Studia i Materiały Informatyki Stosowanej, Tom 18, Nr 1, 2026
str. 5-10

PROJEKTOWANIE I IMPLEMENTACJA GRY 2D Z TUROWYM SYSTEMEM WALKI W ŚRODOWISKU UNITY

Rafał Grabowski, Janusz Dorożyński

Uniwersytet Kazimierza Wielkiego, Wydział Informatyki
Kopernika 1, 85-074 Bydgoszcz
e-mail: rafal.grabowski@student.ukw.edu.pl

Streszczenie: *Artykuł przedstawia proces projektowania, implementacji oraz oceny jednostkowej gry 2D z turowym systemem walki, stworzonej w środowisku Unity z wykorzystaniem języka C#. Głównym celem pracy było zbudowanie kompletnego, modularnego projektu gry, integrującego kluczowe mechaniki charakterystyczne dla gatunku cRPG: eksplorację stałego świata, system rozwoju postaci oparty na ekwipunku, oraz taktyczny, turowy system walki inspirowany produkcją „Broken Ranks". W ramach badań przeprowadzono analizę funkcjonalną zaimplementowanych systemów, w tym zarządzania przedmiotami, logiki walki oraz interakcji z otoczeniem. Wykorzystano metody inżynierii oprogramowania, takie jak diagramy przypadków użycia i diagramy encji, do projektowania architektury systemu. Wyniki potwierdzają poprawność implementacji założonych mechanik oraz wykazują wysoką skalowalność i elastyczność projektu, stanowiącego solidną bazę do dalszego rozwoju. Zaproponowano również kierunki przyszłych prac, w tym rozbudowę sztucznej inteligencji przeciwników, wprowadzenie trybu wieloosobowego oraz proceduralne generowanie treści.*

Słowa kluczowe: *Unity, C#, turowy system walki (TBS), projektowanie gier, RPG, pixel art*

DESIGN AND IMPLEMENTATION OF A 2D GAME WITH A TURN-BASED COMBAT SYSTEM IN THE UNITY ENVIRONMENT

Abstract: *The article presents the process of designing, implementing, and unit-level evaluation of a 2D game with a turn-based combat system, developed in the Unity environment using the C# programming language. The main objective of the study was to build a complete, modular game project integrating key mechanics characteristic of the cRPG genre: exploration of a persistent world, an equipment-based character progression system, and a tactical, turn-based combat system inspired by the production "Broken Ranks." As part of the research, a functional analysis of the implemented systems was conducted, including item management, combat logic, and interactions with the environment. Software engineering methods, such as use case diagrams and entity diagrams, were employed to design the system architecture. The results confirm the correctness of the implemented mechanics and demonstrate high scalability and flexibility of the project, which constitutes a solid foundation for further development. Directions for future work were also proposed, including the expansion of enemy artificial intelligence, the introduction of a multiplayer mode, and procedural content generation.*

Keywords: *Unity, C#, turn-based combat system (TBS), game design, RPG, pixel art*

1. Wprowadzenie

Dynamiczny rozwój niezależnego rynku gier, oraz dostępność zaawansowanych, a jednocześnie przystępnych narzędzi deweloperskich, takich jak silnik Unity, znacząco zdemokratyzował proces tworzenia gier komputerowych. Umożliwiło to realizację ambitnych projektów przez pojedynczych twórców lub małe zespoły, łączące funkcje programisty, projektanta i grafika. Szczególnym wyzwaniem w tym kontekście jest tworzenie gier z gatunku cRPG (computer Role-Playing Game), które charakteryzują się złożonością systemową, wymagającą starannego zaprojektowania współdziałających mechanik rozgrywki, narracji oraz rozwoju postaci [2].

Głównym celem niniejszej pracy było zaprojektowanie i wdrożenie działającego prototypu gry 2D, koncentrującego się na trzech filarach: eksploracji stałego świata, pośrednim rozwoju postaci poprzez ekwipunek oraz taktycznym, turowym systemie walki. Projekt stanowi odpowiedź na potrzebę stworzenia modularnej i skalowalnej bazy kodu, umożliwiającej zarówno weryfikację założonych koncepcji projektowych, jak i stanowiącej fundament pod przyszłą rozbudowę. Wybór turowego systemu walki (ang. *Turn-Based System*, TBS) podyktowany był chęcią podkreślenia roli strategicznego myślenia i planowania akcji, co stanowi istotny element immersji w gatunku RPG.

Za podstawową hipotezę badawczą przyjęto, że wykorzystanie modularnej architektury opartej na komponentach Unity oraz obiektowym paradygmacie programowania w C# pozwoli na efektywną implementację złożonych systemów gry, zapewniając jednocześnie wysoką przejrzystość kodu, łatwość utrzymania i potencjał do przyszłej rozbudowy. Praca ma na celu zweryfikowanie tej hipotezy poprzez praktyczną realizację projektu oraz analizę powstałych artefaktów (kod źródłowy, diagramy, działający prototyp).

2. Projekt gry

2.1. Środowisko i narzędzia

Podstawowym środowiskiem implementacyjnym był silnik Unity w wersji 2022.3.13f1. Unity został wybrany ze względu na swoją wszechstronność, doskonałe wsparcie dla grafiki 2D, rozbudowany system komponentów oraz aktywną społeczność, zapewniającą dostęp do bogatych zasobów edukacyjnych. Językiem programowania logiki gry był C#, będący natywnym i optymalnym wyborem dla Unity, oferujący wydajność, silne typowanie oraz pełnię możliwości programowania obiektowego [1, 6].

Proces tworzenia assetów graficznych przeprowadzono z wykorzystaniem darmowego narzędzia Piskel, specjalizującego się w tworzeniu grafiki typu pixel art. Styl ten został wybrany ze względu na swoją nostalgiczną estetykę, niskie wymagania sprzętowe oraz stosunkową prostotę tworzenia, adekwatną dla projektu realizowanego przez jedną osobę [8].

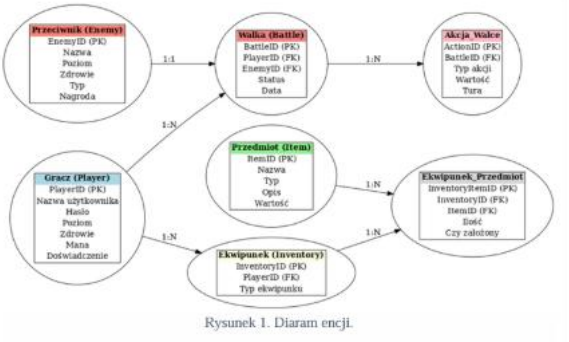
Do zarządzania kodem źródłowym i pisania skryptów wykorzystano Microsoft Visual Studio 2022, zapewniające zaawansowane funkcje takie jak IntelliSense, debugowanie zintegrowane z Unity oraz refaktoryzację [6].

2.2. Metodyka projektowania

Proces tworzenia gry poprzedziła faza projektowania koncepcyjnego i analitycznego. W celu określenia wymagań funkcjonalnych i zaprojektowania architektury systemu wykorzystano standardowe techniki inżynierii oprogramowania:

Przedstawiony na rysunku 1 Diagram przypadków użycia (Use Case Diagram) służył do zidentyfikowania głównych aktorów (gracz) oraz operacji, które system musi obsłużyć. Diagram encji (Entity-Relationship Diagram – ERD). Diagram ten został opracowany dla koncepcyjnego modelu danych, ilustrującego kluczowe encje gry (Gracz, Przeciwnik, Przedmiot, Ekwipunek, Walka) oraz relacje między nimi (Rysunek 2). Model ten, mimo że nie został zaimplementowany jako

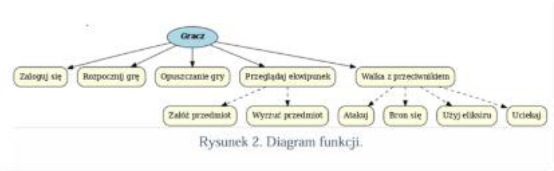
zewnętrzna baza danych, stanowił schemat logiczny dla struktur klas w projekcie.



Rysunek 1. Diaram encji.

Rysunek 1. Diagram przypadków użycia systemu gry.

Diagram przedstawia główne akcje gracza: Rozpocznij grę, Eksploruj świat, Zbieraj przedmioty, Zarządzaj ekwipunkiem, Rozpocznij walkę, Wybierz akcję w walce (Atak/Obrona/Użyj eliksiru/Ucieczka, Zakończ walkę.



Rysunek 2. Diagram funkcji.

Rysunek 2. Diagram encji (ERD) dla modelu danych gry.

Diagram pokazujący encje: Gracz (ID, statystyki), Przeciwnik (ID, typ, statystyki), Przedmiot (ID, typ, statystyki), Ekwipunek (ID, typ, powiązanie z Graczem), Walka (ID, uczestnicy, status). Relacje: Gracz posiada Ekwipunek, Ekwipunek zawiera Przedmioty, Gracz toczy Walkę z Przeciwnikiem.

Projekt przyjął modularną architekturę, gdzie poszczególne systemy (ruch, ekwipunek, walka) są odseparowane i komunikują się poprzez dobrze zdefiniowane interfejsy (np. wywołania metod, zdarzenia Unity). Kluczowymi koncepcjami implementacyjnymi były:

1. **Prefabrykaty (Prefabs)**: Do wielokrotnego używania szablonów obiektów (przeciwnicy, przedmioty, punkty zbierania surowców).
2. **Obiekty ScriptableObject**: Do definiowania danych przedmiotów (np. miecz, pierścień) jako zasobów niezależnych od instancji sceny, co ułatwia zarządzanie i edycję.
3. **System tagów i warstw (Tags & Layers)**: Do filtrowania kolizji i identyfikowania typów obiektów (np. Player, Enemy, Collectible).

2.3. Architektura i główne systemy

Architekturę gry zaprojektowano jako zestaw współpracujących, modularnych systemów, komunikujących się za pośrednictwem dobrze zdefiniowanych interfejsów. Głównym punktem integracji jest obiekt reprezentujący postać gracza (*Player*), który agreguje kluczowe komponenty odpowiedzialne za różne aspekty rozgrywki.

Podstawę systemu stanowią trzy główne moduły funkcjonalne:

1. **Moduł świata i interakcji:** Odpowiada za zarządzanie stałą mapą gry (zaimplementowaną przy użyciu *Tilemap*), kamerą podążającą za graczem oraz wykrywaniem wszystkich interakcji z otoczeniem. To w tym module obsługiwane są kolizje, zbieranie przedmiotów oraz inicjacja zdarzeń, takich jak spotkanie z przeciwnikiem;
2. **Moduł zarządzania zasobami i postacią:** Składa się z dwupoziomowego systemu ekwipunku, oraz klasy zarządzającej statystykami gracza. System ten odpowiada za przechowywanie przedmiotów, aplikowanie ich efektów (rozwój pośredni postaci), a także za konsumpcję eliksiró;
3. **Moduł walki turowej:** Jest to najbardziej złożony podsystem, zarządzany przez dedykowanego menedżera (*BattleManager*). Implementuje on cykl turowy, logikę wyboru akcji przez gracza oraz algorytmy obliczania szansy trafienia i obrażeń.

Komunikacja między tymi modułami odbywa się głównie poprzez wywołania metod, oraz mechanizm zdarzeń (*Unity Events*), co zapewnia niskie sprzężenie i wysoką elastyczność całego systemu. Taka struktura umożliwia niezależne rozwijanie, testowanie i wymianę poszczególnych komponentów.

2.4. System ekwipunku i przedmiotów

Zaimplementowano dwupoziomowy system ekwipunku, logicznie rozdzielający:

1. **Ekwipunek sprzętowy**: Przechowuje przedmioty aktywnie wpływające na statystyki gracza (broń, pancerz, biżuteria). Założenie przedmiotu powoduje natychmiastową aktualizację wartości siły, zwinności, punktów życia i kondycji w profilu gracza.
2. **Ekwipunek surowcowy (plecak)**: Przeznaczony dla przedmiotów konsumpcyjnych (eliksiry zdrowia, kondycji) oraz surowców (drewno, kamienie). Obsługuje stosowanie przedmiotów (np. użycie eliksiru w trakcie walki) oraz ich magazynowanie.

Przedmioty są reprezentowane przez klasy pochodne od Scriptable Object, co pozwala na edycję ich właściwości (nazwa, ikona, opis, bonusy do statystyk) bezpośrednio w edytorze Unity. Mechanizm zbierania opiera się na koliderach (Collider2D) i zdarzeniach OnTriggerEnter2D, gdzie obiekt z tagiem Collectible jest dodawany do odpowiedniego ekwipunku poprzez menedżer BagManager.

2.5 Turowy system walki

Jest to centralny i najbardziej złożony system projektu. Walka jest inicjowana przez kolizję gracza z obszarem agresji przeciwnika (OnTriggerEnter2D). Po jej rozpoczęciu aktywowany jest dedykowany panel interfejsu, a kontrolę przejmuje klasa BattleManager.

Kluczowe założenia systemu:

- Tura: jest podstawową jednostką czasu; gracz i przeciwnik wykonują na przemian jedną akcję.
- Gracz wybiera taktykę: ofensywną (szybka seria 3 ataków) lub defensywną (zredukowanie otrzymywanych obrażeń w zamian za 1 atak).
- Szansa na trafienie: jest dynamicznie obliczana na podstawie różnicy w statystyce zwinności (agility) atakującego i obrońcy.
- Obrażenia: są równe sile (strength) atakującego, pomniejszonej o wartość obrony, jeśli aktywna jest taktyka defensywna.

2.6 Przeciwnicy i prosta sztuczna inteligencja (AI)

Zaimplementowano kilku typów przeciwników (np. wilk, ork), różniących się zestawem statystyk podstawowych (życie, siła, zwinność) oraz **predyfinowaną taktyką walki**. AI przeciwnika jest deterministyczna i opiera się na prostych regułach:

- Wybór między atakiem standardowym a użyciem specjalnej umiejętności (jeśli zdefiniowana) na podstawie aktualnego stanu zdrowia i kondycji.
- Implementacja w formie metody ExecuteTurn() w klasie EnemyAI, wywoływanej przez BattleManager w turze przeciwnika.

Mimo swojej prostoty, system ten zapewnia wystarczającą różnorodność wyzwań, zmuszając gracza do adaptacji strategii w zależności od napotkanego typu wroga.

3. Imlementacja gry

W trakcie prac napotkano i rozwiązano kluczowe wyzwania techniczne:

1. Kolizja vs. Interakcja: Punkt zbierania surowców wymagał jednocześnie blokowania przejścia gracza oraz wykrywania interakcji. Rozwiązaniem było nałożenie na obiekt dwóch koliderów wykrywający wejście gracza w strefę interakcji dla uruchomienia zbierania.
2. Rozróżnianie typów przedmiotów: Aby system wiedział, czy zebrany przedmiot ma trafić do ekwipunku sprzętowego, czy surowcowego, wykorzystano system tagów Unity. Przedmioty otrzymały tagi Wearable lub Consumable, a skrypt zbierający na podstawie tagu kierował obiekt do odpowiedniego menedżera ekwipunku.

Zrealizowany projekt potwierdza słuszność przyjętej metodologii i technologii. Modularna architektura oparta na komponentach Unity okazała się niezwykle efektywna. Rozdzielenie odpowiedzialności na klasy takie jak Player, InventoryManager, BattleManager i EnemyAI** znacznie ułatwiło proces debugowania, testowania poszczególnych funkcji oraz wprowadzania

zmian. Zastosowanie ScriptableObject` dla danych przedmiotów uprościło proces ich tworzenia i balansowania, oddzielając dane od logiki.

Turowy system walki, będący sercem projektu, działa zgodnie z założeniami, oferując wymagającą taktyczną głębię. Prosty, lecz efektywny algorytm obliczania szansy trafienia wprowadza element niepewności, jednocześnie nagradzając inwestycję w statystykę zwinności. Rozdzielenie taktyk na ofensywną i defensywną tworzy przestrzeń dla strategicznych decyzji gracza.

Główne ograniczenia obecnej implementacji obejmują:

- Prostotę AI przeciwników: Obecna, deterministyczna AI jest przewidywalna w dłuższej rozgrywce. Jej rozwinięcie o elementy uczenia maszynowego lub bardziej złożone drzewa behawioralne stanowi naturalny kierunek rozwoju.
- Statyczny świat gry: Mapa, rozmieszczenie przeciwników i przedmiotów są stałe, co ogranicza regrywalność.
- Brak elementów narracyjnych i zadań (questów): Projekt skupił się na mechanikach, pomijając warstwę fabularną, kluczową dla wielu graczy RPG.

Porównanie z podobnymi rozwiązaniami (np. mechanikami „Broken Ranks") wskazuje, że udało się uchwycić istotę turowych, taktycznych pojedynków, przenosząc ją do uproszczonego, 2D środowiska. Kluczową zaletą stworzonego systemu jest jego skalowalność – dzięki czystej architekturze dodanie nowych typów akcji, umiejętności klasowych czy efektów statusowych jest stosunkowo proste.

4. Wnioski

Przeprowadzone prace doprowadziły do sukcesywnego zrealizowania założonych celów: zaprojektowania i implementacji kompletnej, działającej gry 2D z turowym systemem walki. Potwierdzono główną hipotezę badawczą – połączenie silnika Unity, języka C# oraz modularnego, obiektowego podejścia do projektowania okazało się niezwykle skuteczną metodologią tworzenia złożonych systemów gier, zapewniając wysoką jakość kodu, elastyczność i potencjał do ekspansji.

Projekt wykracza poza ramy prostego prototypu, stanowiąc solidną podstawę inżynierską dla pełnoprawnej gry. Zaimplementowane systemy (ekwipunek, walka, zarządzanie stanem) są funkcjonalne, stabilne i zaprojektowane z myślą o rozbudowie.

Kierunki dalszych badań i rozwoju projektu obejmują:

1. Rozwój AI: Wprowadzenie zaawansowanych algorytmów planowania (np. drzewa behawioralne - Behaviour Trees) lub prostych sieci neuronowych dla adaptacyjnych przeciwników.
2. Proceduralna generacja treści: Zastosowanie algorytmów do generowania losowych map, lokacji i rozmieszczenia przeciwników/łupów w celu zwiększenia regrywalności.
3. Rozbudowa systemu walki: Dodanie większej liczby umiejętności, efektów magicznych (np. odurzenie, podpalenie), systemu klasy postaci oraz trybu walki drużynowej.
4. Integracja sieciowa: Przekształcenie projektu w grę wieloosobową (co-op lub PvP), co wymagałoby implementacji sieciowej synchronizacji stanu gry.
5. Wzbogacenie warstwy narracyjnej: Dodanie systemu zadań, dialogów z NPC oraz nieliniowej fabuły.

Podsumowując, praca demonstruje kompleksowy proces tworzenia gry – od koncepcji, przez projektowanie systemowe, implementację, po testowanie. Stanowi wartościowy przypadek studyjny dla adeptów inżynierii oprogramowania i projektowania gier, pokazując, jak za pomocą współczesnych, dostępnych narzędzi można zbudować fundamenty zaawansowanego projektu interaktywnego.

Bibliografia

1. Unity Technologies. *Unity User Manual (2022.3 LTS)*. Dostęp: docs.unity3d.com/Manual/index.html (16.02.2025).

2. Freeman, A., & G. (2022). *Programowanie gier w Unity i C#. Kompletny przewodnik*. Helion.
3. Blog producenta gry *Broken Ranks* (dawniej *The Pride of Taern*). Sekcja opisująca mechanikę walki. Dostęp: brokenranks.com/#sectionFights (16.02.2025).
4. Halpern, J. (2019). *Jak pisać świetne gry 2D w Unity. Niezależne programowanie w języku C#*. Helion.
5. Nystrom, R. (2014). *Game Programming Patterns*. Genever Benning. Dostęp: gameprogrammingpatterns.com/ (16.02.2025).
6. Dokumentacja Microsoft: *Język programowania C#*. Dostęp: docs.microsoft.com/pl-pl/dotnet/csharp/ (16.02.2025).
7. Stellman, A., & Greene, J. (2023). *C#. Rusz głową! Wydanie IV*. Helion.
8. Oficjalna dokumentacja narzędzia *Piskel* do tworzenia pixel art. Dostęp: www.piskelapp.com/ (16.02.2025).