8.12.2015

Solarsystem

SEW – Python3D-Anwendung

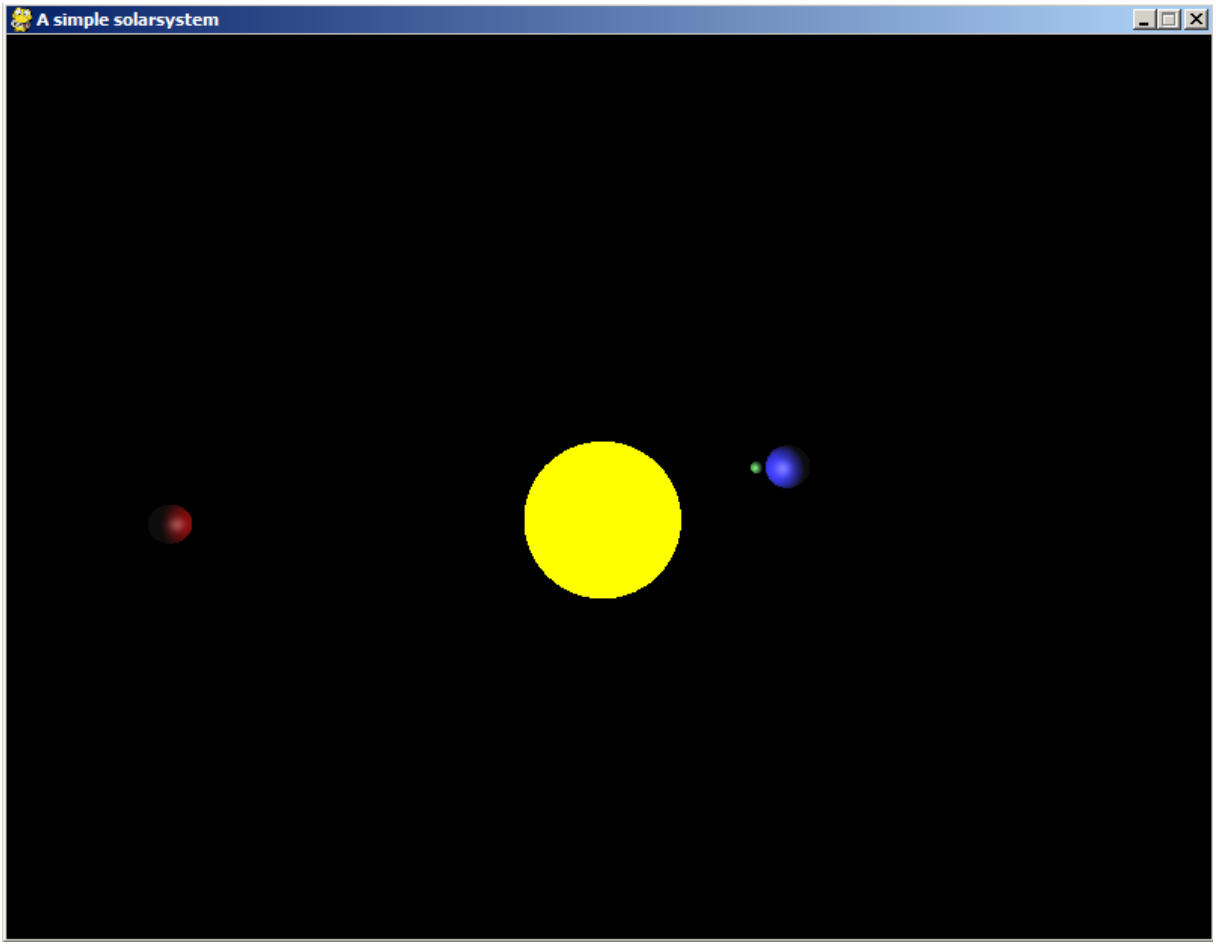
Stefan Erceg & Martin Kritzl
5BHITT

Inhalt

1 Angabe

Wir wollen unser Wissen aus SEW nutzen, um eine kreative Applikation zu erstellen. Die Aufgabenstellung:

Erstelle eine einfache Animation unseres Sonnensystems!



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.
- Schatten: Auch Monde und Planeten werfen Schatten.

Wählt ein geeignetes 3D-Framework für Python (Liste unter https://wiki.python.org/moin/PythonGameLibraries) und implementiert die Applikation unter Verwendung dieses Frameworks.

Abgabe: Die Aufgabe wird uns die nächsten Wochen begleiten und ist wie ein (kleines) Softwareprojekt zu realisieren, weshalb auch eine entsprechende Projektdokumentation notwendig ist. Folgende Inhalte sind in jedem Fall verpflichtend:

- Projektbeschreibung (Anforderungen, Teammitglieder, Rollen, Tools, ...)
- GUI-Skizzen und Bedienkonzept (Schnittstellenentwürfe, Tastaturbelegung, Maussteuerung, ...)
- Evaluierung der Frameworks (zumindest 2) inkl. Beispielcode und Ergebnis (begründete Entscheidung)
- Technische Dokumentation: Architektur der entwickelten Software (Klassen, Design Patterns)
- Achtung: Bitte überlegt euch eine saubere Architektur!
- Den gesamten Source Code in 1 Klasse zu packen ist nicht ausreichend!
- Kurze Bedienungsanleitung
- Sauberes Dokument (Titelblatt, Kopf- und Fußzeile, ...)

Hinweise zu OpenGL und glut:

- Ein Objekt kann einfach mittels glutSolidSphere() erstellt werden.
- Die Planten werden mittels Modelkommandos bewegt: glRotate(), glTranslate()
- Die Kameraposition wird mittels gluLookAt() gesetzt
- Bedenken Sie bei der Perspektive, dass entfernte Objekte kleiner - nahe entsprechende größer darzustellen sind.
  Wichtig ist dabei auch eine möglichst glaubhafte Darstellung. gluPerspective(), glFrustum()
- Für das Einbetten einer Textur kann die Library Pillow verwendet werden! Die Community unterstützt Sie bei der Verwendung.

Viel Spaß und viel Erfolg!

2 Requirementanalyse

2.1 funktionale Anforderungen

Arbeitspaket	Person	Schätzung (min)	Tatsächlich (min)	Erledigt
Himmelskörper zur GUI hinzufügen	**Erceg & Kritzl**	**60**	**80**	✓
Himmelskörper löschen	**Kritzl**	**20**	**10**	✓
Textierung ein-/ausschalten	**Kritzl**	**25**	**15**	✓
Kameraposition anpassen	**Erceg & Kritzl**	**50**	**65**	✓
Geschwindigkeit drosseln	**Kritzl**	**40**	**30**	✓
Geschwindigkeit beschleunigen	**Kritzl**	**20**	**15**	✓
Punktlichtquelle ein-/ausschalten	**Erceg**	**30**	**40**	✓
Schatten zu Himmelskörpern hinzufügen	**Erceg**	**40**	**15**	✓

2.2 nicht funktionale Anforderungen

Arbeitspaket	Person	Schätzung (min)	Tatsächlich (min)	Erledigt
3 verschiedene GUI-Tools in Python evaluieren	**Erceg & Kritzl**	**180**	**165**	✓
Grafische Übersicht erstellen	**Erceg**	**15**	**20**	✓
UML-Diagramm erzeugen	**Erceg & Kritzl**	**60**	**75**	✓
Struktur des UML-Diagramms beschreiben	**Erceg**	**30**	**40**	✓

3 Evaluierung der grafischen Tools in Python

3.1 Requirements

3.1.1 OpenGL

Installieren von OpenGL mit folgendem Befehl:

```
pip install PyOpenGL PyOpenGL_accelerate
```

3.2 Pygame

3.2.1 Herunterladen & Installieren

Heruntergeladen wird eine wheel-Datei von dieser Seite:
http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame
Dabei wird unter Windows die Datei `pygame-1.9.2a0-cp34-none-win32.whl` ausgewählt.

Zum Installieren der Wheel-Datei ist die Installation von wheel notwendig:

```
pip install wheel
```

Die eigentliche Installation von Pygame geschieht durch:

```
pip install <wheel-Datei>
```

3.2.2 Example starten

Dazu wurde das Planetensystem unseres lieben Herr Professor Rafeiner (ein sehr feiner Kerl) von Github heruntergeladen: https://github.com/peterfuchs1/Py01/tree/master/solar

Obwohl alle Requirements vorhanden waren, tritt bei einer Ausführung des Programms ein interner Fehler auf, der sich auf eine Versionsinkompatibilität zurückführen lässt. Da dieses Problem trotz Hilfe von Herrn Professor Dolezal nicht in angemessener Zeit behoben werden konnte.

3.2.3 Fazit

Da wir nicht einmal ein Example zum Laufen gebracht haben, war dieses Tool für unseren Zweck gestorben. Dass dies trotzdem nicht ganz unmöglich ist, haben Projekte der letzten 5ten Klasse gezeigt. Trotzdem war uns die Zeit zu schade, um die Probleme zu lösen.

3.3 Pyglet

3.3.1 Herunterladen & Installieren

Die Installation von Pyglet ist mit einem Command alleine, die wohl einfachste Installation im Vergleich zu den beiden anderen:

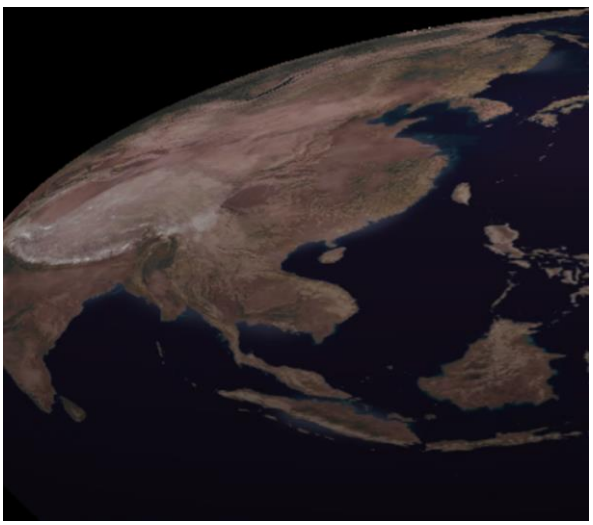
```
pip install pyglet
```

3.3.2 Example starten

Es wurde ein bestehendes Example von Github heruntergeladen:
https://github.com/greenmoss/PyWavefront/tree/master/example

Dieses Example konnte ohne Probleme gestartet werden. Dazu wurde es in IntelliJ geöffnet und die Datei `pyglet_demo2.py` ausgeführt. Zu sehen ist die Erde die sich um sich selbst dreht. Eine Kamera oder andere Planeten sind nicht implementiert.

Mit normaler Fenstergröße



Mit veränderter Fenstergröße

Wie zu sehen ist, reagiert das Tool nicht automatisch auf Veränderung der Fenstergröße und müsste dies deswegen manuell vornehmen.

3.3.3 Code

Der Code des Examples ist sehr schwer zu lesen und zu verstehen, welches aber nicht nur aufgrund der Codequalität zu Stande kommt, sondern auch deswegen, weil sehr viele Konfigurationsparameter notwendig sind, um alleine dieses einfache Programm zu schreiben.
Genauso waren in anderen Examples plötzlich ganz andere Herangehensweisen vorhanden, was das allgemeine Verständnis noch weiter verschlechtert hat

3.3.4 Dokumentation

Die Dokumentation der einzelnen Befehle ist sehr gut beschrieben, es fehlen jedoch Beispiele die ein größeres Ausmaß vorweisen, um einen groben Überblick zu schaffen.

3.3.5 Fazit

Das Beispielprogramm hat zwar gut funktioniert, jedoch war schon für so ein kleines Beispiel sehr viel Code notwendig um dieses zu erstellen. Die Community hinter diesem Tool lässt ebenso zu wünschen übrig.

3.4 Panda3d

3.4.1 Herunterladen & Installieren

Heruntergeladen wird unter Windows eine .exe-Datei, die ein internes Python 2.7 beinhaltet, welches alle notwendigen Libraries mit sich bringt. Die Installation des Python-Interpreters ist jedoch optional.

https://www.panda3d.org/download/panda3d-1.8.1/Panda3D-1.8.1.exe

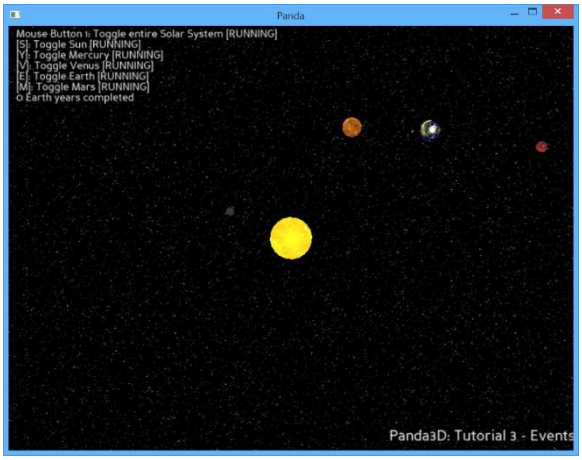
Wird die der mitgelieferte Python-Interpreter nicht verwendet, so muss auf jeden Fall eine Version 2.x verwendet werden und die nötigen Libraries nachinstalliert werden.

Wie haben uns für die Verwendung des mitgelieferten Interpreters entschieden.

3.4.2 Example starten

In dem Installations-Ordner von Panda3D sind 24 gut dokumentierte, sehr hilfreiche, teils umfangreiche Beispielprogramme vorhanden, die den Einstieg sehr leicht gestalten. Diese Programme können über einen lokalen Link sofort ausprobiert werden und die Funktionalität gezeigt werden. Bis auf ein Beispielprogramm konnten alle einwandfrei gestartet werden.

Darunter war ebenfalls ein Beispiel für ein Solar-System vorhanden, welches den Einstieg noch leichter gemacht hat.



3.4.3 Code

Der Code dieses Tools ist sehr leicht zu lesen und zu schreiben, da sehr viele Schritte bereits vom Tool selber durchgeführt werden. Die Methoden sind sprechend und können meist sehr intuitiv verwendet werden. Durch die vorhandenen Beispielprogramme kann sehr viel Funktionalität verstanden werden.

3.4.4 Dokumentation

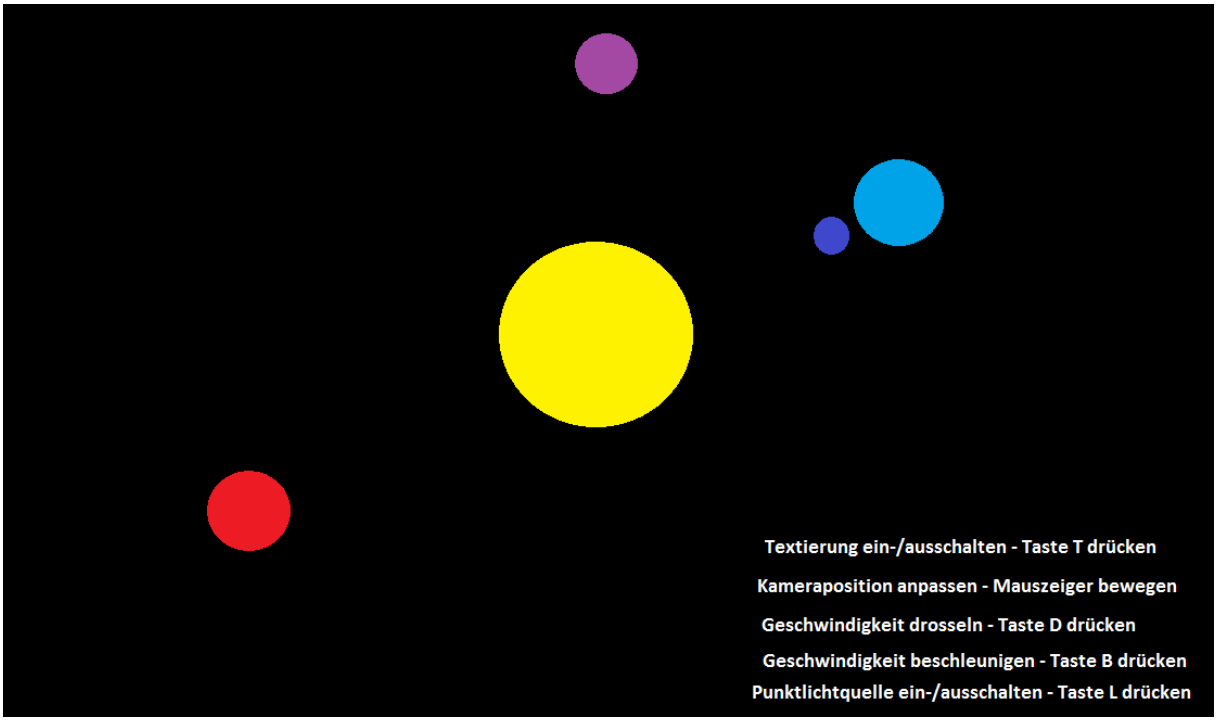
Die Dokumentation ähnelt der von Java sehr stark und kann deswegen auch gut gelesen werden. Die Ausführlichkeit ist zwar nicht immer ganz zufriedenstellend, wird aber durch die gute Community wieder ausgeglichen.

3.4.5 Fazit

Die schon integrierten Beispielprogramme machen den Einstieg sehr einfach. Der Code ist aufgrund der sehr sprechenden Funktionsnamen und des Aufbaus sehr leicht zu lesen und zu verstehen. Viele Fragen lassen sich durch die Community durch schon vorhandene Forumseinträge beantworten.
Damit ist dieses Tool perfekt für unsere Aufgabe geschaffen.

4 Designüberlegung

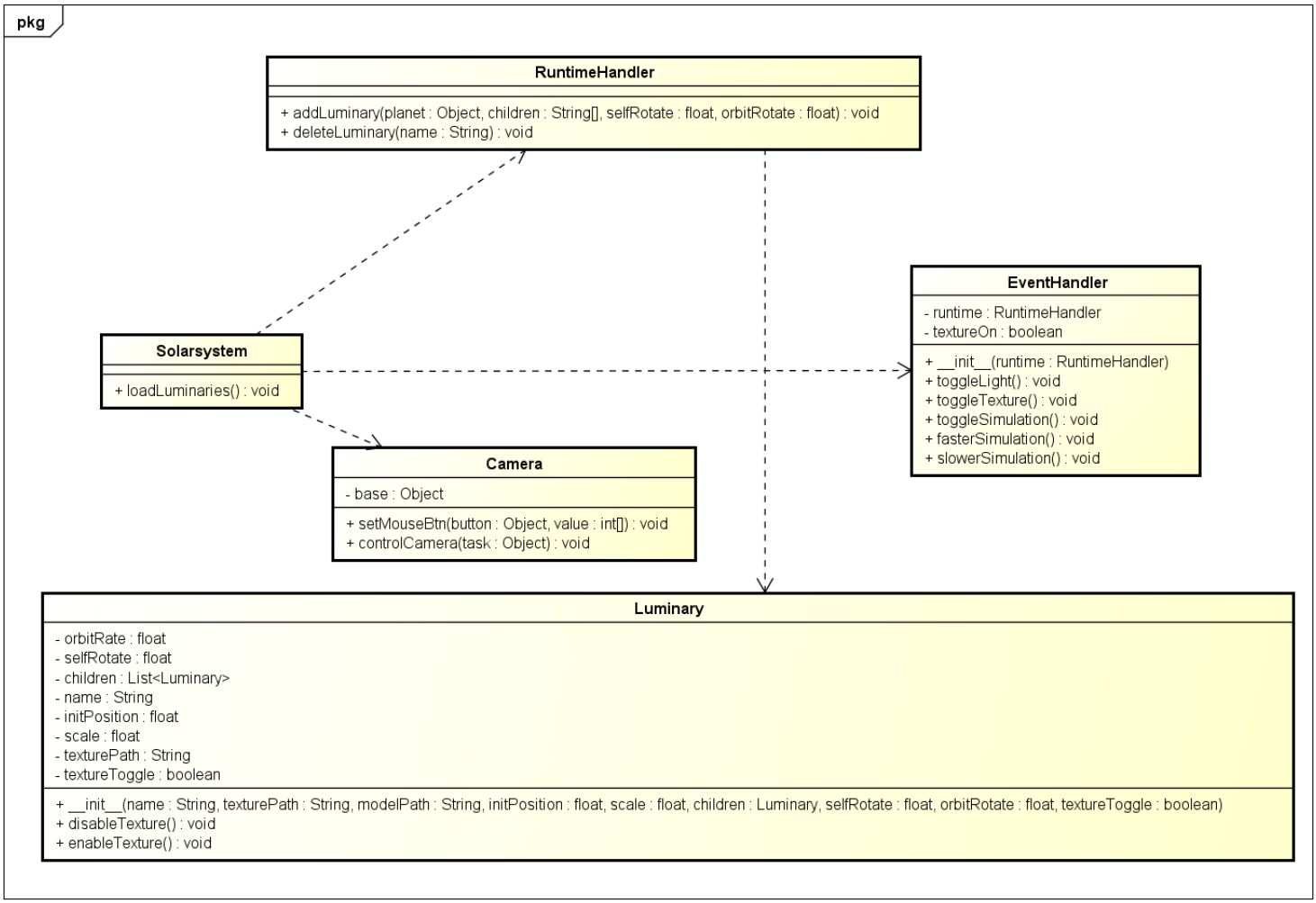
4.1 grafische Übersicht



Wie an der grafischen Übersicht zu erkennen ist, haben wir uns überlegt, in der Mitte die Sonne abzubilden. Um die Sonne herum rotieren die Planeten (vorgenommen haben wir uns 3 zu implementieren) und zur Erde ebenfalls der Mond. Im unteren rechten Abschnitt der grafischen Oberfläche soll eine Legende angezeigt werden, um auf Events nach bestimmten Tastendrücken bzw. nach Bewegen des Mauszeigers hinzuweisen.

4.2 Abbildung des UML-Diagramms

Das UML-Diagramm wurde mit dem Programm „Astah“ erstellt.



4.3 Überlegungen zur Struktur

Wir haben uns überlegt, unser Programm in 5 Klassen aufzuteilen:

4.3.1 Klasse „Luminary“

Diese Klasse nimmt im Konstruktor Parameter zum Laden der jeweiligen Texturen und Körper zu den Himmelskörpern (engl.: „Luminary“) entgegen. Ebenfalls wird hier deren Startposition und Größe gesetzt. Zum Ein- bzw. Ausblenden der Texturen existieren ebenfalls Methoden. Zurückgegeben wird zum Schluss ein Himmelskörper.

4.3.2 Klasse „RuntimeHandler“

Der „RuntimeHandler“ existiert, um einen bestimmten Himmelskörper zum Gesamtsystem bzw. zur grafischen Oberfläche hinzuzufügen. Als Parameter können ebenfalls Kinder definiert werden, wie z.B., dass der Mond ein Kind von der Erde ist. Die jeweilige Selbstrotation bzw. die Rotation um den Mittelpunkt, nämlich der Sonne, werden ebenfalls angegeben. Eine Methode zum Entfernen eines bestimmten Himmelskörpers (als Parameter wird dessen Name angegeben) steht ebenfalls zur Verfügung.

4.3.3 Klasse „Camera“

Die Klasse „Camera“ ist für das Verwalten der aktuellen Kameraposition zuständig. Diese Klasse wird dann vom „EventHandler“ aufgerufen, welcher im nächsten Unterkapitel beschrieben wird.

4.3.4 Klasse „EventHandler“

Wie der Name der Klasse bereits sagt, werden hier bestimmte Events (Tastendrücke und/oder Mausklicks) entgegengenommen. Nachdem eine bestimmte Taste gedrückt bzw. die Maus bewegt wurde, werden entsprechende Methoden aufgerufen (z.B.: beim Drücken der Taste „T“ wird die Textierung ein- bzw. ausgeschaltet).

4.3.5 Klasse „Solarsystem“

Diese Klasse stellt die Main-Klasse des Programms dar. Hier werden die jeweiligen Himmelskörper initialisiert.

5 Eingebaute Patterns

In unserem Programm haben wir 2 Patterns, nämlich das Decorator und Composite Pattern, umgesetzt.

5.1 Umsetzung des Decorator Patterns

In der „Luminary“-Klasse besitzen wir als Parameter beim Konstruktor unter anderem das Attribut „children“.

In dem main-File „SolarSystem“ werden dann in der Methode „loadLuminaries“ neue Himmelskörper initialisiert und dort kann man dann im Parameter angeben, ob der jeweilige Himmelskörper auch „Kinder“ besitzt.

Die Erde besitzt beispielsweise den Mond als Kind und dies haben wir folgendermaßen umgesetzt:

```
earth = Luminary("earth", "models/earth_1k_tex.jpg",
"models/planet_sphere", self.orbitscale, self.sizescale, [moon],
self.dayscale, self.yearscale, True)
```

Man könnte hier auch ohne jegliche Probleme mehrere Kinder angeben. Die Sonne besitzt bei uns beispielsweise 7 Kinder:

```
sun = Luminary("sun", "models/sun_1k_tex.jpg", "models/planet_sphere", 0, 3
* self.sizescale, [mercury, venus, mars, earth, gas, ice, brown], 20, None,
True)
```

5.2 Umsetzung des Composite Patterns

Bei dieser Art von Pattern setzt man die „hat ein“-Beziehung statt einer „ist ein“-Beziehung um. Das bedeutet, dass Klassen nicht von anderen Klassen erben, sondern ein Objekt von der anderen Klasse besitzen.

Wir haben dies in unserem Programm öfters umgesetzt. Beispielsweise wird im Konstruktor der „EventHandler“-Klasse unter anderem ein „RuntimeHandler“- und „Camera“-Objekt übergeben, welches initialisiert wird:

```
def __init__(self, runtime, camera, middle):
    self.runtime = runtime
    self.camera = camera
```

Somit initialisiert man in der jeweiligen Klasse lediglich Objekte von anderen Klassen und kann die Methoden von den anderen Klassen ohne der Umsetzung einer Vererbung aufrufen.

6 Arbeitsdurchführung

6.1 Luminary

Im Grunde ist ein Luminary nichts weiter als eine Ansammlung von Informationen die diesen beschreiben. Als einzige Methoden gibt es das ein- und ausschalten der Texturen. Durch das Attribut children können dem Luminary andere seiner Art untergeordnet werden.

```
moon = Luminary("moon", "models/moon_1k_tex.jpg", "models/planet_sphere",
0.1 * self.orbitscale, 0.1 * self.sizescale, None, .0749 * self.yearscale,
.0749 * self.yearscale, True)

earth = Luminary("earth", "models/earth_1k_tex.jpg",
"models/planet sphere", self.orbitscale, self.sizescale, [moon],
self.dayscale, self.yearscale, True)
```

6.2 RuntimeHandler

Die Hauptfunktion dieser Klasse ist die Verwaltung der Luminaries. Darunter ist die wohl wichtigste Methode das Hinzufügen:

```
def addLuminary(self, render, planet):
    self.luminaryList[luminary.name] = planet
    if luminary.name not in self.rootList:
        self.rootList[planet.name] = render.attachNewNode(planet.name)

    if (luminary.children):
        for child in luminary.children:
            self.rootList[child.name] =
               (self.rootList[planet.name].attachNewNode(child.name))
            self.rootList[child.name].setPos(luminary.initPosition, 0, 0)
            self. addLuminary (render,child)
…
```

Durch diese Methode ist genauso für die Beziehung der jeweiligen Luminaries gesorgt. Dies wird in Bedingung `if (luminary.children)` durchgeführt.

Im Weiteren ist die Rotation der Luminaries eine benötigte Funktion:

```
def rotateLuminaries(self):
    for orbit in self.orbitList:
        self.orbitList[orbit].loop()
…
```

6.3 Solarsystem

Hier werden die Events entgegen genommen und an die Kamera oder an den EventHandler weitergegeben:

```
self.accept("r", self.eventHandler.restartSimulation)
self.accept("w", self.camera.setMouseBtn, [0, 1])
```

Genauso wird die Legende erstellt, die durch eine eigene Funktion erstellt wird:

```
def genLabelText(self, text, i):
    return OnscreenText(text=text, pos=(-1.3, .95 - .05 * i), fg=(1, 1, 1,
        1), align=TextNode.ALeft, scale=.05, mayChange=1)


self.escEventText = self.genLabelText(
    "ESC: Quit program", 0)

self.spaceEventText = self.genLabelText(
    "Space: Toggle entire Solar System", 1)
```

Ebenfalls werden hier alle notwendigen Instanzen der anderen Klassen gehalten:

```
self.runtime = RuntimeHandler()
self.eventHandler = EventHandler(self.runtime)
self.camera = Camera(render, 40)
```

6.4 EventHandler

Im Grunde stehen hier Methoden zur Verfügung, die ein Event weiter bearbeitet. So wie es zum Beispiel bei der Umschaltung der Texturen der Fall:

```
def toggleTexture(self):
    planets = self.runtime.getAllLuminaries()
    if self.textureOn == True:
        for luminary in luminaries:
            if luminaries[luminary].textureToggle==True:
…
```

6.5 Camera

Die Kamera ist für die dreidimensionalen Bewegungen im Raum verantwortlich. In dieser wird auf Eventeingabe mit geeigneter Richtungsänderung reagiert. Wie es zum Beispiel bei den Veränderungen der x-Richtung der Fall ist:

```
if self.mousebtn[2]:
    base.camera.setX(base.camera, -elapsed*30)
if self.mousebtn[3]:
    base.camera.setX(base.camera, elapsed*30)
```

7 Bedienungsanleitung

Bevor man unser Programm nutzen kann, muss das jeweilige System, an dem das Programm läuft, die Panda3D Runtime besitzen. Diese kann auf folgender Seite heruntergeladen werden: https://www.panda3d.org/download.php?runtime

Um das Programm dann zu starten, muss der Befehl

```
python SolarSystem.py
```

im src-Ordner ausgeführt werden.

Die Steuerung des Programms, welche nicht Case-Sensitive ist, erfolgt folgendermaßen:

Taste	Aktion
ESC	Programm beenden
Space	Solarsystem stoppen bzw. weiter ausführen
T	Textur ein-/ausschalten
L	Punktlichtquelle ein-/ausschalten
+	Simulation verschnellern
-	Simulation verlangsamen
B	Vogelperspektive aktivieren
W bzw. Pfeiltaste oben	Hineinzoomen
S bzw. Pfeiltaste unten	Herauszoomen
A bzw. Pfeiltaste links	nach links bewegen
D bzw. Pfeiltaste rechts	nach rechts bewegen
U	nach oben bewegen
J	nach unten bewegen

8 Lessons learned

- Kenntnis über verschiedene Python Graphic-Tools gewonnen
- Vertiefung in Panda3D durchgeführt
- 3-dimensionale Darstellungen in Python erlernt