Temat: "Kosmiczna Strzelanka 2D – Roguelite Arcade" z wykorzystaniem sztucznej inteligencji

1. Wstęp

Celem projektu było stworzenie w pełni funkcjonalnej, dwuwymiarowej gry zręcznościowej w języku Python z wykorzystaniem biblioteki Pygame. Proces tworzenia aplikacji opierał się na ścisłej współpracy z modelami generatywnej sztucznej inteligencji, która pełniła rolę wirtualnego asystenta, ("pair-programmera") oraz
doradcy w zakresie game designu. Gra ewoluowała z prostego prototypu poruszania się obiektem do złożonej aplikacji zawierającej tryb kooperacji, proceduralne generowanie dźwięku, system postępów (meta-progresja) oraz zapisywanie wyników w formacie JSON.

2. Metryka narzędzi AI i licencjonowanie

Poniższa tabela przedstawia szczegóły techniczne i prawne dotyczące wykorzystanych narzędzi generatywnej sztucznej inteligencji:

Narzędzie	Konkretny model	Typ usługi	Możliwość wykorzystania w projekcie
Google Gemini	Gemini 3 Pro	Plan Darmowy / Pro	Tak (własność wygenerowanego kodu leży po stronie użytkownika, wykorzystanie edukacyjne)

Oświadczenie o licencji: Zgodnie z regulaminem usług Google, użytkownik zachowuje prawa do kodu wygenerowanego na podstawie własnych promptów. Skrypty i logika gry powstały jako odpowiedź na precyzyjne komendy inżynieryjne sformułowane w ramach tego projektu edukacyjnego.

3. Struktura repozytorium projektu

Zgodnie z dobrymi praktykami programistycznymi, projekt został zorganizowany w przejrzystą strukturę plików:

/ (Katalog główny)
`game.py` – Główny skrypt aplikacji zawierający pętlę gry (Game Loop), definicje klas (Player, Enemy, Boss, Bullet, Particle) oraz syntezator dźwięku.

`leaderboard.json` – Plik generowany automatycznie, przechowujący dane w formacie JSON o najlepszych wynikach (TOP 10) oraz nazwach graczy.
`save_data.json` – Plik odpowiadający za meta-progresję (sklep). Zapisuje ilość zebranych przez gracza monet oraz poziom zakupionych ulepszeń (np. HP, startowy poziom).
`README.md` – Krótki opis projektu, instrukcja uruchomienia oraz wymagania systemowe (np. `pip install pygame`).

4. Współpraca z AI i inżynieria promptów (Prompt Engineering)

Wykorzystanie AI nie ograniczało się do prostego generowania kodu, ale polegało na iteracyjnym rozwiązywaniu problemów i dodawaniu kolejnych warstw złożoności. Nauczyłem się formułować precyzyjne, techniczne zapytania.

Przykład 1: Wdrażanie systemów mechaniki

Mój prompt: "Dodaj opcje wyboru poziomu trudności"
Odpowiedź AI: AI zaimplementowało słownik `DIFFICULTY_SETTINGS` zawierający mnożniki prędkości i obrażeń, a także utworzyło system zarządzania stanami maszyny (Game States) wprowadzając stan `"MENU"`.

Przykład 2: Rozwiązywanie konfliktów UI

Mój prompt: "Spraw aby pod napisami z poziomem itp nie pojawiały się żadne ulepszenia"
Analiza: AI zrozumiało problem nakładania się interfejsu użytkownika (UI) na pole gry i zastosowało algorytm strefy wykluczenia (Safe Zone) przy losowaniu współrzędnych (klasy `PowerUp` i `HealthPack`), wykorzystując pętlę `while True` z instrukcjami `continue`, dopóki wylosowane współrzędne nie znajdą się poza określonymi pikselami ekranu.

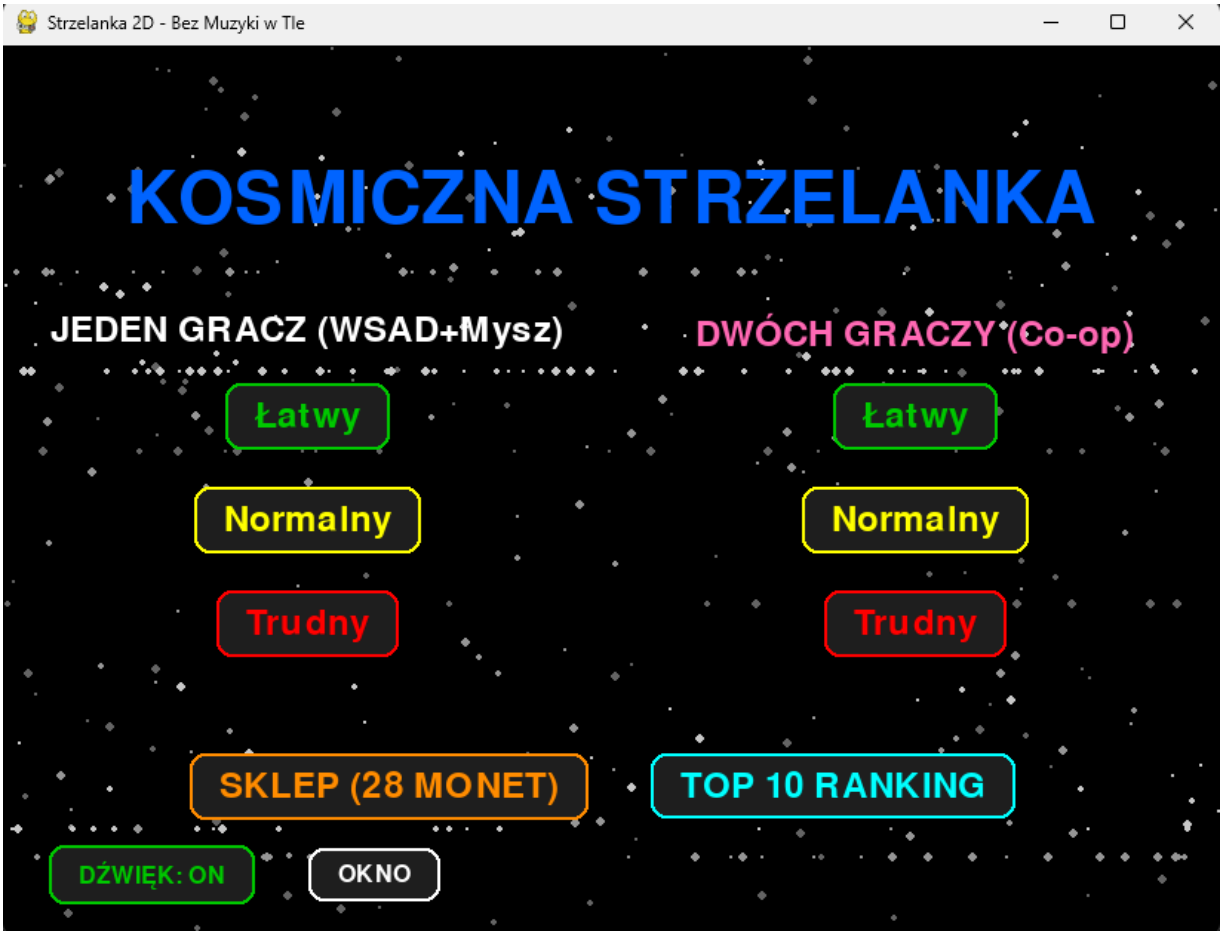
Przykład 3: Debugowanie i naprawa błędów

Podczas implementacji zaawansowanych systemów wystąpił błąd `NameError: name 'Particle' is not defined`. AI potrafiło przeanalizować podany zrzut ekranu konsoli i wywnioskować, że podczas refaktoryzacji kodu usunięto definicję obiektu cząsteczkowego, po czym błyskawicznie wygenerowało łatkę naprawczą.

```
PS C:\W5\10w5tk1> python c:\tmp\game.py
pygame 2.6.1 (SDL 2.28.4, Python 3.13.12)
Hello from the pygame community. https://www.pygame.org/contribute.html
PS C:\W5\10w5tk1> python c:\tmp\game.py
pygame 2.6.1 (SDL 2.28.4, Python 3.13.12)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "c:\tmp\game.py", line 911, in <module>
    main()
    ~~~~^^
  File "c:\tmp\game.py", line 751, in main
    for _ in range(15): particles.append(Particle(enemy.x, enemy.y, enemy.color))
                                         ^^^^^^^^
NameError: name 'Particle' is not defined. Did you mean: 'particles'?
PS C:\W5\10w5tk1>
```

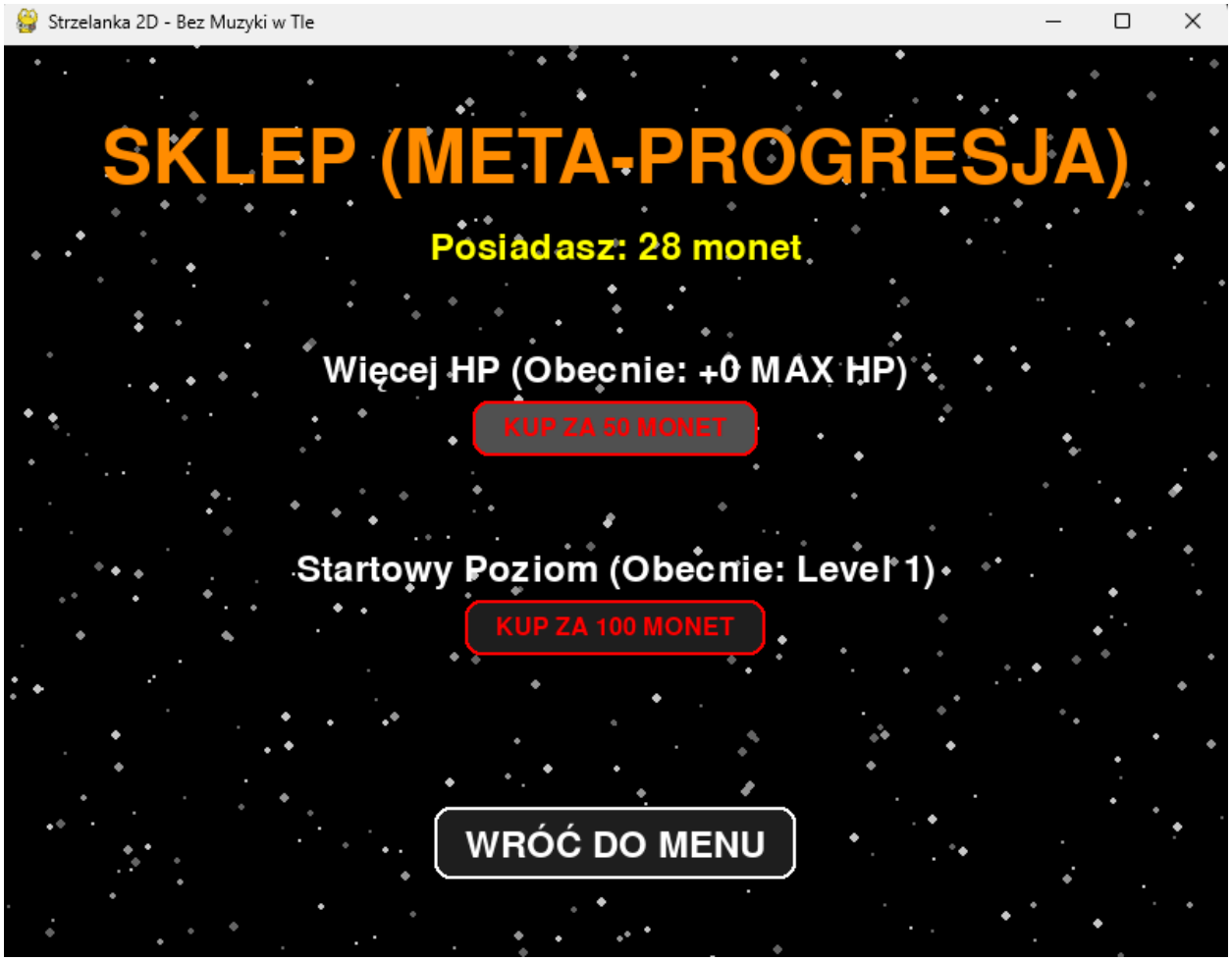
5. Architektura aplikacji i analiza najciekawszych fragmentów kodu

Aplikacja została napisana w paradygmacie programowania obiektowego (OOP). Całość opiera się na wydajnej pętli `while running`, która odświeża logikę i rysuje grafikę 60 razy na sekundę (`clock.tick(60)`).




Strzelan ka 2D - Bez Muzyki w Ale
WYBÓR STATKÓW
Gracz 1 (Obecnie: BALANCED)
Lekki (Szybki, 50HP)
Zbalansowany (100HP)
Ciężki (Wolny, 200HP, Armor)
START GRY!
WRÓĆ


Strzelanka 2D - Bez Muzyki w Tle
WYBÓR STATKÓW
Gracz 1 (Obecnie: BALANCED)
Lekki (Szybki, 50HP)
Zbalansowany (100HP)
Ciężki (Wolny, 200HP, Armor)
Gracz 2 (Obecnie: BALANCED)
Lekki (Szybki, 50HP)
Zbalansowany (100HP)
Ciężki (Wolny, 200HP, Armor)
START GRY!
WRÓĆ



(Rys 1. Menu główne gry z wyborem statków i sklepem)

5.1. Proceduralny Syntezator Dźwięku (Brak zewnętrznych plików .wav)

Jednym z najciekawszych i najbardziej innowacyjnych elementów projektu, napisanym we współpracy z AI, jest system generowania dźwięku bezpośrednio z funkcji matematycznych, eliminujący potrzebę importowania zewnętrznych plików audio.

```
# --- SYNTEZATOR DŹWIĘKÓW ---
def create_melody(notes_with_durations, volume=0.1, wave_type='sine'):
    sample_rate = 44100
    total_samples = sum(int(dur * sample_rate) for freq, dur in notes_with_durations)
    if total_samples == 0: return pygame.mixer.Sound(buffer=array.array('h', []))
    buf = array.array('h', [0] * total_samples)
    current_sample = 0
    for freq, dur in notes_with_durations:
        n_samples = int(dur * sample_rate)
        max_amp = int(32767 * volume)
        for i in range(n_samples):
            if freq > 0:
                time = i / sample_rate
                if wave_type == 'sine':
                    val = int(max_amp * math.sin(2.0 * math.pi * freq * time))
                else:
                    val = max_amp if math.sin(2.0 * math.pi * freq * time) > 0 else -max_amp
                buf[current_sample] = val
            else:
                buf[current_sample] = 0
            current_sample += 1
    return pygame.mixer.Sound(buffer=buf)

```

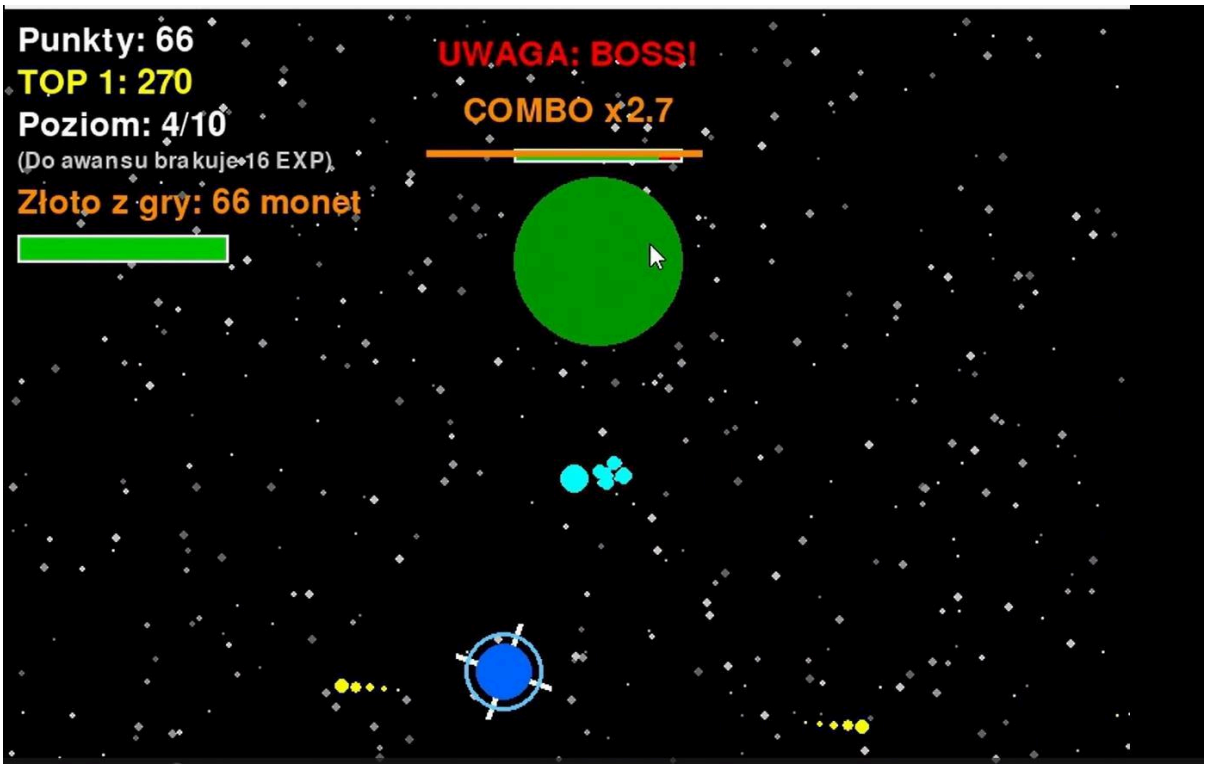
Wyjaśnienie kodu: Zamiast ładować muzykę, skrypt używa biblioteki `math` oraz `array`. Funkcja przyjmuje częstotliwość (Hz) oraz czas trwania. Za pomocą wzoru na falę sinusoidalną (`math.sin(2.0 * math.pi * freq * time)`) tablica wypełniana jest wartościami amplitud z częstotliwością próbkowania 44100 Hz. Pozwala to na stworzenie retro-dźwięków w czasie rzeczywistym.

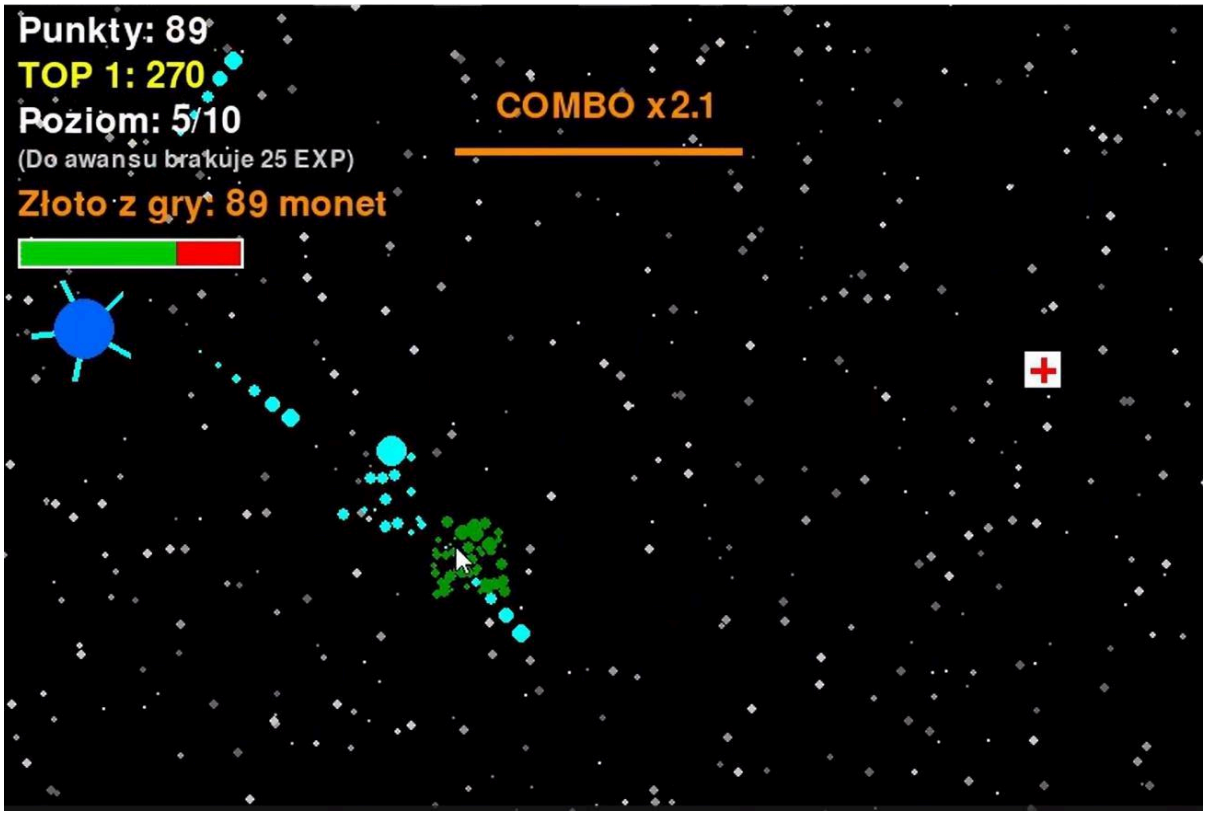
5.2. System Cząsteczek i Trzęsienia Ekranu (Game Feel)

Aby nadać grze dynamiki (tzw. Game Juice), zaimplementowano własny system Particle (klasa `Particle`). Gdy wróg ginie, generowana jest lista małych okręgów o losowych wektorach prędkości (`vx`, `vy`), które z każdą klatką zmniejszają swój promień. Dodatkowo, przy otrzymaniu obrażeń, główna powierzchnia gry (`game_surface`) jest przesuwana o losową wartość, co tworzy efekt trzęsienia ekranu (Screen Shake).

```python
class Particle:
    def __init__(self, x, y, color):
        self.x, self.y = x, y
        self.color = color
        self.vx, self.vy = random.uniform(-3, 3), random.uniform(-3, 3)
        self.radius = random.randint(3, 6)
        self.life = random.randint(20, 40)
    def update(self):
        self.x += self.vx; self.y += self.vy; self.life -= 1
        if self.life % 5 == 0 and self.radius > 0: self.radius -= 1
    def draw(self, surface):
        if self.radius > 0: pygame.draw.circle(surface, self.color, (int(self.x), int(self.y)), self.radius)

```





5.3. Efekt Parallax Scrolling (Pseudogłębia 3D)

Wykorzystano prostą matematykę, aby zasymulować głębię kosmosu. Klasa `ParallaxBackground` dzieli setki wygenerowanych gwiazd na trzy warstwy. Gwiazdy mniejsze są ciemniejsze i poruszają się wolniej w dół osi Y, podczas gdy gwiazdy większe są jaśniejsze i szybsze.

```
# --- EFEKT 3D TŁA (PARALLAX) ---
class ParallaxBackground:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.layers = []
        for i in range(3):
            layer = []
            num_stars = int((width * height) / 3000)
            for _ in range(num_stars):
                layer.append([random.randint(0, width), random.randint(0, height), random.randint(1, 3), (i+1)*0.5])
            self.layers.append(layer)

    def resize(self, width, height):
        self.width = width
        self.height = height

    def update_and_draw(self, surface):
        surface.fill(BLACK)
        for i, layer in enumerate(self.layers):
            color = (100 + i*50, 100 + i*50, 100 + i*50)
            for star in layer:
                star[1] += star[3]
                if star[1] > self.height:
                    star[1] = 0
                    star[0] = random.randint(0, self.width)
                pygame.draw.circle(surface, color, (int(star[0]), int(star[1])), star[2])
```

5.4. Separacja Danych i Meta-progresja

Początkowo gra posiadała jeden licznik punktów. W procesie iteracji zdecydowano o rozdzieleniu punktów rankingowych (Score) od Doświadczenia (EXP). Otrzymanie obrażeń nakłada karę punktową (`shared_score -= 5`), ale nie zmniejsza doświadczenia, zapobiegając frustracji gracza. Dodatkowo wprowadzono walutę (monety) zapisywaną w formacie JSON do słownika, pozwalającą na zakup trwałych ulepszeń.

```python
# --- FUNKCJE POMOCNICZE (Zapis, Odczyt) ---
def load_json(filename, default):
    if os.path.exists(filename):
        try:
            with open(filename, "r", encoding="utf-8") as file: return json.load(file)
        except: return default
    return default

def save_json(filename, data):
    try:
        with open(filename, "w", encoding="utf-8") as file: json.dump(data, file)
    except: pass
```

6. Analiza najciekawszych fragmentów kodu (Architektura)

Aplikacja została zbudowana obiektowo (OOP). Poniżej znajduje się omówienie kluczowych fragmentów kodu, które stanowią o zaawansowaniu technicznym projektu i wymagają głębokiego zrozumienia mechaniki programowania gier.

6.1 Proceduralny Syntezator Dźwięku (Brak plików zewnętrznych)

Zamiast obciążać projekt plikami `.mp3` czy `.wav`, za pomocą biblioteki `math` i `array` zaimplementowano syntezator generujący fale dźwiękowe bezpośrednio na procesorze w czasie rzeczywistym.

Python

```
def create_melody(notes_with_durations, volume=0.1, wave_type='sine'):
    sample_rate = 44100 # Częstotliwość próbkowania
    total_samples = sum(int(dur * sample_rate) for freq, dur in notes_with_durations)
    buf = array.array('h', [0] * total_samples) # Bufor pamięci
    current_sample = 0

    for freq, dur in notes_with_durations:
        n_samples = int(dur * sample_rate)
        max_amp = int(32767 * volume) # Maksymalna amplituda dla 16-bitowego audio
        for i in range(n_samples):
            if freq > 0:
                time = i / sample_rate
                # Wzór na amplitudę fali sinusoidalnej
```

```
            val = int(max_amp * math.sin(2.0 * math.pi * freq * time))
            buf[current_sample] = val
        else:
            buf[current_sample] = 0
        current_sample += 1

    return pygame.mixer.Sound(buffer=buf)
```

Wnioski: Skrypt oblicza wychylenie fali dla każdej z 44100 próbek na sekundę za pomocą funkcji trygonometrycznej sinus (`math.sin`). Dzięki temu gra nie posiada "zewnętrznych paczek" zasobów i może zagrać od razu każdą melodię, np. fanfarę zwycięstwa generowaną na podstawie listy tupli z częstotliwościami.

6.2. Złożony system detekcji kolizji kołowych

Powszechnym podejściem jest używanie kolizji prostokątnych (tzw. AABB - Axis-Aligned Bounding Box), jednak z uwagi na estetykę wektorową (koła), wykorzystano detekcję za pomocą twierdzenia Pitagorasa.

Python
```
def check_collision(x1, y1, r1, x2, y2, r2):
    # Użycie math.hypot jako odpowiednika sqrt((x1-x2)^2 + (y1-y2)^2)
    return math.hypot(x1 - x2, y1 - y2) < (r1 + r2)
```

Wnioski: Metoda `math.hypot` skutecznie oblicza euklidesową odległość między środkami dwóch okręgów. Jeśli odległość ta jest mniejsza niż suma ich promieni (`r1 + r2`), następuje kolizja. Pozwala to na "pixel-perfect" dokładność w grze typu *bullet hell*, gdzie każdy milimetr odskoku od lasera ma znaczenie.

6.3. Dynamiczne zapobieganie overlapowi interfejsu (Strefy Wykluczenia)

Aby rozwiązać problem opisany w rozdziale 4. wdrożono algorytm zabezpieczający pozycje przedmiotów.

Python
```
class PowerUp:
    def __init__(self):
        while True:
            self.x = random.randint(50, WIDTH - 50)
            self.y = random.randint(50, HEIGHT - 50)
            # Upewnienie się, że element nie zespawnuje się pod UI (Lewy i Prawy górny róg)
            if self.x < 320 and self.y < 220: continue
            if self.x > WIDTH - 220 and self.y < 220: continue
            break # Znalazł dobrą, pustą pozycję
```

6.4. Trwały zapis za pomocą modułu JSON (Zarządzanie stanem konta)

Meta-progresja, sklep i rekordy opierają się o serializację struktur danych Pythona (słowniki i listy) do pliku tekstowego w formacie JSON.

```
Python
def load_json(filename, default):
    if os.path.exists(filename):
        try:
            with open(filename, "r", encoding="utf-8") as file: return json.load(file)
        except: return default
    return default

# W kodzie głównym struktura jest tworzona lub ładowana:
save_data = load_json("save_data.json", {"coins": 0, "upgrades": {"hp_bonus": 0,
"start_lvl": 1}})
```

Wnioski: Użycie klauzuli `try-except` jest tu zabezpieczeniem (tzw. fail-safe) przed uszkodzonymi plikami zapisu. Jeśli plik jest pusty lub składnia ulegnie uszkodzeniu, gra awaryjnie wczytuje wartość `default`, chroniąc aplikację przed krytycznym zamknięciem (Crash).

7. Podsumowanie nabytych kompetencji AI i wnioski

Proces tworzenia gry udowodnił, że współczesne modele językowe są potężnym narzędziem w rękach początkującego dewelopera.

Architektura oprogramowania: Dzięki AI zrozumiałem, jak projektować maszyny stanów (Game State Machine) w pętli gry.
Algorytmika: Zrozumiałem implementację kolizji opartych na odległości (twierdzenie Pitagorasa w funkcji `math.hypot`), a także podstawy syntezy fal dźwiękowych.
Krytyczna analiza wyników: Nauczyłem się, że kod generowany przez AI czasami nadpisuje poprzednie funkcje (np. zagubiona klasa Particle w jednej z iteracji), co wymaga uważnego kontrolowania wersji kodu, czytania błędów (Stack Trace) z konsoli i logicznego instruowania modelu o miejscach pomyłki.

8. Wykaz źródeł i materiałów edukacyjnych

Źródła w projekcie:

Kod źródłowy wygenerowany i iterowany we współpracy z asystentem Google Gemini 3 Pro (Deep Think).

Grafika/Wizualizacje w całości opierają się na proceduralnie renderowanej geometrii za pomocą modułu `pygame.draw`.

Wykorzystane materiały edukacyjne (podstawy teoretyczne potrzebne do współpracy z AI):

Oficjalna dokumentacja biblioteki Pygame (https://www.pygame.org/docs/) – niezbędna do zrozumienia składni rysowania powierzchni, obsługi wejścia/wyjścia (event queue) i skalowania okna.

Dokumentacja języka Python (Moduły: json, math, random) – zrozumienie operacji na plikach oraz funkcji trygonometrycznych do tworzenia pocisków latających pod kątem.

Artykuły z zakresu Game Designu i "Game Juice" (np. wykład wideo Juice it or lose it), które zainspirowały komendy (prompty) o dodanie zjawiska screen-shake'a oraz klatek nieśmiertelności (i-frames)