

SolarSystem

Systemtechnik Labor

5BHIT 2015/16, Gruppe A

Hagen Aad Fock & Stefan Polydor

Version 0.1

Betreuer: Begonnen am 3.11.2015

Note: Beendet am 30.11.2015

Inhaltsverzeichnis

[Einführung 3](#_Toc437449844)

[Ziele 3](#_Toc437449845)

[Voraussetzungen 3](#_Toc437449846)

[Aufgabenstellung 4](#_Toc437449847)

[Projektbeschreibung 6](#_Toc437449848)

[Teammitglieder/Rollen 6](#_Toc437449849)

[Tools 6](#_Toc437449850)

[GUI-Skizzen und Bedienkonzept 7](#_Toc437449851)

[Bedienkonzept 7](#_Toc437449852)

[Auswahl der Technologie 8](#_Toc437449853)

[Kurze Bedienungsanleitung 8](#_Toc437449854)

[Technische Dokumentation: Architektur der entwickelten Software (Klassen, Design Patterns) 8](#_Toc437449855)

[CelestialBody 8](#_Toc437449856)

[CameraHandler 8](#_Toc437449857)

[ActionHandler 8](#_Toc437449858)

[Universe 8](#_Toc437449859)

[SpecialClass 8](#_Toc437449860)

[Main 8](#_Toc437449861)

[Ergebnisse 9](#_Toc437449862)

[Ein zentraler Stern 9](#_Toc437449863)

[Ein Planet hat zumindest einen Mond, der sich zusätzlich um seinen Planeten bewegt 9](#_Toc437449864)

[Zumindest 2 Planeten, die sich um die eigene Achse und in elliptischen Bahnen um den Zentralstern drehen 10](#_Toc437449865)

[Zumindest ein Planet wird mit einer Textur belegt (Erde, Mars,... sind im Netz verfügbar) 11](#_Toc437449866)

[Kreativität ist gefragt: Weitere Planeten, Asteroiden, Galaxien,... 11](#_Toc437449867)

[Mittels Maus kann die Kameraposition angepasst werden: Zumindest eine Überkopf-Sicht und parallel der Planentenbahnen 12](#_Toc437449868)

[Quellen 15](#_Toc437449869)

# Einführung

Diese Aufgabe ist dazu da, um die Fertigkeiten mit Python und Frameworks von Python zu verbessern.

## Ziele

* Funktionierendes Solarsystem
* Besseres Verständnis im Umgang mit Python

## Voraussetzungen

* Gewisses Grundverständnis von Python

## Aufgabenstellung

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

Events:

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

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

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!

# Projektbeschreibung

## Teammitglieder/Rollen

|  |  |
| --- | --- |
| **Mitgliedsname** | **Rolle** |
| Hagen Aad Fock | Developer |
| Stefan Polydor | Developer |

## Tools

|  |  |
| --- | --- |
| **Tool** | **Toolname** |
| IDE | PyCharm - v. 5 |
| Framework | Panda3D - v. 1.8.1 & 1.9 |
| Versionierungstool | Github |
| Programmiersprache | Python – 2.7 |

## GUI-Skizzen und Bedienkonzept

Abbildung 1 Balsamiq

## Bedienkonzept

* Rechte Maustaste 🡺 Rein und raus zoomen
* Linke Maustaste 🡺 sich in die gewünschte Richtung verschieben
* Maus-Rad 🡺 Sich auf dem Punkt wo man sich befindet umschauen
* Mittels Leertaste Solarsystem anhalten und an starten

## Auswahl der Technologie

Wir hatten uns drei Technologien angeschaut:

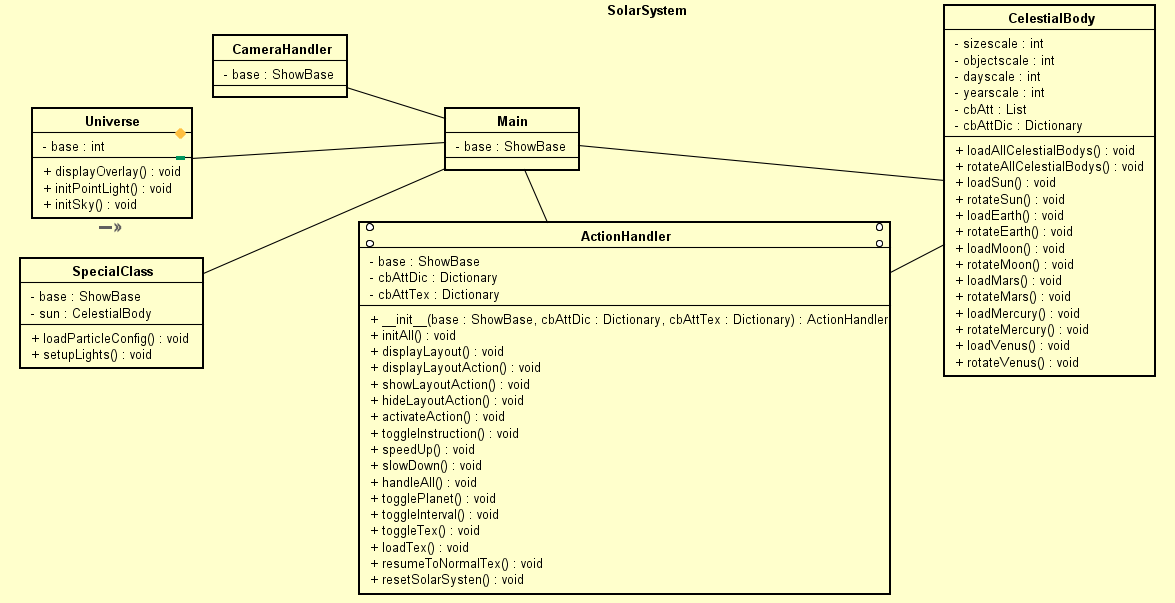
* Pyglet [1]
* Panda3D [2]
* Pygame [3]

Wir haben uns nach ausprobieren aller Frameworks für Panda3D entschieden. Panda 3D hat 24 Samples. Darunter auch ein Sample über ein Solarsystem mit Texturen für die Planeten. Innerhalb der 24 Samples findet man alle Funktionen und Implementierung die man benötigt um die Aufgabenstellung umzusetzen.

## Kurze Bedienungsanleitung

* Rechte Maustaste 🡺 Rein und raus zoomen.
* Linke Maustaste 🡺 sich in die gewünschte Richtung verschieben.
* Maus-Rad 🡺 Sich auf dem Punkt wo man sich befindet umschauen.
* Leertaste 🡺 Solarsystem anhalten und an starten.
* I - Taste 🡺 Bedienungsanleitung in der Anwendung anzeigen und verstecken.
* + und – Tasten 🡺 Regulierung der Geschwindigkeit
* T – Taste 🡺 Texturen der Planeten ein und ausschalten
* R – Taste 🡺 Resetet das ganze Solarsystem
* X, Y, C, B, V 🡺 Easteregg - Funktionen

## Technische Dokumentation: Architektur der entwickelten Software (Klassen, Design Patterns)



### CelestialBody

Diese Klasse ist für all die Himmelskörper zuständig.

### CameraHandler

Diese Klasse ist für die Kamerabewegungen und den Anfangspunkt zuständig.

### ActionHandler

Diese Klasse ist für alle Eingaben des Users zuständig.

### Universe

Diese Klasse ist für den Hintergrund und die Lichquellen zuständig.

### SpecialClass

Diese Klasse ist für die Sonneneruptionen zuständig. Damit sich unser Projekt von den anderen unterscheidet.

### Main

Damit alles aufgerufen wird.

### Pattern

Wir benutzen in unserem Projekt explizit 2 Pattern. Das Factory Pattern und das Dekorateur Pattern. Das Factory Pattern wird bei der Initialisierung der einzelnen Himmelskörper benutzt. Das Dekorateur Pattern wird dann bei der Gestaltung der Himmelskörper angewandt. Nämlich, wie groß soll das Ding sein, wie schnell soll es sich drehen, welche Textur soll es haben, usw.

# Ergebnisse

## Ein zentraler Stern

Als zentralen Stern haben wir klarerweise die Sonne gewählt. Wir realisieren unsere Lösung für „Ein zentraler Stern“, indem wir render zum Mittelpunkt unseres Systems machen und alle Planeten an den Mittelpunkt anbringen. Man kann das wie eine CD sehen. Wenn der Mittelpunkt sich dreht, dreht sich alles mit.

Beispiel: Sonne

*# Hier wird die Form fuer die Sonne geladen  
# In diesem Fall ist eine planet\_sphere*self.sun = loader.loadModel(**"../../models/planet\_sphere"**)  
*# Hier wird die Sonne ins Zentrum des SolarSystems platziert*self.sun.reparentTo(render)  
*# Hier wird der Sonne die gelbe Sonnen Textur geladen*self.sun\_tex = loader.loadTexture(**"../../models/sun\_1k\_tex.jpg"**)  
*# Hier wird die Textur gesetzt*self.sun.setTexture(self.sun\_tex, 1)  
*# Hier wird die Groesse des Himmelskoerper gesetzt*self.sun.setScale(2 \* self.sizescale)

Beispiel: Erde

*#Hier wird die Erde an die Sonne/render (den Mittelpunkt) angehaengt*self.orbit\_root\_earth = render.attachNewNode(**'orbit\_root\_earth'**)

## Ein Planet hat zumindest einen Mond, der sich zusätzlich um seinen Planeten bewegt

Bei dieser Aufgabe haben wir anstatt der Sonne die Erde zu dem Mittelpunkt für den Mond festgelegt.

*#Hier wird die Erde an die Sonne/render (den Mittelpunkt) angehaengt*self.orbit\_root\_earth = render.attachNewNode(**'orbit\_root\_earth'**)

*# Hier wird der Mond an die Erde gehaengt*self.orbit\_root\_moon = (  
 self.orbit\_root\_earth.attachNewNode(**'orbit\_root\_moon'**))

## Zumindest 2 Planeten, die sich um die eigene Achse und in elliptischen Bahnen um den Zentralstern drehen

Siehe Codeausschnitt

**def** loadEarth(self):  
 *#Hier wird die Erde an die Sonne/render (den Mittelpunkt) angehaengt* self.orbit\_root\_earth = render.attachNewNode(**'orbit\_root\_earth'**)  
 *# Load earth* self.earth = loader.loadModel(**"../../models/planet\_sphere"**)  
 self.earth\_tex = loader.loadTexture(**"../../models/earth\_1k\_tex.jpg"**)  
 self.earth.setTexture(self.earth\_tex, 1)  
 self.earth.reparentTo(self.orbit\_root\_earth)  
 self.earth.setScale(self.sizescale)  
 self.earth.setPos(self.orbitscale, 0, 0)  
*# end loadEarth***def** rotateEarth(self):  
 *# earth* self.orbit\_period\_earth = self.orbit\_root\_earth.hprInterval(  
 self.yearscale, (360, 0, 0))  
 self.day\_period\_earth = self.earth.hprInterval(  
 self.dayscale, (360, 0, 0))

**def** loadMars(self):  
 self.orbit\_root\_mars = render.attachNewNode(**"orbit\_root\_mars"**)  
  
 *# Load Mars* self.mars = loader.loadModel(**"../../models/planet\_sphere"**)  
 self.mars\_tex = loader.loadTexture(**"../../models/mars\_1k\_tex.jpg"**)  
 self.mars.setTexture(self.mars\_tex, 1)  
 self.mars.reparentTo(self.orbit\_root\_mars)  
 self.mars.setPos(1.52 \* self.orbitscale, 0, 0)  
 self.mars.setScale(0.515 \* self.sizescale)  
  
**def** rotateMars(self):  
 self.orbit\_period\_mars = self.orbit\_root\_mars.hprInterval(  
 (1.881 \* self.yearscale), (360, 0, 0))  
 self.day\_period\_mars = self.mars.hprInterval(  
 (1.03 \* self.dayscale), (360, 0, 0))

## Zumindest ein Planet wird mit einer Textur belegt (Erde, Mars,... sind im Netz verfügbar)

Siehe Codeausschnitt

**def** loadMars(self):  
 self.orbit\_root\_mars = render.attachNewNode(**"orbit\_root\_mars"**)  
  
 *# Load Mars* self.mars = loader.loadModel(**"../../models/planet\_sphere"**)  
 self.mars\_tex = loader.loadTexture(**"../../models/mars\_1k\_tex.jpg"**)  
 self.mars.setTexture(self.mars\_tex, 1)  
 self.mars.reparentTo(self.orbit\_root\_mars)  
 self.mars.setPos(1.52 \* self.orbitscale, 0, 0)  
 self.mars.setScale(0.515 \* self.sizescale)

## Kreativität ist gefragt: Weitere Planeten, Asteroiden, Galaxien,...

Wir habe uns überlegt was könnte unser Sonnensystem einzigartig machen? Eine Sonne mit Sonneneruptionen und ein Solarsystem mit außergewöhnlichen Texturen.  
Die Sonneneruption verwirklichen wir mithilfe von particels. Man definiert wie sich solche Partikel verhalten soll, innerhalb einer .ptf Datei. Beispiel für eine .ptf Datei folgt.

# Factory parameters  
p0.factory.setLifespanBase(0.5000)  
p0.factory.setLifespanSpread(0.0000)  
p0.factory.setMassBase(1.0000)  
p0.factory.setMassSpread(0.0000)  
p0.factory.setTerminalVelocityBase(400.0000)  
p0.factory.setTerminalVelocitySpread(0.0000)  
# Point factory parameters  
# Renderer parameters  
p0.renderer.setAlphaMode(BaseParticleRenderer.PRALPHAOUT)  
p0.renderer.setUserAlpha(0.22)  
# Sprite parameters  
p0.renderer.setTexture(loader.loadTexture('../../models/sparkle.png'))  
p0.renderer.setColor(LVector4(1.00, 1.00, 1.00, 1.00))  
p0.renderer.setXScaleFlag(1)  
p0.renderer.setYScaleFlag(1)  
p0.renderer.setAnimAngleFlag(0)  
p0.renderer.setInitialXScale(0.000)  
p0.renderer.setFinalXScale(0.0200)  
p0.renderer.setInitialYScale(0.000)  
p0.renderer.setFinalYScale(0.0200)  
p0.renderer.setNonanimatedTheta(0.0000)  
p0.renderer.setAlphaBlendMethod(BaseParticleRenderer.PPNOBLEND)  
p0.renderer.setAlphaDisable(0)  
# Emitter parameters  
p0.emitter.setEmissionType(BaseParticleEmitter.ETRADIATE)  
p0.emitter.setAmplitude(1.0000)  
p0.emitter.setAmplitudeSpread(0.0000)  
p0.emitter.setOffsetForce(LVector3(0.0000, 0.0000, 0.0000))  
p0.emitter.setExplicitLaunchVector(LVector3(0.0000, 0.0000, 0.0000))  
p0.emitter.setRadiateOrigin(LPoint3(0.0000, 0.0000, 0.0000))  
# Disc parameters  
p0.emitter.setRadius(0.5000)  
self.addParticles(p0)

Diese Eruption wird dann genau auf denselben Punkt der Sonne initialisiert.

self.t = sun  
self.t.setPos(0, 0, 0)  
self.t.reparentTo(render)  
self.setupLights()  
self.p = ParticleEffect()  
self.loadParticleConfig(**'../../models/fireish.ptf'**)

Bei den außergewöhnlichen Bildern werden einfach Bilder geloaded. Auf dieselbe Art und Weise wie auch die Texturen geladen werden.

**def** testbildTex(self):  
 self.origTex = False  
 self.loadTex(**"testbild"**)

## Mittels Maus kann die Kameraposition angepasst werden: Zumindest eine Überkopf-Sicht und parallel der Planentenbahnen

In Panda3D wenn man mit ShowBase arbeitet ist prinzipiell schon eine „Kamera“ für die Maus implementiert. Damit meine ich, dass wenn man sie nicht explizit ausstellt (*base.disableMouse()*) kann man sich mit der Maus in der erstellten Umgebung umsehen.

* Rechte Maustaste 🡺 Rein und raus zoomen
* Linke Maustaste 🡺 sich in die gewünschte Richtung verschieben
* Maus-Rad 🡺 Sich auf dem Punkt wo man sich befindet umschauen

Da es sich um eine Animation handelt, kann diese auch gestoppt werden. Mittels Tasten kann die Geschwindigkeit gedrosselt und beschleunigt werden.

Die ganze Animation kann mithilfe von der Leertaste gestoppt und wieder gestartet werden. Außerdem wird die Geschwindigkeit mithilfe der + und – Tasten geregelt werden.

*# When the mouse is clicked, if the simulation is running pause all the  
# planets and sun, otherwise resume it***if** self.simRunning:  
 **print**(**"Pausing Simulation"**)  
 *# For each planet, check if it is moving and if so, pause it  
 # Sun* **if** self.cbAttDic[**"sunDay"**].isPlaying():  
 self.togglePlanet(**"Sun"**, self.cbAttDic[**"sunDay"**], None,  
 self.skeyEventText)  
 **if** self.cbAttDic[**"mercuryDay"**].isPlaying():  
 self.togglePlanet(**"Mercury"**, self.cbAttDic[**"mercuryDay"**],  
 self.cbAttDic[**"mercuryOrbit"**], self.ykeyEventText)  
 *# Venus* **if** self.cbAttDic[**"venusDay"**].isPlaying():  
 self.togglePlanet(**"Venus"**, self.cbAttDic[**"venusDay"**],  
 self.cbAttDic[**"venusOrbit"**], self.vkeyEventText)  
 *#Earth and moon* **if** self.cbAttDic[**"earthDay"**].isPlaying():  
 self.togglePlanet(**"Earth"**, self.cbAttDic[**"earthDay"**],  
 self.cbAttDic[**"earthOrbit"**], self.ekeyEventText)  
 self.togglePlanet(**"Moon"**, self.cbAttDic[**"moonDay"**],  
 self.cbAttDic[**"moonOrbit"**])  
 *# Mars* **if** self.cbAttDic[**"marsDay"**].isPlaying():  
 self.togglePlanet(**"Mars"**, self.cbAttDic[**"marsDay"**],  
 self.cbAttDic[**"marsOrbit"**], self.mkeyEventText)  
 *# jupiter* **if** self.cbAttDic[**"jupiterDay"**].isPlaying():  
 self.togglePlanet(**"jupiter"**, self.cbAttDic[**"jupiterDay"**],  
 self.cbAttDic[**"jupiterOrbit"**], self.jkeyEventText)  
**else**:  
 *#"The simulation is paused, so resume it* **print**(**"Resuming Simulation"**)  
 *# the not operator does the reverse of the previous code* **if not** self.cbAttDic[**"sunDay"**].isPlaying():  
 self.togglePlanet(**"Sun"**, self.cbAttDic[**"sunDay"**], None,  
 self.skeyEventText)  
 **if not** self.cbAttDic[**"mercuryDay"**].isPlaying():  
 self.togglePlanet(**"Mercury"**, self.cbAttDic[**"mercuryDay"**],  
 self.cbAttDic[**"mercuryOrbit"**], self.ykeyEventText)  
 **if not** self.cbAttDic[**"venusDay"**].isPlaying():  
 self.togglePlanet(**"Venus"**, self.cbAttDic[**"venusDay"**],  
 self.cbAttDic[**"venusOrbit"**], self.vkeyEventText)  
 **if not** self.cbAttDic[**"earthDay"**].isPlaying():  
 self.togglePlanet(**"Earth"**, self.cbAttDic[**"earthDay"**],  
 self.cbAttDic[**"earthOrbit"**], self.ekeyEventText)  
 self.togglePlanet(**"Moon"**, self.cbAttDic[**"moonDay"**],  
 self.cbAttDic[**"moonOrbit"**])  
 **if not** self.cbAttDic[**"marsDay"**].isPlaying():  
 self.togglePlanet(**"Mars"**, self.cbAttDic[**"marsDay"**],  
 self.cbAttDic[**"marsOrbit"**], self.mkeyEventText)  
 **if not** self.cbAttDic[**"jupiterDay"**].isPlaying():  
 self.togglePlanet(**"jupiter"**, self.cbAttDic[**"jupiterDay"**],  
 self.cbAttDic[**"jupiterOrbit"**], self.mkeyEventText)  
*# toggle self.simRunning*self.simRunning = **not** self.simRunning

Bei der Einstellung der Geschwindigkeit wird eine Funktion missbraucht die eigentlich für das Abspielen für Musik gedacht war. Nämlich setPlayRate().

*#sun*self.cbAttDic[**"sunDay"**].setPlayRate(self.cbAttDic[**"sunDay"**].getPlayRate()+1)  
*#earth*self.cbAttDic[**"earthOrbit"**].setPlayRate(self.cbAttDic[**"earthOrbit"**].getPlayRate()+1)  
self.cbAttDic[**"earthDay"**].setPlayRate(self.cbAttDic[**"earthDay"**].getPlayRate()+1)  
*#moon*self.cbAttDic[**"moonOrbit"**].setPlayRate(self.cbAttDic[**"moonOrbit"**].getPlayRate()+1)  
self.cbAttDic[**"moonDay"**].setPlayRate(self.cbAttDic[**"moonDay"**].getPlayRate()+1)  
*#mars*self.cbAttDic[**"marsOrbit"**].setPlayRate(self.cbAttDic[**"marsOrbit"**].getPlayRate()+1)  
self.cbAttDic[**"marsDay"**].setPlayRate(self.cbAttDic[**"marsDay"**].getPlayRate()+1)  
*#mercury*self.cbAttDic[**"mercuryOrbit"**].setPlayRate(self.cbAttDic[**"mercuryOrbit"**].getPlayRate()+1)  
self.cbAttDic[**"mercuryDay"**].setPlayRate(self.cbAttDic[**"mercuryDay"**].getPlayRate()+1)  
*#jupiter*self.cbAttDic[**"jupiterOrbit"**].setPlayRate(self.cbAttDic[**"jupiterOrbit"**].getPlayRate()+1)  
self.cbAttDic[**"jupiterDay"**].setPlayRate(self.cbAttDic[**"jupiterDay"**].getPlayRate()+1)  
*#venus*self.cbAttDic[**"venusOrbit"**].setPlayRate(self.cbAttDic[**"venusOrbit"**].getPlayRate()+1)  
self.cbAttDic[**"venusDay"**].setPlayRate(self.cbAttDic[**"venusDay"**].getPlayRate()+1)

Mittels Mausklick kann eine Punktlichtquelle und die Textierung ein- und ausgeschaltet werden.

Wir können die Texturen ein und ausschalten mittels der Taste „T“. Jedoch haben wir nicht das ein und ausschalten der Punktlichtquelle implementiert.

Schatten: Auch Monde und Planeten werfen Schatten.

Durch die Implementierung der Punktlichtquellen werfen alle Planeten Schatten. Wir haben insgesamt 6 Punktlichtquellen erstellt, weil wir Probleme hatten mit nur einer, da die Sonne dann selbst nicht scheint. Deswegen haben wir rund um die Sonne 6 Lichtquellen platziert.

plight2 = PointLight(**'plight'**)  
plight2.setColor(VBase4(0.8, 0.8, 0.8, 1))  
plnp2 = render.attachNewNode(plight2)  
plnp2.setPos(0, 0, -3.8)  
render.setLight(plnp2)  
  
plight3 = PointLight(**'plight'**)  
plight3.setColor(VBase4(0.8, 0.8, 0.8, 1))  
plnp3 = render.attachNewNode(plight3)  
plnp3.setPos(3.8, 0, 0)  
render.setLight(plnp3)

# Quellen

* [1] Pyglet
  + Online: <https://bitbucket.org/pyglet/pyglet/wiki/Home>
    - Zuletzt besucht am 24.11.2015
* [2] Panda3D
  + Online: <https://www.panda3d.org/>
    - Zuletzt besucht am 24.11.2015
* [3] Pygame
  + Online: <http://pygame.org/hifi.html>
    - Zuletzt besucht am 24.11.2015