Astrodynamics

Anleitung und Dokumentation

Version 0.4

Marc Singer, Rafael Stauffer

26.02.2023

Versionshistorie

Version	Datum	Autor(en)	Änderungen
0.1	21.01.2023	RS	Erstellung
0.2	23.02.2023	RS	Einfügen Ziel und Zweck
0.3	25.02.2023	RS	Einfügen Nutzeranleitung
0.4	26.02.2023	MS	Technische Dokumentation

Inhaltsverzeichnis

Kapitel 1

Ziel und Zweck

Dieses Projekt hat das Ziel die teilweise kontraintuitiven Gesetzmässigkeiten welche im Weltraum gelten greifbar zu machen. Der Author der primären Inspirationsquelle "Children of a Dead Earth"[1] schreibt dazu passend:

> For me, though, I wanted a simulation, one that was actually based on real equations. This is because in my experience, whenever you develop system this complex, it tends to surprise you, and will often overturn your assumptions.

Da wir als Team bisher keine Erfahrungen im Bereich von Physiksimulationen hatten wurde als Ziel die Realisierung einer N-Körper-Simulation festgelegt welche zukünftig erweitert werden kann.

[1] qswitched. *Children of a Dead Earth Origin Stories*. https://childrenofadeadearth.wordpress.com/2016/05/06/origin-stories/. Accessed: 2023-02-22. 2016.

Kapitel 2

Benutzeranleitung

2.1 Missionsliste

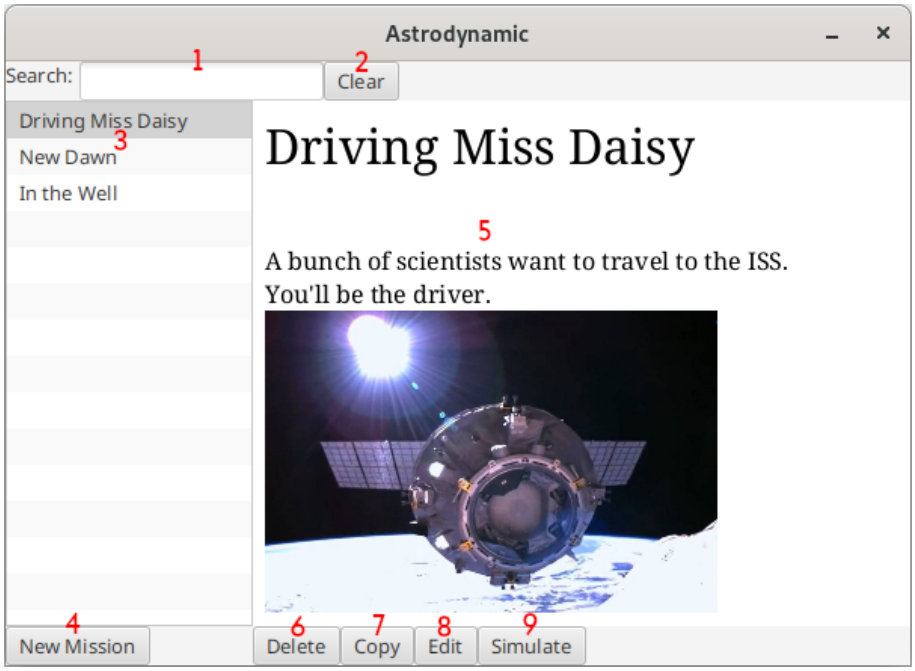

Abbildung 2.1: GUI Missionsliste mit Annotation

1. Suchfeld
2. Clear: Suchfeld leeren
3. Liste verfügbarer Missionen
4. New Mission: Neue Mission öffnen im Missions-Editor
5. Beschreibung der ausgewählten Mission
6. Delete: Ausgewählte Mission löschen
7. Copy: Ausgewählte Mission kopieren
8. Edit: Ausgewählte Mission öffnen im Missions-Editor
9. Simulate: Ausgewählte Mission öffnen im Simulator

2.1.1 Grundlagen

Die Missionsliste ist der Einstiegsbildschirm beim Start des Programs. Hat der Benutzer keine Mission gespeichert welche geladen werden kann so werden drei Testmissionen geladen. Am linken Rand befindet sich die Liste der verfügbaren Missionen.Anwählen einer Mission in der Liste per Klick mit der Linken

Maustaste lädt die Missionsbeschreibung in den rechten Anzeigebereich und ermöglicht mit diese Mission per Buttons unten rechts am Bildschirmrand weiter zu Interagieren.

2.1.2 Missionen nach Beschreibung suchen

Das Suchfeld führt eine sofortige Textsuche auf Missions-Name und -Beschreibung durch und zeigt auf Basis dieser nur passende Missionen in der Liste der verfügbaren Missionen. Durch drücken des Clear-Buttons können längere Sucheingaben sofort gelöscht und die Sortierung zurückgesetzt werden.

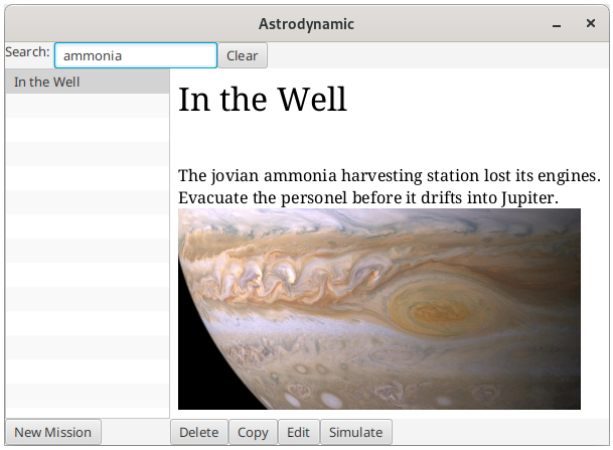

Abbildung 2.2: Missionsfilter bei Suche nach 'ammonia'

2.1.3 Anlegen einer neuen Mission

Klicken sie auf den "Neue Mission öffnen im Missions-Editor"-Button unten links. Es öffnet sich nun der Missions-Editor. Für Details zum editieren einer Mission konsultieren sie das Kapitel Missions-Editor.

2.1.4 Löschen einer Mission

Wählen sie die Mission aus der Liste der verfügbaren Missionen per Mausklick aus. Klicken sie auf Delete. Ein Popup öffnet sich mit der Löschanfrage.

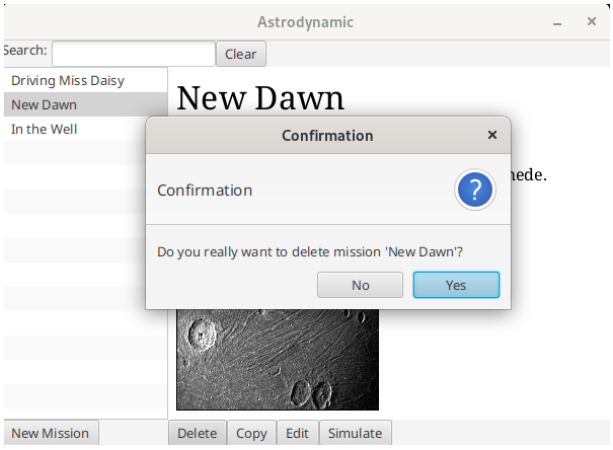

Abbildung 2.3: Sicherheitsabfrage bei Missionslöschung

Bestätigen sie das Popup mit Klick auf Yes. Die Mission wird aus der Liste der verfügbaren Missionen entfernt.

2.1.5 Kopieren einer Mission

Wählen sie die Mission aus der Liste der verfügbaren Missionen per Mausklick aus. Klicken sie auf Copy. Es öffnet sich nun der Missions-Editor mit der kopierten Mission. Für Details zum editieren einer Mission konsultieren sie das Kapitel Missions-Editor.

2.1.6 Editieren einer Mission

Wählen sie die Mission aus der Liste der verfügbaren Missionen per Mausklick aus. Klicken sie auf Edit. Es öffnet sich nun der Missions-Editor mit der ausgewählten Mission. Für Details zum editieren einer Mission konsultieren sie das Kapitel Missions-Editor.

2.1.7 Simulieren einer Mission

Wählen sie die Mission aus der Liste der verfügbaren Missionen per Mausklick aus. Klicken sie auf Simulate. Es öffnet sich nun der Simulator mit der ausgewählten Mission. Für Details zum simulieren einer Mission konsultieren sie das Kapitel Simulator.

2.2 Missions-Editor

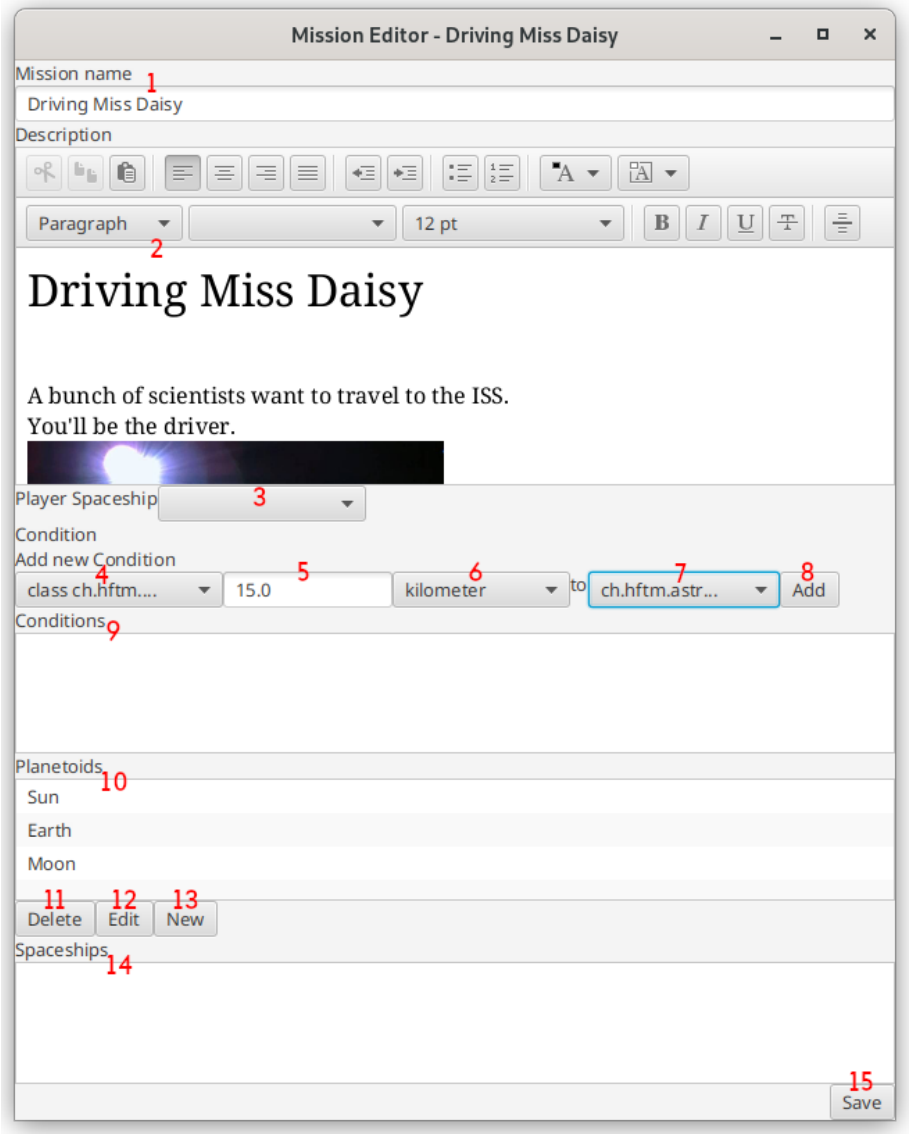

Abbildung 2.4: GUI Mission-Editor mit Annotation. Testmission 'Driving Miss Daisy' geöffnet.

1. Missionsname
2. Missionsbeschreibung HTML-Editor
3. Auswahl Spielerraumschiff
4. Missionsbedingung: Bedingungstyp-Dropdown
5. Missionsbedingung: Zahlenfeld
6. Missionsbedingung: Masseinheit-Dropdown
7. Missionsbedingung: Referenzobjekt-Dropdown
8. Missionsbedingung hinzufügen
9. Missionsbedingungen-Liste
10. Planetoiden-Liste
11. Planetoid entfernen
12. Planetoid editieren
13. Planetoid hinzufügen
14. Raumschiff-Liste
15. Missionsänderungen speichern

2.2.1 Grundlagen

Der Missions-Editor erlaubt das Ändern des Missions-Namen und Beschreibung. Durch das Hinzufügen von Missionsbedingungen, auch Conditions genannt, können Abbruchsbedinungen für die Simulation festgelegt und weitere dynamische Veränderungen an der Mission vorgenommen werden. Verwendete Planetoiden und Raumschiffe werden in den ensprechenden Listen aufgelistet.

2.2.2 Hinzufügen einer Missionsbedingung

Wählen sie im Bedingungstyp-Dropdown den passenden Bedingungstypen. Siehe Abschnitt Missionsbedingen für eine komplette Liste der möglichen Bedingungstypen, ihren Parametern, und Funktionsweise. Zahlenfeld, Grössenangabe und Referenzobjekt werden dynamisch ein- oder ausgeblendet.

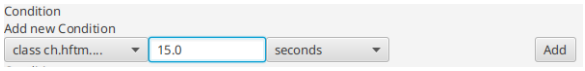

Abbildung 2.5: MaximumTime

Abbildung 2.6: Approach

Sollte das Zahlenfeld oder das Referenzobjekt eingeblendet werden so ist eine Eingabe respektive Auswahl zwingend. Die Masseinheit kann jederzeit geändert werden, ein valider Wert im Zahlenfeld wird in die neue Einheit umgewandelt.

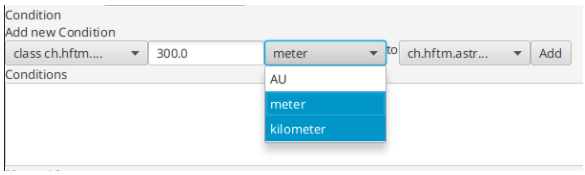

Abbildung 2.7: Masseinheit Meter

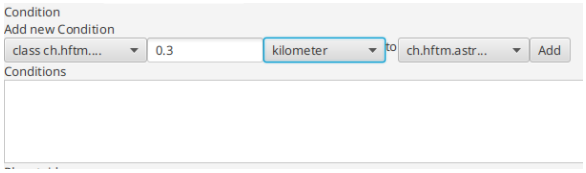

Abbildung 2.8: Masseinheit Kilometer

Klicken sie auf den Add-Button um die Missionsbedingung hinzuzufügen. Die Missionsbedingung wird nun in der Missionsbedingungen-Liste aufgeführt.

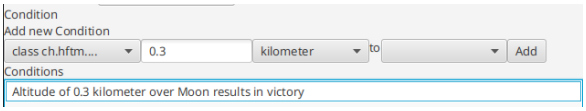

Abbildung 2.9: Neue Approach-Missionsbedingung hinzugefügt

2.2.3 Missionsbedingungen

Bedingung	Parameter	Funktionsweise
MaximumTime	Zeitwert	Mission gilt als Fehlschlag wenn Missionsdauer den Zeitwert überschreitet
HoldoutTime	Zeitwert	Mission gilt als Gewonnen wenn Missionsdauer den Zeitwert überschreitet
Approach	Distanz und Referenzobjekt	Mission gilt als Gewonnen wenn Spielerraumschiff die maximale Distanz zum Referenzobjekt erreicht oder unterschreitet
Avoid	Distanz und Referenzobjekt	Mission gilt als Fehlschlag wenn Spielerraumschiff die maximale Distanz zum Referenzobjekt erreicht oder unterschreitet
Depart	Distanz und Referenzobjekt	Mission gilt als Gewonnen wenn Spielerraumschiff die minimale Distanz zum Referenzobjekt erreicht oder überschreitet
SetupHeavyLander	Distanz und Referenzobjekt	Platziert das Raumschiff 'Heavy Lander' in ein Orbit um Referenzobjekt mit Höhe von Distanz
SetupISS	Distanz und Referenzobjekt	Platziert das Raumschiff 'ISS' in ein Orbit um Referenzobjekt mit Höhe von Distanz

Tabelle 2.1: Verfügbare Missionsbedingungem

2.2.4 Löschen eines Planetoiden

Wählen sie den zu löschenden Planetoiden mit einem Klick aus der Planetoiden-Liste aus. Klicken sie den Delete-Button. Es öffnet sich eine Sicherheitsabfrage.

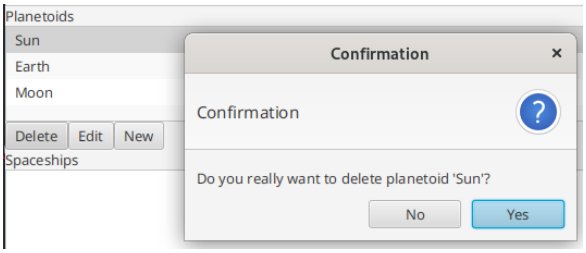

Abbildung 2.10: Sicherheitsabfrage bei Planetoidenlöschung

Bestätigen sie das Popup mit Yes. Der Planetoid ist nun aus der Mission entfernt und die Planetoiden-Liste aktualisiert worden.

2.2.5 Editieren eines Planetoiden

Wählen sie den zu editierenden Planetoiden mit einem Klick aus der Planetoiden-Liste aus. Klicken sie auf den Edit-Button. Es öffnet sich nun der Planetoid-Editior mit dem ausgewählten Planetoiden. Für Details zum Planetoid-Editior konsultieren sie das Kapitel Planetoid-Editior.

2.2.6 Hinzufügen eines Planetoiden

Klicken sie auf den New-Button unterhalb der Planetoiden-Liste. Es öffnet sich nun der Planetoid-Editior. Für Details zum Planetoid-Editior konsultieren sie das Kapitel Planetoid-Editior.

2.2.7 Hinzufügen eines Raumschiffs

Zum hinzufügen eines Raumschiffs benutzen sie die Condition 'SetupHeavyLander'. Siehe Abschnitt Hinzufügen einer Missionsbedingung.

2.3 Planetoid-Editor

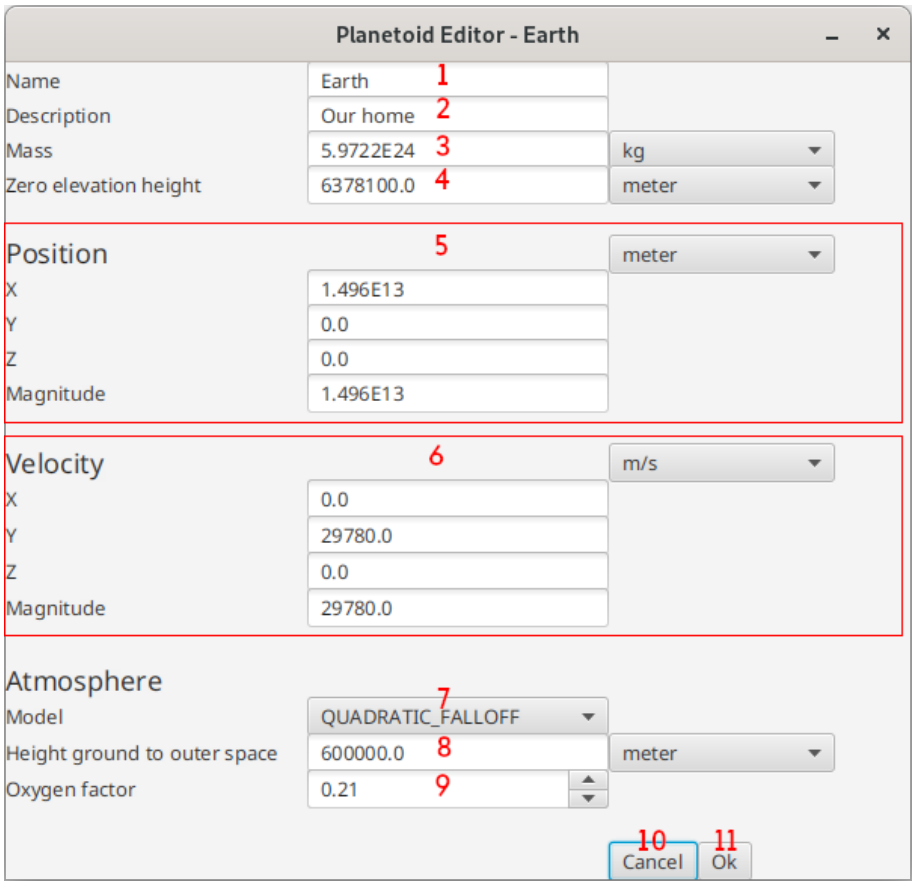

Abbildung 2.11: GUI Planetoid-Editor mit Annotation. Planetoid Erde geöffnet.

1. Planetoidname
2. Kurzbeschreibung
3. Masse mit Grössenumrechnung
4. Nullpunkthöhe mit Grössenumrechnung
5. Positionsdaten
6. Geschwindigkeitsdaten
7. Atmosphärenmodel
8. Atmosphärenhöhe bis Vakuum mit Grössenumrechnung
9. Sauerstoffanteilsfaktor
10. Cancel: Bearbeitung ohne Speicherung abbrechen
11. Ok: Speichern und schliessen

2.3.1 Grundlagen

Der Planetoid-Editor erlaubt das editieren der Simulationswerte eines Planetoiden. Position und Geschwindigkeit können direkt in den jeweiligen Vektordimensionen (X, Y, Z) gepflegt und bei Bedarf mit dem Längen-Feld (Magnitude) skaliert werden. Für übersichtlichere Darstellung können die Zahlenwerte dynamisch mit dem danebenstehenden Grössenumrechnungs-Dropdown umgewandelt werden. Eingaben werden beim Speichern überprüft. Im Fehlerfall wird die Speicherung verhindert und mit dem Feldverweis and den Benutzer gemeldet.

2.3.2 Editieren eines Vektors

Werden die Dimensionswerte angepasst, so wird die Länge automatisch aktualisiert.

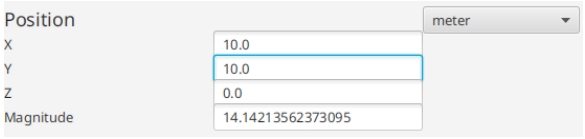

Abbildung 2.12: Kalkulation der Länge Ausgangslage

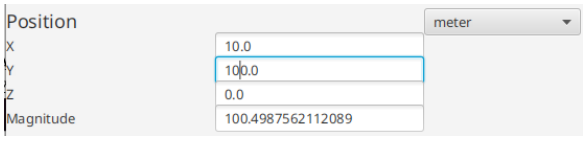

Abbildung 2.13: Kalkulation der Länge nach Anpassung Y-Wert

Wird die Länge angepasst, so werden die Dimensionswerte ensprechend skaliert.

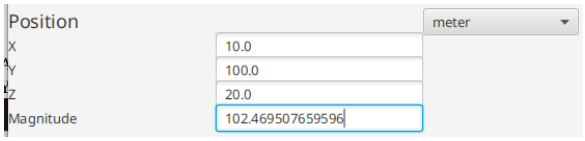

Abbildung 2.14: Vektor-Skalierung der Dimensionswerte Ausgangslage

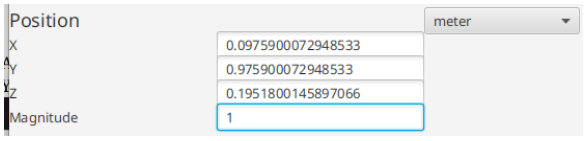

Abbildung 2.15: Vektor-Skalierung der Dimensionswerte nach Anpassung der Länge

Die Grösse kann über das Dropdown am rechten Rand auf Höhe der Vektorbezeichnung ausgewählt werden. Gültige Werte werden automatisch umgewandelt.

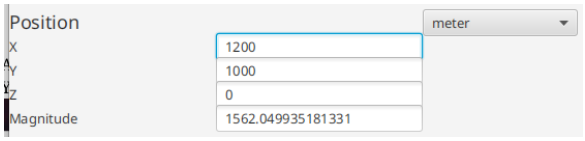

Abbildung 2.16: Vektor-Umwandlung der Grösse Ausgangslage

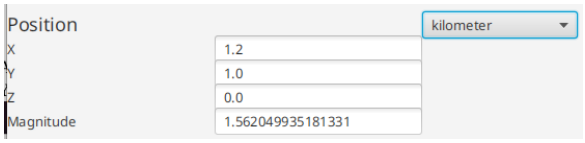

Abbildung 2.17: Vektor-Umwandlung der Grösse nach anpassen der Grösse

2.3.3 Atmosphäreneinstellungen

Feature eingeplant aber unimplementiert in aktueller Version

Das Atmosphärenmodel bestimmt die Berechnungsformel für die Atmosphärendichte. Die Atmosphärenhöhe bis Vakuum skaliert das Model entsprechend, dass auf der angegebenen Höhe über der Nullpunkthöhe die Formel unter den im Programmcode definierten Minimalwert fällt. Der Sauerstofffaktor wird zur Berechnung von luftatmenden Triebwerken benötigt. Eine Atmospäre ohne Sauerstoff gilt als Inert und kann nur von Triebwerken verwendet werden welche keinen Sauerstoff benötigen.

2.3.4 Speichern des Planetoiden

Klicken sie auf Ok. Ist eine Eingabe fehlerhaft so erscheint eine Fehlermeldung mit der Feldbezeichnung und Fehlerbeschreibung.

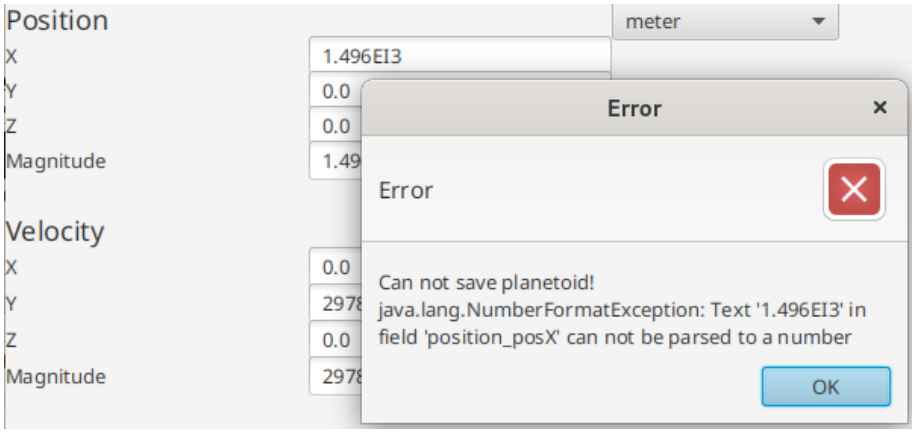

Abbildung 2.18: Planetoid-Editor Fehlermeldung: Buchstabe 'I' im Zahlenfeld

Können alle Eingaben verarbeitet werden, wird der Planetoid gespeichert und der Planetoid-Editor wird geschlossen.

2.4 Simulator

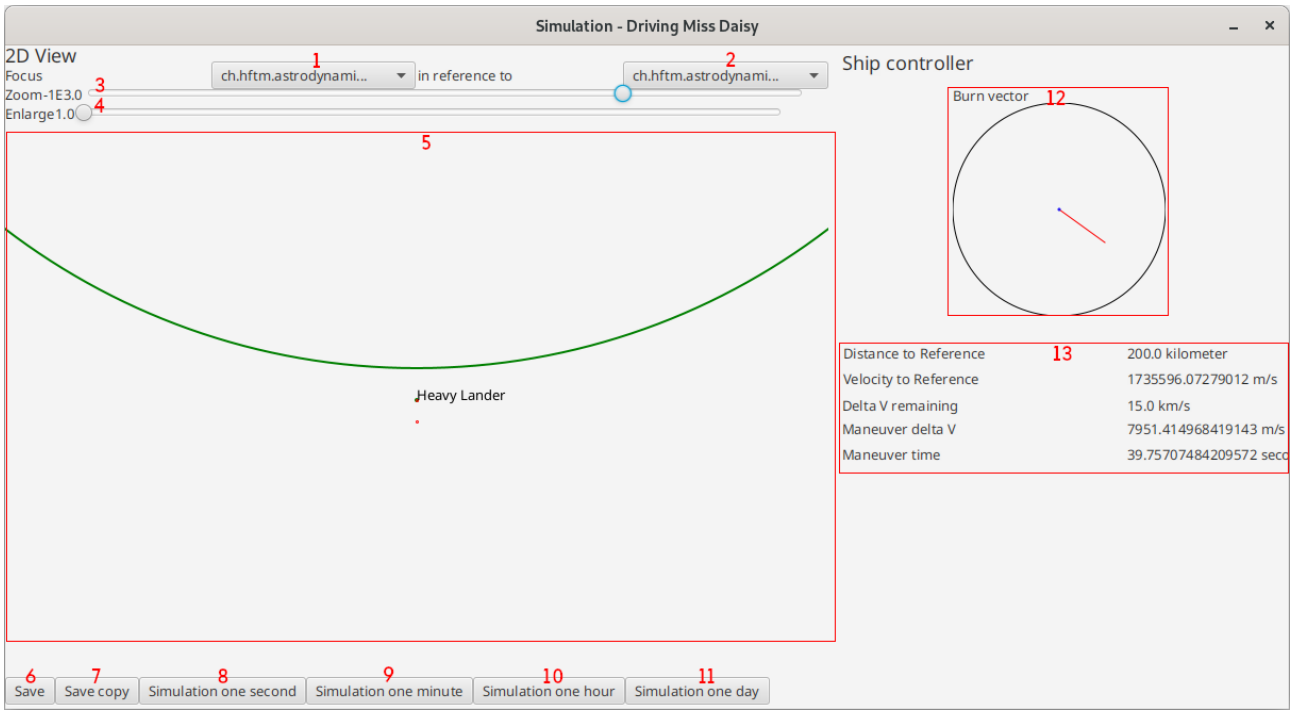

Abbildung 2.19: GUI Simulation mit Annotation. Heavy Lander im Fokus, ISS als Referenz

1. Fokus-Objekt
2. Referenz-Objekt
3. Zoom
4. Objektvergrösserung
5. Orbital-View
6. Save: Speichern der Simulation
7. Save copy: Kopieren der Simulation
8. Simulation eine Sekunde berechnen
9. Simulation eine Minute berechnen
10. Simulation eine Stunde berechnen
11. Simulation einen Tag berechnen
12. Burn vector: Maneuver-Darstellung
13. Informationspanel

2.4.1 Grundlagen

Der Simulator erlaubt eine simple graphische Darstellung der Simulation. Die linke Seite des Fensters wird dabei von der Orbital-View, auch 2D-View genannt, eingenommen welche ein ortographisches Bild der Objekte darstellt. Mit Zoom und Objektvergrösserung können kleinere und grössere Objekte aufgefunden und zentriert werden. Die rechte Seite des Fensters wird beim Fokus eines Raumschiffs mit der Maneuver-Darstellung welche das geplante Maneuver graphisch darstellt sowie dem Informationspanel gefüllt.

Maneuver-Vektor Der schwarze Kreis stellt die maximal verfügbare Geschwindigkeitsänderung welche das aktuell fokussierte Raumschiff durchführen kann dar. Bei einem aktiven Maneuver wird eine rote Linie in die Richtung in welche die Geschwindigkeitsänderung durchgeführt wird gezogen. Die Linienlänge stellt den benötigten Wert an Geschwindigkeitsänderung gegenüber dem verfügbaren

Vorrat (schwarzer Kreis) dar und ändert sich deshalb bei der Durchführung des Maneuvers. Ein Befehl über dem verfügbaren Vorrat ist möglich, es wird jedoch nur bis zum verfügbaren Vorrat durchgeführt.

2.4.2 Navigieren im Orbital-View

Wählen sie das gewünschte Objekt im Fokus-Dropdown aus. Die Orbital-View fokussiert nun auf das eingestellte Objekt. Wählen sie das gewünschte Objekt im Referenz-Dropdown aus. Informationen über Distanz und Geschwindigkeit relativ zur Referenz werden nun im Informationspanel ausgegeben. Stellen sie die Zoom-Stufe und eventuelle Objektvergrösserung auf das gewünschte Level ein. Für Raumschiffe in einem erdnahen Orbit empfielt sich eine Zoomstufe von *-1E3.0* und eine Objektvergrösserungsfaktor von *1.0*.

2.4.3 Ausführen eines Maneuvers

Fokusieren sie ein Raumschiff. Klicken sie in den schwarzen Kreis der Maneuver-Darstellung. Eine rote Linie vom Zentrum zu ihrer angeklickten Position erscheint. Auf dem Informationspanel wird die geplante Geschwindigkeitsänderung (Bezeichnung *Maneuver delta V*) und die Zeitdauer zum Durchführen des Maneuvers (Bezeichnung *Maneuver time*) angegeben.
Klicken sie nun auf einen beliebigen Simulationsbutton bis die Zeitdauer in der Simulation vergangen ist. Nach ausführen des Maneuvers wird die rote Maneuver-Linie entfernt und die Maneuver-Informationen auf dem Informationspanel ausgeblendet.

2.4.4 Speichern des Simulationszustandes

Klicken sie auf den Save-Button. Die Mission ist nun gespeichert. Eine Informationspopup erscheint zur Bestätigung.

Kapitel 3

Technische Dokumentation

3.1 Applikationsdokumentation

3.1.1 Diagramme

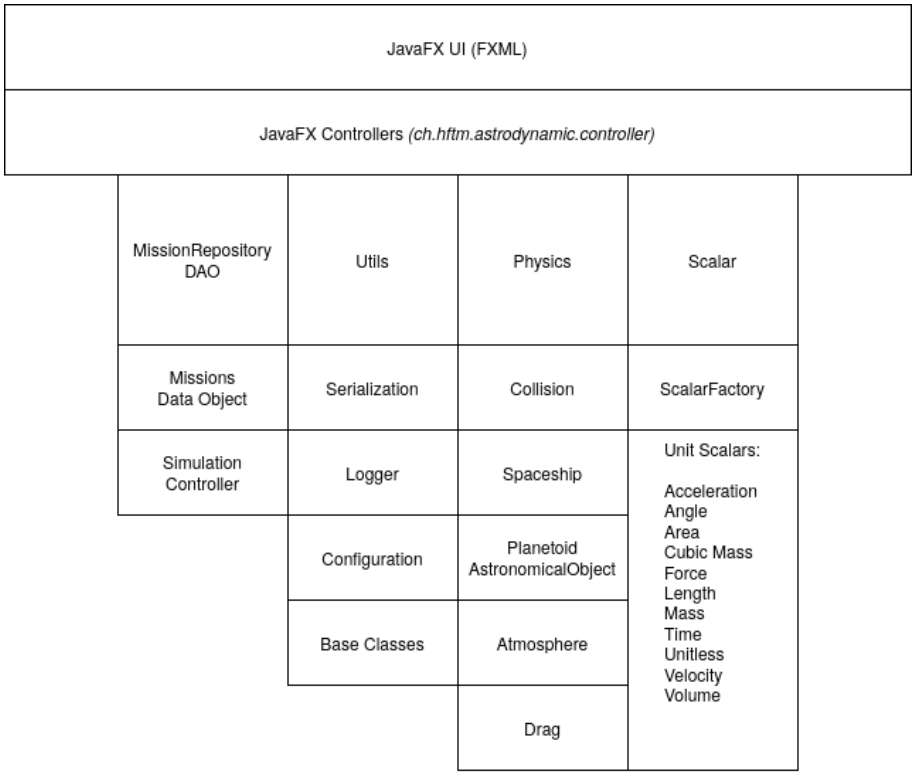

Abbildung 3.1: Übersicht Aufbau Astrodynamic

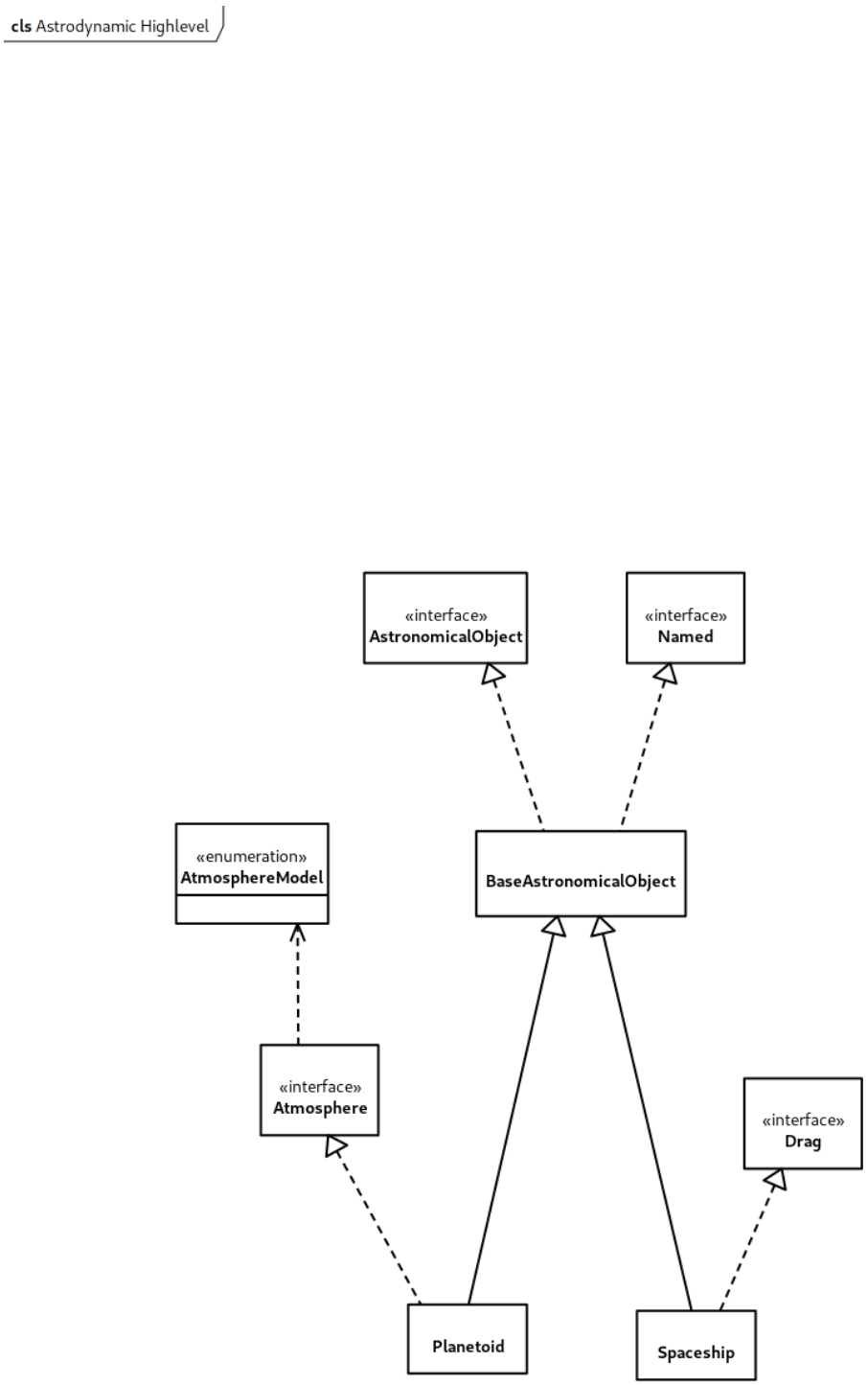

Abbildung 3.2: Übersicht Physik-Klassen und -Interfaces

Das BaseAstronomicalObject bietet die Basis für die Physikberechnung. Spezifische Interfaces wie die Atmosphärendaten für die Planeten oder Drag-Berechnungen für die Raumschiffe werden in den jeweiligen Kindklassen implementiert.

cls Astrodynamic Dimensions

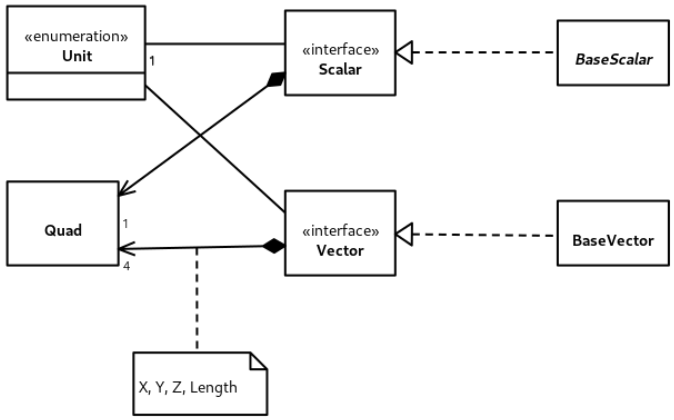

Abbildung 3.3: Übersicht Mathematische-Klassen und -Interfaces

Scalare und Vektoren verwenden die Quad-Klasse um exakte Werte zu speichern. Um die Dimensionsanalyse in den spezifischen Skalarimplementationen (nicht im Diagram aufgeführt) durchzuführen wird der Enum Unit verwendet.

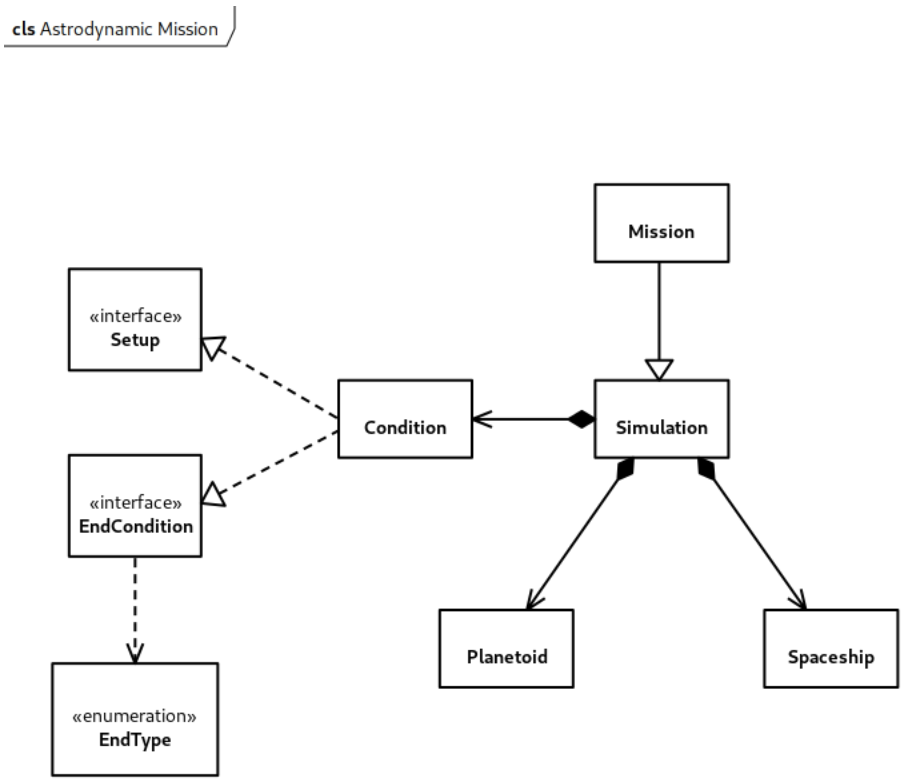

Abbildung 3.4: Übersicht Missions-Klassen und -Interfaces

Eine Simulation besteht aus den dazugehörigen Planetoiden und Raumschiffen. Diese sind getrennt verwaltet für einfacheren Zugriff auf spezifische Funktionen, jedoch bietet die Simulation ebenfalls Methoden diese Objekte in einer Liste abzurufen, je nach Ziel und benötigtem Interface.
Conditions in der Simulation manipulieren ihren Startzustand und können Abbruchbedingungen definieren und überprüfen.
Die Mission stellt einen Wrapper dar welcher weitere Funktionalitäten zur Verfügung stellt welche in einem reinen Simulationskontext nicht benötigt werden.

3.1.2 Maven build

Um mittels Maven ein Build zu erzeugen kann der Befehl *mvn clean package* verwendet werden.

Dieser Befehl testet das gesamte Softwareprojekt und erzeugt anschliessend im Ordner *target* ein neues Java Executable (JAR File).

Docker Wir haben ebenfalls einen Docker Container vorbereitet, mit welchem das Build automatisiert werden kann.

Dieser nutzt das Zulu OpenJDK, um unser Projekt vollständig zu bauen und zu testen:

```
# Docker build
$ docker build -t astrodynamic/build .
$ docker run --name astrodynamic -v ./:/astrodynamic astrodynamic/build
# Das Java Executable erscheint im selben Ordner
```

Mit Docker können wir das Projekt überall mit wenig Aufwand und ohne grossen Installationsaufwand neu bauen.

3.2 Themenumsetzung

3.2.1 Unit Tests

Unsere Unit Tests befinden sich im Ordner *src/test/java/ch/hftm/astrodynamic* und sind nach Anwendungskomponenten gegliedert:

- DAO (Data Access Object); Testet den Zugriff und die Fuktionalität des zentrale MissionRepositorys (in welchem alle Missionen und deren Kindobjekte gespeichert sind)
- Model; Testet die pedefinierten Modelle (z.B Mond, Erde, Sonne) welche mit der Software mitgeliefert werden
- Planetoid; Testet die Funktionalität der Planetoiden (also planetenähnlichen Objekten)
- Quad; Testet die Implementation und Abstraktion unseres Quad Datentyps, welcher für alle Berechnungen innerhalb des Projekts verwendet wird
- Scalar; Testet die Richtigkeit von Scalar basierten Verrechnungen
- SimulationTest; Testet die Simulation der einzelnen Objekte im Sonnensystem
- Unit; Testet die verschiedenen Scalar Physik-Einheiten und deren Verrechnungen
- Vector; Testet die Vektoren und deren Verrechnungen

Wir haben uns primär dafür entschieden, die Business-Logik (primär die Simulation) zu testen, da die Erfolg unseres Projekts primär von der Korrektheit der Simulation abhängig ist.

Unit Tests ausführen Zunächst müssen wir uns ins *astrodynamics* Verzeichnis bewegen in welchem ebenfalls die Datei *pom.xml* gespeichert ist. Anschliessend können wir die Unit-Tests mit einem einfachen *mvn clean test* ausführen.

3.2.2 Enumeration

Unser wichtigster Einsatz von Enum (Enumeration) findet in der Klasse *Unit* statt. Diese Klasse wird dazu genutzt, SI Einheiten, welche in unserem Projekt verwendet werden, darzustellen:

```
// Contains all physical units as enum
public enum Unit {
        TIME, // Seconds (s)
        LENGTH, // Meters (m)
        MASS, // Kilogram (kg)
        CURRENT, // Ampere (A)
        TEMPERATURE, // Kelvin (K)
        MOLECULES, // Mol (mol)
        LUNINOSITY, // Candela (cd)
        // Extensions (Implicit)
        VOLUME,  // m ^ 3
        AREA, // m ^ 2
        FORCE, // N
        ACCELERATION, // m/s ^ 2
        VELOCITY, // m/s
        ANGLE, // radian
        ANGULAR_VELOCITY, // radian/second
```

```
        ANGULAR_ACCELERATION, // radian/second^2
        // kg ^ 2 imaginary unit for gravitational calculations
        CUBIC_MASS,
        // kg ^ 2 / m ^ 2 imaginary unit for gravitational calculations
        M2_DIV_L2,
        // N * m ^ 2 * kg ^ 2 gravitational constant
        //  (s ^ -2 * m ^ 3 * kg ^ - 1)
        F_L2_Mn2,
        // Unitless for scalars without unit
        UNITLESS
    }
```

Wir nutzen dieses Enum in allen Skalaren und in diversen Vektoren, um die Einheiten der Werte darzustellen. z.B nutzt die Klasse *LengthScalar*, welche eine Länge speichert, den Wert *LENGTH*.

In den Kommentaren der Klasse werden die genauen Einheiten explizit definiert. Dies hilft bei den Umrechnungen der Werte z.B zwischen verschiedenen Zeiteinheiten (z.B zwischen Stunden, Tagen oder Wochen).

3.3 Vererbung

Wir nutzen Vererbung in allen unseren Klassen. Spezifisch hervorheben möchten wir in diesem Fall die *Scalar* Klassen im Package *ch.hftm.astrodynamic.scalar*, welche allesamt von der Klasse *BaseScalar* im Package *ch.hftm.astrodynamic.utils* erben.

Die einzelnen *Scalar* Klassen stellen jeweils einen spezialisierten Scalar im Projekt dar. z.B werden Längen mittels dem *LengthScalar* dargestellt. Die Einheiten der jeweiligen Klassen werden in der Kind-Klasse innerhalb des Konstruktors statisch gesetzt und dem Eltern-Konstruktor des *BaseScalar* übergeben und gespeichert.

3.4 Casting

Da wir für die Serialisierung einen eigenen Serializer / Deserializer innerhalb der Klasse *MissionRepository* in *ch.hftm.astrodynamic.utils* implementieren mussten (die Klasse *ObservableList* ist nicht Serialisierbar), verwenden wir Casting in diesem Fall für die Konvertierung zwischen *Object* und *Mission*.

Casting wird ausserdem in einzelnen Fällen innerhalb der Controller verwendet, um z.B einen Double zu einem Integer zu Runden.

3.5 Interfaces

Interfaces werden in unserem Projekt häufig verwendet. Vorwiegend haben wir mittels Interfaces generalisierte Klassen (und deren Vererbungen) implemtiert.

Als Beispiel dient hier das Interface *Scalar* aus dem Package *ch.hftm.astrodynamics.utils*, welches z.B für generalisierte Operationen innerhalb aller Klassen verwendet wird. Mit diesem Interface können wir sicherstellen, dass alle *Scalar* Klassen verglichen oder berechnet werden können.

Ebenfalls ist in diesem Interface eine Implementation für die Funktion *toFittedString()* bereits vorgegeben (und muss aus diesem Grund nicht erneut implementiert werden).

3.6 Abstrakte Klassen

Wir nutzen Abstrakte Klassen unter anderem für unseren *BaseController* im Paket *ch.hftm.astrodynamic.controller* welcher grundlegende Funktionalität für unsere Controller bereitstellt.

Dieser Controller dient als Elternklasse für unsere *Controller* Klassen, wird aber selbst niemals instanziert (daher ist er als Abstract definiert).

Wir verwenden abstrakte Klassen ebenfalls für die *Condition* Klasse für die Mission aus demselben Grund.

3.7 Collection

Der wichtigste Einsatz von Collections in unserem Projekt ist die *ObservableList* innerhalb unseres MissionRepositories. Dieser Typ der Collection stammt aus dem JavaFX Paket und bietet der Frontend-Anwendung die Möglichkeit, den Inhalt der Liste konstant zu überwachen und Änderungen im UI umgehend darzustellen.

Sortierung Wir sortieren die *ObservableList* innerhalb der Funktion *sort()* innerhalb unserer Klasse *MissionRepository* alphabetisch nach Namen der Mission:

```
// Sorts the MissionRepository alphabetically
public static void sort() {
        // Create a comparator for name based sorting
        Comparator<Mission> byName = (Mission a, Mission b) ->
        a.getName().compareTo(b.getName());
        // Sort the Mission Repository based on names
        getInstance().missions.sort(byName);
}
```

Der Name ist ebenfalls der Primärschlüssel, welcher für die Missionen innerhalb der Funktion *getMissionByName* verwendet wird.

3.8 Serialisierung

Serialisierung wird in unserem Projekt zur Speicherung der Klasse *MissionRepository*, welche den kompletten Status unserer Anwendung enthält verwendet.

Die Klasse *Serializer* ist für die Serialisierung des *MissionRepository* zuständig. Aktuell wird die Serialisierbarkeit unserer Klassen mittels dem Interface *Serializable* gewährleistet. Dies ist insbesondere für die Komplexen Datentypen, welche für *Quad* und *ObservableList* verwendet werden, wichtig. Da *ObservableList* das Interface *Serializable* nicht implementiert, musste im *MissionRepository* mittels der Funktionen *WriteObject()* und *ReadObject()* die Missionen in ein Array konvertiert werden. Dieses Array wird beim Deserialisieren wiederum in Missionen gecasted, welche dem *MissionRepository* hinzugefügt werden.

Ebenfalls wurde eine mögliche Serialisierung ins JSON-Format vorbereitet. Dazu verwenden wir die Bibliothek *Jackson*, welche ebenfalls in diversen Java Web Bibliotheken verwendet wird und als performant gilt.

3.9 Speichern und Laden der Missionen

Laden Beim Start der Anwendung werden die gespeicherten Missionen aus einer Datei geladen. Der Pfad dieser Datei kann mittels der Konfiguration *STATE FILE* angepasst werden. Der Standardname der Datei lautet: *state.bin*. Diese Datei kann ebenfalls gelöscht werden, um den Ursprungszustand wiederherzustellen.

Speichern Wenn Missionen geändert werden, werden diese automatisch in die (aktuell konfigurierte) Projektdatei geschrieben.

3.10 Exceptions

Mit dem *UnitConversionError* im *ch.hftm.astrodynamic.utils* Package haben wir unsere eigene Exception implemementiert. Diese Exception dient dazu, inkompatible Einheiten in den *Scalar* Klassen abzufangen und zu handeln. Ebenfalls nutzen wir mit dem *SimulationRuntimeError* eine weitere eigene Exception, welche für Fehler während der Simulation genutzt wird.

3.11 Logging

Wir nutzen unseren eigenen Logger, welchen wir innerhalb der Klasse *Log* in *ch.hftm.astrodynamic.utils* implementiert haben. Diese Klasse konfiguriert den Logger basierend auf Konfigurationswerten aus der Klasse *ConfigRepository*. Um den Logger zu nutzen kann die Methode *build()* verwendet werden, welche den konfigurierten Logger des Typs *java.util.logging.Logger* zurückgibt.

Levels Mittels dem Konfigurationswert *LOG LEVEL* kann der Level des Log Outputs definiert werden. Die Standardkonfiguration für das Level ist auf *INFO* gesetzt. Mit diesem Level werden alle Meldungen mit höherer Priorität als Info ausgegeben.

File Mittels dem Konfigurationswert *LOG FILE* kann ein Log File definiert werden, in welchen die Logzeilen gespeichert werden. In der Standardkonfiguration wird das Logfile *astrodynamic.log* verwendet.

3.12 Pakete

Wir strukturieren unser Projekt innerhalb verschiedenen Java Paketen:

- controller; Enthält alle Controller für das Frontend
- gui; Enthält Komponenten, welche im Frontend verwendet werden
- model; Enthält Modelle für die Simulation
  - conditions; Enthält Bedingungen für Missionen
  - planetoids; Enthält vordefinerte Planeten für Missionen
  - spaceships; Enthält vordefinierte Weltraumfahrzeuge für Missionen
- physicts; Enthält Klassen für physikalische Objekte
- scalar; Enthält Kinder-Klassen für Scalare (Werte mit zugewiesenen Einheiten)
- utils; Enthält Hilfsklassen und Basisklassen
- ressources; Enthält Ressourcen (z.B Bilder), welche in unserem Projekt verwendet werden

Diese Pakete können einzeln oder ganz innerhalb des Projekts oder anderen Projekten importiert werden.

3.13 Singletons

Wir nutzen in unserem Projekt Singleton Klassen, um sicherzustellen, dass nur ein einzelnes statisches Element veränderbar ist.

Ein Beispiel einer solchen Klasse ist das *MissionRepository* aus dem Paket *ch.hftm.astrodynamic.utils*. Die integrierte *ObservabeList missions* muss für das gesamte Projekt während der Laufzeit einmalig sein. Damit wir keine Instanzen übergeben müssen, instanzieren wir diese Liste statisch.

Während die Liste selbst statisch bleibt, kann das MissionRepository jederzeit instanziert werden.

Literatur

qswitched. *Children of a Dead Earth Origin Stories*. https://childrenofadeadearth.wordpress.com/2016/05/06/origin-stories/. Accessed: 2023-02-22. 2016.

Abbildungsverzeichnis

Tabellenverzeichnis