SOLARSYSTEM

SEW-TGM

Thomas Stedronsky, Simon Wortha
tstedronsky@student.tgm.ac.at, swortha@student.tgm.ac.at

Inhalt

1. Aufgabenstellung 2
2. Projektbeschreibung 2
2.1. Anforderungen 2
2.2. Team 2
3. Evaluierung 3
3.1. Dokumentation 3
3.2. Installation 3
3.3. Community 3
3.4. Prototyp 3
3.5. Conclusio 3
4. GUI-Skizzen und Bedienkonzept 4
5. Technische Dokumentation 5
5.1. UML 5
5.2. Pattern 5
5.3. Licht 6
5.4. Texturen und Models 6
5.5. Geschwindigkeit 6
6. Bedienungsanleitung 7
6.1. Tastensteuerung: 7
6.2. Maussteuerung 8
6.2.1. Kameramode-XY 8
6.2.2. Kameramode-Ultra 9
7. Probleme 10
8. Quellen 10
9. Abbildungsverzeichnis 10

1. Aufgabenstellung

In einem Team (2) sind folgende Anforderungen zu erfüllen.

- Ein zentraler Stern
- Zumindest 2 Planeten, die sich um die eigene Achse und in elliptischen Bahnen um den Zentralstern drehen
- Ein Planet hat zumindest einen Mond, der sich zusätzlich um seinen Planeten bewegt
- Kreativität ist gefragt: Weitere Planeten, Asteroiden, Galaxien,...
- Zumindest ein Planet wird mit einer Textur belegt (Erde, Mars,... sind im Netz verfügbar)

Events:

- Mittels Maus kann die Kameraposition angepasst werden: Zumindest eine Überkopf-Sicht und parallel der Planentenbahnen
- Da es sich um eine Animation handelt, kann diese auch gestoppt werden. Mittels Tasten kann die Geschwindigkeit gedrosselt und beschleunigt werden.
- Mittels Mausklick kann eine Punktlichtquelle und die Textierung ein- und ausgeschaltet werden.

2. Projektbeschreibung

Es soll ein Solarsystem mit einer Sonne und mindestens 2 Planeten implementiert werden. Außerdem soll mindestens ein Planet einen Mond haben. Diese Planeten und Monde bewegen sich in realistischer Umlaufbahnen im Solarsystem.

2.1. Anforderungen

Es muss eine IDE zur Implementierung von Python vorhanden sein, in unserem Fall verwenden wir PyCharm 4.5. Wir verwenden das Framework Panda3D, dies muss korrekt installiert sein. Außerdem ist für Panda3D Python 2.7 Voraussetzung (in der Panda Installation enthalten).

2.2. Team

Unser Team besteht aus 2 Mitgliedern.

- Thomas Stedronsky
- Simon Wortha

3. Evaluierung

- PyGame
- PyGlet
- Panda3D

Es wurden alle Frameworks auf Dokumentation, Community, Prototypen und Installation getestet.

3.1. Dokumentation

PyGlet: 1
PyGame: 0
Panda3D: 1

Gute Dokumentation bei Panda3D und PyGlet

3.2. Installation

Die Installation war bei PyGlet sehr einfach und war ohne große Probleme zum Installieren. PyGame konnte auf einer Linux-Distribution nicht installiert werden.

Panda3D war sehr leicht zu installieren. Bei der Installation wurden außerdem reichlich Examples beigefügt um Panda gut zu verstehen.

PyGlet: 1
PyGame: 0
Panda3D: 2

3.3. Community

Die Community ist bei allen ca. gleich, ähnliche Stackoverflow Fragen, …

PyGlet: 2
PyGame: 1
Panda3D: 3

3.4. Prototyp

Der Prototyp wurde nur bei PyGlet zum Laufen gebracht, dieser Prototyp war sehr hilfreich für das Projekt. Bei PyGame konnte kein vollständiges Example zum Laufen gebracht. Bei Panda3D wurden diverse Examples zum Laufen gebracht, durch diese ist Panda leicht verständlich.

PyGlet: 3
PyGame: 1
Panda3D: 4

3.5. Conclusio

PyGlet: 3
PyGame: 1
Panda3D: 4

Durch das Ergebnis haben wir uns für **Panda3D** entschieden.

4. GUI-Skizzen und Bedienkonzept

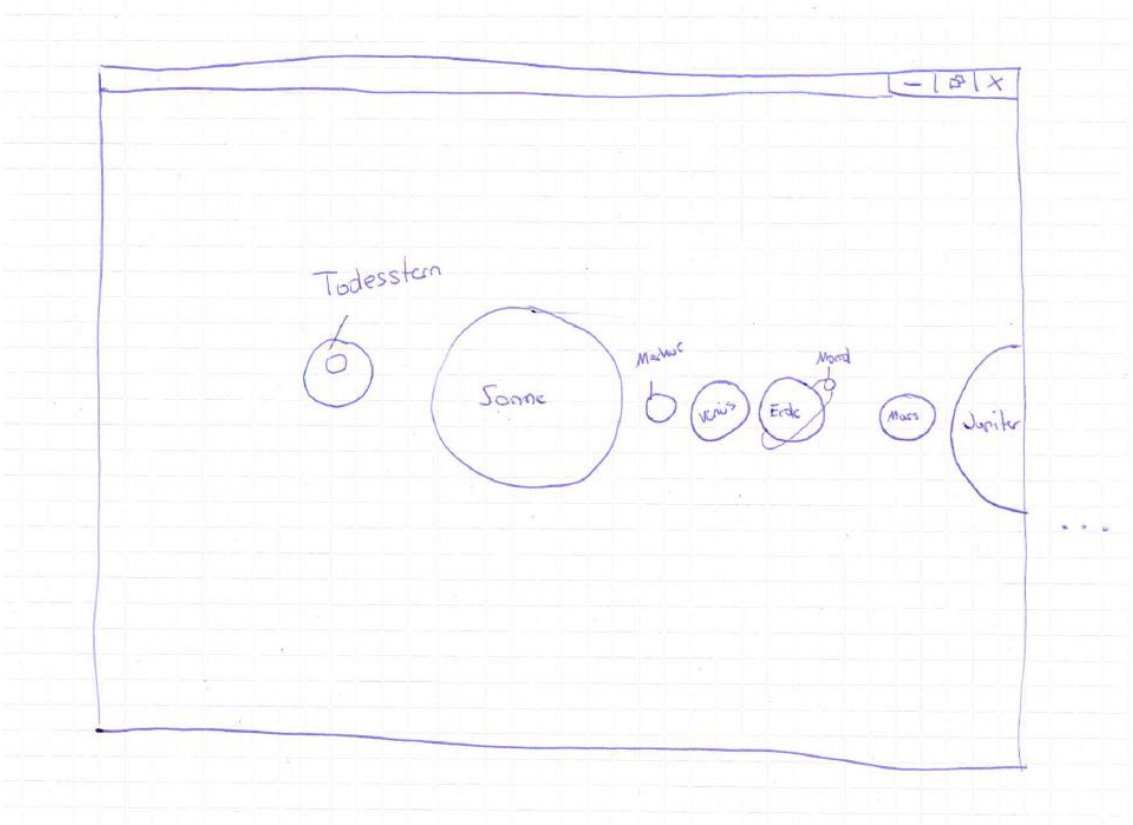

Abbildung 1 skizze

Grundsätzlich unterscheiden wir 2 Events Mausevents und Tastenevents.

Die Idee war das man mit der Tastensteuerung, Pausen einlegen kann, die Geschwindigkeit ändern kann, die Kamera Position (Kamera Mode ändern) kann, …

Für die genaue Tastaturbelegung siehe *7. Bedienungsanleitung*.

Die Maus wird nur verwendet, um sich im Solarsystem zu bewegen.

Die Maussteuerung ist bei Panda3D automatisch enthalten und kann beliebig im Code aktiviert oder deaktiviert werden.

```
ShowBase.useDrive()
```

Bzw.

```
ShowBase.oobe()
```

Mit diesem Befehl aktiviert man die Kamera. Es gibt diverse Modes, mit unterschiedlichen Funktionen.

Bei uns ist außerdem eine Top View vorhanden, wo man das Solarsystem von „oben“ betrachten kann.

5. Technische Dokumentation

5.1. UML

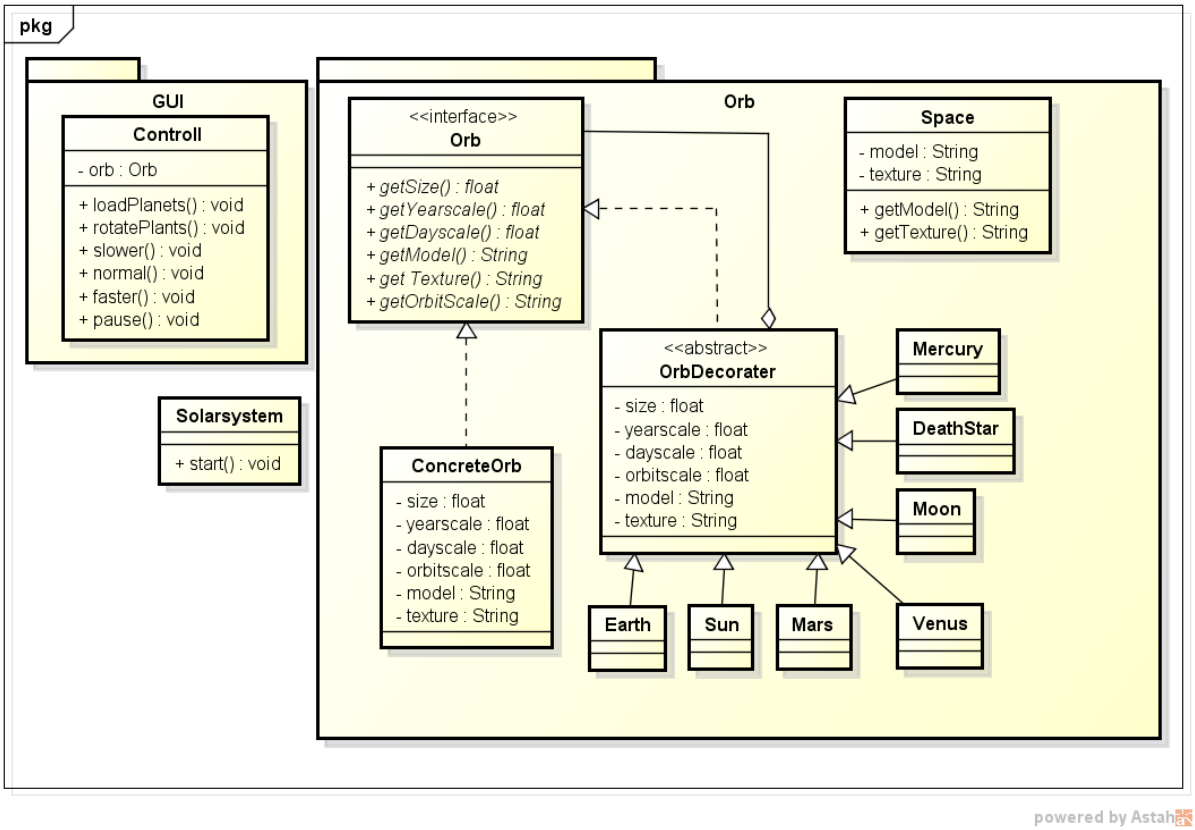

Abbildung 2 uml

5.2. Pattern

Wie im UML schon zu erkennen ist, haben wir das Decorator Pattern angewandt. Die Idee dahinter ist, dass wir einmal einen Himmelskörper (Orb) erzeugen und diesen immer als Parameter bei den jeweiligen Planeten übergeben, damit dieses Himmelskörper Objekt nur noch dekoriert werden muss.

```
# konkreter Himmelskoerper:
self.co = ConcreteOrb()
# dekorierte Himmelskoerper:
self.s = Sun(self.co)
self.merc = Mercury(self.co)
```

Zum Beispiel hat das Attribut `size` in der Klasse `ConcreteOrb` den Wert 0.6, welcher auch der `size` der Erde entspricht. In den konkreten Planetenklassen wird nun `get_size()` aufgerufen und der erhaltene Wert wird noch mit einem andren Faktor multipliziert. So berechnen wir die Größen der einzelnen Planeten immer in Relation zu der Größe der Erde. (Bei den anderen Attributen funktioniert es mit dem gleichen Prinzip).

```
def get_size(self):
    return self.concrete_orb.get_size() * 1.8
```

Weiters hatten wir Idee für die Monde ein Factory Pattern anzuwenden. Doch da der Fixpunkt der Umlaufbahn sehr stark mit dem Framework verbunden ist findet dies in der Klasse `Controller` statt und der Mond wir als ganz normaler Himmelskörper behandelt.

Da wir am Anfang nur wenige Himmelskörper hatten wurden diese alle einzeln Implementiert und nicht in einer Collection zusammengefasst, was im Nachhinein eine bessere Entscheidung gewesen wäre, da so im File Controller eine hohe Inflexibilität herrscht. Weiters wäre es von Vorteil gewesen einige der GUI Funktionen andere Klassen auszulagern um für bessere Lesbarkeit zu sorgen. Es wurde überlegt ob dies noch getan werden soll, doch da unser Programm so funktioniert wie wir es uns vorstellen, kamen wir auf den Entschluss es nicht zu tun. (Never touch a running system)

5.3. Licht

Es wurde eine Punktlichtquelle implementiert, die dafür sorgt, dass die zur Sonne abgewandte Seite der Planeten schwarz ist, da dort kein Licht hin scheint. Diese Punktlichtquelle wurde folgender Maßen implementiert:

```
plight = PointLight('plight')
plight.setColor(VBase4(1, 1, 1, 1))
self.plnp = render.attachNewNode(plight)
self.plnp.setPos(0, 0, 0)
render.setLight(self.plnp)
```

Da wir aber nicht wollten, dass die Rückseite komplett schwarz ist, haben wir uns dafür entschieden auch ein „Ambient Light“ hinzuzufügen. Dieses sorgt dafür das die Rückseiten nur dunkler sind und nicht ganz schwarz. Dies wurde wie folgt implementiert:

```
alight = AmbientLight('alight')
alight.setColor(VBase4(0.2, 0.2, 0.2, 1))
self.alnp = render.attachNewNode(alight)
render.setLight(self.alnp)
```

Da aber auch die Sonne einen Schatten von der Punktlichtquelle bekommt, muss für sie ein eigenes „Ambient Light“ erstellt werden.

5.4. Texturen und Models

Die Texturen und Modelle werden wie folgt geladen:

```
# Erstellung der Erde
self.earth = loader.loadModel(self.e.get_model())
self.earth_tex = loader.loadTexture(self.e.get_texture())
self.earth.setTexture(self.earth_tex, 1)
self.earth.reparentTo(self.orbit_root_earth)
self.earth.setScale(self.e.get_size())
self.earth.setPos(self.e.get_orbitscale(), 0, 0)
```

5.5. Geschwindigkeit

Es ist möglich die Geschwindigkeit der Himmelskörper zu erhöhen oder zu verringern. Wenn die Geschwindigkeit zu weit verringert wird bewegen sich die Himmelskörper rückwärts. Dies ist ganz einfach möglich:

```
self.orbit_period_earth.setPlayRate(self.orbit_period_earth.getPlayRate()-
1)
```

Das Problem hier ist nur das wenn der Wert auf 0 fällt wird die `PlayRate` zurückgesetzt. Also ist noch eine IF-Anweisung notwendig um die 0 Stelle zu umgehen.

6. Bedienungsanleitung

Startansicht, Planeten werden von oben angezeigt.

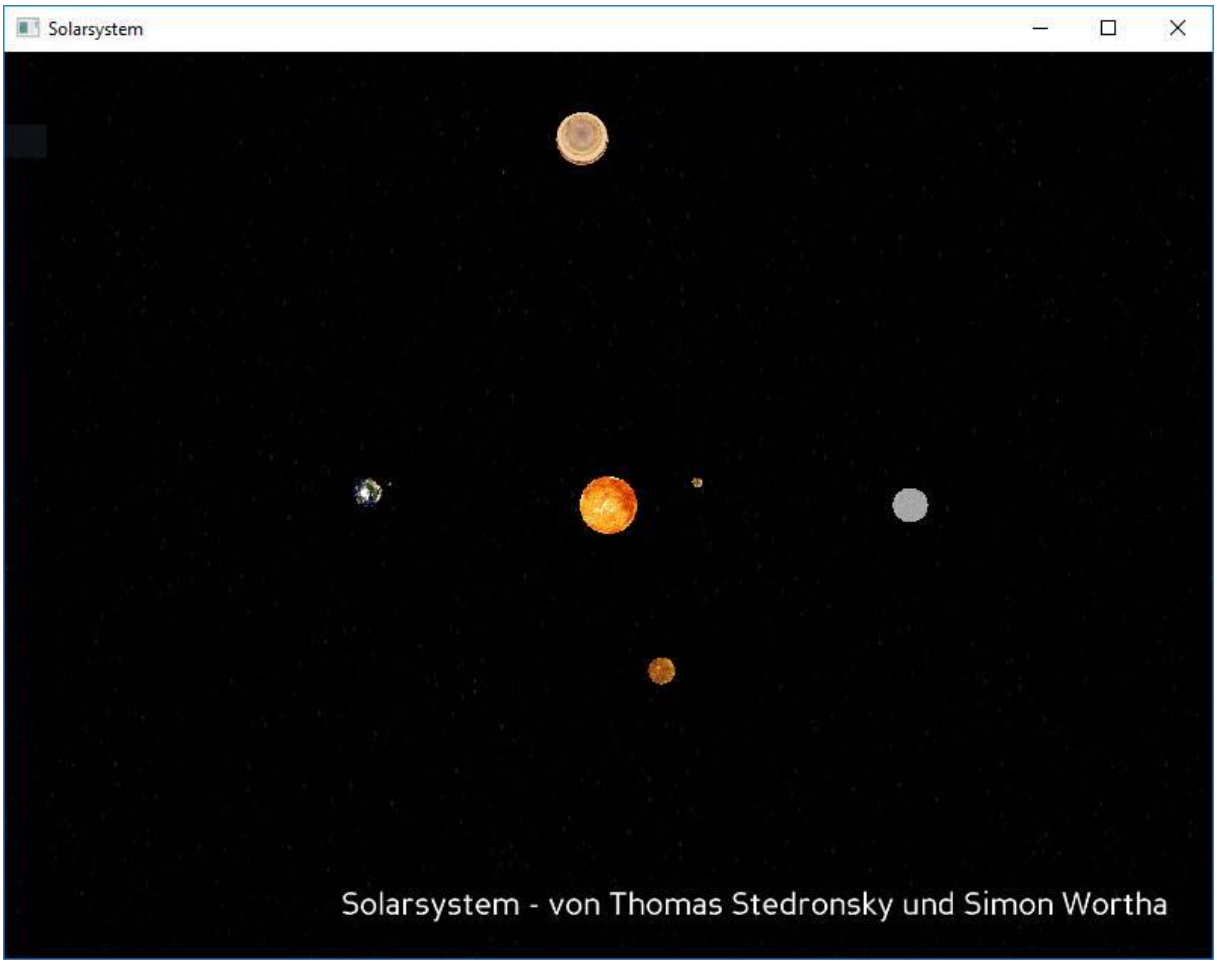

Abbildung 3 top view

6.1. Tastensteuerung:

1 … Planeten werden verlangsamt (Auch in die andere Richtung möglich)

2 … Planeten bewegen sich schneller

T … Texturen können an und ausgeschalten werden (Das Licht wird dabei ebenfalls ausgeschaltet)

O … Wechsel auf die Top View Ansicht

I … Wechsel auf den Kameramode XY

U … Wechsel auf den Kameramode Ultra

P … Solarsystem wird angehalten bzw. gestartet

Außerdem kann mit „H“ die Hilfe in der Laufzeit an und ausgeschalten werden.

Beispiel zum Hinzufügen einer Buttonfunktion:

```
base.accept("p", self.handlePause)
```

man kann mit `accept` einer Taste eine bestimmte Funktion geben. Sollte wie in dem Fall p gedrückt werden, dann wird die Methode `handlePause` ausgeführt.

6.2. Maussteuerung

6.2.1. Kameramode-XY

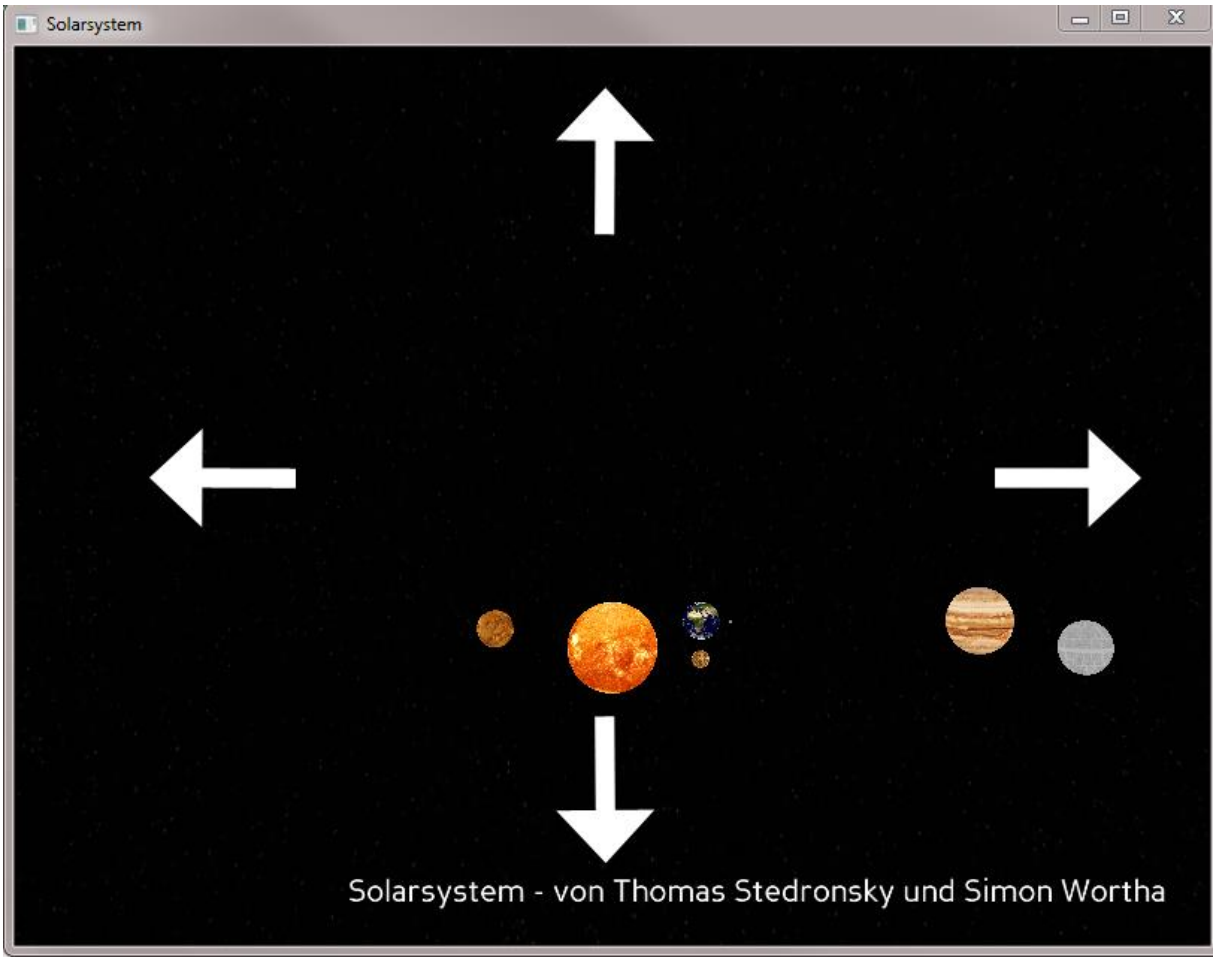

Abbildung 4 drive view

Mit dieser Kameraoption kann mit einen Kick nach oben vorwärts gefahren werden bzw. mit einen Klick nach unten rückwärts. Bei Klick in den Seitlichen Bereich kann entweder nach links oder rechts geschwenkt werden. Mit dieser Kameraoption kann man sich nur in X und in Y Richtung bewegen.

Durch diese Kameraoption kann man sich im Solarsystem bewegen und kann die 3D Effekte gut sehen.

Man hat mit diesem Kameramode einen 360° Blickwinkel, somit kann man jeden Himmelskörper von beliebiger Seite auf der X bzw. Y Achse betrachten.

```
base.enableMouse()
base.useDrive()
#setzt Kamera auf gewuenschte Ausgangsposition
base.drive.node().setPos(0, -40, 0)
```

Mit diesen Codezeilen konnte möglichst einfach der Kameramodus verändert werden.

6.2.2. Kameramode-Ultra

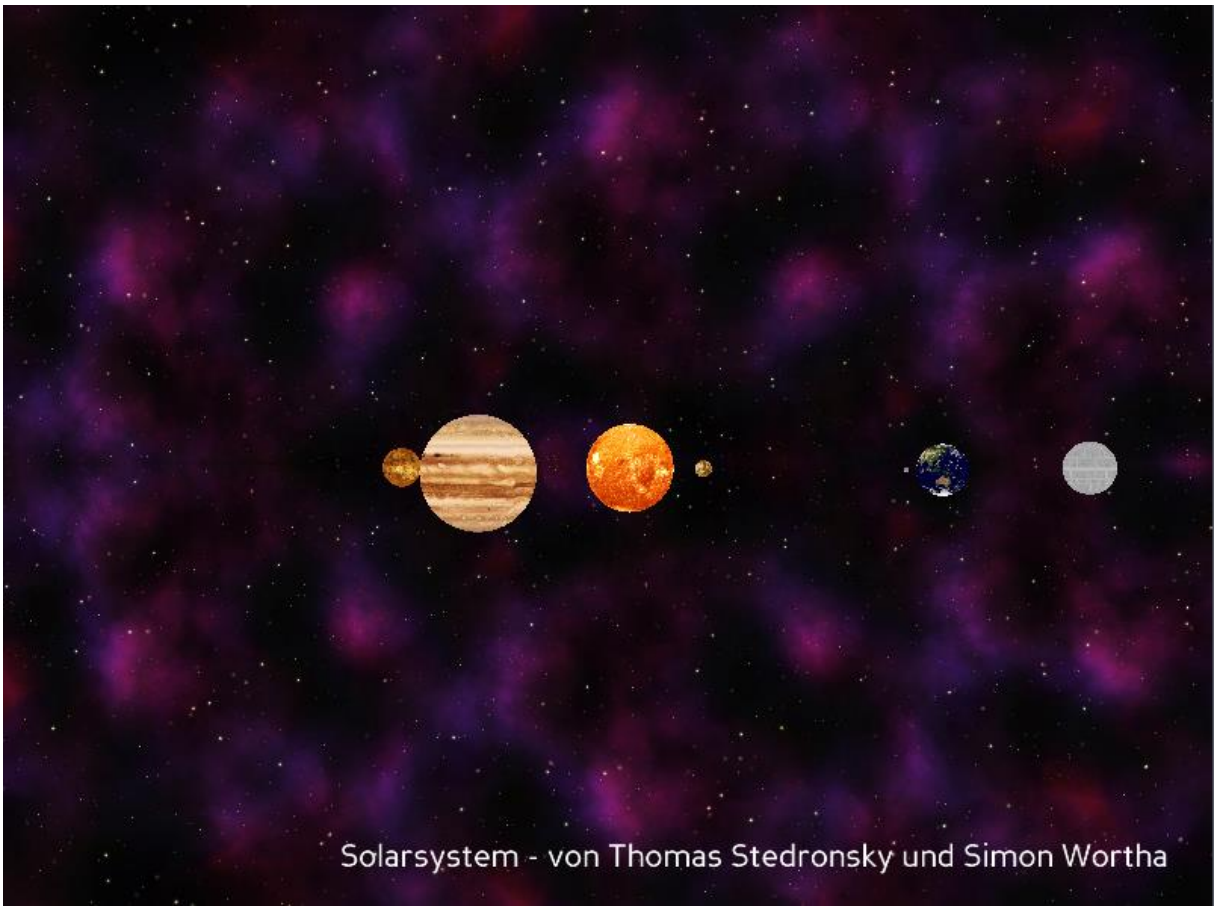

Abbildung 5 camera ultra

Bei diesem Mode kann wie folgt navigiert werden.

Bei Klick auf die linke Maustaste und Bewegung der Maus lässt es sich im Universum nach oben, unten, links und rechts herumfahren, aber dabei die Distanz zu den Planeten nicht zu verändern.

Bei Klick auf die rechte Maustaste kann man die Distanz mit einer Bewegung der Maus verändern.

Um den Blickwinkel zu verändern ist ein Kick auf das Scrollrad erforderlich. Damit kann das Solarsystem von unten betrachtet werden.

Beim Ultra-Modus ging dies so.

```
base.enableMouse()
base.useTrackball()
#setzt Kamera auf gewuenschte Ausgangsposition
base.trackball.node().setPos(0, 40, 0)
```

7. Probleme

Anfangs wurden viele Frameworks ausprobiert, PyGlet bzw. PyGame haben anfangs leicht ausgesehen, allerdings hat sich dann rausgestellt, dass diese nicht optimal waren. Darum haben wir uns wie *in 3.Evaluierung für* Panda3D beschrieben.

Es gab außerdem Probleme, dass ein falscher Interpreter verwendet wurde. Es wurde der Standard Interpreter von PyCharm verwendet(Python 3.5 Interpreter). Allerdings braucht Panda3D die Python Version 2.7, dieser Interpreter wird mit der Installation mitgegeben. Im Installationsordern befindet sich ein Ordner *python*, wo dieser Interpreter enthalten ist. Nach diesen Schritt kann Panda verwendet werden.

Bei der Implementierung kam es zu Problemen bei den Texturen, diese Texturen konnten nicht geladen werden. Dieser Fehler wurde behoben in dem man die Ordnerstruktur angepasst hat. Bei uns sind alle Texturen im Ordern *models* enthalten. Diese Order wird geladen und somit sind alle Texturen verfügbar.

Außerdem gab es das Problem, dass der Ordner *orb* nach einen Pull-Vorgang plötzlich groß geschrieben wurde, daraus folgte, dass das Programm nicht mehr gestartet werden konnte, also am besten alle Ordner klein schreiben.

Es gab außerdem Probleme bei der Positionierung der Kamera unsere Kamera hatte immer einen zu hohen Startpunkt, dies haben wir wie folgt behoben. Für die jeweiligen Kamera Modes musste die Position gesetzt werden z.B. `base.trackball.node().setPos(0, 40, 0).`

8. Quellen

[1] PythonGameLibraries - Python Wiki, https://wiki.python.org/moin/PythonGameLibraries, zuletzt besucht am 30.11.2015

[2] panda3D, https://www.panda3d.org/, zuletzt besucht am 30.11.2015

[3] panda3D, https://www.panda3d.org/manual/index.php/Main_Page, zuletzt besucht am 30.11.2015

[4] PyGame, http://pygame.org/hifi.html, zuletzt besucht am 30.11.2015

[5] PyGlet, https://bitbucket.org/pyglet/pyglet/wiki/Home, zuletzt besucht am 29.11.2015

9. Abbildungsverzeichnis

Abbildung 1 skizze 4
Abbildung 2 uml 5
Abbildung 3 top view 7
Abbildung 4 drive view 8
Abbildung 5 camera ultra 9