Name: Ralenski Doucet
Contents: Cameras and Projection

- **Camera class: UML:**

Public:
Camera()
~Camera()
Void update(float Deltatime)
Glm::vec4 set Perspective(float field of view, float aspect ratio, float near, float far)
Void set Look At(glm::vec3 from, glm::vec3 to, glm::vec3 up)
Void set Position(glm::vec3 position)
Glm::mat4 get World Transform()
Glm:: mat4 get View()
Glm::mat4 get Projection()
Glm::mat4 get Projection View()
Void change Projection(int is Active
Private:
Glm::mat4 world Transform
Glm:: mat4 view Transform
Glm::mat4 projection Transform
Glm::mat4 projection View Transform
Void update Projection View Transform()

- **Fly Camera class UML:**

Public:
Fly Camera()
~Fly Camera()
Void update(float delta Time)
Void set Speed(float value)
Private:
Float speed
Glm::vec3 up

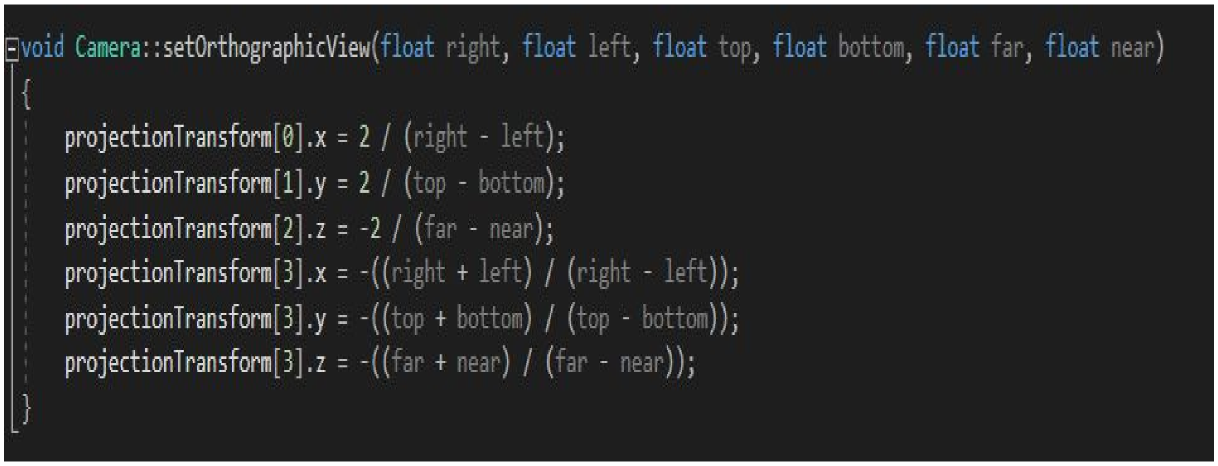
- **Orthographic View: see Image I:**

To be able to setup an orthographic view you need a projection transform and a function that five arguments all of the arguments will be floats that are equal to certain indices of the projection Transform. The argument names for the function are as follows Right, left, top, bottom, far, and near. The **1st index** of the projection transform projection ***Transform[0].x*** is equal to 2 divided by ***right - left.*** The **2nd index** of the projection transform projection ***Transform[1].y*** is equal to 2 divided by ***top - bottom.***The **3rd index** of the projection transform projection ***Transform[2].z*** is equal to -2 divided by ***right - left.*** The **4th index** of the projection transform
projection *Transform[3].x* is equal to **-((right + left)) / ((right - left)).**

Name: Ralenski Doucet
Contents: Cameras and Projection

The **4th index** of the projection transform projection ***Transform[3].y*** is equal to **-((top + bottom)) / ((top - bottom)).** The **4th index** of the projection transform projection ***Transform[3].z*** is equal to **-((far + near)) / ((far - near))**

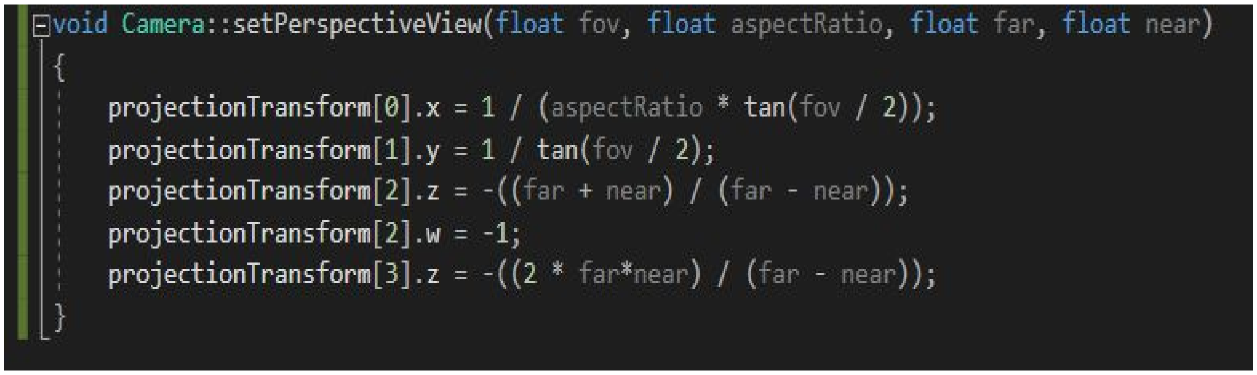
image I:



```
void Camera::setOrthographicView(float right, float left, float top, float bottom, float far, float near)
{
    projectionTransform[0].x = 2 / (right - left);
    projectionTransform[1].y = 2 / (top - bottom);
    projectionTransform[2].z = -2 / (far - near);
    projectionTransform[3].x = -((right + left) / (right - left));
    projectionTransform[3].y = -((top + bottom) / (top - bottom));
    projectionTransform[3].z = -((far + near) / (far - near));
}
```

- **Perspective View: See image II:**
  To be able to setup Perspective view you need a projection transform and you need a function that takes in four arguments. The arguments are as followed Field of view(**FOV**), aspect Ratio, far, and near. The defining of the function goes as followed :
  1st: Projection Transform[0].x equals 1 / (aspect ratio * tan (fov /2)).
  2nd: Projection Transform[1].y equals 1 / tan (fov / 2).
  3rd: Projection Transform[2].z equals -((far + near) / (far - near))
  4th: Projection Transform[2].w equals -1.
  5th: Projection Transform [3].z = -((2 * far * near) / (far - near))

Image II:



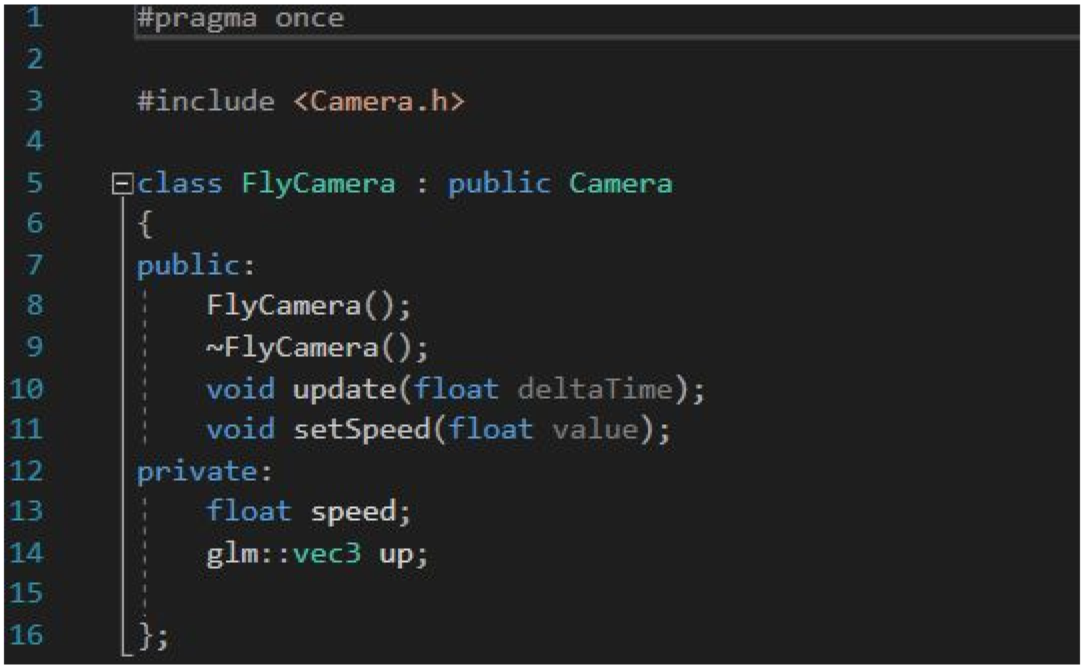
```
void Camera::setPerspectiveView(float fov, float aspectRatio, float far, float near)
{
    projectionTransform[0].x = 1 / (aspectRatio * tan(fov / 2));
    projectionTransform[1].y = 1 / tan(fov / 2);
    projectionTransform[2].z = -((far + near) / (far - near));
    projectionTransform[2].w = -1;
    projectionTransform[3].z = -((2 * far*near) / (far - near));
}
```

Name: Ralenski Doucet
Contents: Cameras and Projection

- **InHeritance: see image III**

My Fly camera class is inheriting from my camera class.

image III:



```cpp
#pragma once

#include <Camera.h>

class FlyCamera : public Camera
{
public:
    FlyCamera();
    ~FlyCamera();
    void update(float deltaTime);
    void setSpeed(float value);
private:
    float speed;
    glm::vec3 up;

};
```