Imię i nazwisko: Julia Antoniewicz Aleksandra Harłacz Patrycja Jakson	Kierunek: INO	Rok studiów i grupa 1 rok, grupa 1
Data zajęć: 18.12.2025	Temat sprawozdania: Projekt zaliczeniowy	

Nazwa projektu: „Ucieczka z AGH”

Cel i opis projektu:

Gracz wciela się w studenta, którego zadaniem jest przejść 5 poziomów–tyle ile lat studiów. Każdy poziom to losowo generowany labirynt, w którym trzeba:

- odnaleźć drogę do wyjścia (zaliczenie roku),
- unikać pułapek,
- zbierać punkty ECTS i ulepszenia,
- walczyć z rosnącą trudnością (większe labirynty, więcej pułapek).

Po każdym poziomie student zdobywa ulepszenia.

Celem jest dotarcie do poziomu 5 (koniec studiów magisterskich), przejście największego i najtrudniejszego labiryntu i "obronić magisterkę".

Zastosowane techniki programistyczne:

Program został podzielony na mniejsze funkcje, dzięki czemu jest czytelny i łatwy do modyfikacji. Do przechowywania danych użyliśmy struktur oraz wektorów. Działanie gry opiera się na przełączaniu stanów (menu, logowanie, gra, podsumowanie). Program reaguje na działania użytkownika za pomocą klawiatury. Do losowych elementów gry użyto funkcji losujących, a dane użytkownika są zapisywane do pliku.

Wykorzystane biblioteki:

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <ctime>
#include <fstream>

#include <SFML/Graphics.hpp>
```

Biblioteka <SFML> odpowiada za rysowanie i wyświetlanie multimediów, grafik.

Biblioteka <vector> to dynamiczna tablica, która przechowuje elementy tego samego typu w ciągłym bloku pamięci.

Biblioteka <cstdlib> zawiera funkcje ogólne takie jak konwersje, alokacja pamięci lub funkcje matematyczne.

Biblioteka <algorithm> ma zdefiniowany szereg wydajnych algorytmów ułatwiających prace programiście, ale również przyspieszających działanie programu.

Biblioteka <string> jest odpowiedzialna za obiektowe podejście do ciągów znaków. Dzięki niej korzystanie jest wygodniejsze niż ze standardowej obsługi tablicy znaków (char *).

Biblioteka <ctime> zawiera funkcje, dzięki którym można przekształcić wartość czasu na postać ciągów znaków i obliczyć ile upłynęło go od startu do końca.

Biblioteka <fstream> daje dostęp do funkcji pozwalającej na zapisywanie plików jak i ich odczyt.

Instalacja biblioteki SFML

1. Pobranie wersji SFML 2.6.2 64bit.

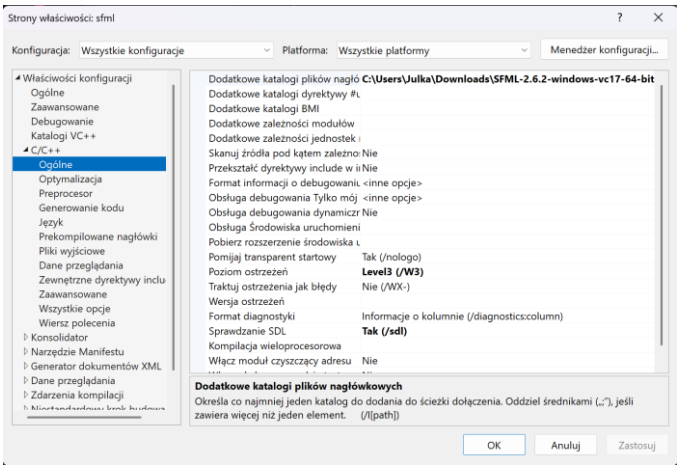
2. Stworzenie nowego projektu aplikacji konsolowej.

3. Ustawienia projektu:

[wszystkie konfiguracje] [wszystkie platformy]

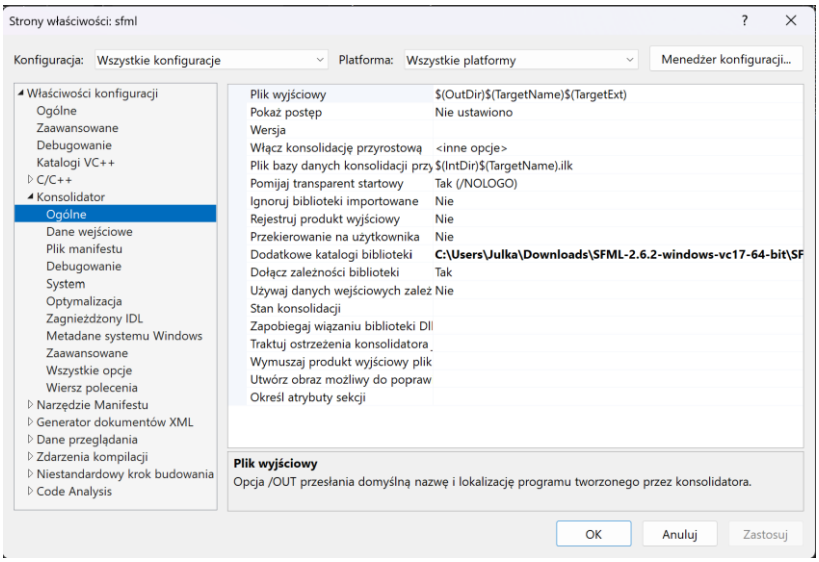
C/C++ / ogólne / dodatkowe katalogi plików nagłówkowych

D:\SFML-2.6.1\include – wklejamy swoją ścieżkę do include



konsolidator / ogólne / dodatkowe katalogi biblioteki

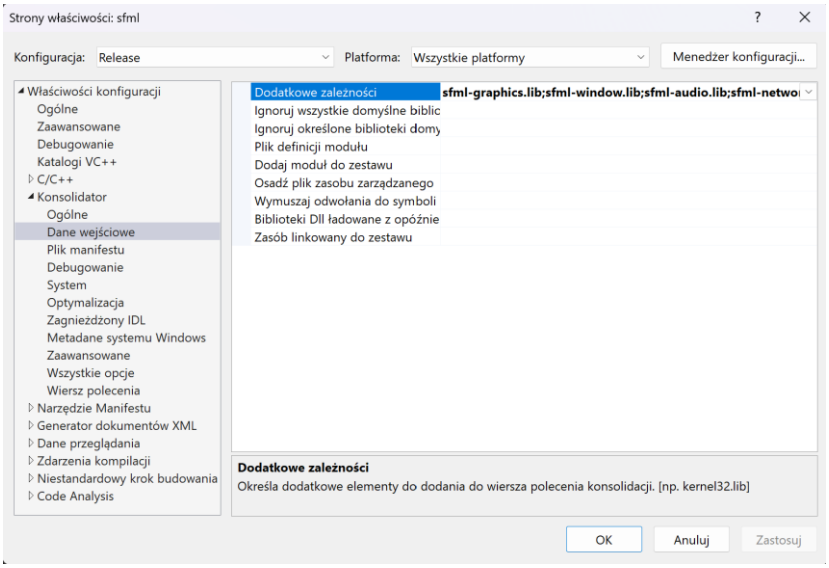
D:\SFML-2.6.1\lib – wklejamy swoją ścieżkę do katalogu lib



[Release] [wszystkie platformy]

konsolidator / dane wejściowe / dodatkowe zależności:

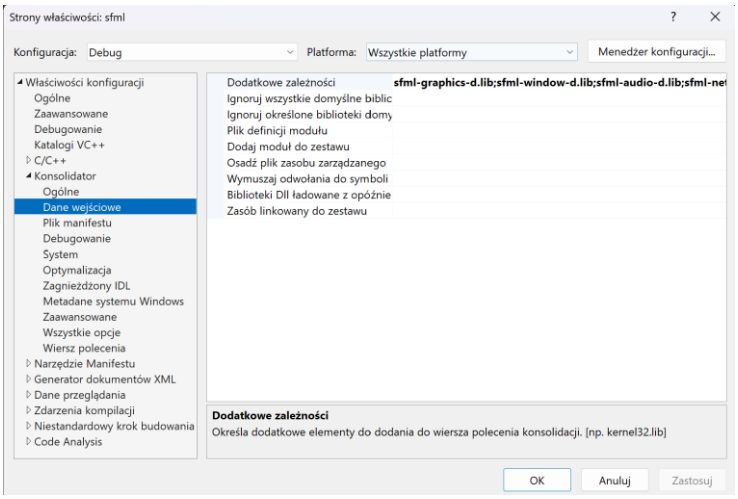
sfml-graphics.lib;sfml-window.lib;sfml-audio.lib;sfml-network.lib;sfml-system.lib;



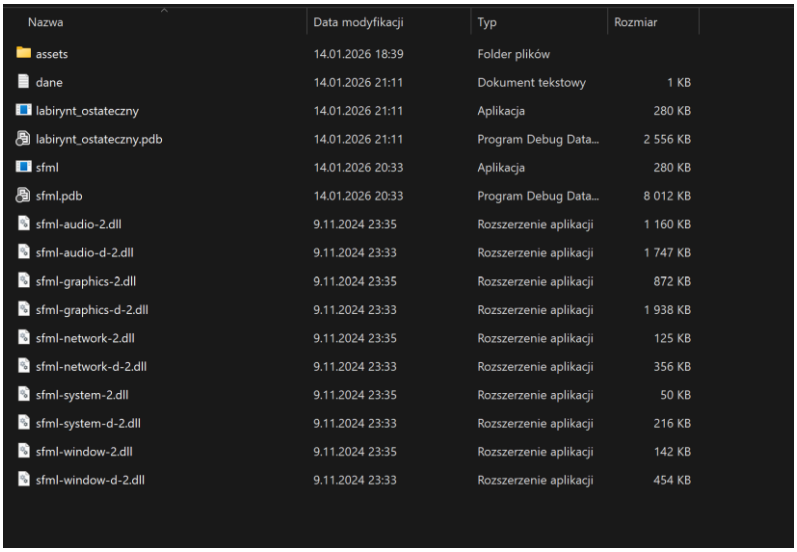
[Debug] [wszystkie platformy]

konsolidator / dane wejściowe / dodatkowe zależności:

sfml-graphics-d.lib;sfml-window-d.lib;sfml-audio-d.lib;sfml-network-d.lib;sfml-system-d.lib;



4. Przeniesienie plików z SFML/bin do folderu projektu.



Wybrane funkcjonalności:

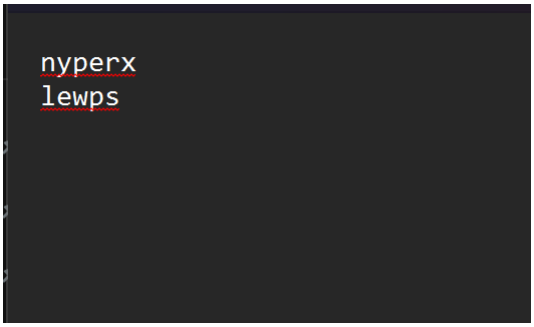
1) pomiar czasu

```
clock_t start=0;

//dalsze main

start=clock();

//dalsze main

clock_t stop = clock();
double czas = (double)(stop-start)/CLOCKS_PER_SEC;
summaryMinutes = totalSeconds / 60;
```

```
summarySeconds = totalSeconds % 60;
```

2) logika autoryzacji i szyfrowania (szyfr Cezara)

```
string szyfrowanie(string tekst) {
    int n = 4;
    for (int i = 0; i < (int)tekst.length(); i++) {
        tekst[i] = tekst[i] + n;
    }
    return tekst;
}
```



3) Struktury stanu gry

Stan gracza

```
struct Player {
    int x = 0, y = 0;
    int knowledge = 100;
```

```
    int ects = 0;
    int energia = 3;
    int talizman = 0;
    int totalECTS = 0;
    vector<string> inventory;
};
```

Wymagania poziomu

```
struct LevelRequirements {
    vector<string> needed;
    bool hasAll() { return needed.empty(); }
};
```

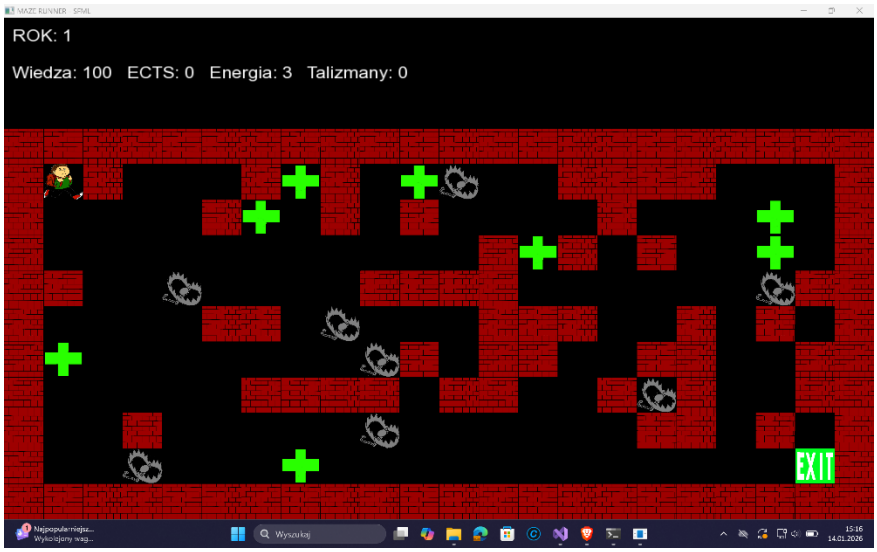
4) Dodatkowe wymagania na poziomach

```
void setupLevelRequirements() {
    player.inventory.clear();
    currentReqs.needed.clear();
    if (rok == 3) {
        currentReqs.needed = { "Promotor", "Praca" };
    }
    else if (rok == 5) {
        currentReqs.needed = { "Praca", "Promotor", "Ankiety", "Literatura"
};
    }
}
```

5) Generowanie labiryntu z rozmieszczeniem obiektów

```
void generateMaze() {
    maze.assign(HEIGHT, vector<char>(WIDTH, ' '));
    for (int x = 0; x < WIDTH; x++) {
        maze[0][x] = '|';
        maze[HEIGHT - 1][x] = '|';
    }
    for (int y = 0; y < HEIGHT; y++) {
        maze[y][0] = '|';
        maze[y][WIDTH - 1] = '|';
    }

    vector<pair<int, int>> path;
    int cx = 1, cy = 1;
    path.push_back({ cx, cy });
    while (cx < WIDTH - 2 || cy < HEIGHT - 2) {
        if (rand() % 2 == 0 && cx < WIDTH - 2) cx++;
        else if (cy < HEIGHT - 2) cy++;
        path.push_back({ cx, cy });
    }

    for (int yy = 1; yy < HEIGHT - 1; yy++) {
        for (int xx = 1; xx < WIDTH - 1; xx++) {
            bool isPath = false;
            for (auto p : path) { if (p.first == xx && p.second == yy) {
isPath = true; break; } }
```

```
                if (!isPath && rand() % 100 < 30) maze[yy][xx] = '|';
            }
        }

        // artefakty
     for (const string& item : currentReqs.needed) {
            bool placed = false;
         while (!placed) {
                int rx = 1 + rand() % (WIDTH - 2), ry = 1 + rand() % (HEIGHT -
    ;
                if (maze[ry][rx] == ' ') {
                    maze[ry][rx] = docCharFor(item);
                    placed = true;
                }
            }
        }

        // ECTS
        for (int i = 0; i < (WIDTH * HEIGHT / 20); i++) {
            int rx = 1 + rand() % (WIDTH - 2), ry = 1 + rand() % (HEIGHT - 2);
            if (maze[ry][rx] == ' ') maze[ry][rx] = '+';
        }

    // pułapki
    for (int i = 0; i < (WIDTH * HEIGHT / 25); i++) {
         int rx = 1 + rand() % (WIDTH - 2), ry = 1 + rand() % (HEIGHT - 2);
            if (maze[ry][rx] == ' ') maze[ry][rx] = 'O';
    }

    maze[HEIGHT - 2][WIDTH - 2] = 'E';

    player.x = 1;
    player.y = 1;
    maze[player.y][player.x] = '@';
}
```



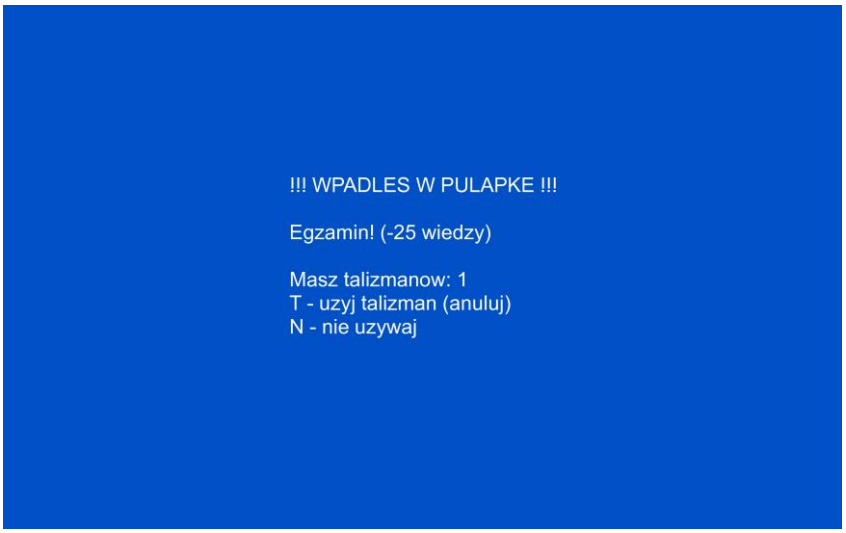
6) Ruch gracza

```
    void movePlayer(int dx, int dy) {
        if (inverted) {
            dx = -dx;
            dy = -dy;
            if (--invertedTurns <= 0) inverted = false;
        }
        int nx = player.x + dx;
```

```
        int ny = player.y + dy;
        if (maze[ny][nx] == '|') return;

        char tile = maze[ny][nx];

        if (tile == 'O') steppedOnTrap = true;

        // artefakty
        if (tile == 'P' || tile == 'R' || tile == 'N' || tile == 'L') {
            string doc = docNameForChar(tile);
            player.inventory.push_back(doc);
         auto it = find(currentReqs.needed.begin(),currentReqs.needed.end(),
doc);
            if (it != currentReqs.needed.end()) currentReqs.needed.erase(it);
        }

        if (tile == '+') {
            player.ects += 5;
            player.totalECTS += 5;
        }

        if (tile == 'E') {
            if (currentReqs.hasAll()) levelCompleted = true;
            else return;
        }

        maze[player.y][player.x] = ' ';
        if (returnToStart) {
            player.x = 1;
            player.y = 1;
            returnToStart = false;
        }
        else {
            player.x = nx;
            player.y = ny;
        }
        maze[player.y][player.x] = '@';
}
```

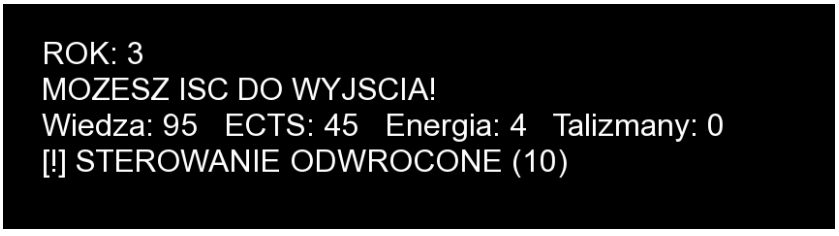
7) logika pułapek i interakcja z Talizmanem

```
// jeśli ma talizman: T/N; jeśli nie: Enter kontynuuje
if (waitingTalisman) {
    if (e.key.code == sf::Keyboard::T) {
        player.talizman--;
        waitingTalisman = false;
        screen = Screen::Game;
    }
```



8) wyświetlanie statystyk i wymagań

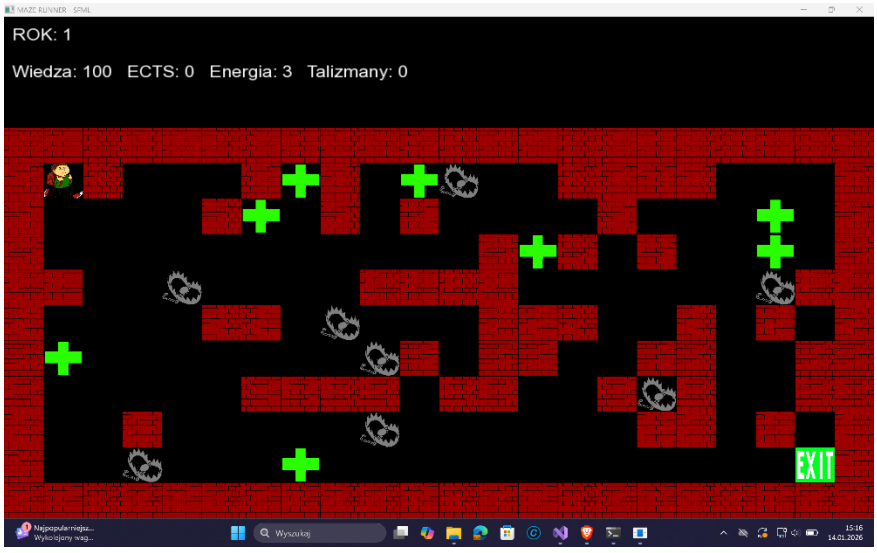
```
string top =
    "ROK: " + to_string(rok) + "\n" +
    neededStr + "\n" +
    "Wiedza: " + to_string(player.knowledge) +
    "   ECTS: " + to_string(player.ects) +
    "   Energia: " + to_string(player.energia) +
    "   Talizmany: " + to_string(player.talizman);

if (inverted) top += "\n[!] STEROWANIE ODWROCONE (" + to_string(invertedTurns) +
")";

hud.setString(top);
hud.setPosition(40.f, 30.f);
window.draw(hud);
```



9) Renderowanie gry (Najważniejsze fragmenty)

```
void drawGame(sf::RenderWindow& window, Assets& assets) {
    const int HUD_H = 150;
    const int MARGIN = 30;

    const auto ws = window.getSize();
    const int screenW = (int)ws.x;
    const int screenH = (int)ws.y;

    const int availW = screenW - 2 * MARGIN;
    const int availH = (screenH - HUD_H) - 2 * MARGIN;
```

```
        int tileW = availW / WIDTH;
        int tileH = availH / HEIGHT;
        int TILE = min(tileW, tileH);
        if (TILE < 8) TILE = 8;

        …

        for (int y = 0; y < HEIGHT; y++) {
            for (int x = 0; x < WIDTH; x++) {
                char c = maze[y][x];

                const sf::Texture* tex = nullptr;
                if (c == '|') tex = &assets.tWall;
                else if (c == '@') tex = &assets.tPlayer;
                else if (c == '+') tex = &assets.tEcts;
                else if (c == 'O') tex = &assets.tTrap;
                else if (c == 'E') tex = &assets.tExit;
                else if (c == 'P') tex = &assets.tPromotor;
                else if (c == 'R') tex = &assets.tPraca;
                else if (c == 'N') tex = &assets.tAnkiety;
                else if (c == 'L') tex = &assets.tLiteratura;
                else continue;

                …

                window.draw(spr);
            }
        }

        window.display();
}
```

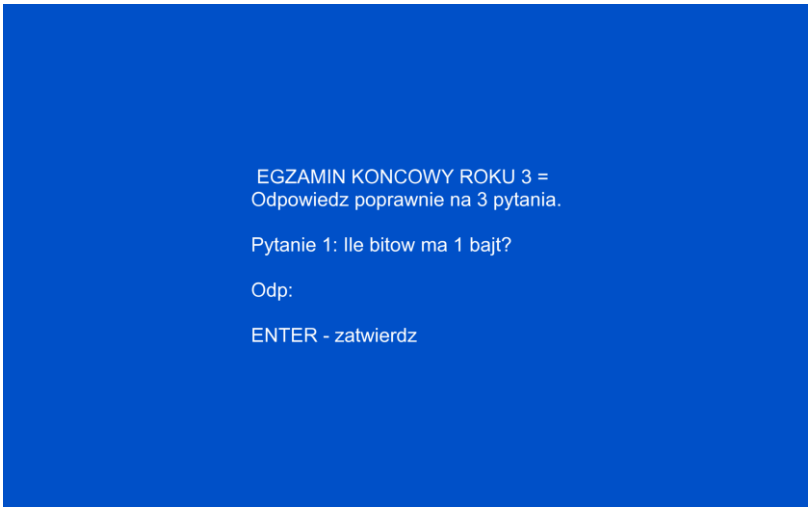


10) Wyświetlanie ekranów gry z sterowaniem

```
    //ekrany
    enum class Screen {//tylko jeden aktywny ekran na raz
        StartSplash,
        AuthMenu,
        Register,
        Login,
        Trap,
        EndYear,
        Exam,
        ExamResult,
```

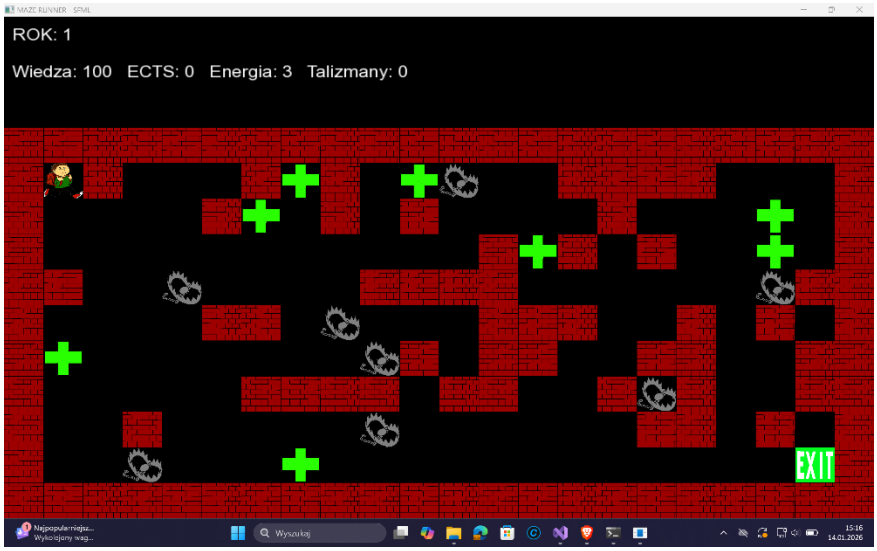
```
            Summary,
            Game
    };

else if (screen == Screen::Game) {
    if (e.key.code == sf::Keyboard::Up) movePlayer(0, -1);
    else if (e.key.code == sf::Keyboard::Down) movePlayer(0, 1);
    else if (e.key.code == sf::Keyboard::Left) movePlayer(-1, 0);
    else if (e.key.code == sf::Keyboard::Right) movePlayer(1, 0);

    if (player.knowledge <= 0 || player.energia <= 0) {
        info = (player.knowledge <= 0 ? "Brak wiedzy! Powtarzasz rok." : "Brak
energii! Powtarzasz rok.");
        startLevel();
    }

    if (levelCompleted) {
        if (rok == 3 || rok == 5) {
            exam = makeExam(rok);
            screen = Screen::Exam;
        }
        else {
            screen = Screen::EndYear;
        }
    }
   }
```

Przykładowy ekran exam



11) Start poziomu

```
    auto startLevel = [&]() {
        WIDTH = 20 + (rok * 2);
        HEIGHT = 10 + rok;
        if (player.knowledge < 50) player.knowledge = 50;
        if (player.energia < 1) player.energia = 1;

        setupLevelRequirements();
        generateMaze();
        levelCompleted = false;

        desktop = sf::VideoMode::getDesktopMode();
        window.create(desktop, "MAZE RUNNER", sf::Style::Fullscreen);
        window.setFramerateLimit(60);

        applyTextStyle(t, window);
    };
```



Link do repozytorium na GitHubie:

https://github.com/julant48/labirynt.git

Instrukcję uruchomienia:

Aby uruchomić grę należy pobrać wszystkie pliki i posiadać skonfigurowaną bibliotekę sfml. Po skopiowaniu kodu jedyne co trzeba zrobić to skompilować program. Gra się uruchomi.