Space Adventure

Aileen Jurkosek und Caterina Sophia Thimm

Inhalte des Projekts

- Hauptplanet
  - Kamerafahrten
  - Hindernisse
  - Textur
- Character
  - Modell
  - Animation (Laufen & Springen)
- Umgebung
  - Skybox
  - Background Objekte
  - Collectables
- Shader
  - Toon Shader
  - Negativ Shader


Hauptplanet

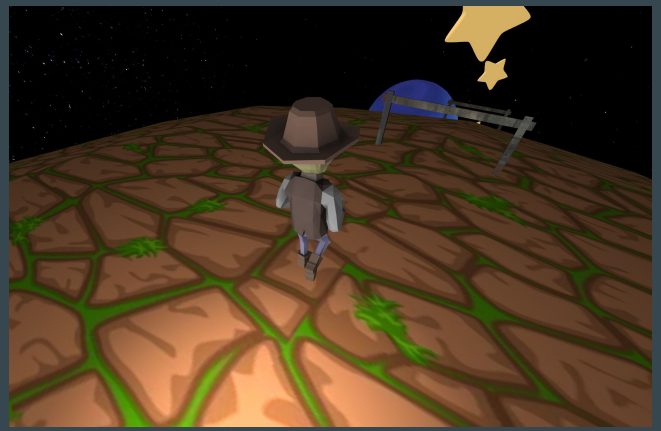
Kameraperspektiven

perspektivisch

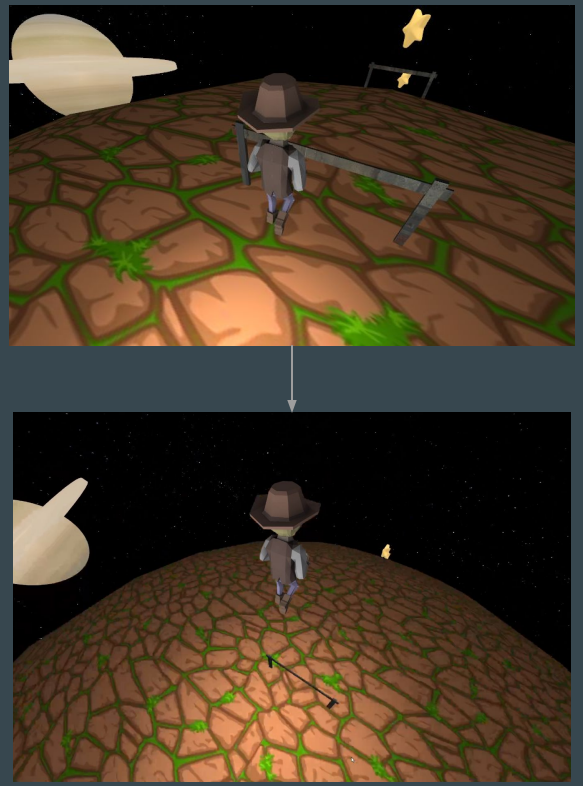
orthographisch

Kameraperspektive

```
if (!turnedCamera) {
    if (window.getKeyState(GLFW_KEY_F1)) {
        tCamera.translateLocal(Vector3f(20.0f, 8f, -10.0f))
        tCamera.rotateLocal(Math.toRadians(-45.0f), Math.toRadians(90.0f),
Math.toRadians(90.0f))
        turnedCamera = true
    }
}else {
    if (window.getKeyState(GLFW_KEY_F2)) {
        tCamera.rotateLocalBack(
            Math.toRadians(45.0f),
            Math.toRadians(-90.0f),
            Math.toRadians(-90.0f)
        )
        tCamera.translateLocal(Vector3f(20.0f, 8f,-10.0f).negate())
        turnedCamera = false
    }
}
```



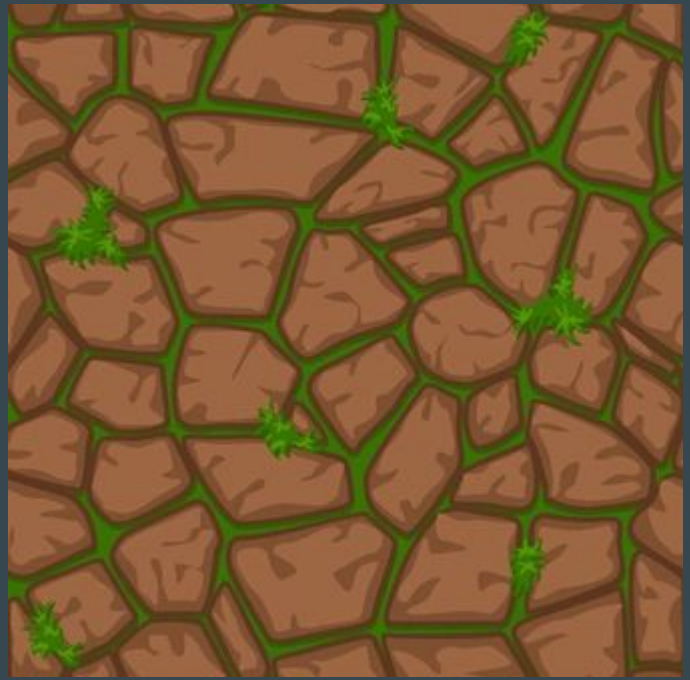
Hindernisse



```
fun checkCollisionWithObstacles (): Boolean {
    for (i in 0 until obstacleAmount) {
        if (i % 2 == 0) {
            val currentDiff = pointDistance3d(
                player!!.x(),
                player!!.y(),
                player!!.z(),
                obstacles[i].x(),
                obstacles[i].y(),
                obstacles[i].z()
            )
            if (currentDiff <= 0.01) {
                return true
            }
        }
    }
    return false
}
```

Textur

```
val emitTex = Texture2D( "project/assets/textures/grass.PNG" , true)
val diffTex = Texture2D( "project/assets/textures/grass.PNG" , true)
val specTex = Texture2D( "project/assets/textures/grass.PNG" , true)

val groundMaterial = Material(diffTex , emitTex, specTex, 1.0f,
Vector2f(20.0f))

emitTex.setTexParams(
    GL_REPEAT,
    GL_REPEAT,
    GL11.GL_LINEAR_MIPMAP_LINEAR,
    GL_LINEAR
)      //Linear = zwischen farbwerten interpolieren
diffTex.setTexParams( GL_REPEAT, GL_REPEAT, GL11.GL_LINEAR_MIPMAP_LINEAR,
GL_LINEAR)
specTex.setTexParams( GL_REPEAT, GL_REPEAT, GL11.GL_LINEAR_MIPMAP_LINEAR,
GL_LINEAR)

groundMesh  = Mesh(objMeshGround.vertexData, objMeshGround.indexData,
objVertexAttributes , groundMaterial)
```




Character

Animation

char_0.obj

...

char_5.obj

...

char_10.obj

...

char_20.obj

Animation

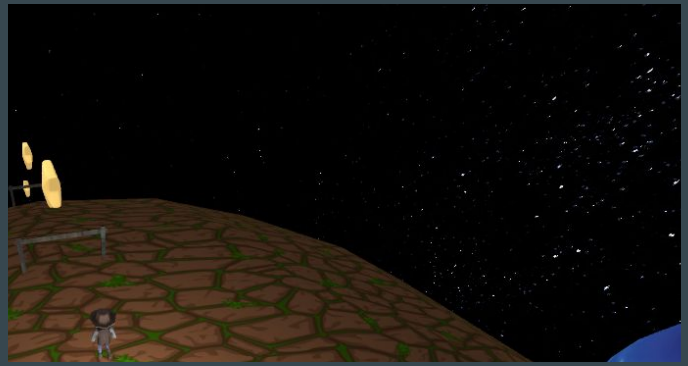
```
character = Animation("project/assets/character/char_", 0, 19, 0f, 180f, 0f)
character.setParent(player!!)
```

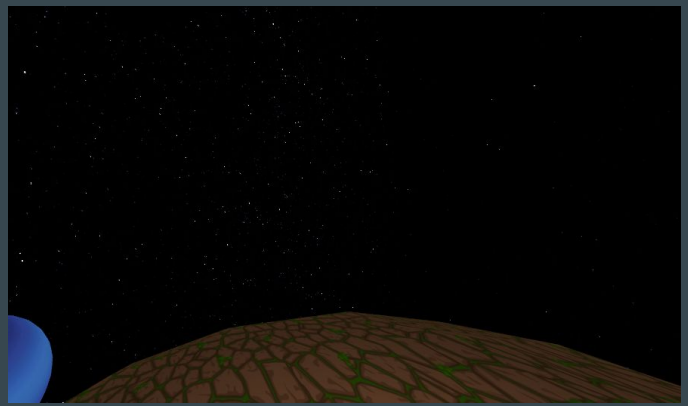
```
fun render(shaderProgram: ShaderProgram, dt: Float) {
    if (movement) {
        animationList[currentFrame]?.render(shaderProgram)
    }
    else {
        animationList[5]?.render(shaderProgram)
    }
}

fun update() {
    if (renderCycle == 0) {
        currentFrame++
    } else if (renderCycle >= 3) {
        renderCycle = -1
    }
    renderCycle++
    if (currentFrame > endNumber) {
        currentFrame = startNumber
    }
}
```


Umgebung

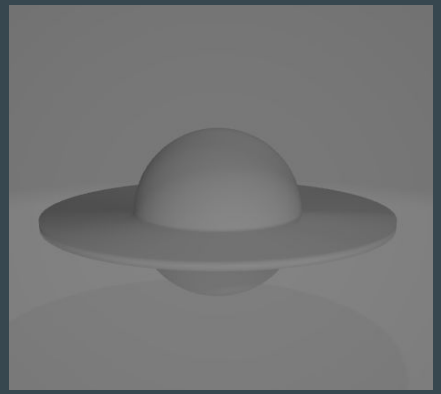
Skybox

```
fun loadCubeMap(faces: ArrayList<String>) : Int {
(...)
   glBindTexture(GL30.GL_TEXTURE_CUBE_MAP, cubeMaptexID)

   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE)
   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE)
   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R,
GL_CLAMP_TO_EDGE)
   glEnable(AMDSeamlessCubemapPerTexture.GL_TEXTURE_CUBE_MAP_SEAMLESS)


   for(i in 0..5) {
       val x = BufferUtils.createIntBuffer(1)
       val y = BufferUtils.createIntBuffer(1)
       val readChannels = BufferUtils.createIntBuffer(1)
       val imageData = STBImage.stbi_load(faces[i], x, y, readChannels, 4)
               ?: throw Exception("Image file \"" + faces[i] + "\"
couldn't be read:\n" + STBImage.stbi_failure_reason())
       STBImage.stbi_set_flip_vertically_on_load(false)

       GL30.glTexImage2D(GL30.GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0,
GL30.GL_RGBA8, x.get(), y.get(), 0, GL30.GL_RGBA, GL30.GL_UNSIGNED_BYTE,
imageData)
       STBImage.stbi_image_free(imageData)
   }
   return cubeMaptexID
}
```





Background Objekte



```
val resSaturn: OBJLoader.OBJResult =
OBJLoader.loadOBJ( "project/assets/models/saturn.obj" )
val objMeshSaturn: OBJLoader.OBJMesh = resSaturn. objects[0].meshes[0]

val saturnEmit = Texture2D( "project/assets/textures/2k_saturn.jpg" , true)
val saturnDiff = Texture2D( "project/assets/textures/2k_saturn.jpg" , true)
val saturnSpec = Texture2D( "project/assets/textures/2k_saturn.jpg" , true)

val saturnMaterial = Material(saturnDiff , saturnEmit, saturnSpec, 10.0f,
Vector2f(1.0f))

saturnEmit.setTexParams( GL_REPEAT, GL_REPEAT, GL11.GL_LINEAR_MIPMAP_LINEAR,
GL_LINEAR)
saturnDiff.setTexParams( GL_REPEAT, GL_REPEAT, GL11.GL_LINEAR_MIPMAP_LINEAR,
GL_LINEAR)
saturnSpec.setTexParams( GL_REPEAT, GL_REPEAT, GL11.GL_LINEAR_MIPMAP_LINEAR,
GL_LINEAR)

val saturnMesh = Mesh(objMeshSaturn. vertexData , objMeshSaturn.indexData ,
objVertexAttributes , saturnMaterial)

saturnRend = Renderable()
saturnRend.meshList.add(saturnMesh)
saturnRend.scaleLocal(Vector3f( 2.0f))
saturnRend.translateGlobal(Vector3f( 0.0f, 15.0f, 10.0f))
saturnRend.rotateLocal( 0.0f, 0.0f, 2.0f)
```

```
saturnRend.rotateAroundPoint(dt / 20, 0.0f, 0.0f, planet.getWorldPosition())
```

Collectables

```
class Star(light : PointLight , collectableObject: Renderable , material:
Material){
(...)

fun render(shader : ShaderProgram , name : String) {
   if (!collected) {
       shader.use()
       shader.setUniform( "farbe", Vector3f(1.0f))
       pointLight.bind(shader , name)
       collectableObject .render(shader)
   }
}

fun distance(other : Renderable) : Float {
   val pos1 = other.getWorldPosition()
   val pos2 = this.collectableObject .getWorldPosition()

   val distance = sqrt(
               (pos1.x() - pos2.x()).toDouble(). pow(2.0) +
               (pos1.y() - pos2.y()).toDouble(). pow(2.0) +
               (pos1.z() - pos2.z()).toDouble(). pow(2.0))

   return distance.toFloat()
}
}
```

```
for (star in collectables) {
   if (star.distance(player!!) < 0.2f) {
       if (star.collect()) {
           score++
           println("Collected
$score/$collectableAmount ")
       }
   }
   star.rotate(dt)
}
```


Shader

Toon Shader

```
vec3 colorFactor;
float lightIntensity = max( 0.0f, dot(vertexData.toPointLight ,
n));


if (lightIntensity > 9.5) {
   colorFactor = vec3(1.0);
} else if (lightIntensity > 0.8) {
   colorFactor = vec3(0.85);
} else if (lightIntensity > 0.6) {
   colorFactor = vec3(0.7);
} else if (lightIntensity > 0.5) {
   colorFactor = vec3(0.6);
} else if (lightIntensity > 0.4) {
   colorFactor = vec3(0.45);
} else {
   colorFactor = vec3(0.3);
}

result += shade(n , lp, v, diffCol, specularCol , shininess) *
pointLightIntensity(PointLightCol , lpLength);
result += shade(n , sp, v, diffCol, specularCol , shininess) *
spotLightIntensity(SpotLightCol , spLength, sp, SpotLightDir);
color = vec4(result * colorFactor , 1.0);
```

Negative Shader

```
color = vec4(1.0 - result.x, 1.0 -
result.y, 1.0 - result.z, 1.0);
```

Quellen

- https://www.youtube.com/playlist?list=PLFt_AvWsXl0fEx02iXR8uhDsVGhmM9Pse
- https://www.solarsystemscope.com/textures/
- https://www.youtube.com/watch?v=8sVvxeKI9Pk&t=282s
- https://learnopengl.com/Advanced-OpenGL/Cubemaps
- https://learnopengl.com/Lighting/Basic-Lighting
- https://www.lighthouse3d.com/tutorials/glsl-12-tutorial/toon-shading-version-i/