Instrukcja 2
Sterowanie oraz podstawowe animacje
Podstawy tworzenia gier

Obsługa wejścia

Obsługa wejścia wprowadzonego przez użytkownika (np. naciśnięcia klawisza na klawiaturze lub myszy) to kluczowy aspekt tworzenia gier. Pozwala ono na interakcję z programem. W Pygame wejście jest obsługiwane za pomocą zdarzeń - Event. Najważniejszymi zdarzeniami związanymi z wejściem są „zdarzenia klawiatury". Pozwalają one „wykryć" naciśnięcia *jakiegokolwiek* klawisza, a następnie wykonanie określonych akcji, w zależności od tego który klawisz został naciśnięty.

1. **Wewnątrz pętli gier znajduje się pętla for, pozwalająca na obsługę poszczególnych zdarzeń.** Kod z poprzedniej listy:

```
running = True
while running: #pętla gry
        for event in pygame.event.get(): #pętla obsługi zdarzeń
                #w tym miejscu możliwa jest obsługa
                #poszczególnych zdarzeń
        pygame.display.update()
pygame.quit()
```

2. **Podstawowym zdarzeniem** (umieszczonym już w pętli w poprzedniej instrukcji) **jest wyłączenie gry. Wartość zmiennej running zostaje zmieniona na False, co kończy działanie pętli gry, a co za tym idzie, działanie programu.**

```
if event.type == pygame.QUIT:
        running = False
```

3. **Kolejnym zdarzeniem jest wykrycie naciśnięcia klawisza. Naciśnięcie *jakiegokolwiek* klawisza na klawiaturze jest samo w sobie zdarzeniem, a rozpoznanie poszczególnych klawiszy odbywa się na kolejnym etapie.**

```
if event.type == pygame.KEYDOWN:
        #w tym miejscu umieszczamy kod obsługujący
        #poszczególne klawisze
```

UWAGA: Poszczególne rodzaje zdarzeń (event.type) są sprawdzane niezależnie od siebie - pozwala to na umieszczenie warunku if pod if, (bez elif) ponieważ spełnienie jednego warunku nie wpływa na wykonanie kolejnego (jak w przypadku bloku else pomijanego gdy wcześniejsze zostają wykonane)

4. **Po wykryciu naciśnięcia klawisza możliwe jest „rozpoznanie” klawisza i wykonanie kodu wynikającego z tego faktu. Każdy klawisz posiada swoją nazwę np. K_UP oznacza strzałkę w górę.**

   Poniższy kod przedstawia obsługę klawisza esc oraz dwóch strzałek

```
if event.key == pygame.K_ESCAPE:
        running = False
if event.key == pygame.K_UP:
        #miejsce na kod, który ma zostać wykonany
        #po naciśnięciu strzałki w górę
if event.key == pygame.K_DOWN:
        #miejsce na kod, który ma zostać wykonany
        #po naciśnięciu strzałki w dół
```

5. **W podobny sposób wykonywana jest obsługa zdarzeń naciśnięcia klawisza myszy. Jego pozycja może zostać wykorzystana w kodzie np. do rysowania**

```
if event.type == pygame.MOUSEBUTTONDOWN:
        position = pygame.mouse.get_pos()
        pygame.draw.circle(game_view, "blue", position, 20)
```

   W powyższym przykładzie, funkcja pygame.mouse.get_pos() zapisuje aktualną pozycję myszy w formie (x, y). Gdy użytkownik kliknie lewym przyciskiem myszy (zdarzenie pygame.MOUSEBUTTONDOWN), rysujemy kółko o promieniu 20 pikseli o punkcie środkowym w miejscu kliknięcia.

Funkcje (przypomnienie)

Funkcja to blok kodu wielokrotnego użytku (grupa instrukcji), który jest wykorzystywany do wykonania określonego działania. Definiujemy je używając słowa kluczowego def, a następnie nazwy funkcji, po której następują nawiasy (). Kod wewnątrz funkcji jest wcięty (przesunięty za pomocą tabulatora)

```
def nazwa_funkcji():
        #Kod wewnątrz funkcji
```

W celu „wywołania” funkcji, to znaczy wykonania kodu znajdującego się w jej wnętrzu, należy użyć jej nazwy wraz z nawiasami

```
nazwa_funkcji()
```

Do funkcji możemy przekazywać parametry - określone zmienne (np. liczby, teksty, listy, itp.), które zostaną wykorzystane wewnątrz funkcji. Nazwy parametrów umieszczamy wewnątrz nawiasów po nazwie funkcji.

```
def draw_rectangle(x, y):
        pygame.draw.rect(game_view, "red", (x, y, 100, 150)
```

Parametry x i y reprezentują współrzędne punktu początkowego prostokąta

W celu narysowania prostokąta z użyciem powyższej funkcji wystarczy „wywołać" jej nazwę, a w nawiasie podać współrzędne punkt początkowego. W takiej sytuacji wymiary i kolor prostokąta pozostaną bez zmian (takie jak zdefiniowane w funkcji), a zmieniać będzie się tylko położenie punktu początkowego. Funkcje można wywoływać wielokrotnie.

```
draw_rectangle(50, 20)
draw_rectangle(170, 120)
draw_rectangle(480, 200)
```

Zadanie 1

Wykorzystaj projekt z poprzedniego zadania. Napisz kod, który po naciśnięciu myszy wywoła napisaną samodzielnie funkcję draw_circle, a jako parametr przekaże jej pozycję kliknięcia myszą. Funkcja ma za zadanie narysować okrąg w miejscu naciśnięcia myszy. W przypadku naciśnięcia spacji, program ma „wyczyścić" ekran (wykorzystując wspomnianą w poprzedniej instrukcji funkcję fill).
ROZSZERZENIE: Funkcja powinna losować kolor, aby za każdym kliknięciem rysowano okrąg o innym kolorze. Skorzystaj ze sposobu deklarowania kolorów w formacie RGB z poprzedniej instrukcji, a także biblioteki random (randint).

Poruszanie elementu

Poniższy przykład przedstawia sposób „animacji" kwadratu o określonym rozmiarze. Na początku zdefiniowane zostaną podstawowe założenia:

```
block_size = 20
block_pos = [width // 2, height // 2]
```

UWAGA: Operator // oznacza dzielenie bez reszty, ponieważ do ustalenia pozycji konieczne jest wskazanie konkretnego piksela, bez ewentualnej części po przecinku. W powyższym przypadku zdefiniowany zostaje punkt środkowy utworzonego okna gry.

1. Na początku w pętli gry należy przygotować obsługę klawiszy góra / dół

```
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                block_pos[1] -= block_size
            if event.key == pygame.K_DOWN:
                block_pos[1] += block_size
```

2. Następnie, wewnątrz pętli running (po sprawdzeniu zdarzeń, za częścią „for event in ...") warto sprawdzić, czy rysowany kwadrat nie przekracza granic okna gry. W tym celu wykorzystana zostaje funkcja min() - w przypadku gdy pozycja bloku jest większa niż rozmiar okna to zwrócony zostaje rozmiar okna, a następnie funkcja max() - w przypadku gdy pozycja bloku jest mniejsza od zera to zwrócone zostanie zero.

```
block_pos[1] = max(0, min(block_pos[1], height - block_size))
```

3. W kolejnym kroku można rozpocząć rysowanie. Będzie się ono odbywać klatka po klatce, od nowa po każdej iteracji pętli gry. W tym celu należy najpierw „wyczyścić" całe pole gry wypełniając je jednolitym kolorem - w przeciwnym wypadku poszczególne klatki nakładałyby się na siebie!

```
game_view.fill("white")
```

4. Rysowanie kwadratu odbywa się za pomocą pygame.draw.rect(). Pozycja bloku zostaje odczytana z listy block_pos (pierwszy element to współrzędna szerokości, a drugi wysokości.

```
pygame.draw.rect(game_view,"green", (block_pos[0], block_pos[1], block_size, block_size))
```

UWAGA: Pamiętaj, że w programowaniu numerację elementów listy rozpoczynamy od zera! Pierwszy = [0], drugi = [1], itd.

5. Na koniec konieczne jest odświeżenie okna gry, w celu ponownego narysowania jego zawartości.

```
pygame.display.update()
```

Zadanie 2

Utwórz nowy projekt. Wykorzystaj kod z instrukcji (pełny kod źródłowy znajduje się na końcu instrukcji). Aktualny stan projektu obsługuje ruch kwadratu w górę i w dół. Na podobnej zasadzie, wykonaj obsługę ruch w lewo i prawo (miejsce na jego pozycję zostało już uwzględnione w tablicy block_pos). Pamiętaj o ograniczeniu możliwości przesuwania kwadratu poza granice okna gry!

ROZSZERZENIE: Wykonaj część podstawową, a następnie zmodyfikuj kod w taki sposób, aby blok poruszał się w trybie ciągłym w określonym kierunku. W tym celu:

1. Utwórz trzeci warunek początkowy (obok rozmiaru bloku i jego położenia) określający jego kierunek ruchu

```
block_dir = [block_size, 0] #blok przesuwa się w lewo
```

2. W zdarzeniach klawiatury zamień przesuwanie na zmianę kierunku w block_dir
3. Przed sprawdzeniem pozycji (w granicach okna) konieczna będzie zmiana aktualnej pozycji bloku o krok określony w block_dir
4. Po narysowaniu obiektu, ustaw odświeżanie na 10 FPS

```
pygame.time.Clock().tick(10)
```

Rozszerzenie

Nazwy funkcji

Python (jak prawie każdy język programowania) posiada wytyczne określające „poprawny" sposób pisania kodu. Zgodnie z nimi, nazwy funkcji powinny być napisane małymi literami, a znajdujące się w nich słowa rozdzielone _ (to tak zwany snake_case)

```
def draw_rectangle(x, y):
```

~~def draw_Rectangle(x, y):~~

~~def drawrectangle(x, y):~~

Właściwe nazywanie funkcji poprawia czytelność kodu, a także późniejsze jego zmiany. Ułatwia to też współpracę w ramach zespołu - pozostali programiści będą wiedzieli „co robi" dana funkcja bez konieczności zagłębienia się w kod. Z tego powodu wszystkie nazwy powinny opisywać DOKŁADNIE TO co robi funkcja, bez używania skrótów, a tym bardziej nazw jednoliterowych!

Który z poniższych przykładów jest najczytelniejszy?

```
def calculate_average(grades_list: String):
```

```
def calc_avg(grades):
```

```
def c_a(x):
```

Wskazówka: Nie zawsze mniejsza ilość „treści" oznacza lepszy kod. Czasami lepiej napisać „więcej", a dzięki temu zaoszczędzić czas konieczny na szukanie i „odnajdywanie się" w (nawet własnym) kodzie.

Zauważ, że dobrze nazwana funkcja nie wymaga komentarzy / opisów.

Typowanie

Python nie wymaga „typowania" zmiennych - czyli określania „z góry" typu danych, które będą w niej przechowywane. Dobrą praktyką jest jednak typowanie zmiennych znajdujących się wewnątrz funkcji. Stanowi to dobrą dokumentację kodu (od razu informuje jakie dane funkcja przyjmuje), a także umożliwia środowisku programistycznemu na podpowiedzi oraz sprawdzenie przekazywanych parametrów pod kątem potencjalnych błędów.

```
def create_player(name: str, level: int)
```

Sytuacja wygląda tak samo w przypadku zwracania wartości przez funkcje.

```
def add_numbers(a: int, b: int) -> int:
    return a + b
```

UWAGA: Jednoliterowe nazwy są dopuszczalne w przypadku tak oczywistych nazw zmiennych jak powyższy przykład. Kod „sam się dokumentuje".

W przypadku bardziej rozbudowanych typów danych jak listy, krotki (tuple), itp. warto skorzystać z biblioteki typing. Umożliwia ona bardziej dokładne wskazanie jaki typ danych znajduje się np. wewnątrz listy. Należy jednak pamiętać, że typy danych z biblioteki typing rozpoczynamy od WIELKIEJ litery.

```
from typing import List

def sum_elements(numbers: List[int]) -> int:
    return sum(numbers)
```

Funkcje lambda

Funkcja lambda (lub wyrażenie lambda) to mała, anonimowa funkcja, którą można szybko utworzyć bez konieczności definiowania jej za pomocą słowa kluczowego def. Ma postać jednego wyrażenia, które może przyjąć dowolną ilość argumentów. Funkcje lambda są idealne do szybkiego, jednorazowego użycia. Struktura funkcji lambda to:

lambda parametry: wyrażenie

Przykłady funkcji lambda:

1. Podniesienie liczby x do kwadratu
   ```
   kwadrat = lambda x: x**2
   print(kwadrat(5)) #25
   ```
2. Dodanie 5 do parametru y
   ```
   wynik = lambda y: y + 5
   print(wynik(10)) #15
   ```
3. Mnożenie dwóch liczb
   ```
   mnozenie = lambda a, b: a * b
   print(mnozenie(2, 3)) #6
   ```
4. Podniesienie każdego elementu listy do potęgi 2
   ```
   lista_liczb = [1, 2, 3, 4]
   potegowanie = lambda x: [i**2 for i in x]
   print(potegowanie(lista_liczb)) #[1, 4, 9, 16]
   ```
   UWAGA: Powyższy przykład wyrażenia lambda zawiera w sobie także pętlę for in, która pozwala na iterację przez wszystkie elementy listy

Funkcje lambda są najczęściej wykorzystywane do szybkich operacji na listach (np. sortowanie, przekształcanie, itp.), a także jako elementy algorytmów oraz do obsługi zdarzeń. Praktycznie wszystkie praktyczne zastosowana opierają się na analizie danych (w szczególności przetwarzania z zakresu Big Data).

Kod źródłowy

Przesuwanie bloku w górę i w dół

```
import pygame

pygame.init()

width, height = 500, 500
game_view = pygame.display.set_mode((width, height))
pygame.display.set_caption("Block Movement")

block_size = 20
block_pos = [width // 2, height // 2]

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                block_pos[1] -= block_size
            if event.key == pygame.K_DOWN:
                block_pos[1] += block_size

    block_pos[1] = max(0,min(block_pos[1], height - block_size))

    game_view.fill("white")
    pygame.draw.rect(game_view, "green", (block_pos[0],
                    block_pos[1], block_size, block_size))
    pygame.display.update()

pygame.quit()
```