Lista zadań nr 5

Zadanie 1 (1 pkt) Zmodyfikuj plik `platform_game_1_player.py` tworząc nowy plik o nazwie `platform_game_2_move.py`. Modyfikacja powinna opierać się m.in. na dodaniu do klasy `Player` następujących metod:

- metoda `turn_right()` - modyfikuje odpowiednio atrybut `movement_x` (wartość dodatnia np. 6) oraz atrybut `direction_of_movement`;
- metoda `turn_left()` - zmienia atrybut `movement_x` (wartość taka sama jak w metodzie `turn_right` ale o ujemnym znaku) oraz atrybut `direction_of_movement`;
- metoda `stop()` - ustawia wartość atrybutu `movement_x` na zero;
- metoda `_move()`, która przyjmuje jako parametr dwuelementową listę grafik (np. `image_list`) i w zależności od wartości atrybutu `count` (który powinien przy każdym wywołaniu tej metody wzrastać o 1, ale cyklicznie w zakresie od 0 do 8), powinien modyfikować wartość atrybuty `image` tj. dla wartości `count` od 0 do 4 atrybut `image` powinien mieć wartość pierwszej grafiki z listy, a w przeciwnym wypadku drugiej;
- metoda `update()`, która modyfikuje położenie gracza (np. atrybut `rect.x`) o wartość atrybutu `movement_x` oraz wywołuje metodę `_move()` z parametrem odpowiednio `gm.IMAGES_R` lub `gm.IMAGES_L` w zależności o wartości `movement_x`;
- metoda `get_event`, która jako parametr przyjmuje zdarzenie i w zależności od wartości tego parametru (wciśnięcie/puszczenie odpowiedniego klawisza) wywołuje metodę `turn_right()`, `turn_left()` lub `stop()`.

W pętli zdarzeń wywołaj metodę `get_event`, a w pętli gry metodę `update()` obiektu `player`

Zadanie 2 (1 pkt) Zmodyfikuj plik `platform_game_2_move.py` tworząc nowy plik o nazwie `platform_game_3_platforms.py`. Modyfikacja powinna polegać na dodaniu trzech klas.

- klasa `Paltform` oparta o klasę wbudowaną `pygame.sprite.Sprite` i posiadającą następujące metody:
  - metoda `__init__()` o parametrach: `colour`, `width`, `height`, `rect_x`, `rect_y`; konstruktor powinien incjalizować następujące atrybuty `width` i `height` wartościami odpowidenich parametrów oraz tworzyć atrybut `rect` inicjalizując

go instancją `pygame.Surface([width, height])`; użyj metody `fill` aby wypełnić prostokąt odpowiednim kolorem oraz zmodyfikuj wartość atrybutów `rect.x` i `rect.y` wartościami odpowiednich parametrów;

  - metoda `draw()` - analogiczna jak w klasie `Player`.

- ogólna klasa reprezentująca planszę gry - `Level`, która posiada następujące metody:

  - metoda `__init__()` o parametrze będącym instancją klasy `Player`; konstruktor powinien tworzyć dwa atrybuty: `player` - incjalizowany parametrem oraz `set_of_platforms` - incjalizowany pustym zbiorem, który będzie zawierał obiekty klasy `Platform`;
  - metoda `update()`, która wywołuje metodę `update()` każdego z obiektów z kolekcji `set_of_platforms`;
  - metoda `draw()`, która wywołuje metodę `draw()` każdego obiektu ze zbioru `set_of_platforms`.

- klasa `Level_1` oparta o klasę `Level` i zawierające metodę konstruktora; konstruktor powinien wywoływać metodę klasy bazowej oraz tworzyć i dodawać do zbioru `set_of_platforms` kilka obiektów klasy `Platform` (uwaga: wysokość platform powinna być równa 70, a jej szerokość powinna być wielokrotnością tej liczby).

Utwórz obiekt klasy `Level_1` i powiąż go wzajemnie z obiektem klasy `Player`. W pętli gry wywołaj odpowiednie metody instancji klasy `Level_1`.

Zadanie 3 (1 pkt) Zmodyfikuj plik `platform_game_3_platforms.py` tworząc plik o nazwie `platform_game_4_jump.py`. Modyfikacja powinna dotyczyć klasy `Player`:

- utwórz metodę `_gravitation`, której wywołanie spowoduje zwiększenie wartość atrybutu `movement_y` do 1 gdy był on zerem lub zwiększenie o np. 0.35; metoda ta powinna być wywoływana w przez metodę `update()`;

- dodaj metodę `jump()`, która ustawi wartość atrybutu `movement_y` np. na -12 i wywołaj ją w metodzie `get_event()` w odpowiedzi na wciśnięcie jakiegoś klawisza;

- w metodzie `update()` dodaj kod, który modyfikuje położenie gracza w pionie oraz kod który wykrywa kolizję z platformami (w czterech kierunkach) - wykorzystaj funkcję `pygame.sprite.spritecollide()` np. gdy gracz uderzy w obiekt platformy swoją prawą stroną to wartość atrybutu gracza `rect.right` powinna być ustawiona na wartość atrybutu platformy `rect.left` (z którą nastąpiła kolizja) - analogicznie należy postąpić w przypadku kolizji w pozostałych kierunkach

ruchu; skorzystaj z grafik `gm.FALL_L`, `gm.FALL_R`, `gm.JUMP_L` oraz `gm.JUMP_L` aby zmienić wygląd obiektu gracza w czasie skoku i spadania.

Zadanie 4 (1 pkt) Zmodyfikuj plik `platform_game_4_jump.py` tworząc plik o nazwie `platform_game_5_platform_scroller`. Modyfikacja kodu powinna dotyczyć dwóch klas: `Platform` oraz `Level`. W pierwszej klasie zmodyfikuj metodę `draw()` tak aby wypełniać platformę odpowiednią grafiką z listy będącej jej parametrem (lista czetoroelmentowa zob. dalej). Wywołując tę metodę w klasie `Level` przekaż jej listę grafik `gm.GRASS_LIST`.

W konstruktorze klasy `Level` dodaj atrybut `world_shift` o wartości początkowej równej zero. Następnie, dodaj parometrową metodę `_shift_world()`, która zwiększy wartość atrybutu `world_shift` o wartość parametru tej metody, a także "przesunie" wszystkie obiekty planszy o wartość tego parametru (na razie są to tylko obiekty ze zbioru `set_of_platforms`). W metodzie `update()` dodaj dwa fragmenty kodu źródłowego. Pierwszy fragment powinien sprawdzać czy np. wartość `player.rect.right` jest większe od 500, obliczać różnicę między `player.rect.right` oraz 500, wywoływać z wartością tej różnicy (ze znakiem przeciwnym) metodę `_shift_world()` oraz ustawić wartość atrybutu `player.rect.right` na 500. Drugi fragment kodu powinien działać analogicznie, gdy np. wartość atrybutu `self.player.rect.left` będzie mniejsza od 150.