Dokumentacja projektu "Platform Jumper"

Tomasz Nazar (197613) Filip Pietrzak (198275)

11 czerwca 2025

1 Opis ogólny projektu

Projekt *Platform Jumper* to klasyczna gra platformowa 2D zrealizowana w bibliotece **Pygame**. Gracz steruje postacią poruszającą się w poziomie, zdolną do skoków i interakcji z otoczeniem. Gra posiada menu główne, menu wyboru poziomu, edytor map oraz mechanikę zapisu wyników.

2 Struktura projektu

- `game.py` – główny plik uruchamiający grę i zarządzający logiką rozgrywki
- `editor.py` – prosty edytor poziomów
- `scripts/` – folder zawierający moduły gry:
  - `audio.py` – obsługa dźwięku
  - `clouds.py` – animacja chmur
  - `entities.py` – definicja gracza i encji fizycznych
  - `tilemap.py` – renderowanie i obsługa mapy
  - `utility.py` – funkcje pomocnicze (np. wczytywanie obrazów, animacje)
- `data/` – zasoby (obrazy, dźwięki, czcionki, mapy)

3 Opis modułów

3.1 `game.py`

Główny plik uruchamiający grę. Klasa `Game` obsługuje:

- inicjalizację Pygame, ekranu i stanu gry,
- obsługę menu głównego, wyboru poziomu i podsumowania wyniku,
- logikę rozgrywki: ruch gracza, kolizje, skoki, kamerę,
- rysowanie interfejsu, licznik czasu, paski siły skoku,
- system pauzy z interaktywnym menu i podświetlaniem przycisków.

Używa wielu komponentów z modułów pomocniczych (audio, tilemap, utility). Obsługuje restart poziomu, koniec gry oraz interakcje z użytkownikiem.

Metody klasy `Game`:

- `__init__()` – konstruktor gry, odpowiedzialny za przygotowanie okna, zegara, zasobów i zmiennych stanu.
- `reset()` – ustawia domyślne wartości zmiennych gry przed ponownym uruchomieniem poziomu.
- `main_menu()` – tworzy i wyświetla główne menu gry z przyciskami i tłem.
- `level_picker()` – menu wyboru poziomu. Renderuje dynamiczne tło i przyciski do wyboru poziomu.
- `display_summary(ending_time)` – ekran z wynikiem po ukończeniu poziomu. Wyświetla czas, liczbę skoków, formularz z nazwą gracza.
- `run(level=None)` – główna pętla gry. Obsługuje wejścia użytkownika, logikę rozgrywki, kolizje, kamerę i rendering.
- `_scale_pixel_art_image(image, w, h)` – skalowanie grafiki bez rozmycia, z zachowaniem stylu pixel-art.

Klasa `Button` odpowiada za tworzenie przycisków z teksturą w stylu pixel-art oraz obsługę ich interakcji i podświetlenia.

Metody klasy `Button`:

- `__init__(...)` – konstruktor, który ustawia tekst, pozycję, wymiary, czcionkę i opcjonalne odbicia tekstury. Wczytuje grafikę i generuje wersję podświetloną.
- `_create_hover_image(image)` – tworzy podświetloną wersję obrazka poprzez nałożenie półprzezroczystej powierzchni z lekkim rozjaśnieniem (efekt hover).
- `_scale_image_pixel_art(image, target_w, target_h)` – skaluje teksturę przycisku bez utraty ostrości, zachowując styl pixel-art (przez całkowite mnożniki rozmiaru).
- `draw(screen, mouse_pos=None)` – renderuje przycisk na ekranie, podświetla go jeśli kursor najeżdża, a następnie wyświetla tekst na środku.
- `is_clicked(mouse_pos=None)` – zwraca `True`, jeśli pozycja myszy znajduje się wewnątrz przycisku (sprawdzenie kolizji prostokątnej).
- `get_scaled_mouse_pos(screen, display)` – przelicza współrzędne myszy z dużego okna na współrzędne powierzchni roboczej gry (jeśli gra jest skalowana).

3.2 `editor.py`

Prosty edytor poziomów działający na zasadzie:

- wczytywania mapy i tekstur z folderu,
- dodawania i usuwania płytek kliknięciem myszy,

- zapisu mapy do pliku JSON,
- wizualnego podglądu edytowanych obiektów.

Edytor pozwala na szybkie tworzenie nowych poziomów bez konieczności edytowania plików ręcznie.

3.3 `audio.py`

Moduł odpowiada za:

- odtwarzanie efektów dźwiękowych i muzyki w tle,
- buforowanie dźwięków dla optymalizacji,
- możliwość ustawiania głośności i zapętleń.

Metody:

- `play_sound(name, volume)` – odtwarza dźwięk,
- `play_music(name, volume)` – odtwarza muzykę z zapętlaniem.

3.4 `clouds.py`

Zawiera klasę `Clouds`, odpowiedzialną za:

- generowanie chmur w tle,
- losowy ruch chmur,
- renderowanie ich z przesunięciem względem kamery.

3.5 `entities.py`

Moduł implementuje klasę `Player` dziedziczącą po `PhysicsEntity`:

- detekcja kolizji i grawitacja,
- kontrola animacji, kierunku ruchu i skoku,
- liczenie liczby skoków i czasu w powietrzu,
- logika wygranej.

3.6 `tilemap.py`

Odpowiada za:

- renderowanie mapy na podstawie JSON,
- obsługę płytek typu off-grid i on-grid,
- zapis i odczyt map.

3.7 `utility.py`

Zawiera:

- funkcje `load_image`, `load_images` do wczytywania tekstur,
- klasę `Animation` – obsługa sekwencji klatek i czasu trwania,
- funkcję `save_to_excel` do zapisu wyników w formacie Excel.

4 Wnioski

Projekt *Platform Jumper* ukazuje opanowanie biblioteki Pygame oraz zasad programowania obiektowego. Wysoki poziom organizacji kodu przekłada się na jego czytelność i możliwość dalszego rozwoju. Cechy szczególne projektu to:

- system animacji i kolizji oparty na własnych klasach fizycznych,
- intuicyjny interfejs użytkownika z efektami hover i kursorem,
- pełna obsługa dźwięku oraz synchronizacja efektów kroków z ruchem postaci,
- integracja z formatem Excel do przechowywania wyników,
- autorski edytor poziomów ułatwiający tworzenie nowych map,
- pixel-artowa estetyka z zachowaniem jakości wizualnej,
- optymalizacje pod kątem wydajności (m.in. pre-generowanie wersji hover dla przycisków).

Dzięki tym rozwiązaniom projekt stanowi solidną bazę do dalszego rozwoju. Możliwe rozszerzenia to:

- dodanie przeciwników z prostą sztuczną inteligencją,
- implementacja systemu punktów lub zbieranych przedmiotów,
- wprowadzenie efektów pogodowych i zaawansowanych animacji,
- rozbudowa systemu poziomów o fabularny kontekst lub kampanię.

Całość projektu świadczy o dobrej współpracy zespołu, umiejętności podziału pracy oraz konsekwentnym podejściu do projektowania gry 2D.