src

* engineTester
  + MainGameLoop.java
    - Note that we must start a shader program before rendering, then we should stop the shader program
* entities
  + Entity.java
    - Think of as instance of TexturedModel, but with translation, rotation, and scale
    - Allows us to have many Entitys using the same TexturedModel, but with different translations, rotations, and scales
  + Camera.java
    - Has move(), which changes its position
    - In Maths class, view matrix is created from Camera object (world is moved in opposite direction of camera)
  + Light.java
* models
  + RawModel.java
    - Represents a 3D model stored in memory as a VAO, has a unique VAO ID
  + TexturedModel.java
    - Contains a RawModel and ModelTexture
* renderEngine
  + DisplayManager.java
    - Handles creating, updating, and destroying the display
  + Loader.java
    - Loads (x,y,z) vertex coordinates, (u,v) texture coordinates, (x,y,z) normals and indices (to use less data when triangles share vertex/texture coordinates into a VAO, and returns a RawModel with the VAO’s ID
    - For indices, which are stored in a VBO of type ***GL\_ELEMENT\_ARRAY\_BUFFER***, we don’t need to use GL20.*glVertexAttribPointer(…)*
    - Don't unbind the index buffer anywhere! Each VAO has one special slot for an index buffer, and unbinding the index buffer will remove it from that slot
    - Loads texture image, called in MainGameLoop when making new ModelTexture object
    - Understand the difference between loading texture coordinates into one of VAO’s attribute lists and loading the actual texture image when making new ModelTexture object and setting the object’s textureID to the texture’s ID
    - Handles cleanup for VAOs, VBOs, and textures
  + Renderer.java
    - Renders an Entity
    - In render(…), *glEnableVertexAttribArray*(…) activates an attribute list
    - render(…) is called once per frame in MainGameLoop. In render(…) , we create a transformation matrix and load it to the vertex shader.
    - Creates projection matrix
    - Constructor takes in StaticShader object, creates projection matrix (only needs to be created once), and loads it to object (shader).
  + OBJLoader.java
    - Takes .obj file, loads its data into a VAO, returns RawModel with VAO ID
    - Constructor takes a loader, since one is needed to load to VAO
    - Assume vertex positions (x,y,z) are “in order”. Texture coordinates (u,v) and normals (x,y,z) are not in order relative to vertex positions. processVertex(…) “orders” texture coordinates and normals
* shaders
  + (abstract) ShaderProgram.java
    - Loads .txt shader files and returns ID for shaders
    - bindAttribute(…) binds attribute list at specified index of currently bound VAO to variable name in vertex shader code
    - getUniformLocation(…) gets location (int) of a uniform variable in shader code
    - Contains definitions for bindAttribute(…) and getUniformLocation(…), but leaves bindAttributes() and getAllUniformLocations() for extending subclasses to define. This is intentional, since programID is private to abstract ShaderProgram class. bindAttributes() simply calls super.bindAttribute(…) for every attribute, and getAllUniformLocations() calls super.getAllUniformLocations() for all uniform variables
    - Loads basic values (floats, vectors, booleans, matrices) to uniform locations in vertex shaders and fragment shaders
    - Handles cleanup for vertex shaders, fragment shaders, and programs
  + StaticShader.java
    - Contains paths for .txt shader files
    - Contains uniform variable locations for transformation, projection, and view matrices, light position and color
    - Uses **super**.loadMatrix(…) to load transformation, projection, and view matrices
  + vertexShader.txt
    - Inputs: vertex (x,y,z) positions, texture (u,v) coordinates, and normal (x,y,z) from VAO attribute lists 0,1, and 2
    - Output: same texture (u,v) coordinates (will get linearly interpolated in fragment shader), normal, vector to light
    - Uniform variables: transformation matrix (translation, rotation, scale), projection matrix (perspective, not orthographic view), view matrix (world moves in opposite direction of camera), light position
  + fragmentShader.txt
    - Input: texture (u,v) coordinates, normal, vector to light
    - Output: color of pixel, uses GLSL method texture(…) and dot product
    - Uniform variables: texture sampler, light color
* textures
  + ModelTexture.java
* toolbox
  + Maths.java
    - Creates a transformation matrix given a translation, rotation, and scale
    - Creates a view matrix given a Camera

res

* image.png for texture

lib

* jars
  + lwjgl.lib
  + lwjgl\_util.lib
  + slick-util.lib
* natives
  + LWJGL natives…