2025

Jumper's Odyssey

AUTORZY:
KAMIL WNUK, RAFAŁ PIWOWAREK, MATEUSZ WSZOŁA

Spis treści

1. Opis projektu 2
1.1. Cel projektu i czego dotyczy 2
1.2. Technologie i narzędzia 2
1.3. Podział zadań 3
1.4. Instrukcja instalacji 4
2. Struktura projektu 5
2.1 Opis katalogów 5
2.2 Opis plików 6
3. Zdjęcia / opisy działania aplikacji oraz funkcjonalności 9
3.1. Ekran gry 9
3.2. Rozgrywka 9
4. Bibliografia 12

1. Opis projektu

1.1. Cel projektu i czego dotyczy

Celem projektu jest stworzenie prostej gry platformowej 2D w Pythonie, z wykorzystaniem biblioteki Pygame oraz edytora map Tiled (.tmx). Gracz może poruszać postacią, strzelać, zbierać przedmioty ("itemy") i otwierać drzwi w określonych momentach (po zebraniu wystarczającej liczby itemów). Mapy są projektowane w Tiled, a kod odczytuje warstwy:

- **baza** – kafelki podłóg, ścian (z kolizją)
- **dekoracja** – kafelki tła (bez kolizji)
- **drzwi1–drzwi5** – warstwy zawierające "drzwi" (kafelki i ich kolizje)
- **itemy** – obiekty, które gracz zbiera

W momencie zebrania określonej liczby itemów (np. 1, 5, 7, 9, 15) dana warstwa drzwi zostaje usunięta (zarówno wizualnie, jak i kolizyjnie), co umożliwia przejście do kolejnego rejonu mapy.

1.2. Technologie i narzędzia

- **Język programowania**: Python 3.8+
- **Biblioteki / moduły**:
    - pygame (do renderowania grafiki, dźwięku, obsługi okna, zdarzeń, sprite'ów)
    - pytmx (moduł load_pygame do wczytywania plików .tmx z Tiled)
    - os, os.path, random, math (wbudowane moduły Pythona)
- **Edytor map**: Tiled Map Editor (wersja ~1.9.3)
- **Formaty plików**:
    - Mapa: .tmx (Tile Map XML)
    - Grafiki: .png (obrazy postaci, wrogów, kafelków)
    - Dźwięki: .wav lub .ogg (audio z folderu audio/)
- **Środowisko uruchomieniowe**: dowolny system z zainstalowanym Pythonem i biblioteką Pygame (Windows/Linux/macOS)

1.3. Podział zadań

Kamil Wnuk

Moduł główny gry (main.py)

1. Inicjalizacja Pygame (okno, tytuł, zegar).
2. **Ustawienie i zarządzanie grupami sprite'ów** (all_sprites, collision_sprites, collision_sprites, bullet_sprites, enemy_sprites, item_sprites).
3. **Metoda setup()** odpowiadająca za załadowanie mapy .tmx.
4. **Metoda create_bullet():** tworzenie pocisku (Bullet) i efektu ognia (Fire), odtwarzanie dźwięku strzału.
5. **Metoda collision().**
6. **Metoda run().**

Rafał Piwowarek:

Definicja i implementacja klas sprite'ów (sprites.py)

1. **Sprite** – klasa bazowa: przyjmuje pos, surface i groups, ustawia self.image i self.rect.
2. **AnimatedSprite** – klasa do animowanych obiektów.
3. **Enemy** (klasa bazowa dla wrogów).
4. **Bee i Worm** – podklasy Enemy.
5. **Player** (klasa gracza).
6. **Bullet** – klasa pocisku.
7. **Fire** – efekt płomienia przy strzale.
8. **Item** – obiekt do zebrania.
9. **CollisionTile** – sprite kolizji (drzwi/ściany).

Mateusz Wszoła:

1. **Moduły pomocnicze i konfiguracja**:
   - settings.py.
   - support.py.
   - timer.py.
   - groups.py.

2. **Stworzenie mapy gry w programie Tiled**

1.4. Instrukcja instalacji

1. **Wymagania wstępne**

    - Python 3.8 lub nowszy zainstalowany w systemie (pobrać ze strony https://www.python.org/).
    - pip (menedżer pakietów Pythona).
    - PyCharm (lub inne IDE obsługujące Pythona; rekomendowane do wygodnej pracy nad projektem).

2. **Otwórz projekt w IDE (np. PyCharm)**

3. **Zainstaluj wymagane biblioteki**
   Otwórz nowy terminal w IDE i uruchom w terminalu polecenie:

```
pip install pygame pytmx
```

4. **Uruchomienie gry**

W terminalu w katalogu głównym projektu uruchom:

```
python code/main.py
```

Lub skorzystaj z kompilatora swojego IDE

2. Struktura projektu

2.1 Opis katalogów

```
Jumpers_Odyssey/
├── audio/                # folder z efektami dźwiękowymi (.wav/.ogg)
├── data/
│   └── maps/
│       ├── kafelki.tsx   # komponent do renderowania pojedynczych kafelków
│       ├── mapa.png      # arkusz kafelków (tileset)
│       └── mapa.tmx      # plik mapy wygenerowany w Tiled
├── images/
│   ├── player/
│   │   ├── 0.png
│   │   └── …             # klatki animacji gracza
│   ├── enemies/
│   │   ├── bee/
│   │   │   ├── 0.png
│   │   │   └── …         # klatki animacji pszczoły
│   │   └── worm/
│   │       ├── 0.png
│   │       └── …         # klatki animacji robaka
│   └── gun/
│       ├── bullet.png
│       └── fire.png        # grafiki pocisku i efektu wystrzału
└── code/
    ├── groups.py           # klasa AllSprites (obsługa kamery + draw)
    ├── settings.py         # stałe globalne
    ├── sprites.py          # definicje wszystkich sprite'ów:
    ├── support.py          # funkcje pomocnicze: import_image,
    │                         import_folder, audio_importer
    ├── timer.py            # klasa Timer (prosty mechanizm odliczający
    │                         czas)
    └── main.py             # główna klasa Game, pętla, obsługa kolizji i
                              rysowania
```

2.2 Opis plików

1. settings.py

Definiuje globalne stałe, używane dla spójności rozmiarów okna i kafelków.

```
WINDOW_WIDTH, WINDOW_HEIGHT = 1920, 1080

TILE_SIZE = 64

FRAMERATE = 60

BG_COLOR = '#fcdfcd'
```

2. groups.py

- **Klasa AllSprites(pygame.sprite.Group) rozszerzona o metodę draw(target_pos):**
  - Oblicza offset kamery na podstawie target_pos
  - Rysuje każdy sprite w pozycji sprite.rect.topleft + offset.
  - Pozwala na przesuwanie widoku (tzw. „kamera śledząca gracza").

3. support.py

- **import_image(*path, format='png', alpha=True):**
  - Wczytuje pojedynczy obrazek i zwraca pygame.Surface.

- **import_folder(*path):**
  - Wczytuje wszystkie pliki z folderu (posortowane według nazwy liczbowej) i zwraca listę Surface (do animacji).

- **audio_importer(*path):**
  - Wczytuje wszystkie pliki dźwiękowe z folderu, zwraca słownik {'nazwa_pliku': Sound}.

4. timer.py

- **klasa TImer:**
  - __init__(self, duration, func=None, repeat=None, autostart=False) – tworzy timer, który po duration (ms) wywoła func.
  - activate() – rozpoczyna odliczanie.
  - update() – sprawdza, czy czas minął, wywołuje func() i deaktywuje się (lub restartuje, jeśli repeat=True).

5. sprites.py

- **class Sprite(pygame.sprite.Sprite):**
  - Baza dla wszystkich sprite'ów (przyjmuje pos, surf, groups).

- **class Bullet(Sprite):**
  - Pocisk wystrzeliwany przez gracza, porusza się w poziomie z prędkością self.speed.

- **class Fire(Sprite):**
  - Efekt płomienia przy wystrzale broni; przyspieszony timer (Timer(100)) usuwa go po krótkiej chwili.

- **class AnimatedSprite(Sprite):**
  - klasa do animowanych obiektów (zawiera listę frames, frame_index, metoda animate(dt).

- **class Enemy(AnimatedSprite):**
  - Bazowa klasa wroga, z timerem death_timer i metodą destroy() dającą efekt umierania.

- **class Bee(Enemy)** i **class Worm(Enemy)**:
  - Dwa typy wrogów, poruszające się losowo w obrębie prostokąta main_rect określonego na mapie przy pomocy Tiled, zmieniające kierunek (flipy animacji) po przekroczeniu granic.

- **class Player(AnimatedSprite)**:
  - **Gracz:**
    - Ruch w osi X (klawisze A/D), skok (W), grawitacja (stała 50).
    - Metoda input() odczytuje naciśnięcia klawiatury.
    - move(dt) – przesuwa gracza w osi X i Y, wywołuje kolizje (self.collision('horizontal'), self.collision('vertical')).
    - check_floor() – sprawdza, czy gracz stoi na powierzchni, ustala flagę on_floor.
    - animate(dt) – zmienia klatkę animacji w zależności od ruchu (i flipu).

  - **class Item(pygame.sprite.Sprite):**
    - Obiekt do zebrania; prostokąt wypełniony kolorem (255, 223, 0), ma atrybut value z liczbą punktów.

  - **class CollisionTile(pygame.sprite.Sprite):**
    - Reprezentuje kolizję z drzwiami oraz czy usunąć dane drzwi. Przyjmuje pos, surf, removable, *groups, group_id.

6. main.py

- **Game:**
  - **__init__()** – inicjalizuje Pygame,
    ustawia okno i zmienne globalne,
    tworzy grupy sprite'ów (all_sprites, collision_sprites, bullet_sprites, enemy_sprites, item_sprites, drzwi#_sprites),
    ładuje zasoby i wywołuje setup().

  - **create_bullet(pos, direction)** – tworzy obiekt Bullet i efekt Fire, odtwarza dźwięk wystrzału.

  - **load_assets()** – wczytuje grafiki (animacje gracza, wrogów, obrazki pocisku/płomienia) i dźwięki (z audio_importer).

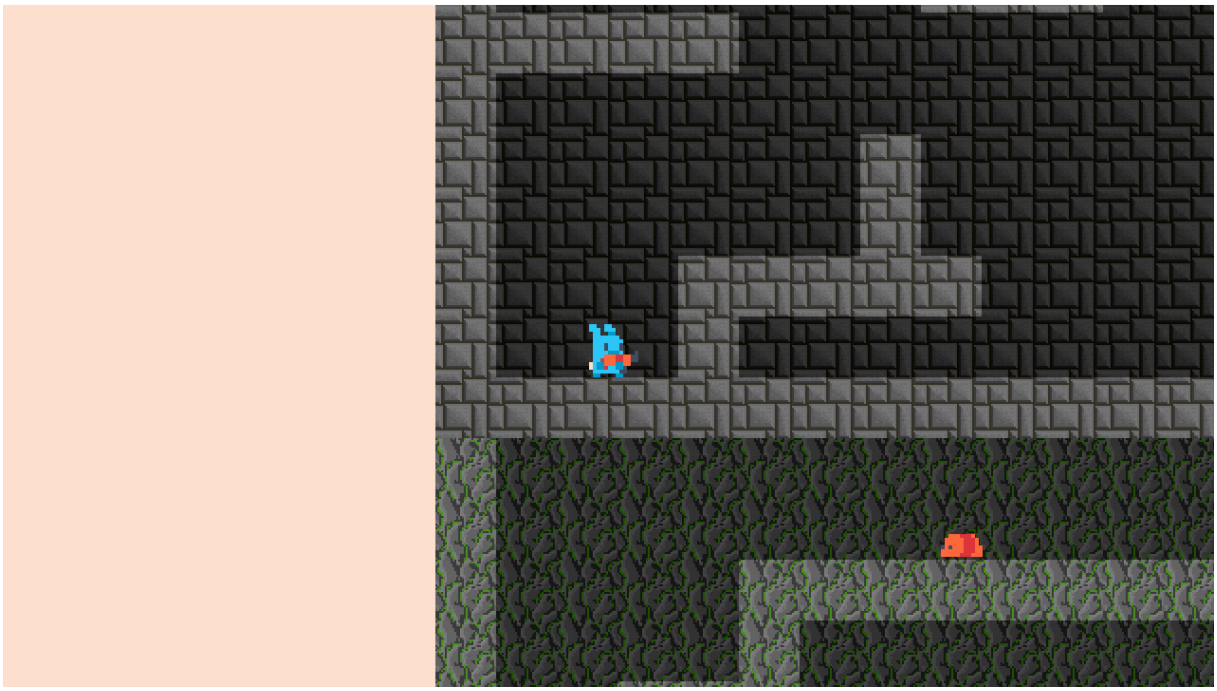
- **setup()** – wczytuje mapę TMX (z pytmx.load_pygame),
  ustala wymiary poziomu,
  iteruje po warstwach drzwi (drzwi1 ... drzwi5),
  tworzy CollisionTile dla każdego kafelka (z group_id='drzwi#'),
  wczytuje obiekty itemów,
  wczytuje warstwę baza (kafelki podłóg → Sprite + collision_sprites),
  warstwę dekoracja → Sprite(all_sprites),
  warstwy graficzne drzwi → Sprite(all_sprites, drzwi#_sprites),
  warstwę przeciwnicy → Player, Worm, Bee.
  Na końcu ustawia collected_count = 0, target_count1 = 1.

- **collision()** – sprawdza kolizje:
  - Pociski → wrogowie (collide_mask) → niszczy wroga, usuwa pocisk, odtwarza dźwięk impact
  - Pociski → powierzchnie (collision_sprites, używając colliderect) → usuwa pocisk, odtwarza dźwięk impact.
  - Wrogowie → gracz (collide_mask) → kończy grę (self.running=False).
  - Gracz → itemy (dokill=True) → usuwa Item, inkrementuje collected_count, odtwarza dźwięk zebrania itemu (impact).
  - Jeśli collected_count >= 1 i not pending_door_sound, usuwa kolizje i sprity z drzwi1. Ustawia pending_door_sound=True.
  - Analogicznie dla progów 5, 7, 9, 15 – usuwa drzwi2, drzwi3, drzwi4, drzwi5.
- **run()** – główna pętla:
  - Obsługuje zdarzenia (QUIT, KEYDOWN → ESCAPE).
  - Wywołuje self.all_sprites.update(dt) i self.collision().
  - Czyści ekran (fill(BG_COLOR)), rysuje wszystkie sprite'y (all_sprites.draw(player.rect.center)), a następnie rysuje itemy z offsetem kamery.
  - pygame.display.update().
  - Po wyjściu z pętli zamyka Pygame (pygame.quit()).

3. Zdjęcia / opisy działania aplikacji oraz funkcjonalności

3.1. Ekran gry

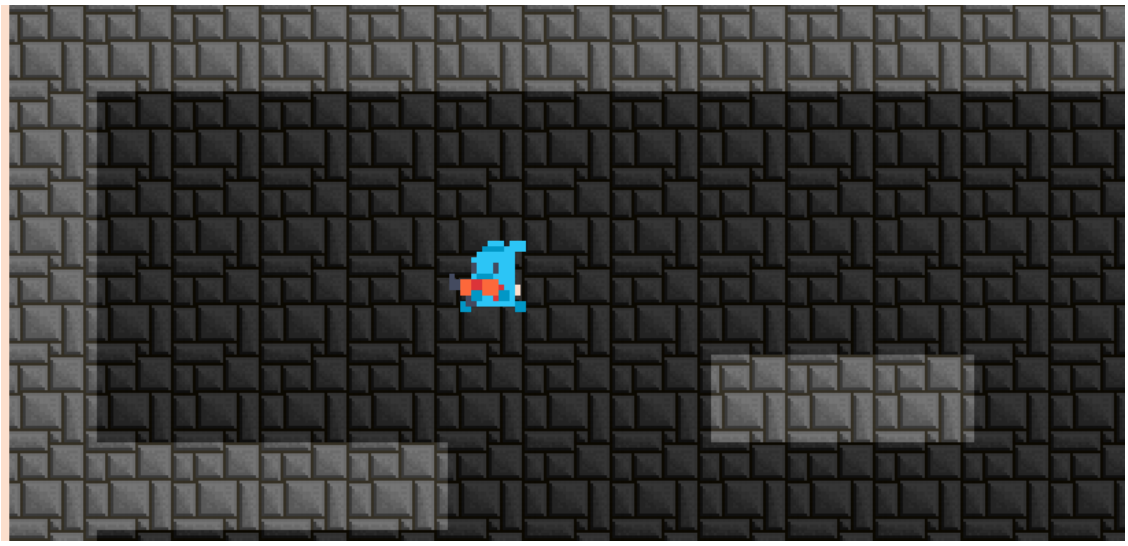
Po uruchomieniu pojawia się okno Pygame o rozdzielczości 1920×1080 z grą. Podkład muzyczny zaczyna się odtwarzać. Zostają załadowane wszystkie tekstury, przeciwnicy itp.



3.2. Rozgrywka

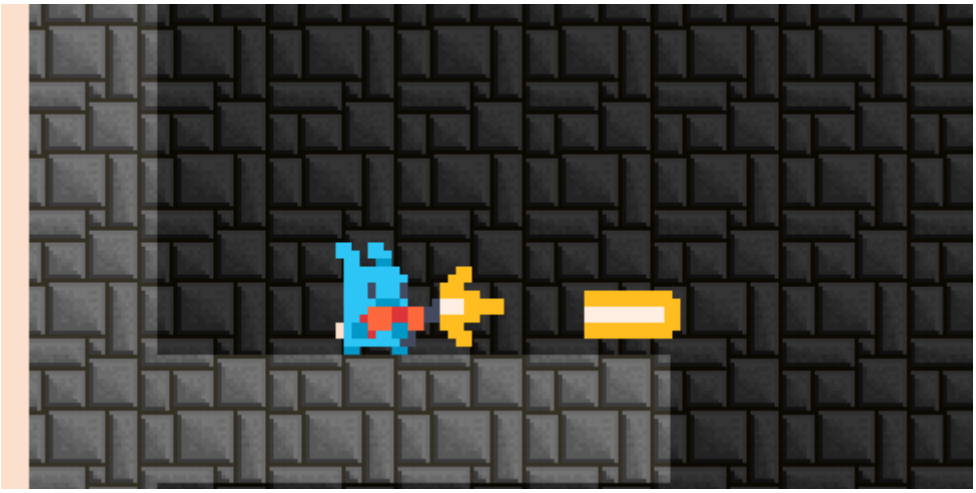
1. Ruch gracza

- Klawisze A/D przesuwają postać w lewo/prawo.
- Klawisz W pozwala na skok.
- Grawitacja (50) powoduje płynne opadanie, a wykrywanie „podłoża” odbywa się przez rysowanie Rect pod nogami i sprawdzenie collidelist.
- Animacja chodzenia i flip (lustrzane odbicie) działa w zależności od kierunku wektora prędkości.



2. Strzał z pistoletu

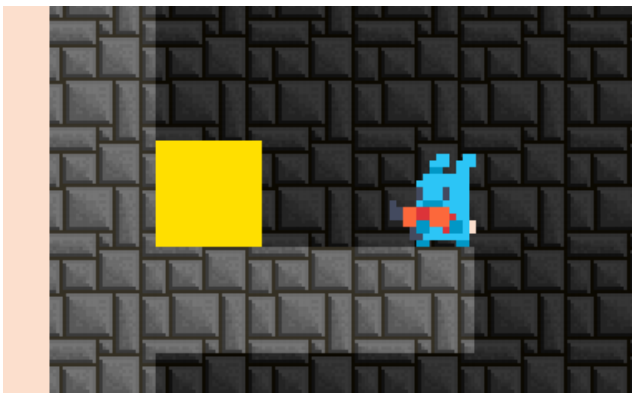
- Klawisz SPACE wystrzeliwuje pocisk (klasa Bullet), który przemieszcza się w lewo lub w prawo z prędkością 850 px/s.
- Efekt płomienia (Fire) pojawia się przy lufie.
- Odtwarza się dźwięk wystrzału (shoot.wav).

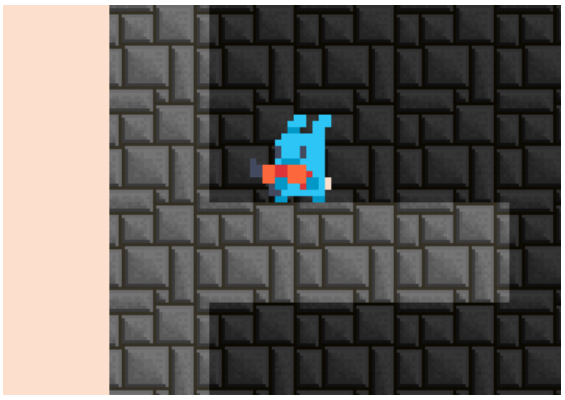


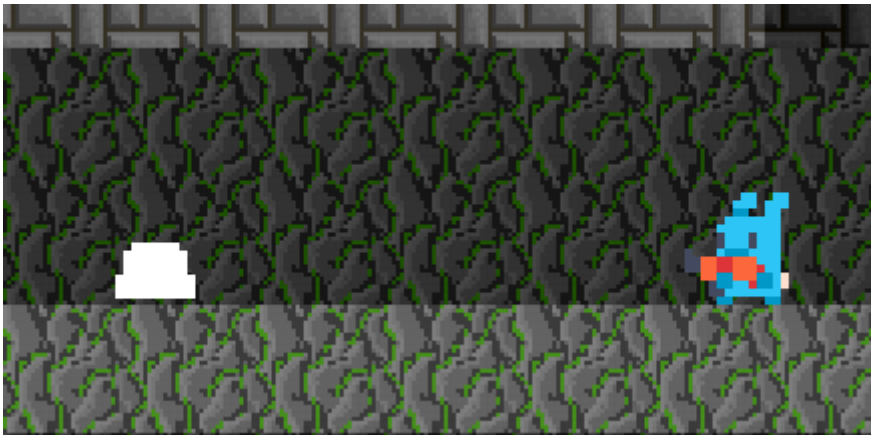
3. Kolizje

- **Pocisk → wróg**: po wykryciu kolizji pocisk znika, wróg zaczyna destroy() → animacja masky i po 200 ms ginie. Jest odtwarzany dźwięk trafienia (impact.wav).
- **Pocisk → ściana (collision_sprites)**: wykrywane przez prostokąt (colliderect), pocisk znika, odtwarzany jest dźwięk impact.
- **Wrogowie → gracz**: jeśli nastąpi maskowa kolizja, self.running=False → gra się zamyka.

- **Gracz → item**: każdy Item po kolizji jest usuwany (dokill=True), do zmiennej collected_count dodawana jest wartość item.value (domyślnie 1). Następuje odtworzenie dźwięku (impact.wav).



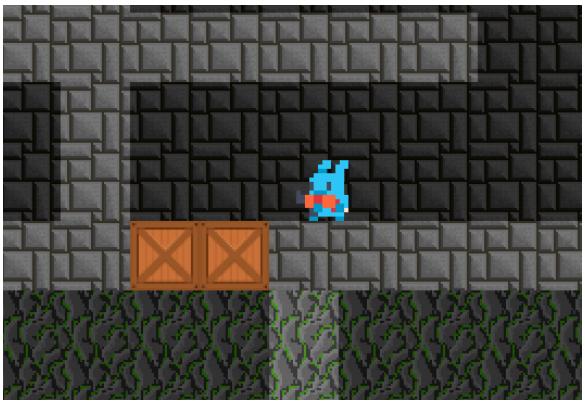




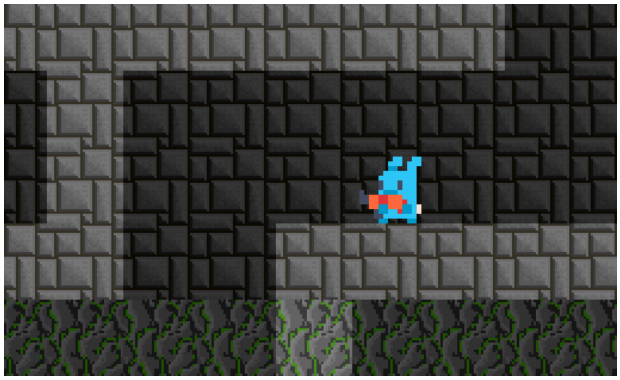
4. Zbieranie itemów i otwieranie drzwi

- Warstwa itemy z pliku mapa.tmx zawiera obiekty, którym w Tiled ustawiono property item=true.
- Po zebraniu 1 itemu:
    - Przypisany group_id='drzwi1' w CollisionTile (warstwa kolizji drzwi1) zostaje usunięty (kolizje i grafika).
- Po zebraniu 5 itemów (w sumie) analogicznie usuwane są drzwi2, po 7 – drzwi3, po 9 – drzwi4, po 15 – drzwi5.
- Dzięki temu poziom stopniowo się otwiera, gdy gracz zbiera przedmioty.

Przed zebranie 1 przedmiotu:



Po zebraniu 1 przedmiotu



5. Dźwięki

- Każda interakcja bullet → ściana / wrogowie, czy gracz → item odtwarza dźwięk impact.wav z głośnością 0.2.

6. Zamknięcie gry

- Kliknięcie krzyżyka w oknie lub naciśnięcie ESCAPE ustawia self.running=False i ładuje pygame.quit(), co zamyka aplikację.

4. Bibliografia

1. **Dokumentacja Pygame**
    - Oficjalna strona: https://www.pygame.org/docs/
    - Poradniki: „Pygame Tutorial” (Kidscancode, Tech with Tim)
2. **Dokumentacja Tiled Map Editor**
    - Oficjalna strona: https://doc.mapeditor.org/
3. **Biblioteka pytmx**
    - GitHub: https://github.com/bitcraft/pytmx
    - Dokumentacja modułu: pytmx.util_pygame.load_pygame
4. **Tutoriale wideo**
    - „Pygame Platformer Tutorial” (Tech with Tim, YouTube)
    - „Pygame Tile Based Game Tutorial: Tilemaps” (CDcodes, YouTube)