|  |
| --- |
|  |
|  |
|  |
|  |
|  |

**IMPLEMENTACJA GRY Z GRAFIKĄ**

Autor: Jakub Domagała

Akademia Górniczo-Hutnicza im. Stanisława Staszica   
w Krakowie

Kraków 2025

Spis treści

[Wstęp 3](#_Toc201330373)

[1. Funkcjonalność (*functionality*) 5](#_Toc201330374)

[2. Projekt techniczny (*technical design*) 6](#_Toc201330375)

[2.1 main.py – główna logika gry i pętla sterująca 11](#_Toc201330376)

[2.2 renderer.py – warstwa wizualna gry 12](#_Toc201330377)

[2.3 actors.py – Postać i fizyka ruchów 12](#_Toc201330378)

[2.4 constants.py – stałe konfiguracyjne 13](#_Toc201330379)

[3. Opis realizacji (*implementation report*) 14](#_Toc201330380)

[4. Opis wykonanych testów (*testing report*) - lista buggów, uzupełnień 16](#_Toc201330381)

[5. Podręcznik użytkownika (*user's manual*) 18](#_Toc201330382)

[5.1 Uruchamianie gry 18](#_Toc201330383)

[6. Metodologia rozwoju i utrzymania systemu (*system maintenance and deployment*) 19](#_Toc201330384)

[Bibliografia 21](#_Toc201330385)

# Wstęp

Gra „Student Tower” powstała z inspiracji klasyczną grą Icy Tower, która znana jest z lat dzieciństwa, która dostarczała wielu godzin wciągającej rozgrywki. Moim głównym celem było odtworzenie tej gry w charakterystycznym, studenckim klimacie – zamiast lodowego świata sterujemy ambitnym studentem wspinającym się po coraz wyższych szczeblach Akademii Górniczo-Hutniczej, który w trakcie wspinaczki może napotkać na symbol studentów – piwo, które po zebraniu daje dodatkową ilość punktów..

**Wymagania systemowe (*requirements*)**

**Środowisko uruchomieniowe:**

* **Python 3.7+** (zalecane 3.8–3.10) – interpreter CPython, ponieważ wykorzystywane są moduły standardowe turtle, random, os oraz (na Windows) winsound.
* **System operacyjny**:
  + **Windows** (pełne wsparcie winsound),
  + **Linux / macOS** (zamiast winsound dźwięki można zastąpić biblioteką import os oraz zamienić w linii kodu winsound na os.system(„aplay nazwapliku.wav&”) w przypadku Linuxa lub os.system(„afplay nazwapliku.wav&”) w przypadku MacOS).
* **Ekran o rozdzielczości co najmniej 800×600** (zaprojektowane pod 800×1000)
* **Biblioteki i zasoby**:
  + Moduły standardowe: turtle, random, os.
  + Grafiki w folderze gry:
    - tła: backgroundAGH.gif (lub alternatywne background.gif natomiast należało by zamienić ręcznie w kodzie)),
    - platformy: plat.gif, plat100.gif, …, plat240.gif,
    - postać: student2.gif, studentprawo.gif, studentlewo.gif,
    - obroty: 45.gif, 90.gif, …, 315.gif, 180.gif,
    - bonusy/monety: bonus.gif (lub alternatywnie przy braku gifa pojawia się zwykła złota moneta).
  + Dźwięki (Windows): cartoonjump.wav, yay.wav.

**Wydajność**

* Gra powinna działać w stałych 60 FPS (FRAME\_TIME = 1000/60)
* Użycie \_\_slots\_\_, cachowanie metod (xcor, ycor) i minimalizacja wywołań graficznych (screen.update()) zapewnia płynność nawet przy dużej liczbie obiektów (~30 platform + bonusy + gwiazdki).

# Funkcjonalność (*functionality*)

1. **Poruszanie postacią**
   * Gracz steruje lądowaniem i skakaniem studenta za pomocą strzałek ← i → oraz spacji.
   * Fizyka ruchu uwzględnia przyspieszenie, tarcie, grawitację oraz odbicia od ścian.
2. **Generowanie i recykling platform**
   * Na początku tworzone są platformy (w tym podłoże) o losowych długościach i pozycjach poziomych.
   * Po przekroczeniu przez gracza pewnej wysokości świat „przewija się” w dół, a platformy, które wypadną poza ekran, są „recyklingowane” na górze.
3. **System punktacji**
   * Za każde kolejne piętro studentowi przyznawane jest 100 punktów.
   * Co 3000 pkt prędkość przewijania świata zwiększa się (trudność rośnie).
   * Wynik wyświetlany jest w czasie rzeczywistym w rogu ekranu.
4. **Efekty cząsteczkowe (“gwiazdki”)**
   * Podczas wysokich skoków generowane są małe obiekty typu Star, które opadając, wirują i znikają poza ekranem.
5. **Bonusowe monety**
   * Z prawdopodobieństwem 25 % losowo pojawiają się obiekty Bonus (piwo) na platformach w sposób dynamiczny powyżej gracza (ustawiono że co najmniej 1.5 platformy nad wysokością gracza aby nie było przypadku że bonus będzie pojawiał się poniżej gracza).
   * Kolizja z bonusem dodaje graczowi 500 punktów, a bonus znika.
   * Bonusy są recyklingowane lub usuwane, gdy wypadną poza ekran.
6. **Restart gry**
   * Po spadnięciu poniżej ekranu gra wyświetla ekran „Koniec gry” z przyciskiem „Zagraj ponownie”.
   * Po kliknięciu ekran jest czyszczony, a wszystkie obiekty i stan gry są ponownie inicjalizowane.

# Projekt techniczny (*technical design*)

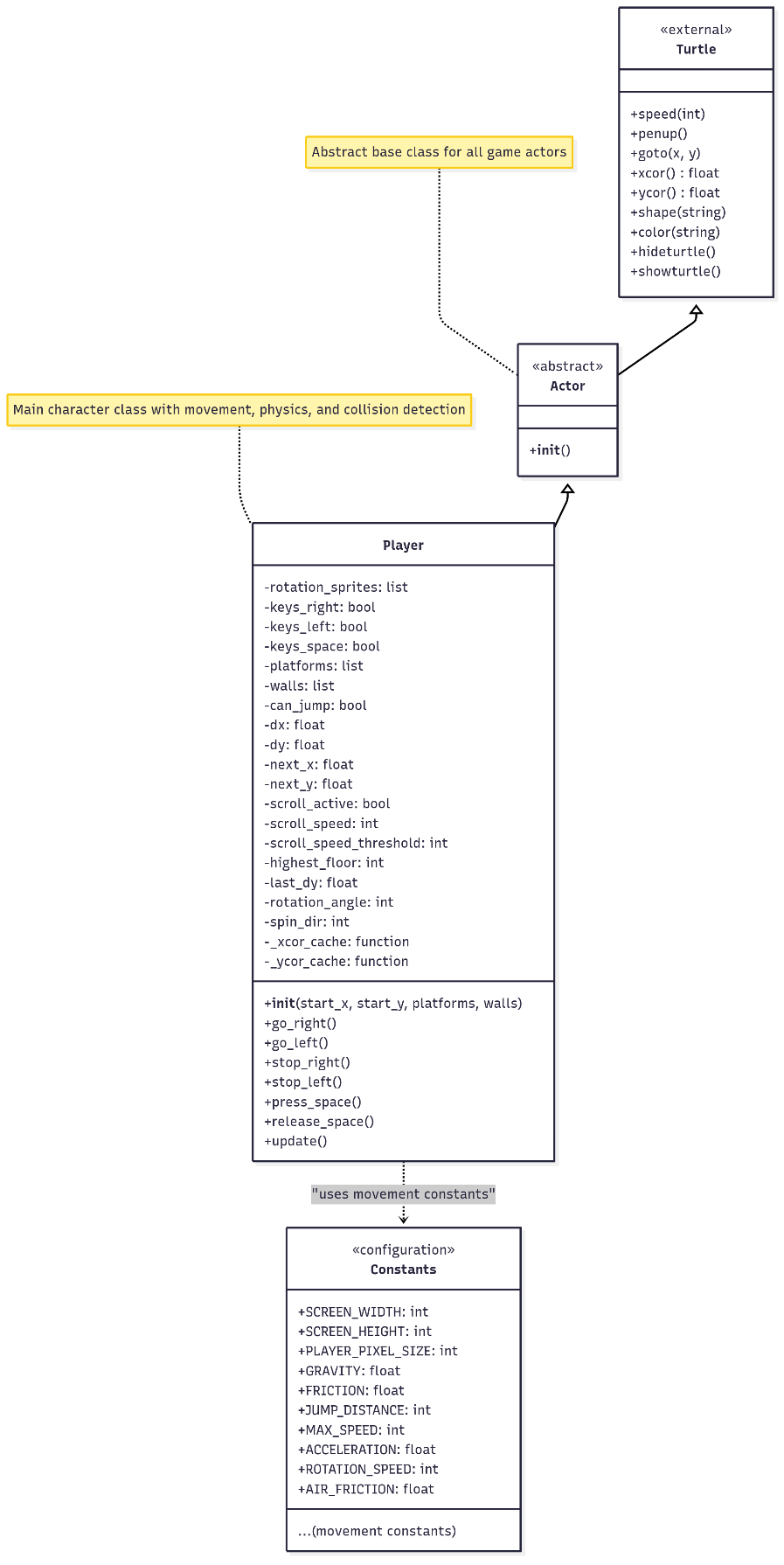
Diagram 1:

Diagram 2:

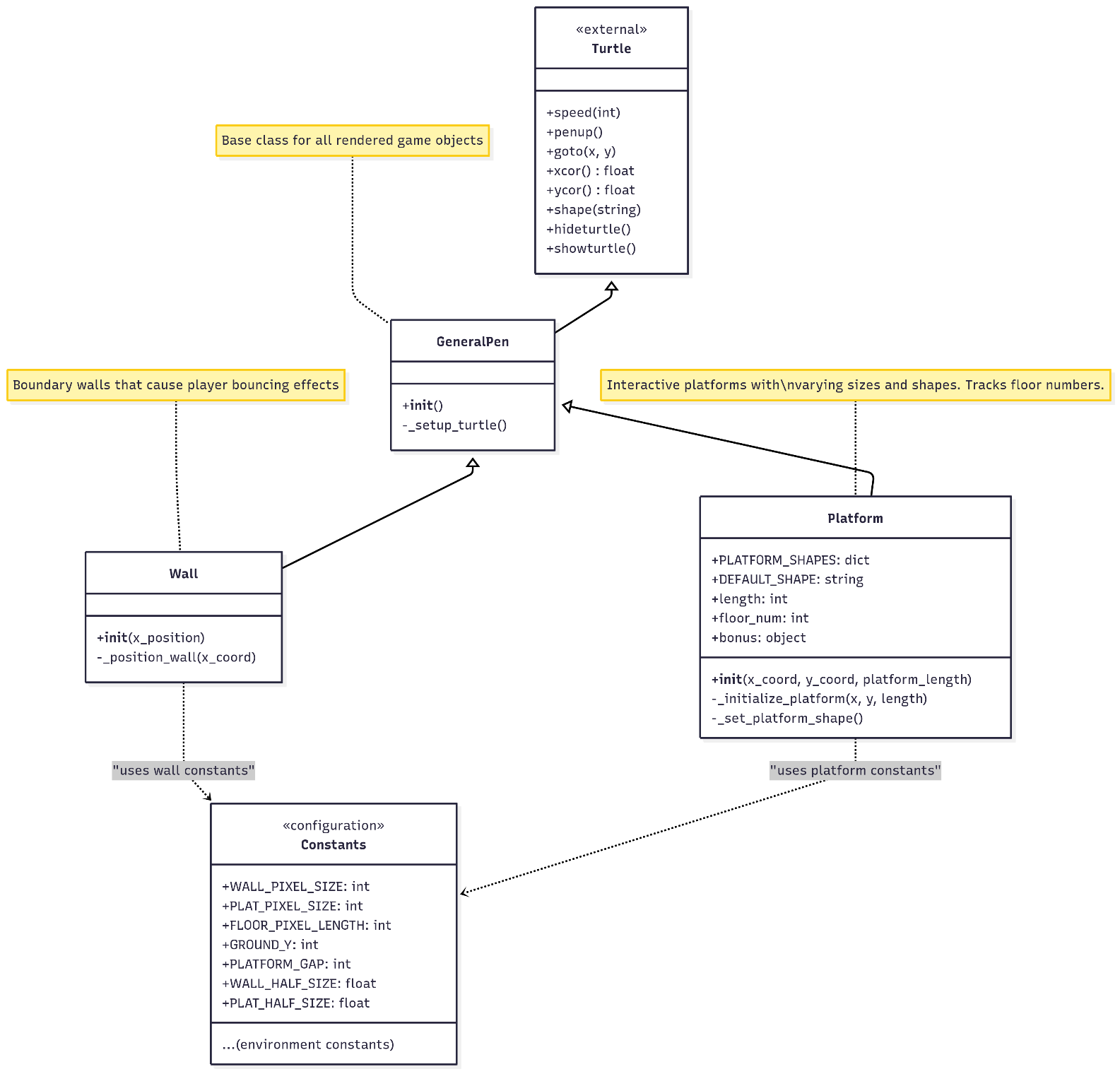


Diagram 3:

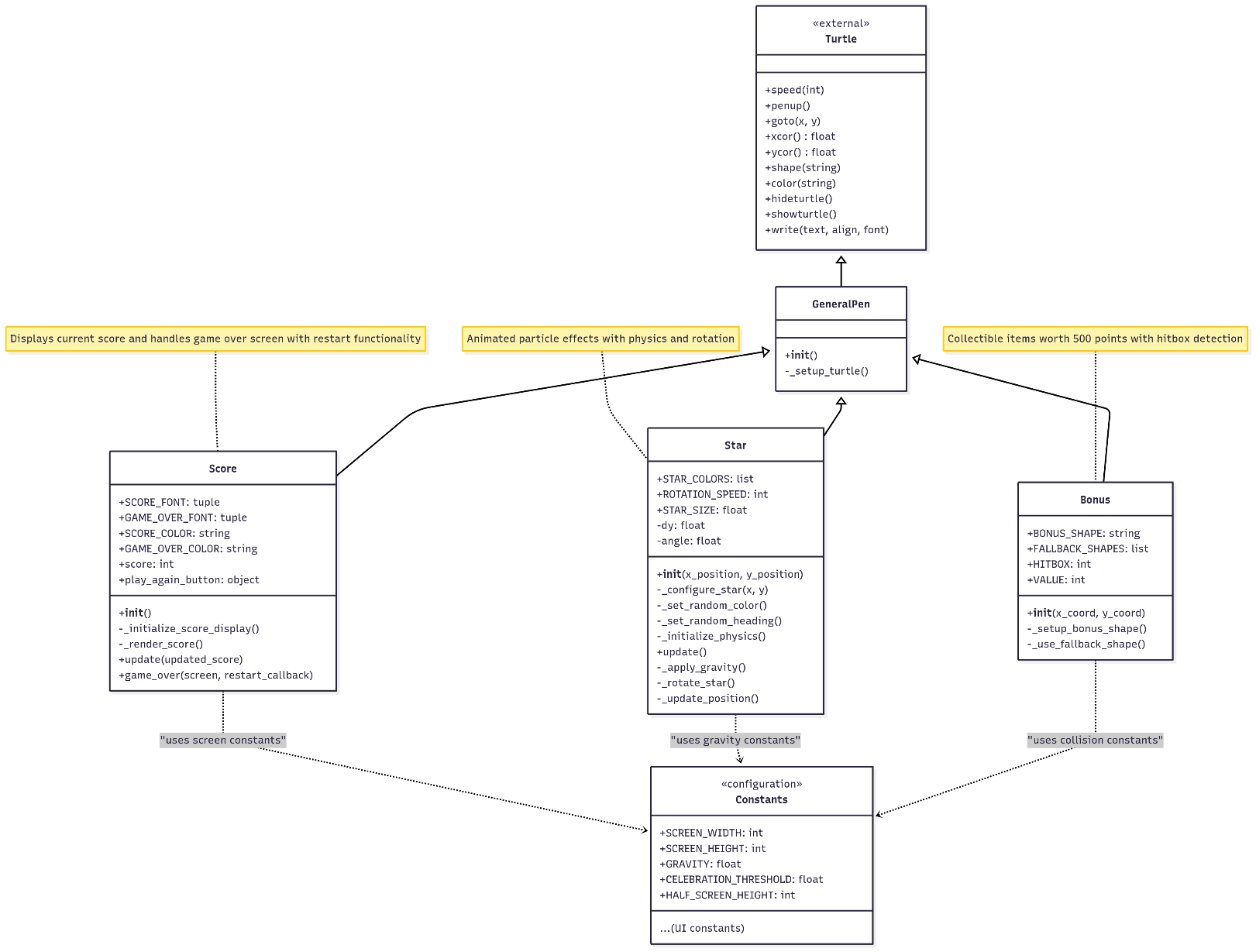
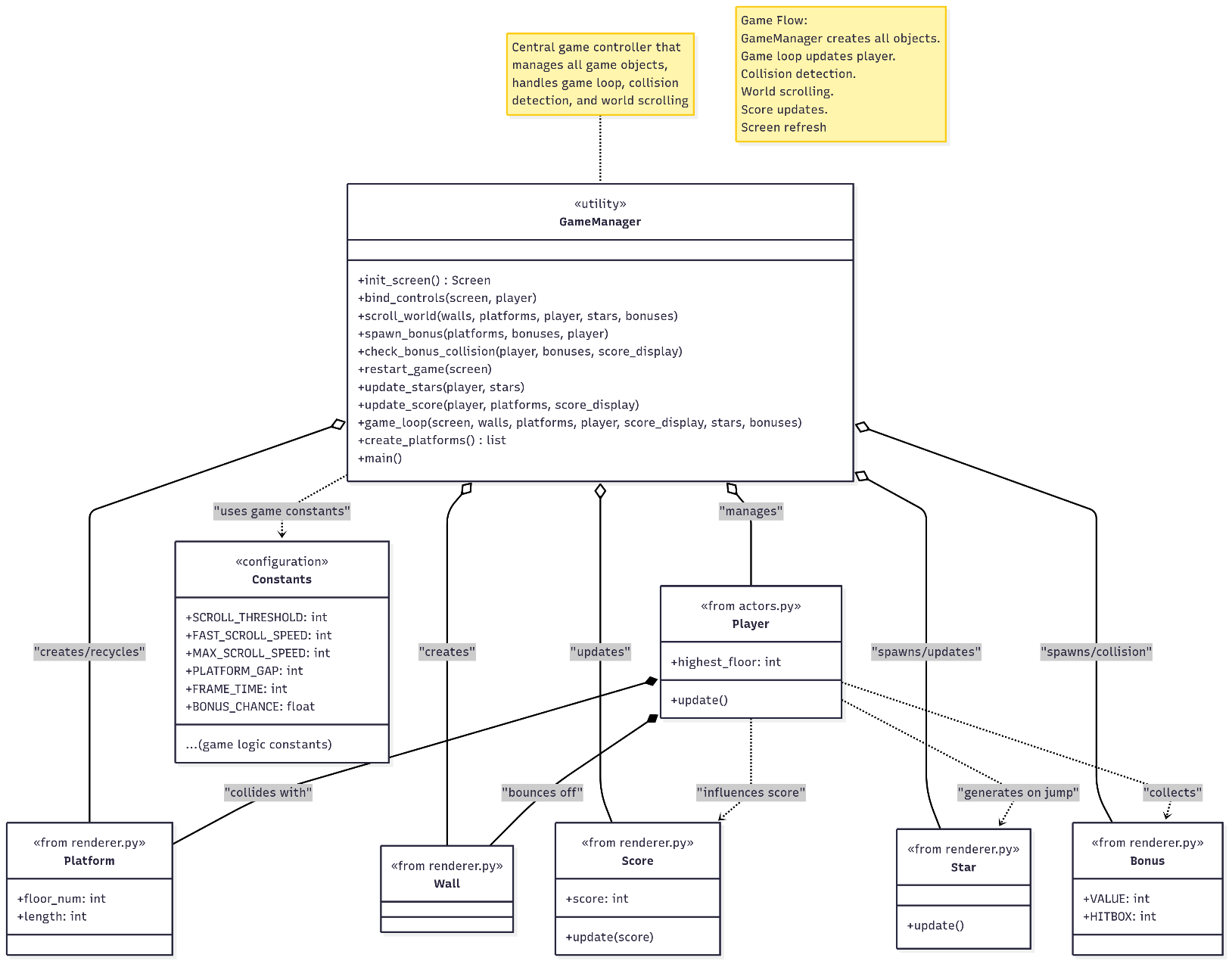


Diagram 4:



**Diagram 1: Główny System Aktora**

* Postać gracza i mechanika ruchu
* Klasa bazowa Turtle, klasa abstrakcyjna Actor, klasa Player oraz stałe związane z ruchem
* Pokazuje główny system postaci z obsługą fizyki i sterowania

**Diagram 2: Obiekty Środowiska Gry**

* Statyczne elementy świata gry
* Bazowa klasa GeneralPen, klasy Wall i Platform oraz stałe środowiskowe
* Pokazuje strukturalne elementy tworzące świat gry

**Diagram 3: System Interfejsu i Efektów Wizualnych**

* Informacje wizualne i interfejs użytkownika
* Wyświetlanie wyniku (Score), efekty cząsteczkowe (Star), bonusy (Bonus) oraz stałe interfejsowe
* Przedstawia wszystkie elementy wizualne, które przekazują informacje zwrotne graczowi

**Diagram 4: System Zarządzania Grą**

* Logika gry i relacje między obiektami
* Klasa pomocnicza GameManager oraz sposób, w jaki wszystkie komponenty współdziałają
* Pokazuje ogólną architekturę systemu i przepływ gry

## main.py – główna logika gry i pętla sterująca

Centralny moduł uruchamiający grę. Obsługuje inicjalizację, tworzy wszystkie obiekty, ustawia zdarzenia klawiatury, oraz zawiera pętlę game\_loop() sterującą całym przebiegiem gry.

**Kluczowe funkcjonalności:**

* Tworzenie ekranu i sceny gry (turtle.Screen)
* Inicjalizacja gracza (Player)
* Tworzenie platform i ścian
* Obsługa bonusów i gwiazdek (efektów)
* Pętla gry: fizyka, przewijanie, kolizje, zbieranie bonusów, aktualizacja punktów
* Reakcja na klawisze (onkeypress)

**Główne funkcje:**

* create\_walls(screen) – tworzy boczne ograniczenia planszy
* create\_initial\_platforms(screen) – generuje platformy startowe na różnych piętrach
* create\_score(screen) – inicjalizuje obiekt punktacji
* scroll\_world(...) – przewija planszę i recykluje platformy gdy gracz się wznosi
* game\_loop() – funkcja uruchamiana cyklicznie (co 16ms) jako główna pętla gry

Korzysta z klas: Player, Wall, Platform, Score, Star, Bonus, oraz importuje stałe z constants.py

## renderer.py – warstwa wizualna gry

Zawiera wszystkie klasy odpowiadające za **grafikę i wyświetlanie** elementów gry na ekranie przy użyciu biblioteki turtle.

**Klasa bazowa:**

* **GeneralPen** – podstawowy długopis graficzny (dziedziczy po turtle.Turtle)

Konfiguruje wspólne cechy: speed(0), penup(), hideturtle(), tracer(0), brak animacji

**Klasy pochodne:**

* **Wall** – pionowa przeszkoda boczna
* **Platform** – pozioma platforma, na której ląduje gracz

Posiada pole bonus, które może wskazywać na Bonus

* **Score** – licznik punktów, wyświetlany w górnej części ekranu
* **Star** – efekt cząsteczkowy przy skokach
* **Bonus** – graficzna reprezentacja obiektów do zebrania

**Inne:**

* Każda klasa ustawia kształt (sprite) przez shape() i pozycję przez goto()
* Często używana jest metoda clone() lub clear() do aktualizacji grafiki

Wszystkie klasy są używane przez main.py i częściowo przez actors.py

## actors.py – Postać i fizyka ruchów

Zawiera abstrakcyjną klasę Actor oraz klasę Player – odpowiadające za ruch, fizykę, kolizje, grawitację, interakcje z otoczeniem.

**Klasy: Actor**

* Klasa abstrakcyjna – wspólne cechy dla obiektów dynamicznych (gracz, NPC, itp.)
* Ustawia \_\_slots\_\_, aby ograniczyć pamięć (optymalizacja)
* Pola: dx, dy, gravity, sprite\_frames
* Metody: goto(x, y), get\_position(), collides\_with(obj)

**Player(Actor):**

* Dziedziczy po Actor
* Obsługuje wejście klawiatury (flagi: keys\_left, keys\_right, keys\_space)
* Posiada metodę update(), która:
  + Zmienia pozycję w oparciu o dx/dy
  + Zderza się z platformami i ścianami
  + Sprawdza możliwość skoku (can\_jump)
  + Dodaje efekty cząsteczkowe (Star) przy dużej prędkości
  + Aktualizuje animację sprite’ów

Wykorzystuje klasy z renderer.py (do kolizji). Używana wyłącznie przez main.py

## constants.py – stałe konfiguracyjne

Zawiera **wszystkie stałe** liczbowo-opisowe, które wpływają na zachowanie gry. Rozdziela logikę od parametrów, dzięki czemu łatwiej dostrajać balans gry i zachowanie fizyki.

**Kategorie:**

**Ruch gracza:**

* PLAYER\_SPEED – maksymalna prędkość poruszania się gracza w poziomie
* GRAVITY – wartość przyspieszenia grawitacyjnego
* JUMP\_STRENGTH – siła skoku gracza
* MAX\_FALL\_SPEED – maksymalna prędkość opadania
* SCROLL\_THRESHOLD – wysokość, przy której zaczyna się przewijanie planszy

**Środowisko:**

* PLATFORM\_WIDTH\_MIN/MAX – minimalna i maksymalna długość platform
* NUM\_INITIAL\_PLATFORMS – ile platform pojawia się na starcie
* PLATFORM\_SPACING – odległość między platformami (pionowa)
* WALL\_WIDTH – szerokość bocznych ścian

**Bonusy:**

* BONUS\_SPAWN\_CHANCE – szansa pojawienia się bonusu na nowej platformie
* BONUS\_VALUE – liczba punktów za jeden bonus

**Inne:**

* CELEBRATION\_THRESHOLD – próg prędkości skoku, po którym uruchamia się efekt gwiazdki
* FRAME\_DELAY – opóźnienie pomiędzy klatkami gry (w ms)

# Opis realizacji (*implementation report*)

**Środowisko programistyczne i testowe:**

Projekt został zrealizowany i testowany w środowisku lokalnym, bez użycia zewnętrznych serwerów czy kontenerów. Jako podstawę środowiska przyjęto klasyczną konfigurację systemu desktopowego.

**Platforma testowa:**

* **System operacyjny:** Windows 11
* **Procesor:** Intel Core i5
* **RAM:** 16 GB
* **Rozdzielczość ekranu:** 1920×1080
* **Python:** wersja 3.10+

**Narzędzia programistyczne:**

* **Edytor kodu:** PyCharm
* **Interpreter:** Python 3.12
* **System wersjonowania:** Git
  + Repozytorium lokalne oraz zdalne (GitHub)

**Debugowanie:**testowanie wizualne poprzez interakcję z oknem turtle

**Dodatkowe narzędzia:**

* **PlantUML** – do wizualizacji klas i zależności

**Backup:** backup na nośniku zewnętrznym

**Proces implementacji**

Projekt powstawał iteracyjnie, zgodnie z uproszczonym cyklem przyrostowym. Wprowadzono kolejne moduły niezależnie i testowano je osobno przed zintegrowaniem całości.

**Etapy implementacji:**

1. **Projekt systemu aktorów** (Actor, Player) oraz podstawowej fizyki ruchu.
2. **Tworzenie statycznych obiektów otoczenia** (Platform, Wall) i ich losowego rozmieszczania.
3. **Wprowadzenie systemu graficznego** z wykorzystaniem turtle (renderer.py).
4. **Dodanie interakcji** – kolizje, punkty, bonusy, przewijanie planszy.
5. **System UI i efektów wizualnych** – licznik punktów, animacje gwiazdek.
6. **Integracja i testy całości** w main.py.

**Napotkane problemy i wyzwania**

Podczas implementacji wystąpiły następujące wyzwania techniczne, które zostały zidentyfikowane jako kluczowe i wymagały rozwiązania:

**Próba określenia funkcji przewijania planszy:**

* Trudność: ustalenie momentu, w którym plansza powinna się przewijać w pionie, gdy gracz osiąga pewien próg wysokości (SCROLL\_THRESHOLD)
* Rozwiązanie: zastosowanie przesunięcia wszystkich obiektów w dół oraz dynamiczne usuwanie/nadpisywanie starych platform

**Próba matematycznego zamodelowania funkcji skoku:**

* Trudność: uzyskanie realistycznego toru lotu przy prostym modelu fizyki (grawitacja, prędkość pionowa)
* Rozwiązanie: wprowadzenie ograniczenia maksymalnej prędkości opadania oraz stałej siły skoku (JUMP\_STRENGTH) z uwzględnieniem gravity

**Zbudowanie cyfrowego generatora funkcji platform:**

* Trudność: zapewnienie losowości rozmieszczenia platform z jednoczesnym zachowaniem grywalności (odstępy, długość, wysokość)
* Rozwiązanie: zaprogramowanie funkcji generującej platformy o długości w zakresie (PLATFORM\_WIDTH\_MIN, PLATFORM\_WIDTH\_MAX) i równomiernym rozkładzie.

**Podsumowanie realizacji**

Projekt został zakończony z sukcesem, osiągając założone podstawowe cele funkcjonalne. Wszystkie elementy gry (fizyka, interfejs, efekty wizualne, przewijanie, logika punktów) zostały w pełni zintegrowane w spójny system.

W dalszej perspektywie możliwa jest rozbudowa o:

* system poziomów trudności,
* więcej typów przeszkód,
* zapis postępu gry,
* portowanie na inną bibliotekę (np. pygame).
* Dodanie menu głównego
* Dodanie zapisywania najlepszych uzyskanych wyników
* Zmiana GUI wraz z poziomami
* Większa ilość PowerUp-ów (urozmaicenie rozgrywki)
* Dodanie menu głównego z możliwością wyboru poziomu lub sprawdzeniem dotychczasowych najlepszych wyników
* Aktualizacja i zrobienie „bardziej przyjemnego dla oka” przycisku zagraj ponownie

# Opis wykonanych testów (*testing report*) - lista buggów, uzupełnień

**Metodyka testowania**

Testowanie gry odbywało się manualnie oraz poprzez obserwację zachowania w czasie rzeczywistym w oknie graficznym turtle.Screen(). Ze względu na interakcyjny charakter gry, nie zastosowano frameworków do testów jednostkowych (np. unittest), lecz skupiono się na testach integracyjnych i funkcjonalnych w trakcie rzeczywistej rozgrywki.

**Testowano następujące aspekty:**

* **Ruch i fizyka gracza**: skok, kolizje, grawitacja
* **Działanie planszy**: generacja i przewijanie platform
* **Zbieranie punktów i bonusów**
* **Poprawność wyświetlania elementów (UI, gwiazdki, napisy)**
* **Stabilność działania gry przy dłuższym uruchomieniu**
* **Reakcja na błędne działania (np. wyjście poza ekran)**

**Napotkane błędy:**

* Gracz przemieszczał się poza ekran bocznie - ograniczono pozycję x gracza do szerokości ekranu
* Brak przewijania planszy przy skoku w górę - dodano SCROLL\_THRESHOLD i scroll\_environment()
* Platformy generowały się poza ekranem - dodano ograniczenia losowej pozycji x i y.
* Bonusy nie znikały po zebraniu - dodano flagę collected i usuwanie z planszy
* Bonusy po zebraniu dodawały 500 punktów które zostało odejmowane po następnym skoku – zaktualizowano funkcje update\_score() w celu wyeliminowania błędu
* Brak resetu po game over – dodano funkcję reset\_game() i restart

# Podręcznik użytkownika (*user's manual*)

## Uruchamianie gry

**Wymagania:**

* Python 3.10 lub nowszy
* Biblioteka turtle (standardowa w Pythonie)
* Pliki: main.py, actors.py, renderer.py, constants.py (wszystkie muszą być w jednym folderze)

**Sposób uruchomienia:**

1. Otwórz terminal / konsolę.
2. Przejdź do katalogu z plikami gry.
3. Uruchom grę komendą: python main.py

**Sterowanie**

* ← (lewa strzałka) – ruch w lewo
* → (prawa strzałka) – ruch w prawo
* ↑ (górna strzałka lub spacja) – skok
* ESC – zakończenie gry (zamknięcie okna)

**Zasady gry**

* Postać automatycznie podlega grawitacji – trzeba utrzymywać się na platformach.
* Skok pozwala przemieszczać się w górę i unikać upadku.
* Dotknięcie platformy od góry umożliwia dalsze skakanie.
* Zbieraj bonusy i gwiazdki, aby zwiększyć wynik.
* Gra kończy się, gdy gracz spadnie poza ekran.

# Metodologia rozwoju i utrzymania systemu (*system maintenance and deployment*)

**Podejście do rozwoju systemu**

Projekt gry był rozwijany zgodnie z podejściem iteracyjnym, polegającym na stopniowym dodawaniu kolejnych funkcjonalności i natychmiastowym testowaniu ich działania w środowisku rzeczywistym. W ramach każdej iteracji implementowano jedną lub kilka funkcji (np. sterowanie, grawitacja, kolizje), a następnie integrowano je z już istniejącym kodem.

Główne etapy rozwoju:

* Prototypowanie rdzenia gry (klasa gracza, grawitacja, sterowanie)
* Tworzenie elementów otoczenia (platformy, ściany)
* Dodanie efektów wizualnych (bonusy, gwiazdki)
* Implementacja interfejsu (wynik, ekran końca gry)
* Refaktoryzacja i optymalizacja kodu
* Testowanie i poprawa błędów

**Organizacja kodu**

Kod podzielono logicznie na kilka plików:

* main.py – główna pętla gry, inicjalizacja obiektów i obsługa zdarzeń
* actors.py – definicje postaci, platform, elementów gry (OOP)
* renderer.py – logika rysowania obiektów i odświeżania ekranu
* constants.py – wartości konfiguracyjne (np. prędkości, kolory, rozmiary)

Takie podejście ułatwia utrzymanie i rozwój gry w przyszłości.

**Narzędzia i środowisko**

* **Język programowania**: Python 3.12
* **IDE**: PyCharm
* **Biblioteki**: turtle, random, time
* **System kontroli wersji**: Git
* **Backup danych**: zwenętrzny dysk
* **System operacyjny**: Windows 11

**Utrzymanie systemu**

Kod gry jest lekki, łatwy do uruchomienia i nie wymaga instalacji zewnętrznych bibliotek, dzięki czemu jego utrzymanie sprowadza się do:

* Aktualizacji wersji Pythona i testów kompatybilności
* Przeglądu i usuwania błędów zgłoszonych przez użytkowników/testerów
* Dodawania nowej zawartości (np. poziomów, przeciwników, power-upów)
* Dokumentowania zmian (README, changelog, komentarze w kodzie)

Kod został udokumentowany i podzielony zgodnie z konwencją, co ułatwia jego dalsze utrzymanie i rozbudowę, nawet przez inne osoby.

# Bibliografia

1. **Wzór projektu** -<https://www.youtube.com/watch?v=Y_aMexQCJSA&ab_channel=Micha%C5%82Kozie%C5%82>
2. **Python Documentation Team**. *turtle — Turtle graphics documentation*.  
   <https://docs.python.org/3/library/turtle.html>
3. GitHub z projektem - <https://github.com/jakdom02/studenttower>