

Projekt zaliczeniowy



Programowanie obiektowe
Rok akademicki 2024/2025

Autorzy:

Autor 1 - Paulina Kania

Autor 2 - Natalia Kruk

Autor 3 - Wiktoria Kowalska

DnD Unit Management

1. Opis Ogólny

Projekt **DnD Unit Management** to system wspomagający graczy Dungeons & Dragons w procesie kreacji bohaterów, formowania drużyn oraz zarządzania ich statystykami w czasie rzeczywistym. System integruje zaawansowane mechaniki RPG z trwałością danych zapewnioną przez bazę danych SQL oraz pliki JSON.

2. Podział Odpowiedzialności

- **Autor 1 (Backend & Core):** Zawartość z folderów: Classes, Core, Exceptions and Interfaces, Races, moduł usług Service (oprócz klasy PartyDbContext), obsługa wyjątków, odczyt z bazy danych, architektura projektu i obsługa pamięci podręcznej
- **Autor 2 (Frontend & UX):** Projekt i Implementacja Interfejsu graficznego (GUI) jak i jego funkcjonalność/obsługa, łączenie frontu z klasami z backendu, instrukcja i dokumentacja, README, ClassDiagram,
- **Autor 3 (Data Base & QA):** Skorzystanie z interfejsu ICloneable w klasie Character i Party oraz implementacja, stworzenie metody DeepCopy w klasie Party, implementacja wzorca Code First, stworzenie klasy PartyDbContext, zapis danych w Bazie Danych SQL, testy jednostkowe, raport projektu zaliczeniowego

3. Architektura Projektu

UnitClass:

- Klasa abstrakcyjna, która zawiera szkielet dla wszystkich klas z folderu Classes.
- Zawiera konstruktor z modyfikatorem dostępu protected, który umożliwia zapis do pliku JSON każdej z klas z folderu Classes, abstrakcyjny string ClassName (reprezentuje nazwę poszczególnej klasy postaci), abstrakcyjną listę StatPrio (reprezentuje spersonalizowaną listę statystyk dla klasy postaci), abstrakcyjny int HitDie (reprezentuje HitDie dla poszczególnej klasy postaci), abstrakcyjny bool Spell (reprezentuje to czy postać może używać zaklęć), abstrakcyjną metodę wirtualną BaseHp(ustawia BaseHp na wartość HitDie poszczególnej klasy postaci), abstrakcyjną wirtualną funkcję AssignStats (niezbędna klasie Unit do przypisywania statystyk postaci), public funkcja Calc (oblicza bonusy do statystyk postaci), wirtualna funkcja AssignStarterPack (reprezentuje ekwipunek przypisany do każdej klasy postaci)

Klasy z folderu Classes:

- Dziedziczą po klasie UnitClass
- Wywołują puste konstruktory z klasy bazowej, aby nawiązać do protected konstruktora w UnitClass
- Przesłaniają wartości z klasy bazowej takie jak: ClassName, HitDie, Spell, StatPrio oraz funkcje AssignStarterPack

UnitRace:

- Klasa abstrakcyjna, która zawiera szkielet dla wszystkich klas z folderu Races.
- Zawiera konstruktor z modyfikatorem dostępu protected, który umożliwia zapis do pliku JSON każdej z klas z folderu Races, abstrakcyjny string RaceName (reprezentuje nazwę rasy postaci), wirtualną funkcję ApplyBonus (wywoływana w klasie UnitClass w funkcji AssignStats, przypisuje bonusy do statystyk wynikające z rasy postaci)

Klasy z folderu Races:

- Dziedziczą po klasie UnitRace
- Wywołują puste konstruktory z klasy bazowej, aby nawiązać do protected konstruktora w UnitRace
- Przesłaniają wartości z klasy bazowej takie jak: RaceName oraz funkcje ApplyBonus

Unit:

- Implementuje interfejsy ILevel (zwiększa poziom postaci) i IAction (oblicza HP po otrzymaniu obrażeń lub uleczeniu postaci)

- Posiada prywatne pola deklarujące statystyki postaci jak i hermetyzacje z modyfikatorem dostępu public (przy każdej hermetyzacji występuje obsługa błędów dotyczących maksymalnej wartości statystyk), prywatne i publiczne pola UnitRace i UnitClass (niemapowane; pola prywatne są wykorzystywane do zapisu wartości do bazy danych, publiczne pobierają wartości dotyczące klasy i rasy postaci, na których później operujemy), publiczną metodę ProficiencyBonus (liczy bonus do statystyk postaci z zależności od poziomu), publiczną metodę Level (pobiera i ustawia poziom postaci), publiczne pole bool LifeStatus (informuje o tym czy postać żyje), protected konstruktor (ustawia domyślne wartości statystyk postaci), protected wirtualną funkcję DeathScreen (wyświetla komunikat o śmierci postaci).

Character:

- Dziedziczy po klasie Unit
- Implementuje interfejsy: IEquatable (sprawdza czy podana postać istnieje), IComparable (porównuje postacie po nazwie i wartości HP), ICloneable (klonuje postacie)
- Zwiera publiczne pola CharacterId, PartyId, Party (identyfikator postaci, informacja zwrotna do klasy Party), prywatne pola name i gold (informacja o nazwie postaci o liczbie złota postaci), publiczną metodę MaxxSpellCount (ustawia maksymalną ilość zaklęć postaci), publiczne i prywatne listy Spells, Proficiencies, Equipment (niemapowane; pola prywatne są wykorzystywane do zapisu wartości do bazy danych, publiczne pobierają wartości dotyczące zaklęć, bonusów do statystyk i ekwipunku postaci, na których później operujemy), hermetyzacje pól name i gold wraz z obsługą błędów, publiczny konstruktor bazowy klasy Unit, publiczny konstruktor, w którym użytkownik podaje nazwę postaci, jej rasę i klasę (przypisuje wszystkie statystyki i wartości do postaci), publiczną funkcję AddSpell (przypisuje zaklęcia do postaci), publiczną funkcję AddProficiencies (dodaje bonusy do statystyk postaci), publiczną metodę CanLearnMoreSpells odpowiada na pytanie czy postać może posiadać większą liczbę zaklęć niż na początku), publiczną metodę public int RollProficiency (losuje wartość statystyk z bonusami), przestąpienie metody ToString dla postaci, przestąpienie metody DeathScreen (wyświetla nazwę postaci, która zginęła i wartość obrażeń, którą odniosła).

Party:

- Implementuje interfejs ICloneable (klonowanie drużyny)
- Zawiera identyfikator Party i dodaje relacje 1:n dla PartyMembers
- Posiada funkcję zapisu i odczytu Party do bazy danych, publiczny pusty konstruktor i konstruktor, w którym użytkownik podaje nazwę drużyny, funkcje dodawania postaci do drużyny, sprawdzania czy podana postać jest już w tej drużynie, usuwania członków, znajdowania członków określonej rasy lub klasy,

sortowania członków po nazwie, HP, Strength, Dexterity, przesłonięcie metody ToString dla drużyny, publiczną funkcję DeepCopy

Klasy z folderu Exceptions and Interfaces:

- Umożliwiają sortowanie po odpowiednich statystykach
- Klasa InvalidStatValueException zawiera wyjątek dla nieprawidłowych statystyk

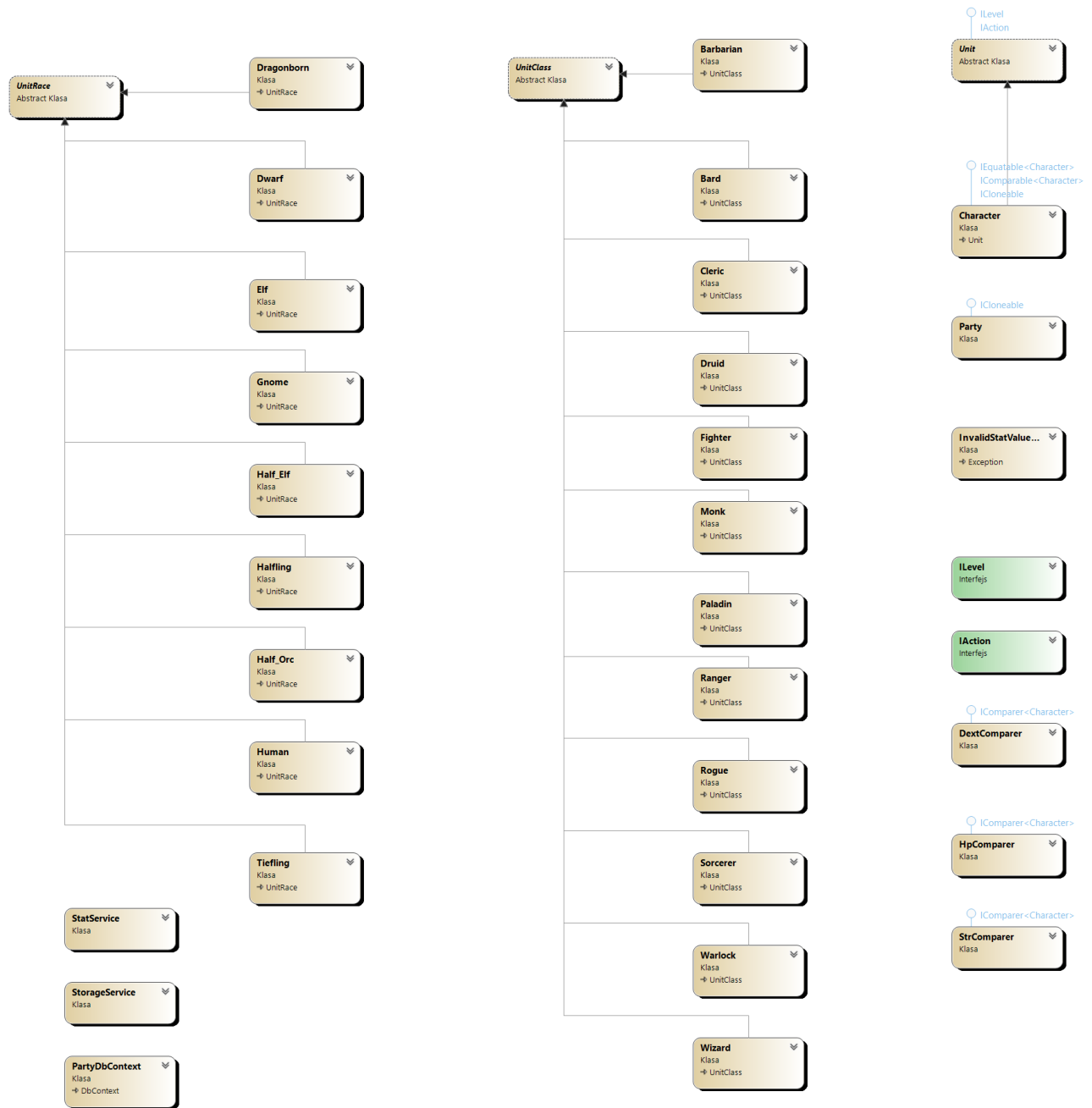
Klasy z folderu Service:

- Klasa PartyDbContext jest niezbędna do stworzenia bazy danych.
- Klasa StatService odpowiada za losowanie statystyk postaci.
- Klasa StorageService zawiera zapisywanie drużyny i postaci do plików JSON jak i odczyt.

4. Opis funkcjonalności

1. Stworzenie postaci podając nazwę postaci, jej klasę i rasę. Klasy zapewniające tę funkcjonalność to: poszczególna klasa z folderu Classes, poszczególna klasa z folderu Races, UnitClass, UnitRace, Unit, Character. Dzięki tym klasom program tworzy postać, dla której, oprócz podanych wartości, przypisywane są statystyki bazowe wraz z odpowiednimi bonusami, HP (HitPoints czyli inaczej punkty życia postaci), bazowe AC (Armor Class czyli punkty osłony postaci) oraz podstawowy ekwipunek. Dodatkowo jeśli postać (z klasy) ma do tego predyspozycję (jeśli nie użytkownik zostaje o tym poinformowany), istnieje możliwość dodania zaklęć do karty postaci.
2. Modyfikowanie postaci. Klasy zapewniające tę funkcjonalność: Unit, Character. Użytkownik jest w stanie zwiększyć poziom swojej postaci (co przekłada się na wartość HP oraz bonusów do statystyk), dodać nowe zaklęcia, uaktualnić swoje HP poprzez odniesienie obrażeń (zmniejsza punkty HP) lub uleczenie (zwiększa punkty HP).
3. Stworzenie drużyny poprzez podanie jej nazwy i dodanie członków. Klasy zapewniające tę funkcjonalność: Character, Party.
4. Zarządzanie drużyną: . Klasa zapewniająca tę funkcjonalność to Party. Użytkownik ma możliwość usunąć członka, sprawdzić czy dany gracz należy do drużyny, wyświetlić wszystkich członków drużyny o podanej klasie czy rasie, sortować członków drużyny po nazwie, HP, wartości statystyki Strength lub Dexterity.
5. Zapis jest wykonywany zarówno przez serializację JSON jak i wykorzystując Entity Framework Core do bazy danych w tabelach SQL.

5. Diagram klas



6. Wykorzystane Technologie i Biblioteki

- **Język:** C# 12.0 / .NET 8.0
- **Framework GUI:** WPF (Windows Presentation Foundation)
- **ORM:** Entity Framework Core (MS SQL Server)
- **Biblioteki zewnętrzne:**
 - System.Text.Json – do obsługi plików płaskich i serializacji list.
 - HandyControl – zaawansowane kontrolki interfejsu użytkownika (Badge, InfoBar).
 - WpfAnimatedGif – obsługa animacji kości w kreatorze.