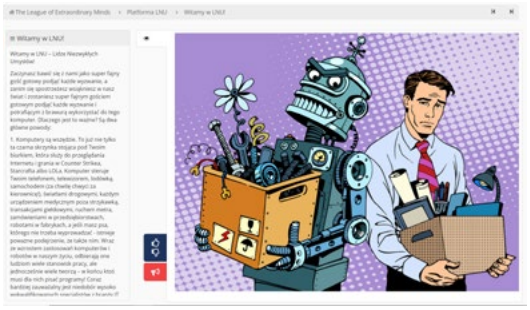
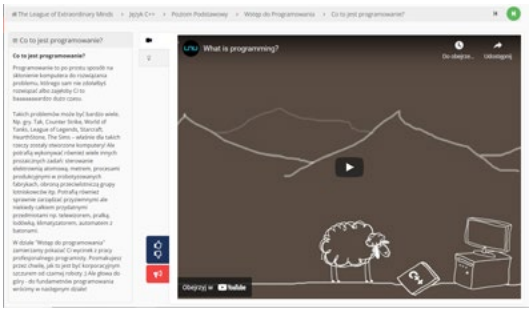
Instrukcja przygotowania misji programistycznej na platformę LNU.

Typy zadań / lekcji

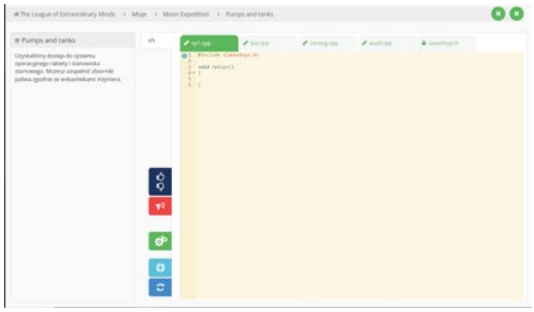
Lekcja z grafiką



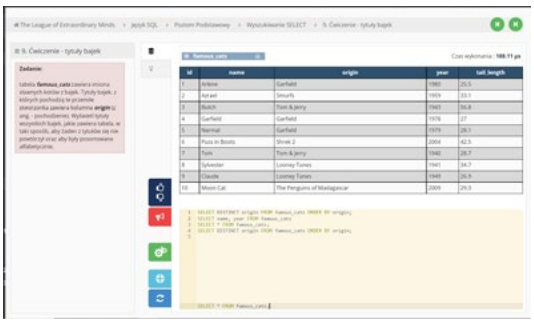
Lekcja z filmem (youtube)



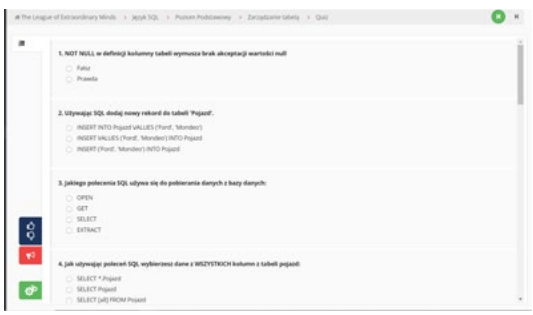
Lekcja z plikiem źródłowym (opcjonalnie wieloma plikami) / C++, Python, HTML /CSS



Lekcja z bazą danych SQL



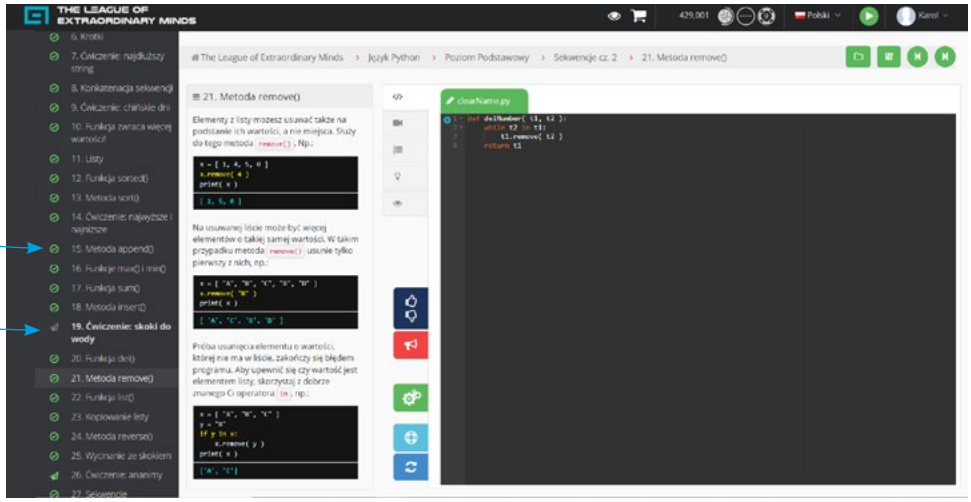
QUIZ



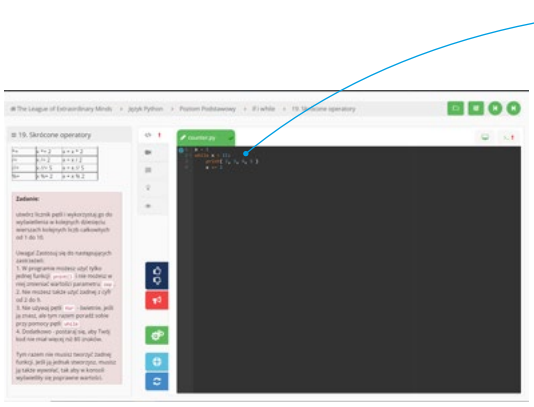
Typy zadań / lekcji w menu

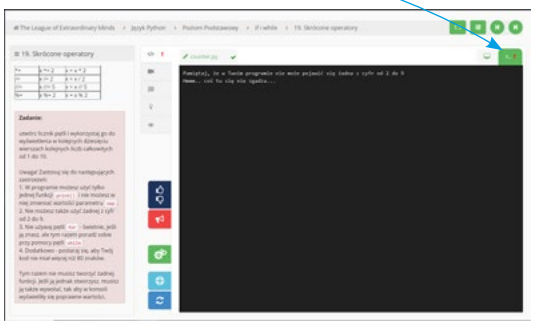
Zadanie obowiązkowe:
trzeba je rozwiązać, aby przejść do następnego

Zadanie opcjonalne:
można je pominąć i przejść

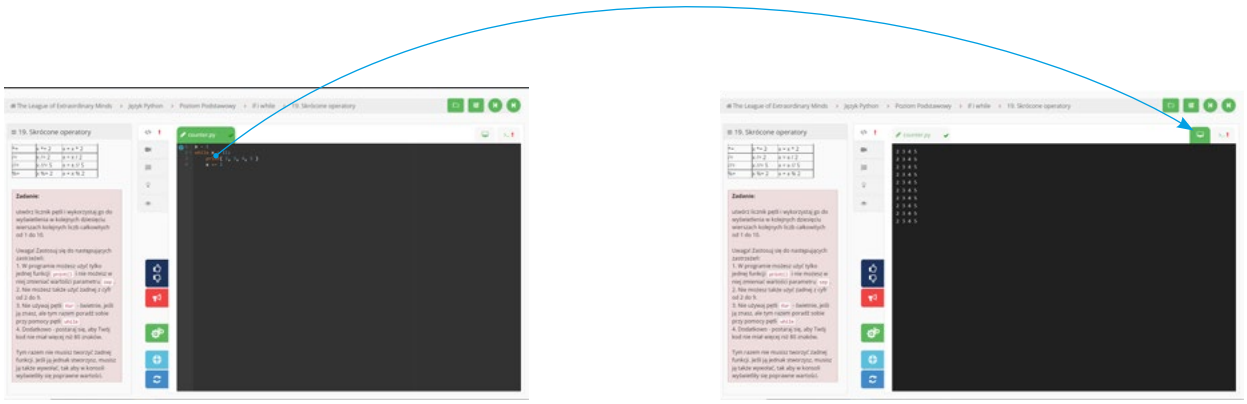


Typy dodatkowych zakładek otwieranych po uruchomieniu zadania

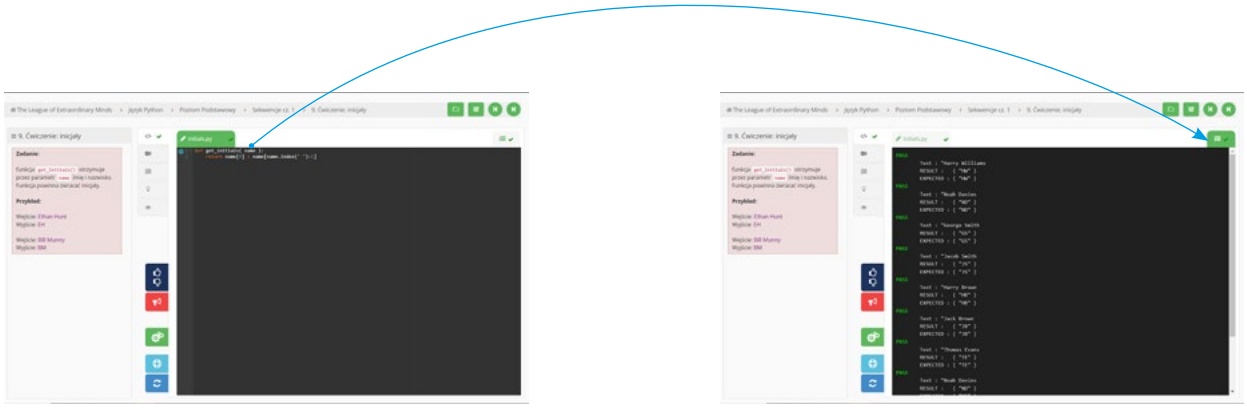




Zakładka z opisem błędów:
- jeżeli kod użytkownika nie spełnia określonych warunków
- jeżeli kod nie daje się poprawnie skomilować



Zakładka z danymi wyjściowymi programu, jeżeli program zawiera instrukcje wyjścia, np. print(), std::cout itp.



Zakładka z wynikami testów, jeżeli w zadaniu ustawione są testy, które mają sprawdzać wielokrotnie funkcję użytkownika dla różnych argumentów

Przygotowanie materiałów do zadań

Wszystkie materiały do misji przygotywane są poza platformą LNU, korzystając z dowolnych narzędzi: edytorów tekstu, programów graficznych, środowisk programistycznych itd.

Komplet materiałów należy przygotować tworząc nadrzędny folder dla całej misji, oraz podrzędne foldery zawierające materiały do poszczególnych lekcji / zadań w misji.

Treść zadania

Na treść zadania mogą składać się: tytuł zadania, tekst, grafiki, linki do dodatkowych materiałów oraz układ całości (ułożenie poszczególnych elementów w ramce treści) oraz formatowanie (użycie dodatkowych ramek, pogrubień itp.).

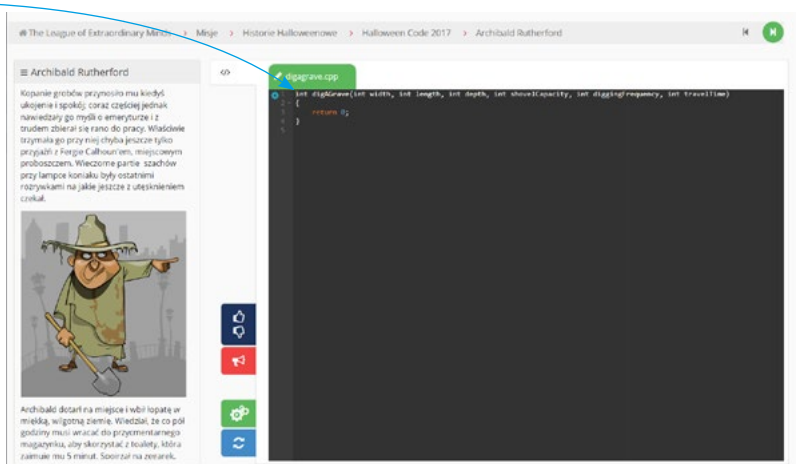
Materiały powinny być przygotowane w następującej formie:
1. Poglądowy PDF z całym układem treści i formatowaniem.
2. Osobno sam tekst w postaci pliku tekstoweg bez formatowania (plik .txt).
3. Osobno pliki z grafikami.
4. Osbono pliki z dodatkowymi materiałami.

Pliki lekcji

1. Plik edytora kodu (lub opcojnalne wiele plików) - nie dotyczy zadań z bazą danych oraz quizów
2. Pliki dodatkowe (np. pliki tekstowe, pliki API, pliki z bazami danych dla zadań SQL).
3. Plik z funkcją testującą
4. Plik definicji lekcji

Plik edytora kodu:

plik .py lub .cpp. Plik powinien zawierać początkową wersję kodu źródłowego, który użytkownik zobaczy po pierwszym otworzeniu zadania. Może to być np. początkowa wersja funkcji lub tylko komentarz.



Plik z funkcją testującą

plik .py (preferowany) lub opcjonalnie .cpp. Plik powinien zawierać Waszą funkcję test(), która będzie ukryta przed użytkownikiem i będzie uruchamiana po wciśnięciu przycisku „start” przez użytkownika. Funkcja ma dwa główne zadania:
1. wytworzyć komunikat wyświetlany użytkownikowi po uruchomieniu rozwiązania w zakładce z testami
2. zaliczyć lub nie zaliczyć lekcji

Ad.1. Wytworzenie komunikatu polega na utworzeniu zmiennej **komunikat** i przypisaniu jej wartości w postaci ciągu znaków (string).
Ad.2. Lekcja jest zaliczona jeżeli funkcja testująca zwróci wartość True.

Funkcja testująca zwykle wywołuje funkcję (lub funkcje) użytkownika, porównuje zwracane przez nią wyniki z oczekiwanymi i zwraca wartość True jeżeli wyniki są takie same. Nie jest to natomiast reguła, której trzeba się bezwzględnie trzymać :)

W funkcji testującej można także użyć standardowych testów LNU. W takiej sytuacji należy w niej jedynie utworzyć dwie zmienne:
oczekiwana - przypisać do niej wartość oczekiwaną, która będzie porównywana do wartości zwracanej przez funkcję użytkwonika
argumenty - przypisać do niej krotkę, zawierającą argumenty, które mają być użyte w wywołaniu funkcji użytkownika.
Korzystając ze standardowych testów LNU nie trzeba (choć można) tworzyć zmiennej **komunikat** - testy zostaną zaprezentowane w zakładce z testami w standardowy sposób.

Plik definicji lekcji

plik .txt

W pliku definicji lekcji należy zdefiniować następujące wartości:

Pełne nazwy wszystkich plików edytora kodu oraz plików dodatkowych. Dla każdego z plików należy określić parametry:
1. widoczny: TAK/NIE - parametr określa, czy plik będzie widoczny w edytrze dla użytkownika
2. do edycji: TAK/NIE - parametr określa, czy plik będzie edytowalny przez użytkwonika. Dla wartości NIE użytkownik będzie mógł zobaczyć zawartość pliku, ale nie będzie mógł jej edytować

Dodoatkowo należy określić parametry lekcji:
1. output: TAK/NIE - parametr określa, czy po uruchomieniu lekcji ma pojawić się zakłada konsoli z danym wyjściowymi programu (zakładka z ikoną monitora)
2. testy: TAK/NIE - parametr określa, czy po uruchomieniu lekcji ma pojawić się zakładka z testami, w której zostaną wyświetlone wyniki działania funkcji testującej - wartość zmiennej **komunikat** lub wyniki standardowych testów LNU.
Dla wartości TAK należy określić kolejny parametr:
2.1 liczba_testów: wartość - parametr określa ile razy (wartość) ma zostać wywołana funkcja testująca.
W przypadku gdy określimi liczbę testów na większą od 1 można opcjnalnie określić:
2.2. Dla każdego kolejnego testu pewne wartości, z których chcemy skrzystać w danym wywołaniu funkcji testującej. Wartości należy podać w postaci listy, zawierającej od 1 do wielu wartości dla każdego testu.
2.3 min_zaliczonych: wartość - minimalna liczba (wartość) zaliczonych testów, aby lekcja została uznana za zaliczoną.

Statyczna analiza kodu:

Jeżeli chcesz poddać kod użytkownika statycznej analizie, możesz określić następujące warunki:

cały_kod(fragment_kodu) == True/False - sprawdza występowanie lub brak (do wyboru) podanego fragmentu kodu w kodzie użytkownika, łącznie z komentarzami

kod_bez_komentarzy(fragment_kodu) == True/False - sprawdza występowanie lub brak (do wyboru) podanego fragmentu kodu w kodzie użytkownika, bez uwzględniania tego co jest w kodzie zawarte w komentarzach

czysty_kod(fragment_kodu_bez_białych_znaków) = True/False - sprawdza występowanie lub brak (do wyboru) podanego fragmentu kodu z usuniętymi białymi znakami (spacje, znaki przejścia do nowej linii) w kodzie użytkownika

wszystkie_słowa(słowo) == != < <= > >= wartość - porównuje liczbę wystąpień w kodzie pewnego słowa z zadaną liczbą.

słowa_bez_komentarzy(słowo) == != < <= > >= wartość - porównuje liczbę wystąpień w kodzie pewnego słowa z zadaną liczbą, bez uwzględniania wystąpień słowa zawartych w komentarzach.

output(string) - sprawdza czy na wyjściu programu użytkownika jest wskazany string

Dla każdego z wymienionych warunków osobno, należy określić:

komunikat_if_false(string) - komunikat, który ma się wyświetlić użytkownikowi po uruchomieniu zadania w zakładce z błędami, jeżeli dany warunkem nie jest spełniony.

Aby lekcja została zaliczona muszą być spełnione wszystkie zdefiniowane warunki.

Przykłady

pole_prostokata.txt

Pole prostokąta

Uzupełnij funkcje pole() w taki sposób, aby zwracała pole prostkąta o długościach boków przekazanych przez parametry x i y.

plik z treścią zadania

pole.py

```python
def pole( x, y ):
    return 0
```

plik edytora kodu

test.py

```python
from pole.py import pole
from random import randint

def test():

    a = randint( 1, 10 )
    b = randint( 1, 10 )

    try:
        wartosc = pole( a, b )
        oczekiwana = a * b

        komunikat = "pole( " + str(a) + ", " + str(b) + " ) = " + str(wartosc)

        if wartosc > oczekiwana:
            komunikat += " - uuu, coś za dużo!"
        elif wartosc < oczekiwana:
            komunikat += " - uuu, coś za mało!"
        else:
            komunikat += "- super!"

        return wartosc == oczekiwana


    except:
        komunikat = "Coś poszło nie tak..."
        return False

test()
```

plik z funkcją testującą

definicja_lekcji.txt

pole.py
widoczny: TAK
do edycji: TAK

output: TAK
testy: TAK
liczba testów: 5
min_zaliczonych: 5

plik definicji lekcji

pole.py

```python
def pole( x, y ):
    return x + y
```

plik edytora kodu po edycji przez użytkownika

wyniki wyświetlane po uruchomieniu w zakładce z testami

```
FAILED:         pole( 8, 10 ) = 18 - uuu, coś za mało!
FAILED:         pole( 8, 5 ) = 13 - uuu, coś za mało!
FAILED:         pole( 9, 1 ) = 10 - uuu, coś za dużo!
FAILED:         pole( 7, 3 ) = 10 - uuu, coś za mało!
FAILED:         pole( 7, 1 ) = 8 - uuu, coś za dużo!
```

Poprzedni przykład z wykorzystaniem standardowych testów LNU

pole_prostokata.txt

Pole prostokąta

Uzupełnij funkcje pole() w taki sposób, aby zwracała pole prostkąta o długościach boków przekazanych przez parametry x i y.

plik z treścią zadania

pole.py

```python
def pole( x, y ):
    return 0
```

plik edytora kodu

test.py

```python
from pole.py import pole
from random import randint

def test():
    a = randint( 1, 10 )
    b = randint( 1, 10 )
    argumenty = ( a, b )
    oczekiwana = a * b


test()
```

plik z funkcją testującą

definicja_lekcji.txt

pole.py
widoczny: TAK
do edycji: TAK

output: TAK
testy: TAK
liczba testów: 5
min_zaliczonych: 5

plik definicji lekcji

pole.py

```python
def pole( x, y ):
    return x + y
```

plik edytora kodu po edycji przez użytkownika

wyniki wyświetlane po uruchomieniu w zakładce z testami

```
FAILED:         pole( 8, 10 ):
                    Result: 18
                Expected: 80
FAILED:         pole( 8, 5 ):
                    Result: 13
                Expected: 40
FAILED:         pole( 9, 1 ) :
                    Result: 10
                Expected: 9
FAILED:         pole( 7, 3 ):
                    Result: 10
                Expected: 21
FAILED:         pole( 7, 1 ):
                    Result: 8
                Expected: 7
```

Przykład z wykorzystaniem danych dla poszczególnych testów zdefi-

pole_prostokata.txt

Pole prostokąta

Uzupełnij funkcje pole() w taki sposób, aby zwracała pole prostkąta o długościach boków przekazanych przez parametry x i y.

plik z treścią zadania

pole.py

```
1 def pole( x, y ):
2     return 0
```

plik edytora kodu

test.py

```
1  from pole.py import pole
2  from random import randint
3
4  def test():
5      a = t[0] #dane pobierane z pliku definicji lekcji dla każdego testu osobno
6      b = t[1] #dane pobierane z pliku definicji lekcji dla każdego testu osobno
7      argumenty = ( a, b )
8      oczekiwana = a * b
9
10
11 test()
```

plik z funkcją testującą

definicja_lekcji.txt

pole.py
widoczny: TAK
do edycji: TAK

output: TAK
testy: TAK
liczba testów: 5
min_zaliczonych: 5
dane do testów:
[2, 5]
[3, 7]
[2, 2]
[5, 4]
[9, 2]

Liczba testów ustalona na 5 zatem 5 list z danymi dla poszczgól-

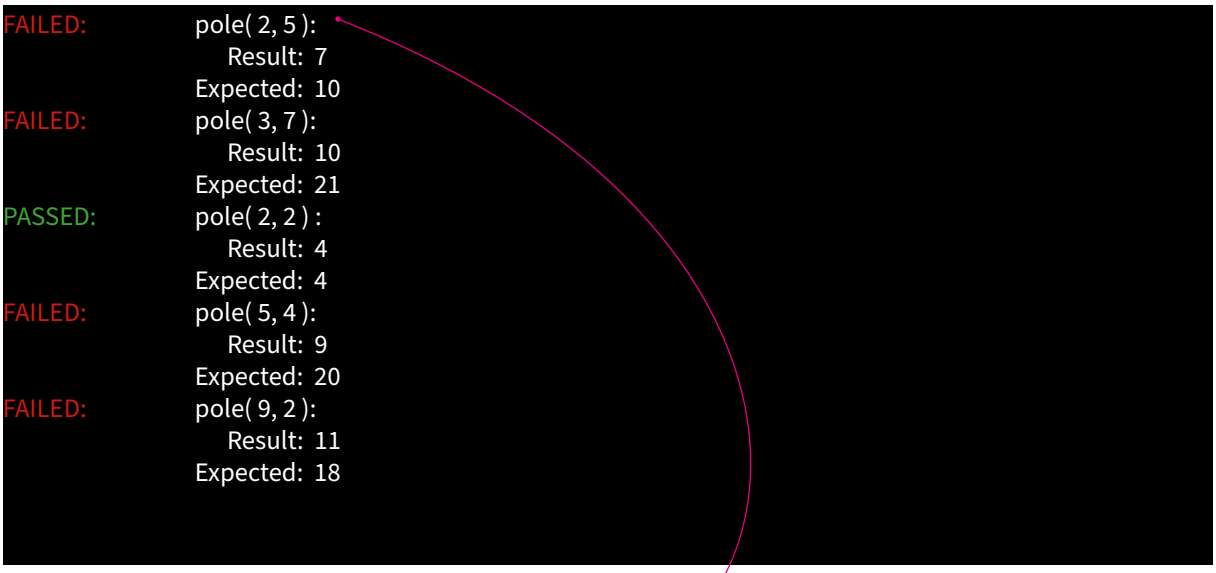
plik definicji lekcji

pole.py

```
1 def pole( x, y ):
2     return x + y
```

plik edytora kodu po edycji przez użytkownika

wyniki wyświetlane po uruchomieniu w zakładce z testami



Argumenty przekazywane do funkcji w poszczególnych testach to dane zdefiniowane w pliku definicji lekcji.

Oczywiście w funkcji testującej można z tych danych korzystać w dowolny sposób, nie muszą być wprost używane jako argumenty wywołania funkcji użytkwonika.

Zadanie z bazą danych - SQL

W zadaniach z bazą daych zamiast pliku z funkcją testującą potrzebny jest plik z bazą danych sqlite.

Plik ma mieć postać sekwencji poleceń SQL tworzących tabele i wprowadzających dane to tabel. Np. zawartość takiego pliku może wyglądać następująco:

```
CREATE TABLE famous_cats (id INTEGER PRIMARY KEY, name TEXT NOT NULL, origin TEXT NOT NULL, year INTEGER, tail_length FLOAT);
INSERT INTO famous_cats VALUES(1,'Arlene','Garfield',1980,25.499999999999999999);
INSERT INTO famous_cats VALUES(2,'Azrael','Smurfs',1959,33.100000000000001421);
INSERT INTO famous_cats VALUES(3,'Butch','Tom & Jerry',1943,56.799999999999997157);
INSERT INTO famous_cats VALUES(4,'Garfield','Garfield',1978,27.0);
INSERT INTO famous_cats VALUES(5,'Nermal','Garfield',1979,28.100000000000001421);
INSERT INTO famous_cats VALUES(6,'Puss in Boots','Shrek 2',2004,42.5);
INSERT INTO famous_cats VALUES(7,'Tom','Tom & Jerry',1940,28.699999999999999289);
INSERT INTO famous_cats VALUES(8,'Sylvester','Looney Tunes',1941,34.700000000000002843);
INSERT INTO famous_cats VALUES(9,'Claude','Looney Tunes',1949,26.899999999999998578);
INSERT INTO famous_cats VALUES(10,'Moon Cat','The Penguins of Madagascar',2009,29.300000000000000709);
CREATE TABLE cartoons (title TEXT PRIMARY KEY, release_year INTEGER, authors TEXT, country TEXT );
INSERT INTO cartoons VALUES(,Garfield',1978,'Jim Davies','USA');
INSERT INTO cartoons VALUES(,Smurfs',1958,'Peyo','Belgium');
INSERT INTO cartoons VALUES(,Tom & Jerry',1940,'William Hanna, Joseph Barbera','USA');
INSERT INTO cartoons VALUES(,Shrek 2',2004,'Andrew Adamson','USA');
INSERT INTO cartoons VALUES(,Looney Tunes',1930,NULL,'USA');
INSERT INTO cartoons VALUES(,The Penguins of Madagascar',2008,'Tom McGrath, Eric Darnell','USA');
```

W powyższym pliku utworzono dwie tabele: famous_cats oraz cartoons i do każdej z tabel wprowadzono pewne rekordy.

Plik z bazą danych powinien być plikiem tekstowym bez formatowania (baza_danych.txt).

Aby ułatwić sobie proces tworzenia pliku z bazą danych, można utworzyć tabelę z danymi np. w arkuszu kalkulacyjnym, zapisać ją w formacie CSV i skorzystać z kowertera CSV do SQL, np.:
https://www.convertcsv.com/csv-to-sql.htm

Rezultatem kowersji będzie sekwencja poleceń SQL jak powyżej w przykładzie.

W definicji kolumn w poleceniu CREATE TABLE korzystajmy jedynie z następujących rzech typów danych: INTEGER, FLOAT, TEXT.

Plik definicji lekcji z bazą danych

plik .txt

W pliku definicji lekcji z bazą danych należy zdefiniować:

1. string_query: polecenie SQL, z którym zostanie porównane na zasadzie porównania dwóch stringów polecenie wpisane przez użytkownika. Jeżeli oba polecenia będą identyczne, zadanie zostanie zaliczone - niezależnie od wszelkich innych warunków zdefiniowanych w dalszej części pliku definicji lekcji!
2. result_query: Polecenie SQL, którego rezultat będzie porównywany z rezultatem polecenia uruchomionego przez użytkwonika. Jeżeli oba rezultaty będą identyczne, zadanie zostanie zaliczone - chyba, że do zaliczenia potrzebne jest spełnienie także innych warunków zdefiniowanych w dalszej części pliku definicji lekcji.

W pliku można zdefiniować oba polecenia lub tylko jedno z nich. Oba polecenia mogą być także identyczne.

Definiowanie dodatkowych warunków:
Można sprawdzać występowanie bądź niewystępowanie w poleceniu użytkownika pewnych stringów, np.:

„GROUP BY":TAK - warunek jest spełniony, jeżeli w poleceniu użytkownika występuje string „GROUP BY"
„Clint" : NIE - warunek jest spełniony, jeżeli w poleceniu użytkownika nie występuje string „Clint"

W obu podanych wyżej warunkach wielkość liter jest bez znaczenia, tzn. warunek zadziała w ten sam sposób, jeżeli użytkwonik użyje formy „GROUP BY" jak i „group by".

Ukryta tabela

W zadaniu można utworzyć ukrytą tabelę, która nie będzie widoczna dla użytkownika. Ukryta tabela może służyć np. do porównywania rezultatów polecenia użytkownika na tabeli w bazie danych do naszego polecenia na ukrytej tabeli jeżeli obawiamy się, że użytkownik spróbuje obejść treść zadania (np. usunie z tabeli w bazie danych wszystkie rekordy i rezlutaty dowolnego zapytania SELECT będą takie same - pusta tabela).

Ukrytej tabeli należy używać we wszystkich zadaniach, w których następuje modyfikowanie tabeli, np. CREATE TABLE, UPDATE, ALTER TABLE, INSERT, DELETE itp. W takich zadaniach należy porównywać tabelę powstałą po uruchomieniu polecenia użytkownika z ukrytą tabelą.

Ukrytą tabelę tworzymy w osobnym pliku tekstowym: ukryta_baza_danych.txt, w sposób analogiczny do widocznej bazy danych.

Podpowiedzi

Do wszystkich typów zadań można zdefiniować podpowiedź dla użytkownika, która będzie dostępna pod przyciskiem z kołem ratunkowym. Podpowiedź należy umieścić w poglądowym pliku PDF oraz pliku z txt z tekstem zadania i oznaczyć nagłówkiem „Podpowiedź”.

Osiągnięcia

W misji można zdefiniować osiągnięcia do zdobycia przez użytkownika w postaci pieczątek wbijanych do książeczki z osiągnięciami na „Twoim biurku”.

Osiągnięcia są przyznawane użytkownikowi za pewną liczbę etykietek, które zdobywa się za zaliczenia zadań, które zawierają etykietki. Etykietki to po prostu pewne stringi nie zawierające spacji. Dla każdego zadania można zdefiniować dowolny zestaw etykietek. Definicje etykietek dla danego zadania należy umieścić w poglądowym pliku PDF oraz pliku z txt z tekstem zadania i oznaczyć nagłówkiem „Etykietki”. Może to wyglądać np. tak:

Etykietki:
KROL_DZUNGLI, KOT_PREZYDENTA

Takie same lub różne etykietki można zdefiniować w różnych zadaniach. Można także nie definiować żadnych etykietek.

Następnie w osobnym dokumencie dotyczącym całej misji, zaierającym m.in. tytuł misji, kolejność zadań, rodzaj zadań (obowiązkowe, nieobowiązkowe) + ewentualne dodatkowe informacje, które uznacie za istotne, umieście również definicje osiągnięć do zdobycia. Definicja osiągnięcia musi zawierać:
- tytuł osiągnięcia
- opis osiągnięcia
- nazwę etykietki
- liczbę zdobytych etykietek o podanej nazwie, którą musi zdobyć użytkownik, aby zdobyć osiągnięcie
- nazwę pliku png, zawierającego grafikę osiągnięcia

Np.:
Osiągnięcia:
tytuł: Król dżungli!
opis: Za poradzenie sobie z miejską dżunglą w misji „Znajdź kota prezydenta”
nazwa etykietki: KROL_DZUNGLI
liczba etykietek: 10
plik: krol_dzungli.png

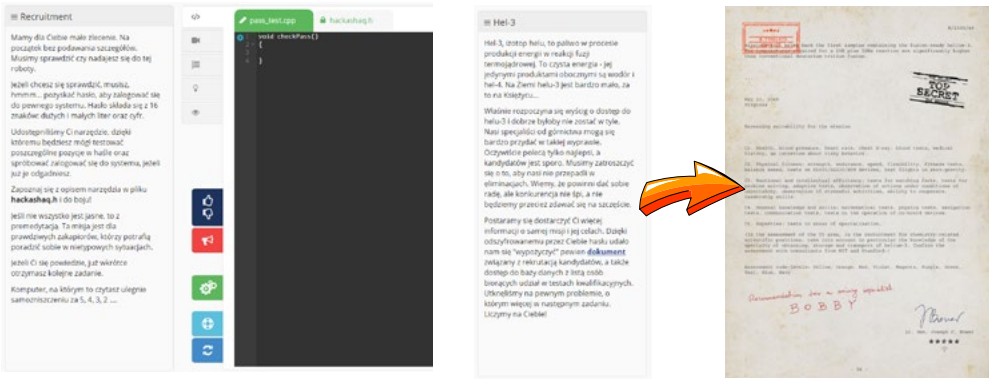
Plik z grafiką osiągnięcia musi być plikiem png z przezroczystym tłem, rozdzielczość 100x100 pikseli.

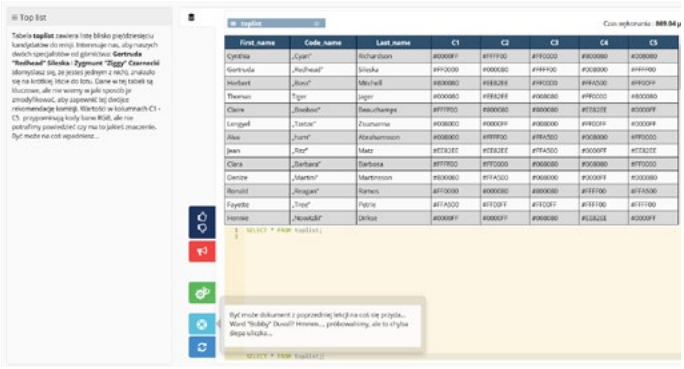
Liczba osiągnięć w teorii może być nieograniczona ale ograniczymy ją okrutnie do 3 na całą misję :)

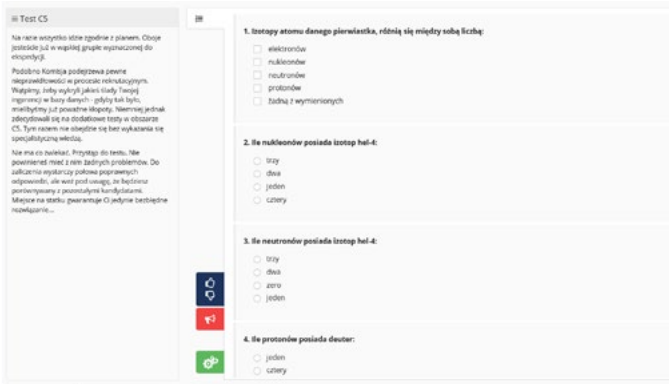
Przykłady misji

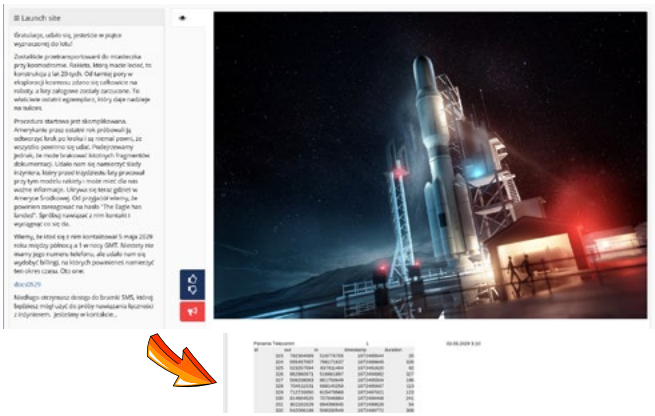
moon mission

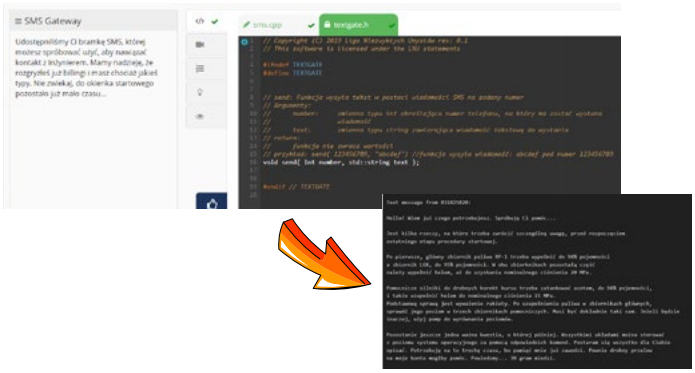
Misja zaprojektowana w ramach konkursu związanego z Polsko-Amerykańskim Mostem Innowacji.
W trakcie misji uczestnik wciela się w osobę, która ma doprowadzić do uruchomienia rakiety opartej o technologię sprzed kilkudziesięciu lat. Rakieta jest ostatnią nadzieją na wyprawę na księżyc po nowe źródło energii - hel 3...

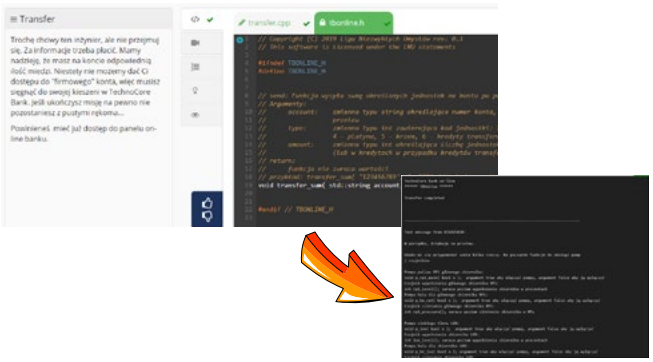


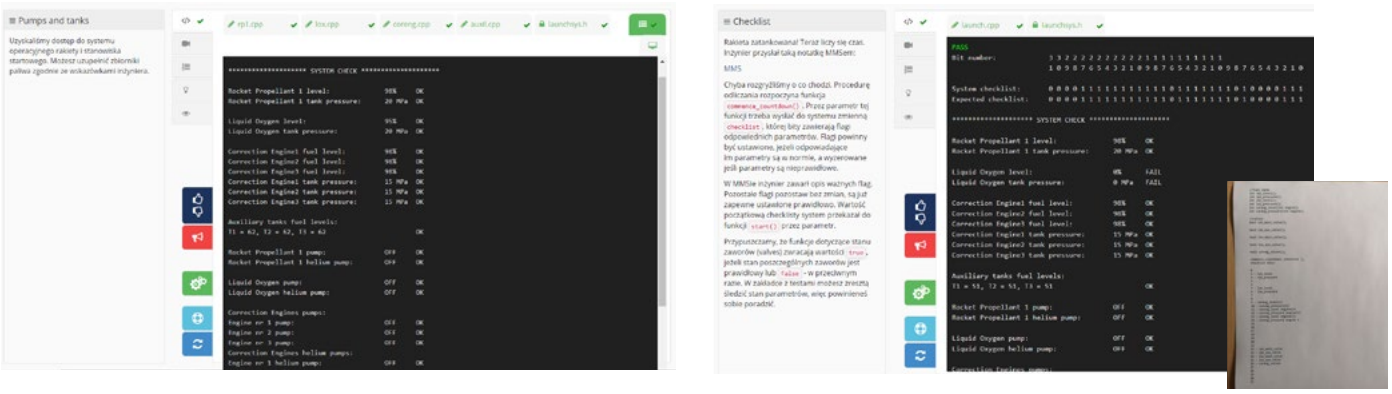


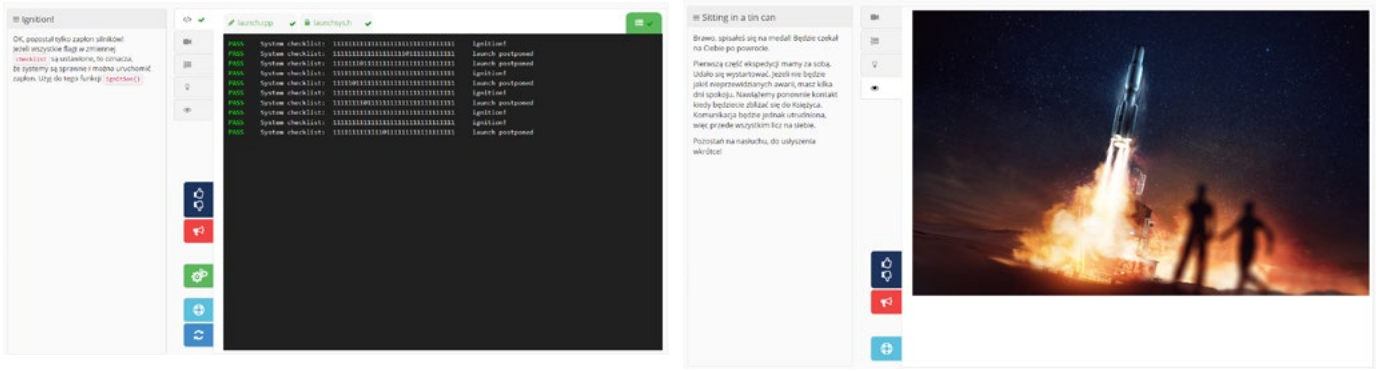






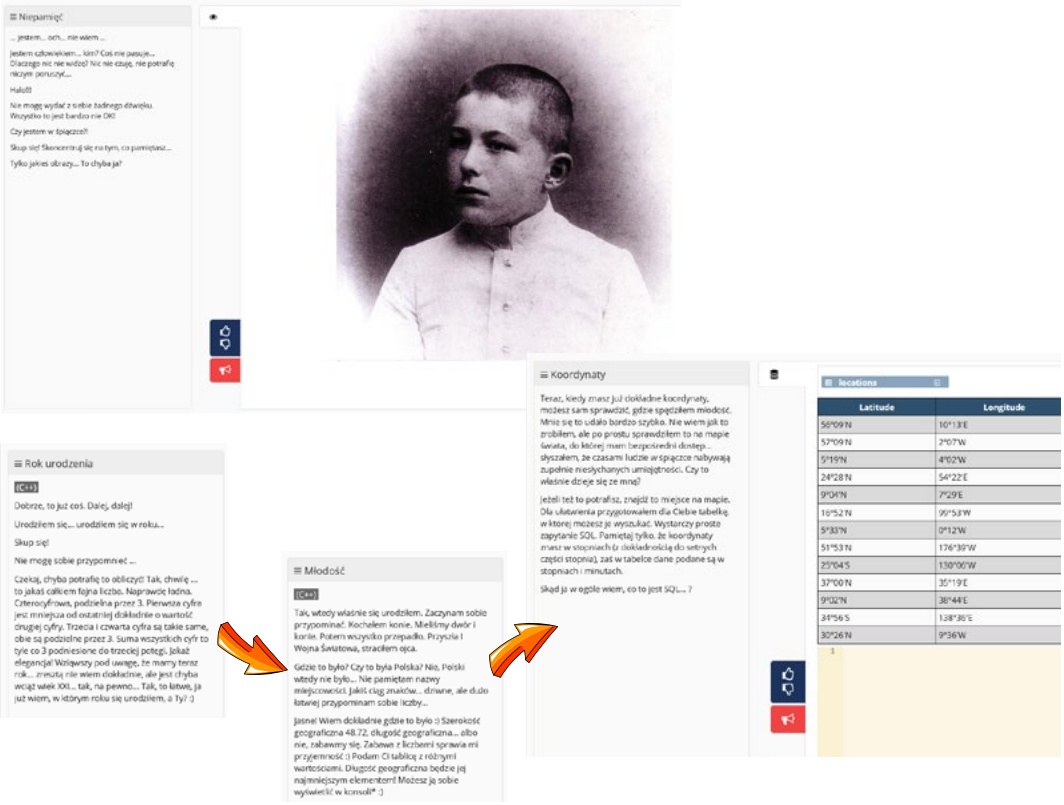


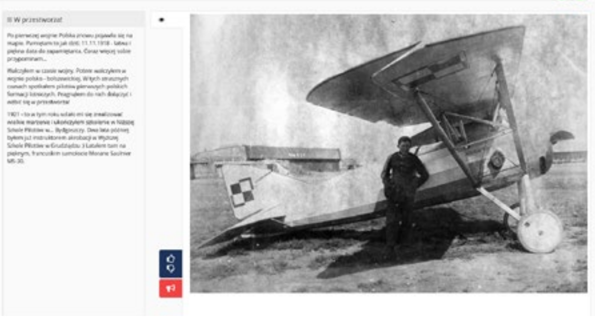


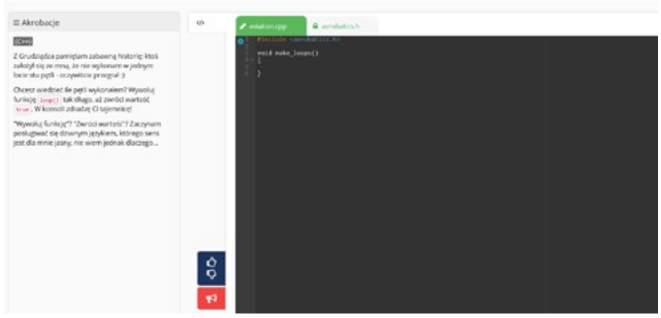


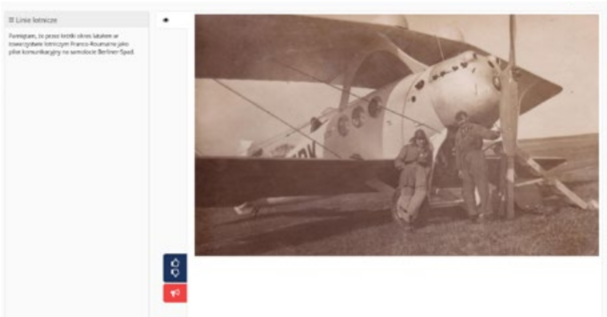
100-lecie niepodległości

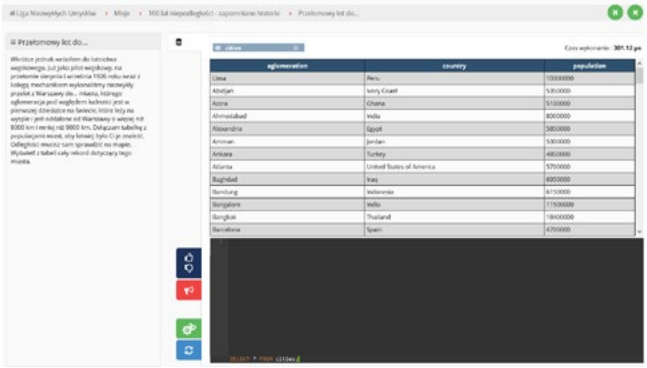
Misja udostępniona z okazji 100-lecia niepodległości państwa polskiego.

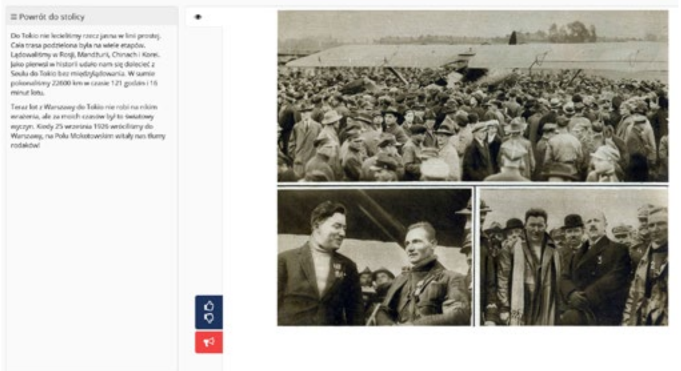


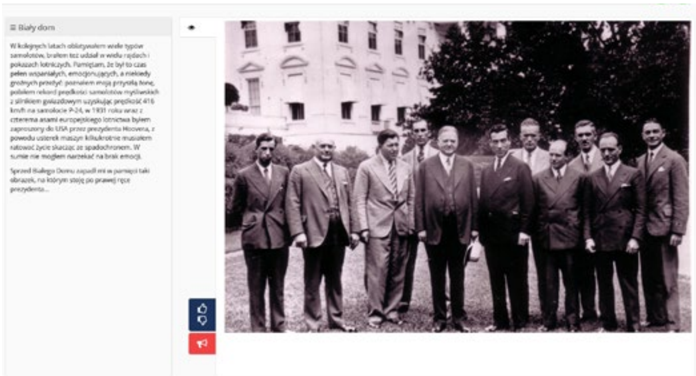

W przestworzach

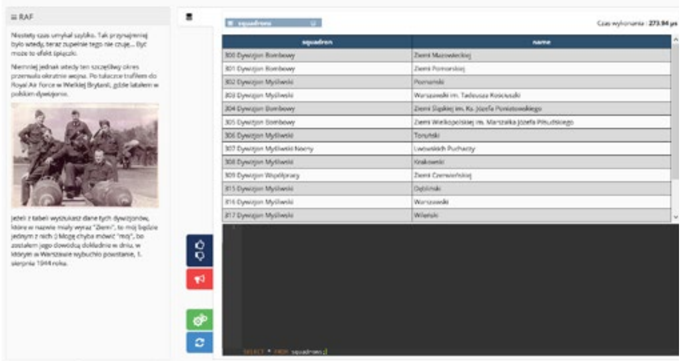

Akrobacje

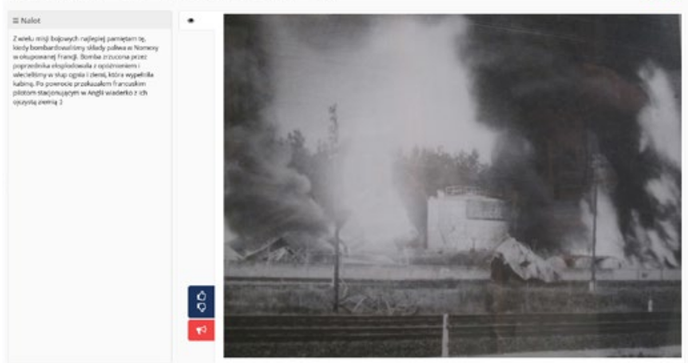

Linie lotnicze

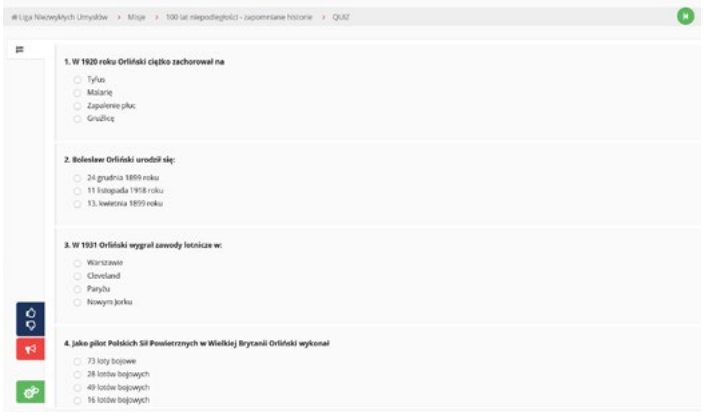

Przełomowy lot do...


Powrót do stolicy


Biały dom

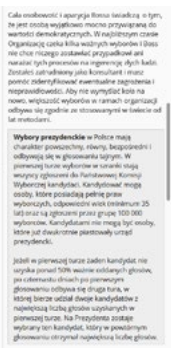

RAF


Nalot



Wyzwanie Wyborcze Aglomeracji Wrocławskiej

Misja konkursowa dla uczniów gmin - członków Stowarzyszenia Aglomeracja Wrocławska uczestniczących w projekcie „Aglomeracja Wrocławska Koduje”.



Elektorzy

Powszechny i bezpośredni system wyboru Bossa, stosowany w ostatnich wyborach nie bardzo przypadł do gustu większości wódzów, preferował bowiem gminy z największą liczbą mieszkańców.

W poszukiwaniu rozwiązania kontrymiejscowego dla mniejszych gmin (których w Organizacji była większość) i grzywieńcenia równowagi w Organizacji, wielu spojrzało na ordynację wyborczą w dalekiej Ameryce, która - jak wmawiały wielu wódzów "gwarantowała wybór osób o wszechnych kwalifikacjach intelektualnych i etycznych, osób prawdomównych, prawych i wiernych konstytucji". Cóż, może warto zaryzykować?

Wyboru Prezydenta USA dokonuje się poprzez głosowanie tzw. elektorów (nie mylić z elektrowami), zebranych w Kolegium Elektorów Stanów Zjednoczonych. Każdy stan dysponuje liczbą elektorów równą liczbie kongresmenów z tego stanu (liczbi sumie senatorów i reprezentantów). Wyjątkowo Dystrykt Kolumbia, który nie jest stanem, dysponuje trzema elektorami. Wyborcy w poszczególnych stanach głosują na swojego kandydata na prezydenta, a następnie elektorzy z danego stanu zobowiązani są oddać swój głos na kandydata, który w stanie otrzymał największą liczbę głosów.

Może się jednak zdarzyć (i zdarzało się!), że elektor przez złą wolę lub pomyłkę oddawał głos na innego kandydata. W niektórych stanach za takie działanie groziła elektorom konsekwencje karne. Niemniej jednak nawet jeden "wiarołomny" elektor może zadecydować o wyniku wyborów w całych Stanach!

Wybór Bossa Stowarzyszenie Aglomeracja Wrocławska organizacja postanowiła przeprowadzić stosując system przypominający amerykański system elektorski: mieszkańcy głosują na kandydata A lub kandydata B. Następnie radni z danej gminy oddają głos na kandydata, który w ich gminie wygrał. Przywilejmiej tuż był powinno...

Organizacja poprosiła Cię, abyś sprawdził, czy wszyscy radni oddali prawidłowo swój głos. Do funkcji `sprawdzGlosy()` przez parametr `dane` dostaniesz listę, zawierającą czternastoelementowe krotki. Pierwszym elementem każdej krotki jest nazwa gminy, drugim - liczba radnych, trzecim - liczba głosów oddanych przez mieszkańców na kandydata A, czwartym - liczba głosów oddanych przez mieszkańców na kandydata B. Przez parametr `wynik` otrzymasz krotkę, zawierającą dwie liczby, oznaczające liczby głosów oddanych przez radnych kolejno na kandydata A oraz B.

Funkcja powinna zwrócić wartość **True**, jeżeli wyniki głosów radnych są prawidłowe lub **False** - jeżeli coś się w nich nie zgadza. W tej drugiej sytuacji organizacja przeprowadzi odpowiednie dochodzenie.

Przykład 1:

Wejście:

```
[
("Syreng", 21, 5000, 2000),
("Czernika", 15, 900, 1300),
("Zmigród", 19, 2500, 1200)
],
( 36, 15 )
```

Wyjście: **True**

Przykład 2:

Wejście:

```
[
("Kierwiów", 15, 800, 1300),
("Długolesie", 21, 8000, 5000),
("Żychów", 13, 2030, 1003),
("Zawonia", 15, 1500, 1000),
],
( 59, 16 )
```

Wyjście: **False**

```
election.py
def sprawdzGlosy(dane, wynik):
    pass
```

Metoda D'Hondta

Boss ma do swojej dyspozycji ciało doradcze w postaci tzw. Prawdziwych Mędrców, która składa się z 77 osób. Boss zna się na wszystkim, ale nie na wszystko ma czas, dlatego także ciała jest mu niekiedy przydatne (rzadko).

Nie bez znaczenia są także apanaże wiążące się z byciem członkiem lub członkinią tego szacownego grona. Stronnictwa, które wystawiają swoich kandydatów na Mędrców, mogą w ten sposób spotykować część środków Organizacji, zaskarbiając sobie wdzięczność kandydatów i zapewniając powszechną szczęśliwość.

Najprostsze metody podziału stanowisk na podstawie wyników wyborów - takie jak ma ordynacja większościowa - nie zawsze dobrze się sprawdzają. Np. ordynacja większościowa do parlamentu może powodować bardzo duże rozdrobnienie parlamentu: dostają się do niego posłowie z tak wielu partii politycznych, że trudno o stabilną większość, dającą szansę skutecznych rządów do czasu kolejnych wyborów. Z tego powodu powstały różne metody podziału mandatów, które mają takim sytuacjom zapobiegać, jedną z nich jest **metoda D'Hondta**. W tej metodzie dla każdego komitetu wyborczego, który proporcjonalnie, obliczane są tzw. ilorazy wyborcze, czyli kolejne ilorazy całkowitej liczby głosów uzyskanych przez dany komitet i kolejnych liczb naturalnych. Kolejność wartości tych ilorazów pomiędzy komitetami decyduje o podziale między poszczególnych komitetów wyborczych. Można to przedstawić wzorem:

Ii = G / i

w którym:
Ii - to kolejny iloraz wyborczy
G - liczba głosów oddanych na dany komitet wyborczy
i - kolejna liczba naturalna

Przykład:
W wyborach startowały trzy komitety A, B i C i uzyskały kolejno liczby głosów: 450, 150, 210. Do podziału jest 6 mandatów. Ilorazy wyborcze:

dzielnik	A	B	C
1	450	150	210
2	225	75	105
3	150	50	70

Kolejność dzielników i podział mandatów:
1. 450 (A)
2. 225 (A)
3. 210 (C)
4. 150 (A)
5. 150 (B)
6. 105 (C)

Cwarty mandat został przydzielony komitetowi A, pomimo że iloraz 150 miał także komitet B. W tej sytuacji zadecydowała ogólna większa liczba głosów oddanych na komitet A. Kolejność przydzielania mandatów w przypadku takich samych ilorazów ma znaczenie w przypadku ostatnich rozdzielanych mandatów.

Podział A - 3 mandaty, B - 1 mandat, C - 2 mandaty.

Metoda D'Hondta stosowana jest w wielu różnych ordynacjach, m.in. w Polsce w ordynacji do wyborów do Sejmu, do rad gmin (powyżej 20 tys. mieszkańców), powiatów i sejmików województw, a także częściowo w wyborach do Parlamentu Europejskiego, gdzie decyduje o podziale mandatów pomiędzy komitety, ale już nie pomiędzy listy okręgowe danych komitetów.

Między wystawiani są przez poszczególne stronnictwa i wybierani metodą D'Hondta. Aby Między z danego stronnictwa weszli do rady, stronnictwo musi zdobyć **minimum 5%** ogólnej liczby głosów.

Boss poprosił Cię o sprawdzenie systemu przeliczania głosów, który posiada organizacja. Twoim zadaniem będzie przeprowadzenie symulacji wyborów. Do funkcji `test(Dhondta)` otrzymasz przez parametr `glosy` słownik, zawierający sześcioelementową listę, z licbami głosów oddanymi na poszczególne sześć stronnictw uczestniczących w wyborach: 1 element - zawiera głosy oddane na stronnictwo nr 1, 2 element - na stronnictwo nr 2 itd.

Funkcja `test(Dhondta)` powinna zwrócić **sześcioelementową listę**, której poszczególne **elementy** będą zawierały liczby mandatów dla poszczególnych stronnictw obliczonych metodą **d'Honta** z zachowaniem **5% progu**.

Pamiętaj, że do podziału jest 77 mandatów.

Przykład:

Wejście:
`[12561, 31504, 16739, 10745, 19335, 5853]`

Wyjście:
`[10, 26, 13, 8, 16, 4]`

Możesz założyć, iż nie zdarzy się przypadek, że dla ostatnich przydzielanych mandatów kilka komitetów będzie miało jednocześnie taki sam iloraz wyborczy oraz identyczną liczbę głosów. Mogą natomiast zdarzyć się przypadki takich samych ilorazów.

Inne metody

Po ostatnich spotkaniach Boss nabrał podejrzeń, że prezes jednego ze stronnictw może próbować majstrować przy systemie obliczania głosów do rady Prawdziwych Mędrców i poprosił swoje wpływy sprzątowac zmienić metodę D'Hondta na inną, która będzie bardziej dla niego korzystna.

Będziesz musiał również przeliczyć głosy i sprawdzić którą z metod rozdzielił mandaty system Organizacji. Jeżeli nie będzie to metoda D'Hondta, trzeba będzie przygotować się sprawę bliżej i wyciągnąć konsekwencje...

Poza metodą D'Hondta jest jeszcze kilka innych metod przeliczania głosów na mandaty. Do najpopularniejszych należą: metoda Hare'a-Niemeyera i metoda Sainte-Lague.

W **metodzie Hare'a-Niemeyera**, zwanej także metodą największych reszt liczbę mandatów dla danego komitetu określa podłoga (czyli wartość z pominięciem części po przecinku) wyrażenia:

(Gk * M) / Gt

gdzie:
Gk - liczba głosów oddanych w okręgu na dany komitet
M - liczba mandatów do rozdzielenia w danym okręgu
Gt - ogólna liczba ważnych głosów oddanych w danym okręgu.

Przykład:
W wyborach startowały trzy komitety A, B i C i uzyskały kolejno liczby głosów: 450, 150, 210. Do podziału jest 5 mandatów. Liczba mandatów dla komitetów:

A = (450 * 6) / 810 = 3,33 → **3**
B = (150 * 6) / 810 = 1,11 → **1**
C = (210 * 6) / 810 = 1,55 → **1**

Jeżeli nie zostaną w ten sposób rozdzielone wszystkie mandaty, pozostałe przypadają tym komitetom, które w swoich ilorazach uzyskały najwyższe reszty, czyli najwiekszą wartość po przecinku. W powyższym przykładzie szósty mandat przypada komitetowi C, który ma 0,55 reszty.

Metodą Hare'a-Niemeyera są m. in. dzielone mandaty przypadające na poszczególne listy okręgowe komitetu wyborczego w wyborach do Parlamentu Europejskiego - liczba mandatów przypadających na cały komitet oblicza się metodą D'Hondta, o czym wiesz już z poprzedniego zadania. Jest to zrobione specjalnie, aby nikt nie zdołał się w tym połapać.

Metoda Sainte-Laguë jest podobna do metody D'Hondta i polega na rozdziale mandatów w oparciu o największe ilorazy wyborcze, przy czym zamiast kolejnych liczb naturalnych, jako dzielników używa się kolejnych liczb nieparzystych.

Przykład:
W wyborach startowały trzy komitety A, B i C i uzyskały kolejno liczby głosów: 450, 150, 210. Do podziału jest 6 mandatów. Ilorazy wyborcze:

dzielnik	A	B	C
1	450	150	210
3	150	50	70
5	90	30	42

Kolejność dzielników i podział mandatów:
1. 450 (A)
2. 210 (C)
3. 150 (A)
4. 150 (B)
5. 90 (A)
6. 70 (C)

Podział: A - 3 mandaty, B - 1 mandat, C - 2 mandaty.

Podobnie jak w metodzie D'Hondta, przy tej samej wartości ilorazu decyduje wyższa suma głosów oddanych na dany komitet.

Podobnie jak w poprzednim zadaniu, do funkcji `sprawdzMetode()` przez parametr `glosy` dostaniesz listę z liczbami głosów oddanych na poszczególne stronnictwa. Przez parametr `wynik` otrzymasz sześcioelementową listę, z liczbami przydzielonych poszczególnym stronnictwom mandatami. Twoim zadaniem będzie zidentyfikowanie metodą, których reguł użył system przydzielający. Przekazuj pod tym kątem metody:
1. d'Hondta
2. Hare'a-Niemeyera
3. Sainte-Laguë

Funkcja powinna zwrócić **zbiór** z numerami lub numerami metod, które przydzielają identyczne liczby mandatów, jak te, które zostały przekazane przez parametr `wynik`.

Uwaga: pięcioprocentowy próg wyborczy obowiązuje tylko w przypadku metody D'Hondta!

Oczywiście do podziału jest 77 mandatów.

Przykład 1:

Wejście:
`[41648, 26693, 1331, 2151, 7446, 1150], [43, 27, 0, 0, 7, 0]`

Wyjście:
`{ 1 }`

Przykład 1:

Wejście:
`[41648, 26693, 1331, 2151, 7446, 1150], [43, 27, 0, 0, 7, 0]`

Wyjście:
`{ 1 }`

```
election.py
def sprawdzMetode(glosy, wynik):
    pass
```

Pierwsza tura

Mamy wyniki pierwszej tury wyborów wójtów, burmistrzów i prezydentów gmin zrzeszonych w Organizacji. W niektórych miejscach pierwsza tura okazała się rozstrzygającal

Będziesz musiał sprawdzić, które to gminy i czy Twoje obliczenia zgadzają się z ogłoszonymi rezultatami.

W pozostałych gminach na końcowe wyniki przyjdzie nam poczekać jeszcze dwa tygodnie.

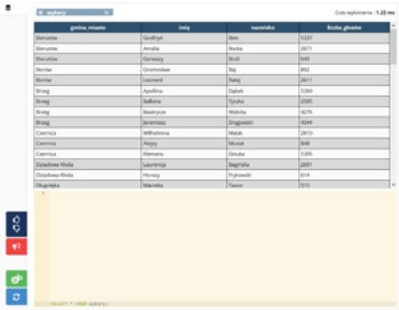
Wybory wójtów, burmistrzów i prezydentów miast odbywają się w Polsce wg ordynacji podobnej do wyborów prezydenta kraju. Wygrywają ci kandydaci, którzy zdobędą powyżej 50% spośród ważnie oddanych głosów. Jeżeli żaden z kandydatów nie osiągnie tego progu, dwóch z największą liczbą głosów przechodzi do drugiej tury, która odbywa się 14 dni po pierwszej. W drugiej turze wygrywa pretendent z większą liczbą głosów.

Zadanie:

tabela wyboru zawiera dane kandydatów w wyborach na wójtów, burmistrzów i prezydentów w gminach zrzeszonych w Organizacji oraz liczbę głosów jakie otrzymali od wyborców w swoich gminach w pierwszej turze wyborów. Wyświetl **w kolejności alfabetycznej samej nazwy gmin***, w których pierwsza tura okazała się rozstrzygająca.

Jesteś profesjonalistą. Twoje zapytanie powinno być uniwersalne, tzn. powinno zwrócić prawidłowe wyniki dla dowolnych danych w kolumnie **liczba głosów**. Zdarzają się bowiem sytuacje, kiedy przegrany kandydat krzyczy, wyrywając sobie włosy: "Wygrałem! Okradli mnie! Skończyli wybory!". Dla jego zdrowia psychicznego czasami przyjdzie się wtedy ponownie głosy i dokonuje ewentualnych korekt.

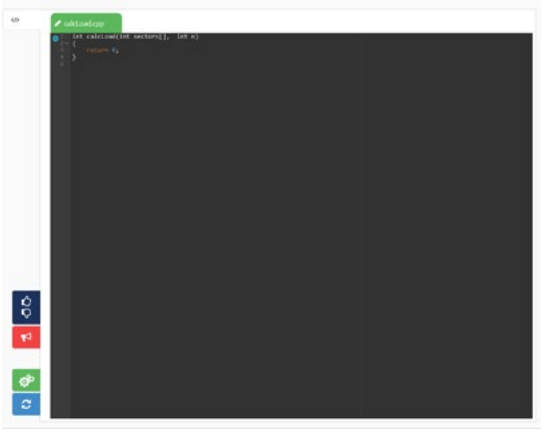
**Uwaga: w tej wersji SQLite polskie znaki mają wyższą wartość niż standardowe litery tabeli ASCII. Zatem w tym przypadku np. nazwa Grudziądz Kholfa będzie alfabetycznie przed nazwą Długołęka ("ł" > "ą"), a Zawonia przed Środa Śląska ("S" > "Z"). Użyj taką kolejność za prawidłową.*

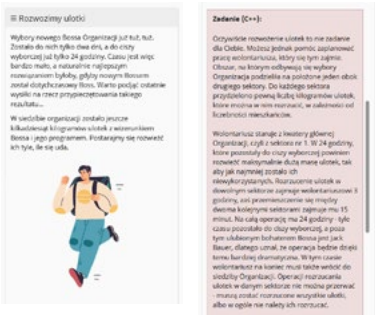


gmina_miasto	imię	nazwisko	liczba_głosów
Barwałdy	Godfryd	Bies	5237
Barwałdy	Amalia	Rocka	2671
Barwałdy	Gerwazy	Brud	940
Barwałdy	Gromisław	Raj	852
Barwałdy	Leonard	Rataj	2611
Brzeg	Apolinia	Dąbek	1269
Brzeg	Balbina	Tyszka	2585
Brzeg	Beatrycze	Walicka	4276
Brzeg	Jeremiasz	Długowski	4244
Czarnica	Wilhelmina	Wałak	2810
Czarnica	Alojzy	Musiał	640
Czarnica	Klemens	Dziuba	1385
Działowa Wieś	Laurencja	Bagińska	2691
Działowa Wieś	Horacy	Prykowski	614
Długołęka	Marietta	Tawor	515

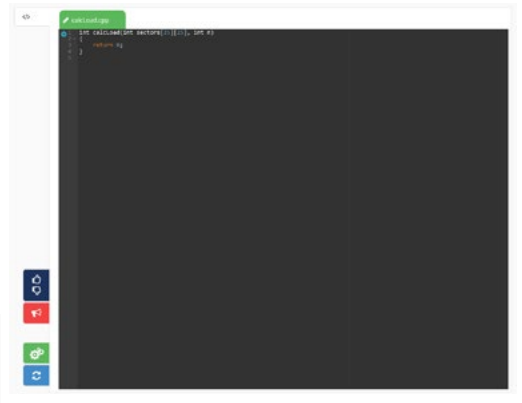

Druga tura
Po czternastu dniach, w gminach w których nie udało się wyłonić zwycięzców, przeprowadzono drugą turę wyborów, w których startowało już tylko dwóch kandydatów na jedno miejsce.
Zadanie:
Wynikowa tabela powinna zawierać kolumny:
gmina_miasta, imię, nazwisko.


wybory
Czas wykonania : 623.23 µs
gmina_miasto
imię
nazwisko
liczba_głosów
druga_tura

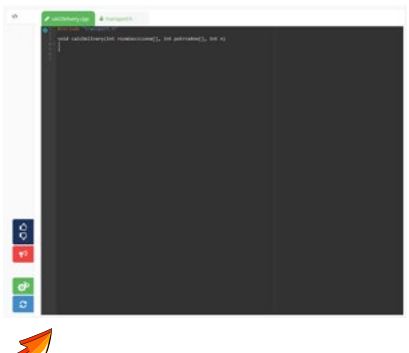

calc.cpp

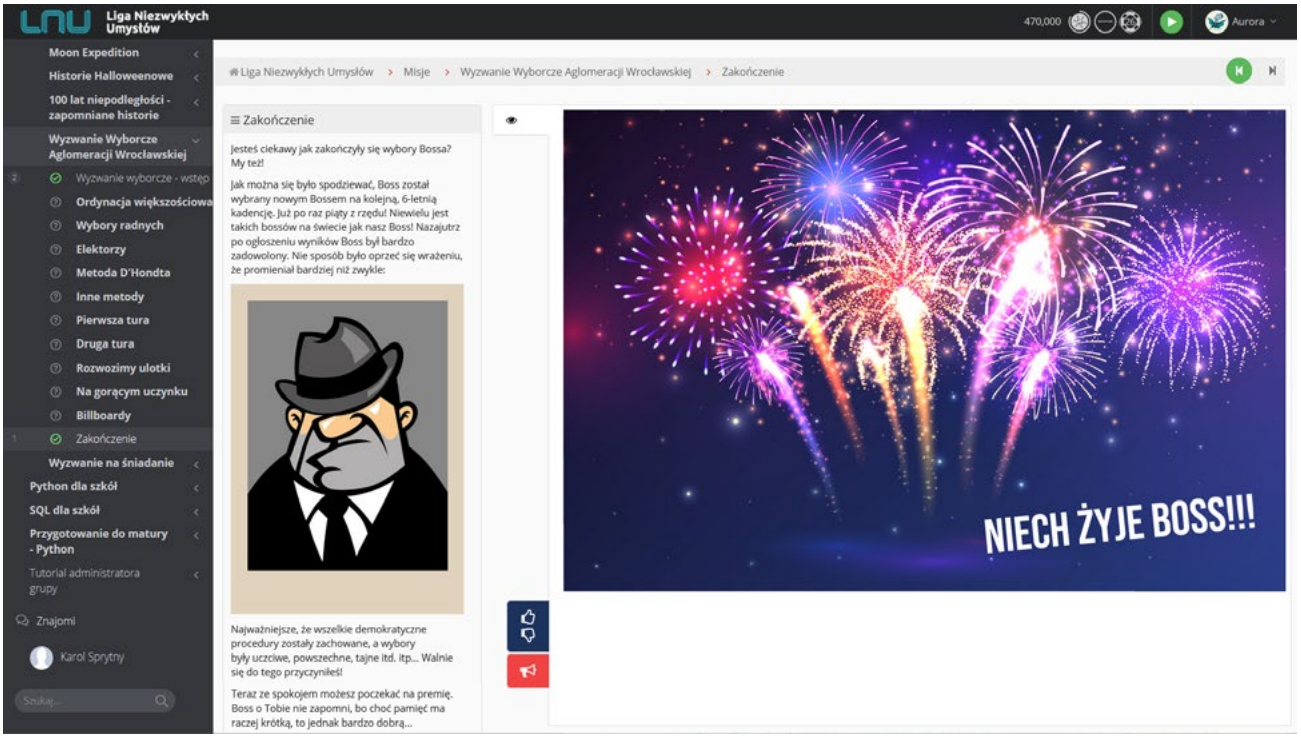

Rozwozimy ulotki
Zadanie (C++):
Wejście:
Wyjście:
Przykład:
Wyjaśnienie:


Na gorącym uczynku
Zadanie (C++):
Wejście:
Wyjście:
Przykład:
Wyjaśnienie:


calc.cpp


Billboardy
Zadanie (C++):
API:
Przykład:
Wyjaśnienie:


calc.cpp
transport.cpp



© Fundacja Liga Niezwykłych Umysłów