8. Übungsblatt - C++

Henrik Gerdes, Manuel Eversmeyer
D10

17. Dezember 2018

```
cmake_minimum_required(VERSION 3.0.2)


# The name of out project

project(CPP18_TASK01)
project(CPP18_TASK02)
project(CPP18_TASK03)


set( CMAKE_CXX_FLAGS           "-O3 -Wall " )
set( CMAKE_CXX_FLAGS_RELEASE "-O3 -msse3 -Wno-deprecated-declarations -Wno-
    unused -Wcpp" )
set( CMAKE_CXX_FLAGS_DEBUG    "-g -Wall" )


set(TASK1_SOURCES
     Aufgabe1_1.cpp
     ../math/Randomizer.cpp
)

set(TASK2_SOURCES
    Aufgabe1_2.cpp
    ../math/Randomizer.cpp
)

set(TASK3_SOURCES
     Aufgabe1_3.cpp
)

# The executable fpr our project

add_executable(IntVector ${TASK1_SOURCES})
add_executable(StringVector ${TASK2_SOURCES})
add_executable(Sum ${TASK3_SOURCES})
```

Aufgabe1/CMakeLists.txt

```
#include <vector>
#include <stdlib.h>
#include <algorithm>
#include <iterator>
#include <iostream>
#include "../math/Randomizer.hpp"
```

```
7
8
9  int somenumber()
10 {
11     return (int)asteroids::Randomizer::instance()->getRandomNumber(0,100);
12 }
13
14 int main(int argc, char const *argv[])
15 {
16     //New Int-Vector
17     std::vector<int> numVec(100);
18
19     //Call a Function-Obj to fill Vector
20     std::generate(numVec.begin(), numVec.end(),somenumber);
21
22     //OutPutInterator
23     std::ostream_iterator<int> out_it (std::cout,", ");
24
25     //Use Outstream to stdcout
26     std::copy ( numVec.begin(), numVec.end(), out_it );
27     std::cout << std::endl;
28
29     asteroids::Randomizer::instance()->getRandomNumber(0,100);
30     //(*getRandomNumber)(0,100)
31
32     return 0;
33 }
```

Aufgabe1/Aufgabe1_1.cpp

```
1  #include <vector>
2  #include <stdlib.h>
3  #include <algorithm>
4  #include <iterator>
5  #include <iostream>
6  #include <string>
7  #include "../math/Randomizer.hpp"
8
9  //Save Randomizer instance
10 asteroids::Randomizer* raninst = asteroids::Randomizer::instance();
11
12 //Genarate a string by random numbers matching characters in ASCII
13 std::string someString()
14 {
15     int size = (int)raninst->getRandomNumber(5,10);
16     char a[size + 1];
17
18     for(int i = 0; i < size; i++)
19     {
20         a[i] = (char)(int)raninst->getRandomNumber(65,90);
21     }
22     //Terminate c string
23     a[size] = '\0';
24
25     std::string str(a);
26     return str;
27 }
28
```

```cpp
int main(int argc, char const *argv[])
{
    ///New StringVector
    std::vector<std::string> strVec(100);

    ///Fill String
    std::generate(strVec.begin(), strVec.end(), someString);

    ///Sort String
    std::sort( strVec.begin(), strVec.end());

    //Print to std-out using a outstream interator
    std::ostream_iterator<std::string> out_it (std::cout,"\n");
    std::copy ( strVec.begin(), strVec.end(), out_it );
    std::cout << std::endl;

    return 0;
}
```

Aufgabe1/Aufgabe1_2.cpp

```cpp
#include <iostream>
#include <vector>
#include <functional>
#include <algorithm>

struct adder : public std::unary_function<int, void>
{
    int sum;
    adder() : sum(0) { }
    void operator()(int x)
    {
        sum += x;
    }
};

int main(int argc, char const *argv[])
{
    std::vector<int> numVec(4, 100);

    adder result = std::for_each(numVec.begin(), numVec.end(), adder());
    std::cout << "Sum is " << result.sum << std::endl;
    std::cout << "Sum should be 400" << std::endl;
}
```

Aufgabe1/Aufgabe1_3.cpp

```cpp
#ifndef TEXTUREFACTORY_HPP_
#define TEXTUREFACTORY_HPP_

#include <map>
#include <string>
#include <dirent.h>
#include <stdexcept>
#include "../rendering/Texture.hpp"
#include "../util/Util.hpp"

namespace asteroids
```

```
12 {
13 class TextureFactory
14 {
15     public:
16
17     enum FileType
18     {
19         JPG,
20         TGA,
21         TIFF,
22         PPM
23     };
24
25         /**
26          * @brief   Static function to get the one instance of
27          *          TextureFacrory
28          *
29          * @return  Reference to TextureFactory
30          */
31         static TextureFactory& getinstance();
32
33         /**
34          * @brief Set the Base Path object
35          *
36          * @param base Path to textures
37          */
38         void setBasePath(const std::string& base);
39
40         /**
41          * @brief Get the Texture object
42          *
43          * @param filename
44          * @return Texture*
45          */
46         Texture* getTexture(std::string& filename);
47
48     private:
49
50         /**
51          * @brief Construct a new Texture Factory object
52          */
53         TextureFactory();
54
55         /**
56          * @brief Destroy the Texture Factory object
57          */
58         virtual ~TextureFactory();
59
60         ///Path to Basefolder
61         std::string m_basepath = "";
62
63         ///Map for Textures
64         std::map<std::string,Texture*> texturemap;
65
66         ///Pointer to the one instance
67         static TextureFactory* p_instance;
68
69 };
```

```
70
71 }
72
73 #endif
```

io/TextureFactory.hpp

```
1  #include "TextureFactory.hpp"
2  #include "BitmapReader.hpp"
3  #include "ReadJPG.hpp"
4  #include "ReadTIFF.hpp"
5  #include "ReadPPM.hpp"
6  #include "ReadTGA.hpp"
7
8  namespace asteroids
9  {
10     TextureFactory::TextureFactory()
11     {
12         //new Instance
13     }
14
15     TextureFactory::~TextureFactory()
16     {
17         // for(std::map<std::string,Texture*>::iterator it=texturemap.begin
               ();it!=texturemap.end(); it++)
18         // {
19         //      if(it->second)
20         //      {
21         //          delete it->second;
22         //      }
23         // }
24     }
25
26     TextureFactory& TextureFactory::getinstance()
27     {
28         static TextureFactory instance;
29         return instance;
30     }
31
32     void TextureFactory::setBasePath(const std::string& base)
33     {
34         DIR* dir;
35         dir = opendir(base.c_str());
36         if(dir == NULL)
37         {
38             throw std::invalid_argument("BasePath does not exist");
39         }
40         else
41         {
42             m_basepath = base;
43         }
44         closedir(dir);
45     }
46
47     Texture* TextureFactory::getTexture(std::string& filename)
48     {
49
50         //Look for texture in map
```

```
51          auto search = texturemap.find(filename);
52          if(search != texturemap.end())
53          {
54              std::cout << "Found texture in Factory" << std::endl;
55              return search->second;
56          }
57
58          BitmapReader* read = 0;
59
60          //Genarate new texture
61          if(GetExtensionFromFileName(filename) == "jpg")
62          {
63              read = new ReadJPG(m_basepath + filename);
64
65          }
66          else if(GetExtensionFromFileName(filename) == "ppm")
67          {
68              read = new ReadPPM(m_basepath +filename);
69
70          }
71          else if(GetExtensionFromFileName(filename) == "tga")
72          {
73              read = new ReadTGA(m_basepath + filename);
74          }
75          else if(GetExtensionFromFileName(filename) == "tiff" ||
76              GetExtensionFromFileName(filename) == "tif")
77          {
78              read = new ReadTIFF(m_basepath + filename);
79          }
80          else
81          {
82              cout << "TextureFactory: Unable to read file " << filename << "
                    ." << endl;
83          }
84
85          //Get Instance and put in map
86          Texture* texture = new Texture(read->getPixels(),read->getWidth(),
                read->getHeight());
87          texturemap[filename] = texture;
88          return texture;
89
90      }
91  } //asteroids
```

io/TextureFactory.cpp

```
1  #ifndef TRIANGLEMESHFACTORY_HPP_
2  #define TRIANGLEMESHFACTORY_HPP_
3
4  #include <string>
5  #include <dirent.h>
6  #include <stdexcept>
7  #include "../rendering/TriangleMesh.hpp"
8  #include "../util/Util.hpp"
9  #include "ReadPLY.hpp"
10 #include "Read3DS.hpp"
11 #include "MeshReader.hpp"
12
```

```
13 namespace asteroids
14 {
15 class TriangleMeshFactory
16 {
17     public:
18         /**
19          * @brief   Static function to get the one instance of
20          *          TriangleMeshFactory
21          *
22          * @return  Reference to TriangleMeshFactory&
23          */
24         static TriangleMeshFactory& getinstance();
25
26         /**
27          * @brief Set the Base Path object
28          *
29          * @param base Path to textures
30          */
31         void setBasePath(const std::string& base);
32
33         /**
34          * @brief Get the Mesh object
35          *
36          * @param filename Path ot wanted mesh
37          * @return TriangleMeshFactory*
38          */
39         TriangleMesh* getMesh(const std::string& filename) const;
40
41     private:
42         ///Saves the BasePath
43         std::string m_basepath = "";
44
45         /**
46          * @brief Construct a new Triangle Mesh Factory object
47          */
48         TriangleMeshFactory();
49
50         /**
51          * @brief Destroy the Triangle Mesh Factory object
52          */
53         ~TriangleMeshFactory();
54
55 };
56
57 }
58
59 #endif
```

io/TriangleMeshFactory.hpp

```
1 #include "TriangleMeshFactory.hpp"
2
3 namespace asteroids
4 {
5
6 TriangleMeshFactory::TriangleMeshFactory()
7 {
8     //New TriangleMeshFactory
```

```
}

TriangleMeshFactory::~TriangleMeshFactory()
{
    //Do nothing
}

TriangleMeshFactory& TriangleMeshFactory::getinstance()
{
    static TriangleMeshFactory instance;
    return instance;
}

void TriangleMeshFactory::setBasePath(const std::string& base)
{
    DIR* dir;
    dir = opendir(base.c_str());

    if(dir == NULL)
    {
        throw std::invalid_argument("BasePath does not exist");
    }
    else
    {
        m_basepath = base;
    }
    closedir(dir);
}

TriangleMesh* TriangleMeshFactory::getMesh(const std::string& filename)
    const
{
    MeshReader* mreader;

    if(GetExtensionFromFileName(filename) == "ply")
    {
        mreader = new ReadPLY(m_basepath + filename);
        return mreader->getMesh();
    }
    else if(GetExtensionFromFileName(filename) == "3ds")
    {
        mreader = new Read3DS(m_basepath + filename);
        return mreader->getMesh();

    }

    return NULL;
}

}
```

io/TriangleMeshFactory.cpp

```
/*
 * Read3DS.cpp
 *
 *  @date 18.11.2018
 *  @author Thomas Wiemann
```

```
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#include "Read3DS.hpp"

#include <SDL2/SDL.h>
#include "../ext/load3ds/C3DSMaterialReader.h"
#include "C3DSMeshReader.h"
#include "C3DSMeshNormalCalculator.h"

#include <iostream>
#include <map>
using std::cout;
using std::endl;
using std::map;

namespace asteroids
{

void Read3DS::getMesh(TexturedMesh& texMesh)
{
    SDL_RWops* rw = SDL_RWFromFile(m_filename.c_str(), "rb");

    unsigned found = m_filename.find_last_of("/\\");
    string basePath = m_filename.substr(0, found+1);

    TextureFactory::getinstance().setBasePath(basePath);
    // TODO: Set base path in factory. DONE

    // Parse materials
    map<string, int> matMap;

    C3DSMaterialReader matReader(rw);
    matReader.parse();

    for(size_t i = 0; i < matReader.materials().size(); i++)
    {
        C3DSMaterialReader::Material m = matReader.materials()[i];

        // Convert name into ID
        matMap[m.name] = (int)i;

        // Convert to Material struct
        Material* mat = new Material;
        mat->m_ambient.r = m.colorAmbient[0];
        mat->m_ambient.g = m.colorAmbient[1];
        mat->m_ambient.b = m.colorAmbient[2];

        mat->m_diffuse.r = m.colorDiffuse[0];
        mat->m_diffuse.g = m.colorDiffuse[1];
        mat->m_diffuse.b = m.colorDiffuse[2];

        mat->m_specular.r = m.colorSpecular[0];
        mat->m_specular.g = m.colorSpecular[1];
```

```
63          mat->m_specular.b = m.colorSpecular[2];
64          mat->m_shininess = m.shininess;
65
66          // Load texture from file
67          if(m.texMaps.size() > 0)
68          {
69              // TODO: Load texture from factory and save it in DONE
70              // mat->m_texture = ...
71              mat->m_texture = TextureFactory::getinstance().getTexture(m.
                    texMaps[0].filename);
72          }
73          else
74          {
75              mat->m_texture = 0;
76          }
77
78          texMesh.addMaterial(mat);
79      }
80
81      // Reset file pointer
82      SDL_RWseek(rw, 0, SEEK_SET);
83
84      // Read geometry
85      C3DSMeshReader        meshReader(rw);
86      meshReader.parse();
87
88      // Create a mesh with normals
89      C3DSMeshNormalCalculator normalMesh(meshReader);
90      normalMesh.calculate();
91
92      // Save geometry information and merge with material info
93      for(size_t i = 0; i < normalMesh.meshes().size(); i++)
94      {
95
96          C3DSMeshNormalCalculator::Mesh &mesh = normalMesh.meshes()[i];
97          m_numVertices += mesh.vertices.size();
98          m_numFaces += mesh.faces.size();
99      }
100
101     // Alloc memory for buffers
102     m_vertexBuffer  = new float [3 * m_numVertices];
103     m_normalBuffer  = new float [3 * m_numVertices];
104     m_indexBuffer   = new int   [3 * m_numFaces];
105     float* textureBuffer = new float[2 * m_numVertices];
106
107     // Fill buffers
108     size_t vertexCount  = 0;
109     size_t startFace    = 0;
110
111     for(size_t i = 0; i < normalMesh.meshes().size(); i++)
112     {
113         C3DSMeshNormalCalculator::Mesh &n_mesh = normalMesh.meshes()[i];
114         C3DSMeshReader::Mesh &mesh = meshReader.meshes()[i];
115
116         for(size_t j = 0; j < mesh.vertices.size(); j++)
117         {
118             size_t vBufferIndex = 3 * vertexCount;
119             size_t cBufferIndex = 2 * vertexCount;
```

```
120
121             for(int a = 0; a < 3; a++)
122             {
123                 m_vertexBuffer[vBufferIndex + a] = mesh.vertices[j][a];
124                 m_normalBuffer[vBufferIndex + a] = n_mesh.normals[j][a];
125             }
126
127             if(j < mesh.mapCoords.size())
128             {
129                 textureBuffer[cBufferIndex]     = mesh.mapCoords[j].u;
130                 textureBuffer[cBufferIndex + 1]  = mesh.mapCoords[j].v;
131              }
132             else
133             {
134                 textureBuffer[cBufferIndex]  = 0.0f;
135                 textureBuffer[cBufferIndex + 1]  = 0.0f;
136             }
137
138             vertexCount++;
139         }
140
141         for(size_t j = 0; j < mesh.faceMaterials.size(); j++)
142         {
143             C3DSMeshReader::FaceMaterial fm = mesh.faceMaterials[j];
144             MaterialFaceList* matList = new MaterialFaceList;
145             matList->m_matIndex = matMap[fm.name];
146
147             for(size_t a = 0; a < fm.faces.size(); a++)
148             {
149                 C3DSMeshReader::Face cFace = mesh.faces[fm.faces[a]];
150                 matList->m_faces.push_back(cFace.indices[0] + startFace);
151                 matList->m_faces.push_back(cFace.indices[1] + startFace);
152                 matList->m_faces.push_back(cFace.indices[2] + startFace);
153             }
154
155             texMesh.addMaterialFaceList(matList);
156         }
157         startFace += mesh.vertices.size();
158
159     }
160
161     // Save buffer in mesh
162     texMesh.setVertexBuffer(m_vertexBuffer, m_numVertices);
163     texMesh.setIndexBuffer(m_indexBuffer, m_numFaces);
164     texMesh.setNormalBuffer(m_normalBuffer);
165     texMesh.setTextureBuffer(textureBuffer);
166
167     SDL_RWclose(rw);
168 }
169
170 Read3DS::Read3DS(string filename)
171 {
172     m_filename = filename;
173 }
174
175 void Read3DS::readSimpleMesh()
176 {
177     SDL_RWops *rw = SDL_RWFromFile(m_filename.c_str(), "rb");
```

```
178
179     //C3DSMaterialReader  matReader(rw);
180
181     // Parse mesh
182     C3DSMeshReader       meshReader(rw);
183     meshReader.parse();
184
185     // Create a mesh with normals
186     C3DSMeshNormalCalculator normalMesh(meshReader);
187     normalMesh.calculate();
188
189     // Calc vertices and faces in all mesh groups
190     for(size_t i = 0; i < normalMesh.meshes().size(); i++)
191     {
192
193         C3DSMeshNormalCalculator::Mesh &mesh = normalMesh.meshes()[i];
194
195         m_numVertices += mesh.vertices.size();
196         m_numFaces += mesh.faces.size();
197     }
198
199     // Alloc memory for buffers
200     m_vertexBuffer  = new float [3 * m_numVertices];
201     m_normalBuffer  = new float [3 * m_numVertices];
202     m_indexBuffer   = new int   [3 * m_numFaces];
203
204     // Fill buffers
205     size_t vertexCount  = 0;
206     size_t faceCount    = 0;
207
208     size_t startFace    = 0;
209
210     for(size_t i = 0; i < normalMesh.meshes().size(); i++)
211     {
212         C3DSMeshNormalCalculator::Mesh &mesh = normalMesh.meshes()[i];
213
214         for(size_t j = 0; j < mesh.vertices.size(); j++)
215         {
216             size_t vBufferIndex = 3 * vertexCount;
217             for(int a = 0; a < 3; a++)
218             {
219                 m_vertexBuffer[vBufferIndex + a] = mesh.vertices[j][a];
220                 m_normalBuffer[vBufferIndex + a] = mesh.normals[j][a];
221             }
222             vertexCount++;
223         }
224
225
226         for(size_t j = 0; j < mesh.faces.size(); j++)
227         {
228             size_t fBufferIndex = 3 * faceCount;
229             for(int a = 0; a < 3; a++)
230             {
231                 m_indexBuffer[fBufferIndex + a] = mesh.faces[j].indices[a]
                        + startFace;
232
233             }
234             faceCount++;
```

```
235             }
236             startFace += mesh.vertices.size();
237         }
238 
239 }
240 
241 TriangleMesh* Read3DS::getMesh()
242 {
243 
244     TexturedMesh* mesh = new TexturedMesh;
245     getMesh(*mesh);
246 
247     return mesh;
248 }
249 
250 
251 Read3DS::~Read3DS()
252 {
253     // Auto-generated destructor stub
254 }
255 
256 }
```

io/Read3DS.cpp

```
1  /*
2   *  Skybox.cpp
3   *
4   *  @date 18.11.2018
5   *  @author Thomas Wiemann
6   *
7   *  Copyright (c) 2018 Thomas Wiemann.
8   *  Restricted usage. Licensed for participants of the course "The C++
        Programming Language" only.
9   *  No unauthorized distribution.
10  */
11 #include "Skybox.hpp"
12 namespace asteroids
13 {
14 
15 
16 Skybox::Skybox(int width, string files[6])
17     : m_width(width)
18 {
19     TextureFactory& textfact = TextureFactory::getinstance();
20     // Create textures
21     m_textures = new Texture*[6];
22     for(int i = 0; i < 6; i++)
23     {
24         m_textures[i] = textfact.getTexture(files[i]);
25     }
26 }
27 
28 Skybox::~Skybox()
29 {
30     if(m_textures)
31     {
32         for(int i = 0; i < 6; i++)
```

```
33          {
34              if(m_textures[i])
35              {
36                  delete m_textures[i];
37              }
38          }
39          delete[] m_textures;
40          m_textures = 0;
41      }
42  }
43
44
45  void Skybox::render(Camera& cam)
46  {
47      // Enable/Disable features
48
49      glPushMatrix();
50      glPushAttrib(GL_ENABLE_BIT);
51      glEnable(GL_TEXTURE_2D);
52      glDisable(GL_DEPTH_TEST);
53      glDisable(GL_LIGHTING);
54      glDisable(GL_BLEND);
55
56      cam.applyRotationOnly();
57
58      // Set color
59      glColor3f(1.0f, 1.0f, 1.0f);
60
61      float pos = 0.5f * m_width;
62      // Render right quad
63      if(m_textures[3])
64      {
65          m_textures[3]->bind();
66          glBegin(GL_QUADS);
67          glTexCoord2f(1, 1); glVertex3f(  pos, -pos,  pos );
68          glTexCoord2f(1, 0); glVertex3f(  pos, -pos, -pos );
69          glTexCoord2f(0, 0); glVertex3f(  pos,  pos, -pos );
70          glTexCoord2f(0, 1); glVertex3f(  pos,  pos,  pos );
71          glEnd();
72      }
73
74      // Render rear quad
75      if(m_textures[0])
76      {
77          m_textures[0]->bind();
78          glBegin(GL_QUADS);
79          glTexCoord2f(0, 1); glVertex3f( -pos, -pos, -pos );
80          glTexCoord2f(1, 1); glVertex3f(  pos, -pos, -pos );
81          glTexCoord2f(1, 0); glVertex3f(  pos,  pos, -pos );
82          glTexCoord2f(0, 0); glVertex3f( -pos,  pos, -pos );
83          glEnd();
84      }
85
86      // Render the front quad
87      if(m_textures[2])
88      {
89          m_textures[2]->bind();
90          glBegin(GL_QUADS);
```

```
91          glTexCoord2f(0, 1); glVertex3f( -pos, -pos,  pos );
92          glTexCoord2f(1, 1); glVertex3f(  pos, -pos,  pos );
93          glTexCoord2f(1, 0); glVertex3f(  pos,  pos,  pos );
94          glTexCoord2f(0, 0); glVertex3f( -pos,  pos,  pos );
95          glEnd();
96      }
97
98      // Render the left quad
99      if(m_textures[1])
100     {
101         m_textures[1]->bind();
102         glBegin(GL_QUADS);
103         glTexCoord2f(1, 1); glVertex3f( -pos, -pos, -pos );
104         glTexCoord2f(0, 1); glVertex3f( -pos, -pos,  pos );
105         glTexCoord2f(0, 0); glVertex3f( -pos,  pos,  pos );
106         glTexCoord2f(1, 0); glVertex3f( -pos,  pos, -pos );
107         glEnd();
108     }
109
110     // Render the top quad
111     if(m_textures[4])
112     {
113         m_textures[4]->bind();
114         glBegin(GL_QUADS);
115         glTexCoord2f(0, 1); glVertex3f( -pos,  pos, -pos );
116         glTexCoord2f(0, 0); glVertex3f( -pos,  pos,  pos );
117         glTexCoord2f(1, 0); glVertex3f(  pos,  pos,  pos );
118         glTexCoord2f(1, 1); glVertex3f(  pos,  pos, -pos );
119         glEnd();
120     }
121
122     // Render the bottom quad
123     if(m_textures[5])
124     {
125         m_textures[5]->bind();
126         glBegin(GL_QUADS);
127         glTexCoord2f(0, 0); glVertex3f( -pos, -pos, -pos );
128         glTexCoord2f(0, 1); glVertex3f( -pos, -pos,  pos );
129         glTexCoord2f(1, 1); glVertex3f(  pos, -pos,  pos );
130         glTexCoord2f(1, 0); glVertex3f(  pos, -pos, -pos );
131         glEnd();
132     }
133
134    glPopAttrib();
135    glPopMatrix();
136 }
137
138 }
```

rendering/Skybox.cpp

```
1  /*
2   *  SpaceCraft.cpp
3   *
4   *  Created on: Nov. 04 2018
5   *      Author: Thomas Wiemann
6   *
7   *  Copyright (c) 2018 Thomas Wiemann.
```

```
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#include "SpaceCraft.hpp"

namespace asteroids
{

SpaceCraft::SpaceCraft(const std::string &filename, const Vector3f&
    position, float movespeed, float rotatespeed)
    : m_movespeed(movespeed), m_rotatespeed(rotatespeed)
{
    // TODO: Get mesh from TriangleMesh factory and save it DONE
    // in m_mesh. Set correct intial position
    m_mesh = TriangleMeshFactory::getinstance().getMesh(filename);
    m_mesh->setPosition(position);
}

void SpaceCraft::handleKeyInput(const Uint8* keyStates)
{
    if (keyStates[SDL_SCANCODE_UP])
    {
        m_mesh->rotate(TriangleMesh::YAW, m_rotatespeed);
    }
    if (keyStates[SDL_SCANCODE_DOWN])
    {
        m_mesh->rotate(TriangleMesh::YAW, -m_rotatespeed);
    }
    if (keyStates[SDL_SCANCODE_LEFT])
    {
        m_mesh->rotate(TriangleMesh::ROLL, m_rotatespeed);
    }
    if (keyStates[SDL_SCANCODE_RIGHT])
    {
        m_mesh->rotate(TriangleMesh::ROLL, -m_rotatespeed);
    }
    if (keyStates[SDL_SCANCODE_W])
    {
        m_mesh->move(TriangleMesh::ACCEL, -m_movespeed);
    }
    if (keyStates[SDL_SCANCODE_S])
    {
        m_mesh->move(TriangleMesh::ACCEL, m_movespeed);
    }
    if (keyStates[SDL_SCANCODE_A])
    {
        m_mesh->move(TriangleMesh::STRAFE, -m_movespeed);
    }
    if (keyStates[SDL_SCANCODE_D])
    {
        m_mesh->move(TriangleMesh::STRAFE, m_movespeed);
    }
}

void SpaceCraft::render()
{
```

```
64      if (m_mesh)
65      {
66          m_mesh->render();
67      }
68  }
69
70  bool SpaceCraft::hasMesh() const
71  {
72      return m_mesh != nullptr;
73  }
74
75  SpaceCraft::~SpaceCraft()
76  {
77      if (m_mesh)
78      {
79          delete m_mesh;
80          m_mesh = nullptr;
81      }
82  }
83
84  } // namespace asteroids
```

rendering/SpaceCraft.cpp

```
1   /*
2    *   AsteoridField.cpp
3    *
4    *   @date 18.11.2018
5    *   @author Thomas Wiemann
6    *
7    *   Copyright (c) 2018 Thomas Wiemann.
8    *   Restricted usage. Licensed for participants of the course "The C++
         Programming Language" only.
9    *  No unauthorized distribution.
10   */
11
12  #include "AsteroidField.hpp"
13  #include "Asteroid.hpp"
14  #include "../math/Randomizer.hpp"
15  #include "../math/Vector.hpp"
16  namespace asteroids
17  {
18
19  AsteroidField::AsteroidField(int quantity, const std::string& filename,
        float rangemax, float sizemin, float sizemax)
20  {
21      //Get instances
22      Randomizer* randominst = Randomizer::instance();
23      TriangleMeshFactory& triangleinst = TriangleMeshFactory::getinstance();
24
25      //Generate asteroids
26      for(int i = 0; i < quantity; i++)
27      {
28          std::cout << "Genarate Asteroid " << i << filename << " at " << std
                ::endl;
29          randominst->getRandomVertex(2).printVector();
30
31          TexturedMesh* textmesh = (TexturedMesh*)(triangleinst.getMesh(
```

```
            filename));
32          m_asteroids.push_back(new Asteroid(textmesh, randominst->
                getRandomVertex(500),
33          randominst->getRandomNumber(0, 0.5)));
34
35          /// TODO: Get mesh from class TriangleMeshFactory and add new DONE
                Except its just one asteroid
36          /// Asteroid to internal list DONE
37      }
38  }
39
40  AsteroidField::~AsteroidField()
41  {
42      for(Asteroid* a : m_asteroids)
43      {
44          if(a)
45          {
46              delete a;
47          }
48      }
49      //m_asteroids.for_each(delete Asteroid);
50
51  }
52
53  void AsteroidField::render()
54  {
55      for(auto& t : m_asteroids)
56      {
57          t->render();
58      }
59
60  }
61
62  } // namespace asteroids
```

rendering/AsteroidField.cpp

```
1   /*
2    * List.hpp
3    *
4    *  @date 02.12.2018
5    *  @author Thomas Wiemann
6    *
7    *  Copyright (c) 2018 Thomas Wiemann.
8    *  Restricted usage. Licensed for participants of the course "The C++
         Programming Language" only.
9    *  No unauthorized distribution.
10   */
11
12
13  #ifndef LIST_H
14  #define LIST_H
15
16  namespace asteroids
17  {
18
19  /**
20   * @brief   A simple generic list class
```

```
 */
template<typename T> class List
{
   public:

      /**
       * @brief Constructs an empty list.
       */
      List<T>();
      /**
       * @brief  Destructor. Frees the generated nodes.
       */
      ~List<T>();
      /**
       * @brief Inserts an item into the list, i.e. a new node
       *        constaining @ref item is created.
       * @param item To be inserted
       *
       */
      void push_back(T item);

      /**
       * @brief Iterates over all items in the list and calls
       * the given function @ref do_something(...) for
       * every item stored in the list.
       *
       * @param do_something Function pointer to apply to all elements.
       */
      void for_each(void (*do_something)(T& item));


      class Interator
      {
      public:
            explicit Interator(typename List<T>::Node* num):index(num){}
            friend class List;

            bool operator!=(const Interator& other)
            {
                  return (index != other.index);
            }
            const Interator& operator++()
            {
                  index = index->next;
                  return *this;
            }

            T& operator*() const
            {
                  return index->data;
            }

      private:
            typename List<T>::Node* index;
      };

            /**
       * @brief
```

```
79          */
80         Interator begin()
81         {
82                 return List::Interator(m_root);
83         }
84 
85         /**
86          * @brief
87          */
88         Interator end()
89         {
90                 return List::Interator(nullptr);
91         }
92 
93 private:
94 
95         /**
96          * @brief Struct to represent an inner node of the list.
97          */
98         class Node {
99         public:
100                Node( T _data, Node* _next) : data(_data), next(_next) {};
101                friend class List;
102        private:
103                T data;
104                Node* next;
105        };
106 
107        Node* m_root;
108 };
109 
110 }
111 
112 #include "List.tcc"
113 
114 #endif
115 /* end of include guard: LIST_H */
```

util/List.hpp

```
1  /*
2   * List.tcc
3   *
4   *  @date 02.12.2018
5   *  @author Thomas Wiemann
6   *
7   *  Copyright (c) 2018 Thomas Wiemann.
8   *  Restricted usage. Licensed for participants of the course "The C++
        Programming Language" only.
9   *  No unauthorized distribution.
10  */
11 
12 #include <iostream>
13 
14 namespace asteroids
15 {
16 
17 template<typename T> List<T>::List()
```

```
18 {
19     m_root = nullptr;
20 }
21
22 template<typename T> List<T>::~List()
23 {
24     Node* next = m_root;
25     do
26     {
27         Node* to_delete = next;
28         next = next->next;
29         delete to_delete;
30     }
31     while(next);
32 }
33
34 template<typename T> void List<T>::push_back(T item)
35 {
36     if (m_root == nullptr)
37     {
38         m_root = new Node(item, nullptr);
39     }
40     else
41     {
42         m_root = new Node(item, m_root);
43     }
44 }
45
46 template<typename T> void List<T>::for_each(void (*do_something)(T& item))
47 {
48     Node* tmp = m_root;
49     while (tmp != nullptr)
50     {
51         do_something(tmp->data);
52         tmp = tmp->next;
53     }
54 }
55
56 } // namespace asteroids
```

util/List.tcc