Space Invaders

Gabriel Walak

PyCharm 2023.3.4 Python 3.12

Biblioteka pygame

Opis:

"Space Invaders" to klasyczna gra typu shoot 'em up, w której gracz kontroluje statek kosmiczny i musi bronić się przed falami nadciągających wrogów. Gra polega na niszczeniu wszystkich wrogów, zanim ci dotrą do dolnej części ekranu. Gracz może poruszać się w lewo i w prawo oraz strzelać pociskami w celu eliminacji przeciwników.

 **Cel gry:**

* Zniszcz wszystkich wrogów, zanim dotrą do dolnej części ekranu.
* Zdobywaj punkty za każdego zniszczonego wroga.

 **Sterowanie:**

* **Lewy klawisz strzałki:** Przesunięcie statku w lewo.
* **Prawy klawisz strzałki:** Przesunięcie statku w prawo.
* **Spacja:** Wystrzelenie pocisku.

 **Mechanika gry:**

* Wrogowie poruszają się w lewo i w prawo oraz stopniowo zbliżają się do dolnej części ekranu.
* Jeśli wróg dotrze do poziomu gracza, gra kończy się (Game Over).
* Gracz może strzelać pociskami, aby zniszczyć wrogów.
* Po zniszczeniu wszystkich wrogów na ekranie, pojawiają się nowi wrogowie

 **Koniec gry:**

* Gra kończy się, gdy wróg dotrze do poziomu gracza.
* Po zakończeniu gry wyświetlany jest komunikat "GAME OVER".

**Wymagania funkcjonalne**

1. **Ruch statku gracza:**
   * Gracz może poruszać statkiem kosmicznym w lewo i w prawo za pomocą klawiszy strzałek.
   * Ruch statku gracza jest ograniczony do granic ekranu.
2. **Strzelanie:**
   * Gracz może strzelać pociskami za pomocą spacji.
   * Pociski poruszają się w górę ekranu.
3. **Generowanie wrogów:**
   * Wrogowie są generowani w grupach i poruszają się w lewo i w prawo, zbliżając się stopniowo do dolnej części ekranu.
   * W czasie niszczenia wrogów pojawiają się nowi wrogowie
4. **Kolizje:**
   * Pociski wystrzelone przez gracza mogą niszczyć wrogów po trafieniu.
   * Jeśli wróg dotrze do dolnej części ekranu (poziomu gracza), gra kończy się.
5. **Koniec gry:**
   * Po zakończeniu gry wyświetla się komunikat "GAME OVER".
   * Gra kończy się, gdy wróg dotrze do poziomu gracza.
6. **Interfejs użytkownika:**
   * Ekran gry ma rozdzielczość 800x600 pikseli.
   * Na ekranie wyświetlany jest statek gracza, wrogowie oraz pociski.

**Wymagania niefunkcjonalne**

1. **Wydajność:**
   * Gra powinna działać płynnie przy 60 klatkach na sekundę (FPS).
   * Czas reakcji na naciśnięcie klawisza powinien być minimalny.
2. **Skalowalność:**
   * Kod powinien być modularny, umożliwiając łatwe dodawanie nowych funkcji, takich jak nowe typy wrogów czy bronie.
3. **Niezawodność:**
   * Gra nie powinna się niespodziewanie zawieszać ani wyłączać.
4. **Użyteczność:**
   * Interfejs użytkownika powinien być intuicyjny i łatwy w obsłudze.
   * Gracz powinien być w stanie szybko nauczyć się sterowania.
5. **Zgodność:**
   * Powinna być zgodna z najnowszą wersją biblioteki pygame.

#### Struktura projektu

1. **Główne pliki:**
   * main.py: Plik uruchamiający grę.
   * Graphics.py: Plik zawierający klasę Graphics, odpowiedzialną za interfejs graficzny i obsługę gry.
   * Game.py: Plik zawierający klasę Game, zarządzającą logiką gry.
   * Engine.py: Plik zawierający klasę Engine, zarządzającą mechaniką gry.
   * Objects.py: Plik zawierający klasę Objects, bazową klasę dla wszystkich obiektów w grze.
   * Gracz.py: Plik zawierający klasę Gracz, reprezentującą statek gracza.
   * Wróg.py: Plik zawierający klasę Wróg, reprezentującą wrogów.
   * Pocisk.py: Plik zawierający klasę Pocisk, reprezentującą pociski wystrzeliwane przez gracza.
2. **Klasa Graphics (API)**
   * Odpowiada za wyświetlanie grafiki i interfejsu użytkownika.
   * Przechwytuje i obsługuje dane wejściowe od gracza (klawiatura).
   * Wyświetla komunikat "GAME OVER" po zakończeniu gry.
   * Rysuje wszystkie obiekty na ekranie (gracza, wrogów, pociski).
3. **Klasa Game**
   * Dziedziczy po klasie Engine.
   * Odpowiada za główną pętlę gry.
   * Wywołuje metody odpowiedzialne za aktualizację stanu gry i renderowanie grafiki.
4. **Klasa Engine**
   * Zarządza logiką gry i mechaniką działania.
   * Tworzy wrogów i zarządza ich ruchem.
   * Obsługuje kolizje między pociskami a wrogami.
   * Zarządza stanem gry i sprawdza warunki zakończenia gry.
5. **Klasa Objects**
   * Bazowa klasa dla wszystkich obiektów w grze.
   * Zawiera wspólne atrybuty i metody dla obiektów (np. rysowanie, aktualizacja).
6. **Klasa Gracz**
   * Dziedziczy po klasie Objects.
   * Reprezentuje statek gracza.
   * Odpowiada za ruch gracza.
7. **Klasa Wróg**
   * Dziedziczy po klasie Objects.
   * Reprezentuje wrogów.
   * Zarządza ruchem wrogów i ich reakcją na kolizje.
8. **Klasa Pocisk**
   * Dziedziczy po klasie Objects.
   * Reprezentuje pociski wystrzeliwane przez gracza.
   * Zarządza ruchem pocisków i ich reakcją na kolizje.

Opis metod

Utwórz\_wrogów

1. Zewnętrzna pętla for y in range(2)::
   * Ta pętla wykonuje się dla wartości y od 0 do 1 (czyli dwukrotnie).
   * Jest odpowiedzialna za iterację po osi Y, co oznacza, że iteruje przez dwie linie wrogów.
2. Wewnętrzna pętla for x in range(10)::
   * Ta pętla wykonuje się dla wartości x od 0 do 9 (czyli dziesięciokrotnie).
   * Jest odpowiedzialna za iterację po osi X, co oznacza, że iteruje przez dziesięć kolumn wrogów.

Mnożenie x \* 50 + 60 i y \* 50 + 50 służy do ustalenia pozycji (w pikselach) każdego wroga na ekranie. Oto jak to działa:

* Każdy wrog znajduje się w prostokącie o szerokości 40 i wysokości 30 pikseli.
* Na każdej linii (wartość y) wrogowie są oddaleni o 50 pikseli wzdłuż osi Y.
* W każdej kolumnie (wartość x) wrogowie są oddaleni o 50 pikseli wzdłuż osi X.
* Wartość 60 dodawana do x \* 50 służy do dodania przesunięcia w prawo, aby wrogowie nie zaczynali się od lewej krawędzi okna.
* Wartość 50 dodawana do y \* 50 służy do dodania przesunięcia w dół, aby wrogowie nie zaczynali się od górnej krawędzi okna.

Podsumowując, te pętle iterują przez dwuwymiarową siatkę, w której każdy wrog ma określoną pozycję na ekranie, ustaloną na podstawie wartości x i y, z uwzględnieniem przesunięć i odstępów między wrogami.

Strzelaj

 Pobranie aktualnego czasu: Metoda pygame.time.get\_ticks() zwraca liczbę milisekund od uruchomienia gry. Jest to wykorzystywane do monitorowania czasu między kolejnymi strzałami.

 Sprawdzenie, czy upłynął wystarczający czas od ostatniego strzału: Warunek aktualny\_czas - self.czas\_ostatniego\_strzału > self.opóźnienie\_strzału sprawdza, czy różnica między aktualnym czasem a czasem ostatniego strzału jest większa niż opóźnienie strzału. Jeśli tak, gracz może wystrzelić kolejny pocisk.

 Utworzenie nowego pocisku: Jeśli warunek jest spełniony, tworzony jest nowy obiekt Pocisk. Parametry konstruktora to:

* Pozycja X pocisku: self.gracz.x + 22. Pocisk zaczyna się nieco po prawej stronie gracza.
* Pozycja Y pocisku: self.gracz.y. Pocisk zaczyna się na wysokości gracza.
* Szerokość pocisku: 6 pikseli.
* Wysokość pocisku: 10 pikseli.
* Obraz pocisku: "pocisk.png".
* Prędkość pocisku: 8 pikseli na klatkę w kierunku przeciwnym do ruchu wrogów (-1).

 Dodanie pocisku do listy pocisków: Nowy pocisk jest dodawany do listy self.lista\_pocisków.

 Aktualizacja czasu ostatniego strzału: Czas ostatniego strzału (self.czas\_ostatniego\_strzału) jest ustawiany na aktualny czas, aby można było ponownie strzelać dopiero po upływie odpowiedniego opóźnienia.

Aktualizuj

 Aktualizacja położenia gracza: Wywoływana jest metoda aktualizuj dla gracza (self.gracz.aktualizuj()), co pozwala na kontrolę jego położenia.

 Dodawanie wrogów: Sprawdzane jest, czy upłynął wystarczający czas od ostatniego dodania wroga. Jeśli tak, tworzony jest nowy obiekt Wróg o ustalonych parametrach, a następnie dodawany jest do listy wrogów (self.lista\_wrogów.append(wróg)).

 Aktualizacja położenia i sprawdzenie warunków dla każdego wroga: Wywoływana jest metoda aktualizuj dla każdego wroga znajdującego się na planszy. Ponadto sprawdzane jest, czy wrogowie przekroczyli poziom gracza. Jeśli któryś z wrogów osiągnie wysokość gracza minus jego wysokość (wróg.y > self.gracz.y - self.gracz.wysokość), gra kończy się.

 Aktualizacja położenia i usuwanie pocisków: Wywoływana jest metoda aktualizuj dla każdego pocisku znajdującego się na planszy. Jeśli pocisk opuści obszar gry (jego pozycja Y jest mniejsza niż -10), zostaje usunięty z listy self.lista\_pocisków.

 Sprawdzenie kolizji: Wywoływana jest metoda sprawdź\_kolizje(), która sprawdza kolizje pomiędzy pociskami a wrogami. Jeśli pocisk trafi w wroga, odpowiednie obiekty są usuwane z listy.

Sprawdz\_klozje

 Pętle for iterują po kopiach listy wrogów i pocisków (self.lista\_wrogów[:] i self.lista\_pocisków[:]). Kopie są tworzone za pomocą składni [:], aby uniknąć problemów z modyfikacją listy podczas iteracji po niej.

 Dla każdej pary wrogów i pocisków sprawdzane jest, czy ich prostokątne obszary kolizji (pygame.Rect) nachodzą na siebie za pomocą metody colliderect.

 Jeśli wykryto kolizję, to odpowiednie obiekty są usuwane z listy za pomocą metody remove. W tej implementacji, przy wykryciu kolizji, usuwany jest tylko pierwszy znaleziony wrog i pocisk. Następnie pętla wewnętrzna jest przerywana za pomocą instrukcji break, aby nie kontynuować sprawdzania kolizji dla danego wroga, ponieważ już został on usunięty z listy.

Obraz zawierający tekst, pismo odręczne, szkic, rysowanie

Opis wygenerowany automatycznie

Instrukcja obsługi

Gracz może sterowac strałkami lewo prawo i używac spacji do wystrzelania pocisku w celu zniszczenia wrogiego statku.