# Specyfikacja Wymagań Oprogramowania (SRS)

## 1. Wstęp

### 1.1 Cel

Celem tego dokumentu jest szczegółowe opisanie systemu oprogramowania gry 2.5D. Dokument ten obejmuje funkcjonalność systemu, ograniczenia oraz interakcje z zewnętrznymi interfejsami.

### 1.2 Zakres Projektu

Opisywany system oprogramowania to gra komputerowa rozwijana przy użyciu biblioteki Raylib. Zawiera funkcje zarządzania postaciami, śledzenia osiągnięć, zarządzania ekwipunkiem oraz różne stany gry, takie jak eksploracja, walka i zakupy.

### 1.3 Definicje, akronimy i skróty

- **SRS**: Specyfikacja Wymagań Oprogramowania
- **GUI**: Graficzny Interfejs Użytkownika
- **API**: Interfejs Programowania Aplikacji

### 1.4 Materiały uzupełniające

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications
- Dokumentacja Biblioteki Raylib

## 2. Opis ogólny

### 2.1 Perspektywa Produktu

System oprogramowania jest samodzielną aplikacją zaprojektowaną do celów rozrywkowych. Gra 2.5D wykorzystuje Raylib do renderowania grafiki i zapewnia modułową strukturę dla elementów gry.

### 2.2 Klasy oraz charakterystyki użytkowników

- **Gracze**: Użytkownicy, którzy grają dla rozrywki.
- **Programiści**: Osoby, które mogą rozszerzać lub utrzymywać kod gry.

### 2.3 Środowisko operacyjne

Oprogramowanie jest zaprojektowane do uruchamiania w środowiskach desktopowych z obsługą Raylib. Jest kompatybilne z systemami operacyjnymi Windows, macOS i Linux.

### 2.4 Ograniczenia projektu oraz implementacji

- Gra jest rozwijana przy użyciu biblioteki Raylib, która musi być wspierana na docelowej platformie.

- Struktura kodu opiera się na konwencjach języka programowania C i API Raylib.

## 2.5 Założenia i zależności

- Użytkownicy mają podstawową wiedzę na temat korzystania z aplikacji komputerowych.
- Środowisko wykonawcze musi spełniać minimalne wymagania sprzętowe i programowe do uruchomienia gry.

# 3. Funkcjonalności systemu

## 3.1 Zarządzanie postaciami

### 3.1.1 Opis

Moduł zarządzania postaciami pozwala graczom tworzyć, dostosowywać i kontrolować swoje postacie w grze. Jest to kluczowa funkcja gry.

### 3.1.2 Wymagania funkcjonalne

- System musi umożliwiać graczom tworzenie postaci z określonymi atrybutami (np. zdrowie, atak, ekwipunek).
- System musi aktualizować status postaci na podstawie rozgrywki.
- System musi umożliwiać zapisywanie i wczytywanie stanu postaci.

## 3.2 System osiągnięć

### 3.2.1 Opis

System osiągnięć śledzi postęp gracza i nagradza osiągnięcia na podstawie zdefiniowanych kryteriów, co zwiększa zaangażowanie gracza.

### 3.2.2 Wymagania funkcjonalne

- System musi śledzić osiągnięcia na podstawie działań gracza.
- System musi wyświetlać osiągnięcia graczowi w interfejsie użytkownika.

## 3.3 Zarządzanie ekwipunkiem

### 3.3.1 Opis

Moduł zarządzania ekwipunkiem pozwala graczom zarządzać ekwipunkiem swoich postaci, co jest istotne dla postępu i strategii gry.

### 3.3.2 Wymagania funkcjonalne

- System musi umożliwiać graczom zakładanie i zdejmowanie przedmiotów.
- System musi aktualizować atrybuty postaci na podstawie ekwipunku.
- System musi przechowywać informacje o ekwipunku w zapisie gry.

## 3.4 Stany gry

### 3.4.1 Opis

System obsługuje różne stany gry, takie jak eksploracja, walka, zakupy, zarządzanie ekwipunkiem, zapisywanie i wczytywanie gry.

#### 3.4.2 Wymagania funkcjonalne

- System musi umożliwiać przełączanie między różnymi stanami gry.
- System musi zapisywać i wczytywać stan gry.
- System musi obsługiwać interakcje gracza w każdym stanie gry.

# 4. Wymagania dotyczące danych

## 4.1 Logiczny model danych

Model danych obejmuje struktury dla postaci, ekwipunku, osiągnięć i stanów gry.

## 4.2 Słownik danych

**Struktura danych dla postaci**

- **Nazwa postaci**: Tekst, identyfikujący postać w grze.
- **Zdrowie**: Liczba całkowita, reprezentująca poziom zdrowia postaci.
- **Atak**: Liczba całkowita, reprezentująca siłę ataku postaci.
- **Zmęczenie**: Liczba całkowita, reprezentująca poziom zmęczenia postaci.
- **ID postaci**: Unikalny identyfikator postaci.
- **Dialog**: Identyfikator aktualnego dialogu postaci.
- **Kierunek**: Kierunek, w którym zwrócona jest postać.
- **Części ciała**: Lista części ciała postaci i ich stan.
- **Pancerz**: Lista elementów pancerza i ich stan.
- **Broń**: Typ broni posiadanej przez postać.
- **Ataki**: Lista dostępnych ataków postaci.

**Struktura danych dla ekwipunku**

- **Nazwa przedmiotu**: Tekst, identyfikujący przedmiot w ekwipunku.
- **Typ przedmiotu**: Typ przedmiotu (np. broń, pancerz, mikstura).
- **Atrybuty przedmiotu**: Atrybuty wpływające na statystyki postaci.
- **Ilość**: Liczba określająca ilość przedmiotów danego typu w ekwipunku.

## 4.3 Raporty

- System powinien generować raporty dotyczące stanu postaci i osiągnięć gracza.
- Raporty powinny być dostępne w interfejsie użytkownika oraz zapisywane w plikach zapisu.

## 4.4 Zdobywanie, integralność, przechowywanie i usuwanie danych

- Dane dotyczące postaci i stanu gry muszą być przechowywane bezpiecznie i zgodnie z najlepszymi praktykami.
- System musi zapewniać integralność danych poprzez odpowiednie mechanizmy zapisu i wczytywania.
- System musi umożliwiać usuwanie danych zapisu na żądanie użytkownika.

# 5. Wymagania interfejsów zewnętrznych

### 5.1 Interfejsy użytkownika

- Aplikacja zapewnia graficzny interfejs użytkownika (GUI) do tworzenia postaci, rozgrywki i zarządzania ekwipunkiem.
- Interfejs musi być intuicyjny i łatwy w nawigacji.

### 5.2 Interfejsy oprogramowania

- System integruje się z biblioteką Raylib do renderowania grafiki i obsługi wejścia.
- Interfejsy muszą być zgodne z API Raylib i muszą umożliwiać rozszerzanie funkcjonalności aplikacji.

### 5.3 Interfejsy sprzętowe

- System wymaga urządzeń wejściowych, takich jak klawiatura i mysz do interakcji.
- System musi być kompatybilny z typowymi urządzeniami wejściowymi obsługiwanymi przez Raylib.

### 5.4 Interfejsy komunikacyjne

- Nie dotyczy dla tej samodzielnej aplikacji, która nie wymaga komunikacji sieciowej.

## 6. Atrybuty jakościowe

### 6.1 Użyteczność

- Interfejs powinien być intuicyjny i łatwy w nawigacji.
- System powinien zapewniać dostępność funkcji dla szerokiego grona użytkowników.

### 6.2 Wydajność

- System powinien zapewniać płynne renderowanie i responsywne sterowanie.
- Aplikacja powinna działać sprawnie na docelowych platformach sprzętowych.

### 6.3 Bezpieczeństwo

- System powinien bezpiecznie zarządzać danymi zapisu, aby zapobiec ich utracie lub uszkodzeniu.
- System powinien obsługiwać błędy w sposób zapobiegający awariom.

### 6.4 Niezawodność

- Aplikacja powinna działać stabilnie podczas rozgrywki i minimalizować ryzyko awarii.
- System musi zapewniać regularne tworzenie kopii zapasowych danych zapisu.

### 6.5 Łatwość utrzymania

- Kod powinien być modułowy i dobrze udokumentowany, aby ułatwić aktualizacje i naprawę błędów.
- System powinien umożliwiać łatwe rozszerzanie funkcjonalności poprzez dodawanie nowych modułów.

### 6.6 Przenośność

- Aplikacja powinna być łatwa do przeniesienia na różne platformy wspierane przez Raylib.