

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

| | | |
|----------|--|-----------|
| 1 | Ziel und Zweck | 4 |
| 2 | Benutzeranleitung | 5 |
| 2.1 | Missionsliste | 5 |
| 2.1.1 | Grundlagen | 5 |
| 2.1.2 | Missionen nach Beschreibung suchen | 6 |
| 2.1.3 | Anlegen einer neuen Mission | 6 |
| 2.1.4 | Löschen einer Mission | 6 |
| 2.1.5 | Kopieren einer Mission | 7 |
| 2.1.6 | Editieren einer Mission | 7 |
| 2.1.7 | Simulieren einer Mission | 7 |
| 2.2 | Missions-Editor | 8 |
| 2.2.1 | Grundlagen | 9 |
| 2.2.2 | Hinzufügen einer Missionsbedingung | 9 |
| 2.2.3 | Missionsbedingungen | 10 |
| 2.2.4 | Löschen eines Planetoiden | 10 |
| 2.2.5 | Editieren eines Planetoiden | 10 |
| 2.2.6 | Hinzufügen eines Planetoiden | 10 |
| 2.2.7 | Hinzufügen eines Raumschiffs | 11 |
| 2.3 | Planetoid-Editor | 12 |
| 2.3.1 | Grundlagen | 12 |
| 2.3.2 | Editieren eines Vektors | 13 |
| 2.3.3 | Atmosphäreneinstellungen | 13 |
| 2.3.4 | Speichern des Planetoiden | 13 |
| 2.4 | Simulator | 15 |
| 3 | Technische Dokumentation | 16 |
| 3.1 | Applikationsdokumentation | 16 |
| 3.1.1 | Diagramme | 16 |
| 3.1.2 | Maven build | 16 |
| 3.2 | Themenumsetzung | 17 |
| 3.2.1 | Unit Tests | 17 |
| 3.2.2 | Enumeration | 17 |
| 3.3 | Vererbung | 18 |
| 3.4 | Casting | 18 |
| 3.5 | Interfaces | 18 |
| 3.6 | Abstrakte Klassen | 19 |
| 3.7 | Collection | 19 |
| 3.8 | Serialisierung | 19 |
| 3.9 | Speichern und Laden der Missionen | 20 |
| 3.10 | Exceptions | 20 |

| | |
|------------------------------|-----------|
| 3.11 Logging | 20 |
| 3.12 Pakete | 20 |
| 3.13 Singletons | 21 |
| Literatur | 22 |
| Abbildungsverzeichnis | 23 |
| Tabellenverzeichnis | 24 |

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.

¹qswitched. *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

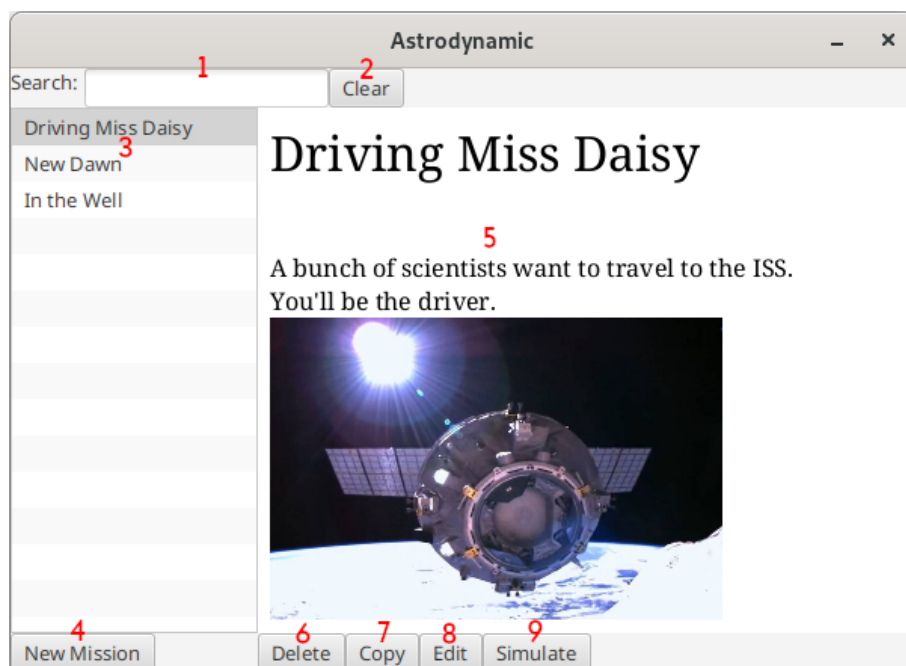


Abbildung 2.1: GUI Missionsliste mit Annotation

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.

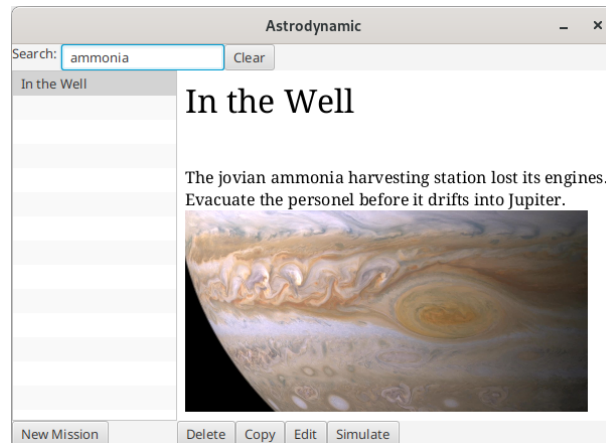


Abbildung 2.2: Missionsfilter bei Suche nach 'ammonia'

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.

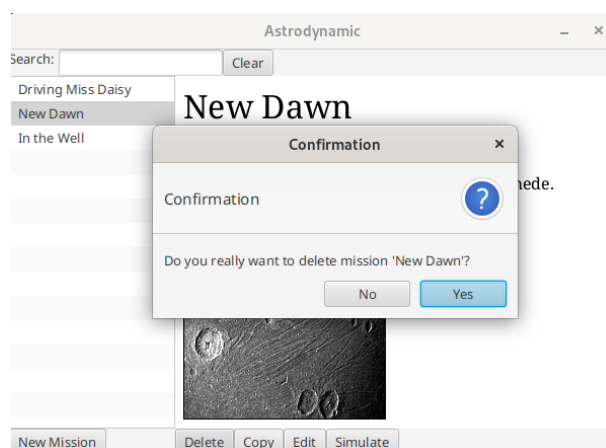


Abbildung 2.3: Sicherheitsabfrage bei Missionslöschung

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

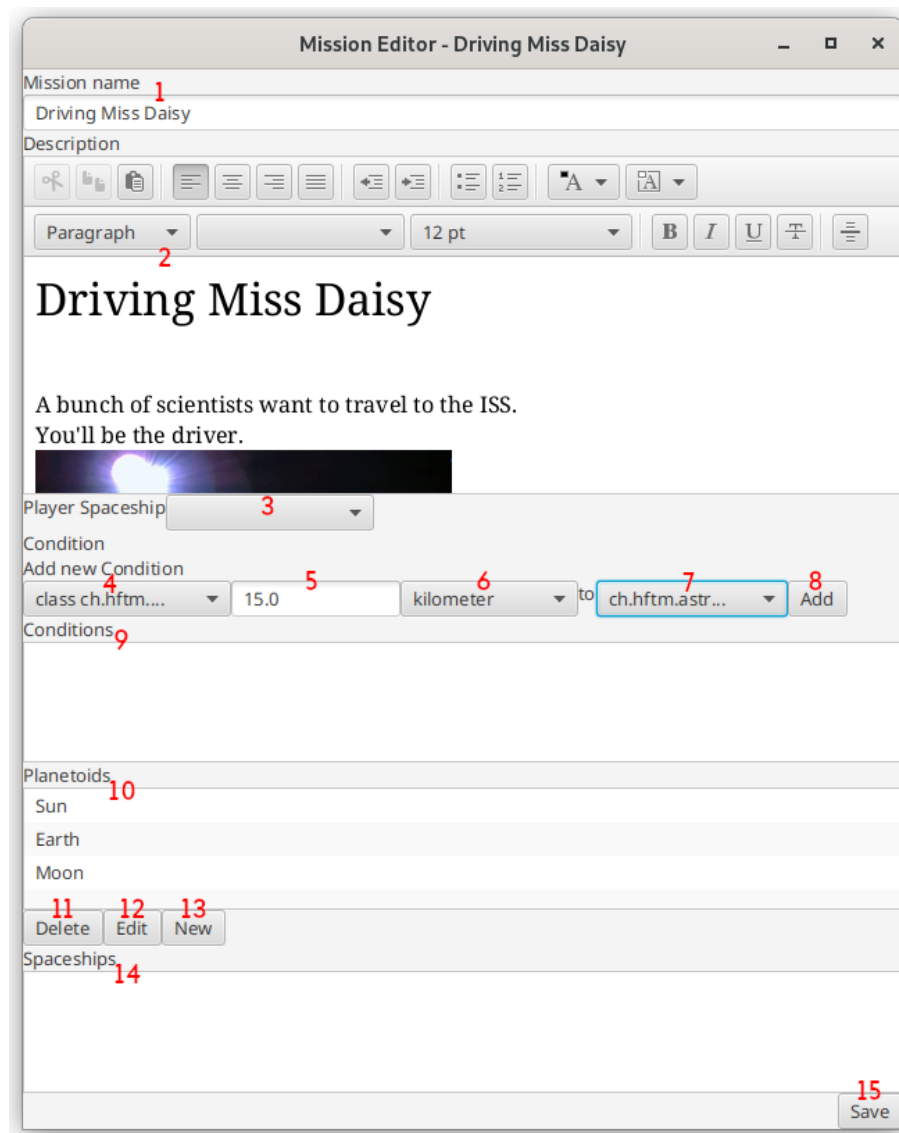


Abbildung 2.4: GUI Mission-Editor mit Annotation. Testmission 'Driving Miss Daisy' geöffnet.

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 Abbruchsbedingungen für die Simulation festgelegt und weitere dynamische Veränderungen an der Mission vorgenommen werden. Verwendete Planetoiden und Raumschiffe werden in den entsprechenden Listen aufgelistet.

2.2.2 Hinzufügen einer Missionsbedingung

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

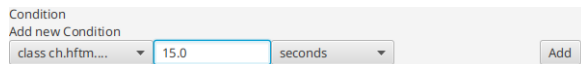


Abbildung 2.5: MaximumTime

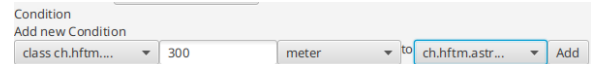


Abbildung 2.6: Approach

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.

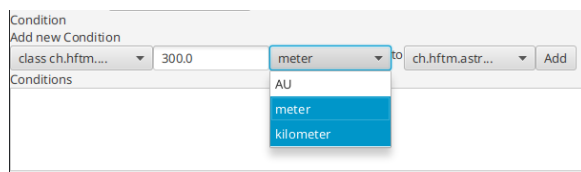


Abbildung 2.7: Masseinheit Meter

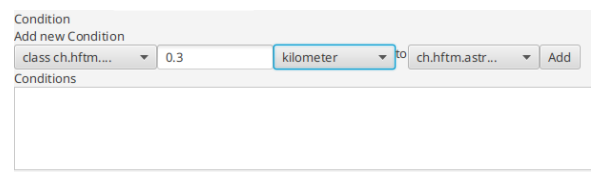


Abbildung 2.8: Masseinheit Kilometer

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

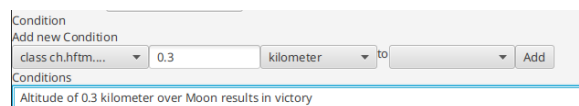


Abbildung 2.9: Neue Approach-Missionsbedingung hinzugefügt

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.

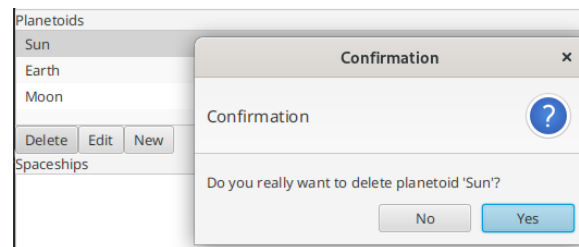


Abbildung 2.10: Sicherheitsabfrage bei Planetoidenlöschung

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-Editor mit dem ausgewählten Planetoiden. Für Details zum Planetoid-Editor konsultieren sie das [Kapitel Planetoid-Editor](#).

2.2.6 Hinzufügen eines Planetoiden

Klicken sie auf den New-Button unterhalb der Planetoiden-Liste. Es öffnet sich nun der Planetoid-Editor. Für Details zum Planetoid-Editor konsultieren sie das [Kapitel Planetoid-Editor](#).

2.2.7 Hinzufügen eines Raumschiffs

Zum hinzufügen eines Raumschiffs benutzen sie die Condition 'SetupHeavyLander'. Siehe [Abschnitt Hinzufügen einer Missionsbedingung](#).

2.3 Planetoid-Editor

The screenshot shows the 'Planetoid Editor - Earth' window. It contains several sections with input fields and dropdown menus. Red numbers 1 through 11 are placed next to specific fields to indicate their functions:

- Name:** 'Earth' (1)
- Description:** 'Our home' (2)
- Mass:** '5.9722E24' (3), unit 'kg' (4)
- Zero elevation height:** '6378100.0' (4), unit 'meter' (4)
- Position:** (5)
 - X: '1.496E13'
 - Y: '0.0'
 - Z: '0.0'
 - Magnitude: '1.496E13'
- Velocity:** (6), unit 'm/s'
 - X: '0.0'
 - Y: '29780.0'
 - Z: '0.0'
 - Magnitude: '29780.0'
- Atmosphere:**
 - Model: 'QUADRATIC_FALLOFF' (7)
 - Height ground to outer space: '600000.0' (8), unit 'meter' (4)
 - Oxygen factor: '0.21' (9)
- Buttons:** 'Cancel' (10) and 'Ok' (11)

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. Atmosphärenmodell
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.

| Position | | meter |
|-----------|------|-------------------|
| X | 10.0 | |
| Y | 10.0 | |
| Z | 0.0 | |
| Magnitude | | 14.14213562373095 |

Abbildung 2.12: Kalkulation der Länge Ausgangslage

| Position | | meter |
|-----------|-------|-------------------|
| X | 10.0 | |
| Y | 100.0 | |
| Z | 0.0 | |
| Magnitude | | 100.4987562112089 |

Abbildung 2.13: Kalkulation der Länge nach Anpassung Y-Wert

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

| Position | | meter |
|-----------|-------|------------------|
| X | 10.0 | |
| Y | 100.0 | |
| Z | 20.0 | |
| Magnitude | | 102.469507659596 |

Abbildung 2.14: Vektor-Skalierung der Dimensionswerte Ausgangslage

| Position | | meter |
|-----------|--------------------|-------|
| X | 0.0975900072948533 | |
| Y | 0.975900072948533 | |
| Z | 0.1951800145897066 | |
| Magnitude | | 1 |

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.

| Position | | meter |
|-----------|------|-------------------|
| X | 1200 | |
| Y | 1000 | |
| Z | 0 | |
| Magnitude | | 1562.049935181331 |

Abbildung 2.16: Vektor-Umwandlung der Größe Ausgangslage

| Position | | kilometer |
|-----------|-----|-------------------|
| X | 1.2 | |
| Y | 1.0 | |
| Z | 0.0 | |
| Magnitude | | 1562.049935181331 |

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

2.3.3 Atmosphäreneinstellungen

Feature eingeplant aber unimplementiert in aktueller Version

Das Atmosphärenmodell bestimmt die Berechnungsformel für die Atmosphärendichte. Die Atmosphärenhöhe bis Vakuum skaliert das Modell 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 Atmosphäre 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.

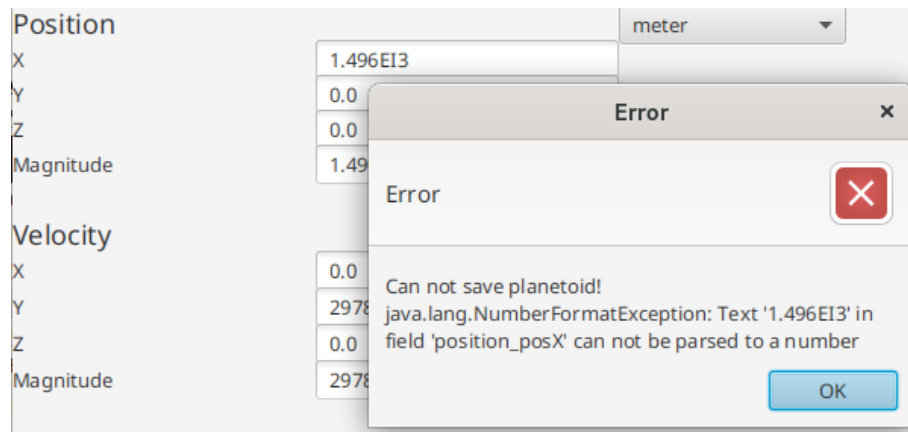


Abbildung 2.18: Planetoid-Editor Fehlermeldung: Buchstabe 'I' im Zahlenfeld

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

2.4 Simulator

Kapitel 3

Technische Dokumentation

3.1 Applikationsdokumentation

3.1.1 Diagramme

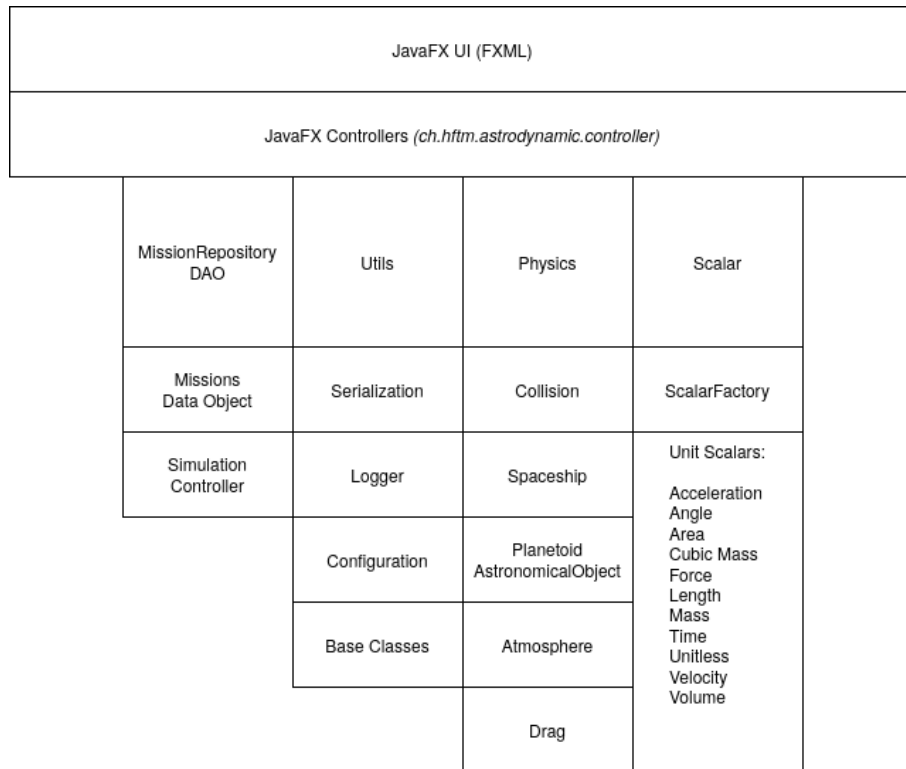


Abbildung 3.1: Übersicht Aufbau Astrodynamic

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 astrodynamics/build .
$ docker run --name astrodynamics -v ./:/astrodynamic astrodynamics/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 Funktionalität 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
UNTLESS
}

```

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) implementiert.

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 instanziiert (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 implementiert. 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 vordefinierte Planeten für Missionen
 - *spaceships*; Enthält vordefinierte Weltraumfahrzeuge für Missionen
- *physics*; Enthält Klassen für physikalische Objekte
- *scalar*; Enthält Kinder-Klassen für Skalare (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 *ObservableList 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 instanziiert 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

| | | |
|------|---|----|
| 2.1 | GUI Missionsliste mit Annotation | 5 |
| 2.2 | Missionsfilter bei Suche nach 'ammonia' | 6 |
| 2.3 | Sicherheitsabfrage bei Missionslöschung | 6 |
| 2.4 | GUI Mission-Editor mit Annotation. Testmission 'Driving Miss Daisy' geöffnet. | 8 |
| 2.5 | MaximumTime | 9 |
| 2.6 | Approach | 9 |
| 2.7 | Masseinheit Meter | 9 |
| 2.8 | Masseinheit Kilometer | 9 |
| 2.9 | Neue Approach-Missionsbedingung hinzugefügt | 9 |
| 2.10 | Sicherheitsabfrage bei Planetoidenlöschung | 10 |
| 2.11 | GUI Planetoid-Editor mit Annotation. Planetoid Erde geöffnet. | 12 |
| 2.12 | Kalkulation der Länge Ausgangslage | 13 |
| 2.13 | Kalkulation der Länge nach Anpassung Y-Wert | 13 |
| 2.14 | Vektor-Skalierung der Dimensionswerte Ausgangslage | 13 |
| 2.15 | Vektor-Skalierung der Dimensionswerte nach Anpassung der Länge | 13 |
| 2.16 | Vektor-Umwandlung der Grösse Ausgangslage | 13 |
| 2.17 | Vektor-Umwandlung der Grösse nach anpassen der Grösse | 13 |
| 2.18 | Planetoid-Editor Fehlermeldungsbeispiel | 14 |
| 3.1 | Übersicht Aufbau Astrodynamics | 16 |

Tabellenverzeichnis

2.1 Verfügbare Missionsbedingungen 10