Dokumentacja projektu “Steady as she goes!”

Tomasz XXXXX

Treść zadania

Kapitan planuję przepłynąć akwen reprezentowany przez kwadrat wielkości NxN. Problemem jest jednak fakt że akwen znajduje się w płytkim rejonie, w związku z czym wasz statek może przepływać tylko przez pola o głębokości większej niż jego zanurzenie. Jako wprawny nawigator doskonale wiesz co musisz w tym momencie zrobić.

Korzystając z dokładnej locji (mapy obrazującej wysokość dna), pomóż kapitanowi zaplanować trasę (dowolną/najkrótszą) z górnego lewego pola mapy [0,0] do dolnego prawego [n-1,n-1]. Wynikową trasę nanieś na mapę i przedstaw kapitanowi.

Przygotuj interfejs graficzny który umożliwi komunikację z programem. Interfejs powinien umożliwić stworzenie locji w formie mapy NxN w którą użytkownik będzie mógł wpisywać wysokość terenu w m.n.p.m. Interfejs powininen w czytelny sposób obrazować głębokości i wysokości (jeżeli teren jest powyżej 0 powinien zostać odpowiednio zaznaczony jako ląd, poniżej 0 jako morze). Głębokość i wysokość terenu powinna mieć wpływ na wizualizację.

Wejście:

`Macierz NxN z liczbami (float) opisującymi teren, zanurzenie fregaty`

Wyjście:

`Wizualizacja najkrótszej trasy`

Biblioteki:

numpy, matplotlib, Pillow

Obsługa programu

Projekt jest napisany w języku Python3, więc niezbędna jest jego instalacja, aby uruchomić program. Zalecana wersja to 3.8.

Po zainstalowaniu niezbędnych pakietów (`pip install -r requirements.txt`) należy uruchomić plik `main.py` znajdujący się w głównym folderze. Pojawi się okno edytora map:

Opis kontrolek, od góry:

1. Tablica do wpisywania wysokości w m.n.p.m
   - obsługuje wklejanie wartości przez ctrl+v do zaznaczonego regionu komórek
2. Import
3. Export
4. Zmiana rozmiaru mapy
   - Mapa nie jest czyszczona podczs rozszerzania i zmniejszania, więc można łatwo modyfikować istniejące mapy
5. Rozwiązanie obecnej mapy
   - Program wyświetli kolorową mapę z zaznaczoną trasą, lub poinformuje użytkownika o braku rozwiązania
6. Pole do wpisania zanurzenia statku
   - Tutaj wpisujemy wymaganą głębokość dna oceanu, więc powinna to być liczba ujemna
7. Ignoruj wartości poniżej głębokości statku
   - Dodatkowa możliwość uproszczenia tych wartości, które i tak nic nie wnoszą do rozwiązania. Zwiększa czytelność gdy mapa ma nieregularne wartości (więcej informacji w części o wizualizacji)

Figure 1: Początkowe okno edytora

Import i Export

Program ma możliwośc czytania oraz zapisywania stworzonych map w formacie JSON, zawierających kwadratową macierz w postaci dwuwymiarowej tablicy.

Przykladowy plik mapy o rozmiarze 3:

```
[
    [-1,2,3],
    [-1.2,-2.56,-3.141],
    [-5,-10.5,-15.01]
]
```

W przypadku problemów przy czytaniu lub zapisywaniu mapy, program wyświetla informacje o błędzie w okienku, z nadzieją że pomoże to użytkownikowi określić przyczynę błędu.

Wizualizacja

Mapa oceanu wraz rozwiązaniem (o ile istnieje) jest wyświetlana w formie obrazka `NxN` pikseli. Każdy piksel ma kolor zależny od m.n.p.m danej lokalizacji. Rozwiązanie jest zaznaczone na czerwono

Kolory dobierane są na podstawie mapy kolorów:

Figure 2: mapa kolorów ‘delta’ z biblioteki `cmocean`

Lecz problem polega na tym, że nie możemy bezpośrednio objąć całego zakresu taką mapą, ponieważ wtedy punkt 0 m.n.p.m może otrzymać różne kolory w zależności od wartości na mapie, a sensowne jest żeby miał kolor biały. Więc wybieramy początek i koniec skali jako:

$$\pm \max |mapa|$$

W ten sposób różne mapy mają różne kolory dla tych samych wysokości, więc kolory mogą dostosować się do przypadku. Zachowana jest również proporcja co do lądu i oceanu.

Ponieważ skala kolorów jest liniowa, takie rozwiązanie ma problem z wartościami o dużym odchyleniu standardowym, ponieważ wartość maksymalna "kradnie" kolory dla wartości bliższych 0, które mogą być ważne dla statku. Aby trochę zniwelować ten problem można skorzystać z opcji "Cut values below ship depth", która wymusza wartość minimalną na skali jako głębokość statku. Wynikiem jest dokładniejszy rozkład kolorów powyżej głebokości statku:

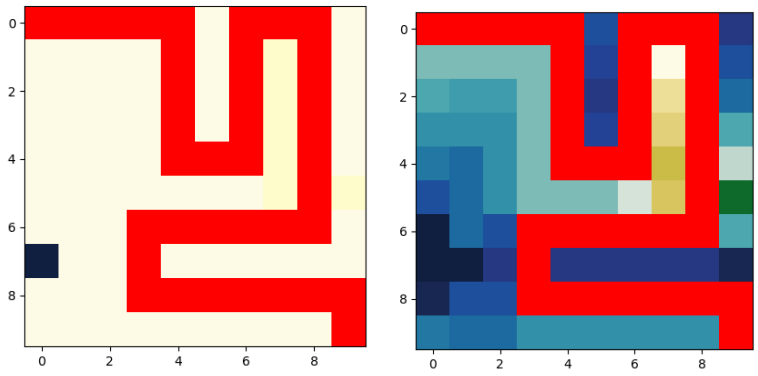

Figure 3: Obcięcie niskiej wartości powoduje odkrycie szczegółów

Powyższa zmiana dotyczy tylko końcowego obrazu, nie wpływa na rozwiązanie

Struktura kodu

Kod został podzielony na dwa moduły: `frontend` i `mapsolver`. Plik `main.py`, który uruchamia cały program, jedynie oczekuje od frontendu funkcji `frontend.start()`.

Moduł `mapsolver` definiuje klasę `MapSolver`, która jest używana przez frontend do rozwiązania mapy. Frontend wywołuje metodę `MapSolver.solve_map()`, podając w argumentach tylko macierz `NxN` oraz głębokość statku, i oczekuje rozwiązania w postaci listy kroków. Tak długo jak `MapSolver` spełnia ten "standard", może działać w dowolny sposoób, tutaj traktuje mape jako graf oraz korzysta z prostego algorytmu Breadth First Search do znalezienia najkrótszej trasy. Do implementacji grafu stworzona została mała klasa `Node`, która pozwala zrekonstruować ścieżkę do punktu startowego, gdy algorytm dojdzie do końca. W implementacji algorytmu isnieje możliwosc zmiany punktu początkowego i końcowego, ale nie jest to używane w tym projekcie.

Frontend jest oparty o bibliotekę `PySide6`, z tego powodu, choć trochę nie zgadza się to z założeniem frontendu, zarządza również faktyczną tablicą pythonową w której przechowywane są dane o mapie. Jest to konieczne, ponieważ większość kodu modyfikującego tą tablicę jest zależna od `PySide6` (model tabeli). Klasa modelu została wydzielona do pliku `table.py`, a reszta logiki okna jest w pliku `window.py`

Gdy frontend zna już rozwiązanie, może użyć pliku `visualise.py` do pokazania użytkownikowi trasy. W tym pliku znajduje się jedna funkcja `show_map()`, której argumentami jest tablica oraz rozwiązanie. Tutaj znowu sposób implementacji wizualizacji pozostaje do wyboru, byleby plik `visualise.py` zawierał opisaną funkcję.