Gra w życie w języku Python

Źródło: Michael Schiffer, domena publiczna.

Co wspólnego mają ze sobą pożary lasów, kształtowanie opinii publicznej oraz biofizyka? Ich wspólnym mianownikiem są automaty komórkowe, które wykorzystuje się do tworzenia modeli ilustrujących konkretne zjawiska.

W e-materiale Gra w życie poznaliśmy jeden z najpopularniejszych automatów komórkowych. Działanie *Gry w życie* do dziś jest analizowane przez specjalistów z wielu różnych dziedzin. Pokazuje ona, że za pomocą kilku prostych reguł, można utworzyć skomplikowane struktury.

Prostota zasad *Gry w życie* sprawia, że jej implementacja nie wymaga wielu linii skomplikowanego kodu. Mimo to gra zadziwia swoją formą oraz interakcjami zachodzącymi na przestrzeni kolejnych pokoleń. W tym e-materiale zaimplementujemy *Grę w życie* w języku Python.

Z implementacją omawianego zagadnienia w pozostałych językach programowania możesz się zapoznać w e-materiałach:

- Gra w życie w języku C++,
- Gra w życie w języku Java.

> **Twoje cele**

- Użyjesz bibliotek i modułów: `NumPy`, `random`, `time`, `os` w języku Python.
- Prześledzisz implementację *Gry w życie* w języku Python bez zastosowania biblioteki graficznej.
- Przeanalizujesz zastosowanie biblioteki graficznej Matplotlib w implementacji *Gry w życie*.

Przeczytaj

Implementacja w języku Python

Do implementacji *Gry w życie* w języku Python wykorzystamy cztery moduły i biblioteki:

- `NumPy` – biblioteka zawierająca funkcje i obiekty matematyczne; użyjemy jej do wygodnej obsługi tablic dwuwymiarowych;
- `random` – moduł odpowiadający za generowanie liczb pseudolosowych wykorzystywanych do inicjalizacji pierwszej populacji;
- `time` – moduł zawierający wiele funkcji związanych z czasem oraz datami; użyjemy go do zatrzymywania pętli programu na określoną liczbę sekund;
- `os` – moduł zawierający funkcje służące do komunikacji z systemem operacyjnym; wykorzystamy go do czyszczenia okna konsoli.

Na początku je zaimportujemy:

```
1 import numpy as np
2 import random
3 import time
4 import os
```

NumPy pozwala na wygodną i wydajną obsługę wielowymiarowych tablic. W automacie komórkowym pracujemy na tablicach dwuwymiarowych.

> **Ciekawostka**
>
> Biblioteka NumPy ma wiele zastosowań, wśród nich znajdują się między innymi: zaawansowane obliczenia matematyczne, analizy danych oraz uczenie maszynowe. Cechuje ją prostota użycia oraz wydajność.

Potrzebne będą nam również zmienne globalne do przechowywania informacji o stanie komórki oraz rozmiaru siatki.

```
1 ROZMIAR_SIATKI = 10
2 ZYWA = True
3 MARTWA = False
```

Dodatkowo w programie wykorzystamy następujące funkcje:

- `wyczysc_terminal()` – funkcja czyszcząca konsolę;

- `stworzSiatke()` – funkcja inicjalizująca siatkę oraz generująca pierwszą populację komórek. Siatkę inicjalizujemy, tworząc dwuwymiarową tablicę. Pierwsza populacja generowana jest w taki sposób, że dla każdej komórki losujemy liczbę od 0 do 4. Jeśli wylosowana zostanie liczba 0, dana komórka oznaczona jest jako żywa;
- `wyswietlSiatke()` – funkcja wypisująca aktualną zawartość siatki na ekran terminala;
- `ewolucja()` – funkcja przeprowadzająca proces ewolucji komórek znajdujących się w siatce zgodnie z zasadami *Gry w życie;*
- `policzZywychSasiadow()` – funkcja zwracająca dla danej komórki liczbę otaczających ją żywych sąsiadów.

Zaczniemy od stworzenia funkcji `wyczysc_terminal()`. Do wyczyszczenia okna konsoli, po poprzednim wypisaniu zawartości siatki, użyjemy odpowiednich poleceń systemowych:

- dla systemu Windows: `cls`,
- dla systemu Linux: `clear`.

Wywołujemy funkcję `system()`, zawartą w module `os`. Nazwę systemu operacyjnego, na którym działamy, sprawdzimy poprzez zmienną `os.name`. Dla systemu Windows będzie ona równa `nt`. Chcemy, aby program dobrał odpowiednią instrukcję dla danego systemu.

```
# Instrukcja wyczyści ekran konsoli
os.system('cls' if os.name == 'nt' else 'clear')
```

Następnie zapiszemy funkcję `stworzSiatke()`. Planszę będzie reprezentowała dwuwymiarowa tablica, przechowująca informację na temat tego, które komórki są żywe (wartość `True`), a które martwe (wartość `False`). Zaczniemy od wypełnienia tej tablicy zerami, wykorzystując poniższą instrukcję:

```
tablica = numpy.zeros((liczba_wierszy, liczba_kolumn))
```

W naszym programie będzie miała następującą postać:

```
def stworz_siatke():
    siatka = np.zeros((ROZMIAR_SIATKI, ROZMIAR_SIATKI))
```

Potrzebujemy informacji, które z komórek są żywe, a które martwe w stanie początkowym. Możemy to ustalić na wiele sposobów, między innymi:

- wczytać zdefiniowane ustawienie siatki, np. z pliku,

- wylosować dowolne ustawienie.

Wygenerujemy pierwszą populację, korzystając z liczb pseudolosowych. Losowanie polega na wylosowaniu liczby z zakresu `<0, 4>` i ustawieniu jej na pozycji `siatka[i, j]` wartości `ZYWA`, jeżeli wylosowana liczba to 0, lub wartości `MARTWA` w pozostałych przypadkach. Funkcja zwraca stworzoną siatkę.

Do wygenerowania pseudolosowej liczby całkowitej z danego zakresu służy funkcja `randint()`. Oto jej przykładowe zastosowanie:

```
# Instrukcja wypisze losową liczbę
# całkowitą z zakresu [1, 100]
print(random.randint(1, 100))
```

Realizacja w programie:

```
def stworz_siatke():
    siatka = np.zeros((ROZMIAR_SIATKI, ROZMIAR_SIATKI))

    for i in range(ROZMIAR_SIATKI):
        for j in range(ROZMIAR_SIATKI):
            temp = random.randint(0, 4)

            if temp == 0:
                siatka[i, j] = ZYWA
            else:
                siatka[i, j] = MARTWA

    return siatka
```

Kolejna funkcja, którą zapiszemy, to funkcja `policz_zywych_sasiadow()`. Funkcja ta dla zadanej pozycji `siatka[y, x]` zwraca liczbę żywych sąsiadów tej pozycji. Funkcja przechodzi po kwadracie 3 x 3 dookoła komórki `[y, x]`, zliczając liczbę żywych sąsiadów. Następnie odejmuje 1, jeżeli komórka `[y, x]` jest żywa, ponieważ nie powinna zostać policzona jako swój sąsiad. Funkcja zwraca liczbę żywych sąsiadów.

```
def policz_zywych_sasiadow(y, x, siatka):
    ile = 0

    for i in range(y - 1, y + 2):
```

```
        for j in range(x - 1, x + 2):
            if 0 <= i < ROZMIAR_SIATKI and 0 <= j < ROZMIAR_SIATK
                ile += 1

    if siatka[y, x] == ZYWA:
        ile -= 1

    return ile
```

W następnym kroku zapisujemy funkcję `ewolucja()`. Dla danej siatki dokonuje ona jednej iteracji ewolucji. Funkcja tworzy tablicę `ile_sasiadow` o wymiarach takich jak siatka, przechowującą liczbę żywych sąsiadów dla każdej komórki w siatce. Następnie przechodzi po każdej komórce i – w zależności od jej stanu (żywej lub martwej) i liczby żywych sąsiadów – wykonuje odpowiednią akcję:

- Jeżeli komórka jest martwa i ma dokładnie 3 sąsiadów, ożywa.
- Jeżeli komórka jest żywa i ma mniej niż 2 sąsiadów, umiera z samotności.
- Jeżeli komórka jest żywa i ma więcej niż 3 sąsiadów, umiera z przeludnienia.
- W pozostałych przypadkach stan komórki się nie zmienia.

Funkcja `wyswietl_siatke()` służy do wyświetlania zawartości siatki na ekranie w postaci graficznej. W tym celu na podstawie wartości komórek używa znaków Unicode o kodach `U+25A0` (czarny kwadrat) i `U+25A1` (biały kwadrat), aby zbudować wizualizację siatki na ekranie.

W tej wersji *Gry w życie* nie będziemy używać biblioteki graficznej. Zastąpimy ją, wypisując informacje o stanie planszy w oknie terminala. Wykorzystamy czarne i białe kwadraty, które są znakami Unicode.

```
print("Pełny kwadrat: " + u'\u25a0')
print("Pusty kwadrat: " + u'\u25a1')
```

Po wywołaniu kodu w terminalu ujrzymy:

```
Pełny kwadrat: ■
Pusty kwadrat: □
```

Funkcja `main()` jest główną funkcją programu i odpowiada za sterowanie automatem komórkowym. W pierwszym kroku tworzy ona siatkę, wywołując funkcję `stworz_siatke()`. Następnie w nieskończonej pętli wywołuje funkcję

`wyswietl_siatke(siatka)` do wyświetlenia aktualnego stanu siatki, następnie wywołuje funkcję `ewolucja(siatka)` do wyznaczenia nowego stanu siatki, a na końcu wykonuje funkcję `time.sleep(0.5)`, która opóźnia program na pół sekundy, aby umożliwić użytkownikowi obserwowanie zmian w siatce. Pętla nieskończona powoduje, że program działa, dopóki nie zostanie zatrzymany przez użytkownika.

Do spowolnienia programu użyjemy funkcji `sleep()` z modułu `time`.

```
# Instrukcja zatrzyma program na dokładnie
# trzy sekundy
time.sleep(3)
```

Implementacja *Gry w życie* w języku Python bez użycia biblioteki graficznej wygląda następująco:

```
# Najpierw importujemy potrzebne moduły
import numpy as np
import random
import time
import os

# Inicjalizujemy przydatne zmienne globalne
ROZMIAR_SIATKI = 10
ZYWA = True
MARTWA = False

def wyczysc_terminal():
    os.system('cls' if os.name == 'nt' else 'clear')

def stworz_siatke():
    # Tworzymy dwuwymiarową tablicę za pomocą
    # modułu NumPy
    siatka = np.zeros((ROZMIAR_SIATKI, ROZMIAR_SIATKI))

    # Generujemy pierwszą populacje komórek
    for i in range(ROZMIAR_SIATKI):
        for j in range(ROZMIAR_SIATKI):
            # Losujemy liczbę w zakresie [0, 4]
            temp = random.randint(0, 4)

```

```
            # Komórka staje się żywa, gdy wylosujemy zero
            if temp == 0:
                siatka[i, j] = ZYWA
            else:
                siatka[i, j] = MARTWA

    return siatka


def policz_zywych_sasiadow(y, x, siatka):
    ile = 0

    # Przechodzimy po kwadracie 3 x 3 dookoła komórki
    for i in range(y - 1, y + 2):
        for j in range(x - 1, x + 2):
            # Sprawdzamy, czy nie wyszliśmy poza zakres siatki i
            if 0 <= i < ROZMIAR_SIATKI and 0 <= j < ROZMIAR_SIAT
                ile += 1

    # Ponieważ sprawdzaliśmy również podaną komórkę,
    # to jeżeli ona jest żywa, musimy ją odjąć
    # - sama komórka nie jest swoim sąsiadem
    if siatka[y, x] == ZYWA:
        ile -= 1

    return ile


def ewolucja(siatka):
    # Inicjalizujemy tablicę przechowującą liczbę
    # żywych sąsiadów dla danej komórki
    ile_sasiadow = np.zeros((ROZMIAR_SIATKI, ROZMIAR_SIATKI))

    # Dla każdej komórki liczymy liczbę żywych
    # sąsiadów
    for i in range(ROZMIAR_SIATKI):
        for j in range(ROZMIAR_SIATKI):
            ile_sasiadow[i, j] = policz_zywych_sasiadow(i, j, si

    # Dokonujemy ewolucji każdej z komórek zgodnie
    # z regułami Gry w życie
    for i in range(ROZMIAR_SIATKI):
```

```
68          for j in range(ROZMIAR_SIATKI):
69              # Komórka jest martwa i ma 3 sąsiadów - oży
70              if siatka[i, j] == MARTWA:
71                  if ile_sasiadow[i, j] == 3:
72                      siatka[i, j] = ZYWA
73
74                  # Komórka jest żywa i ma za mało sąsiadów - gini
75                  elif siatka[i, j] == ZYWA and ile_sasiadow[i, j]
76                      siatka[i, j] = MARTWA
77
78                  # Komórka jest żywa i ma za dużo sąsiadów (przel
79                  elif siatka[i, j] == ZYWA and ile_sasiadow[i, j]
80                      siatka[i, j] = MARTWA
81              elif siatka[i, j] == ZYWA and ile_sasiadow[i, j] > 3
82                  siatka[i, j] = MARTWA
83
84
85  def wyswietl_siatke(siatka):
86      # Czyścimy terminal po poprzednim wypisaniu
87      # zawartości siatki
88      wyczysc_terminal()
89
90      # Wypisujemy zawartość siatki
91      for i in range(ROZMIAR_SIATKI):
92          for j in range(ROZMIAR_SIATKI):
93              if siatka[i, j] == ZYWA:
94                  print(u'\u25a0', end=' ')
95              else:
96                  print(u'\u25a1', end=' ')
97
98          print('\n', end='')
99
100     print('\n', end='')
101
102
103 def main():
104     siatka = stworz_siatke()
105
106     while True:
107         wyswietl_siatke(siatka)
108         ewolucja(siatka)
109         # Zatrzymujemy program na pół sekundy,
```

```
110         # aby móc lepiej obserwować działanie
111         # naszego automatu komórkowego
112         time.sleep(0.5)
113 
114 if __name__ == '__main__':
115     main()
```

Słownik

klasa

rodzaj szablonu (wzorca) opisującego zbiór zmiennych oraz funkcji służących do ich przetwarzania; korzystając z klasy można tworzyć obiekty, będące rozbudowanymi odpowiednikami zmiennych; można powiedzieć, że obiekt jest zmienną typu klasa

metoda

funkcja należąca do klasy i służąca do operowania na jej obiektach (zmiennych)

lista

struktura danych; w języku Python wykorzystywana jest do implementacji tablic

liczba pseudolosowa

liczba wygenerowana za pomocą generatora pseudolosowego; ciąg liczb przez niego wygenerowanych wydaje się losowy dla obserwatora

Matplotlib

biblioteka służąca do tworzenia wykresów w języku Python; posiada również funkcje odpowiadające za wyświetlanie grafik oraz animacji; składnią oraz nazwami funkcji jest bardzo zbliżona do języka programowania *Matlab*

Prezentacja multimedialna

Polecenie 1

Zapoznaj się z symulacją interaktywną *Gry w życie*. W narzędziu można zmieniać kolor komórek w każdym kroku, a także obserwować przebieg gry. Przetestuj narzędzie dla różnych aparatów komórkowych.

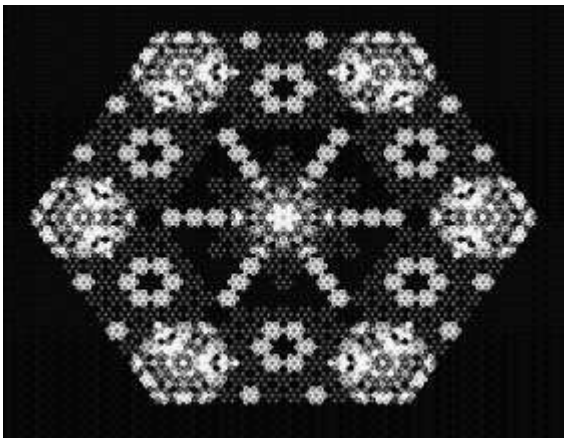
Implementacja z użyciem biblioteki graficznej

Polecenie 2

Zapoznaj się z prezentacją multimedialną, przedstawiającą implementację *Gry w życie* z użyciem biblioteki graficznej.

Ważne!

By program zadziałał poprawnie, uruchom go na swoim komputerze np. za pomocą IDE PyCharm.



Model matematyczny na przykładzie automatu komórkowego, którego struktury opisane są przez siatkę komórek.
Źródło: pl.wikipedia.org, domena publiczna.

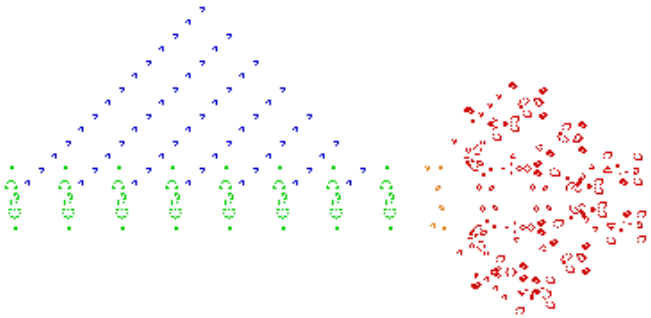
Materiał audio dostępny pod adresem:

1

https://zpe.gov.pl/b/PfHPZZRiE

Do graficznej implementacji automatu komórkowego użyjemy popularnej biblioteki do wyświetlania grafik, wykresów i animacji o nazwie Matplotlib.

2



John Horton Conway, brytyjski matematyk, w 1970 roku wymyślił *Grę w życie* na przykładzie aparatu komórkowego. Odbywa się ona na kwadratowej siatce komórek, gdzie każda komórka może być albo żywa, albo martwa.
Źródło: en.wikipedia.org, domena publiczna.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PfHPZZRiE

Większość przedstawionego wcześniej kodu pozostaje bez zmian. Zmodyfikujemy jedynie kod funkcji `ewolucja()` oraz usuniemy z programu funkcję `wyswietlSiatke()`. Zmienimy również postać głównej pętli gry. Zamiast pętli `while` wykorzystamy funkcję `FuncAnimation()` z modułu `matplotlib.animation`.

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PfHPZZRiE

Zaczniemy od importu potrzebnych modułów. Tym razem będziemy potrzebować

dodatkowych modułów z biblioteki `matplotlib`.

```
# Najpierw zaimportujemy potrzebne nam moduły
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import random
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PfHPZZRiE

Kolejne kroki powtarzamy zgodnie z tym, co zostało zaprezentowane w sekcji „Przeczytaj”.

Inicjalizujemy zmienne i tworzymy siatkę.

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PfHPZZRiE

W dalszej części funkcji generujemy, wykorzystując liczby pseudolosowe, pierwszą populację komórek.

Definiujemy funkcję `policz_zywych_sasiadow()`.

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PfHPZZRiE

Definiujemy funkcję `ewolucja()`. Na tym etapie pojawiają się większe zmiany w stosunku do programu z sekcji „Przeczytaj”.

Funkcja ta przyjmuje trzy argumenty: `frameNum` (liczbę klatek), `img` (obiekt `matplotlib` reprezentujący obraz siatki) oraz `siatka` (tablica NumPy przechowująca stan komórek).

Na początku funkcja inicjalizuje tablicę `ilu_sasiadow` jako tablicę zer. Następnie w dwóch zagnieżdżonych pętlach `for` dla każdej komórki siatki obliczana jest liczba żywych sąsiadów poprzez wywołanie funkcji `policz_zywych_sasiadow`, która zwraca liczbę żywych komórek w sąsiedztwie danej komórki. Wynik ten zapisywany jest w tablicy `ilu_sasiadow`.

Następnie w kolejnych zagnieżdżonych pętlach `for` dokonywana jest ewolucja każdej z komórek zgodnie z regułami Gry w życie. Pierwszy warunek sprawdza, czy komórka jest martwa i ma trzech żywych sąsiadów – wtedy ożywa. Drugi warunek sprawdza, czy komórka jest żywa i ma mniej niż dwóch żywych sąsiadów – wtedy ginie. Trzeci warunek sprawdza, czy komórka jest żywa i ma więcej niż trzech żywych sąsiadów (przeludnienie) – wtedy też ginie.

Na końcu funkcja aktualizuje obraz reprezentujący siatkę komórek danymi po ewolucji i zwraca ten obraz.

```
1 def ewolucja(frameNum,
  img, siatka):
2     # Inicjalizujemy
  tablicę przechowującą
  liczbę
3     # żywych sąsiadów dla
  danej komórki
```

```
    ilu_sasiadow = np.zeros((ROZMIAR_SIATKI, ROZMIAR_SIATKI))

    # Dla każdej komórki liczymy liczbę żywych
    # sąsiadów
    for i in range(ROZMIAR_SIATKI):
        for j in range(ROZMIAR_SIATKI):
            ilu_sasiadow[i, j] = policz_zywych_sasiadow(i, j, siatka)

    # Dokonujemy ewolucji każdej z komórek zgodnie
    # z regułami Gry w życie
    for i in range(ROZMIAR_SIATKI):
        for j in range(ROZMIAR_SIATKI):
            # Komórka jest martwa i ma 3 sąsiadów - ożywa
            if siatka[i, j] == MARTWA and ilu_sasiadow[i, j] == 3:
                siatka[i, j] = ZYWA

            # Komórka jest żywa i ma za mało sąsiadów - ginie
            elif siatka[i, j] == ZYWA and ilu_sasiadow[i, j] < 2:
                siatka[i, j] = MARTWA

            # Komórka jest żywa i ma za dużo sąsiadów (przeludnienie) - ginie
```

```
            elif siatka[i, j] == ZYWA and ilu_sasiadow[i, j] > 3:
                siatka[i, j] = MARTWA

    # Aktualizujemy obraz danymi po ewolucji
    img.set_data(siatka)

    return img
```


Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PfHPZZRiE

Definiujemy funkcję `main`. To główna funkcja programu, która wykonuje kilka zadań. Tworzy początkową siatkę, wykorzystując funkcję `stworz_siatke()`. Następnie tworzy nowe okno wykresu przy użyciu funkcji `subplots()` z biblioteki `matplotlib` i zapisuje zwrócone przez nią obiekty `fig` i `ax`. Funkcja w kolejnym kroku wstawia siatkę jako obraz do okna przy użyciu funkcji `imshow()` z biblioteki `matplotlib` i zapisuje zwrócony przez nią obiekt `img`. W następnym kroku funkcja tworzy animację, wykorzystując funkcję `FuncAnimation()` z biblioteki `matplotlib`. Wywołuje ona funkcję `ewolucja()` jako funkcję, która ma być wywoływana w każdym kroku animacji, a także przekazuje obiekt `img` i siatkę jako argumenty dla tej funkcji. Określa liczbę klatek animacji (`frames`), przerwę między nimi (`interval`) oraz liczbę klatek do zapisania (`save_count`).

Ustawia widoczność okna animacji za pomocą funkcji `show()` z biblioteki `matplotlib`.

Ostatecznie, jeśli kod jest wykonywany jako program, funkcja main() jest wywoływana.

```
def main():
    siatka = stworz_siatke()

    # Tworzymy okno wykresu
    fig, ax = plt.subplots()

    # Wstawiamy do okna naszą siatkę
    img = ax.imshow(siatka)

    # Animujemy ją wykorzystując funkcję z biblioteki
    # matplotlib. Zastępuje to naszą główną pętlę
    # gry z poprzedniej implementacji
    ani = animation.FuncAnimation(fig, ewolucja, fargs=(img, siatka), frames=10, interval=150, save_count=50)

    # Ustawiamy widoczność naszego okna z animacją
    plt.show()
if __name__ == '__main__':
    main()
```


Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PfHPZZRiE

Cały kod programu:

```
1
2  # Najpierw zaimportujemy
   potrzebne nam moduły
3  import numpy as np
4  import matplotlib.pyplot
   as plt
5  import
   matplotlib.animation as
   animation
6  import random
7
8  # Inicjalizujemy
   przydatne zmienne
   globalne
9  ROZMIAR_SIATKI = 50
10 MARTWA = 0
11 ZYWA = 1
12
13 def stworz_siatke():
14     # Tworzymy
   dwuwymiarową tablicę przy
   pomocy
15     # modułu NumPy
16     siatka =
   np.zeros((ROZMIAR_SIATKI,
   ROZMIAR_SIATKI))
17
18     # Generujemy pierwszą
   populacje komórek
19     for i in
   range(ROZMIAR_SIATKI):
20         for j in
   range(ROZMIAR_SIATKI):
21             # Losujemy
   liczbę w zakresie [0, 4]
22             temp =
   random.randint(0, 4)
23
24             # Komórka
   staje się żywa, gdy
   wylosujemy zero
25             if temp == 0:
```

```
                siatka[i, j] = ZYWA
            else:
                siatka[i, j] = MARTWA

    return siatka


def policz_zywych_sasiadow(y, x, siatka):
    ile = 0

    # Przechodzimy po kwadracie 3x3 dookoła komórki
    for i in range(y - 1, y + 2):
        for j in range(x - 1, x + 2):
            # Sprawdzamy, czy nie wyszliśmy poza zakres siatki i czy sąsiad jest żywy
            if 0 <= i < ROZMIAR_SIATKI and 0 <= j < ROZMIAR_SIATKI and siatka[i, j] == ZYWA:
                ile += 1

    # Ponieważ sprawdzaliśmy również podaną komórkę,
    # to jeżeli ona jest żywa, musimy ją odjąć
    # - sama komórka nie jest swoim sąsiadem
    if siatka[y, x] == ZYWA:
        ile -= 1

    return ile


```

```
52 def ewolucja(frameNum,
   img, siatka):
53     # Inicjalizujemy
   tablicę przechowującą
   liczbę
54     # żywych sąsiadów dla
   danej komórki
55     ilu_sasiadow =
   np.zeros((ROZMIAR_SIATKI,
   ROZMIAR_SIATKI))
56
57     # Dla każdej komórki
   liczymy liczbę żywych
58     # sąsiadów
59     for i in
   range(ROZMIAR_SIATKI):
60         for j in
   range(ROZMIAR_SIATKI):
61
   ilu_sasiadow[i, j] =
   policz_zywych_sasiadow(i,
   j, siatka)
62
63     # Dokonujemy ewolucji
   każdej z komórek zgodnie
64     # z regułami Gry w
   życie
65     for i in
   range(ROZMIAR_SIATKI):
66         for j in
   range(ROZMIAR_SIATKI):
67             # Komórka
   jest martwa i ma 3
   sąsiadów - ożywa
68             if siatka[i,
   j] == MARTWA and
   ilu_sasiadow[i, j] == 3:
69                 siatka[i,
   j] = ZYWA
70
71             # Komórka
   jest żywa i ma za mało
   sąsiadów - ginie
72             elif
   siatka[i, j] == ZYWA and
```

```
ilu_sasiadow[i, j] < 2:
                siatka[i,
j] = MARTWA

            # Komórka
jest żywa i ma za dużo
sąsiadów (przeludnienie)
- ginie
            elif
siatka[i, j] == ZYWA and
ilu_sasiadow[i, j] > 3:
                siatka[i,
j] = MARTWA

    # Aktualizujemy obraz
danymi po ewolucji
    img.set_data(siatka)

    return img


def main():
    siatka =
stworz_siatke()

    # Tworzymy okno
wykresu
    fig, ax =
plt.subplots()

    # Wstawiamy do okna
naszą siatkę
    img =
ax.imshow(siatka)

    # Animujemy ją
wykorzystując funkcję z
biblioteki
    # matplotlib.
Zastępuje to naszą główną
pętlę
    # gry z poprzedniej
implementacji
    ani =
animation.FuncAnimation(f
```

```
    ig, ewolucja, fargs=(img,
    siatka), frames=10,
    interval=150,
    save_count=50)

    # Ustawiamy
widoczność naszego okna z
animacją
    plt.show()


if __name__ ==
'__main__':
    main()

```

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wskaż, której biblioteki graficznej użyjemy do wizualizacji działania *Gry w życie* w języku Python.

- ◯ `Pillow`
- ◯ `VPython`
- ◯ `Bokeh`
- ◯ `Matplotlib`
- ◯ `PLPlot`

Ćwiczenie 2

Zaimportowano bibliotekę `os` w następujący sposób:

```
1 import os
```

Zaznacz instrukcję w języku Python odpowiadającą za czyszczenie ekranu konsoli w systemie Windows w sytuacji, gdy biblioteka `os` została zaimportowana w zaprezentowany sposób sposób.

- ◯ `os("cls")`
- ◯ `system.os("cls")`
- ◯ `os.system("cls")`
- ◯ `system("cls")`

Ćwiczenie 3

Zaznacz funkcję, którą wykorzystujemy do tworzenia animacji.

- ◯ `Animate()`
- ◯ `FuncAnimate()`
- ◯ `FuncAnimation()`
- ◯ `Animation()`

FuncAnimation

Ćwiczenie 4

Zaznacz funkcję z modułu `matplotlib.pyplot`, która odpowiada za pokazanie okna z wykresem.

- ◯ `plotShow()`
- ◯ `changeVisibility()`
- ◯ `show()`
- ◯ `makeVisible()`

Ćwiczenie 5

Połącz w pary nazwy modułów z ich opisami.

`time`	moduł zawierający funkcje związane z czasem oraz datami
`random`	moduł zawierający funkcje służące do komunikacji z systemem operacyjnym
`os`	biblioteka zawierająca funkcje i obiekty matematyczne
`NumPy`	moduł odpowiadający za generowanie liczb pseudolosowych

Ćwiczenie 6

Wskaż instrukcję z modułu NumPy, która tworzy dwuwymiarową tablicę wypełnioną zerami o wymiarach `liczba_wierszy` na `liczba_kolumn`.

- ◯ `tablica = numpy.getZerosTable((liczba_wierszy, liczba_kolumn)))`
- ◯ `tablica = numpy.empty((liczba_wierszy, liczba_kolumn))`
- ◯ `tablica = numpy.zeros((liczba_wierszy, liczba_kolumn))`
- ◯ `tablica = numpy.zerosTable((liczba_wierszy, liczba_kolumn))`

NumPy

Ćwiczenie 7

Uzupełnij kod o funkcję `ewolucja()`, której zadaniem jest przeprowadzenie kroku ewolucji komórek znajdujących się na planszy `siatka`, zgodnie z zasadami *Gry w życie*. Następnie przetestuj działanie funkcji dla przykładowej siatki:

```
siatka = [
    [1, 1, 0, 0, 0],
    [0, 0, 1, 0, 1],
    [1, 0, 1, 1, 0],
    [1, 0, 0, 1, 0],
    [1, 1, 0, 1, 1]
]
```

Specyfikacja:

Dane:

- `ROZMIAR_SIATKI` – wysokość i szerokość planszy; liczba naturalna
- `siatka` – plansza, na której ma się odbyć ewolucja; dwuwymiarowa lista

Wynik:

- plansza `siatka` zmodyfikowana o krok ewolucji, zgodnie z regułami *Gry w życie*

Wykonaj poniższe polecenie implementacyjne.

Twoje zadania

1. Napisz funkcję `ewolucja()`, która odpowiada za przeprowadzenie procesu ewolucji komórek znajdujących się na planszy zgodnie z regułami *Gry w życie*. Do policzenia żywych sąsiadów danej komórki możesz wykorzystać funkcję `policzZywychSasiadow()`.

```
ROZMIAR_SIATKI = 5
ZYWA = 1
```

```
MARTWA = 0

def policzZywychSasiadow(y, x, siatka):
    ile = 0

    for i in range(y - 1, y + 2):
        for j in range(x - 1, x + 2):
            if i > 0 and j > 0 and i < ROZMIAR_SIATKI and j < ROZMIAR_SIATKI and siatka[i][j] == ZYWA:
                ile += 1

    if siatka[y][x] == ZYWA:
```

```

```

Dla nauczyciela

Autor: Bartosz Zadrożny
Przedmiot: Informatyka

Temat: Gra w życie w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi, w tym: znajomość zasad działania urządzeń cyfrowych i sieci komputerowych oraz wykonywania obliczeń i programów.

IV. Rozwijanie kompetencji społecznych, takich jak: komunikacja i współpraca w grupie, w tym w środowiskach wirtualnych, udział w projektach zespołowych oraz zarządzanie projektami.

Treści nauczania – wymagania szczegółowe

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

IV. Rozwijanie kompetencji społecznych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) analizuje i charakteryzuje wpływ trendów w historycznym rozwoju pojęć, metod informatyki oraz technologii na możliwości rozwiązywania problemów teoretycznych i praktycznych;

3) przygotowuje się do świadomego wyboru kierunku i zakresu dalszego kształcenia, głównie informatycznego, z myślą o przyszłej karierze zawodowej.

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Użyjesz bibliotek i modułów: `NumPy`, `random`, `time`, `os` w języku Python.
- Prześledzisz implementację *Gry w życie* w języku Python bez zastosowania biblioteki graficznej.
- Przeanalizujesz zastosowanie biblioteki graficznej Matplotlib w implementacji *Gry w życie*.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. Chętne lub wybrane osoby przygotowują krótkie wystąpienia na temat aparatów komórkowych.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Gra w życie w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Wyświetlenie przez nauczyciela tematu i celów lekcji. Określenie wiążących dla uczniów kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. Przedstawienie uczniowskich prezentacji.
2. Chętna lub wybrana osoba przypomina zasady gry w życie.
3. **Praca z tekstem**. Uczniowie analizują implementację zaprezentowaną w sekcji „Przeczytaj". Zadają pytania odnośnie do analizy, na które w razie potrzeby nauczyciel odpowiada.
4. **Praca z multimedium.** Uczniowie zapoznają się z symulacją interaktywną i wykonują Polecenie 1.
5. Uczniowie analizują prezentację multimedialną z sekcji „Prezentacja multimedialna". Zadają pytania, na które w razie potrzeby nauczyciel odpowiada.
6. **Ćwiczenie umiejętności.** Uczniowie wykonują Ćwiczenia 1–6 z sekcji „Sprawdź się".

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie rozwiązują Ćwiczenie 7 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj", „Prezentacja multimedialna", „Sprawdź się" jako materiał do lekcji powtórkowej.
- Jeśli nauczyciel uzna, że klasa sobie poradzi z zadaniem, może wprowadzić implementację gry w życie z wykorzystaniem biblioteki graficznej (sekcja „Prezentacja multimedialna").