Grupa: Szymon Krzysztofik, Michał Kuckowicz, Tadeusz Kogut

Sprawozdanie z Projektu - Maze 3D

Link do projektu: https://github.com/Chudybyk333/maze3D

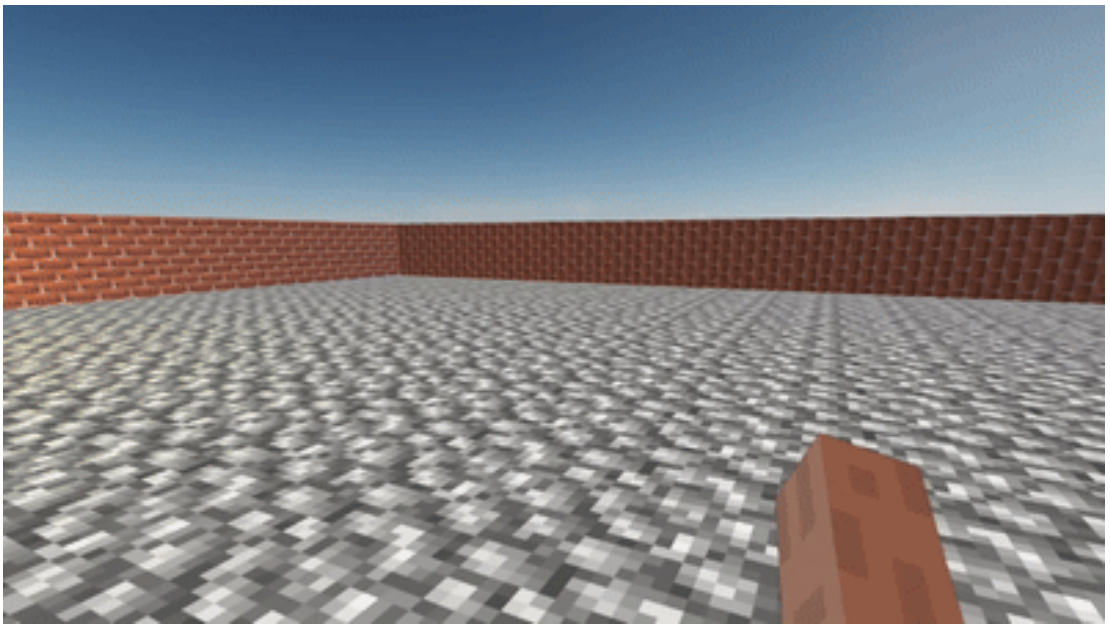
Wstęp

Celem naszego projektu było stworzenie prostej gry - labiryntu, używając technologii OpenGL (3.3), GLFW, GLM i STB w języku C++. W naszym projekcie umieściliśmy wiele obiektów 3D, w tym możliwość generowania ich na nieskończoną ilość możliwości podczas tworzenia labiryntu; do obiektów można przypisać dowolne tekstury; zaimplementowaliśmy także wiele źródeł światła - zarówno point light, jak i directional light; dodaliśmy także filtrowania trilinear i bilinear, z multisamplingiem dla gładkich tekstur i krawędzi.

Funkcjonalności

Sterowanie postacią (kamerą):

- Myszka - Obrót kamery
- W - Ruch do przodu
- S - Ruch do tyłu
- A - Ruch w lewo
- D - Ruch w Prawo
- Spacja - Skok
- Shift - Bieg
- Alt - Latanie



Wczytywanie layoutu labiryntu z pliku maze.txt:

- Ściana

K - Klucz

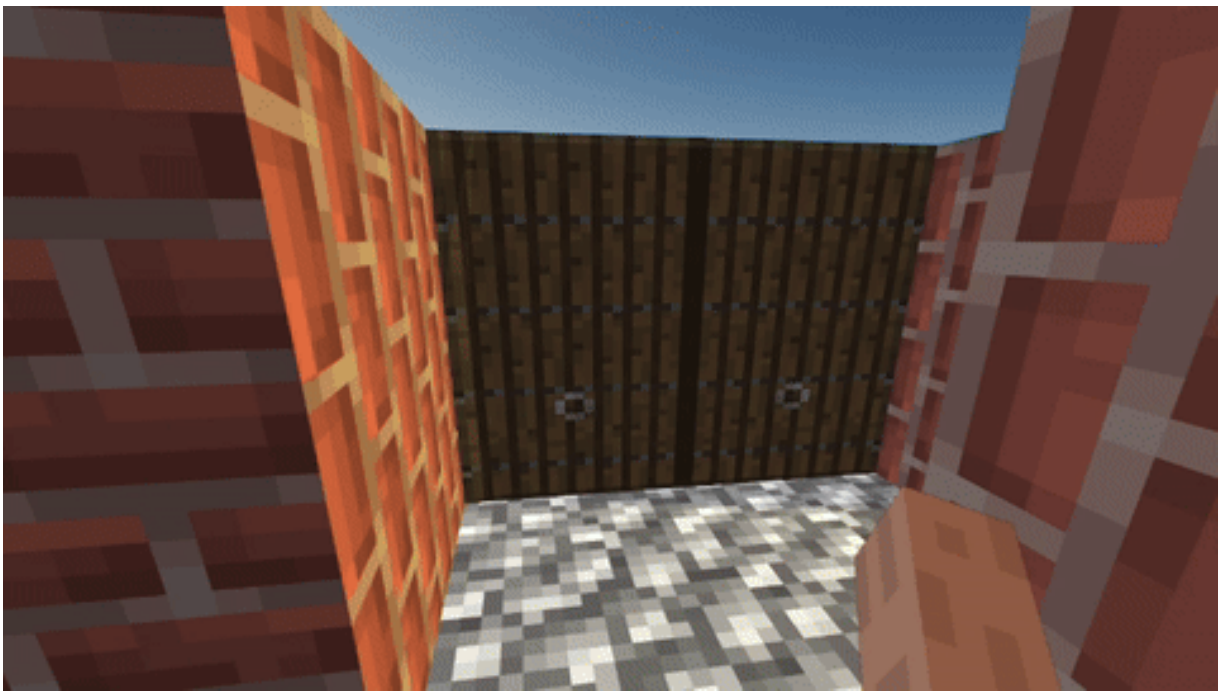
D, E - Drzwi

```
############################
#    #    # #      #    #    #
# # # # # # # ### ### # # #
#K# #    #      #    #    # #
### # ###### ### # ###### #
#    #       #    # #      # #
# ### ### # ### ###### # #
#       # # #      #    # # #
# ### # # ### ### # # # #
#    # #       #    # #    # #
### ######## # # # ### ###
#    #       # # # # #       #
# # # ### #   ## ### ### #
# # #    #    #    #       # #
# ### # ###### ### ### # #
#    # #       #    #    # # #
# ### ###### ### ### # ###
#    #    #       #       #    #
### # # ### ### ### ### #
#    # #    #    # #    #    #
# ### ### ### # # ### # #
#    #       #    # #    # # #
### ###### ### ### ### # #
#       #       #       #    # #
####################DE#####
```

^ przykładowy plik maze.txt

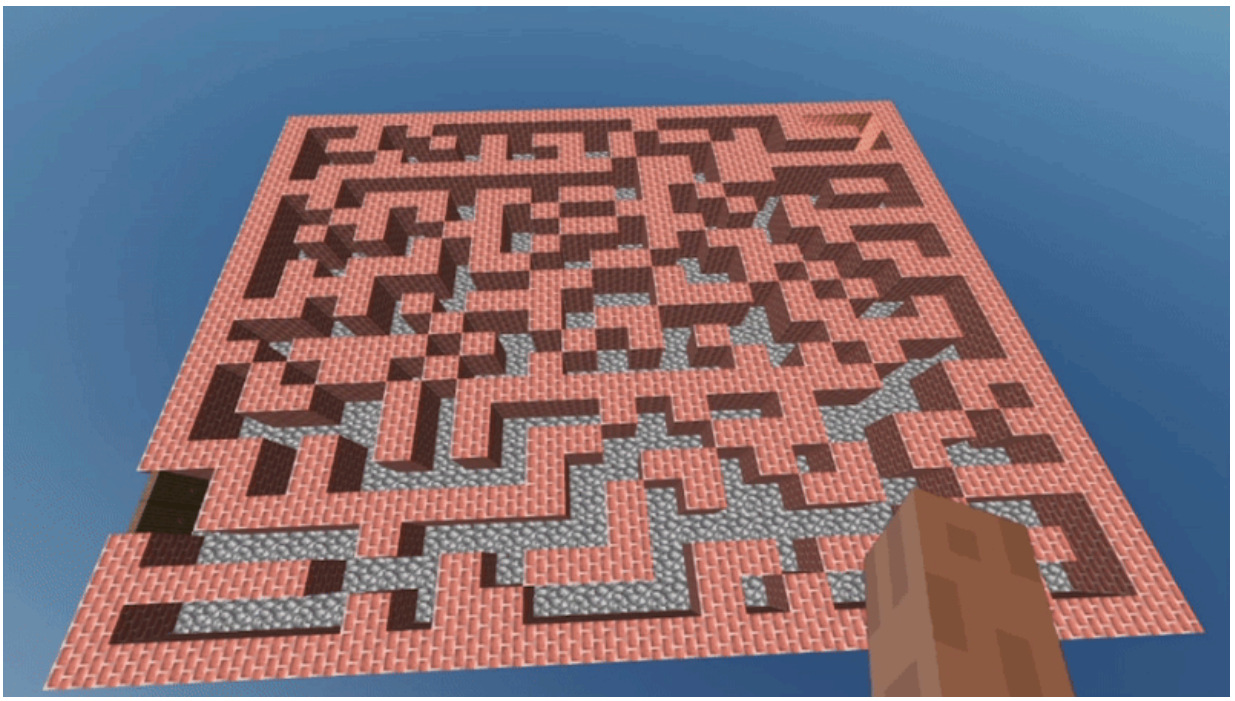
^ wygenerowany labirynt w grze

Logika gry:
Gracz ma za zadanie znalezienia klucza lub kilku kluczy, które po zebraniu otwierają bramę potrzebną do przejścia gry.

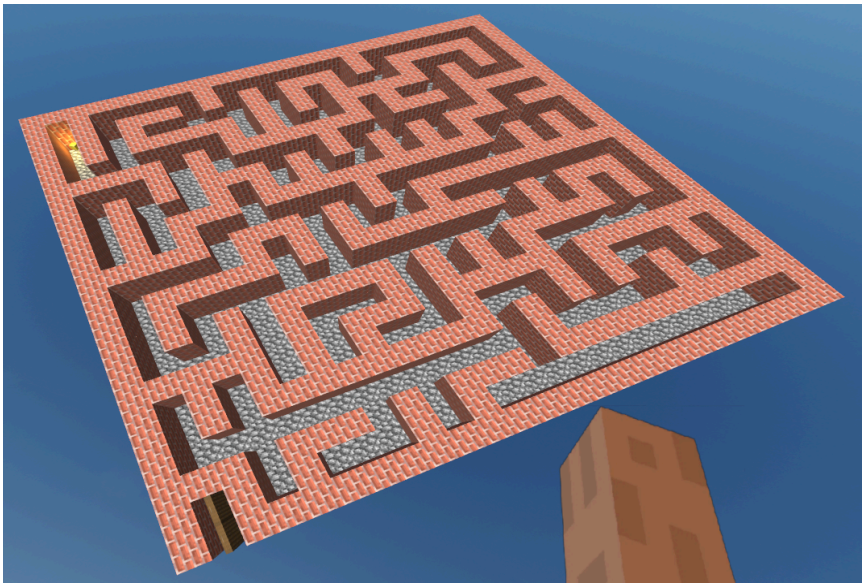


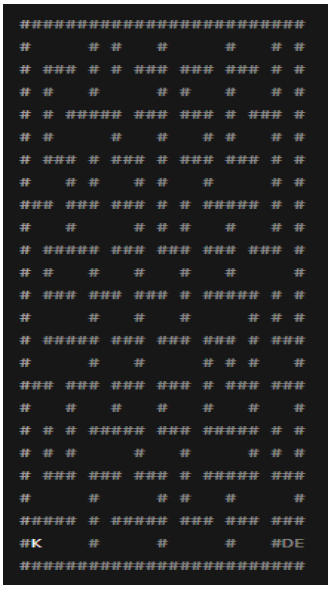
Możliwe rozwinięcie mechanik w przyszłości:

Dodanie poruszających się ścian - Ta mechanika jednak rodzi problemy takie jak na przykład aktualizowanie hitboxów ścian w czasie rzeczywistym, tak aby gracz był przesuwany razem ze ścianą, czy potencjalne uniemożliwienie ukończenia gry.



Generowanie losowych labiryntów przy użyciu AI - Na przykład API od OpenAI. Problem pojawia się przy sprawdzaniu czy wygenerowany labirynt jest zdatny do użytku, gdyż niestety większość wygenerowanych labiryntów przez AI nie ma sensu i nie da się ich przejść - przykład poniżej.

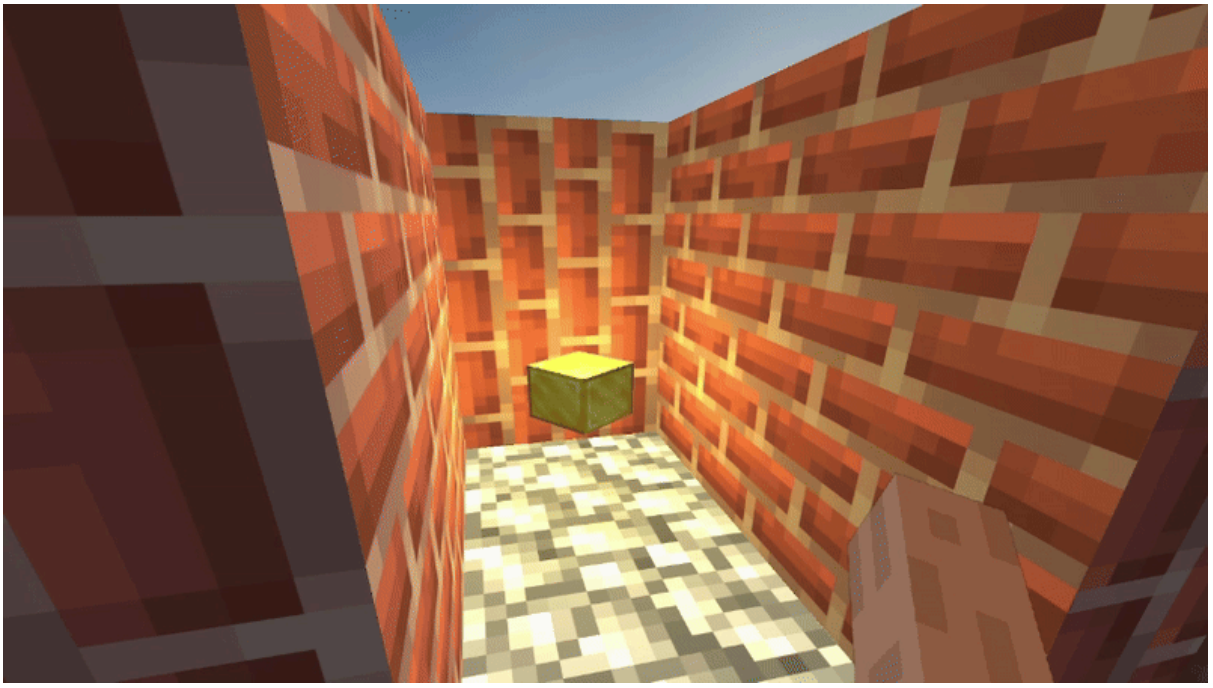




Ekran końcowy z podsumowaniem czasu:

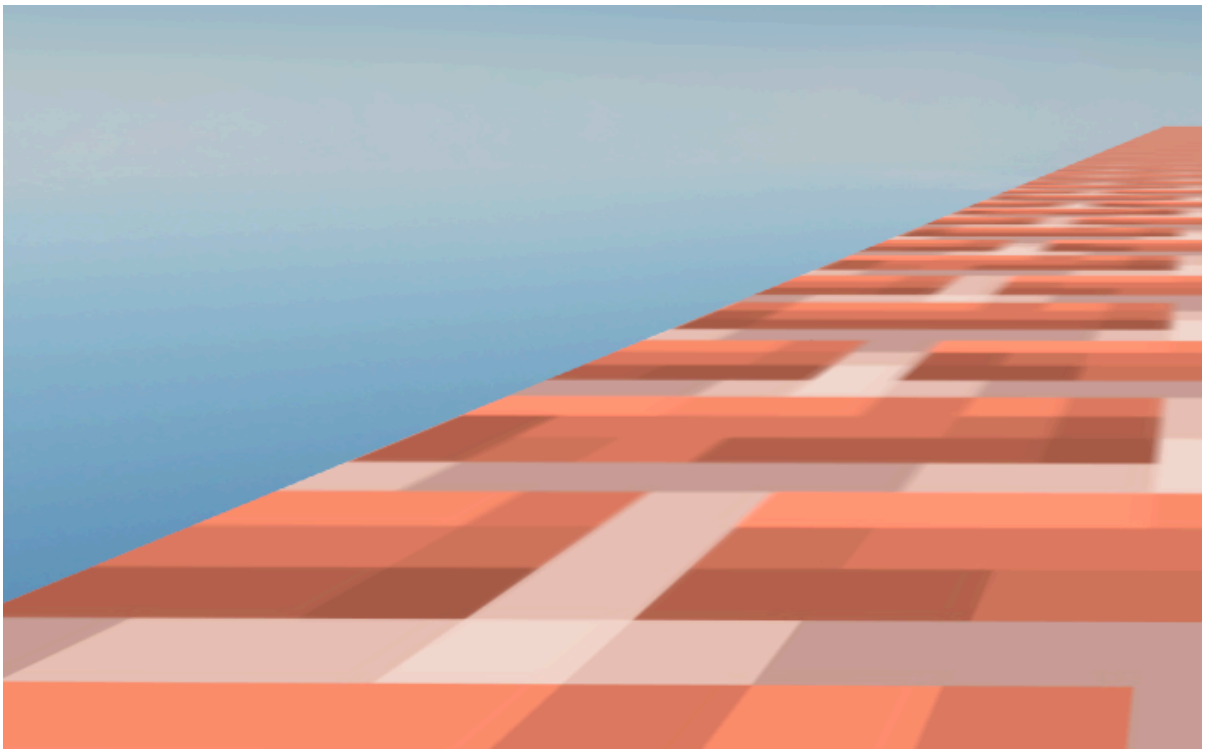
Grafika:

Phong lighting, animacja i tekstury



Skybox

Trilinear filtering i multisampling



Fragmenty kodu:

Kod do wyświetlania tekstur i shaderów:

```
// Renderowanie podłogi
ground.Render(*shader);

// Renderowanie labiryntu
maze.Render(*shader);
```

```
// Renderowanie kluczy
shader->setFloat("keyLightIntensity", 0.0f);

for (auto& key : keys) {
    key.Render(*shader, camera.GetViewMatrix(), camera.GetProjectionMatrix());
}
// Renderowanie drzwi
for (auto& door : doors) {
    door.Render(*shader, camera.GetViewMatrix(), camera.GetProjectionMatrix());
}
for (auto& door : doors) {
    door.RenderPortal(*portalShader, camera.GetViewMatrix(), camera.GetProjectionMatrix());
}
// Renderowanie skyboxa
if (skybox) {
    skybox->Draw(camera.GetViewMatrix(), camera.GetProjectionMatrix());
}
// Renderowanie UI
ui.Render();
```

Kod do aktualizowania danych:

```
void Game::Update() {
    if (!gameFinished) {
        camera.Update();
        camera.UpdatePhysics(deltaTime);
    }
    else {
        camera.UpdatePrevPosition();
    }

    for (auto& key : keys) {
        if (!key.IsCollected() &&
            glm::distance(camera.GetPosition(), key.GetPosition()) < 1.2f) {
            key.Collect();
        }
    }

    // Sprawdź, czy wszystkie klucze zostały zebrane
    if (!allKeysCollected) {
        allKeysCollected = true;
        for (const auto& key : keys) {
            if (!key.IsCollected()) {
                allKeysCollected = false;
                break;
            }
        }
    }
    if (allKeysCollected) {
        for (auto& door : doors) {
            door.RemoveCollidersFrom(maze);
        }
    }
    for (auto& door : doors) {
        door.Update(deltaTime, camera.GetPosition(), allKeysCollected);
    }

    if (!gameFinished && allKeysCollected && glm::distance(camera.GetPosition()
        endTime = glfwGetTime();
        gameFinished = true;
        std::cout << "Game finished";

    }
```

Generowanie ścian z pliku:

```
unsigned int Maze::GenerateWallTexture() {
    int texWidth = width;
    int texHeight = height;
    std::vector<float> wallData(texWidth * texHeight, 0.0f);

    for (int z = 0; z < height; ++z) {
        for (int x = 0; x < width; ++x) {
            if (map[z][x] == '#') {
                wallData[z * width + x] = 1.0f;
            }
        }
    }
```

Ładowanie tekstur:

```
unsigned int LoadTexture(const char* path) {
    unsigned int textureID;
    glGenTextures(1, &textureID);

    int width, height, nrChannels;
    stbi_set_flip_vertically_on_load(true);
    unsigned char* data = stbi_load(path, &width, &height, &nrChannels, 0);
    if (data) {
        GLenum format = (nrChannels == 1) ? GL_RED :
            (nrChannels == 3) ? GL_RGB :
            GL_RGBA;

        glBindTexture(GL_TEXTURE_2D, textureID);
        glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

        // Domyœlne parametry tekstury (mog¹ byæ nadpisane póŸniej)
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        stbi_image_free(data);
    }
    else {
        std::cerr << "Failed to load texture: " << path << std::endl;
        stbi_image_free(data);
    }

    return textureID;
}
```

Animacja klucza:

```
void Key::Render(Shader& shader, const glm::mat4& view, const glm::mat4& projection) {
    if (collected) return;

    // Przeka¿ parametry œwiat³a klucza do shadera
    shader.use();
    shader.setVec3("keyLightPos", position + glm::vec3(0.0f, 0.2f, 0.0f)); // Lekko nad kluczem
    shader.setVec3("keyLightColor", lightColor);
    shader.setFloat("keyLightIntensity", lightIntensity);

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, textureID);

    shader.setMat4("view", view);
    shader.setMat4("projection", projection);

    glm::mat4 model = glm::mat4(1.0f);
    model = glm::translate(model, position);
    model = glm::translate(model, glm::vec3(0.0f, 0.1f, 0.0f)); // Delikatne uniesienie
    model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::scale(model, glm::vec3(0.5f)); // Zmniejszenie rozmiaru
    shader.setMat4("model", model);

    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, vertices.size() / 8);
    glBindVertexArray(0);
}
```

Otwieranie portalu:

```
void Door::Update(float dt, const glm::vec3& playerPos, bool hasKey) {
    glm::vec3 center = (leftPosition + rightPosition) * 0.5f;
    float dist = glm::distance(center, playerPos);
    if (!isOpening && hasKey && dist < triggerDistance) {
        isOpening = true;
        portal.Show();
    }
    if (isOpening) {
        float delta = openSpeed * dt;
        leftAngle = glm::clamp(leftAngle - delta, -maxAngle, 0.0f);
        rightAngle = glm::clamp(rightAngle + delta, 0.0f, maxAngle);
        portal.Update(dt);
    }
}
```