Państwowa Wyższa Szkoła Zawodowa w Tarnowie
Katedra Informatyki

Labirynt 3D

Przedmiot:	**Grafika 3D i programowanie kart graficznych II**
Prowadzący:	**dr. inż. Jędrzej Byrski**
Kierunek:	**Informatyka**
Semestr:	**4**
Rok:	**2**
Grupa:	**PO2**
Autorzy:	**Robert Małek, Jakub Niemiec, Jakub Serwin**

Tarnów, 30 Maj 2021

Opis projektu

Implementacja gry polegającej na wydostaniu się z losowo generowanego labiryntu. Gracz po starcie pojawia się w mrocznym otoczeniu, którego charakter zbudowaliśmy dzięki wykorzystaniu odpowiednio dobranych tekstur a do dyspozycji jako jedyne źródło światła ma latarkę. Jego zadaniem jest znalezienie klapy w podłożu, która umożliwia mu wydostanie się z niego i zarazem kończy rozgrywkę zwycięstwem.

Tworząc projekt nasz zespół musiał stawić czoła kilkunastu zadaniom począwszy od zaplanowania projektu, dobrania odpowiednich bibliotek, wybrania tekstur, dobrego researchu, zaimplementowania poruszania się kamery, dodania obiektów i tekstur, wykrywania kolizji z tymi obiektami, dodania dźwięku, sterowania oraz wiele innych pomniejszych zadań z nimi związanych. Przy tym wszystkim pamiętając o zasadach i dobrych praktykach w trakcie pisania kodu oraz wiedzy uzyskanej z tego przedmiotu w poprzednim semestrze.

Naszym celem była wciągająca, wywołująca lekkie dreszcze gra, w którą będziemy mogli sami pograć oraz udostępnić ją naszym znajomym, a kto wie może i w przyszłości szerszemu gronu po wcześniejszym rozbudowaniu projektu o dodatkowe funkcjonalności o których wspomniane zostanie w dalszej części dokumentacji.

Analiza problemu oraz wykorzystane algorytmy i narzędzia

Aby gra była bardziej ciekawa chcieliśmy aby labirynt był generowany losowo, to stanowiło główny problem naszego projektu, rozwiązaliśmy go dzięki wykorzystaniu algorytmu z biblioteki maze_generator. Również musieliśmy zadbać o podstawowe funkcjonalności każdej gry wykorzystując algorytmy dostarczane przez uniwersalne API do tworzenia grafiki - OpenGL. Na stan OpenGL w danym momencie składa się szereg parametrów i trybów działania, które można ustawić lub zapamiętać na stosie i później odtworzyć. Ich konfiguracja będzie miała bezpośredni lub pośredni wpływ na otrzymany rezultat renderingu.

Wykorzystane technologie i narzędzia

Technologie

- C++
- OpenGL

Dodatkowe biblioteki

- GLAD
- GLFW
- GLM
- irrKlang

Narzędzia

- Messenger - podział i przydzielenie zadań
- Messenger - komunikacja
- Git - zarządzanie historią kodu źródłowego
- Visual Studio, CLion - edytory kodu

Implementacja

Poszczególne etapy programowania, których zaimplementowanie doprowadziło do finalnego projektu:

1. Podpięcie bibliotek
2. Utworzenie obiektów maze i camera oraz zmiennych z ustawieniami
3. Zainicjalizowanie GLFW i GLAD
4. Załadowanie shaderów
5. Utworzenie współrzędnych obiektów
6. Utworzenie VAO, VBO i przypisanie im odpowiednich wartości

```
unsigned int skyboxVAO, skyboxVBO;
    glGenVertexArrays(1, &skyboxVAO);
    glGenBuffers(1, &skyboxVBO);
    glBindVertexArray(skyboxVAO);
    glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices),
&skyboxVertices, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 *
sizeof(float), (void*)0);
```

```
   unsigned int floorVBO, floorVAO;
    glGenVertexArrays(1, &floorVAO);
    glGenBuffers(1, &floorVBO);

    glBindVertexArray(floorVAO);

    glBindBuffer(GL_ARRAY_BUFFER, floorVBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(floorVertices),
floorVertices, GL_STATIC_DRAW);
```

```
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 14 *
sizeof(float), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 14 *
sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 14 *
sizeof(float), (void*)(6 * sizeof(float)));
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, 14 *
sizeof(float), (void*)(8 * sizeof(float)));
    glEnableVertexAttribArray(4);
    glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE, 14 *
sizeof(float), (void*)(11 * sizeof(float)));
```

7. Załadowanie tekstur

```
    unsigned int floorDiffuseMap =
loadTexture("assets/textures/gravel/albedo.jpg");
    unsigned int floorNormalMap =
loadTexture("assets/textures/gravel/normal.jpg");

    unsigned int wallDiffuseMap =
loadTexture("assets/textures/brickwall/albedo.jpg");
    unsigned int wallNormalMap =
loadTexture("assets/textures/brickwall/normal.jpg");

    unsigned int trapdoorDiffuseMap =
loadTexture("assets/textures/trapdoor/albedo.jpg");
    unsigned int trapdoorNormalMap =
loadTexture("assets/textures/trapdoor/normal.jpg");
```

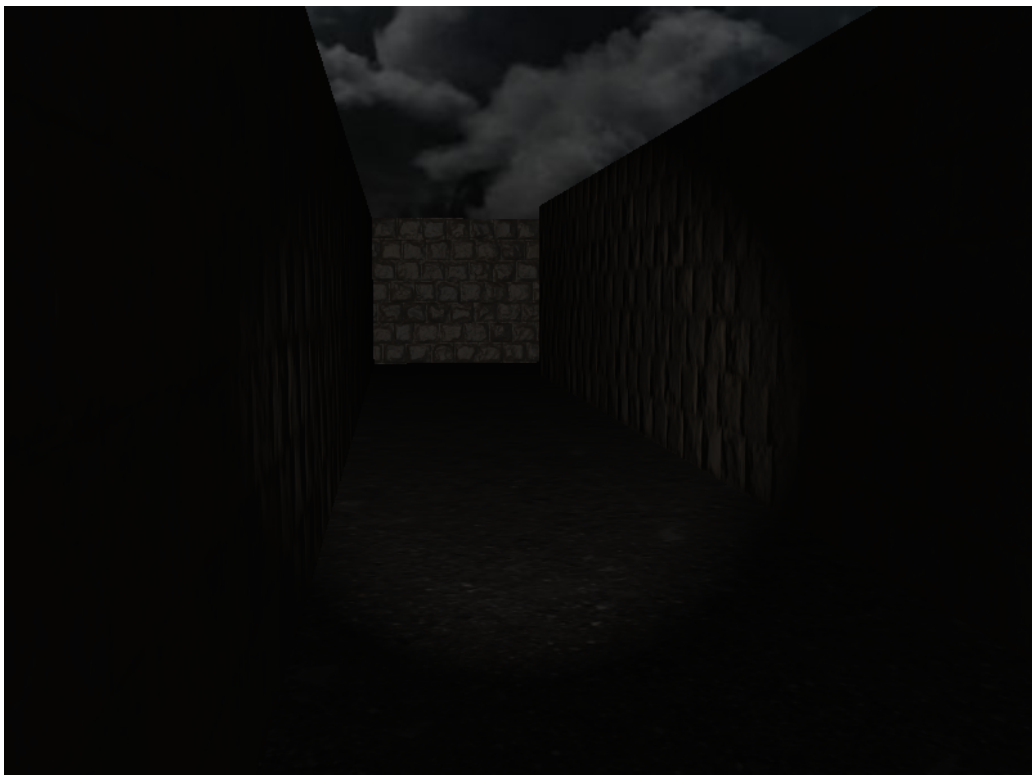
8. Przypisanie odpowiednich wartości do zmiennych w shaderach

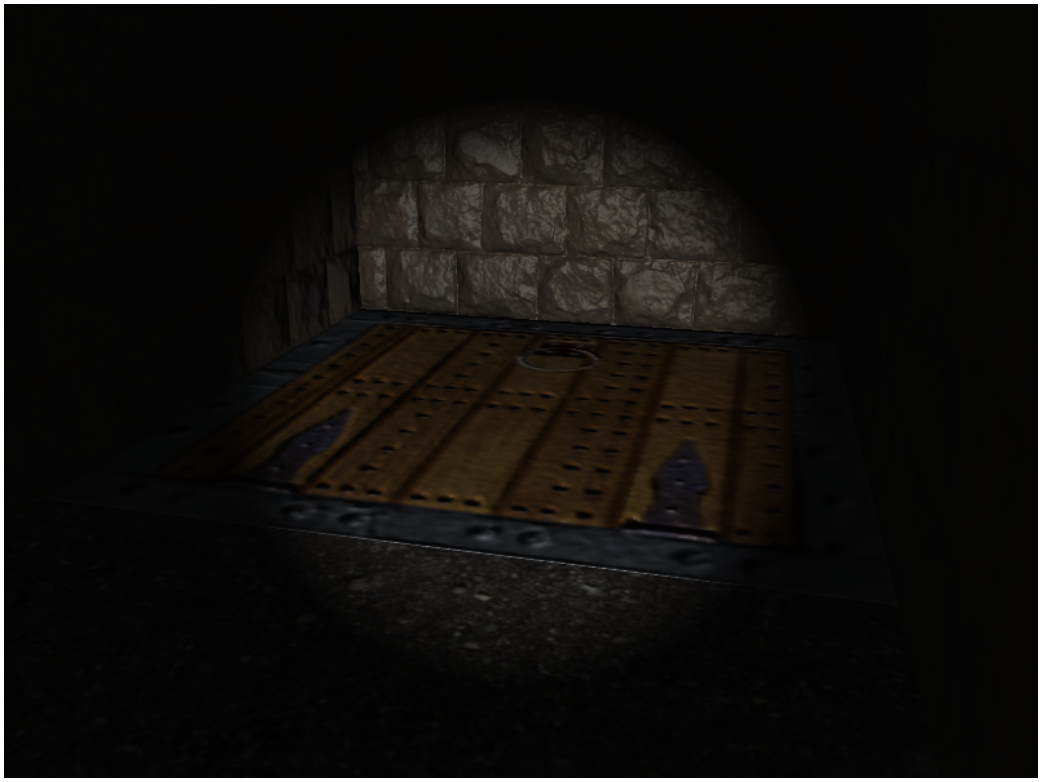
9. Wyrenderowanie obiektów
10. Obsługa danych wejściowych z klawiatury i myszki, kamery

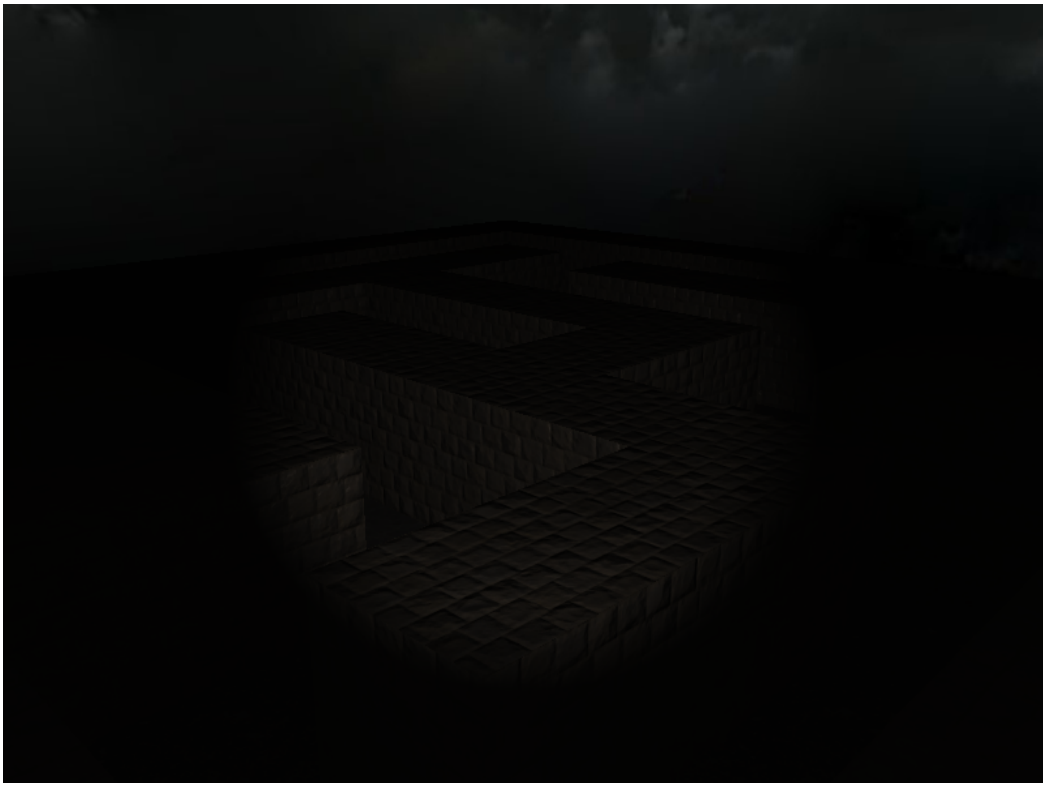
```
void processInput(GLFWwindow* window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
        camera.ProcessKeyboard(FORWARD, deltaTime);
    }

    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyboard(BACKWARD, deltaTime);


    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(LEFT, deltaTime);


    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(RIGHT, deltaTime);


    if (glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) {
        if (camera.FlyMode == false) {
            camera.returnPosition = camera.Position;
        }
        else {
            camera.Position = camera.returnPosition;
        }
        camera.FlyMode = !camera.FlyMode;
    }
}
```

11. Sprawdzenie czy gracz trafił do wyjścia

Prezentacja aplikacji







Testowanie

Program był testowany przez wszystkich członków naszego zespołu na różnych urządzeniach pod kątem wydajności oraz potencjalnych błędów, które po zidentyfikowaniu od razu staraliśmy się poprawić. W tym celu wcieliliśmy się w rolę gracza w celu znalezienia tzw. bugów oraz zbadania płynności gry.

Przetestowaliśmy podstawową funkcjonalność gry a mianowicie czy:

- odpowiednie klawisze wyzwalają przypisane im metody sterowania
- zostały załadowane odpowiednie tekstury i są poprawnie wyświetlane
- kolizja z obiektami działa w poprawny sposób
- obracanie kamery jest poprawne
- dźwięk jest wywoływany w odpowiednim momencie
- labirynt jest losowo generowany za każdym razem
- znalezienie wyjścia kończy grę
- oraz wiele innych pomniejszych funkcji

Również zadbaliśmy o to aby kod był czytelny, nazewnictwo zmiennych, funkcji itp. zgodne z ustalonymi normami. Oraz odpowiednio udokumentowaliśmy kod i na końcu wygenerowali dokumentacje przy użyciu generatora Deoxygen

Dalsze możliwości rozbudowy

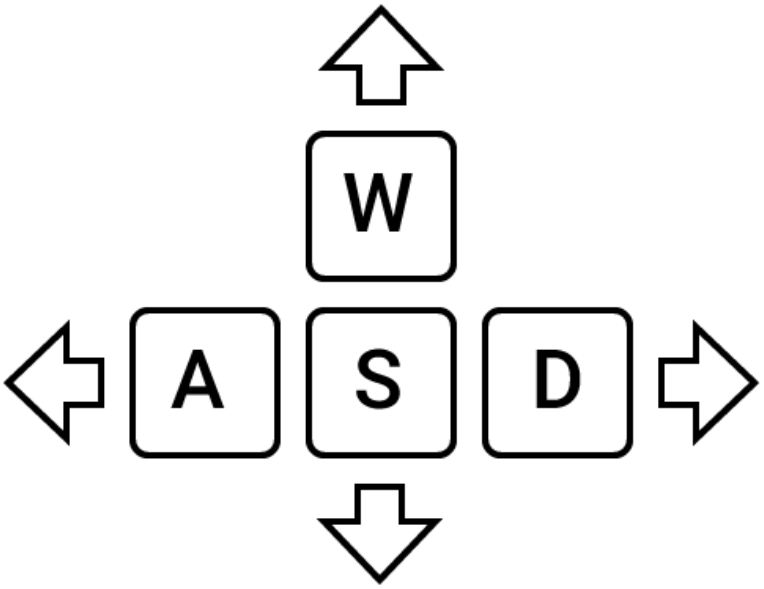
Aby projekt miał szansę dotrzeć do szerszego grona odbiorców mamy pomysły na kilka możliwości jego dalszej rozbudowy. Duża część osób lubi spędzać swój czas w grze ze znajomymi dlatego pierwszą z nich jest dodanie możliwości gry nie tylko samemu ale w grupach do czterech osób czy to online czy przez podzielony ekran na jednym komputerze. Również samo chodzenie i szukanie tylko i wyłącznie wyjścia może być niewystarczające dla niektórych graczy dlatego kolejnym etapem byłoby dodanie urozmaiceń i trudności w postaci przeszkód oraz przeciwników, których możemy spotkać w labiryncie. Ciekawym pomysłem również wydaje się wprowadzenie dodatkowym pomieszczeń lub lokacji w labiryncie gdzie gracz musiałbym wykonać konkretne zadania aby się z nich wydostać. Dodany ekwipunek również otworzyłby wiele drzwi rozbudowy, moglibyśmy wtedy dodać baterię bez których gracz straciłby swoje jedyne źródło światła a zdobyć je mogłby przemierzając labirynt. Skoro umieścilibyśmy też przeciwników to aby zrekompensować to graczowi dodalibyśmy mu broń i amunicję oraz inne atrakcyjne rzeczy, które pomogłyby mu pokonać labirynt i jego wymagające wyzwania.

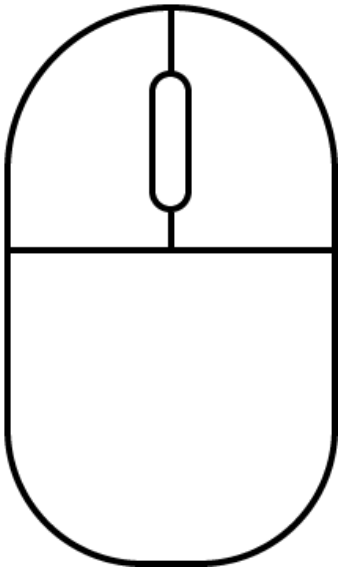
Podsumowując w skrócie możliwości dalszej rozbudowy to:

- Możliwość gry ze znajomymi
- Dodanie przeszkód oraz przeciwników
- Dodatkowe lokacje wymagające wykonania zadań aby się z nich wydostać
- Ekwipunek a w nim broń oraz inne rzeczy
- Ograniczona ilość baterii i amunicji, którą można znaleźć w różnych miejscach w labiryncie

Instrukcja obsługi i kompilacji

Poruszanie się w grze





W - Ruch do przodu
A - Ruch w lewo
S - Ruch do tyłu
D - Ruch w prawo
Q - Tryb wolnej kamery
Mysz - obracanie kamerą
Scroll - przybliżenie/oddalenie kamery

Program nie wymaga wcześniejszej kompilacji użytkownika w celu zagrania. Wystarczy że tak jak w przypadku większości gier komputerowych gracz uruchomi plik **.exe** który może znaleźć w folderze bin