SOLAR SYSTEM

Dienesch Florian | Rathbauer Alexander

Contents

Allgemein

Der Sourcecode ist auf https://github.com/arathbauer/Solar-System zu finden.

Aufgabenstellung

Wir wollen nun unser Wissen aus Medientechnik und SEW nützen um eine etwas kreativere Applikation zu erstellen.

Eine wichtige Library zur Erstellung von Games mit 3D-Grafik ist Pygame. Die 3D-Unterstützung wird mittels PyOpenGL erreicht.

Die Kombination ermöglicht eine einfache und schnelle Entwicklung.

Während pygame sich um Fensteraufbau, Kollisionen und Events kümmert, sind grafische Objekte mittel OpenGL möglich.

Die Aufgabenstellung:

Erstellen Sie 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.

Hinweise:

- 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 wird die Library Pillow benötigt! Die Community unterstützt Sie bei der Verwendung.

Tutorials:

- Pygame: https://www.youtube.com/watch?v=K5F-aGDIYaM

Viel Erfolg!

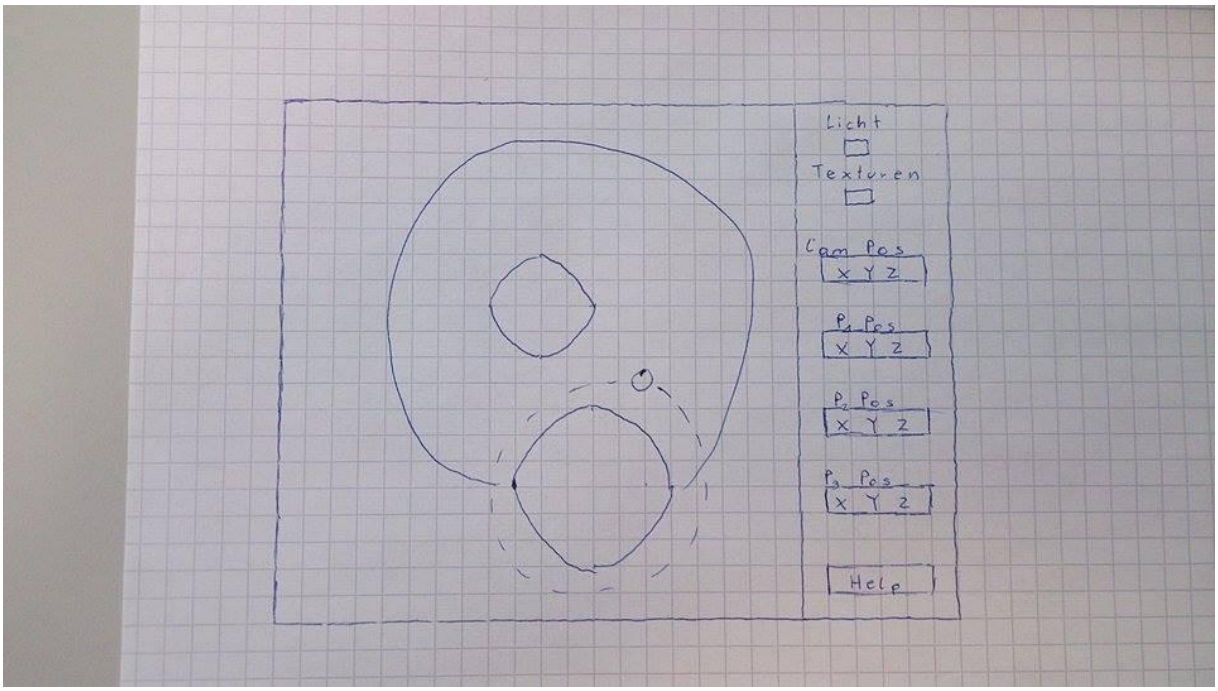
Zeitaufzeichnung

User Story	**Verantwortlicher**	**Zeit [std]**		**Status**			
		Gesch.	Wirkl	Impl	Tests	Doku	Fertig
Einarbeitung	D,R	4	3	X			X
UML	R	3	3	X			X
Galaxy anzeigen	D	1	0.5	X	X	X	X
Splashscreen erstellen	R	2	1	X	X	X	X
Splashscreen einbinden	D	1	0.5	X	X	X	X
Texturen erstellen oder finden	R	0.5	0.5	X	X	X	X
Texturen auf Objekte legen	R	2	1	X	X	X	X
Modelle erstellen	R	0.5	0.5	X	X	X	X
Modelle um sich selbst drehen	R	4	1	X	X	X	X
Modelle um andere Modelle drehen	R	4	3	X	X	X	X
Modelle in ellipsenbahn drehen lassen	D	4	2	X	X	X	X
Animation	D	1	0.5	X	X	X	X
Geschwindigkeit änderbar	R	1	0.5	X	X	X	X
Licht Quelle gesetzt	D	6	3	X	X	X	X
Texturen	R	6	3	X	X	X	X
Steuerbare Kamera	D	3	0.5	X	X	X	X
Hintergrund	R	1.5	0.5	X	X	X	X
Userinterface	D	2	1	X	X	X	X
Tests	R, D	3	4.5	X		X	X
Zusammenfassung		30	72				

D...Dienesch
R...Rathbauer
X...Fertig
Y...In Arbeit

Arbeitsschritte

Skizze Layout



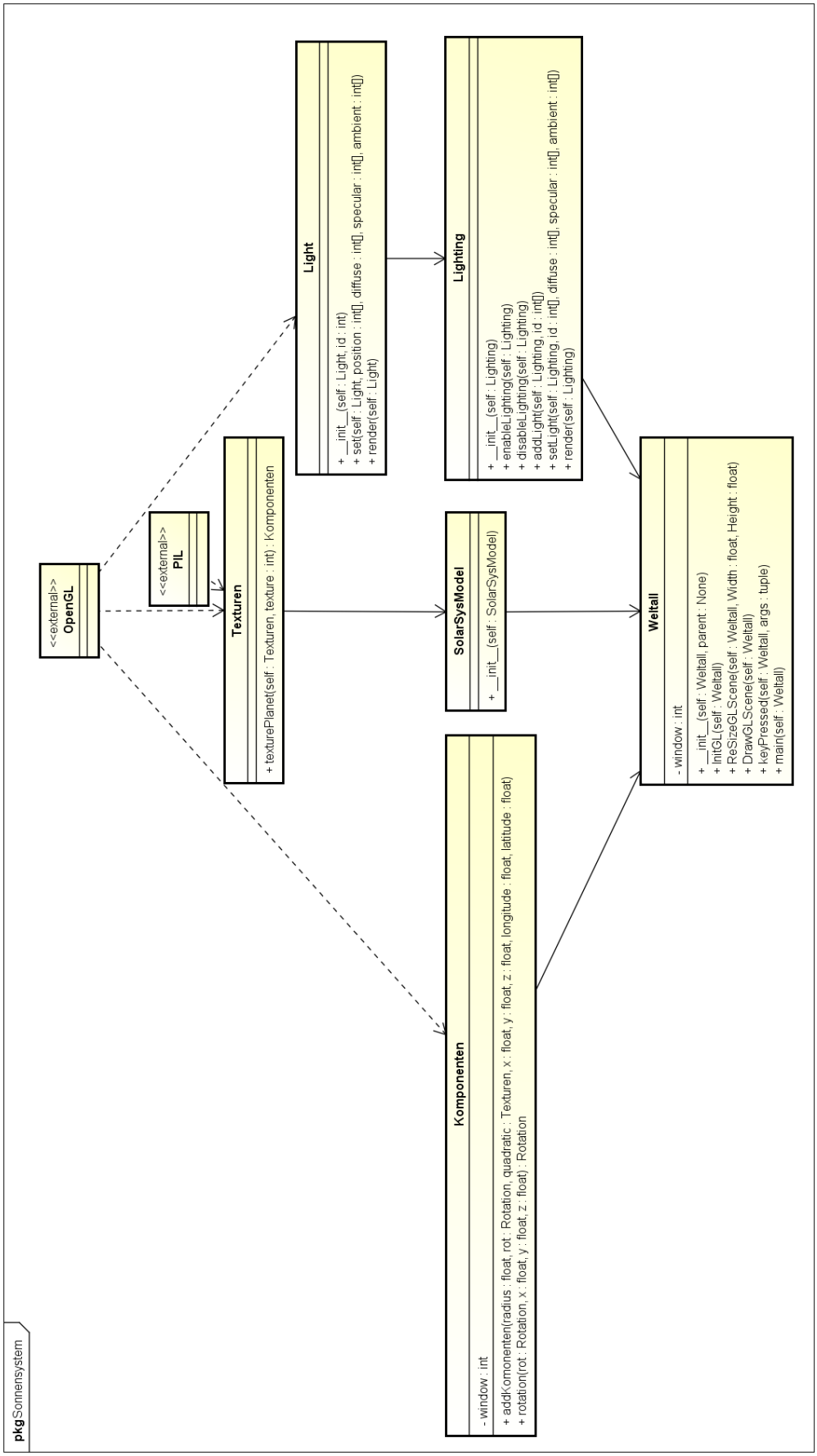
Im Bild können Sie unsere derzeitige Idee von unserem Design finden.

Bevor der Benutzer dieses Fenster sieht, wird davor noch ein paar Sekunden ein Splashscreen eingeblendet.

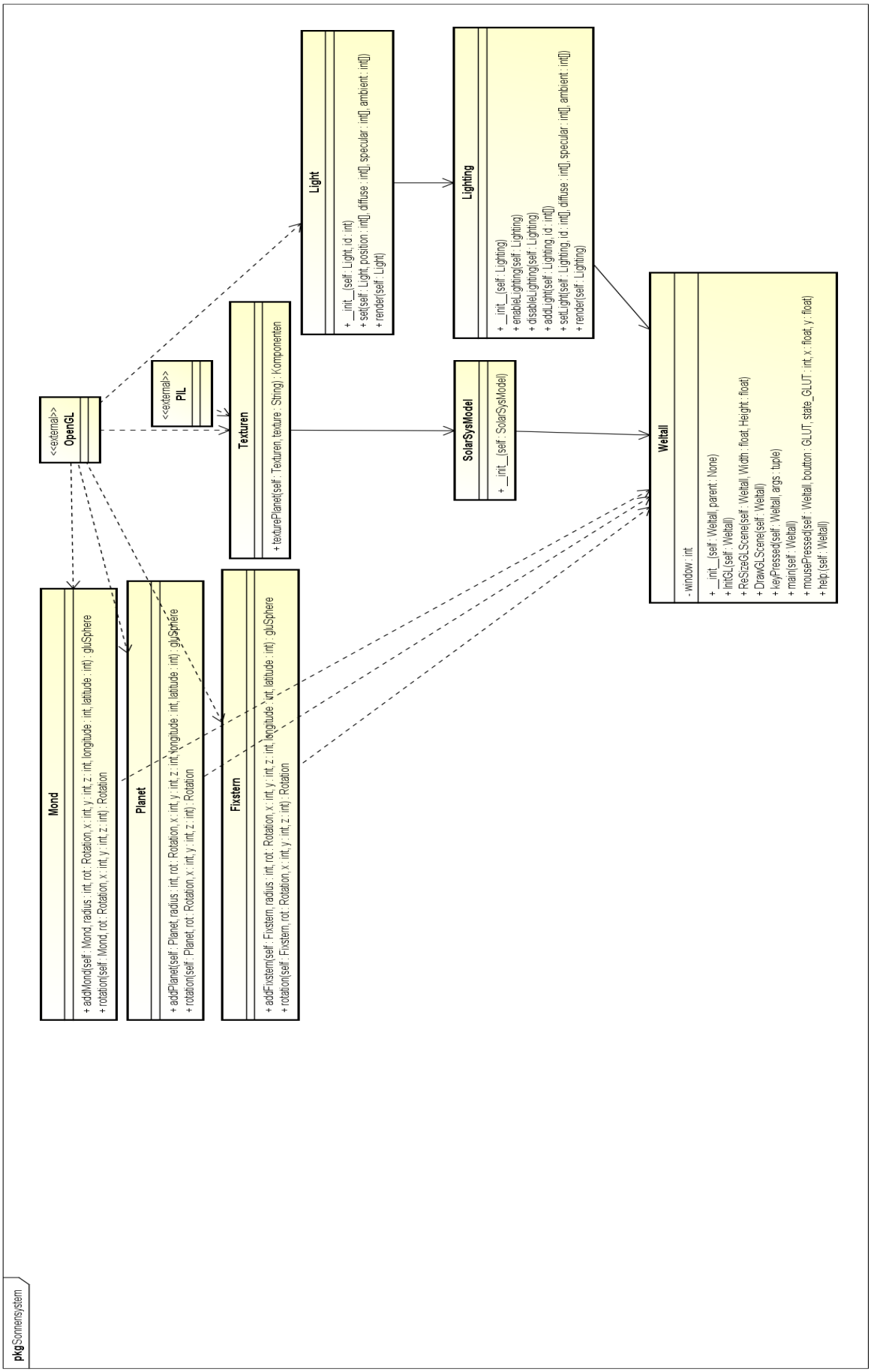
Auf der rechten Seite des Bildes, kann man Informationen erfahren über den derzeitigen Aufenthaltsortes eines Planeten und man kann zudem auch das Licht oder Texturen ein beziehungsweise ausblenden.

Unten befindet sich ein „Help“ Button. Wird dieser gedrückt, öffnet sich ein weiteres Fenster bei dem die Benutzerinteraktion mit Tastatur oder Maus erklärt wird.

UML – Klassendiagramm

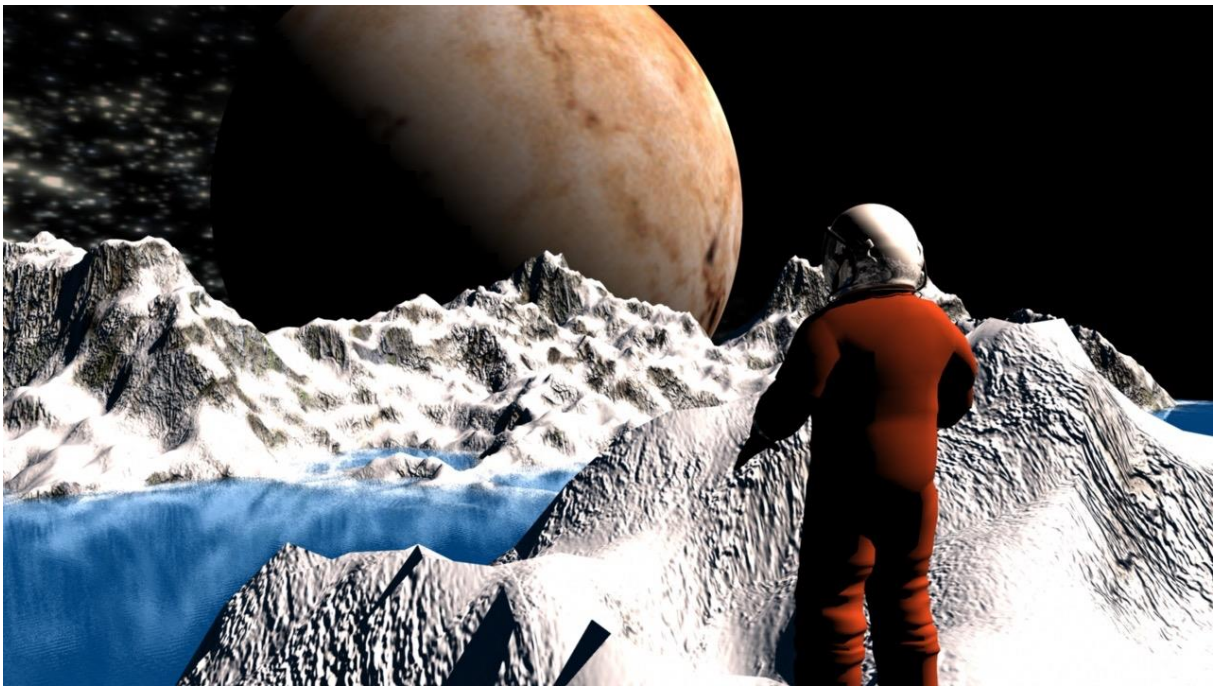


V1.0


pkg Sonnensystem
<<external>>
OpenGL
<<external>>
PIL
Light
+ __init__(self: Light, id: int)
+ set(self: Light, position: int[], diffuse: int[], specular: int[], ambient: int[])
+ render(self: Light)
Lighting
+ __init__(self: Lighting)
+ enableLighting(self: Lighting)
+ disableLighting(self: Lighting)
+ addLight(self: Lighting, id: int[])
+ setLight(self: Lighting, id: int[], diffuse: int[], specular: int[], ambient: int[])
+ render(self: Lighting)
Texturen
+ texturePlanet(self: Texturen, texture: String): Komponenten
SolarSysModel
+ __init__(self: SolarSysModel)
Weltall
- window: int
+ __init__(self: Weltall, parent: None)
+ InitGL(self: Weltall)
+ ReSizeGLScene(self: Weltall, Width: float, Height: float)
+ DrawGLScene(self: Weltall)
+ keyPressed(self: Weltall, args: tuple)
+ main(self: Weltall)
+ mousePressed(self: Weltall, boutton: GLUT, state_GLUT: int, x: float, y: float)
+ help(self: Weltall)
Mond
+ addMond(self: Mond, radius: int, rot: Rotation, x: int, y: int, z: int, longitude: int, latitude: int): gluSphere
+ rotation(self: Mond, rot: Rotation, x: int, y: int, z: int): Rotation
Planet
+ addPlanet(self: Planet, radius: int, rot: Rotation, x: int, y: int, z: int, longitude: int, latitude: int): gluSphere
+ rotation(self: Planet, rot: Rotation, x: int, y: int, z: int): Rotation
Fixstern
+ addFixstern(self: Fixstern, radius: int, rot: Rotation, x: int, y: int, z: int, longitude: int, latitude: int): gluSphere
+ rotation(self: Fixstern, rot: Rotation, x: int, y: int, z: int): Rotation
powered by Astah

Splashscreen

Hier ist ein Bild von dem von uns erstellten Splashscreen für die Animation des Sonnensystem



Sonnensystem

Informationen über das Sonnensystem

Name	**Durchmesser**	**Fluchtgeschwindigkeit**
Sonne	1 390 000 km	
Merkur	4 900 km	3,70 km/s
Venus	12 100 km	8,87 km/s
Erde	12 800 km	9,77 km/s
Mars	6 800 km	3,69 km/s
Jupiter	143 000 km	23 km/s
Saturn	120 500 km	8,8 km/s
Uranus	51 100 km	8,6 km/s
Neptun	49 500 km	11 km/s

Lichtpunkt setzen

```
class Light(object):
    def __init__(self, id):
        self.id = id
        glEnable(id)
        self.position = []
        self.diffuse = []
```

```
        self.specular = []
        self.ambient = []

    def set(self, position, diffuse, specular, ambient):
        """
        Method set
        This method sets the light
        """
        self.position = position
        self.diffuse = diffuse
        self.specular = specular
        self.ambient = ambient

    def render(self):
        """
        Method render
        This method render the light
        """
        glLight(self.id, GL_POSITION, self.position)
        glLight(self.id, GL_DIFFUSE, self.diffuse)
        glLight(self.id, GL_SPECULAR, self.specular)
        glLight(self.id, GL_AMBIENT, self.ambient)
```

Texturen auf Objekte legen

```
class Texturen(object):
    """
    Class Texturen
    This class loads a texture
    """

    def textureOrbit(self, imageName):
        """
        Method texturePlanet
        Load an image from a file using PIL, produces 3 textures of filter types.
        Converts the paletted image to RGB format.
        """
        # load the image
        im = open(imageName)
        try:
            # Note the conversion to RGB the crate bitmap is paletted!
            im = im.convert('RGB')
            ix, iy, image = im.size[0], im.size[1], im.tostring("raw", "RGBA", 0, -1)
        except SystemError:
            ix, iy, image = im.size[0], im.size[1], im.tostring("raw", "RGBX", 0, -1)
        assert ix*iy*4 == len(image), """Image size != expected array size"""
        IDs = []
        # a Nearest-filtered texture...
        ID = glGenTextures(1)
        IDs.append(ID)
        glBindTexture(GL_TEXTURE_2D, ID)
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1)
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
```

```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
    glTexImage2D(GL_TEXTURE_2D, 0, 3, ix, iy, 0, GL_RGBA, GL_UNSIGNED_BYTE, image)
    # linear-filtered
    ID = glGenTextures(1)
    IDs.append(ID)
    glBindTexture(GL_TEXTURE_2D, ID)
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    glTexImage2D(GL_TEXTURE_2D, 0, 3, ix, iy, 0, GL_RGBA, GL_UNSIGNED_BYTE, image)
    # linear + mip-mapping
    ID = glGenTextures(1)
    IDs.append(ID)
    glBindTexture(GL_TEXTURE_2D, ID)
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_NEAREST)
    gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, ix, iy, GL_RGBA, GL_UNSIGNED_BYTE,
image)
    return IDs
```

Sich um andere Objekte drehen

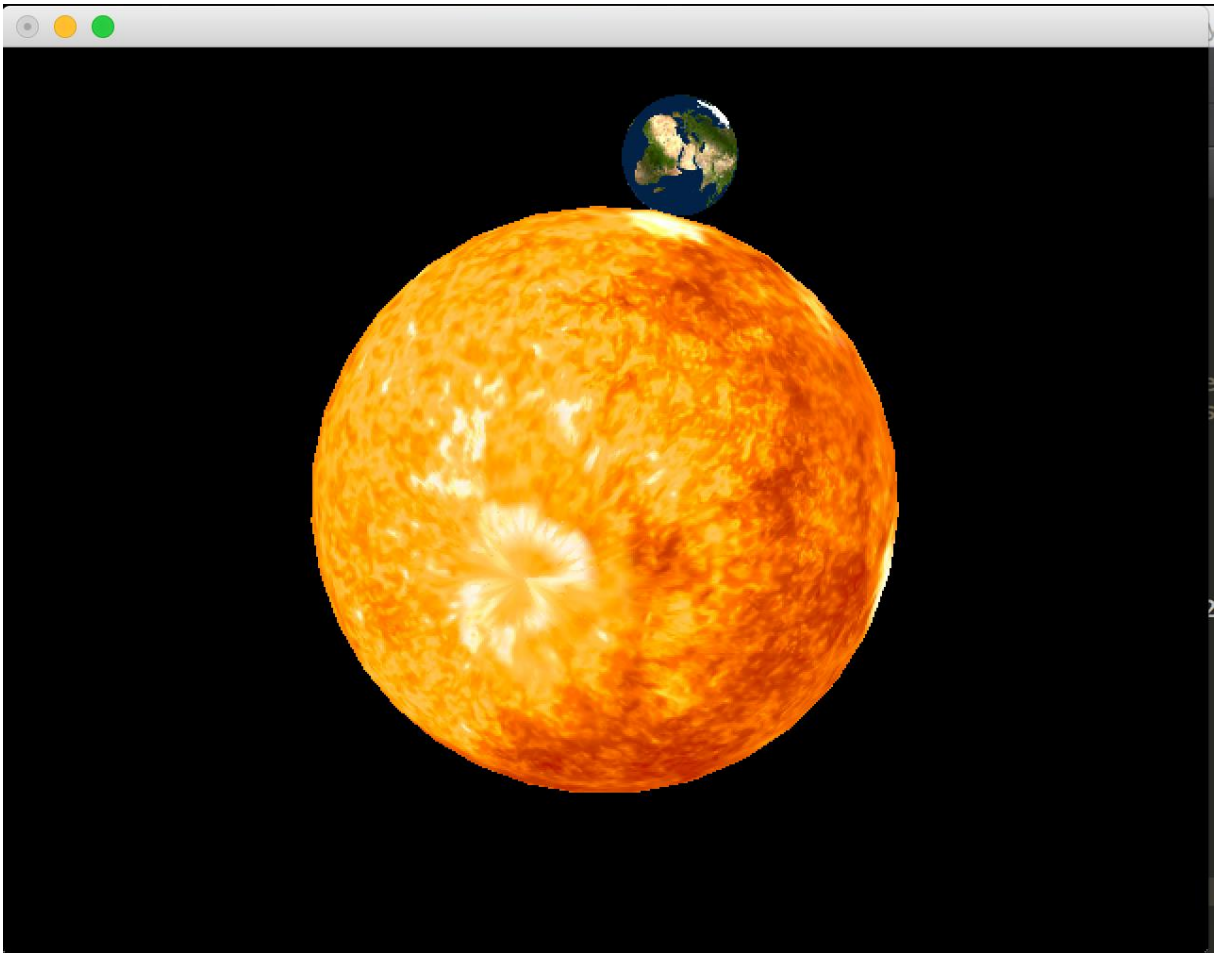
```
def addMond(self, radius, rot, x, y, z, longitude, latitude):
    """
    Method addMond
    This Method adds a mond to an existing universe
    :param radius: size of the planet
    :param rot: roation
    :param x: translation to x
    :param y: --
    :param z: --
    :param longitude: how many vertexes
    :param latitude:
    :return:
    """
    try:
        glLoadIdentity()
        glTranslatef(x, y, z)

        glRotatef(rot[1], 0.0, 1.0, 0.0)

        glTranslatef(3.0, 0.0, 3.0)

        # create a mond
        quadric = gluNewQuadric()
        gluQuadricTexture(quadric, GL_TRUE)
        gluSphere(quadric, radius, longitude, latitude)
    except Exception:
        print("Please enter an integer")
```

Implementierung

V 0.1 am 02.03.2015

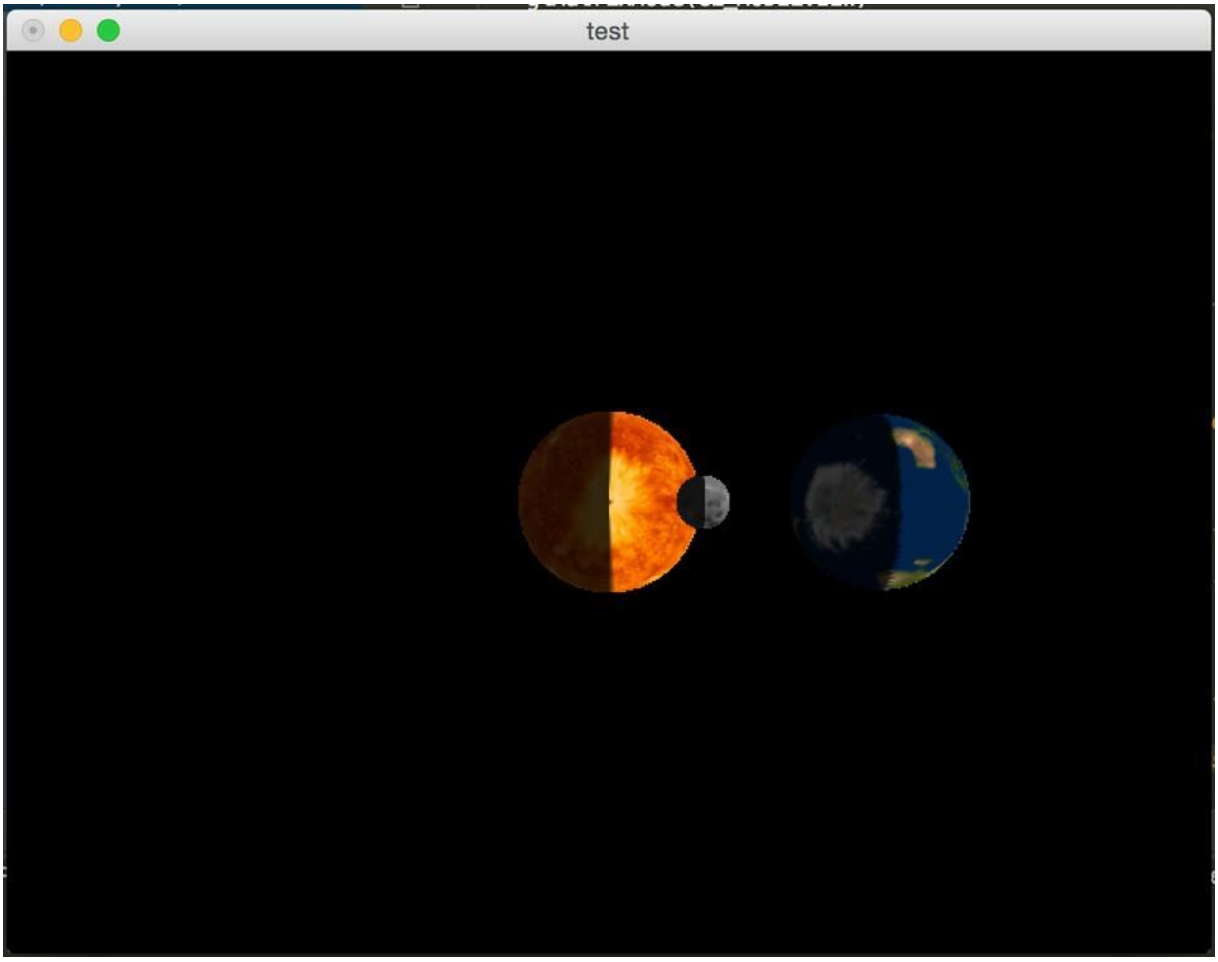
Wie verwenden für die Implementierung pyopengl, pygame sowie pillow.
Die Steuerung des Benutzers wir mittels pygame realisiert, die Erstellung von Objekten, licht usw. mit pyopengl und das importieren für Texturen verwenden wir pillow.
Wobei man sagen kann, dass die Library von pyopengl sehr viel

Damit wir beide den aktuellen Stand haben verwenden wir ein privates Repository auf Github das von Herrn Rathbauer erstellt worden ist.



V0.4 am 09.03.2015

Es wurden Licht und Geschwindigkeit mit Benutzersteuerung implementiert.



V0.5 am 16.03.2015

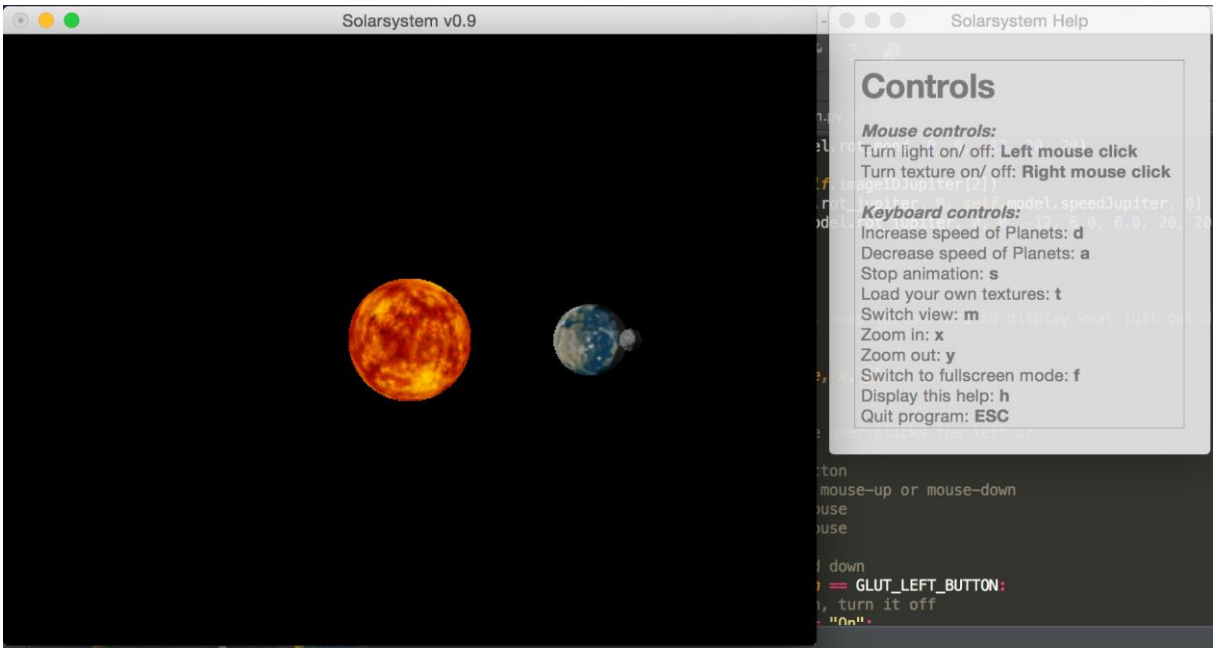
Implementiert:

- Ein/Ausschalten von Texturen
- Ein/Ausschlaten von Lichtern
- Splashscreen implementiert
- Usersteuerung implementiert
- Fullscreen implementierung

ToDo:
Implementierung von Steuerbare Kammera
Implementierung von Monde
Steuerungserklärung

V0.9 am 27.03.2015

Es wurden soweit alle Anforderungen der Auftraggeber implementiert.



Es wurden auch Testfälle erstellt.

Es wurde auch wie im Screenshot zu sehen ist, ein weiteres Feature hinzugefügt (Laden der eigenen Texturen). Falls der User nicht mit unseren Texturen zufrieden ist (weil sie zu unscharf, ..) kann er währen der Laufzeit des Programms seine eigene Texturen laden. Dies hat den entscheidenden Vorteil, dass er seine Texturen nicht umbenennen muss und in den Order des Programms verschieben muss.

Probleme

Derzeit sind noch keine großen Probleme aufgetreten, außer dass das Licht nicht so wie gewollt funktioniert hatte.

Umsteigen auf Python 3.4, weil Version 3.3 nicht mit der neuesten Version von PyQt (5) nicht kompatibel ist.

Wie im Kapitel Testfälle zu lesen ist, hatten wir bei der Erstellung der Testfälle einen Fehler, dass diese nicht ausführbar waren.

Wir hatten zuerst um unittest zu schreiben, das framework nose verwendet. Dies hat aber nicht funktioniert, da unsere Module nicht gefunden werden können. Ein möglicher workaround wäre eventuell gewesen unseren PYTHONPATH in den Umgebungsvariablen des Systems zu ändern beziehungsweise hinzuzufügen. Wir sind dann aber zum Beschluss gekommen, dass wir das schon in Pycharm integrierte unittest framework verwenden.

Testfälle

Pylint

Ein Linter hilft uns schwächen von unserem Code aufzuzeigen, jedoch hat es nicht das letzte Wort. Deshalb haben wir wie im Source-Code zu sehen ein paar Tweaks gemacht, die dem Linter sagen, dass er manche Inhalte ignorieren soll.
Bei der „Model"-Klasse war dies z.b nötig. (too-many-instance-attributes, invalid-name, …)
Weiters ist auch anzumerken, dass auch getter und setter Klassen laut dem Linter mit einem Docstring verseht werden müssen, was aber nicht sehr klug ist, da diese Methoden ohnehin schon sehr kurz und verständlich sind und einen weiteren Aufwand für den Programmierer ist die Kommentare bei einer Änderung zu entfernen.

Mit Pylint erhalten wir folgende Punktezahl auf unsere Dateien:

```
Global evaluation
-----------------
Your code has been rated at 10.00/10 (previous run: 9.67/10, +0.33)
```

solarSysModel:

```
Global evaluation
-----------------
Your code has been rated at 7.50/10 (previous run: 6.39/10, +1.11)
```

lighting:

Wobei Fehler wie zum Beispiel (too-many-arguments oder redefined-builtin) angezeigt werden, obwohl diese nicht stimmen.

```
Global evaluation
-----------------
Your code has been rated at 7.50/10 (previous run: 1.88/10, +5.62)
```

planet:

Wobei Fehler wie zum Beispiel (too-many-arguments oder redefined-builtin) angezeigt werden, obwohl diese nicht stimmen.

```
Global evaluation
-----------------
Your code has been rated at 9.49/10 (previous run: 8.21/10, +1.28)
```

texturen:

Wobei Fehler wie zum Beispiel

no-self-use bei Methode oder redefined-outer-name bei ID

```
Global evaluation
-----------------
Your code has been rated at 9.35/10 (previous run: 8.82/10, +0.53)
```

weltall:

Unittesting

„The Python unit testing framework, sometimes referred to as “PyUnit,” is a Python language version of JUnit, by Kent Beck and Erich Gamma. JUnit is, in turn, a Java version of Kent’s Smalltalk testing framework. Each is the de facto standard unit testing framework for its respective language.

unittest supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework. The unittest module provides classes that make it easy to support these qualities for a set of tests“ [2]

Beim Unittesting gab es zuerst ein Problem da wenn zum Beispiel ein Planet getestet wird, hat sich Python ohne Fehlercode beendet. Nach einer Stunde haben wir jedoch herausgefundet, dass in der setUp Methode von den Unittests pyopengl initialisiert werden muss und ein glutWindow geöffnet werden muss. Dieses muss aber nur kurz geöffnet werden und anschließend kann es wieder geschlossen werden.

Hier ist ein kleiner Auszug von einem Unittest von der Klasse Fixstern:

```python
class TestFixstern(TestCase):

    def setUp(self):
        glutInit(sys.argv)
        glutCreateWindow("test")

        self.fixstern = Fixstern()
        self.model = SolarSunModel()

    def tearDown(self):
        self.fixstern = None
        self.model = None

    def test_addFixstern(self):
        self.assertEqual(self.fixstern.addFixstern("dd", self.model.rot_mond, 0, 0, 0, 20, 20), None)
```

Entwicklungsumgebung - Tools

Pycharm

“PyQt is a set of Python v2 and v3 bindings for Digia's Qt application framework and runs on all platforms supported by Qt including Windows, MacOS/X and Linux. PyQt5 supports Qt v5. PyQt4 supports Qt v4 and will build against Qt v5. The bindings are implemented as a set of Python modules and contain over 620 classes.

Digia have announced that support for Qt v4 will cease at the end of 2015. PyQt5 and Qt v5 are strongly recommended for all new development.

PyQt is dual licensed on all supported platforms under the GNU GPL v3 and the Riverbank Commercial License. Unlike Qt, PyQt is not available under the LGPL. You can purchase the commercial version of PyQt here. More information about licensing can be found in the License FAQ.

PyQt does not include a copy of Qt. You must obtain a correctly licensed copy of Qt yourself. However, a binary Windows installers of the GPL version of both PyQt5 and PyQt4 are provided and this includes a copy of the LGPL version of Qt."[3]

Python OpenGL

"PyOpenGL is the most common cross platform Python binding to OpenGL and related APIs. The binding is created using the standard ctypes library, and is provided under an extremely liberal BSD-style Open-Source license."[5]

Pillow

"Pillow is the 'friendly' PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors."[6]

PyQT5

"PyQt is a set of Python v2 and v3 bindings for Digia's Qt application framework and runs on all platforms supported by Qt including Windows, MacOS/X and Linux. PyQt5 supports Qt v5. PyQt4 supports Qt v4 and will build against Qt v5. The bindings are implemented as a set of Python modules and contain over 620 classes.

Digia have announced that support for Qt v4 will cease at the end of 2015. PyQt5 and Qt v5 are strongly recommended for all new development.

PyQt is dual licensed on all supported platforms under the GNU GPL v3 and the Riverbank Commercial License. Unlike Qt, PyQt is not available under the LGPL. You can purchase the commercial version of PyQt here. More information about licensing can be found in the License FAQ.

PyQt does not include a copy of Qt. You must obtain a correctly licensed copy of Qt yourself. However, a binary Windows installers of the GPL version of both PyQt5 and PyQt4 are provided and this includes a copy of the LGPL version of Qt."[8]

Sphinx

"Sphinx is a tool that makes it easy to create intelligent and beautiful documentation, written by Georg Brandl and licensed under the BSD license.

It was originally created for the new Python documentation, and it has excellent facilities for the documentation of Python projects, but C/C++ is already supported as well, and it is planned to add special support for other languages as well. Of course, this site is also created from reStructuredText sources using Sphinx! The following features should be highlighted:

- **Output formats:** HTML (including Windows HTML Help), LaTeX (for printable PDF versions), ePub, Texinfo, manual pages, plain text
- **Extensive cross-references:** semantic markup and automatic links for functions, classes, citations, glossary terms and similar pieces of information
- **Hierarchical structure:** easy definition of a document tree, with automatic links to siblings, parents and children
- **Automatic indices:** general index as well as a language-specific module indices

- **Code handling:** automatic highlighting using the Pygments highlighter
- **Extensions:** automatic testing of code snippets, inclusion of docstrings from Python modules (API docs), and more
- **Contributed extensions:** more than 50 extensions contributed by users in a second repository; most of them installable from PyPI

Sphinx uses reStructuredText as its markup language, and many of its strengths come from the power and straightforwardness of reStructuredText and its parsing and translating suite, the Docutils."[4]

Pylint

„Pylint is a Python source code analyzer which looks for programming errors, helps enforcing a coding standard and sniffs for some code smells (as defined in Martin Fowler's Refactoring book).

Pylint has many rules enabled by default, way too much to silence them all on a minimally sized program. It's highly configurable and handle pragmas to control it from within your code. Additionally, it is possible to write plugins to add your own checks.

It's a free software distributed under the GNU Public Licence." [1]

Code Snippets

Pillow

Wird zum laden von Texturen verwendet.

```
im = open(imageName)
try:
   # Note the conversion to RGB the crate bitmap is paletted!
   im = im.convert('RGB')
   ix, iy, image = im.size[0], im.size[1], im.tostring("raw", "RGBA", 0, -1)
except SystemError:
   ix, iy, image = im.size[0], im.size[1], im.tostring("raw", "RGBX", 0, -1)
assert ix*iy*4 == len(image), """Image size != expected array size"""
IDs = []
# a Nearest-filtered texture...
ID = glGenTextures(1)
IDs.append(ID)
glBindTexture(GL_TEXTURE_2D, ID)
glPixelStorei(GL_UNPACK_ALIGNMENT, 1)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
glTexImage2D(GL_TEXTURE_2D, 0, 3, ix, iy, 0, GL_RGBA, GL_UNSIGNED_BYTE, image)
```

Python OpenGL

```
  def InitGL(self):
     """
     Method InitGL
     This Method initializes our Solar System
     It sets the lighting and enables or rather sets the textures
     """
     # make the objects transparent
```

```
        glEnable(GL_DEPTH_TEST)
        # enable texturing
        glEnable(GL_TEXTURE_2D)

        # enable lighting
        self.lighting.enableLighting()
        # add a new light
        self.lighting.addLight(GL_LIGHT0)

        # set the position of the light to the sun
        self.lighting.setLight(GL_LIGHT0, self.model.lightOn[0], self.model.lightOn[1],
                               self.model.lightOn[2], self.model.lightOn[3])

        # set the textures when starting the program
        if self.model.fileSet == False:
            try:
                self.imageIDMoon = self.model.t.textureOrbit(self.model.file[0])
                self.imageIDEarth = self.model.t.textureOrbit(self.model.file[1])
                self.imageIDSun = self.model.t.textureOrbit(self.model.file[2])
                self.imageIDJupiter = self.model.t.textureOrbit(self.model.file[3])
            except:
                print("Can't find the textures!")
        # load the textures which are assigned by the user
        else:
            self.imageIDMoon = self.model.t.textureOrbit(self.model.file[0][0])
            self.imageIDEarth = self.model.t.textureOrbit(self.model.file[1][0])
            self.imageIDSun = self.model.t.textureOrbit(self.model.file[2][0])
            self.imageIDJupiter = self.model.t.textureOrbit(self.model.file[3][0])


        # open the help window
        self.help()

        # set color of the back of the planet not to black
        glEnable(GL_COLOR_MATERIAL)
        # set the Backgroundcolor
        glClearColor(0.0, 0.0, 0.0, 0.0)
```

Quellen

[1] Pylint: https://pypi.python.org/pypi/pylint, gesehen am 28.03.2015

[2] Unittest: https://docs.python.org/2/library/unittest.html, gesehen am 28.03.2015

[3] Pycharm : https://www.jetbrains.com/pycharm/ gesehen am: 23.03.2015

[4] Sphinx: http://sphinx-doc.org/ gesehen am:23.03.2015

[5] PythonOpenGL: http://pyopengl.sourceforge.net/ gesehen am: 23.03.2015

[6] Pillow: https://pillow.readthedocs.org/ gesehen am: 23.03.2015

[7] PyQt5: http://www.riverbankcomputing.com/software/pyqt/intro gesehen am: 29.03.2015