Problem ośmiu hetmanów w języku Python

Źródło: Hassan Pasha, domena publiczna.

Reguły gry w szachy nie są skomplikowane, jednak każda partia składa się z wielu decyzji taktycznych. Przekłada się to na niepowtarzalność rozgrywki. Szachownica i figury szachowe nie muszą jednak służyć do rozegrania klasycznej partii. Można dzięki nim przygotować łamigłówkę, której rozwiązanie wymaga ustawienia wybranych figur tak, aby spełnione zostały konkretne warunki.

Jednym z takich zadań jest problem ośmiu hetmanów. Ten e-materiał poświęcimy napisaniu rozwiązującego ten problem programu w języku Python. Wykorzystamy przy tym rekurencję z nawrotami.

Podstawowe informacje na temat omawianego zagadnienia znajdziesz w e-materiale Problem ośmiu hetmanów. Implementację w innych językach programowania przedstawiamy w pozostałych e-materiałach z tej serii:

- Problem ośmiu hetmanów w języku C++,
- Problem ośmiu hetmanów w języku Java.

Więcej zadań? Sięgnij do: Problem ośmiu hetmanów – zadania maturalne.

> **Twoje cele**
>
> - Zaimplementujesz algorytm rozwiązujący problem ośmiu hetmanów.

- Przeanalizujesz algorytm zaimplementowany w języku Python.
- Wyjaśnisz, na czym polega algorytm z nawrotami.

Film samouczek

Problem 1

Napisz program, który rozmieści n hetmanów na szachownicy o rozmiarze n wierszy i n kolumn tak, aby nie atakowały się wzajemnie.

Zasady atakowania się hetmanów wyglądają następująco:

- Hetman może poruszać się dowolną liczbę pól do przodu, do tyłu, w prawo, w lewo i po skosach.
- Hetman zbija inne figury znajdujące się na jego drodze.

Działanie swojego programu przetestuj dla `n = 8`.

Specyfikacja:

Dane:

- n – liczba hetmanów oraz liczba wierszy i kolumn szachownicy; liczba naturalna dodatnia; `n > 3`

Wynik:

Program wypisuje na standardowe wyjście kolejne, dozwolone układy ustawień hetmanów.

Przykładowy wynik:

```
[0, 1, 5, 8, 6, 3, 7, 2, 4]
[0, 1, 6, 8, 3, 7, 4, 2, 5]
[0, 1, 7, 4, 6, 8, 2, 5, 3]
[0, 1, 7, 5, 8, 2, 4, 6, 3]
[0, 2, 4, 6, 8, 3, 1, 7, 5]
[0, 2, 5, 7, 1, 3, 8, 6, 4]
[0, 2, 5, 7, 4, 1, 8, 6, 3]
[0, 2, 6, 1, 7, 4, 8, 3, 5]
[0, 2, 6, 8, 3, 1, 4, 7, 5]
[0, 2, 7, 3, 6, 8, 5, 1, 4]
```

```
[0, 2, 7, 5, 8, 1, 4, 6, 3]
[0, 2, 8, 6, 1, 3, 5, 7, 4]
[0, 3, 1, 7, 5, 8, 2, 4, 6]
[0, 3, 5, 2, 8, 1, 7, 4, 6]
[0, 3, 5, 2, 8, 6, 4, 7, 1]
[0, 3, 5, 7, 1, 4, 2, 8, 6]
[0, 3, 5, 8, 4, 1, 7, 2, 6]
[0, 3, 6, 2, 5, 8, 1, 7, 4]
[0, 3, 6, 2, 7, 1, 4, 8, 5]
[0, 3, 6, 2, 7, 5, 1, 8, 4]
[0, 3, 6, 4, 1, 8, 5, 7, 2]
[0, 3, 6, 4, 2, 8, 5, 7, 1]
[0, 3, 6, 8, 1, 4, 7, 5, 2]
[0, 3, 6, 8, 1, 5, 7, 2, 4]
[0, 3, 6, 8, 2, 4, 1, 7, 5]
[0, 3, 7, 2, 8, 5, 1, 4, 6]
[0, 3, 7, 2, 8, 6, 4, 1, 5]
[0, 3, 8, 4, 7, 1, 6, 2, 5]
[0, 4, 1, 5, 8, 2, 7, 3, 6]
[0, 4, 1, 5, 8, 6, 3, 7, 2]
[0, 4, 2, 5, 8, 6, 1, 3, 7]
[0, 4, 2, 7, 3, 6, 8, 1, 5]
[0, 4, 2, 7, 3, 6, 8, 5, 1]
[0, 4, 2, 7, 5, 1, 8, 6, 3]
[0, 4, 2, 8, 5, 7, 1, 3, 6]
[0, 4, 2, 8, 6, 1, 3, 5, 7]
[0, 4, 6, 1, 5, 2, 8, 3, 7]
[0, 4, 6, 8, 2, 7, 1, 3, 5]
[0, 4, 6, 8, 3, 1, 7, 5, 2]
```

Polecenie 1

Porównaj swoje rozwiązanie z przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/Rs9xD7sFmh6jK

Lekcja poświęcona problemowi ośmiu hetmanów w języku Python.

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 1.28 KB w języku polskim

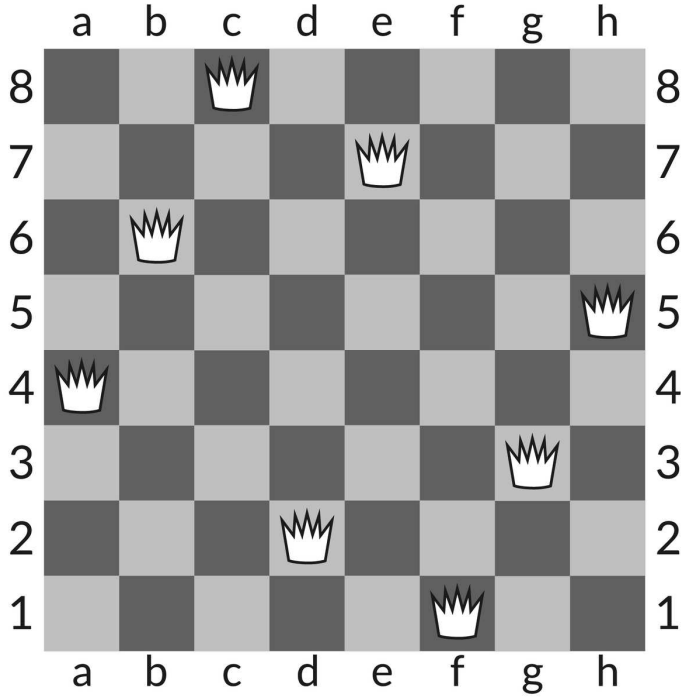
> **Polecenie 2**
>
> Dodaj do swojego programu komentarze tak, żeby był zrozumiały dla osoby, która nie potrafi programować.

Przeczytaj

Problem ośmiu hetmanów

W poprzednim e-materiale omówiliśmy, na czym polega problem ośmiu hetmanów. Przedstawiliśmy także algorytm rozwiązujący ten problem. W sekcji „Film samouczek" zaimplementowaliśmy algorytm.

W tej sekcji przedstawimy alternatywną implementację algorytmu, pokażemy dlaczego metoda siłowa (ang. *brute-force*), generująca wszystkie możliwe ustawienia hetmanów nie jest optymalna, a także zaprezentujemy graficzną planszę z wynikiem działania algorytmu.



Jedno z wielu możliwych rozwiązań problemu ośmiu hetmanów.

Metoda 1

Zastanówmy się nad siłowym rozwiązaniem problemu (ang. *brute-force*). Polega ono na przejrzeniu wszystkich możliwych ustawień hetmanów na szachownicy i sprawdzeniu, czy figury są poprawnie ustawione. Ta metoda ma jednak podstawową wadę: ogromną liczbę przypadków do rozpatrzenia - ponad 4 mld.

> **Dla zainteresowanych**
>
> Skąd taka liczba? Każde ustawienie to wybór 8 pól spośród wszystkich 64 pól szachownicy. Takich ustawień jest tyle, ile 8-elementowych kombinacji bez powtórzeń zbioru 64-elementowego. Ich liczbę można wyznaczyć z użyciem wzoru na liczbę k-elementowych kombinacji bez powtórzeń ze zbioru n-elementowego: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$,

gdzie $\binom{n}{k}$ to współczynnik dwumianowy Newtona. Po podstawieniu wartości otrzymujemy:

$$\binom{64}{8} = \frac{64!}{8!(64-8)!} = 4426165368.$$

Taka liczba przypadków do rozpatrzenia jest ogromna nawet dla bardzo szybkiego komputera. Przy założeniu, że procesor będzie wykonywał 4 000 obliczeń na sekundę, to i tak znalezienie rozwiązania zajmie ponad 278 godzin.

Metoda 2

Znacznie lepszym sposobem jest algorytm przeszukiwania z nawrotami, przedstawiony w sekcji „Film samouczek" i omówiony szczegółowo w e-materiale Problem ośmiu hetmanów.

Przykład 1

Przygotujmy program, który zrealizuje ten algorytm, a jako wynik zwróci listę (tablicę) zawierającą osiem elementów – będą to numery kolumn, w których należy ustawić hetmana w kolejnych wierszach szachownicy. Jeżeli ustawienia nie da się zrealizować, program powinien wypisać komunikat: `Nie istnieje rozwiązanie`.

Specyfikacja:

Dane:

- `N` – liczba hetmanów oraz liczba wierszy i kolumn szachownicy; liczba naturalna dodatnia

Wynik:

Program wypisuje numery kolumn, w których należy ustawić hetmana w kolejnych wierszach szachownicy lub wypisuje komunikat: `Nie istnieje rozwiązanie` – jeśli hetmanów nie da się ustawić zgodnie z warunkami zadania.

```
# Funkcja sprawdza czy pole o współrzędnych w, k nie
# jest atakowane przez innego hetmana. Zwracaną wartością jest
# jeżeli pole jest bezpieczne, bądź False w p.p.
def dopuszczalny(w, k):
    wynik = a[w] and b[w + k] and c[w - k]
    return wynik

# Procedura "ustawia" hetmana na wskazanym polu. Do tablic info
# czy pola są bezpiczne zapisywana jest informacja, o ataku.
def zapisz(w, k):
```

```
    x[k] = w # tu zapisujemy numer wiersza
    a[w] = False
    b[w + k] = False
    c[w - k] = False

# Procedura oznacza pola jako ponownie bezpieczne.
def wymaz(w, k):
    a[w] = True
    b[w + k] = True
    c[w - k] = True

# Funkcja próbuje ustawić hetmana w kolumnie k.
def probuj(k):
    udany = False
    w = 1

    while (not udany) and (w <= N):
        if dopuszczalny(w, k):
            zapisz(w, k)

            if k < N:
                udany = probuj(k + 1)

                if not udany:
                    wymaz(w, k)
            else:
                udany = True

        w += 1

    return udany

# definiujemy wartości globalne
# skorzystamy z właściwości możliwości zmiany elementów listy
# niezależnie od przestrzeni nazw w języku Python

N = 8  # bok szachownicy i jednocześnie liczba hetmanów
x = (N + 1) * [0] # x[i] to pozycja hetmana w wierszu i
a = (N + 1) * [True] # a[j] == True to brak hetmana w wierszu j
# b[k] == True to brak hetmana na przekątnej k [/].
b = (2 * (N + 1) - 1) * [True]
# Suma wiersz + kolumna od 0 do (2N - 2).
```

```
# c[k] == True to brak hetmana na przekątnej k.
c = (2 * (N + 1) - 1) * [True]
# Różnica wiersz-kolumna od (-N + 1) do (N - 1).
# pierwsze możliwe miejsce dla hetmana (0,0)
startowa_pozycja = 1

if probuj(startowa_pozycja):
    print(x[1:])
else:
    print("Nie istnieje rozwiązanie")

# przykładowe wykonanie tego proramu daje wynik:
# Mamy rozwiązanie: [1, 5, 8, 6, 3, 7, 2, 4]
```

Przykład 2

Zdefiniujmy funkcję, która korzystając z danych przygotowanych przez program z Przykładu 1, pozwoli wyświetlić prostą reprezentację wyników.

```
def rysuj(dane, n):
    for w in range(1, n + 1):
        print(f"Dla wiersza {w} kolumna {dane[w]}.")

    print("-----[przykładowa szachownica]-----")

    for w in range(1, n + 1):
        print(f"W:{w} K:{dane[w]}->", end="")

        for k in range(1, n + 1):
            if dane[w] == k:
                print(" H", end="")
            else:
                print(" .", end="")

        print("")

# wywołujemy program, podając jako parametry
# dane uzyskane z poprzedniego programu

rysuj(x, N)

# przykładowy wynik działania funkcji rysuj
```

```
# Dla wiersza 1 kolumna 1.
# Dla wiersza 2 kolumna 5.
# Dla wiersza 3 kolumna 8.
# Dla wiersza 4 kolumna 6.
# Dla wiersza 5 kolumna 3.
# Dla wiersza 6 kolumna 7.
# Dla wiersza 7 kolumna 2.
# Dla wiersza 8 kolumna 4.
# -----[przykładowa szachownica]-----
# W:1 K:1-> H . . . . . . .
# W:2 K:5-> . . . . H . . .
# W:3 K:8-> . . . . . . . H
# W:4 K:6-> . . . . . H . .
# W:5 K:3-> . . H . . . . .
# W:6 K:7-> . . . . . . H .
# W:7 K:2-> . H . . . . . .
# W:8 K:4-> . . . H . . . .
```

Dla zainteresowanych

Prezentację wyników możemy wykonać za pomocą biblioteki Pygame Zero. Pozwoli ona w łatwy sposób wyświetlić grafiki hetmanów na szachownicy.

Przykład 3

Przygotujmy kod, który – wykorzystując bibliotekę Pygame Zero – zrealizuje graficzną wizualizację.

```
TITLE = "8 hetmanów"
WIDTH = 400
HEIGHT = 400
pozycje = [1, 5, 8, 6, 3, 7, 2, 4]
hetmany = [
    Actor("queen.png"),
    Actor("queen.png"),
    Actor("queen.png"),
    Actor("queen.png"),
    Actor("queen.png"),
    Actor("queen.png"),
    Actor("queen.png"),
    Actor("queen.png")
]

```

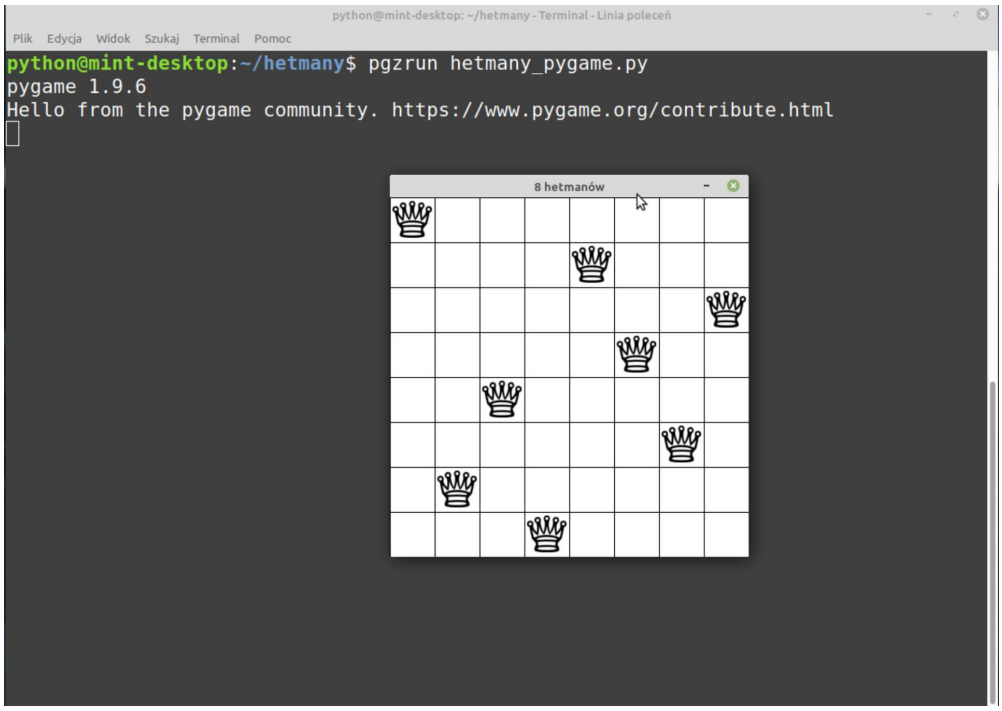
```
def draw():
    screen.fill((255, 255, 255)) # ustawiamy białe tło

    # rysujemy linie pionowe i poziome
    for x in range(0, 400, 50):
        start = (x, 0)
        koniec = (x, 400)
        screen.draw.line(start, koniec, (0, 0, 0))

    for y in range(0, 400, 50):
        start = (0, y)
        koniec = (400, y)
        screen.draw.line(start, koniec, (0, 0, 0))

    # wykreślamy hetmany na ekranie
    for w in range(1, 9):
        k = pozycje[w - 1]
        x = (k * 50) - 25
        y = (w * 50) - 25
        # ustawiamy pozycję na szachownicy
        hetmany[w - 1].pos = (x, y)
        # metodą draw() wyświetlamy dany element
        hetmany[w - 1].draw()
```

Ważne!

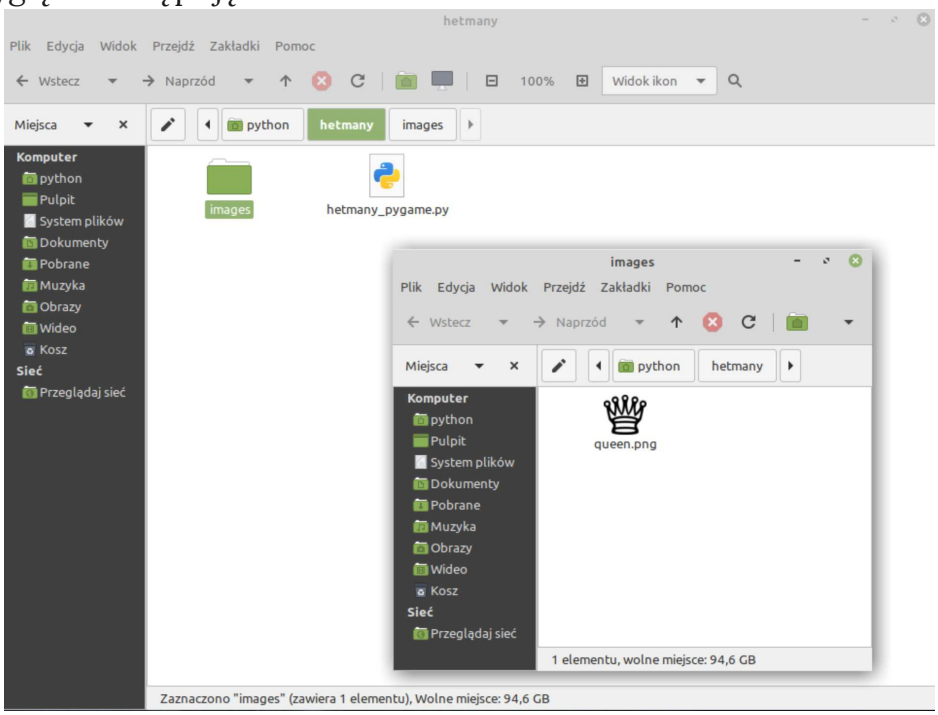
Aby używać biblioteki `Pygame Zero`, uruchomimy program zapisany w języku Python za pomocą polecenia `pgzrun program.py`, korzystając z linii poleceń systemu operacyjnego.

```
pgzrun nazwa_programu.py
```

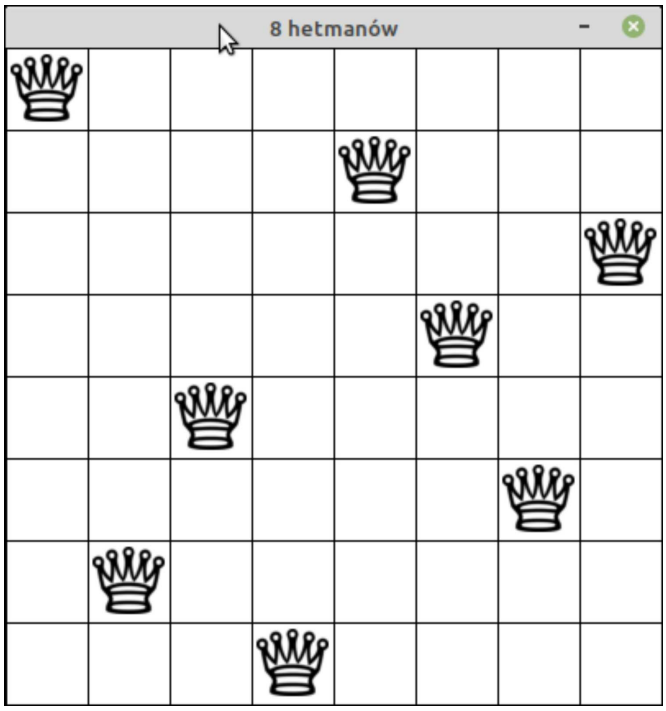
Oto przykład wykonania naszego kodu:



Biblioteka `Pygame Zero` wymaga, aby obrazki, które wykorzystujemy w naszym programie, były zapisane w postaci plików png w podkatalogu `images`, w katalogu, w którym istnieje nasz program. Użyjemy w tym celu grafiki pochodzącej z jednego z serwisów oferujących ilustracje na otwartej licencji. Musimy tylko zmienić rozdzielczość tej grafiki, aby mieściła się w polu 50x50 pikseli. Przykładowy katalog z plikami wygląda następująco:



Efekt końcowy działania programu to graficzna prezentacja możliwych ustawień hetmanów na szachownicy.



Słownik

algorytm z nawrotami

(z ang. *backtracking*) algorytm, który wyszukuje rozwiązania niektórych problemów obliczeniowych, stopniowo generując kandydatów na rozwiązanie; gdy zauważy, że kandydat nie może być poprawnym rozwiązaniem, wraca do punktu, w którym może zmienić decyzję

Pygame Zero

biblioteka służąca do szybkiego tworzenia gier w języku Python; oparta na bibliotece `Pygame`, umożliwia tworzenie programów graficznych za pomocą minimalnej ilości kodu; nie jest dostępna w standardowej instalacji języka Python – należy ją zainstalować, korzystając z polecenia `pip install pgzero`

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program wyświetlający ustawienie nieatakujących się n hetmanów na szachownicy o n wierszach i n kolumnach dla sytuacji, w której dwa hetmany mają ustalone już swoje pozycje. Jeżeli takie ustawienie nie istnieje, program powinien wypisać komunikat:
`Ustawienia nie da się zrealizować`.

Sprawdź działanie programu dla następujących ustawień dwóch hetmanów:

- `n = 8`
- `(x1, y1) = (1, 8)`
- `(x2, y2) = (8, 5)`

Specyfikacja:

Dane:

- `n` – liczba hetmanów oraz liczba wierszy i kolumn szachownicy; liczba naturalna
- `(x1, y1)` – ustawienie pierwszego hetmana; para liczb naturalnych z przedziału od 1 do *n*, gdzie `x1` oznacza numer wiersza, z kolei `y1` numer kolumny
- `(x2, y2)` – ustawienie drugiego hetmana; para liczb naturalnych z przedziału od 1 do *n*, gdzie `x2` oznacza numer wiersza, z kolei `y2` numer kolumny

Wynik:

Na standardowym wyjściu wyświetlany jest ciąg liczb naturalnych będący kodem znalezionego ustawienia hetmanów (*i*-ta cyfra oznacza numer wiersza, na którym znajduje się hetman w *i*-tej kolumnie) lub komunikat: `Ustawienia nie da się zrealizować` – gdy rozwiązanie problemu nie istnieje.

Twoje zadania

1. Program wyświetla ustawienie ośmiu hetmanów – numery wierszy oddzielone znakiem spacji lub komunikat: `Ustawienia nie da się zrealizować` – gdy rozwiązanie problemu nie istnieje.

```
n = 8
x1, y1 = 1, 8
x2, y2 = 8, 5
h = [0 for i in range(n+1)]

def hetman(wiersz):
    # Tu uzupełnij kod
    # Do wyświetlania wyniku użyj funkcji print z parametrem end=" ":
    # print(x, end=" ")
    return False

udany = hetman(1)
if not udany:
```

```

```

Ćwiczenie 2

Napisz program wyświetlający ustawienie nieatakujących się ośmiu hetmanów na szachownicy o 8 wierszach i 8 kolumnach dla sytuacji, w której dowolna liczba hetmanów ma już ustalone pozycje. Jeżeli ustawienie takie nie istnieje, program powinien wypisać komunikat: `Ustawienia nie da się zrealizować`.

Przetestuj działanie programu dla następującego zbioru ustawień hetmanów:

- `[(1, 8), (8, 5), (4, 3)]`

Specyfikacja:

Dane:

- `znane_ustawienia` – lista dwulementowych krotek liczb naturalnych z przedziału $\langle 1, 8 \rangle$, gdzie pierwsza pozycja krotki odpowiada za wskazanie numeru wiersza, z kolei druga za numer kolumny

Wynik:

Na standardowym wyjściu wyświetlany jest ciąg liczb naturalnych będący kodem znalezionego ustawienia hetmanów (*i*-ta cyfra oznacza numer wiersza, na którym znajduje się hetman w *i*-tej kolumnie) lub komunikat: `Ustawienia nie da się zrealizować` - gdy rozwiązanie problemu nie istnieje.

Twoje zadania

1. Program wyświetla ustawienie ośmiu hetmanów – numery wierszy oddzielone znakiem spacji lub komunikat: `Ustawienia nie da się zrealizować` – gdy rozwiązanie problemu nie istnieje, z ograniczeniem polegającym na tym, że na początku znane są już niektóre pozycje hetmanów.

```
1 h = [0 for i in range(9)]
2 znane_ustawienia = [(1, 8), (8, 5), (4, 3)]
3
4
5 def hetman(wiersz):
```

```
    # Tu uzupełnij kod
    # Do wyświetlania wyniku użyj funkcji print z parametrem end=" ":
    # print(x, end=" ")
    return False

udany = hetman(1)
if not udany:
    print("Ustawienia nie da się zrealizować")
```

```
1
```

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Problem ośmiu hetmanów w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

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.

Treści nauczania – wymagania szczegółowe

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

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

b) rekurencję (do generowania ciągów liczb, potęgowania, sortowania liczb, generowania fraktali),

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):

- Zaimplementujesz algorytm rozwiązujący problem ośmiu hetmanów.

- Przeanalizujesz algorytm zaimplementowany w języku Python.
- Wyjaśnisz, na czym polega algorytm z nawrotami.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- metody aktywizujące.

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. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Problem ośmiu hetmanów w języku Python". Uczniowie zapoznają się z multimedium w sekcji „Przeczytaj".

Faza wstępna:

1. Chętna lub wybrana osoba przedstawia najważniejsze informacje związane z rekurencją, iteracją oraz problemem ośmiu hetmanów.
2. Nauczyciel wyświetla uczniom temat zajęć oraz cele. Prosi, by na ich podstawie uczniowie sformułowali kryteria sukcesu.

Faza realizacyjna:

1. Uczniowie piszą program przedstawiony w Problemie 1 w sekcji „Film samouczek". Następnie chętne lub wybrane osoby prezentują swój kod. Nauczyciel omawia

programy uczniów, wskazując, gdzie mogłyby zostać poprawione lub zoptymalizowane.
2. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Wybrany uczeń podsumowuje zajęcia z programowania w Pythonie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 2 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:

- Treści w sekcji „Przeczytaj" można wykorzystać na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.