**WYŻSZA SZKOŁA TECHNOLOGII INFORMATYCZNYCH W KATOWICACH**

**WYDZIAŁ INFORMATYKI**

**KIERUNEK: INFROMATYKA**

**Leon Raczyński**

**Nr albumu 8233**

**Studia stacjonarne**

**PROJEKT I IMPLEMENTACJA GRY PRZEGLĄDARKOWEJ Z WYKORZYSTANIEM TECHNOLOGII WEBOWYCH**

**Praca dyplomowa napisana**

**pod kierunkiem**

**mgr inż. Łukasza Iwanickiego  
  
w roku akademickim 2023/2024**

**Katowice 2024**

Spis treści

[WSTĘP 4](#_Toc159754464)

[ROŹDZIAŁ 1 ANALIZA WYMAGAŃ 5](#_Toc159754465)

[1.1. Określenie celów projektu 5](#_Toc159754466)

[1.1.1. Zaprojektowanie i implementacja 5](#_Toc159754467)

[1.1.2. Wykorzystanie technologii webowych 5](#_Toc159754468)

[1.1.3. Tworzenie grafiki i dźwięku 5](#_Toc159754469)

[1.1.4. Dostępność i użyteczność 5](#_Toc159754470)

[1.2. Wymagania funkcjonalne i niefunkcjonalne 5](#_Toc159754471)

[1.2.1. Wymagania funkcjonalne 5](#_Toc159754472)

[1.2.2. Wymagania niefunkcjonalne 8](#_Toc159754473)

[ROŹDZIAŁ 2 PROJEKTOWANIE GRY 9](#_Toc159754474)

[2.1. Koncepcja gry 9](#_Toc159754475)

[2.2. Projekt interfejsu użytkownika 9](#_Toc159754476)

[2.2.1. Ekran logowania 9](#_Toc159754477)

[2.2.2. Ekran główny gry 10](#_Toc159754478)

[2.2.3. Ekwipunek 10](#_Toc159754479)

[2.2.4. Menu gry 11](#_Toc159754480)

[2.3. Projekt tabel bazy danych wraz z relacjami 11](#_Toc159754481)

[ROŹDZIAŁ 3 WYBÓR NARZĘDZI PRACY 12](#_Toc159754482)

[3.1. Środowisko programistyczne 12](#_Toc159754483)

[3.2. Środowisko uruchomieniowe 12](#_Toc159754484)

[3.3. Serwer 12](#_Toc159754485)

[3.4. System zarządzania bazą danych 12](#_Toc159754486)

[3.5. Tworzenie i edycja elementów graficznych 12](#_Toc159754487)

[3.6. Edycja elementów dźwiękowych 12](#_Toc159754488)

[ROŹDZIAŁ 4 STRUKTURA BAZY DANYCH 13](#_Toc159754489)

[4.1. Przeznaczenie bazy danych 13](#_Toc159754490)

[4.2. Tabele bazy danych 13](#_Toc159754491)

[4.2.1. Tabela user 13](#_Toc159754492)

[4.2.2. Tabela hero 14](#_Toc159754493)

[4.2.3. Tabela inventory 14](#_Toc159754494)

[4.2.4. Tabela items 15](#_Toc159754495)

[4.3. Model logiczny 17](#_Toc159754496)

[4.4. Model relacyjny 17](#_Toc159754497)

[ROŹDZIAŁ 5 IMPLEMENTACJA GRY 18](#_Toc159754498)

[5.1. Struktura projektu 18](#_Toc159754499)

[5.2. Implementacja serwera 18](#_Toc159754500)

[5.2.1. Instalacja modułów 18](#_Toc159754501)

[5.2.2. Budowa serwera 18](#_Toc159754502)

[5.3. Implementacja bazy danych 20](#_Toc159754503)

[5.3.1. Instalacja oprogramowania 20](#_Toc159754504)

[5.3.2. Utworzenie bazy danych wraz z tabelami 20](#_Toc159754505)

[5.3.3. Wdrożenie do serwera 24](#_Toc159754506)

[5.4. Implementacja aplikacji webowej 24](#_Toc159754507)

[5.4.1. Budowa aplikacji 24](#_Toc159754508)

[ROŹDZIAŁ 6 FUNKCJONALNOŚCI 27](#_Toc159754509)

[6.1. Panel logowania/rejestracji 27](#_Toc159754510)

[6.1.1. Logowanie 28](#_Toc159754511)

[6.1.2. Rejestracja 28](#_Toc159754512)

# WSTĘP

Celem niniejszej pracy jest zaprojektowanie i implementacja gry przeglądarkowej w technologiach webowych.

W dobie rozwijających się różnego rodzaju silników do gier komputerowych, konsolowych i mobilnych, gry przeglądarkowe stają się coraz mniej popularne. Motywacją do zrealizowania tematu pracy jest stworzenie gry, która nie wymaga od użytkownika instalacji jakiegokolwiek dodatkowego oprogramowania, a dostępna jest bezpośrednio z poziomu przeglądarki internetowej. Dodatkowo oparta jest tylko i wyłącznie o technologie takie jak HTML, CSS i JavaScript. Aspekt ten daje możliwość twórcy swobodnej realizacji zamysłów projektowych bez konieczności trzymania się ściśle określonych i narzuconych szablonów oraz opcji przez silniki do tworzenia gier.

Ponadto, rozwijanie gry przeglądarkowej pozwala na praktyczne zastosowanie wielu aspektów inżynierii oprogramowania, takich jak projektowanie interfejsu użytkownika, zarządzanie bazą danych, czy też implementacji logiki gry.

W ramach pracy zostanie przedstawiony proces tworzenia gry, od projektowania interfejsu użytkownika, przez zarządzanie bazą danych, aż po implementację całości.

# ANALIZA WYMAGAŃ

## Określenie celów projektu

W celu uzyskania efektywniejszej realizacji zamysłu projektowego zostały nakreślone następujące cele projektowe.

### Zaprojektowanie i implementacja

Głównym celem projektu jest stworzenie gry przeglądarkowej, która będzie atrakcyjna dla potencjalnego gracza.

### Wykorzystanie technologii webowych

Projekt wykorzystuje takie technologie jak HTML5, CSS3, JavaScript oraz Node.js.

### Tworzenie grafiki i dźwięku

Projekt zawiera (czy powinien zawierać?) atrakcyjne elementy graficzne typu pixel art w perspektywie 2D oraz dźwiękowe, które przyciągną użytkownika i zwiększą jego zainteresowanie.

### Dostępność i użyteczność

Finalnie, gra powinna być łatwa w obsłudze i dostępna dla szerokiego grona użytkowników, niezależnie od ich doświadczenia z grami przeglądarkowymi.

## Wymagania funkcjonalne i niefunkcjonalne

W projekcie powinny być zrealizowane następujące wymagania funkcjonalne oraz niefunkcjonalne.

### Wymagania funkcjonalne

**Tabela 1**. Wymaganie funkcjonalne: Rejestracja

|  |  |
| --- | --- |
| Nr. wymagania | 1 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Rejestracja |
| Opis | Użytkownik chcąc zacząć nową grę zakłada konto wpisując imię dla swojego bohatera oraz hasło. |
| Uzasadnienie | Użytkownik powinien mieć możliwość założenia nowego konta w celu rozpoczęcia rozgrywki od nowa. |

**Tabela 2.** Wymaganie funkcjonalne: Logowanie

|  |  |
| --- | --- |
| Nr. wymagania | 2 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Logowanie |
| Opis | Użytkownik chcąc zalogować się na swoje wcześniej utworzone konto wpisuje imię swojego bohatera i hasło. |
| Uzasadnienie | Użytkownik powinien mieć możliwość zalogowania na swoje konto w celu kontynuacji gry od ostatniego etapu w którym zostały zapisane jego postępy. |

**Tabela 3**. Wymaganie funkcjonalne: Sterowanie

|  |  |
| --- | --- |
| Nr. wymagania | 3 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Sterowanie |
| Opis | Gracz ma możliwość sterowania swoim bohaterem (tj. poruszanie się oraz atak). |
| Uzasadnienie | W celu poruszania się po mapie i interakcji z przeciwnikiem, gracz ma powinien mieć możliwość sterowania postacią za pomocą odpowiednich klawiszy. |

**Tabela 4.** Wymaganie funkcjonalne: Zarządzanie ekwipunkiem

|  |  |
| --- | --- |
| Nr. wymagania | 4 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Zarządzanie ekwipunkiem |
| Opis | Gracz ma możliwość zarządzania przedmiotami w znajdującymi się w ekwipunku swojego bohatera. Przekładania ich z miejsca na miejsca, ubierania oraz zdejmowania z ciała postaci. |
| Uzasadnienie | W celu polepszenia statystyk bohatera, gracz powinien mieć możliwość zakładania przedmiotów na swoją postać. Otwarcie ekwipunku powinno odbywać się za pomocą wciśnięcia odpowiedniego klawisza. |

**Tabela 5.** Wymaganie funkcjonalne: Interakcja z bohaterem niezależnym

|  |  |
| --- | --- |
| Nr. wymagania | 5 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Interakcja z bohaterem niezależnym |
| Opis | Gracz ma możliwość interakcji z postaciami niezależnymi w celu prowadzenia dialogów. |
| Uzasadnienie | Gracz powinien mieć możliwość prowadzenia dialogu z postaciami niezależnymi co skutkuje przywróceniem życia głównego bohatera, czy też zakończeniem etapu zadania w zależności od konkretnego *NPC*. Interakcja powinna odbywać się poprzez wciśnięcie odpowiedniego klawisza. |

**Tabela 6.** Wymaganie funkcjonalne: Pominięcie dialogu

|  |  |
| --- | --- |
| Nr. wymagania | 6 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Pominięcie dialogu |
| Opis | Gracz ma możliwość pominięcia aktualnie wyświetlanego dialogu prowadzonego z bohaterem niezależnym. |
| Uzasadnienie | W momencie gdy gracz przeczyta aktualny dialog lub wyrazi chęć jego pominięcia, powinien mieć możliwość wykonania tej czynności. Pominięcie dialogu powinno odbywać się poprzez wciśnięcie odpowiedniego klawisza. |

**Tabela 7**. Wymaganie funkcjonalne: Otwarcie okna menu

|  |  |
| --- | --- |
| Nr. wymagania | 7 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Otwarcie okna menu |
| Opis | Gracz ma możliwość otwarcia okna menu z opcjami „Wczytaj ostatni zapis” i „Wyloguj się” w celu zrealizowania którejś z tych pozycji. |
| Uzasadnienie | W celu uzyskania dostępu do opcji „Wczytaj ostatni zapis” i „Wyloguj się” gracz powinien mieć możliwość otwarcia okna menu. |

**Tabela 8.** Wymaganie funkcjonalne: Wczytanie ostatniego zapisu

|  |  |
| --- | --- |
| Nr. wymagania | 8 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Wczytanie ostatniego zapisu |
| Opis | Gracz ma możliwość wczytania stanu gry z ostatniego punktu kontrolnego w którym gra została zapisana. |
| Uzasadnienie | Gracz powinien mieć możliwość cofnięcia się do stanu gry z ostatniego zapisu. Wybranie tej opcji powinno odbywać się poprzez wybranie LPM odpowiedniej pozycji w menu. |

Tabela 9. Wymaganie funkcjonalne: Wylogowywanie

|  |  |
| --- | --- |
| Nr. wymagania | 9 |
| Typ wymagania | Funkcjonalne |
| Przypadek użycia | Wylogowywanie |
| Opis | Gracz ma możliwość wylogowania się z poziomu gry. Czynność ta jest zwieńczona powrotem do ekranu logowania. |
| Uzasadnienie | Gracz powinien mieć możliwość wylogowanie się w celu zmiany konta. |

### Wymagania niefunkcjonalne

Tabela 10. Wymaganie niefunkcjonalne: Wspierany system operacyjny

|  |  |
| --- | --- |
| Nr. wymagania | 10 |
| Typ wymagania | Niefunkcjonalne |
| Przypadek użycia | Wspierany system operacyjny |
| Opis | Serwer na którym działa gra uruchamia się na urządzeniach z systemem Windows. |
| Uzasadnienie | System Windows jest najpopularniejszym systemem operacyjnym na komputery personalne. Serwer powinien pracować z tym systemem. |

Tabela 11. Wymaganie niefunkcjonalne: Wspierane przeglądarki internetowe

|  |  |
| --- | --- |
| Nr. wymagania | 11 |
| Typ wymagania | Niefunkcjonalne |
| Przypadek użycia | Wspierane przeglądarki internetowe |
| Opis | Gra uruchamia się w popularnych przeglądarkach internetowych tj. Microsoft Edge, Google Chrome, Mozilla Firefox i Opera. |
| Uzasadnienie | Gra powinna się uruchamiać w najbardziej popularnych przeglądarkach internetowych, aby zwiększyć liczbę odbiorców. |

Tabela 12. Wymaganie niefunkcjonalne: Wygodny interfejs

|  |  |
| --- | --- |
| Nr. wymagania | 12 |
| Typ wymagania | Niefunkcjonalne |
| Przypadek użycia | Wygodny interfejs |
| Opis | Interfejs gry jest intuicyjny, zrozumiały i łatwy w użyciu dla nowych graczy. |
| Uzasadnienie | Interfejs gry powinien być zbudowany w taki sposób aby nowi gracze oraz osoby które nie są biegłe w grach mogły z łatwością się po nim poruszać. |

# PROJEKTOWANIE GRY

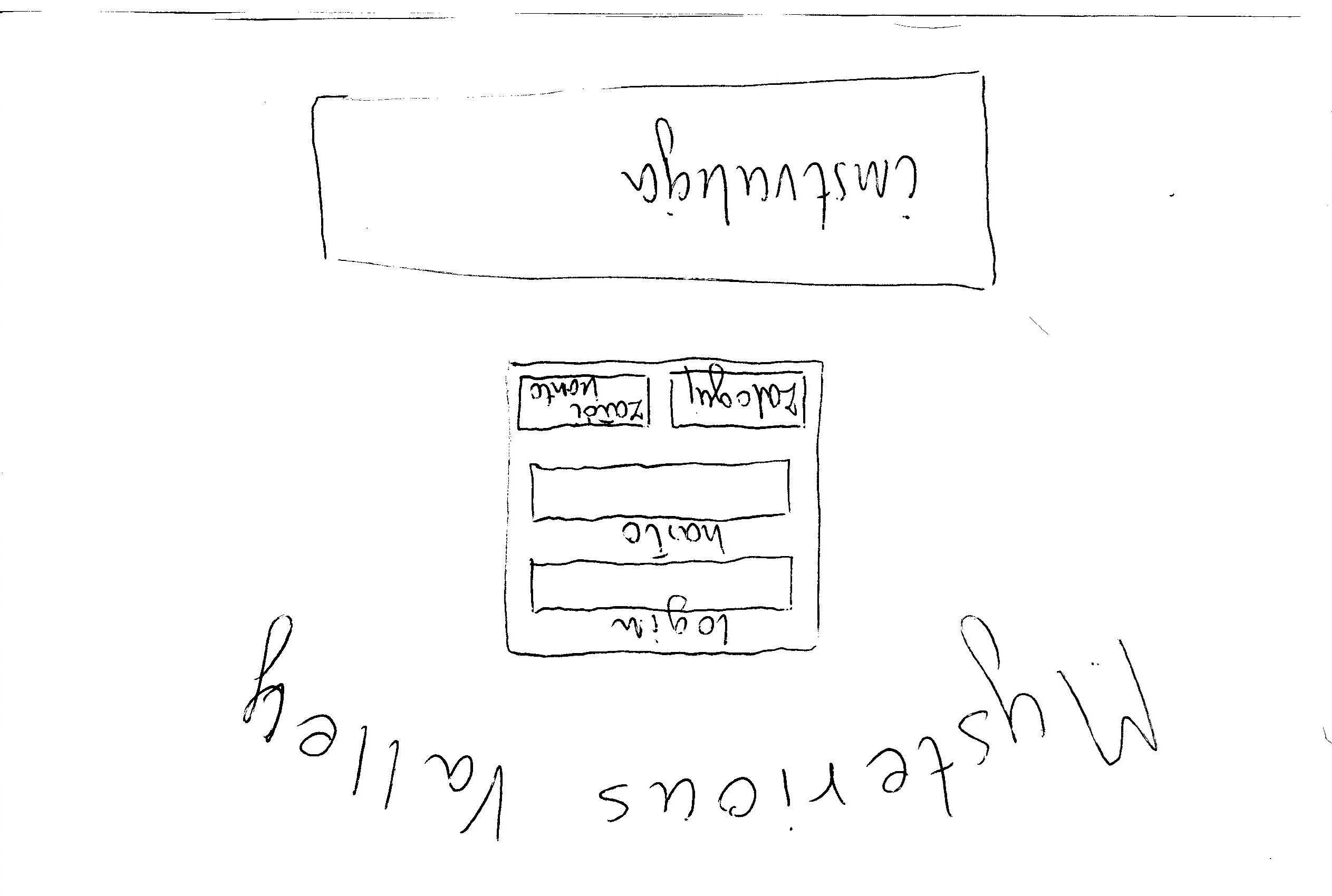
## Koncepcja gry

„Mysterious Valley” jest grą przeglądarkową o charakterze „RPG”. Gracz przenosi się do świata fantasy w którym musi sprostać wyzwaniu jakim jest pomoc mieszkańcom miasta. W osiągnięciu celu przygody, przeszkadzają mu magiczne stworzenia, które po dostrzeżeniu głównego bohatera natychmiast go atakują. W razie potrzeby na pomoc bohaterowi przyjdą mieszkańcy, którzy pomogą mu wyleczyć poniesione rany, czy też użyczą mu dodatkowego ekwipunku.

## Projekt interfejsu użytkownika

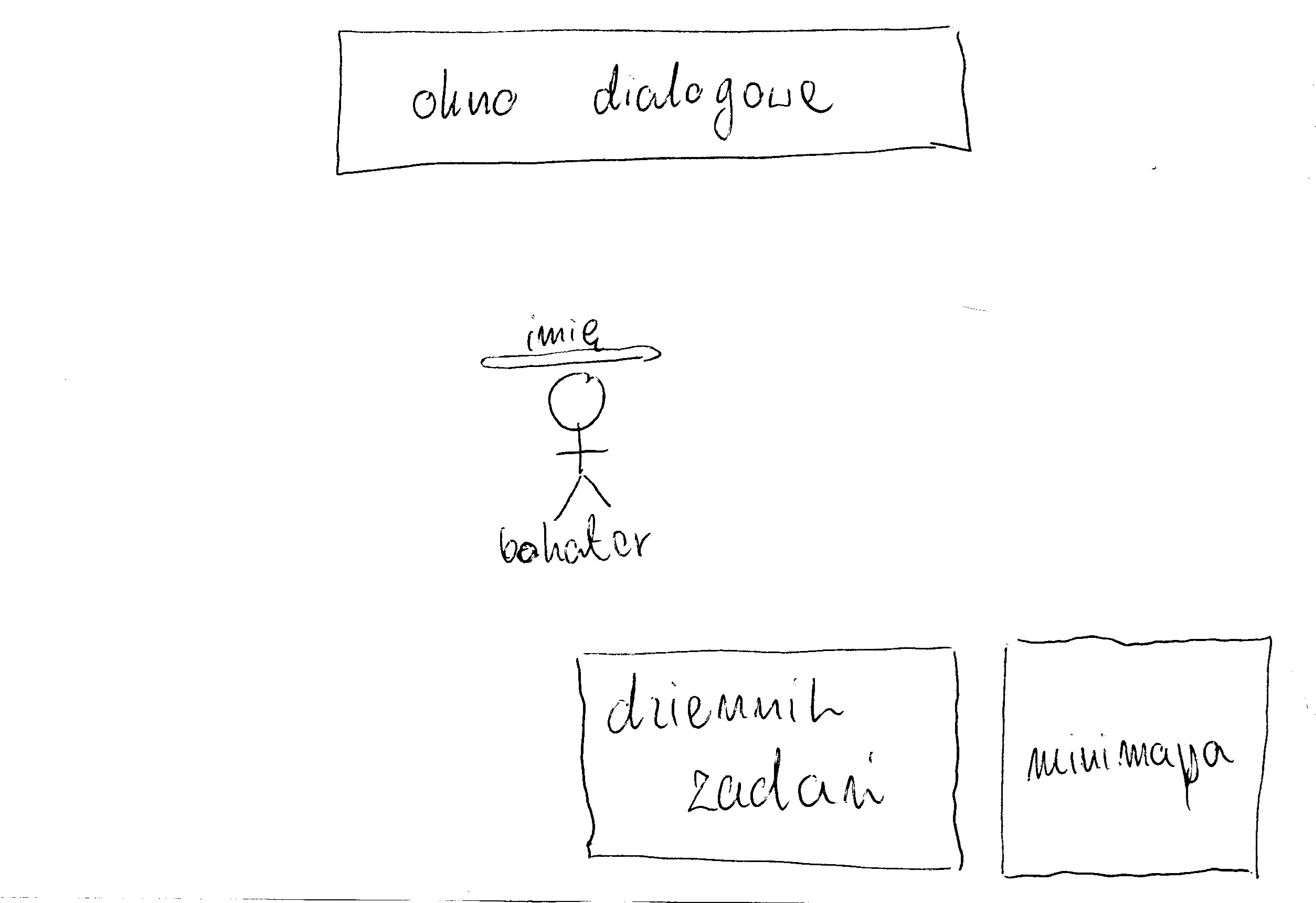
Proces tworzenia gry został poprzedzony stworzeniem projektu wizualnego poszczególnych elementów interfejsu użytkownika.

### Ekran logowania



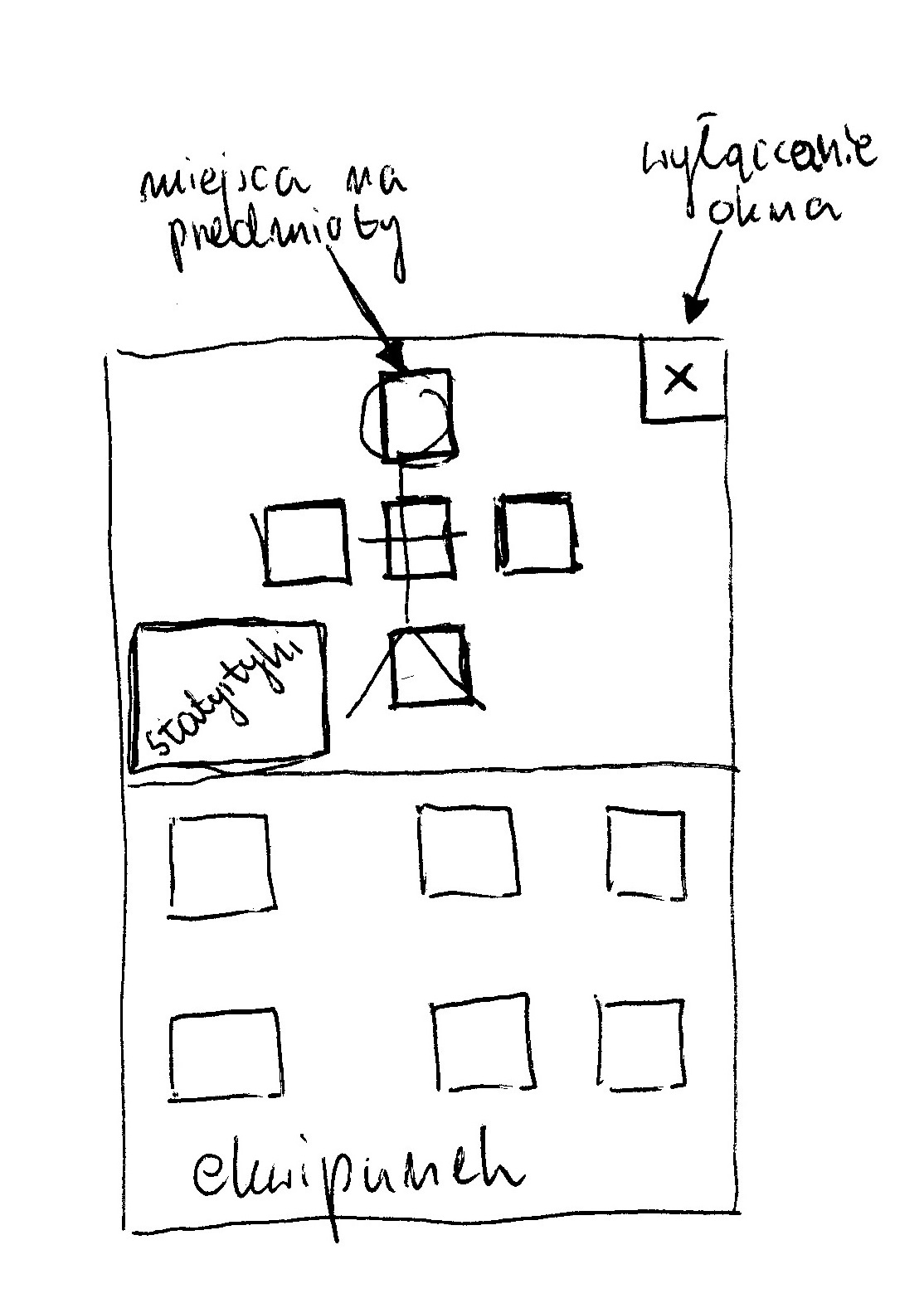
Rysunek 1. Projekt: Ekran logowania

### Ekran główny gry



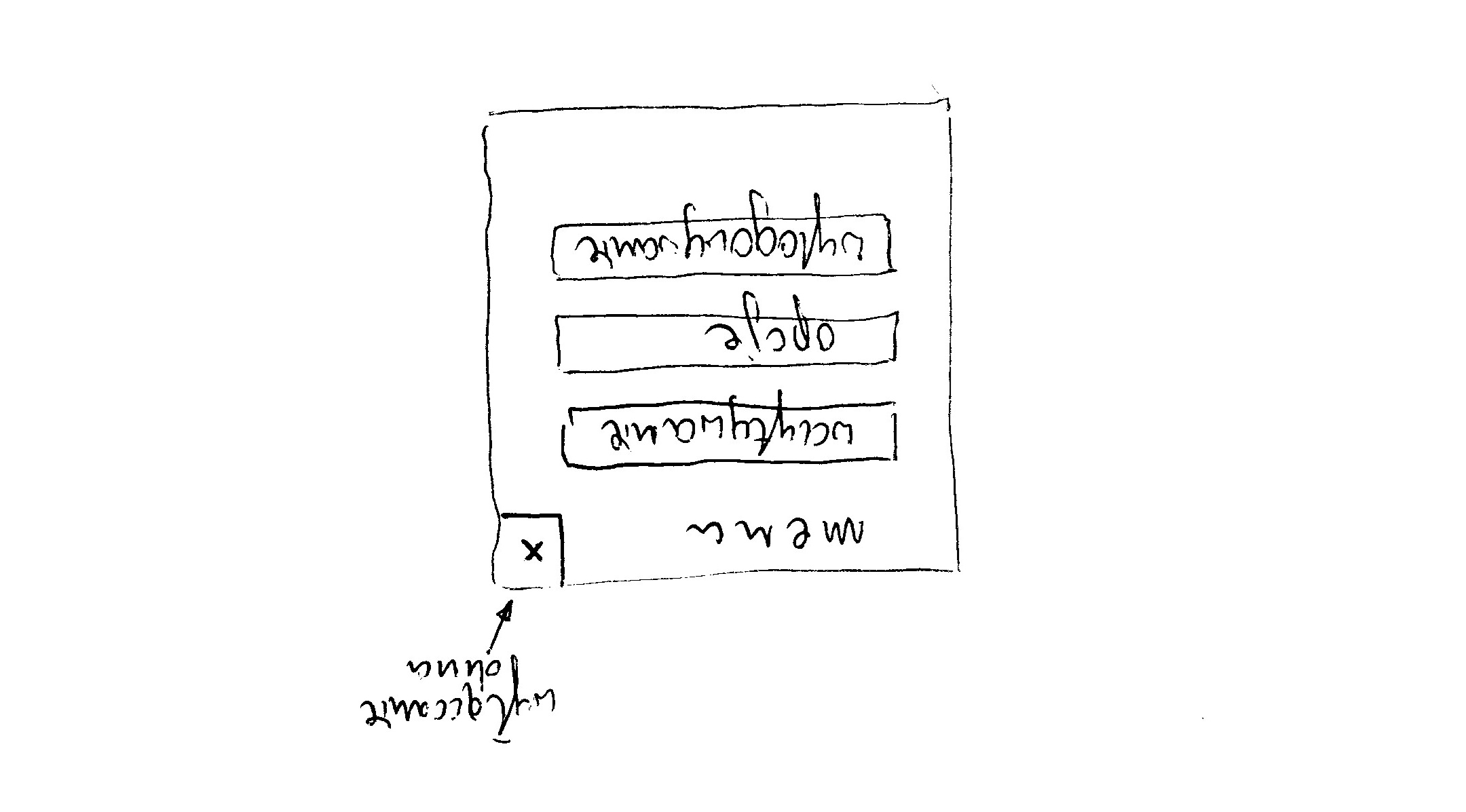
Rysunek 2. Projekt: Ekran główny gry

### Ekwipunek



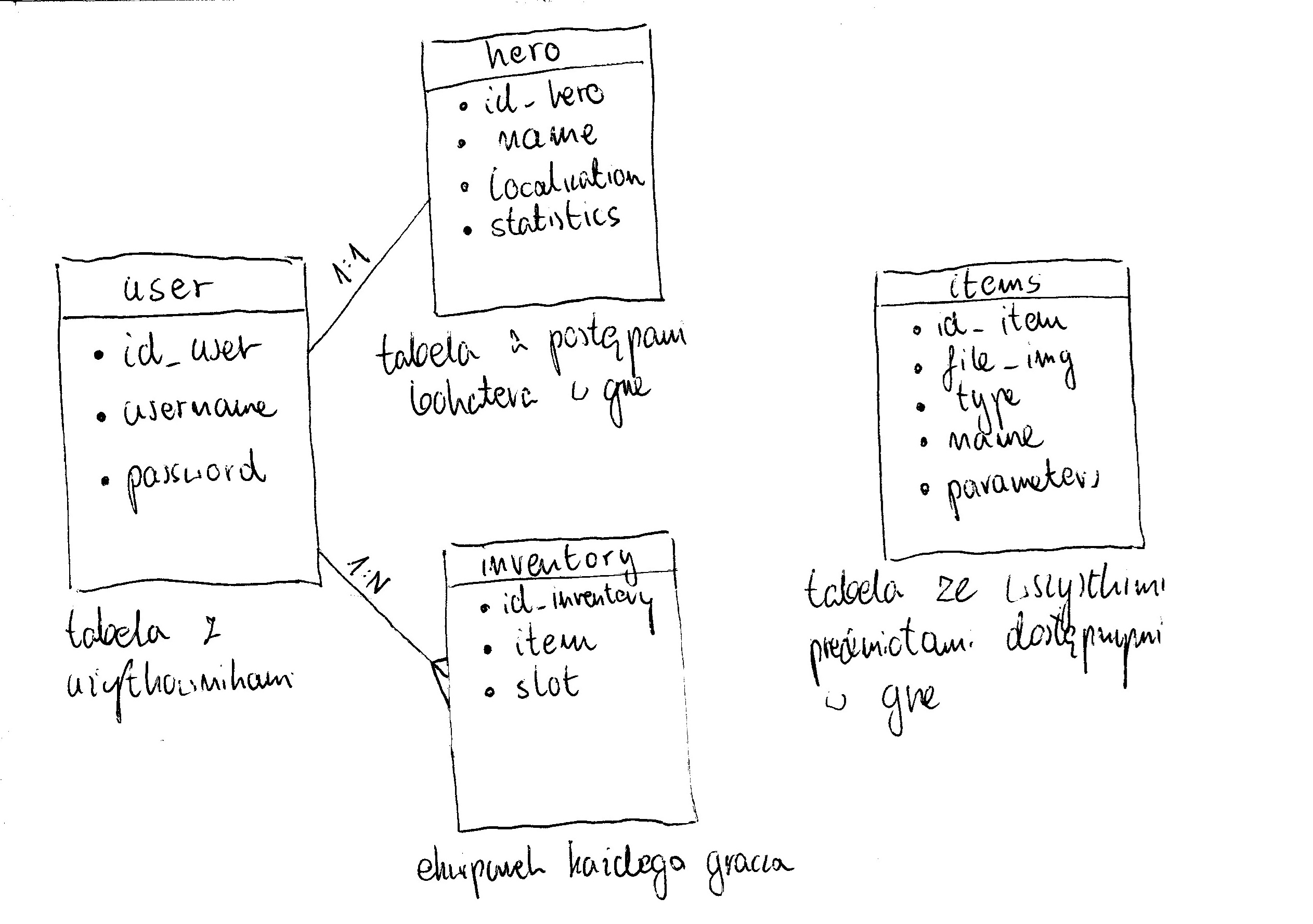
Rysunek 3. Projekt: Ekwipunek

### Menu gry



Rysunek 4. Projekt: Menu gry

## Projekt tabel bazy danych wraz z relacjami



Rysunek 5. Projekt: Baza danych

# WYBÓR NARZĘDZI PRACY

## Środowisko programistyczne

Do zrealizowania projektu, jako środowisko programistyczne został wybrany program „Visual Studio Code” firmy **Microsoft**. Program nie zajmuje dużo miejsca na dysku i nie pobiera wielu zasobów komputera podczas pracy. Ponadto użytkownik ma możliwość zainstalowania wielu dodatkowych rozszerzeń bezpośrednio z poziomu aplikacji.

## Środowisko uruchomieniowe

Aby umożliwić funkcjonowanie bazy danych w projekcie został użyty **Node.js** – środowisko uruchomieniowe. Pozwoliło to również wykorzystanie serwera

## Serwer

**JavaScript** sam w sobie nie obsługuje operacji na zewnętrznych bazach danych, dlatego w projekcie został użyty serwer **Express** oraz moduł **Express Session** w celu zastosowania sesji użytkownika.

## System zarządzania bazą danych

W celu przechowywania danych użytkownika oraz przedmiotów dostępnych w grze została zastosowana baza danych **SQLite**.

**SQLite** jest systemem zarządzania bazą danych który charakteryzuje się zajmowaniem małej ilości przestrzeni dyskowej i łatwością przenoszenia całej bazy danych. Taka baza zawarta jest tylko w jednym pliku.

Baza danych **SQLite** może być tworzona i modyfikowana poprzez okno konsolowe systemu operacyjnego na którym pracuje użytkownik. Jednakże w celu wizualizacji całego procesu w projekcie zostały użyte programy **SQLiteStudio** oraz **DB Browser for SQLite**.

## Tworzenie i edycja elementów graficznych

Jako że projekt ma na celu stworzenie gry, to elementy graficzne są równie ważne co kod obsługujący całą mechanikę. W tym celu zostały wykorzystane programy **Aseprite** i **Adobe Photoshop**.

**Aseprite** jest programem idealnie nadającym się do tworzenia i edycji grafiki typu **pixel art**. W tym właśnie stylu zaprojektowana jest gra.

**Photoshop** jest programem graficznym przeznaczonym głównie do tworzenia i obróbki grafiki rastrowej. Posłużył on do stworzenia grafiki znajdującej się na ekranie logowania.

## Edycja elementów dźwiękowych

Dźwięki są bardzo ważnym aspektem we wszystkich rodzajach gier. Powodują one integralność czynności wykonywanych przez gracza z elementami w grze, poprzez emisje ich w odpowiednich momentach i czasie.

Jako że w celach projektowych nie zostało uwzględnione tworzenie dźwięków, zostały wykorzystane gotowe już ścieżki dźwiękowe pobrane z internetu. W celu dopasowania ich do konkretnych zdarzeń w grze poddane zostały edycji (tj. czas trwania, edycja głośności w odpowiednich miejscach) w programie **Audacity**.

# STRUKTURA BAZY DANYCH

## Przeznaczenie bazy danych

W procesie tworzenia gry zawarty jest udział bazy danych. Ma ona na celu przechowywanie danych w odpowiednich tabelach o danych logowania każdego gracza, aktualnych postępach w grze, aktualnego ekwipunku z przedmiotami oraz informacji o wszystkich przedmiotach dostępnych w grze wraz z ich parametrami i obrazami.

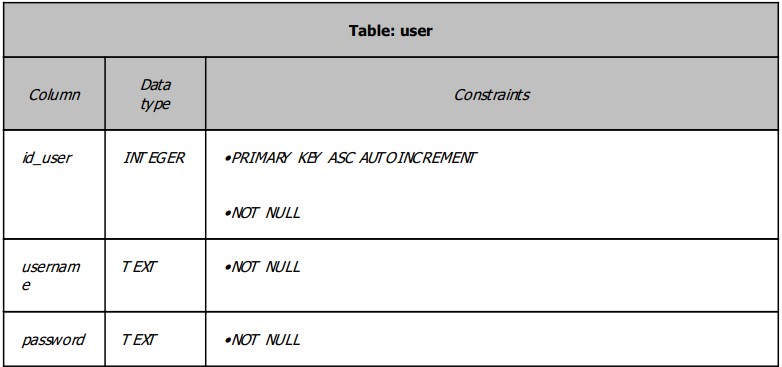
## Tabele bazy danych

Cała baza danych składa się łącznie z 4 tabel. Tabele *user*, *hero* i *inventory* są połączone ze sobą odpowiednimi relacjami.

### Tabela user

W tabeli tej przechowywane są dane o wszystkich zarejestrowany użytkownikach. Opis atrybutów:

* *id\_user* (klucz główny) – przechowywane jest tu unikalne id konta każdego użytkownika.
* *username* – nazwa użytkownika i zarazem imię bohatera w grze.
* *password* – hasło służące do logowania się na własne konto.

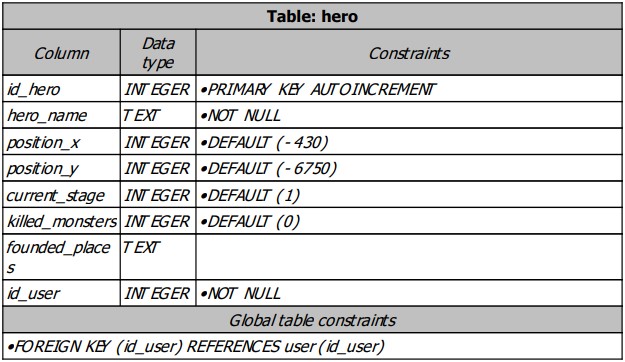


Rysunek 6. Tabela: user

### Tabela hero

Tabela zawiera informacje bohaterze głównym i jego statystykach. Opis atrybutów:

* id\_hero (klucz główny) – Unikalne id każdego bohatera głównego.
* hero\_name – Imię bohatera nadane przez użytkownika podczas rejestracji.
* position\_x – Ostatnia zapisana pozycja bohatera na osi X. Podczas tworzenia nowego konta bazowo jest ustawiona watość -430.
* position\_y – Ostatnia zapisana pozycja bohatera na osi Y. Podczas tworzenia nowego konta bazowo jest ustawiona watość -6750.
* current\_stage – Aktualnie wykonywane przez bohatera zadanie.
* killed\_monsters – Zlikwidowane potwory.
* founded\_places – Lista dotychczasowo odkrytych przez gracza miejsc na mapie.
* id\_user (klucz obcy) – Komórka stworzona, aby połączyć relacjami ze sobą tabele.

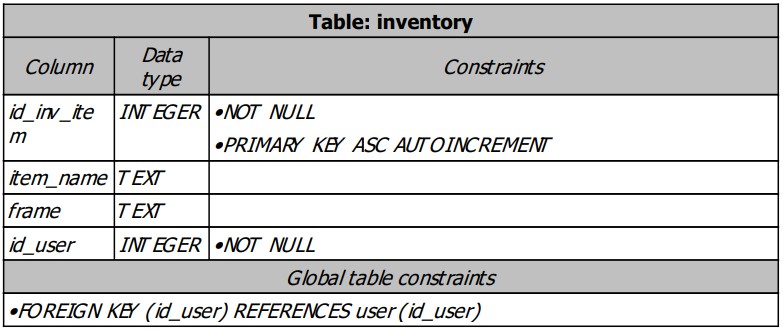


Rysunek 7. Tabela: hero

### Tabela inventory

W tej tabeli zawarte są informację o aktualnie noszonym przez bohatera ekwipunku. Podczas tworzenia nowego konta automatycznie tworzone są elementy tej tabeli w ilości odpowiadającej ilości dostępnego miejsca w ekwipunku. Podczas rozgrywki, gdy bohater otrzyma nowy przedmiot, element tabeli odpowiadający konkretnej komórce w ekwipunku jest aktualizowany. Opis atrybutów:

* **id\_inv\_item** (klucz główny) – Unikalne id każdego elementu tabeli.
* **item\_name** – Nazwa przedmiotu umiejscowionego w tej komórce w ekwipunku.
* **frame** – Konkretna komórka w ekwipunku.
* **id\_user** (klucz obcy) – Komórka stworzona, aby połączyć relacjami ze sobą tabele.

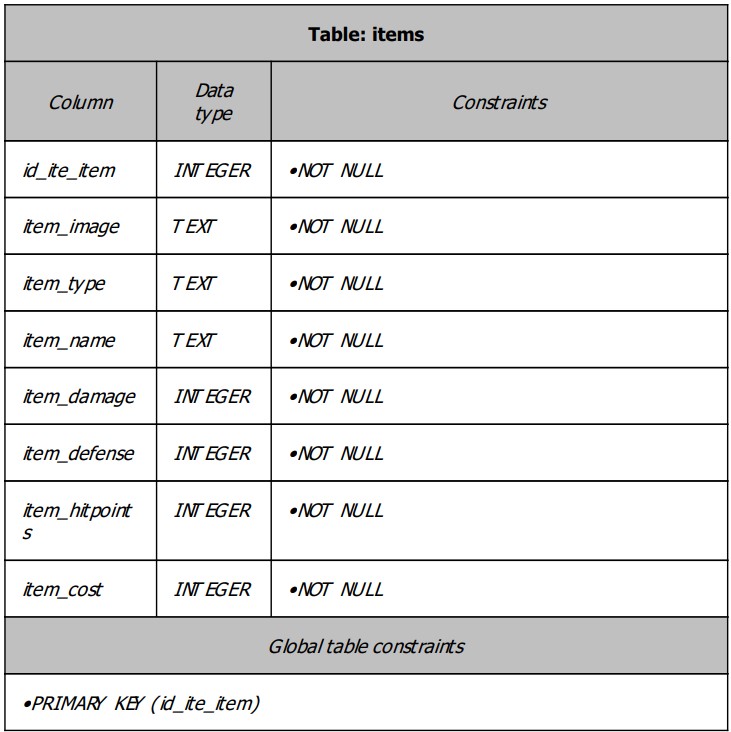


Rysunek 8. Tabela: invenotry

### Tabela items

Tabela przechowuje informacje o wszystkich przedmiotach, które bohater może zdobyć podczas przygody. Zawarte są również tutaj parametry każdego przedmiotu wraz z jego plikiem graficznym zapisanym w postaci kodu Base64.

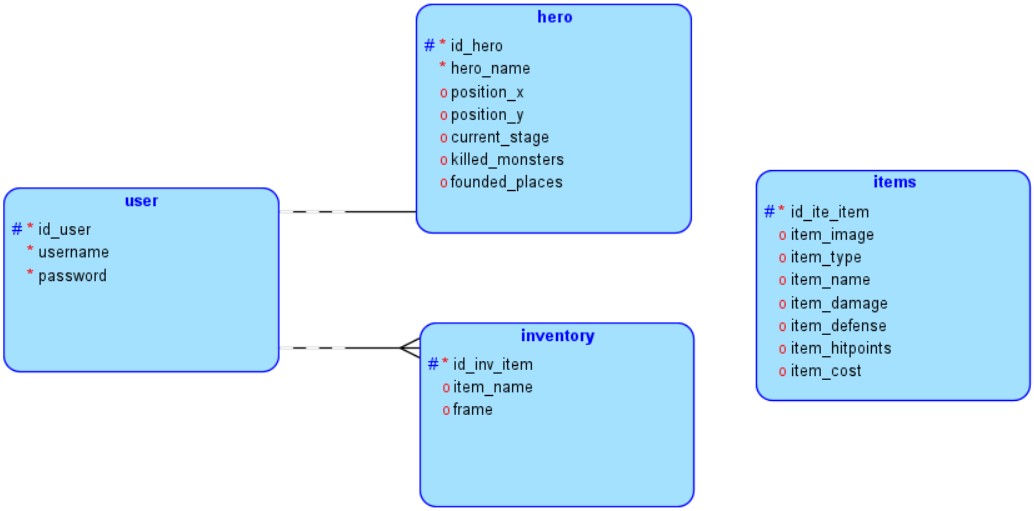
* **id\_ite\_item** (klucz główny) – Unikalne id każdego przedmiotu.
* **item\_image** – Atrybut przechowuje obraz przedmiotu zakodowany w Base64.
* **item\_type** – Typ przedmiotu. W grze prawie każdy przedmiot można założyć na bohatera. Typ przedmiotu pozwala uniknąć sytuacji w której gracz np. wkłada miecz w miejsce które przeznaczone jest do noszenia hełmu.
* **item\_name** – Nazwa danego przedmiotu.
* **item\_damage** – Parametr, który określa ile obrażeń zadaje dany przedmiot. Przedmioty, które nie są broniami nie maja w tym miejscu wartość 0.
* **item\_defense** – Określa liczbę punktów obrony, którą bohater otrzymuje po założeniu przedmiotu z tym parametrem. Analogicznie do poprzedniej pozycji – przedmioty, które są broniami mają tą wartość ustawioną na 0.
* **item\_hitpoints** – W zależności o wartości tego parametru, bohater po założeniu przedmiotu z tym atrybutem otrzymuje konkretną wartość dodatkowych punktów życia.
* **item\_cost** – Informacja o wartości danego przedmiotu. Parametr utworzony w celu ewentualnej, dalszej ewaluacji projektu.



Rysunek 9. Tabela: items

## Model logiczny

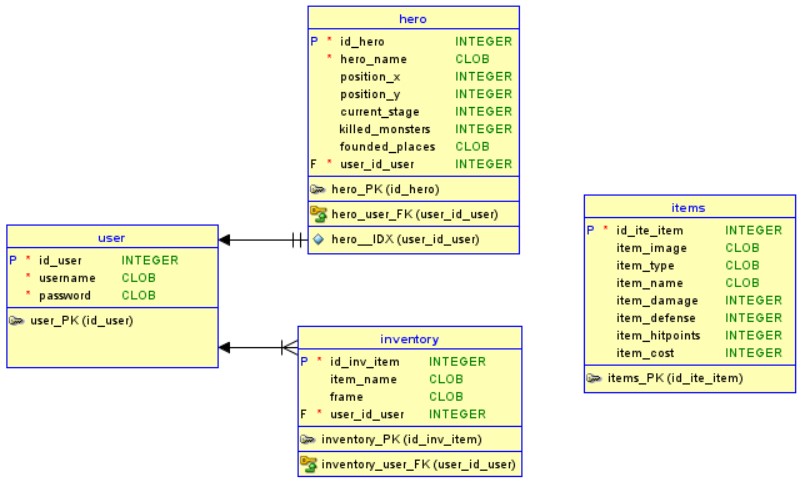
Model logiczny bazy danych stworzony w programie **SQL Developer Data Modeler** firmy **Oracle**.



Rysunek 10. Model logiczny bazy danych

## Model relacyjny

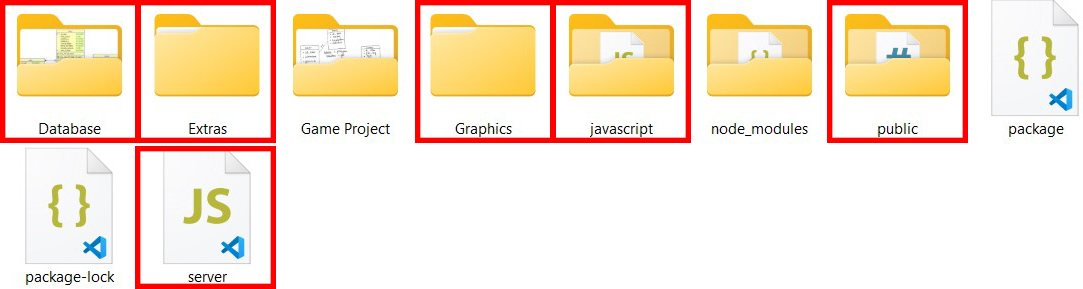
Model relacyjny bazy danych stworzony w programie **SQL Developer Data Modeler** firmy **Oracle**.



Rysunek 11. Model relacyjny bazy danych

# IMPLEMENTACJA GRY

## Struktura projektu

Projekt oparty jest o środowisko uruchomieniowe oraz serwer w którym są ściśle określone ścieżki dostępu do poszczególnych folderów w lokalizacji. Zachowanie więc odpowiedniej struktury i nazewnictwa folderów i plików zaznaczonych na poniższym zdjęciu jest niezbędne.

Rysunek 12. Struktura folderu głównego projektu

* folder główny projektu

## Implementacja serwera

Cały proces implementacji serwera składa się z kilku etapów. Każdy z nich jest opisany poniżej.

### Instalacja modułów

Korzystanie z serwera **Express** w projekcie **JavaScript** wymaga instalacji odpowiedniego modułu. Proces ten odbywa się poprzez otworzenie konsoli/terminala systemowego w głównym folderze projektu i wykonanie polecenia:

* + - ***npm install express***

W rezultacie menadżer pakietów środowiska **Node.js** pobierze i zainstaluje żądany moduł w lokalizacji z projektem.

Oprócz samego serwera potrzebna jest również obsługa sesji użytkownika, ścieżek dostępu do plików i folderów oraz bazy danych **SQLite**. Niezbędna jest więc instalacja odpowiednich pakietów za to odpowiadających:

* + - ***npm install express-session***
    - ***npm install url***
    - ***npm install path***
    - ***npm install sqlite3***

### Budowa serwera

Kod serwera można podzielić na kilka części. W pierwszej znajdują się polecenia importujące potrzebne do działania moduły.

import express from 'express';

import session from 'express-session';

import { fileURLToPath } from 'url';

import path from 'path';

import sqlite3 from 'sqlite3';

Blok kodu 1. Budowa serwera: Importowanie modułów

Kolejna część zawiera polecenia pobierające nazwę pliku i ścieżkę do lokalizacji w której element się znajduje. Zdefiniowany jest tu również serwer Express, wraz z jego konfiguracją, port na którym ma być uruchamiany oraz zmienna służąca w późniejszym etapie do przechowywania bazy danych.

const \_\_filename = fileURLToPath(import.meta.url);

const \_\_dirname = path.dirname(\_\_filename);

const databasePath = path.join(\_\_dirname,'Database','gamebase.db');

const app = express();

const port = process.env.PORT || 3000;

let db = null;

Blok kodu 2. Budowa serwera: Konfiguracja serwera

Mając zdefiniowany serwer, należy skonfigurować sesję dla użytkownika w następujący sposób.

app.use(session({

  secret: 'session-secret-key',

  resave: false,

  saveUninitialized: true

}));

Blok kodu 3. Budowa serwera: Konfiguracja sesji użytkownika

Aby serwer brał pod uwagę zawartość folderów w projekcie, niezbędnym jest zdefiniowanie ścieżek do ich lokalizacji.

app.use(express.json());

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/Extras/Cursors', express.static(path.join(\_\_dirname, '/Extras/Cursors')));

app.use('/Extras/Fonts', express.static(path.join(\_\_dirname, '/Extras/Fonts')));

app.use('/Extras/Sounds', express.static(path.join(\_\_dirname, '/Extras/Sounds')));

app.use('/Graphics/GUI', express.static(path.join(\_\_dirname, '/Graphics/GUI')));

app.use('/Graphics/Heroes', express.static(path.join(\_\_dirname, '/Graphics/Heroes')));

app.use('/Graphics/NPCs', express.static(path.join(\_\_dirname, '/Graphics/NPCs')));

app.use('/Graphics/Maps', express.static(path.join(\_\_dirname, '/Graphics/Maps')));

app.use('/Graphics/Monsters', express.static(path.join(\_\_dirname, '/Graphics/Monsters')));

app.use('/Graphics/Weapons', express.static(path.join(\_\_dirname, '/Graphics/Weapons')));

Blok kodu 4. Budowa serwera: Ustawienie ścieżek do plików statycznych

W środkowej części pliku wykonywane są metody *HTTP* dostarczone przez interfejs *REST API*. Poniższy rysunek ukazuje niektóre z żądań *GET*, które umożliwiają wykonywanie plików **JavaScript** przez serwer.

app.get('/', (req, res) => {

  res.sendFile(path.join(\_\_dirname, 'public', 'login.html'));

});

app.get('/login', (req, res) => {

  res.sendFile(path.join(\_\_dirname, 'public', 'login.html'));

});

app.get('/game', (req, res) => {

  res.sendFile(path.join(\_\_dirname, 'public', 'game.html'));

});

app.get('/login.mjs', (req, res) => {

  res.sendFile(path.join(\_\_dirname, 'javascript', 'login.mjs'));

});

app.get('/skrypt.mjs', (req, res) => {

  res.sendFile(path.join(\_\_dirname, 'javascript', 'skrypt.mjs'));

});

app.get('/progress.mjs', (req, res) => {

  res.sendFile(path.join(\_\_dirname, 'javascript', 'progress.mjs'));

});

Blok kodu 5. Budowa serwera: Metody HTTP

W celu uruchomienia komunikacji z serwerem niezbędne jest ustawienie jego *IP* wraz portem. W efekcie dodany jest również komunikat informujący użytkownika o działaniu serwera.

app.listen(port, '0.0.0.0', () => {

  console.log(`Serwer działa na porcie ${port}`);

});

Blok kodu 6. Budowa serwera: Uruchomienie serwera

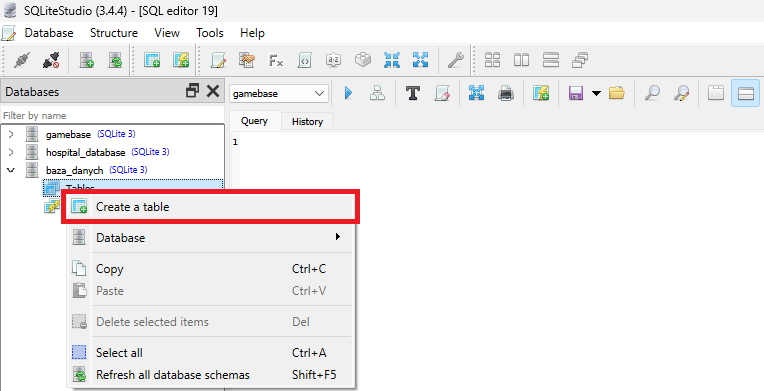
## Implementacja bazy danych

### Instalacja oprogramowania

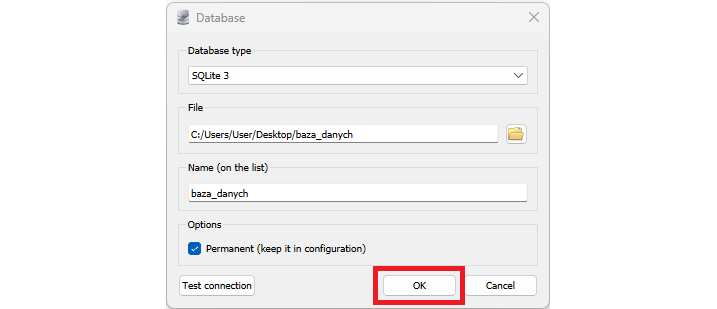
Pierwszym krokiem podczas procesu implementacji bazy danych jest zainstalowanie programów **SQLiteStudio** i **DB Browser for SQLite**.

### Utworzenie bazy danych wraz z tabelami

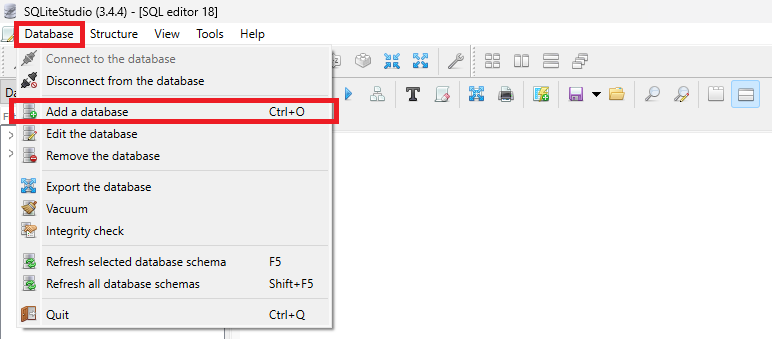
Tworzenie bazy danych wraz z tabelami odbywa się w programie **SQLiteStudio**. Cały proces pokazany jest po kolei etapami na zdjęciach poniżej.



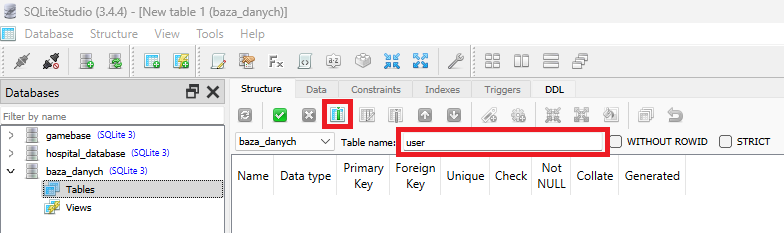
Rysunek 13. Proces tworzenia bazy danych: Dodanie nowej bazy danych 1



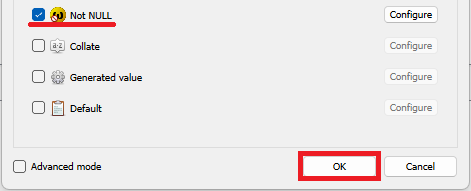
Rysunek 14. Proces tworzenia bazy danych: Dodanie nowej bazy danych 2



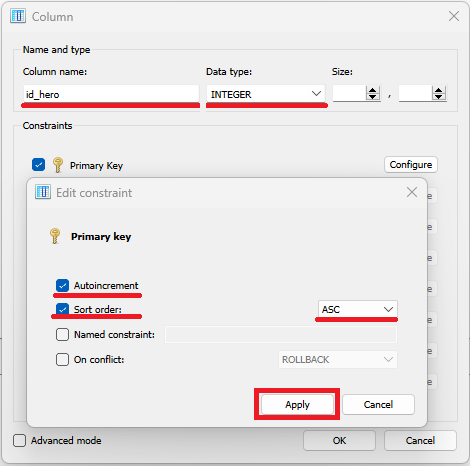
Rysunek 15. Proces tworzenia bazy danych: Dodanie nowej bazy danych 3



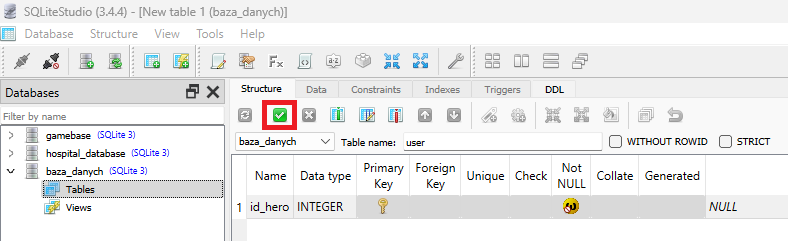
Rysunek 16. Proces tworzenia bazy danych: Dodanie nowej bazy danych 4



Rysunek 17. Proces tworzenia bazy danych: Dodanie nowej bazy danych 5



Rysunek 18. Proces tworzenia bazy danych: Dodanie nowej bazy danych 6



Rysunek 19. Proces tworzenia bazy danych: Dodanie nowej bazy danych 6

Powyższe kroki pokazują proces tworzenia nowej bazy danych i tabeli wraz z jedną kolumną o nazwie **id\_hero**. Analogicznie etapy te zostały powtórzone w celu utworzenia wszystkich tabel i kolumn tak jak zostało to przedstawione w projekcie bazy danych (rys.5). Kod SQL potrzebny do utworzenia tabel dla projektu:

CREATE TABLE hero (

    id\_hero         INTEGER PRIMARY KEY AUTOINCREMENT,

    hero\_name       TEXT    NOT NULL,

    position\_x      INTEGER DEFAULT ( -430),

    position\_y      INTEGER DEFAULT ( -6750),

    current\_stage   INTEGER DEFAULT (1),

    killed\_monsters INTEGER DEFAULT (0),

    founded\_places  TEXT,

    id\_user         INTEGER NOT NULL,

    FOREIGN KEY (

        id\_user

    )

    REFERENCES user (id\_user)

);

CREATE TABLE inventory (

    id\_inv\_item INTEGER   NOT NULL

                          PRIMARY KEY ASC AUTOINCREMENT,

    item\_name   TEXT (25),

    frame       TEXT (10),

    id\_user     INTEGER   NOT NULL,

    FOREIGN KEY (

        id\_user

    )

    REFERENCES user (id\_user)

);

CREATE TABLE items (

    id\_ite\_item    INTEGER   NOT NULL,

    item\_image     TEXT (15) NOT NULL,

    item\_type      TEXT (10) NOT NULL,

    item\_name      TEXT (25) NOT NULL,

    item\_damage    INTEGER   NOT NULL,

    item\_defense   INTEGER   NOT NULL,

    item\_hitpoints INTEGER   NOT NULL,

    item\_cost      INTEGER   NOT NULL,

    CONSTRAINT id\_ite\_item PRIMARY KEY (

        id\_ite\_item

    )

);

CREATE TABLE user (

id\_user  INTEGER   PRIMARY KEY ASC AUTOINCREMENT

                   NOT NULL,

    username TEXT (15) NOT NULL,

    password TEXT (15) NOT NULL

);

Blok kodu 7. Tworzenie tabel

### Wdrożenie do serwera

W momencie, gdy baza danych została stworzona i znajduje się w odpowiednim katalogu głównej lokalizacji projektu, następuje proces podłączenia jej do serwera. Operacja odbywa się poprzez dodanie nowego żądania z połączeniem z bazą danych. Wykonanie żądania skutkuje również otrzymaniem informacji zwrotnej o pomyślnym (lub nie) połączeniu z bazą. Żądanie wykonywane jest zawsze podczas wczytywania strony z logowaniem i samą grą.

app.get('/game/database/connect', (req, res) => {

  db = new sqlite3.Database(databasePath, (err) => {

    if (err) {

      res.status(500).json({ message: err.message });

    } else {

      res.json({ message: 'Connected to the database successfully!' });

    }

  });

});

Blok kodu 8. Żądanie połączenia z bazą danych

## Implementacja aplikacji webowej

### Budowa aplikacji

Aplikacja składa się z wielu plików zależnych od siebie, dlatego idealnym sposobem na przedstawienie całej struktury są diagramy poniżej.

Rysunek 20. Struktura folderu głównego 1

Rysunek 21. Struktura folderu głównego 2

Wszystkie pliki z językiem skryptowym JavaScript zapisane są z rozszerzeniem **.mjs**. Rozszerzenie to jest niezbędne, aby zapewnić integralność plików ze środowiskiem uruchomieniowym **Node.js** oraz umożliwić operacje **import/export**. Pochodzi ono z modułu **ECMAScript**. Pierwotnie Node.js obsługiwał moduł **CommonJS**, jednakże podczas ciągłej ewaluacji środowiska, dodano również wsparcie dla plików bazujących na module **ECMAScript**. Opis plików znajdujących się w projekcie:

* **server.mjs** – Kod źródłowy serwera.
* **game.html** – Kod strony internetowej w której uruchamiana jest gra. Zapisany w języku **HTML.**
* **login.html** – Kod strony internetowej w której uruchamiany jest formularz logowania oraz rejestracji. Zapisany w języku **HTML**.
* **login-styles.css** – Kod służący do stylizacji elementów strony internetowej **login.html** zapisany w języku **CSS**.
* **styles.css** - Kod służący do stylizacji elementów strony internetowej **game.html** zapisany w języku **CSS**.
* **characters.mjs** – Plik z kodem **JavaScript**. Znajdują się tu klasy opisujące bohatera, postacie niezależne oraz potwory wraz z metodami oraz funkcje niezależne.
* **collisions.mjs** – Zawarta jest tutaj klasa opisująca element kolizyjny wraz z metodą i funkcjami niezależnymi służącymi m.in. do tworzenia nowych obiektów (elementów kolizyjnych) i obliczania odległości bohatera lub potwora od najbliższej przeszkody.
* **config.mjs** – Plik konfiguracyjny w którym znajdują się stałe opisujące różne elementy gry. Wartości są eksportowane do konkretnych plików, gdzie są wykorzystywane.
* **images.mjs** – Znajdują się w nim stałe do których przypisane są obiekty **Image** oraz funkcje przypisujące do nich konkretne pliki graficzne. Elementy te są również eksportowane do innych plików w projekcie.
* **inventory.mjs** – Zdefiniowane są tutaj klasy opisujące miejsce w ekwipunku oraz przedmioty wraz z ich metodami, funkcje niezależne powiązane z ekwipunkiem bohatera i tworzeniem obiektów.
* **login.mjs** – Kod JavaScript obsługujący stronę logowania i wszystkie elementy, które się na niej znajdują.
* **progress.mjs** – Kod, który odpowiada za obsługę mechanizmu odkrywania nowych lokacji i wykonywania zadań.
* **requests.mjs** – Zdefiniowane i eksportowane są tutaj funkcje asynchroniczne wysyłające żądania do serwera.
* **skrypt.mjs** – Główny skrypt aplikacji odpowiadający za uruchomienie i sprawdzenie poprawności poszczególnych funkcji. Jest wykonywany w pierwszej kolejności, podczas ładowania strony z grą.
* **sounds.mjs** – W kodzie zdefiniowane i eksportowane są obiekty audio oraz ich konfiguracja.
* **utilities.mjs** – W pliku znajdują się funkcje użytkowe, wykorzystywane podczas działania aplikacji.
* **charactersMap.mjs** – Plik z którego eksportowana jest tablica z danymi o lokalizacji umieszczonych na mapie postaci niezależnych.
* **collisionsMap.mjs** - Plik z którego eksportowana jest tablica z danymi o lokalizacji umieszczonych na mapie elementów kolizyjnych.
* **dialogues.json** – Znajdują się tutaj treści dialogów między bohaterem a postaciami niezależnymi wraz informacjami o zadaniach w których występują.
* **locationsMap.mjs** - Plik z którego eksportowana jest tablica z danymi o lokalizacji umieszczonych na mapie lokacji do odkrycia.
* **monstersMap.mjs** - Plik z którego eksportowana jest tablica z danymi o lokalizacji miejsc występowania potworów.
* **quests.json** – Plik z treścią dostępnych zadań w grze. Każde zadanie zapisane jest w postaci obiektu i zawiera informacje o jego zrealizowaniu.

# FUNKCJONALNOŚCI

## Panel logowania/rejestracji

Pierwszą funkcjonalnością aplikacji, na którą natrafia użytkownik jest panel logowania.



Rysunek 22. Panel logowania

### Logowanie

Proces logowania obsługuje metoda **addEventListener()** interfejsu **EventTarget**. Celem zdarzenia typu **click** jest przycisk **loginButton**. Po interakcji użytkownika z tym elementem wykonywana jest funkcja asynchroniczna w której przeprowadzana jest walidacja wprowadzonych danych oraz wysyłanie żądania do serwera.

loginButton.addEventListener('click', async function() {

    sound.CLICK\_SOUND.play();

    const username = document.getElementById('username').value;

    const password = document.getElementById('password').value;

    if (username.length < 3 || password.length < 3) {

        const message = `Nazwa użytkownika i/lub hasło jest za krótkie`;

        return displayErrorMessage(message);

    }

    if (!await request.loginRequest(username, password)) {

        const message = `Nieprawidłowa nazwa użytkownika lub hasło!`;

        return displayErrorMessage(message);

    }

    loadGame();

})

Blok kodu 9. Proces logowania

Po pomyślnej weryfikacji danych logowania, użytkownik przekierowywany jest do strony z grą. Przekierowanie obsługuje funkcja **loadGame()**.

function loadGame() {

    return window.location.assign('http://192.168.1.138:3000/game');

}

Blok kodu 10. Przekierowanie do strony z grą

### Rejestracja

Rejestracja nowego użytkownika przebiega w bardzo podobny sposób co logowanie. Po wciśnięciu przycisku **„Nowa gra”,** wysyłane jest żądanie do serwera w celu sprawdzenia czy użytkownik już istnieje w bazie danych. Jeżeli odpowiedź nie jest negatywna, po stronie serwera wykonywana jest operacja dodawania nowego użytkownika wraz z tworzeniem nowego bohatera i jego ekwipunku startowego.

registerButton.addEventListener('click', async function() {

    sound.CLICK\_SOUND.play();

    const username = document.getElementById('username').value;

    const password = document.getElementById('password').value;

    if (username.length < 3 || password.length < 3) {

        const message = `Nazwa użytkownika i/lub hasło jest za krótkie`;

        return displayErrorMessage(message);

    }

    if (!await request.registerRequest(username, password)) {

        const message = `Bohater o podanym imieniu już istnieje!`;

        return displayErrorMessage(message);

    }

    if (!await request.createHeroRequest()) {

        const message = `Błąd podczas tworzenia bohatera!`;

        return displayErrorMessage(message);

    }

    for (let i = 0; i < 14; i++) {

        const frame = 'frame' + i;

        if (!await request.createInventoryRequest(frame)) {

            const message = `Błąd podczas tworzenia ekwipunku!`;

            return displayErrorMessage(message);

        }

    }

    loadGame();

})

Blok kodu 11. Proces rejestracji

Pomyślne wykonanie operacji kończy się przekierowaniem użytkownika do strony z grą.

## Gra

Funkcjonalności stricte związane z samą rozgrywką dotyczą głównie interakcji z ekwipunkiem, menu, działania minimapy, dziennika zadań, mechaniki związanej z ruchem postaci, interakcji z obiektami, systemem walki, wykonywania zadań i odkrywania nowych lokacji.

### Ekwipunek

Otwarcie ekwipunku bohatera odbywa się poprzez wciśnięcie klawisza „I”. Kod klawisza przechwytywany jest dzięki zdarzeniu **onkeydown** połączonym ze zdarzeniem **KeyboardEvent.KeyCode**.

this.document.onkeydown = async function(e) {

        switch (e.keyCode) {

            case 73:

                if (e.repeat) {

                    break;

                }

                inventoryButton.style.filter = "contrast(50%)";

                drawInventory();

                break;

}}

Blok kodu 12. Otwieranie ekwipunku

Następnie wykonywana jest funkcja **drawInventory()** wyświetlająca ekwipunek. W sytuacji, gdy ekwipunek jest już otwarty, zostaje on zamknięty.

function drawInventory() {

        const inventoryWindow = document.querySelector('.inventory');

        if (isInvenoryOpen) {

            isInvenoryOpen = false;

            inventory.drawItems();

            inventoryWindow.style.display = 'none';

        } else if (!isInvenoryOpen) {

            isInvenoryOpen = true;

            inventory.writeHeroStats();

            inventory.writeHeroMoney();

            inventory.drawItems();

            inventoryWindow.style.display = 'block';

        }

}

#### Wyświetlanie szczegółów przedmiotu

Jeżeli bohater posiada jakikolwiek przedmiot na wyposażeniu, gracz może podejrzeć parametry tego przedmiotu poprzez najechanie kursorem na ramkę w której się znajduje.



Rysunek 23. Wyświetlanie szczegółów przedmiotu

Każda ramka w ekwipunku jest obiektem klasy **InvenotryFrame** i posiada zdarzenia **mouseover** i **mouseout** obsługujące ten proces.

export class InventoryFrame {

    constructor(number, x, y, item=null) {

        this.frame = document.createElement('div');

        this.frame.setAttribute('class', 'itemFrame');

        this.frame.id = 'frame' + number;

        this.frame.style.top = y + 'px';

        this.frame.style.left = x + 'px';

        this.frame.addEventListener('mouseover', this.mouseOverEffect.bind(this));

        this.frame.addEventListener('mouseout', this.mouseOutEffect.bind(this));

        this.frame.addEventListener('mousedown', this.chooseItem.bind(this));

        gameArea.appendChild(this.frame);

        this.isFrameEmpty = true;

        this.item = item;

        this.setParameters();

    }

Blok kodu 13. Klasa InventoryFrame

mouseOverEffect() {

        if (isInvenoryOpen) {

            this.frame.style.opacity = 0.2;

            onHoverFrame = this;

        }

        if (isInvenoryOpen && !this.isFrameEmpty) {

            const informationFrame = document.querySelector('.itemDetailsFrame');

            const firstTextLine = '<span style="color: #936537;">Nazwa: <span style="color: whitesmoke;">' + this.item.name + '</span> </span>';

            let secondTextLine = '';

            let thirdTextLine = '';

            let fourthTextLine = '';

            let describtionTextLine = '';

            switch (this.item.type) {

                case 'weapon':

                    secondTextLine = '<span style="color: #936537;">Obrażenia: <span style="color: whitesmoke;">' + this.item.damage + '</span> </span>';

                    describtionTextLine = '<span style="font-size: 0.7rem; color: #936537; font-style: italic;"> "Muszę uważać żeby się nie skaleczyć!" </span>';

                    break;

                default:

                    secondTextLine = '<span style="color: #936537;">Punkty życia: <span style="color: whitesmoke;">' + this.item.hitpoints + '</span> </span>';

                    thirdTextLine = '<span style="color: #936537;">Obrona: <span style="color: whitesmoke;">' + this.item.defense + '</span> </span>'

                    describtionTextLine = '<span style="font-size: 0.7rem; color: #936537; font-style: italic;"> "Dzięki temu czuję się bezpiecznie!" </span>';

                    break;

            }

            informationFrame.innerHTML = firstTextLine + secondTextLine + thirdTextLine + describtionTextLine;

            informationFrame.style.display = 'grid';

            gameArea.addEventListener('mousemove', this.moveItem.bind(this));

        }

}

Blok kodu 14. Metoda mouseOverEffect

mouseOutEffect() {

        const informationFrame = document.querySelector('.itemDetailsFrame');

        informationFrame.style.display = 'none';

        this.frame.style.opacity = 0;

        onHoverFrame = null;

}

Blok kodu 15. Metoda mouseOutEffect

#### Podnoszenie przedmiotu

Przedmiot znajdujący się w ekwipunku można przenieść w celu ubrania go na swojego bohatera lub zmiany jego miejsca. W tym celu wystarczy na niego najechać kursorem myszy i kliknąć lewy przycisk myszy. Przedmiot zostanie podniesiony.



Rysunek 24. Podnoszenie przedmiotu

Przedmiot zostanie „przyczepiony” do kursora myszy i będzie przesuwał się razem z nim do momentu umieszczania go w nowym miejscu. Zdarzenie obsługuje metoda **chooseItem()**.

chooseItem() {

        if (isInvenoryOpen) {

            if (!this.isFrameEmpty && pickedItem == null) {

                pickedItem = this.item;

                this.pickUpItem();

                if (wearableFrames.includes(this.frame.id)) {

                    calculateStats(pickedItem, 'takeoff');

                }

                this.isFrameEmpty = true;

                this.item = null;

                gameArea.addEventListener('mousemove', this.moveItem.bind(this));

            }

Blok kodu 16. Podnoszenie przedmiotu

#### Upuszczanie przedmiotu

Upuszczenie aktualnie podniesionego przedmiotu pozwala na umiejscowienie go w wybranym przez gracza okienku ekwipunku. Proces obsługuje dalszy fragment kodu z metody powyżej (Blok kodu 16).

            else if (this.isFrameEmpty && pickedItem != null && this.checkItemType(pickedItem)) {

                this.placeItem();

                onHoverFrame.setParameters();

                saveInventory(this.item.name, this.frame.id);

                pickedItem = null;

            }

            else if (!this.isFrameEmpty && pickedItem != null) {

                const inFrameItem = onHoverFrame.item;

                const itemToPlace = pickedItem;

                const itemToPlaceID = document.getElementById(pickedItem.getItemID());

                this.placeItem(itemToPlaceID);

                onHoverFrame.item = itemToPlace;

                onHoverFrame.isFrameEmpty = false;

                pickedItem = inFrameItem;

                this.pickUpItem();

                saveInventory(itemToPlace.name, this.frame.id);

            }}}

Blok kodu 17. Upuszczanie przedmiotu

#### Zakładanie przedmiotu

Niektóre z przedmiotów mogą zostać założone na bohatera. Podczas umieszczania przedmiotu w danym okienku w ekwipunku, zostaje wykonana funkcja warunkowa, sprawdzająca czy docelowe okienko jest miejscem na przedmiot ubioru lub broni. Po założeniu lub ściągnięciu przedmiotu aktualizowane są statystyki bohatera.

#### Statystyki bohatera

Statystyki bohatera obejmują kolejno punkty życia (PŻ), obrony oraz ataku. Pokazane są po lewej stronie w ekwipunku.



Rysunek 25. Statystyki