Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

# Programowanie komputerów 4

Sprawozdanie z projektu semestralnego

**autor** Michał Opiełka
**prowadzący** dr.inż. Anna Gorawska
**rok akademicki** 2019/2020
**kierunek** informatyka
**rodzaj studiów** SSI
**semestr** 4
**termin laboratorium** wtorek 13:30-15:00

# 1. Temat projektu

Tematem projektu była gra 2D napisana w języku C++, zaprojektowana przy użyciu biblioteki SFML w wersji 2.5.1. Została ona stworzona w widoku izometrycznym. W grze dysponujemy armią wojowników różnych typów, której celem jest pokonanie przeciwnika.

# 2. Analiza zadania

## 2.1 Struktury danych

### 2.1.1 Lista z biblioteki STL

W celu stworzenia dwóch armii, skorzystano z listy z biblioteki standardowej. Pojedyncze jednostki wraz z graczem zostają wczytane przed rozpoczęciem rozgrywki. Następnie, w głównej pętli gry przez listę przechodzi iterator wywołujący funkcje aktualizujące pozycję, animację, atak oraz renderujące bohaterów na ekran.

### 2.1.2 Zaimplementowana lista jednokierunkowa

Zaimplementowano również listę jednokierunkową w celu stworzenia prostego mechanizmu wyszukiwania drogi z użyciem algorytmu A* (A star). Lista służy przechowywaniu pól odwiedzonych oraz nieodwiedzonych, a po znalezieniu najkrótszej drogi pozwala na wygenerowanie gotowej ścieżki.
“Ręcznie” stworzona lista okazała się minimalnie szybsza. Można ją było także swobodniej modyfikować wycinając i dodając nowe elementy.

### 2.1.3 std::vector

Wektor był najczęściej używaną strukturą danych w projekcie, ponieważ pozwalał na zdecydowanie szybszy dostęp do elementów “w środku” struktury, w przeciwieństwie do listy, pozwalającej jedynie na sprawny dostęp do elementów brzegowych.

Wektor dwuwymiarowy został użyty podczas projektowania mapy. Pozwala to na stworzenie mapy o dowolnym rozmiarze, który nie musi być definiowany w programie.

### 2.1.4 std::map

Mapa została wykorzystana w menadżerze tekstur oraz w menadżerze audio. Struktura ta została wybrana ze względu na łatwość dostępu do elementów przy użyciu unikalnego klucza.

## 2.2 Algorytmy i mechaniki

### 2.2.1 Generowanie mapy

Mapa składa się z trzech warstw: podłoże, flora(kwiaty) oraz obiekty, z którymi można kolidować. Wszystkie warstwy są zapisane w plikach *.txt w postaci cyfr - każda odpowiada jednej teksturze.
W momencie uruchomienia programu wszystkie 3 warstwy są wczytywane do odpowiednich wektorów 2d. Pobierane są także długość i szerokość mapy. Następnie kolejnym wartościom z pobranych plików tekstowych przypisywane są odpowiednie tekstury. Gotowa mapa jest renderowana na ekranie w odpowiedniej kolejności. Najpierw pojawia się podłoże, a zaraz za nim flora. Dopiero po wyrenderowaniu postaci pojawiają się obiekty kolizyjne. Taka kolejność renderowania sprawia wrażenie, że postacie wchodzą za obiekt.

### 2.2.2 Wyszukiwanie drogi

Postacie sterowane przez komputer, w celu wyszukania najkrótszej drogi do swojego przeciwnika korzystają z algorytmu A*. Bazuje on w dużej mierze na algorytmie Djikstry. Jedyną różnicą jest funkcja $f(x) = g(x) + h(x)$ definiująca drogę w linii prostej od punktu startowego do docelowego, będąca odniesieniem do poszukiwania właściwej ścieżki.
$g(x)$ - droga pomiędzy wierzchołkiem początkowym a x.
$h(x)$ - droga od x do wierzchołka docelowego.

Dla właściwego przebiegu algorytmu zostały utworzone dwie listy jednokierunkowe. W jednej przechowywane są już odwiedzone wierzchołki, a w drugiej nieodwiedzone, lecz sąsiadujące z odwiedzonymi. Pętla wykonująca algorytm zostaje przerwana w momencie, kiedy lista nie odwiedzonych wierzchołków jest pusta. Zdarza się to w sytuacji kiedy dotarliśmy do celu lub w przypadku braku możliwości stworzenia ścieżki.
Jeżeli udało się znaleźć drogę, jest ona wpisywana do kolejnej listy i zwracana. Jeżeli droga nie została znaleziona, zwracana jest wartość *nullptr*.

### 2.2.3 Walka

Każdy typ dostępnego oręża ma przypisany hitbox określający obszar, na który może zadawać obrażenia. W przypadku włóczni lub miecza, po wciśnięciu odpowiedniego przycisku rozpoczyna się animacja ataku. W tym momencie, każdy przeciwnik, który znajdzie się w obrębie hitboxa otrzyma obrażenia. Odbywa się to poprzez przejście iteratora przez listę przeciwników i sprawdzenie przecięcia hitboxów.

Do każdej postaci będącej łucznikiem przypisany jest obiekt odpowiadający strzale. Strzała również posiada swój hitbox. Atak łucznika odbywa się poprzez kliknięcie w dowolne miejsce na mapie. Sczytywana zostaje pozycja kursora. Następnie obliczany jest kąt wystrzału oraz prędkość strzały wzdłuż odpowiednich osi. Zadawanie obrażeń odbywa się podobnie jak w przypadku miecza - sprawdzane jest przecięcie hitboxów postaci z hitboxem strzały.

### 2.2.4 Kolizja

Każda z postaci dysponuje hitboxem, co pozwala rozpoznać moment, w którym otrzyma obrażenia lub zetknie się z elementem otoczenia.
Z racji tego, że gra została stworzona w rzucie izometrycznym, postacie sterowane przez komputer posiadają własną, odpowiednio spreparowaną kopię mapy, dzięki której omijają punkty z którymi mogą kolidować.

W przypadku bohatera sterowanego przez gracza sprawdzany jest niewielki obszar 3x3 kratek w celu wykrycia ewentualnej kolizji.
Kiedy kolizja zostaje wykryta, blokowana jest możliwość poruszania się w danym kierunku, co pozwala zawrócić i obrać inną ścieżkę.

#### 2.2.5 Tryb obserwatora

Kiedy postać sterowana przez gracza polegnie, możliwe jest przełączenie się na innych, żywych sojuszników. Po kliknięciu lewym przyciskiem myszy w dowolne miejsce losowany jest jeden “żywy” sojusznik z listy. Zostaje na niego wyśrodkowana kamera i cały czas za nim podąża.

# 3. Specyfikacja zewnętrzna

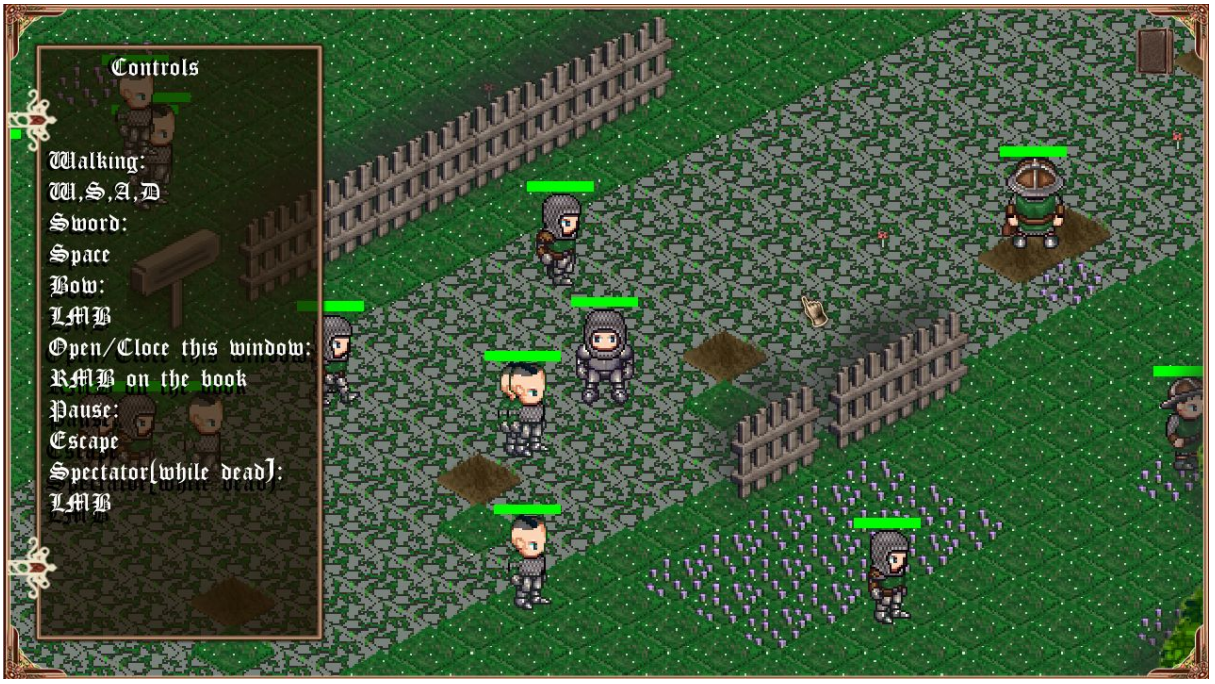
## 3.1 Interfejs

Na interfejs składa się okno z podpowiedzią dotyczącą sterowania wraz z przyciskiem, który to okno aktywuje oraz trzy ekrany - ekran pauzy, ekran informujący o porażce, ekran informujący o zwycięstwie. W przypadku ekranu pauzy istnieje możliwość poddania meczu, co jest równoznaczne z automatyczną porażką. Ekrany informacyjne dysponują jednym przyciskiem, który po wciśnięciu zamyka program.

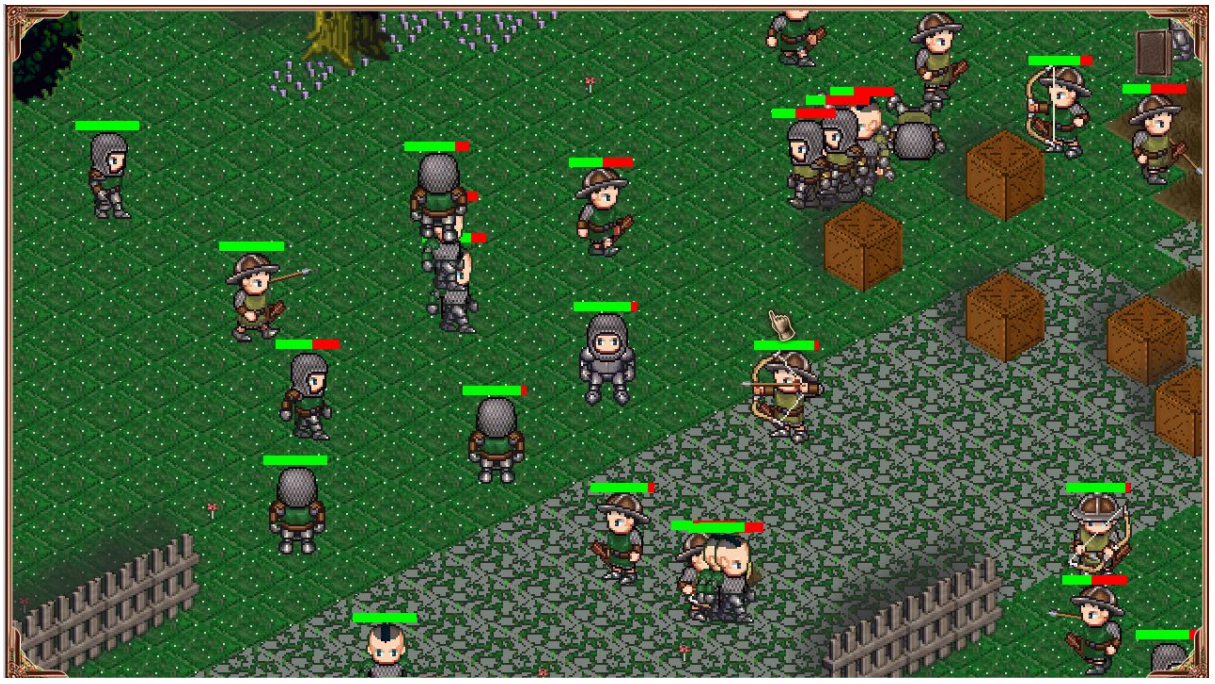
## 3.2 Sterowanie

Bohaterem steruje się przy użyciu myszy oraz klawiatury. Mysz służy do wybierania kierunku w którym poleci strzała oraz oddania samego strzału.
Klawiatura służy do przemieszczania się - klawisze W,S,A,D - oraz do ataku mieczem - spacja.
Ponadto, w celu przejścia do trybu obserwatora należy kliknąć lewy przycisk myszy w dowolnym miejscu ekranu.

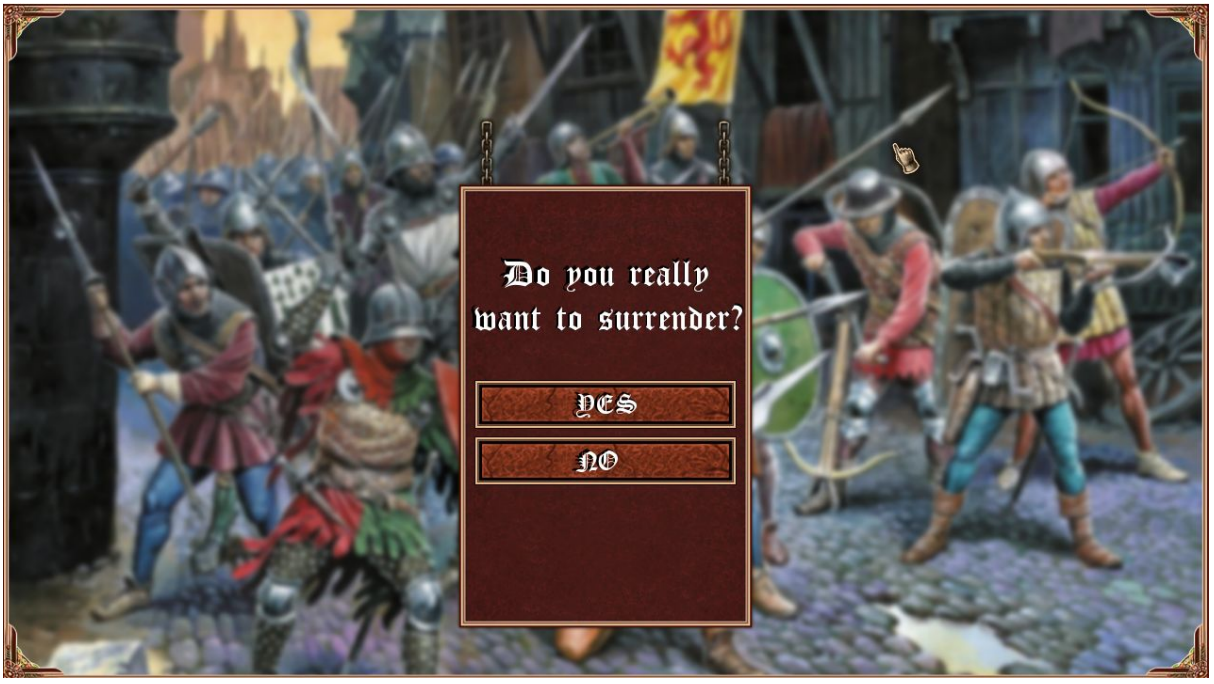
## 3.3 Zrzuty ekranu



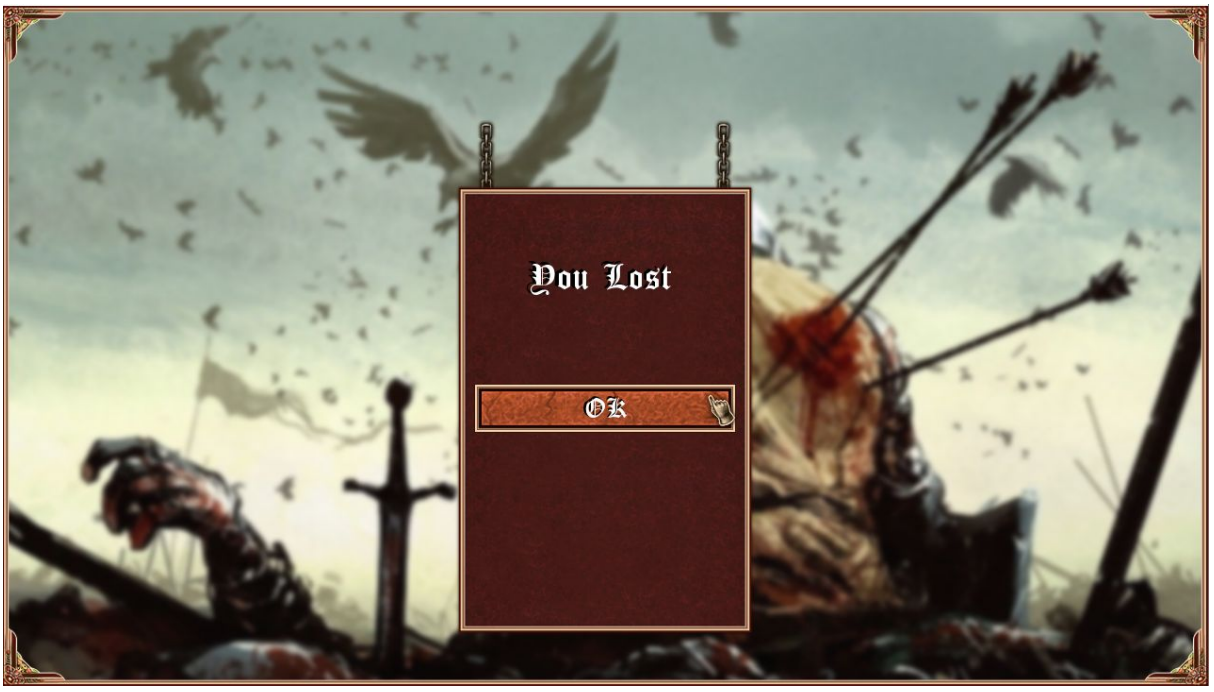

*Rys.1 Rozpoczęcie rozgrywki - po lewej widoczne podpowiedzi dotyczące sterowania.*



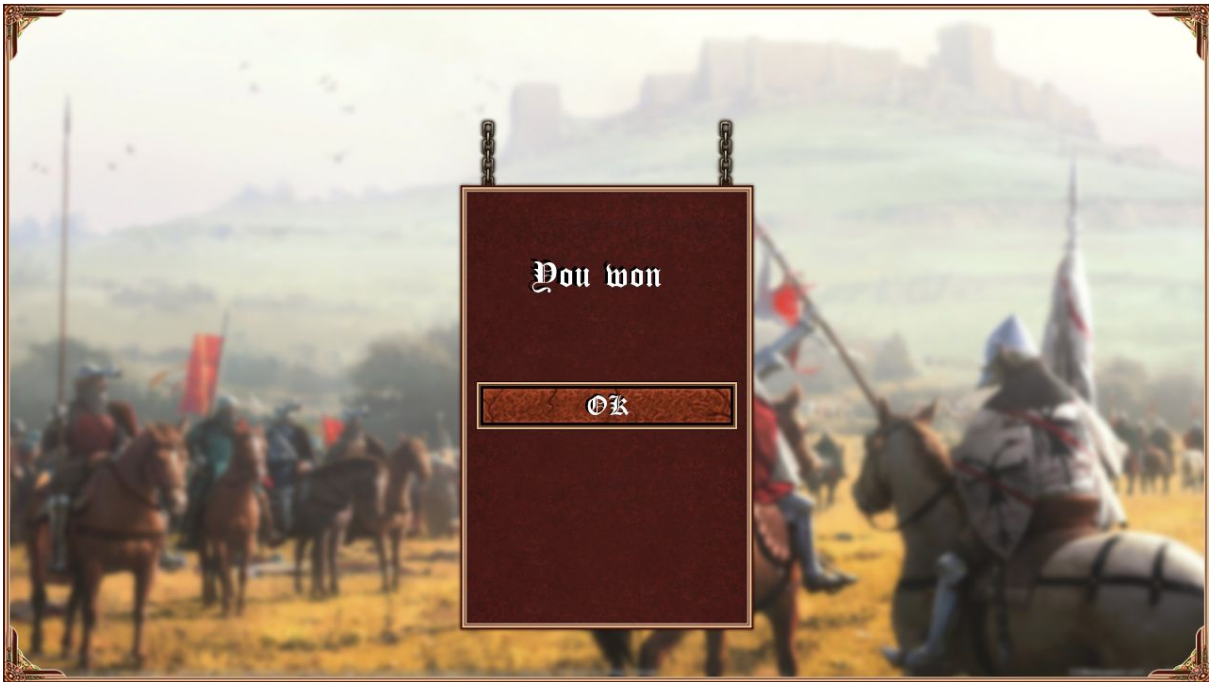
*Rys.2 Środkowa faza rozgrywki - postacie podążają do swoich przeciwników.*




*Rys.3 Menu pauzy - możliwość powrotu do gry lub poddanie walki.*




*Rys.4 Ekran symbolizujący porażkę.*




*Rys.4 Ekran symbolizujący zwycięstwo.*

# 4. Realizacja tematów z ćwiczeń laboratoryjnych

## 4.1 Wyjątki

Mechanizm wyjątków odpowiada za sprawdzenie czy udało się otworzyć plik zawierający mapę. Został wykorzystany w klasie TileMap w metodzie _loadFromFile.

```
try {
    if (!map_file.is_open() || !flora_file.is_open() || !objects_file.is_open())
        throw "Could not open file";
}
catch (std::string& except)
{
    std::cerr << except;
}
```

## 4.2 Inteligentne wskaźniki

Inteligentne wskaźniki zostały wykorzystane w wielu miejscach w projekcie. Przykładowo klasa GUI odpowiedzialna za interfejs:

```
class GUI
{
private:
    sf::RenderWindow* _window;
    std::unique_ptr<Cursor> _cursor;
    std::shared_ptr<Button> _button_yes, _button_no, _button_ok;
    std::unique_ptr<WindowBorder> _window_border;
    std::unique_ptr<Panel> _panel;
    std::unique_ptr<Menu> _menu;
    std::unique_ptr<Message> _message_menu, _message_lost, _message_won;
    std::shared_ptr<BookButton> _book;
    RiD::AssetManager _asset_manager;
    sf::Sprite _menu_background, _lost_background, _won_background;
    sf::View _camera, _gui;
    sf::Event _event;
    bool _show_left_panel;
```

```
GUI::GUI(sf::RenderWindow& window) :
    _window(&window), _show_left_panel(true)
{
    _window_border = std::make_unique<WindowBorder>();
    _menu = std::make_unique<Menu>();
    _cursor = std::make_unique<Cursor>();
    _panel = std::make_unique<Panel>();
    _book = std::make_shared<BookButton>();
    _button_yes = std::make_shared<Button>("YES");
    _button_no = std::make_shared<Button>("NO");
    _button_ok = std::make_shared<Button>(" OK");
    _message_menu = std::make_unique<Message>("  Do you really \nwant to surrender?", 40);
    _message_lost = std::make_unique<Message>("    You Lost", 40);
    _message_won = std::make_unique<Message>("    You won", 40);
```

## 4.3 Kontenery STL

Została wykorzystana wcześniej wspomniana lista (std::list) do przechowywaniu wszystkich postaci. Pojawia się w konstruktorze klasy RealTimeBattle.

```
std::list<std::shared_ptr<Character>> _list_of_enemies;
std::list<std::shared_ptr<Character>> _list_of_allies;
```

### 4.4 Iteratory STL

Skorzystano z iteratora w celu przechodzenia przez w/w listę.

```
for (std::list<std::shared_ptr<Character>>::iterator iterator = _list_of_enemies.begin(); iterator != _list_of_enemies.end(); iterator++)
{
    if ((*iterator)->isAlive())
    {
        (*iterator)->update(_clock.getElapsedTime(), _tile_map->getCollidableObjects(), _list_of_allies, *_window);
        (*iterator)->dealDamage(_clock.getElapsedTime(), _list_of_allies, *_window);
        (*iterator)->render(*_window);
    }
}
```

# 5. Wnioski

Realizacja projektu pozwoliła bliżej zapoznać się z działaniem biblioteki SFML oraz z bardziej zaawansowanymi mechanizmami języka C++.

Najbardziej złożonym oraz najbardziej problematycznym aspektem projektu było stworzenie generatora trasy dla botów. Został on zbudowany w oparciu o algorytm A*.

Kolejnym wymagającym problemem okazała się konwersja koordynatów poszczególnych elementów z systemu kartezjańskiego na rzut izometryczny. O zamianie współrzędnych należało pamiętać w przypadku renderowania każdego obiektu. Brak konwersji prowadził do błędów.

# 6. Źródła

W projekcie wykorzystano różnego rodzaju tekstury oraz audio, do których linki zamieszczone są poniżej.
Postacie:
http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/
Elementy mapy:
https://devilsworkshop.itch.io/big-pixel-isometric-block-pack-free-2d-sprites
Muzyka:
https://www.youtube.com/watch?v=S33ykouh0kI

# 7. Szczegółowy opis typów

Diagramy oraz opisy metod zostały stworzone przy użyciu narzędzia Doxygen.