Platformer.py

Celem projektu było stworzenie bazy kodu dla gier platformowych napisanych z wykorzystaniem pygame. Miała ona zawierać:

- kolizje i prostą fizykę
- poruszenie się postacią gracza
- animacje
- proste w edycji poziomy

Projekt jest rozbity na 6 plików:

platformer.py
Główny plik. Inicjalizuje pygame oraz poziom pierwszy, a także zawiera główną pętle odpowiedzialną za aktualizowanie położenia obiektów w poziomie oraz wyświetlanie ich, a także za zamknięcie aplikacji.

level.py
Zawiera klase Map odpowiedzialną za wszystko co dzieje się w obrębie naszego poziomu, jak np. kolizje z nim.

- __init__(self,data,surface,lvlNr)
  Konstruktor naszego poziomu. Ustawia powierzchnie na jakiem będziemy wyświetlać poziom, inicjuje zmienne currentX (potrzebna do kolizji horyzontalnych) i currentLevel (wykorzystywana przy zmianie poziomu) oraz wywołuje funkcje setup(data)
- setup(self,data)
  Buduje poziom z użyciem podanej tablicy stringów – data, iterując po niej i stawiając na określonej pozycji na ekranie blok jeśli natrafi na tablicy na 'X', gracza jeśli natrafi na 'P' lub metę jeśli natrafi na 'M'. Poziomy mogą być dowolnego rozmiaru, ale należy dostosować rozmiar ekranu w settings.py
- horizontalMovement(self)
  Funkcja ta zajmuje się ruchem i kolizją gracza w osi x. Działa ona na prostej zasadzie: dla każdego bloku w poziomie, jeśli koliduje z graczem: jeśli blok jest metą: załaduj kolejny poziom, jeśli nie: jeśli gracz poruszał się w lewo (kolizja po lewej): zrównaj pozycje gracza po lewej do pozycji bloku po prawej, ustaw zmienną onLeft na true (wykorzystywana do odpowiedniego ustawiania pozycji gracza w trakcie odtwarzania animacji, tak by gracz nie przenikał częściowo przez bloki) oraz ustaw currentX na player.rect.left, a jeśli gracz poruszał się w prawo: postąp tak samo ale dla pozycji po prawej zrównaj ją do lewej, ustaw zmienną onRight i currentX. Jeśli gracz obecnie koliduje po lewej lub po prawej (onLeft lub onRight) oraz currentX jest odpowiednio mniejsze (dla kolizji po lewej) i większe (po prawej) lub gracz zaczął przemieszczać się w przeciwnym kierunku: ustaw onLeft lub onRight na false, indykując koniec kolizji.
- verticalMovement(self)
  Metoda działająca podobnie do horizontalMovement() ale z drobnymi różnicami: jeśli kolidujemy od góry lub od dołu to zerujemy jego prędkość w osi y, a kiedy sprawdzamy czy kolizja się zakończyła to patrzymy jedynie na prędkość gracza w osi y (jeśli koliduje od góry to czy zaczął już spadać, a jeśli od dołu to czy podskoczył)
- update(self)
  Metoda służąca do aktualizowania informacji o poziomie co klatkę, uruchamiana w głównej pętli programu. Wywołuje funkcje update bloków i gracza, a także metody horizontalMovement i verticalMovement i wyświetla wszystko.

player.py
Zawiera wszystkie informacje związanie z graczem, jak stan animacji, wektor z jakim się przemieszcza, kierunek w którym jest zwrócony i od których stron koliduje, a także metody zajmujące się tym wszystkim

- __init__(self,pos)
  Inicjuje wszystkie wspomniane wyżej zmienne, wywołuje funkcje importCharacterGraphics().
- importCharacterGraphics(self)
  Ładuje klatki animacji z folderu i układa je w słowniku według nazw animacji.
- runAnimation(self)
  Służy do odtwarzania animacji. Najpierw wybiera odpowiednią animacje wg kierunku w jakim purusza się gracz (np. gdy gracz porusza się w dół to wybiera animacje spadania), potem inkrementuje licznik klatek animacji lub jeśli ten wyjdzie poza ilość klatek w wybranej wcześniej animacji zeruje go, potem przyporządkowuje odpowiednią klatkę jako teksturę gracza i obraca ją w kierunku w którym zwrócony jest gracz, a na koniec ustawiamy obecną pozycje gracza bazując na to po których stronach obecnie koliduje aby uniknąć błędów graficznych polegających na przenikaniu przez bloki.
- getInput(self)
  Jak nazwa wskazuje pobiera wciśnięte przez gracza klawisze i na tej postawie przypisuje odpowiednie zmienne jak prędkość gracza czy kierunek w którym jest zwrócony.
- applyGravity(self)
  Nadaje graczowi pęd w kierunku dolnym.
- jump(self)
  Kiedy gracz wciśnie spacje stojąc na ziemi nadajemy mu prędkość w osi y równą wcześniej przypisanej sile skoku.
- update(self)
  Wywołuje co klatkę getInput() i runAnimation().

settings.py
Zawiera informacje o poziomach oraz inne ustawienia jak rozdzielczość okna. W zamyśle jest to plik dostępny do edycji dla użytkownika, by ten mógł tworzyć własne poziomy i np. dzielić się nimi z innymi graczami. Każdy poziom jest zapisany jako lista stringów:

```
level_layout2 = [
  'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX',
  'X                                MX',
  'X             XXXXXXXXXXXXXXXXXXXX',
  'X           X                    X',
  'X     XXX                        X',
  'X      X        X                 X',
  'X      X        X                 X',
  'X      X       XX                 X',
  'X      X        X   X   X    X    X',
  'X      X        X   X   X    X    P  X',
  'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX',
]
```

Przykładowy poziom.

- X – blok
- P – pozycja startowa gracza
- M – meta

support.py
Zawiera funkcje pomocnicze, wykorzystane w różnych miejscach programu.

- importFolder(path)
  Zwraca grafiki z podanego folderu w postaci listy powierzchni gotowych do wyświetlenia. Powoduje zatrzymanie pracy programu jeśli w podanym folderze znajdują się nie tylko obrazy.

tile.py
Zawiera klase reprezentującą blok z których zbudowany jest poziom.

- __init__(self, pos, size):
  Stawia blok na określonej pozycji, importuje i nadaje mu teksturę oraz ustawie że blok domyślnie nie jest metą (zmienia to klasa poziomy w funkcji setup())
- importGraphics(self)
  Działa tak samo jak w przypadku gracza ale bez słownika dzielącego grafiki na animacje
- update(self,x_shift)
  Pozwala przesuwać poziom (w przypadku kiedy użytkownik zaprojektowałby naprawdę długi poziom) oraz koloruje blok mety na czerwono.