert die Anzeige des ADV über das config.properties File.
1b. Der Student ersetzt die Landschaft durch eine zufällig generierte.
2a. Der Student zeichnet mit einer zur Verfügung gestellten Funktion einen Kreis auf die Landschaft, um eine Übersicht über die Dimension zu erhalten.

Definition of Done

Unsere Definition of Done besteht ausfolgenden Punkten:

- Acceptance Criteria erfüllt
- Unit Tests abgeschlossen
- Code review gemacht
- Non Functional Requirements getestet
- Dokumentation nachgeführt
- Build erfolgreich
- Deployment erfolgreich

Code Guidelines

Für den Java Source Code werden wir uns an die üblichen Java Codier-Richtlinien halten. Weitere Regeln werden mit Checkstyle überprüft.

Version Control und Branches

Der bestehende Source Code wurde in mehreren Repositories zur Verfügung gestellt, welche jeweils die folgenden Verantwortlichkeiten haben:

- ADV-Lib enthält die Server-seitige Logik, welche unter anderem auch die grafische Darstellung in einem GUI beinhaltet.
- ADV-UI enthält die Client-seitige Logik, welche von einem Studenten benötigt wird.
- ADV-Commons enthält die Klassen, welche in beiden anderen ADV Projekten verwendet werden können.
- ADV-User_Codebase enthält Code-Beispiele, welche die Benutzung erläutern.

Die Repositories werden für diese Studienarbeit mit Azure DevOps gehostet. Diese vier Git-Repositories weiterhin separat zu halten hat den Vorteil, dass z.B. die Version-Geschichte erhalten bleibt und für spätere Arbeiten die Möglichkeit erhalten bleibt, diese Repositories wieder in die ursprünglichen Repositories zurückzuführen.

Feature-Branches werden gleich benannt, wie die dazu gehörende Story und deren ID. Sofern sinnvoll werden die gleichen Branches für die verschiedenen Git-Repositories verwendet. Das Schema sieht wie folgt aus:
[ID]-[Titel (mit Bindestrich getrennte Wörter)]
Ein möglicher Branch-Name ist: feature/89-create-event

Code Review

Damit der Code Branches gemerged werden können muss mindestens eine Person, die nicht an dem Branch beschäftigt war, den Code prüfen. Sollten Unklarheiten oder kritische Codeabschnitte vorhanden sein werden diese im Team besprochen.

Architektur und Design

Die Architektur wurde uns durch das bereits bestehende ADV grösstenteils vorgegeben. Wir haben für den Studenten eine Factory Method die ihm eine ADVLandscapeImpl erstellt. So sieht der Student nur die Methoden, die er benötigt und nicht diejenigen, die wir für die Erstellung der Landschaft brauchen.

Das bisherige System arbeitet mit Snapshots, die verschiedenen Stände aufzeigen, die von Studenten definiert sind. Wir erzeugen für die Darstellung und für die Berechnungen viele Daten in der Landscape. Die gesamte Landscape mit allen Daten für die Darstellung beim Client in einen String zu verwandeln und diesen beim Host wieder zu Parsen geht mehr als doppelt so lange als die Landschaft neu zu berechnen. Dieser Effekt kann mit Zunahme der Snapshots besser beobachtet werden.

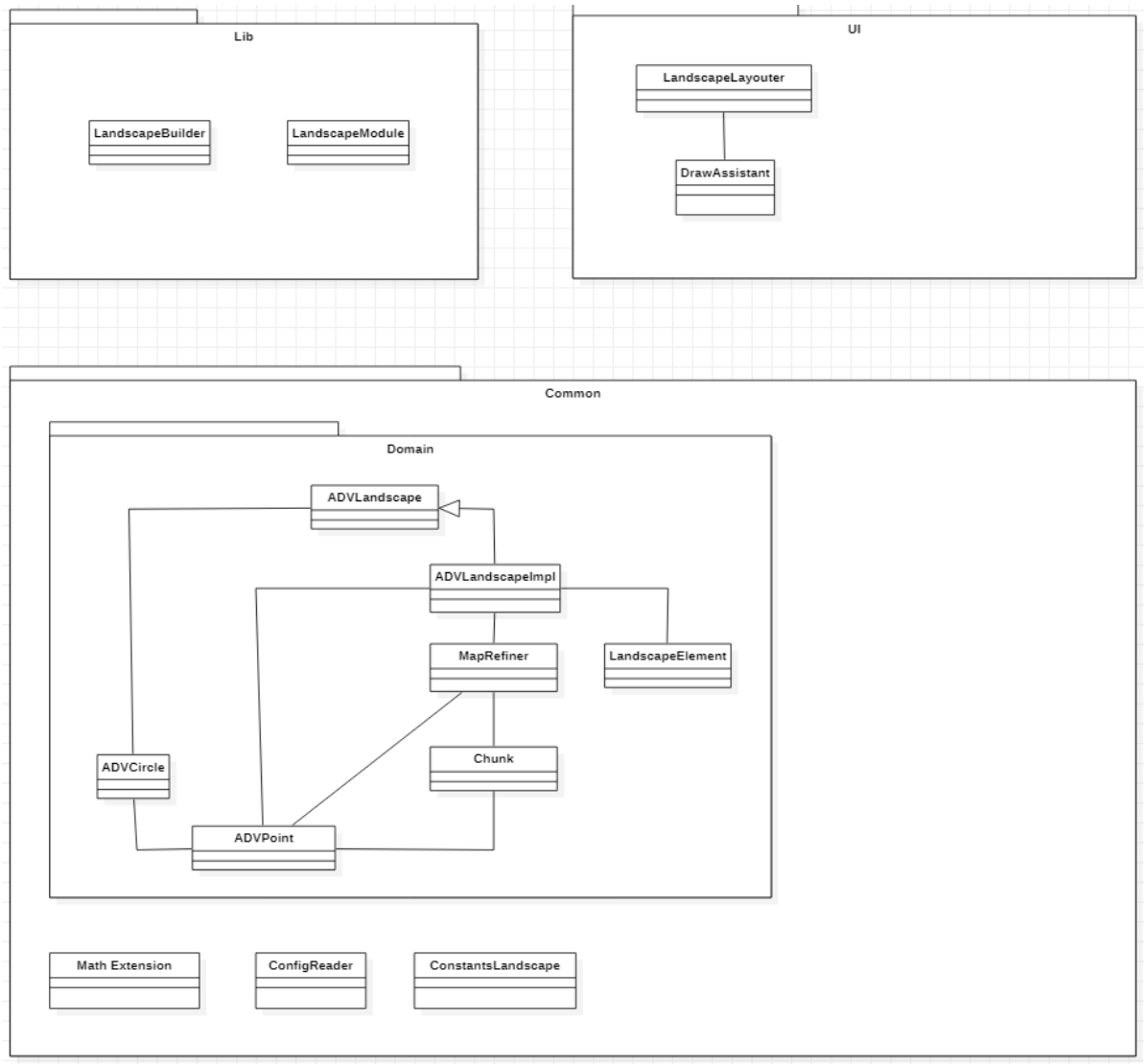

Abbildung 1 Klassendiagramm über die neuen Klassen des ADV Landscape Modules

Anfangs des Projekts wurde ein Prototyp erstellt, um herauszufinden wie einfach sich eine Landschaft mit JavaFX implementieren lässt. Dieser Prototyp wurde durch das ganze Projekt weiterverwendet und entwickelt.

Das Einpflegen hat sich als schwieriger als erwartet herausgestellt, da das gesamte System auf Snapshots aufbaut. Diese Snapshots haben uns dazu gezwungen diverse Änderungen zur Verbesserung der Performance vorzunehmen, am besten für die Landschaft wäre aber die Snapshots komplett zu entfernen, da für jeden Snapshot eine Klasse Namens Landscape Module vorhanden sein muss, die eine Landschaft beinhaltet. Wir haben es geschafft, dass das Triangle Mesh bei jeder Landschaft dasselbe ist, was der Performance immens geholfen hat. Aber diese Dreiecke nur einmal anzuzeigen würde die Geschwindigkeit des Programmes um einiges verbessern.

Literaturverzeichnis

Trentini, M., & Wieland, M. (2018). *Algorithm & Data Structure Visualizer Benutzerhandbuch.* Rapperswil.

Winter, J., & Meier, F. (2018). *ADV-Tree-Module.* Rapperswil.

Tabellenverzeichnis

Tabelle 1 Übersicht über die Meilensteine in dieser SA 9
Tabelle 2 Beschreibung der Risiken 10

Abbildungsverzeichnis

Abbildung 1 Klassendiagramm über die neuen Klassen des ADV Landscape Modules 15
Abbildung 2 Auswertung der geleisteten Stunden pro Woche 19
Abbildung 3 Verbuchte Stunden pro Aufgaben-Kategorie und Person 20

Administrative Anhänge

Projektplanung

Phase	Inception	Elaboration				Construction								Transition
Iteration	1	2		3		4		5		6		7		8
Woche	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Datum	19.09	26.09	03.10	10.10	17.10	24.10	31.10	07.11	14.11	21.11	28.11	05.12	12.12	20.12
Meilenstein	M0		M1		M2				M3		M4		M5	M6
Artefakte														
Besprechungsprotokolle														x
Artefakt-Übersicht														x
Wireframes														
Projektplan	x													
Anforderungsspezifikation		x												
Evaluation			x											
Architektur- und Designspezifikation					x									
Prototyp					x									
Systemtests Definitionen					x									
Systemtests Protokolle													x	
ADV Release													x	
ADV-Landscape-Module													x	
Abstract														x
Management Summary														x
Technischer Bericht														x
Benutzerhandbuch														x
Metriken														x
Zeitauswertung														x
Glossar und Abkürzungsverzeichnis														x
Selbstständigkeitserklärung														x
Persönliche Berichte														x

M0:	M1:	M2:	M3:	M4:	M5:	M6:
Vision	Projektplan und Anforderungsspezifikation	Architektur- und Designspezifikation und Prototyp	MVP	Release 1	Release 2	Dokumentation

Zeitabrechnung

Wenn wir die geleisteten Stunden auf die entsprechenden Wochen aufteilen, sind diese Stunden insgesamt gleichmässig verteilt. Zu Beginn sowie auch am Ende der Arbeit ist weniger Zeit eingetragen, da die Auswertung nicht von Tag 1 bzw. nicht bis zum letzten Arbeitstag geht. In den letzten Wochen fielen vor allem noch Dokumentation und kleinere Bugfixes an, so dass weniger Stunden anfielen als z.B. mitten im Projekt.

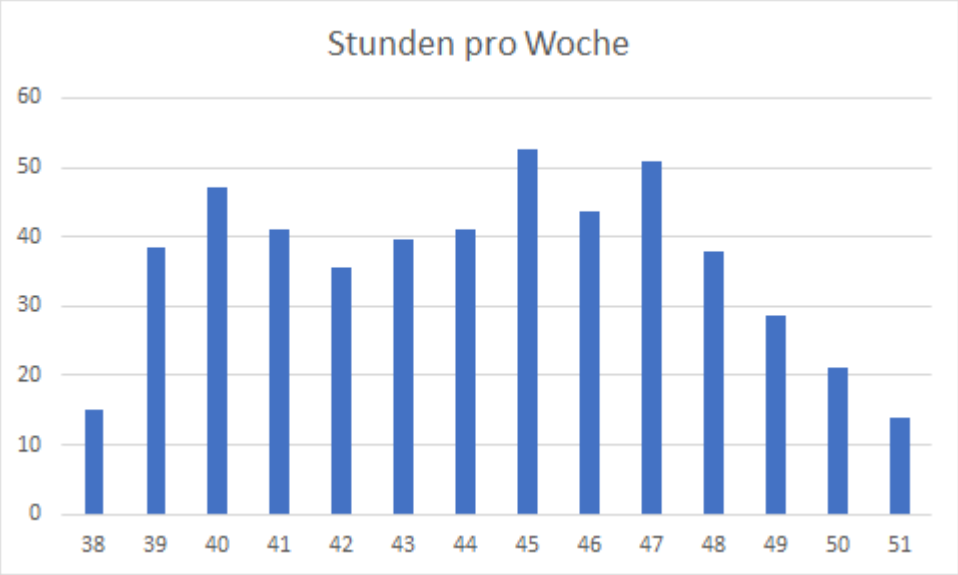

Abbildung 2 Auswertung der geleisteten Stunden pro Woche

Die abgeschlossenen Aufgaben können in folgende Kategorien aufgeteilt werden:

- Bugfixing: Behebung von Fehlern sowie Verbesserung der Performance
- CI: Einrichtung von Continuous Integration
- Dokumentation: Benutzer-Handbuch sowie Studienarbeits-Dokumentation
- Einarbeitung: Erarbeitung der JavaFX-/3D-Grundlagen
- Features: Implementationen im ADV
- Meeting: Wöchentliche Sitzungen mit dem Betreuer
- Planung: Planen des weiteren Vorgehens
- Prototyp: Erstellung einer separaten Applikation mit einer 3D-Landschaft
- Refactoring: Optimierungen/Umstrukturierungen des Source Codes
- Release: Bereitstellung der Dateien für die jeweiligen Releases
- Andere: Aufgaben, welche keiner anderen Kategorie zugeordnet werden können

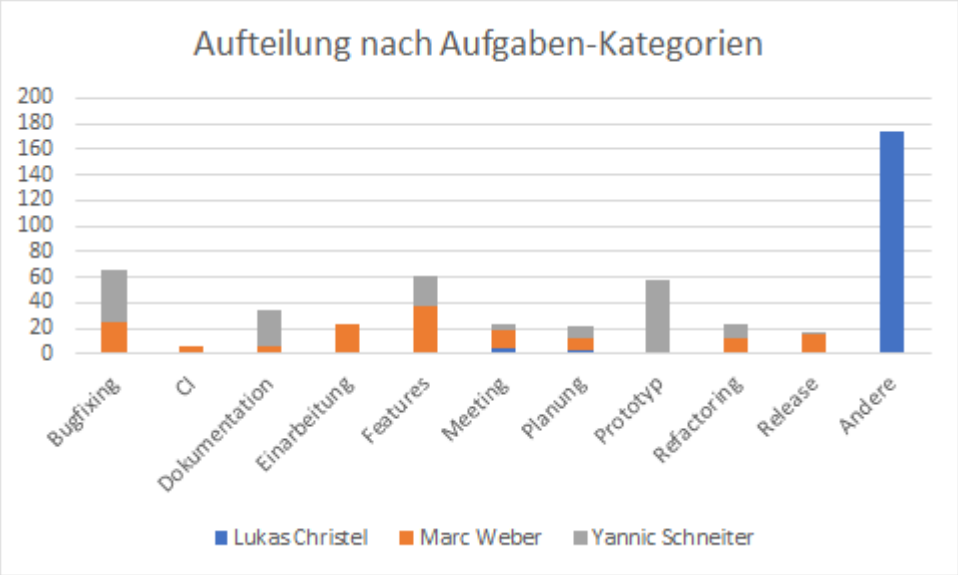

Abbildung 3 Verbuchte Stunden pro Aufgaben-Kategorie und Person