Algorytmy tekstowe

- Wprowadzenie
- Przeczytaj
- Prezentacja multimedialna
- Sprawdź się
- Dla nauczyciela

Źródło: Jessica Lewis, domena publiczna.

Algorytmy tekstowe są przydatne do przetwarzania oraz przeszukiwania danych tekstowych. Jeden z nich już poznaliśmy, to algorytm Knutha-Morrisa-Pratta.

Używamy ich, korzystając z edytora tekstu czy klienta e-mail. Znajdują również zastosowanie w grach. Możemy je wykorzystać, projektując implementację popularnej gry w wisielca.

Implementację algorytmów tekstowych w wybranych językach programowania znajdziesz w e-materiałach:

- Algorytmy tekstowe w języku C++,
- Algorytmy tekstowe w języku Java,
- Algorytmy tekstowe w języku Python.

Twoje cele

- Prześledzisz algorytm realizujący założenia gry w wisielca, zapisany za pomocą pseudokodu.
- Rozwiążesz kilka zadań sprawdzających znajomość algorytmów tekstowych.
- Przeanalizujesz sposoby rozbudowania gry w wisielca o dodatkowe funkcje.

Przeczytaj

Gra w wisielca – zasady

Przed przystąpieniem do omawiania algorytmu, który zrealizuje założenia gry w wisielca, zapoznajmy się z przebiegiem rozgrywki.

1. Gracz 1. wymyśla słowo, które drugi gracz musi odgadnąć. Gracz 2. zna tylko długość słowa. Ma także określoną z góry liczbę prób odgadnięcia wyrazu.
2. Gracz 2. podaje literę, która według niego znajduje się w zgadywanym słowie lub podejmuje próbę odgadnięcia całego słowa.

- Jeżeli gracz 2. zdecydował się podać wybraną literę, gracz 1 sprawdza, czy znajduje się ona w słowie do odgadnięcia. W sytuacji, gdy jest ona w nim obecna, gracz 1. wpisuje odgadniętą literę na wszystkich miejscach, na których występuje ona w słowie. Przykładowo, jeżeli gracz 1. wymyślił słowo „INFORMATYKA", a podana przez gracza 2. litera to „A", zgadujący zostanie poinformowany, że wymyślonym słowem jest: „`_ _ _ _ _ _ A _ _ _ A`". W przypadku, gdy gracz 2. wybrał literę, która nie znajduje się w wymyślonym słowie, traci jedną szansę odgadnięcia wyrazu („jedno życie").
- Jeżeli gracz 2. zgaduje całe słowo, gracz 1 sprawdza, czy podany przez niego wyraz jest taki sam jak słowo wymyślone przez gracza 1. Gdy tak jest, gracz 2. wygrywa. W przeciwnym przypadku gracz 2. traci jedną szansę odgadnięcia wyrazu lub litery.

Czynności opisane w punkcie drugim powtarza się do momentu, w którym gracz 2. poda słowo wymyślone przez gracza 1. lub gdy wyczerpie limit prób odgadnięcia wyrazu.

Pseudokod algorytmu

Spróbujmy napisać algorytm realizujący opisaną partię gry w wisielca za pomocą pseudokodu.

> **Ważne!**
>
> W tej wersji algorytmu nie ma możliwości odgadywania całego słowa.

Specyfikacja:

Dane:

- `zycia` — liczba szans gracza 2. na odgadnięcie litery; liczba naturalna większa od zera
- `slowo` — podane przez gracza 1. hasło, które gracz 2. będzie próbował odgadnąć; ciąg znaków

Wynik:

Program na standardowym wyjściu drukuje komunikat o powodzeniu lub niepowodzeniu gracza 2. w próbie odgadnięcia wszystkich liter hasła.

Algorytm wygląda następująco:

```
zycia ← 10
nadpisane ← ""
slowo ← wczytaj ciąg znaków

dla i = 1, 2, ..., rozmiar(slowo) wykonuj
    nadpisane ← nadpisane + "_"

wyświetl nadpisane
czyZgadniete ← fałsz

dopóki zycia > 0 wykonuj
    wyświetl "Podaj literę do sprawdzenia"
    odgadywane ← wczytaj ciąg znaków
    czyZawiera ← fałsz

    jeżeli rozmiar(odgadywane) = 1
        dla i = 1, 2, ..., rozmiar(slowo) wykonuj
            jeżeli slowo[i] = odgadywane[1]
                czyZawiera ← prawda
                nadpisane[i] ← odgadywane[1]

        jeżeli czyZawiera = fałsz
            zycia ← zycia - 1
            wyświetl "Słowo nie zawiera danej litery"

        czyZgadniete ← prawda

        dla i = 1, 2, ..., rozmiar(nadpisane) wykonuj
            jeżeli nadpisane[i] = '_'
                czyZgadniete ← fałsz
                przerwij pętlę

        jeżeli czyZgadniete = prawda
            przerwij pętlę
    w przeciwnym razie
```

```
        wyświetl "Podana została więcej niż jedną litera"

    wyświetl nadpisane
    wyświetl "Pozostałe życia " + zycia

jeżeli czyZgadniete = fałsz
    wyświetl "Niestety nie udało ci się odgadnąć słowa"
w przeciwnym razie
    wyświetl "Udało ci się odgadnąć słowo!"
```

> **Ważne!**
>
> W algorytmie wykorzystujemy funkcję `rozmiar()` zwracającą długość zadanego ciągu znaków.

Pseudokod jest dosyć obszerny. Omówmy go teraz linijka po linijce.

Najpierw deklarowane są dwie zmienne: `zycia` oraz `nadpisane`. Pierwsza przyjmuje początkowo wartość 10 i będzie służyć do przechowywania informacji o pozostającej graczowi 2. liczbie prób zgadnięcia litery. Zmienna `nadpisane` przechowywać będzie słowo z zaznaczonymi literami, które udało się odgadnąć.

Następnie pobieramy od użytkownika słowo, które gracz 2. będzie próbował odgadnąć. Zapisujemy je jako zmienną `slowo`.

```
zycia ← 10
nadpisane ← ""
slowo ← wczytaj ciąg znaków
```

Kolejnym etapem jest wypełnienie zmiennej `nadpisane` znakami podkreślenia (liczba znaków jest równa liczbie liter składających się na wymyślone słowo). Następnie zmienna ta jest wyświetlana na ekranie, dzięki czemu gracz 2. dowie się, jaką długość ma słowo wymyślone przez gracza 1. Inicjowana jest także zmienna `czyZgadniete`, która określa, czy gracz 2. odgadł słowo czy nie.

```
dla i = 1, 2, ..., rozmiar(slowo) wykonuj
    nadpisane ← nadpisane + "_"

wyświetl nadpisane
czyZgadniete ← fałsz
```

Tworzymy pętlę, w której zapiszemy główną część algorytmu. Warunkiem działania pętli jest to, aby gracz 2. miał do dyspozycji większą od zera liczbę prób odgadnięcia wyrazu (`zycia > 0`). Wewnątrz pętli wyświetlamy komunikat dla gracza 2. oraz pobieramy od użytkownika literę i zapisujemy ją w zmiennej `odgadywane`.

```
dopóki zycia > 0 wykonuj
    wyświetl "Podaj literę do sprawdzenia"
    odgadywane ← wczytaj ciąg znaków
```

Kolejna pętla służy do sprawdzenia, czy litera podana przez użytkownika pojawia się w wymyślonym przez gracza 1. słowie. Wynik (`prawda` lub `fałsz`) zapisywany jest w zmiennej `czyZawiera`.

Należy poza tym sprawdzić, czy użytkownik na pewno podał tylko jedną literę. W przypadku gdy gracz wpisze ciąg znaków, zostanie wyświetlony odpowiedni komunikat.

```
czyZawiera ← fałsz

jeżeli rozmiar(odgadywane) = 1
    dla i = 1, 2, ..., rozmiar(slowo) wykonuj
        jeżeli slowo[i] = odgadywane[1]
            czyZawiera ← prawda

w przeciwnym razie
    wyświetl "Podana została więcej niż jedną litera"
```

Jeżeli słowo zawiera podaną przez gracza literę, nadpisywana jest zmienna `nadpisane` – znaki podkreślenia (`_`) są w odpowiednich miejscach zastępowane odgadniętą literą.

W przypadku gdy słowo nie zawiera podanej litery, gracz traci jedną próbę odgadnięcia („jedno życie”) i wyświetlany jest odpowiedni komunikat.

```
jeżeli rozmiar(odgadywane) = 1
    dla i = 1, 2, ..., rozmiar(slowo) wykonuj
        jeżeli slowo[i] = odgadywane[1]
            czyZawiera ← prawda
            nadpisane[i] ← odgadywane[1]

    jeżeli czyZawiera = fałsz
        zycia ← zycia - 1
```

```
        wyświetl "Słowo nie zawiera danej litery"

w przeciwnym razie
    wyświetl "Podana została więcej niż jedną litera"
```

Kolejna pętla pozwala sprawdzić, czy całe słowo zostało odgadnięte czy też zostały jeszcze znaki podkreślenia. Jeśli znajdują się one wciąż w wyrazie, to zmienna logiczna `czyZgadniete` przyjmuje wartość `fałsz`, a działanie pętli jest przerywane.

Następnie pojawia się instrukcja warunkowa. Jeżeli zmienna logiczna `czyZgadniete` ma wartość `prawda`, czyli gdy wszystkie litery składające się na słowo zostały odgadnięte, następuje przerwanie pętli zewnętrznej.

```
dopóki zycia > 0 wykonuj
    wyświetl "Podaj literę do sprawdzenia"
    odgadywane ← wczytaj ciąg znaków
    czyZawiera ← fałsz

    jeżeli rozmiar(odgadywane) = 1
        dla i = 1, 2, ..., rozmiar(slowo) wykonuj
            jeżeli slowo[i] = odgadywane[1]
                czyZawiera ← prawda
                nadpisane[i] ← odgadywane[1]

        jeżeli czyZawiera = fałsz
            zycia ← zycia - 1
            wyświetl "Słowo nie zawiera danej litery"

        czyZgadniete ← prawda

        dla i = 1, 2, ..., rozmiar(nadpisane) wykonuj
            jeżeli nadpisane[i] = '_'
                czyZgadniete ← fałsz
                przerwij pętlę

        jeżeli czyZgadniete = prawda
            przerwij pętlę
    w przeciwnym razie
        wyświetl "Podana została więcej niż jedną litera"
```

Każde próba zgadnięcia litery powinna zakończyć się wyświetleniem słowa z widocznymi, odgadniętymi literami oraz aktualnej liczby dostępnych szans.

```
dopóki zycia > 0 wykonuj
    wyświetl "Podaj literę do sprawdzenia"
    odgadywane ← wczytaj ciąg znaków
    czyZawiera ← fałsz

    jeżeli rozmiar(odgadywane) = 1
        dla i = 1, 2, ..., rozmiar(slowo) wykonuj
            jeżeli slowo[i] = odgadywane[1]
                czyZawiera ← prawda
                nadpisane[i] ← odgadywane[1]

        jeżeli czyZawiera = fałsz
            zycia ← zycia - 1
            wyświetl "Słowo nie zawiera danej litery"

        czyZgadniete ← prawda

        dla i = 1, 2, ..., rozmiar(nadpisane) wykonuj
            jeżeli nadpisane[i] = '_'
                czyZgadniete ← fałsz
                przerwij pętlę

        jeżeli czyZgadniete = prawda
            przerwij pętlę
    w przeciwnym razie
        wyświetl "Podana została więcej niż jedną litera"

    wyświetl nadpisane
    wyświetl "Pozostałe życia " + zycia
```

Ostatnim etapem jest wyświetlenie komunikatu końcowego:

```
jeżeli czyZgadniete = fałsz
    wyświetl "Niestety nie udało ci się odgadnąć słowa"
w przeciwnym razie
    wyświetl "Udało ci się odgadnąć słowo!"
```

Słownik

algorytm

przepis postępowania prowadzący do rozwiązania ustalonego problemu, określający ciąg czynności elementarnych, które należy w tym celu wykonać

pseudokod

sposób zapisu algorytmu; zachowuje strukturę charakterystyczną dla kodu zapisanego w języku programowania (wcięcia, bloki, instrukcje), jednak rezygnuje ze ścisłych reguł składniowych na rzecz prostoty i czytelności

Prezentacja multimedialna

Polecenie 1

Zmodyfikuj pseudokod w ten sposób, aby gracz miał możliwość odgadnięcia nie tylko pojedynczych liter, ale także całego słowa.

```
zycia ← 10
nadpisane ← ""
slowo ← wczytaj ciąg znaków

dla i = 1, 2, ..., rozmiar(slowo) wykonuj
    nadpisane ← nadpisane + "_"

wyświetl nadpisane
czyZgadniete ← fałsz

dopóki zycia > 0 wykonuj
    wyświetl "Podaj literę do sprawdzenia"
    odgadywane ← wczytaj ciąg znaków
    czyZawiera ← fałsz
```

Polecenie 2

Porównaj swoje rozwiązanie z prezentacją.

1

Z algorytmów tekstowych korzystamy codziennie. Przykładem wykorzystania algorytmów tekstowych mogą być filtry wykorzystywane przez skrzynkę e-mail. Na podstawie treści wiadomości szacują, czy przychodząca poczta to tzw. *spam*, czyli niechciana korespondencja.
Źródło: Stephan, Pixabay, pixabay.com, CC 0.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Zmodyfikujemy algorytm napisany w sekcji „Przeczytaj". Dodamy do niego instrukcje, dzięki którym gracz będzie miał możliwość odgadnięcia nie tylko pojedynczych liter, ale także całego słowa.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Zastanówmy się, w którym miejscu napisanego wcześniej pseudokodu powinniśmy umieścić sekcję odpowiedzialną za możliwość podjęcia próby odgadnięcia całego słowa.

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Po przeanalizowaniu algorytmu zauważymy, że należy to zrobić w miejscu, w którym sprawdzaliśmy, czy użytkownik podał tylko jedną literę i otrzymaliśmy odpowiedź przeczącą.

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Należy sprawdzić, czy wprowadzone przez gracza słowo jest takie samo jak wyraz, który należy odgadnąć.

```
jeżeli odgadywane = slowo
```

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

W przypadku gdy gracz podał poprawne słowo, zmienna logiczna `czyZgadniete` przyjmuje wartość `prawda` oraz następuje przerwanie działania pętli. Dalsze zgadywanie nie ma już w tym przypadku sensu.

```
jeżeli odgadywane = slowo
    czyZgadniete ← prawda
    przerwij pętlę
```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Musimy wziąć też pod uwagę sytuację, w której gracz poda niepoprawne słowo.

```
w przeciwnym razie
    jeżeli odgadywane = slowo
        czyZgadniete ← prawda
        przerwij pętlę
    w przeciwnym razie
```

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Jeżeli gracz poda niewłaściwe hasło, należy zmniejszyć liczbę prób pozostałych do odgadnięcia słowa.

```
w przeciwnym razie
    jeżeli odgadywane = slowo
        czyZgadniete ← prawda
        przerwij pętlę
    w przeciwnym razie
        zycia ← zycia - 1
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Możemy także poinformować gracza, że podane przez niego słowo nie jest odgadywanym hasłem.

```
w przeciwnym razie
    jeżeli odgadywane = slowo
        czyZgadniete ← prawda
        przerwij pętlę
    w przeciwnym razie
        zycia ← zycia - 1
        wyświetl "Podane słowo nie jest odgadywanym hasłem"
```

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Pseudokod algorytmu uwzględnia już możliwość odgadnięcia całego słowa. Zastanów się, co jeszcze można dodać do naszej implementacji gry w wisielca i samodzielnie spróbuj rozbudować ją o nowe funkcje.

10

Gdy jesteś na wakacjach i wykorzystujesz aplikacje, których zadaniem jest tłumaczenie

tekstów, również korzystasz z algorytmów tekstowych.
Źródło: Pixabay, pixabay.com, CC 0.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PVFv963UE

Oto cały pseudokod:

```
zycia ← 10
nadpisane ← ""
slowo ← wczytaj ciąg znaków

dla i = 1, 2, ..., rozmiar(slowo) wykonuj
    nadpisane ← nadpisane + "_"

wyświetl nadpisane
czyZgadniete ← fałsz

dopóki zycia > 0 wykonuj
    wyświetl "Podaj literę lub słowo do sprawdzenia"
    odgadywane ← wczytaj ciąg znaków
    czyZawiera ← fałsz

    jeżeli rozmiar(odgadywane) = 1
        dla i = 1, 2, ..., rozmiar(slowo) wykonuj
            jeżeli slowo[i] = odgadywane[1]
                czyZawiera ← prawda

nadpisane[i] ← odgadywane[1]

        jeżeli czyZawiera = fałsz
            zycia ← zycia - 1
```

```
24                wyświetl "Słowo nie zawiera danej litery"
25
26            czyZgadniete ← prawda
27
28            dla i = 1, 2, ..., rozmiar(nadpisane) wykonuj
29                jeżeli nadpisane[i] = '_'
30                    czyZgadniete ← fałsz
31                    przerwij pętlę
32
33            jeżeli czyZgadniete = prawda
34                przerwij pętlę
35
36        w przeciwnym razie
37            jeżeli odgadywane = slowo
38                czyZgadniete ← prawda
39                przerwij pętlę
40            w przeciwnym razie
41                zycia = zycia - 1
42                wyświetl "Podane słowo nie jest odgadywanym hasłem"
43
44        wyświetl nadpisane
45        wyświetl "Pozostałe życia " + zycia
46
47    jeżeli czyZgadniete = fałsz
48        wyświetl "Niestety nie udało ci się odgadnąć słowa"
49    w przeciwnym razie
50        wyświetl "Udało ci się odgadnąć słowo!"
```

Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Polecenie 3

Podaj przykłady algorytmów tekstowych, z jakich korzystasz w życiu codziennym.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wskaż, jaką wartość zwróci funkcja `rozmiar(a)`, jeżeli zmienna a przechowuje ciąg znaków „`edukacyjny`".

- ◯ 12
- ◯ a
- ◯ 9
- ◯ 10

Ćwiczenie 2

Wskaż, które algorytmy są algorytmami tekstowymi. Zaznacz wszystkie poprawne odpowiedzi.

- ☐ algorytm Knutha-Morrisa-Pratta
- ☐ algorytm sprawdzający warunki palindromu
- ☐ algorytm Karpa-Rabina
- ☐ algorytm obliczania silni

Ćwiczenie 3

Wskaż, które słowo zostanie wyświetlone po wykonaniu algorytmu. Przyjmij, że funkcja `ascii()` zwraca wartość kodu ASCII danego znaku.

```
slowo ← "HDNSBASH"

dla i = 1, 2, ..., rozmiar(slowo) - 1 wykonuj
        jeżeli ascii(slowo[i]) > ascii(slowo[i + 1])
                zamień(slowo[i], slowo[i + 1])

wyświetl slowo
```

- ◯ DBHNSSHA
- ◯ ABDHHNSS
- ◯ SSNHHDBA
- ◯ DHNBASHS

Ćwiczenie 4

Wskaż, które słowo zostanie wyświetlone po wykonaniu poniższego algorytmu. Przyjmij, że funkcja `ascii()` zwraca kod ASCII podanego znaku.

```
slowo ← "HDNSBASH"

dla i = 1, 2, ..., rozmiar(slowo) - 1 wykonuj
        dla j = 1, 2, ..., rozmiar(slowo) - 1 wykonuj
                jeżeli ascii(slowo[j]) > ascii(slowo[j + 1])
                        zamień(slowo[j], slowo[j + 1])

wyświetl slowo
```

- SNSABDHH
- BNSSADHH
- ABDHHNSS
- SSNNHDBA

Ćwiczenie 5

Wskaż, które słowo zostanie wyświetlone po wykonaniu algorytmu. Przyjmij, że funkcja `ascii()` zwraca kod ASCII podanego znaku.

```
slowo ← "JDHSBMEIFHRH"

dla i = 1, 2, ..., rozmiar(slowo) wykonuj
        jeżeli ascii(slowo[i]) = 72
                slowo[i] ← ""

wyświetl slowo
```

- ◯ JDHSBEIFHRH
- ◯ DHSBMEIFHRH
- ◯ JDSBMEIFR
- ◯ JDHBMEIFHRH
- ◯ JHSBMEIFHRH

Ćwiczenie 6

Wskaż, które słowo zostanie wyświetlone po wykonaniu poniższego algorytmu. Przyjmij, że funkcja `ascii()` zwraca kod ASCII podanego znaku.

```
slowo ← "KROKODYL"

dla i = 1, 2, ..., rozmiar(slowo) wykonuj
        jeżeli ascii(slowo[i]) = 75
                slowo[i] ← "G"

wyświetl slowo
```

- ◯ KROKODGL
- ◯ KGOKODYL
- ◯ KROKODYG
- ◯ GROGODYL

Ćwiczenie 7

Połącz w pary wprowadzone wartości zmiennych z odpowiadającymi im wynikami działania algorytmu.

```
slowo ← wprowadź ciąg znaków
dla i = 1, 2, ..., rozmiar(slowo) wykonuj
        jeżeli i mod 3 = 0
                slowo[i] ← slowo[rozmiar(slowo)]
                slowo[i - 1] ← slowo[1]

wyświetl slowo
```

`slowo ← "kanarek"`	wynik: kkeake
`slowo ← "armata"`	wynik: kkkakkk
`slowo ← "korale"`	wynik: aaaaaa
`slowo ← "nowoczesny"`	wynik: nnyonyenyy

Ćwiczenie 8

Dla podanych wartości zmiennych wprowadź wyniki działania algorytmu (zwracany ciąg znaków). Przyjmij, że funkcja `ascii( )` zwraca kod ASCII podanego znaku.

```
slowo ← wprowadź ciąg znaków
dla i = 1, 2, ..., rozmiar(slowo) - 1 wykonuj
        jeżeli slowo[i] = slowo[i+1]
                jeżeli ascii(slowo[i]) = 65
                        slowo[i] = "B"
                w przeciwnym razie
                        slowo[i] = "A"
        w przeciwnym razie
                slowo[i] = "C"

wyświetl slowo
```

Wynik dla zmiennej `slowo ← "GNRTLP"`: _____

Wynik dla zmiennej `slowo ← "AAAA"`: _____

Wynik dla zmiennej `slowo ← "PLAACCW"`: _____

Wynik dla zmiennej `slowo ← "IIWOABAA"`: _____

Wynik dla zmiennej `slowo ← "LLLUUAAA"`: _____

Dla nauczyciela

Autor: zespół autorski Contentplus.pl sp. z o.o.
Przedmiot: Informatyka

Temat: Algorytmy tekstowe

Grupa docelowa:

Liceum ogólnokształcące i technikum, 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. Rozumienie, analizowanie i rozwiązywanie problemów.

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

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

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

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

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

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):

- Prześledzisz algorytm realizujący założenia gry w wisielca, zapisany za pomocą pseudokodu.
- Rozwiążesz kilka zadań sprawdzających znajomość algorytmów tekstowych.
- Przeanalizujesz sposoby rozbudowania gry w wisielca o dodatkowe funkcje.

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.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Algorytmy tekstowe”. Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj”.

Faza wstępna:

1. Nauczyciel wyświetla uczniom temat zajęć oraz cele. Prosi, by na ich podstawie uczniowie sformułowali kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytanie dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, opartego o programowanie.

Faza realizacyjna:

1. Uczniowie analizują przykład z sekcji „Przeczytaj" i powtarzają zaprezentowane rozwiązanie na swoim komputerze.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna", wybrany uczeń czyta treść polecenia nr 1: „Zmodyfikuj pseudokod w ten sposób, aby gracz miał możliwość odgadnięcia nie tylko pojedynczych liter, ale także całego słowa" i omawia przykładowe rozwiązanie postawionego problemu. Następnie uczniowie analizują rozwiązanie zawarte w prezentacji.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują pierwsze ćwiczenia interaktywne z sekcji „Sprawdź się". Wyniki pracy omawiane są na forum i komentowane przez nauczyciela.
4. Nauczyciel dzieli uczniów na grupy czteroosobowe. Uczniowie wykonują ćwiczenia nr 2-6 z sekcji „Sprawdź się", a następnie każda grupa wyznacza jedną osobę, którą wymienia się z inną grupą. W nowych zespołach uczniowie porównują swoje rozwiązania i wybierają najbardziej efektywne.

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, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenia 7-8 z sekcji „Sprawdź się".

Wskazówki metodyczne:

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