// clang-format off #include "../include/glad/glad.h" #include <GLFW/glfw3.h>

#include

#include <glm/glm.hpp> #include <glm/gtc/matrix_transform.hpp> #include <glm/gtc/type_ptr.hpp>

#include "../include/Shader.h" #include "../include/Game.h" #include "../include/Input.h" #include "../include/TextRenderer.h" #include "../include/Renderer.h"

// clang-format on

void framebuffer_size_callback(GLFWwindow *window, int width, int height) { glViewport(0, 0, width, height); }

void processInput(GLFWwindow *window) { if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS) glfwSetWindowShouldClose(window, true); } // Tracks wether the player has won or lost int gamestate = 0; // 1 means won, 2 means lost

int main() { // Initialize GLFW glfwInit(); glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3); glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3); glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

GLFWwindow *window = glfwCreateWindow(800, 600, "LearnOpenGL", NULL, NULL); if (window == NULL) { std::cout « "Failed to create GLFW window" « std::endl; glfwTerminate(); return -1; } glfwMakeContextCurrent(window);

// Load OpenGL function pointers using GLAD if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress)) { std::cout « "Failed to initialize GLAD" « std::endl; return -1; }

// Enable alpha blending for text transparency glEnable(GL_BLEND); glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

// Load shaders Shader ourShader("assets/shaders/shader.vs", "assets/shaders/shader.fs"); Shader textShader("assets/shaders/text.vs", "assets/shaders/text.fs");

// Initialize TextRenderer with font and font size Renderer renderer(ourShader); TextRenderer textRenderer("assets/fonts/arial.ttf", 48);

// Set up projection matrix once and set uniforms on text shader glm::mat4 projection = glm::ortho(0.0f, 800.0f, 0.0f, 600.0f); textShader.use(); textShader.setInt("text", 0); // Texture unit 0 textShader.setMat4("projection", projection);

// Set the viewport size and register framebuffer callback glViewport(0, 0, 800, 600); glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

// Setup Input and Gamelogic int input_return_code = init_input(window); int game_return_code = init_game(); if (input_return_code != 0) { std::cout «

“ERROR::FAILED_INPUT_SETUP”; } if (game_return_code != 0) { std::cout « “ERROR::FAILED_GAME_SETUP”; }

// Main render loop while (!glfwWindowShouldClose(window)) { // Input processInput(window);

```
int GAME_WON = 1;
int GAME_LOST = 2;

// Here comes the game logic
gamestate = update_game();
if (gamestate == GAME_WON) {
  // has won
  break; // exit game for now
} else if (gamestate == GAME_LOST) {
  // has lost
  break; // exit game for now
}

// Clear screen
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

// Draw your triangle
renderer.Draw();

// Draw the text. get_board_letters calls from the Game.cpp file
textRenderer.RenderText(textShader, get_board_letters(), 300.0f, 300.0f,
                        1.0f, glm::vec3(1.0f, 1.0f, 1.0f));

glfwSwapBuffers(window);
glfwPollEvents();
```

}

// Clean up and exit glfwTerminate(); return 0; }