Zestaw 6

W tym zestawie poznamy dwa różne podejścia do tworzenia grafiki w Pythonie, które mogą się uzupełniać, ale mają też istotne różnice: `turtle` i `pygame`. Moduł `turtle` jest wbudowany w Pythona, służy do tworzenia prostych grafik wektorowych poprzez rysowanie za pomocą „żółwia”. Jest idealny do nauki podstaw programowania graficznego i sprawdza się w wizualizacji prostych animacji, symulacji geometrycznych, czy nauki podstaw algorytmów rysujących.

Zacznijmy od prostego przykładu.

```
import turtle
# Tworzymy ekran i żółwia
screen = turtle.Screen()
t = turtle.Turtle()
# Zmieniamy kształt na strzałkę
t.shape("arrow")
t.forward(100)  # Rysowanie z nowym kształtem
screen.mainloop()
```

Funkcja `turtle.Screen()` tworzy ekran, domyślnie kwadratowy o rozmiarach dostosowanych do wielkości ekranu, np. 600×600 pikseli. Można dostosować rozmiar okna używając metody: `screen.setup(width=800, height=600)`. Następnie tworzymy obiekt „żółwia”, któremu można zmienić kształt na jeden z predefiniowanych: "turtle"- domyślnie żółw, "arrow" - strzałka, "circle" - koło, "square" - kwadrat, "triangle" - trójkąt, "classic" - klasyczny trójkątny wskaźnik. Zasygnalizujmy tylko, że można dużo więcej – zmieniać kolor żółwia, rozciągać jego rozmiar w X i Y, a nawet załadować własną grafikę. Układ współrzędnych w `turtle` jest oparty na standardowym kartezjańskim systemie współrzędnych, Punkt początkowy (0, 0) to środek ekranu. `t.forward(100)` – żółw rysuje linię o długości 100 pikseli. Aby przenieść żółwia do innego punktu bez rysowania, używamy `t.penup()` (podnosi pióro, aby nie rysować) i `t.goto(x, y)` (przesuwa żółwia do współrzędnych (x, y)), `t.pendown()` – po przeniesieniu, możemy opuścić pióro, aby znowu zacząć rysować. Metoda `screen.mainloop()` w bibliotece `turtle` służy do utrzymywania otwartego okna programu i uruchomienia głównej pętli zdarzeń aplikacji graficznej. Dzięki tej metodzie okno z grafiką pozostaje otwarte, a program jest w stanie reagować na różne zdarzenia, takie jak kliknięcia myszki lub wciśnięcia klawiszy. Innymi słowy, `mainloop()` to mechanizm, który „utrzymuje życie” interaktywnego okna `turtle`. Pętla `screen.mainloop()` trwa bez przerwy, dopóki użytkownik nie zamknie okna lub nie zostanie wykonana akcja, która przerwie działanie programu.

Można zdefiniować własny sposób zamknięcia pętli przy użyciu zdarzeń klawiatury. Na przykład, przypisać zamykanie okna do określonego klawisza (np. zamknięcie okna klawiszem "q"):

```
import turtle

screen = turtle.Screen()
t = turtle.Turtle()
t.shape("classic")
t.color("blue")    # Kolor żółwia
t.pensize(2)       # Grubość linii

t.forward(100)

# Funkcja zamykająca okno
def close_window():
    screen.bye()  # Zamyka okno turtle

# Przypisanie klawisza "q" do zamykania okna
screen.listen()
screen.onkey(close_window, "q")

screen.mainloop()
```

Rysowanie i wypełnianie kolorem polega na wołaniu metod `forward()` oraz skręcania o zadaną liczbę stopni `left()`, `right()`, a wypełnianie kolorem wykonane jest w obszarze zakreślonym przez żółwia pomiędzy `t.begin_fill()` oraz `t.end_fill()`, nawet jeśli nie narysujemy figury domkniętej to wypełnią się obszary zamknięte tak jak gdyby żółw wrócił do pozycji początkowej.

```
import turtle
screen = turtle.Screen()

t = turtle.Turtle()
t.speed("fast")
t.color("blue")       # Ustaw kolor linii na niebieski
t.fillcolor("yellow") # Ustaw kolor wypełnienia na żółty

# Rozpocznij wypełnianie
t.begin_fill()

# Rysowanie prostokąta
t.forward(100)  # Rysuj pierwszą linię długości 100 pikseli
t.left(90)      # Skręć w lewo o 90 stopni
t.forward(50)   # Rysuj drugą linię długości 50 pikseli
t.left(90)      # Skręć w lewo o 90 stopni
t.forward(100)  # Rysuj trzecią linię
t.left(90)      # Skręć w lewo o 90 stopni
t.forward(50)   # Rysuj czwartą linię

# Zakończ wypełnianie
t.end_fill()

screen.mainloop()
```

Szybkość poruszania się żółwia można ustawić metodą `t.speed()`, która ma nazwy prędkości ("slowest", "slow", "normal", "fast", "fastest") oraz wartości liczbowe (od 0 do 10). Poniżej program ilustrujący rysowanie spirali z wielokąta, losując po drodze kolor oraz powiększając grubość linii. Jak widać na końcu, rysowanie można zakończyć funkcją `turtle.done()`, której używa się, gdy chcemy coś narysować i nie oczekujemy dalszej interakcji z użytkownikiem.

```
import turtle
import random

# Tworzymy ekran i żółwia
t = turtle.Turtle()
t.speed("fastest")

# Pętla rysująca spiralę
for i in range(300):
    # Zmieniamy grubość linii co 10 kroków
    if i % 10 == 0:
        t.pensize(i // 10 + 1)  # Grubość rośnie co 10 kroków

    # Zmieniamy kolor linii co 5 kroków
    if i % 5 == 0:
        t.color(random.random(), random.random(), random.random())  # Losowy kolor RGB

    t.forward(i * 2)  # Żółw rysuje linię
    t.left(45)        # Żółw skręca o 45 stopni

# Zakończenie programu
turtle.done()
```

Ostatnia część samouczka to interakcja z użytkownikiem. Po utworzeniu ekranu, za pomocą metody `screen.listen()` możemy reagować na wciśnięcie klawisza i wykonanie wtedy funkcji `func` podanej jako argument: `screen.onkey(func, "key")`. Prosty przykład, gdzie strzałki w lewo i prawo zmieniają orientację żółwia, a strzałka do przodu rysuje linię o długości 50 pikseli.

```
import turtle

t = turtle.Turtle()

# Funkcje sterujące
def move_forward():
```

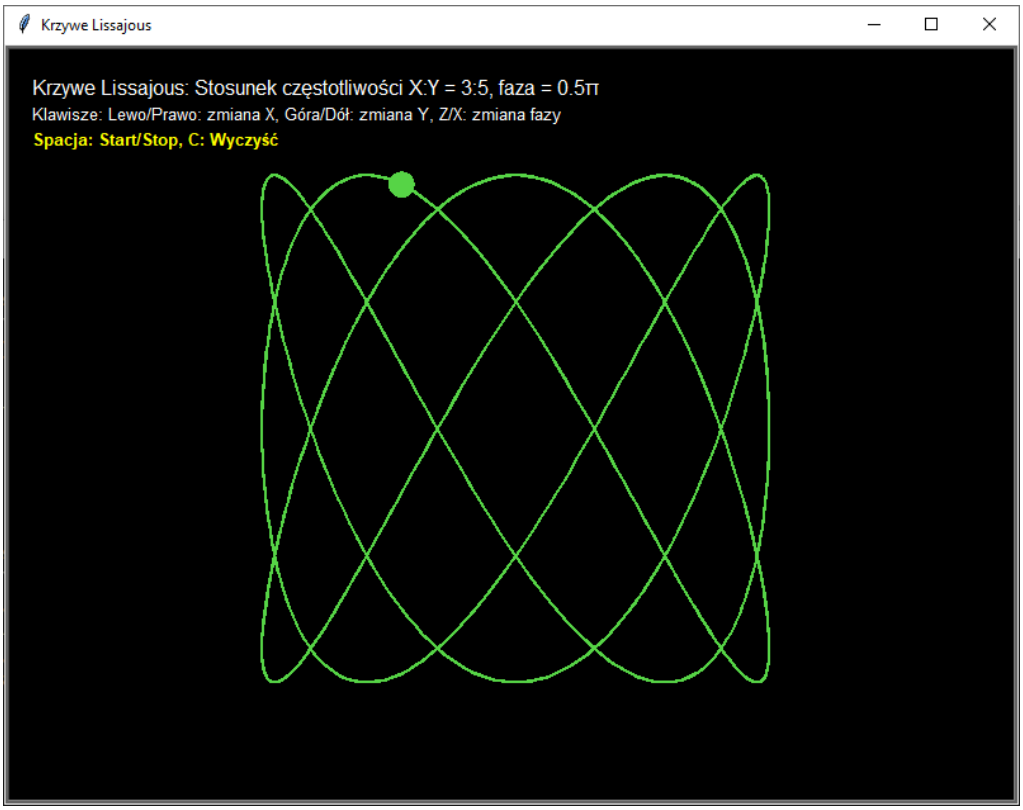
```
    t.forward(50)

def turn_left():
    t.left(45)

def turn_right():
    t.right(45)

# Nasłuchiwanie klawiatury
screen = turtle.Screen()
screen.listen()
screen.onkey(move_forward, "Up")
screen.onkey(turn_left, "Left")
screen.onkey(turn_right, "Right")

screen.mainloop()
```

W bibliotece `turtle` możliwa jest interakcja za pomocą myszki. `screen.onclick(func)` – reaguje na kliknięcia myszki w dowolnym miejscu na ekranie. Funkcja `func` powinna przyjąć dwa argumenty: współrzędne X i Y miejsca kliknięcia. `screen.onscreenclick(func, btn=1)` – to alternatywna wersja `onclick`, z możliwością określenia, który przycisk myszy ma reagować: `btn=1` to domyślny przycisk myszy (lewy), ale można też używać `btn=2` (prawy) czy `btn=3` (środkowy). `turtle.onrelease(func)` – reaguje na moment puszczenia przycisku myszy, co może być użyte do dodatkowej interakcji. Poniższy kod rysuje losowe wypełnione koła, w miejscach kliknięcia myszką.

```
import turtle
import random

screen = turtle.Screen()
t = turtle.Turtle()
t.speed("fastest")
t.shape("circle")

# Funkcja pozycjonująca żółwia i rysująca losowy wypełniony okrąg po kliknięciu
def draw_circle_at_click(x, y):
    # Wybierz losowy kolor i rozmiar
    t.color(random.random(), random.random(), random.random())  # Losowy kolor RGB
    radius = random.randint(10, 50)  # Losowy promień między 10 a 50

    # Przenieś żółwia do odpowiedniego miejsca, aby okrąg miał środek w kliknięty punkcie
    t.penup()
    t.goto(x, y - radius)  # Przesuń żółwia tak, aby zacząć rysowanie od dolnej części okręgu
    t.pendown()

    # Rysowanie okręgu z odpowiednim wypełnieniem
    t.begin_fill()
    t.circle(radius)
    t.end_fill()
    t.penup()
    t.goto(x, y)  # Powrót żółwia do środka


# Przypisanie kliknięcia myszką do funkcji rysującej okrąg
screen.onclick(draw_circle_at_click)

screen.mainloop()
```

1. Zadanie będzie polegać na rozwinięciu zawartości pliku `ZADANIE1/main.py` tak, żeby finalnie program rysował krzywe Lissajous, czyli krzywe powstające przez połączenie dwóch harmonicznych ruchów drgających wzdłuż prostopadłych osi – osi X i osi Y. Rysowane są przez wykreślanie pozycji w przestrzeni `(x, y)`, gdzie położenie zależy od sinusoidalnych funkcji o różnych częstotliwościach i fazach. Te krzywe są wizualizacją złożonego ruchu harmonicznego, które często można zobaczyć na ekranach oscyloskopów. Wzory opisujące krzywe Lissajous wyglądają następująco: x(t)=Asin(at+δ), y(t)=Bsin(bt), gdzie A i B to amplitudy drgań wzdłuż osi X i Y, a i b to częstotliwości drgań wzdłuż osi X i Y, δ to różnica faz między ruchami wzdłuż osi X i Y, t to czas. Krzywe Lissajous są określane przez stosunek częstotliwości drgań wzdłuż osi X i Y, który nazywamy proporcją a : b. Przykładowo,

proporcja 1:1 tworzy okrąg lub elipsę, zależnie od fazy. Inne proporcje mogą prowadzić do bardziej złożonych kształtów. Faza δ wpływa na to, jak są ułożone obie sinusoidy względem siebie. Gdy faza wynosi 0 lub jest wielokrotnością π, krzywe mają specyficzną symetrię. Krzywe Lissajous są zamknięte, gdy stosunek częstotliwości drgań a : b jest liczbą wymierną, czyli gdy a / b jest stosunkiem dwóch liczb całkowitych. Oznacza to, że obie sinusoidy mają okresy, które się ze sobą synchronizują, co prowadzi do powrotu do tej samej pozycji po określonym czasie. Poniżej przykładowy ekran programu, po narysowaniu jednej z krzywych.



Znacząca część kodu, tworząca okno, dwa żółwie (jeden – kółko, drugi do rysowania krzywej, ale oba poruszające się tak samo), wypisanie i aktualizacja legendy, oraz przypisanie odpowiednim klawiszom funkcji reakcji (jak widać w opisie), są pozostawione w kodzie. Dodatkowo na początku jest ustawiona funkcja `screen.tracer(0)`, która wyłącza aktualizowanie ekranu po każdym ruchu żółwia, co poprawia płynność – ale wtedy po wykonaniu ruchu należy wywołać `screen.update()`.

Wymagania formalne Uzupełnić plik `ZADANIE1/main.py` w repozytorium GitHub Classroom. W zasadzie do napisania jest niewiele, bo tylko funkcja `def draw_lissajous()`, która jest podpięta pod `screen.onkey(draw_lissajous, "space")`, czyli rysowanie i zatrzymywanie rysowania po naciśnięciu spacji. Ogólnie rysowanie też najlepiej jest ograniczyć do pewnej zdefiniowanej liczby kroków w pętli. Należy pamiętać, że kolejne położenia wyliczamy na przykład tak:

```
t_rad = t_val * step_size # symulacja upływu czasu, step_size ustawić na mały
x = A * math.sin(a * t_rad + delta)
y = B * math.sin(b * t_rad)
```

gdzie parametry a, b, delta, można zmieniać odpowiednimi klawiszami (patrz przypisane funkcje). Warto też przy każdym uruchomieniu losować kolor krzywej (i okręgu wiodącego). Zadanie nie ma testów – będzie uruchamiane i oglądane! [**zadanie za 1,5 pkt**]

Kolejna biblioteka, **PyGame**, jest znacznie bardziej rozbudowana, wykorzystywana głównie do tworzenia gier oraz złożonych aplikacji multimedialnych. PyGame oferuje dużo więcej możliwości, takich jak obsługa dźwięku, zaawansowane animacje, interakcje użytkownika (mysz, klawiatura), oraz tworzenie bardziej złożonej logiki gry. Zapoznamy się z podstawowymi funkcjami biblioteki („silnika" do tworzenia gier) `pygame`. Nie będzie tu kompletnego „samouczka", tylko minimalne, konieczne do napisania zadań, wprowadzenie. Polecam lekturę załączonego 16. rozdział książki (Rob Miles, „Python. Zacznij programować."), plik Pygame_Python_EN.pdf lub plik Pygame_Python_PL.pdf. Ponadto na stronie https://www.pygame.org dostępne jest sporo przykładowych projektów wraz z kodami źródłowymi. Aby możliwe było korzystanie z `pygame`, oraz zaktualizowaliśmy menadżer pakietów `pip`, na przykład tak: `python -m pip install -U pip`, to instalujemy (na ten moment zainstaluje się pygame-2.6.1): `pip install pygame`

Szkielet programu (gry) w `pygame` wygląda następująco:

```
1.  import pygame, sys
2.  pygame.init()
3.
4.  def main():
5.
6.      size = width, height = 800, 600
7.      screen = pygame.display.set_mode(size)
8.
9.      while True:
10.         for event in pygame.event.get():
11.             if event.type == pygame.QUIT: sys.exit()
12.
13. if __name__ == '__main__':
14.     main()
15.     pygame.quit()
16.     sys.exit()
```

Zaczynając od linii 13. – interpreter Pythona czytając plik źródłowy, definiuje kilka specjalnych zmiennych. Kiedy uruchamiany moduł (plik źródłowy) jest głównym programem, to interpreter przypisze na stałe zakodowany ciąg `'__main__'` do zmiennej `__name__`. I jeśli tak jest, to uruchamiamy pożądane funkcje – tutaj `main()`, a po jej zakończeniu metody zwalniające zasoby (`pygame.quit()`) oraz wychodzące z Pythona (`sys.exit()`).

Linia 1. Importujemy pożądane moduły. Linia 2. Inicjalizacja wszystkich modułów pygame (nie wszystkie może potrzebujemy, ale jest to wygodne). Linia 4. i kolejne – definicja naszej funkcji main() – nie musi się tak nazywać, ale dobrze wygląda. Linie 6-7. Definiujemy krotkę (tuplę) o nazwie size, a przy okazji zmienne szerokość i wysokość, są to oczywiście rozmiary naszego okienka, po czym inicjalizujemy je. Linie 9-11. Program pygame działa na zasadzie nieskończonej pętli, w której można pobierać różnego rodzaju zdarzenia. Linie 10-11. są konieczne, żeby okienko dało się zamknąć.

Z powyższym kodem proszę zrobić dwa testy:
1) zamiast linii 11-12. wstawić pass, proszę jednak mieć na uwadze, że program tak uruchomiony da się „ubić" tylko przez zewnętrzną interwencję (np. Menedżer programów w Windows).
2) dopisać pod linią 10, a przed linią 11, `print(event)` i zobaczyć, co się drukuje: jest to bardzo pouczające, bo zobaczymy, jak `pygame` reaguje na ruchy myszki, klikanie, wciskanie klawiszy itd. Proszę spróbować!
Dodajmy kilka linii kodu w `main()`:

```
1.  def main():
2.      clock = pygame.time.Clock()
3.
4.      pygame.mixer.music.load(r'C:\jakas_sciezka\music.mp3')
5.      pygame.mixer.music.play(-1)
6.
7.      size = width, height = 800, 600
8.      screen = pygame.display.set_mode(size)
9.
10.     while True:
11.         for event in pygame.event.get():
12.             if event.type == pygame.QUIT: sys.exit()
13.
14.         clock.tick(60)
15.         print(clock.get_fps())
16.         #pygame.time.delay(50)
```

Linia 2. Pobieramy obiekt zegar, dzięki któremu można ustawić „frame rate” w naszej grze – tu ustawiłem 60 fps w linii 14. Aby się przekonać jak jest naprawdę, w linii 15. drukujemy (na razie do terminalu, nie w okienku) faktyczną liczbę klatek na sekundę. Proszę poeksperymentować z różnymi wartościami! Można celowo ustawić niski „frame rate” i uzyskać efekt „poklatkowy”. Czasem dla płynności akcji przydaje się jednak co innego – opóźnienie realizacji pętli, proszę odkomentować linię 16. i też poeksperymentować (`delay` jest w ms), przy okazji obserwując fps.

Dla urozmaicenia – dodajmy też muzykę. Linia 4. nie wymaga komentarza, proszę oczywiście użyć jakiegoś swojego pliku .mp3 (lub ściągnąć plik dodany do materiałów zadania i wpisać właściwą ścieżkę). Linia 5. uruchamia odtwarzanie (opcja -1 to nieskończona pętla, inne opcje – zajrzeć do dokumentacji na stronie https://www.pygame.org/docs/ref/music.html#pygame.mixer.music.play)

Dodajmy teraz tytuł, ikonkę oraz tło – przeskalowane i pozycjonowane np. w środku.

```
1.  def main():
2.      clock = pygame.time.Clock()
3.
4.      pygame.display.set_caption('Tytul naszego okienka')
5.      icon = pygame.image.load('jakies_zdjecie.jpg')
6.      pygame.display.set_icon(icon)
7.
8.      pygame.mixer.music.load(r'jakas_sciezka/music.mp3') # zależnie od systemu operacyjnego
9.      pygame.mixer.music.play(-1)
10.
11.     size = width, height = 800, 600
12.     screen = pygame.display.set_mode(size)
13.
14.     image = pygame.image.load(r'jakas_sciezka/moon.jpg')
15.     image = pygame.transform.scale(image, size)
16.
17.     surf_center = (
18.         (width-image.get_width())/2,
19.         (height-image.get_height())/2
20.     )
21.
22.     screen.blit(image, surf_center)
23.     pygame.display.flip()
24.
25.     while True: # reszta jak poprzednio
```

Linie 4-6. sprawią, że nasze okienko będzie miało zdefiniowany przez nas tytuł, a jego rogu pojawi się mała ikonka (miniaturka zdjęcia, które wybierzemy). Swoją ikonkę można też narysować w serwisie https://www.favicon.cc/

Linia 14. Wczytanie obrazka tła oraz ewentualne przeskalowanie go (linia 15.) do wielkości naszego okienka. Linie 17-20. to krotka wyliczająca środek. Parametr ten użyty jest w linii 22. Linia 22-23. Narysowanie tła (drugi parametr to punkt lewy górny okna), zaś `flip()` odświeża (przerysowuje) wszystko.

A teraz tuż przed linią 23. dodajmy jeszcze obrazek np. z piłeczką. Proszę jakąś samemu narysować, albo znaleźć, warto jako .gif z przezroczystą warstwą dookoła piłki!

```
1.      ball = pygame.image.load('ball.gif')
2.      screen.blit(ball, (width/2, height/2))
3.      ballrect = ball.get_rect(center=(width / 2, height / 2))
4.      pygame.display.flip()
```

Linia 3. to odczytanie prostokątnych wymiarów obrazka, co przyda się do sprawdzania kolizji obiektu np. z krawędziami okna. Można się postarać i spozycjonować piłkę w samym środku.

Dopiszę jeszcze teraz pewne zmienne potrzebne do mechaniki naszej piłki – prędkość `speed` (początkową, składowe `x` i `y` zero), „przyspieszenie” (`accel`, również jako składowe). Cały kod `main()` do pętli `while` może wyglądać mniej więcej tak:

```
1.  def main():
2.      clock = pygame.time.Clock()
3.
4.      pygame.display.set_caption('Tytul naszego okienka')
5.      icon = pygame.image.load('ikonka.jpg')
6.      pygame.display.set_icon(icon)
7.
8.      pygame.mixer.music.load(r'jakas_sciezka/music.mp3')
9.      pygame.mixer.music.play(-1)
10.
11.     size = width, height = 800, 600
12.     screen = pygame.display.set_mode(size)
13.
14.     speed = [0, 0]
15.     accel = [0.1, 0.1]
16.
17.       image=pygame.image.load(r'jakas_sciezka/moon.jpg')
18.     image = pygame.transform.scale(image, size)
19.
20.     surf_center = (
21.         (width-image.get_width())/2,
22.         (height-image.get_height())/2
23.     )
24.
25.     screen.blit(image, surf_center)
26.     ball = pygame.image.load('ball.gif')
27.     ball = pygame.transform.scale(ball, (ball.get_width()//2, ball.get_height()//2))
28.
29.     screen.blit(ball, (width/2, height/2))
30.
31.     ballrect = ball.get_rect(center=(width/2, height/2))
32.     pygame.display.flip()
```

Teraz jak odczytujemy i działamy w pętli.

```
1.      while True:
2.          clock.tick(60)
3.          pygame.time.delay(50)
4.
5.          for event in pygame.event.get():
6.              if event.type == pygame.QUIT: sys.exit()
7.
8.          keys = pygame.key.get_pressed()
9.          if keys[pygame.K_ESCAPE]: sys.exit()
10.
11.         if keys[pygame.K_UP]:
12.             pass # zamienić na jakieś przeliczenie
13.         elif keys[pygame.K_DOWN]:
14.             pass
15.         elif keys[pygame.K_LEFT]:
16.             pass
17.         elif keys[pygame.K_RIGHT]:
18.             pass
19.
20.         ballrect = ballrect.move(speed)
21.         if ballrect.left < 0 or ballrect.right > width:
22.             speed[0] = -speed[0]
23.         if ballrect.top < 0 or ballrect.bottom > height:
24.             speed[1] = -speed[1]
25.
26.         screen.blit(image,surf_center)
27.         screen.blit(ball,ballrect)
28.         pygame.display.flip()
```

Wciśnięcie klawiszy odczytujemy poprzez `key.get_pressed()`, otrzymujemy wartości logiczne, wpisane to tablicy `keys`. I jak widać, linia 9., można np. zareagować na fakt wciśnięcia `Esc` i wyjść z programu. Analogicznie można reagować na strzałkę w górę, dół, lewo, prawo (linie 11-18.). A jak reagować – to będzie właśnie przedmiotem zadania. Ruch piłki to linia 20., zmiana zwrotu odpowiedniej składowej prędkości to linie 21-24. Można na początku oczywiście ustawić jakąś „prędkość” początkową (jak wyżej, `speed`). Ostatnie trzy linie to przerysowanie i odświeżenie.

2. Na bazie powyższego kodu zrobimy symulację ruchu w „polu grawitacyjnym”. Proszę zatem ustalić jakieś wartości prędkości początkowej piłki, **przyspieszenie** ma składową **pionową** (0, `<tu_jakas_wartosc>`) (składowe x, **y**). I teraz, **jeśli** piłka jest nieruchoma na początku – to będzie to spadek swobodny (z przyspieszeniem **y**), jeśli „rzucona w górę” ($v_y > 0$), to rzut pionowy, jeśli „rzucona w bok” ($v_x > 0$) to rzut poziomy i ogólnie – rzut ukośny. Piłka powinna poruszać się realistycznie (w sensie: należy wyliczać jej prędkości według **ruchu przyspieszonego w pionie** i **jednostajnego w poziomie**). Wartość „przyspieszenia” (numerycznie) może być dowolnie dobrana tak, żeby ruch odbywał się płynnie, nie za wolno i nie za szybko. Ważne – proszę odbijać piłkę **doskonale sprężyście** (bez strat energii! – czyli w sumie w nieskończoność). **Dodatkowo** do naturalnego ruchu „w polu grawitacyjnym”: dopisać reakcję na wciskanie kursorów, a mianowicie zwiększanie / zmniejszanie składowej prędkości piłki, gdy kursor jest wciśnięty: jeśli zakładamy v_x w prawo jako dodanie, to strzałka w prawo zwiększa v_x o jakąś wartość (zapewne jakiś ułamek typu 0.1, ale trzeba empirycznie dobrać), a strzałka w lewo zmniejsza v_x o jakąś wartość. Podobnie, strzałka w górę lub w dół będzie zwiększać lub zmniejszać wartość składowej prędkości v_y.

Wymagania formalne Zmodyfikować plik `ZADANIE2/main.py` w repozytorium GitHub Classroom, można go oczywiście najpierw uruchomić i zobaczyć działanie! Można użyć te same pliki graficzne / muzyczne (radzę do testów wyłączyć odtwarzanie muzyki, bo szybko irytuje…), albo wstawić własne – ale uwaga! Bardzo proszę zadbać w kodzie o to, żeby **ścieżki do tych plików były względne** i pozwalały na działanie programu po umieszczeniu go w dowolnym podkatalogu. Zadanie nie ma testów – będzie uruchamiane i oglądane! [**zadanie za 1,5 pkt**]

3. Najpierw należy przestudiować załączony kod (`ZADANIE3/main.py` wszystko w jednym pliku), jest to klasyczna gra w ping-ponga, napisana z użyciem biblioteki `pygame`. Proszę po kolei przestudiować kod, który jest komentowany i choć (ewentualnie) zawiera rzeczy nowe, to można się domyślić o co chodzi. W szczególności na początku są definicje dwóch klas `Rakietka` i `Pilka`, które zapisane są jako dziedziczące z klasy `pygame.sprite.Sprite` (proszę zobaczyć w kodzie jak to wygląda). Klasy są dość proste, ich metody dbają o zmianę i sprawdzenie położeń granicznych oraz ustalanie (np. losowanie w pewnym zakresie) wartości prędkości piłki. Program zaczyna się od narysowania ekranu, rakietek, piłki (piłka jest o rozmiarze 10x10 punktów), utworzeniu listy widzialnych w grze obiektów (właśnie odziedziczonych z klasy `Sprite`). Sama mechanika ruchów rakietek powinna być już znana po przestudiowaniu wprowadzenia, ciekawa jest metoda `collide_mask` sprawdzająca czy dane dwa obiekty nie są ze sobą w styczności / kolizji, jeśli tak jest, to na rzecz piłeczki wołamy metodę `bounce()`, która zmienia (i trochę losuje) składową prędkości piłki po odbiciu.

Wymagania formalne Na bazie pliku `ZADANIE3/main.py` w repozytorium GitHub Classroom, po przestudiowaniu i uruchomieniu kodu zadanie będzie polegać na takim jego zmodyfikowaniu, żeby: (a) rakietka była tylko jedna, poruszająca się w poziomie na dole ekranu (w lewo i prawo, strzałkami), (b) piłeczka uruchamiana losowo z góry, punkty mają być naliczane za poprawne odbicie od rakietki, (c) gra ma się zakończyć jeśli piłeczka minie rakietkę i zderzy się ze ścianą – wtedy powinien się wyświetlić wynik końcowy oraz dotychczasowy najwyższy wynik. Najlepszy wynik zapisywać do i odczytywać z **pliku**. Oczywiście pionowa linia jest teraz zbędna. Innymi słowy – przerobić to na grę „jednoosobową”. Wszelkie innowacje estetyczne typu menu, a w każdym razie takie zaprogramowanie gry, żeby gracz wiedział co się dzieje i co ma zrobić – mile widziane. [**zadanie za 2 pkt**]