

PyGame

Podstawy biblioteki do tworzenia gier

Cel: Poznajemy podstawy biblioteki PyGame oraz koncepcję pętli programu i sprite-ów

Ćwiczenie 0 – O co chodzi?

Na dzisiejszych zajęciach poznamy podstawowe możliwości biblioteki (modułu) do tworzenia gier w Pythonie o nazwie PyGame.

Ćwiczenie 1 – Moduły

W ramach przypomnienia tego, co robiliśmy na poprzednich zajęciach, niech każdy przygotuje prościutki program, który będzie pytał użytkownika ile ma wyświetlić liczb a potem je wyświetlał.

Np:

```
licz = input("Podaj liczbę powtórzeń: ")
licz = int(licz)
for i in range(licz):
    print(i)
```

```
Podaj liczbę powtórzeń: 5
0
1
2
3
4
```

Teraz zastanówmy się co powinniśmy zrobić, gdybyśmy chcieli nasz program zmienić w rodzaj sekundnika tak, aby wyświetlane liczby oznaczały mijające sekundy.

Oczywiście trzeba by kazać programowi czekać sekundę po wyświetleniu każdej liczby. Aby to jednak zrobić będziemy musieli skorzystać ze specjalnego modułu (biblioteki) do obsługi czasu.

Chwilę pomówmy o tym co to takiego jest moduł, w innych językach programowania często nazywany biblioteką. Otóż zastanówcie się – gdy programista tworzy jakiś program, czy myślicie, że wszystko pisze od początku do końca sam – wyświetlanie grafiki, obliczenia, obsługę dźwięków itp. itd.? Oczywiście nie – w wielu miejscach korzysta z gotowych rozwiązań!

Można to trochę porównać do konstruowania samochodu – inżynierowie nie wymyślają zazwyczaj własnych śrubek czy nakrętek ale korzystają z tego, co ktoś już wcześniej wymyślił, stworzył i przetestował. Dzięki temu projektanci samochodów mogą skupić się na ich kreatywnym zastosowaniu a nie „wyważaniu otwartych drzwi”.

Podobnie jest w programowaniu – zamiast samemu tworzyć o d podstaw WSZYSTKO można korzystać właśnie z bibliotek. Niektóre z nich są dostępne za darmo, inne płatne. Pewna liczba najczęściej używanych modułów jest dostępna „domyślnie” z językiem programowania, pozostałe natomiast trzeba samemu pobrać.

Moduły importuje się tak:

```
import time
```

Ta linijka mówi Pythonowi, że chcemy korzystać z biblioteki time. Teraz możemy korzystać ze wszystkich funkcji w tej bibliotece. Nam potrzebna będzie funkcja sleep(), która zatrzymuje wykonywanie programu na określonej liczbie sekund.

Wszystkie funkcje należące do zewnętrznych modułów wywołujemy podając:

```
(nazwa modułu).(nazwa funkcji)
```

Czyli w naszym przypadku

```
time.sleep(1)
```


Każdy modyfikuje program tak, żeby czekał sekundę po wyświetleniu każdej liczby.

Ćwiczenie 2 – Instalacja PyGame

Wiemy już jak zaimportować moduł, więc jesteśmy gotowi zacząć korzystać z modułu do tworzenia gier w Pythonie – PyGame.

Jako, że nie jest to moduł domyślny musimy go najpierw zainstalować. Pobieramy go z tego linka:

<https://goo.gl/2OZhRA>

Pobieramy cały plik ***.whl** czyli od razu po wejściu na stronę pod tym odnośnikiem klikamy w prawym górnym rogu ikonkę 

Następnie otwieramy konsolę w folderze, gdzie mamy ściągnięty plik (plik -> otwórz wiersz polecenia) i wpisujemy po kolei dwie komendy:

```
pip install wheel
```

Umożliwia ona instalowanie modułów z plików *.whl. A następnie

```
pip install pygame-1.9.2a0-cp34-none-win_amd64.whl
```

Która zainstaluje nam nasz moduł. Pamiętajcie, że używając klawisza TAB można dopełnić wpisywaną nazwę pliku – nie trzeba jej samemu pisać w całości. Czyli wpisujemy

```
pip install py [TAB]
```

I system sam uzupełni brakującą nazwę pliku.

W celu sprawdzenia, czy wszystko poszło zgodnie z planem odpalamy w konsoli interpretera Pythona wpisując polecenie python a następnie wpisujemy import pygame

Jeśli zainstalowaliśmy go poprawnie nie powinniśmy dostać żadnych błędów:

```
C:\Users\teflooon>python
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC
v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>> import pygame
>>>
```

Ćwiczenie 3 – Boilerplate czyli szkielet każdej gry

Wracamy do SublimeText-a i stworzymy najprostszy możliwy program w PyGame:

```
import pygame

pygame.init()
WIELKOŚĆ_EKRANU = (800, 600)
ekran = pygame.display.set_mode(WIELKOŚĆ_EKRANU)

while True:
    for event in pygame.event.get:
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()

    pygame.display.update()
```

Rozbijmy to na części składowe. Po kolei:

```
import pygame  
importuje bibliotekę
```

```
pygame.init()
```

Inicjalizuje Pygame. Jest to funkcja która „za kulisami” wykonuje wszystkie przygotowania, abyśmy mogli korzystać z modułu. Zawsze trzeba ją raz wywołać zanim będziemy używali jakichkolwiek innych funkcji

```
WIELKOŚĆ_EKRANU = (800, 600)
```

Jest to zmienna, której, jak sama nazwa wskazuje, użyjemy w następnej linijce do ustawienia wielkości okna gry. Jest to tuple (pol. krotka) – to samo co lista (tablica), ale po stworzeniu nie można jej już w żaden sposób zmieniać. Można go odróżnić od listy, bo ma okrągłe nawiasy a nie kwadratowe.

```
ekran = pygame.display.set_mode(SCREEN_SIZE)
```

Tworzymy okienko naszej gry. W nawiasie podaje się jego wymiary – u nas zamiast wpisywać je bezpośrednio użyliśmy naszej zmiennej WIELKOŚĆ_EKRANU. Zapisujemy nasze okienko do zmiennej ekran – będzie nam ona potem potrzebna, gdy będziemy chcieli np. poruszać elementami na ekranie (bo w Pythonie można obsługiwać na raz wiele wyświetlaczy, więc trzeba powiedzieć programowi na którym z nich chcemy czymś poruszyć)

```
while True
```

To jest nasza główna pętla programu. Każdy program okienkowy w praktycznie dowolnym języku programowania ma taką pętlę. W niej wykonywane są wszystkie instrukcje naszego programu już po odpaleniu. Gdy pętla się kończy – kończy się program. Ponieważ pętla while wykonuje się dopóki warunek jest prawdziwy a naszym warunkiem jest *True* to jest to pętla nieskończona – *True* jest zawsze prawdą 😊

```
for event in pygame.event.get()
```

To jest pętla for, która „wyłapuje” wszystkie eventy (wydarzenia) w naszym oknie. Może to być wiele różnych rzeczy – np. ruch myszki czy przyciśnięcie klawisza. To tutaj będziemy umieszczali kod sterujący naszą grą – np. co ma się stać, gdy wciśniemy spację. Mamy nawet już wpisaną obsługę pierwszego eventu:

<code>if event.type == pygame.QUIT:</code>	<i>jeśli wydarzeniem, które przetwarzamy jest wyjście z gry, to...</i>
<code>pygame.quit()</code>	<i>... zakończ pygame...</i>
<code>quit()</code>	<i>... i zakończ cały program.</i>

Odpalcie program i zobaczcie czy działa!

Jedyną rzeczą, która może być trochę trudna do zrozumienia to idea eventów – co to takiego? Czy jest ich dużo? Jakiego są rodzaju?

Żeby lepiej je pojąć dodamy jedną małą linijkę, zaraz pod linijką `for event in pygame.event.get():`

```
...  
    for event in pygame.event.get():  
        print(event)  
...
```

Odpalcie jeszcze raz nasz program i spróbujcie poruszać myszką nad naszym okienkiem a jednocześnie patrzcie na konsolę – to co się tam pokaże to właśnie te nasze eventy! Możecie też klikać myszką i wciskać klawisze na klawiaturze.

Skąd te wszystkie linie? Otóż tą jedną linijką kazaliśmy programowi wypisywać wszystkie eventy jakie wyłapie. Niech każdy spróbuje wydedukować co oznaczają elementy w każdej linijce.

Zwróćcie też uwagę na ostatnią linijkę, która się wyświetla po zamknięciu programu:

<Event(12-Quit {})> - to jest właśnie ten event **pygame.QUIT**, który wyłapujemy w kodzie naszym **if-em!**

Ćwiczenie 4 – Podstawowe komendy

Na razie nasza gra nie jest specjalnie porywająca – czas ją trochę podkręcić. Najpierw możemy zmienić tło, żeby nie było takie czarne i smutne:

```
ekran.fill( (R, G, B) )
```

gdzie R,G,B to liczby z zakresu 0 – 255. Wytlumaczcie dzieciom o co chodzi w systemie RGB i niech każdy dobierze sobie kolor, który mu się podoba. Niech każdy się zastanowi w którym miejscu kodu umieściłby tę linijkę.

W pygame kolejność malowania elementów ma znaczenie. Jeśli najpierw napiszemy komendę malującą koło a za nią malującą tło to tło „zamaluje” nam koło i nie będzie go widać. Jest to przyczyna wielu błędów.

Wypadałoby też stworzyć jakiś obiekt, którym będziemy mogli sterować. Na początek dodamy zwykłe kółko. Funkcja malująca koło to:

```
pygame.draw.circle(powierzchnia, kolor, miejsce, grubość)
```

powierzchnia – nasz ekran, który stworzyliśmy wcześniej

kolor – tak jak w przypadku tła w formie (R, G, B)

miejsce – środek naszego koła w formie (x, y)

grubość – grubość konturu naszego kółka (w pikselach). Można nie podawać lub ustawić na 0 – wtedy koło będzie zamalowane

Ćwiczenie 5 - Sterowanie

Umieszczamy nasze rysowanie koła w funkcji, która przyjmuje x i y za argumenty

```
def rysuj(x, y):  
    pygame.draw.circle(ekran, (255,0,0), (x,y), 10, 2)
```

Tworzymy dwie zmienne, x i y przed pętlą główną i ustawiamy je np. na 50. Dodajemy dwa nowe przypadki do naszego if-a wyłapującego eventy

```
if (event.type == pygame.KEYDOWN and event.key == pygame.K_LEFT):  
    x -= 50  
  
if (event.type == pygame.KEYDOWN and event.key == pygame.K_RIGHT):  
    x += 50
```

Wywołujemy naszą funkcję wysowania w pętli głównej. Cały kod:

```
import pygame  
  
pygame.init()  
  
SCREEN_SIZE = (800, 600)  
screen = pygame.display.set_mode(SCREEN_SIZE)  
  
def rysuj(x, y):  
    pygame.draw.circle(screen, (255, 0, 0), (x, y), 10, 2)  
  
x = 50  
y = 50  
  
while True:  
    for event in pygame.event.get(): # wychwytuje wszystkie zdarzenia  
        if event.type == pygame.QUIT: # jeśli zdarzenie to wyjście z gry...  
            pygame.quit() # ...to zakończ pygame...  
            quit() # ... i zakończ skrypt.  
  
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_LEFT):  
            x -= 50  
  
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_RIGHT):  
            x += 50  
  
    screen.fill((0, 0, 255))  
    rysuj(x, y)  
  
    pygame.display.update()
```