PyGame – Gra w Kółko i Krzyżyk

Opis implementacji: Używając biblioteki PyGame oraz języka Python, stworzymy prostą grę w kółko i krzyżyk.

Autorzy: Łukasz Zarzecki, Robert Bednarz

Czas realizacji: 90 min.

Poziom trudności: Poziom 3

Biblioteka PyGame ułatwia tworzenie aplikacji multimedialnych, w tym gier. Poniższy scenariusz prezentuje implementację prostej gry Kółko i krzyżyk.

Spis treści

PyGame – Gra w Kółko i Krzyżyk....1
I. Zmienne i plansza gry....2
Kod I.1....2
II. Rysuj planszę gry....2
Kod II.1....2
III. Sztuczna inteligencja....3
Kod III.1....3
IV. Główna pętla programu....4
Kod IV.1....4
POĆWICZ SAM....7

I. Zmienne i plansza gry

Tworzymy plik **tictactoe.py** w terminalu lub w wybranym edytorze i zaczynamy od zdefiniowania zmiennych określających właściwości obiektów w naszej grze.

Kod I.1

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import pygame, sys, random
from pygame.locals import * #udostępnienie nazw metod z locals

# inicjacja modułu pygame
pygame.init()

# przygotowanie powierzchni do rysowania, czyli inicjacja okna gry
OKNOGRY = pygame.display.set_mode((150, 150), 0, 32)
# tytuł okna gry
pygame.display.set_caption('Kółko i krzyżyk')

# lista opisująca stan pola gry, 0 - pole puste, 1 - gracz, 2 - komputer
POLE_GRY = [0,0,0,
            0,0,0,
            0,0,0]

RUCH = 1 # do kogo należy ruch: 1 - gracz, 2 - komputer
WYGRANY = 0 # wynik gry: 0 - nikt, 1 - gracz, 2 - komputer, 3 - remis
WYGRANA = False
```

W instrukcji **`pygame.display.set_mode()`** inicjalizujemy okno gry o rozmiarach 150x150 pikseli i 32 bitowej głębi kolorów. Tworzymy w ten sposób powierzchnię główną do rysowania zapisaną w zmiennej **`OKNOGRY`**. POLE_GRY to lista elementów reprezentujących pola planszy, które mogą być puste (wartość 0), zawierać kółka gracza (wartość 1) lub komputera (wartość 2). Pozostałe zmienne określają, do kogo należy następny ruch, kto wygrał i czy nastąpił koniec gry.

II. Rysuj planszę gry

Planszę można narysować na wiele sposobów, np. tak:

Kod II.1

```
# rysowanie planszy gry, czyli linii oddzielających pola
def rysuj_plansze():
    for i in range(0,3):#x
        for j in range(0,3):#y
            # argumenty: powierzchnia, kolor, x,y, w,h, grubość linii
            pygame.draw.rect(OKNOGRY, (255,255,255), Rect((j*50,i*50),(50,50)), 1)

# narysuj kółka
def rysuj_pole_gry():
    for i in range(0,3):
        for j in range(0,3):
            pole = i*3+j #zmienna pole przyjmuje wartości od 0-8
            # x i y określają środki kolejnych pól,
            # a więc wartości: 25,25, 25,75 25,125 75,25 itd.
            x = j*50+25
            y = i*50+25

            if POLE_GRY[pole] == 1:
                pygame.draw.circle(OKNOGRY,(0,0,255), (x,y),10)#rysuj kółko gracza
            elif POLE_GRY[pole] == 2:
```

```
pygame.draw.circle(OKNOGRY,(255,0,0), (x,y),10)#rysuj kółko komputera
```

Pierwsza funkcja, **`rysuj_plansze()`**, wykorzystując zagnieżdżone pętle, rysuje nam 9 kwadratów o białym obramowaniu i szerokości 50 pikseli (formalnie są to obiekty **`Rect`** zwracane przez metodę **`pygame.draw.rect()`**). Zadaniem funkcji **`rysuj_pole_gry()`** jest narysowanie w zależności od stanu planszy gry zapisanego w liście **`POLE_GRY`** kółek o niebieskim (gracz) lub czerwonym (komputer) kolorze za pomocą metody **`pygame.draw.circle()`**.

III. Sztuczna inteligencja

Decydującą rolę w grze odgrywa komputer, od którego inteligencji zależy, czy rozgrywka przyniesie jakąś satysfakcję. Dopisujemy więc funkcje obsługujące sztuczną inteligencję:

Kod III.1

```
# postaw kółko lub krzyżyk (w tej wersji też kółko, ale w innym kolorze :-))
def postaw_znak(pole, RUCH):
    if POLE_GRY[pole] == 0:
        if RUCH == 1: # ruch gracza
            POLE_GRY[pole] = 1
            return 2
        elif RUCH == 2: # ruch komputera
            POLE_GRY[pole] = 2
            return 1

    return RUCH

# funkcja pomocnicza sprawdzająca, czy komputer może wygrać, czy powinien
# blokować gracza, czy może wygrał komputer lub gracz
def sprawdz_pola(uklad, wygrany = None):
    wartosc = None;
    # lista wielowymiarowa, której elementami są inne listy zagnieżdżone
    POLA_INDEKSY = [ # trójki pól planszy do sprawdzania
        [0,1,2], [3,4,5], [6,7,8], # indeksy pól w poziomie (wiersze)
        [0,3,6], [1,4,7], [2,5,8], # indeksy pól w pionie (kolumny)
        [0,4,8], [2,4,6] # indeksy pól na skos (przekątne)
    ]

    for lista in POLA_INDEKSY:
        kol = [] # lista pomocnicza
        for ind in lista:
            kol.append(POLE_GRY[ind]) # zapisz wartość odczytaną z POLE_GRY
        if (kol in uklad): # jeżeli znalazłeś układ wygrywający lub blokujący
            # zwróć wygranego (1,2) lub indeks pola do zaznaczenia
            wartosc = wygrany if wygrany else lista[kol.index(0)]

    return wartosc

# ruchy komputera
def ai_ruch(RUCH):
    pole = None # które pole powinien zaznaczyć komputer

    # listy wielowymiarowe, których elementami są inne listy zagnieżdżone
    uklady_wygrywam = [[2, 2, 0], [2, 0, 2], [0, 2, 2]]
    uklady_blokuje = [[1, 1, 0], [1, 0, 1], [0, 1, 1]]

    # sprawdź, czy komputer może wygrać
    pole = sprawdz_pola(uklady_wygrywam)
    if pole is not None:
        return postaw_znak(pole, RUCH)

    # jeżeli komputer nie może wygrać, blokuj gracza
```

```
    pole = sprawdz_pola(uklady_blokuje)
    if pole is not None:
        return postaw_znak(pole, RUCH)

    # jeżeli nie można wygrać i gracza nie trzeba blokować, wylosuj pole
    while pole == None:
        pos = random.randrange(0,9) #wylosuj wartość od 0 do 8
        if POLE_GRY[pos] == 0:
            pole = pos

    return postaw_znak(pole, RUCH)
```

Za sposób gry komputera odpowiada funkcja **`ai_ruch()`** (*ai* – ang. *artificial intelligence*, sztuczna inteligencja). Na początku zawiera ona definicje dwóch list (**`uklady_wygrywam`**, **`uklady_blokuje`**), zawierających układy wartości, dla których komputer wygrywa oraz które powinien zablokować, aby nie wygrał gracz. O tym, które pole należy zaznaczyć, decyduje funkcja **`sprawdz_pola()`** przyjmująca jako argument najpierw układy wygrywające, później blokujące.

Podstawą działania funkcji **`sprawdz_pola()`** jest lista **`POLA_INDEKSY`** zawierająca jako elementy listy indeksów pól tworzących wiersze, kolumny i przekątne **`POLA_GRY`** (czyli planszy). Pętla **`for lista in POLA_INDEKSY:`** pobiera kolejne listy, tworzy w liście pomocniczej **`kol`** trójkę wartości odczytanych z **`POLA_GRY`** i próbuje ją dopasować do przekazanego jako argument układu wygrywającego lub blokującego. Jeżeli znajdzie dopasowanie zwraca liczbę oznaczającą gracza lub komputer, o ile opcjonalny argument **`WYGRANY`** ma wartość inną niż **`None`**, w przeciwnym razie zwracany jest indeks **`POLA_GRY`**, na którym komputer powinien postawić swój znak.

Jeżeli indeks zwrócony przez funkcję **`sprawdz_pola()`** jest inny niż **`None`**, przekazywany jest do funkcji **`postaw_znak()`**, której zadaniem jest zapisanie w **`POLU_GRY`** pod otrzymanym indeksem wartości symbolizującej znak komputera (czyli 2) oraz nadanie i zwrócenie zmiennej RUCH wskazującej na gracza (wartość 1).

O ile na planszy nie ma układu wygrywającego lub nie ma konieczności blokowania gracza, komputer w pętli losuje przypadkowe pole (**`random.randrange(0,9)`**), dopóki nie znajdzie pustego, i przekazuje jego indeks do funkcji **`postaw_znak()`**.

IV. Główna pętla programu

Programy interaktywne, w tym gry, reagujące na działania użytkownika, takie jak ruchy czy kliknięcia myszą, działają w pętli, której zadaniem jest:

1. przechwycenie i obsługa działań użytkownika, czyli tzw. zdarzeń (ruchy, kliknięcia myszą, naciśnięcie klawiszy),
2. aktualizacja stanu gry (przesunięcia elementów, aktualizacja planszy),
3. aktualizacja wyświetlanego okna (narysowanie nowego stanu gry).

Dopisujemy więc do kodu główną pętlę wraz z obsługą zdarzeń oraz dwie funkcje pomocnicze w niej wywoływane:

Kod IV.1

```
# sprawdź, kto wygrał, a może jest remis?
def kto_wygral():
    # układy wygrywające dla gracza i komputera
    uklad_gracz = [[1,1,1]]
    uklad_komp = [[2,2,2]]

    WYGRANY = sprawdz_pola(uklad_gracz,1) # czy wygrał gracz?
```

```
    if not WYGRANY: # jeżeli gracz nie wygrywa
        WYGRANY = sprawdz_pola(uklad_komp,2) # czy wygrał komputer?

    # sprawdź remis
    if 0 not in POLE_GRY and WYGRANY not in [1,2]:
        WYGRANY = 3

    return WYGRANY

# funkcja wyświetlająca komunikat końcowy
# tworzy nowy obrazek z tekstem, pobiera jego prostokątny obszar
# pozycjonuje go i rysuje w oknie gry
def drukuj_wynik(WYGRANY):
    fontObj = pygame.font.Font('freesansbold.ttf', 16)
    if WYGRANY == 1:
        tekst = u'Wygrał gracz!'
    elif WYGRANY == 2:
        tekst = u'Wygrał komputer!'
    elif WYGRANY == 3:
        tekst = 'Remis!'
    tekst_obr = fontObj.render(tekst, True, (20,255,20))
    tekst_prost = tekst_obr.get_rect()
    tekst_prost.center = (75, 75)
    OKNOGRY.blit(tekst_obr, tekst_prost)

# pętla główna programu
while True:
    # obsługa zdarzeń generowanych przez gracza
    for event in pygame.event.get():
        # przechwyć zamknięcie okna
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

        if WYGRANA == False:
            if RUCH == 1:
                if event.type == MOUSEBUTTONDOWN:
                    if event.button == 1: # jeżeli naciśnięto pierwszy przycisk
                        mouseX, mouseY = event.pos # rozpakowanie tupli
                        pole = ((mouseY//50)*3)+(mouseX//50)  # wylicz indeks klikniętego pola
                        RUCH = postaw_znak(pole, RUCH)
            elif RUCH == 2:
                RUCH = ai_ruch(RUCH)

            WYGRANY = kto_wygral()
            if WYGRANY != None:
                WYGRANA = True


    OKNOGRY.fill((0,0,0))# definicja koloru powierzchni w RGB
    rysuj_plansze()
    rysuj_pole_gry()
    if WYGRANA:
        drukuj_wynik(WYGRANY)
    pygame.display.update()
```

W obrębie głównej pętli programu pętla **`for`** odczytuje kolejne zdarzenia zwracane przez metodę **`pygame.event.get()`**. Jak widać, w pierwszej kolejności obsługujemy wydarzenie typu (właściwość **`.type`**) QUIT, czyli zakończenie aplikacji. Później, o ile nikt nie wygrał (zmienna **`WYGRANA`** ma wartość **`False`**), a kolej na ruch gracza (zmienna **`RUCH`** ma wartość 1), przechwytujemy wydarzenie **`MOUSEBUTTONDOWN`**, tj. kliknięcie myszą. Sprawdzamy, czy naciśnięto pierwszy przycisk, pobieramy współrzędne kursora (**`.pos`**) i wyliczamy indeks klikniętego pola. Na koniec wywołujemy omówioną wcześniej funkcję **`postaw_znak()`**.

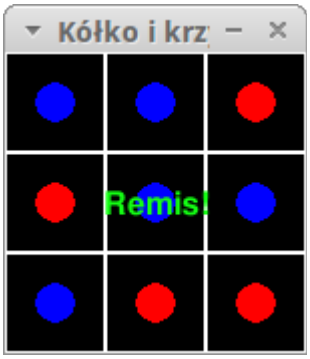
Jeżeli kolej na komputer, uruchamiamy sztuczną inteligencję (`ai_ruch()`).

Po wykonaniu ruchu przez komputer lub gracza trzeba sprawdzić, czy któryś z przeciwników nie wygrał. Korzystamy z funkcji `kto_wygral()`, która definiuje dwa układy wygrywające (`uklad_gracz` i `uklad_komputer`) i za pomocą omówionej wcześniej funkcji `sprawdz_pola()` sprawdza, czy można je odnaleźć w `POLU_GRY`. Na końcu sprawdza możliwość remisu i zwraca wartość symbolizującą wygranego (1, 2, 3) lub `None`, o ile możliwe są kolejne ruchy. Wartość ta wpływa w pętli głównej na zmienną WYGRANA kontrolującej obsługę ruchów gracza i komputera.

Funkcja `drukuj_wynik()` ma za zadanie przygotowanie końcowego napisu. W tym celu tworzy obiekt czcionki z podanego pliku (`pygame.font.Font()`)[1], następnie renderuje nowy obrazek z odpowiednim tekstem (`.render()`), pobiera jego powierzchnię prostokątną (`.get_rect()`), pozycjonują ją (`.center()`) i rysują na głównej powierzchni gry (`.blit()`).

Ostatnie linie kodu wypełniają okno gry kolorem (`.fill()`), wywołują funkcję rysujące planszę (`rysuj_plansze()`), stan gry (`rysuj_pole_gry()`, czyli znaki gracza i komputera), a także ewentualny komunikat końcowy (`drukuj_wynik()`). Funkcja `pygame.display.update()`, która musi być wykonywana na końcu rysowania, aktualizuje obraz gry na ekranie.

Grę możemy uruchomić poleceniem wpisanym w terminalu: `python tictactoe.py`.



Słownik

- **Kanał alfa (ang. alpha channel)** – w grafice komputerowej jest kanałem, który definiuje przezroczyste obszary grafiki. Jest on zapisywany dodatkowo wewnątrz grafiki razem z trzema wartościami barw składowych RGB.
- **Inicjalizacja** – proces wstępnego przypisania wartości zmiennym i obiektom. Każdy obiekt jest inicjalizowany różnymi sposobami zależnie od swojego typu.
- **Iteracja** – czynność powtarzania (najczęściej wielokrotnego) tej samej instrukcji (albo wielu instrukcji) w pętli. Mianem iteracji określa się także operacje wykonywane wewnątrz takiej pętli.
- **Zdarzenie (ang. event)** – zapis zajścia w systemie komputerowym określonej sytuacji, np. poruszenie myszką, kliknięcie, naciśnięcie klawisza.
- **pygame.display.set_mode()** – inicjuje okno lub ekran do wyświetlania, parametry: rozdzielczość w pikselach = (x,y), flagi, głębia koloru.
- **pygame.display.set_caption()** – ustawia tytuł okna, parametr: tekst tytułu.

1 Plik wykorzystywany do wyświetlania tekstu (`freesansbold.ttf`) musi znaleźć się w katalogu ze skryptem.

- **pygame.Surface()** – obiekt reprezentujący dowolny obrazek (grafikę), który ma określoną rozdzielczość (szerokość i wysokość) oraz format pikseli (głębokość, przezroczystość); **SRCALPHA** – oznacza, że format pikseli będzie zawierać ustawienie alfa (przezroczystości); **.fill()** – wypełnia obrazek kolorem; **.get_rect()** – zwraca prostokąt zawierający obrazek, czyli obiekt **Rect**; **.convert_alpha()** – zmienia format pikseli, w tym przezroczystość; **.blit()** – rysuje jeden obrazek na drugim, parametry: źródło, cel.
- **pygame.draw.rect()** – rysuje prostokąt na wskazanej powierzchni, parametry: powierzchnia, kolor, obiekt **Rect**, grubość obramowania.
- **pygame.draw.ellipse()** – rysuje okrągły kształt wewnątrz prostokąta, parametry: przestrzeń, kolor, prostokąt.
- **pygame.font.Font()** – tworzy obiekt czcionki z podanego pliku; **.render()** – tworzy nową powierzchnię z podanym tekstem, parametry: tekst, antyaliasing, kolor, tło.
- **pygame.event.get()** – pobiera zdarzenia z kolejki zdarzeń; **event.type()** – zwraca identyfikator SDL typu zdarzenia, np. KEYDOWN, KEYUP, MOUSEMOTION, MOUSEBUTTONDOWN, QUIT.
- **SDL** (Simple DirectMedia Layer) – międzyplatformowa biblioteka ułatwiająca tworzenie gier i programów multimedialnych.
- **pygame.Rect** – obiekt pygame przechowujący współrzędne prostokąta; **.centerx, .x, .y, .top, .bottom, .left, .right** – wirtualne własności obiektu prostokąta określające jego położenie; **.colliderect()** – metoda sprawdza czy dwa prostokąty nachodzą na siebie.

POĆWICZ SAM

Zmień grę tak, aby zaczynał ją komputer.

Dodaj do gry możliwość rozgrywki wielokrotnej bez konieczności ponownego uruchamiania skryptu.

Zmodyfikuj funkcję rysującą pole gry tak, aby komputer rysował krzyżyki, a nie kółka.