Gliwice, 11-12-2017 r.

# Sprawozdanie do Projektu końcowego

Autor: Arkadiusz Duliban, Wiktor Gruszczyński

Kierunek : Informatyka (st.1, sem. 7)

***Temat projektu:***

Tematem projektu jest zaimplementowanie i porównanie kilku przykładowych algorytmów służących do przeszukiwania grafów: przeszukiwanie wszerz (BFS), przeszukiwanie w głąb (DFS), General Best First (GBF) oraz A\*.

***Cel projektu:***

Naszym celem jest zaimplementowanie algorytmów BFS, DFS, GBF oraz A\*. W tym celu stworzone zostanie narzędzie służące do demonstracji działania algorytmów przeszukiwania grafów. Porównywanie będzie się odbywać poprzez znajdowanie drogi między dwoma punktami umieszczonych na planszy składającej się z siatki kwadratów o wymiarze *n* x *n*. Dodatkowo niektóre elementy siatki będą mogły zostać wyłączone – tzn. droga nie będzie mogła przeciąć żadnego z nich.

Algorytmy zostaną przetestowane dla różnych wariantów danych wejściowych (położenia punktu startowego i końcowego, inna konfiguracja i liczebność przeszkód). Porównane zostaną: liczba iteracji, długość drogi, ilość wyznaczonych sąsiadów.

***Sposób realizacji projektu:***

Projekt zostanie przygotowany w formie aplikacji webowej. Użytkownik będzie zadawać parametry wejściowe poprzez modyfikowanie interaktywnego schematu mapy (w postaci siatki). W ten sposób będzie dokonywany wybór punktu startowego, końcowego oraz bloków – przeszkód. Algorytm będzie wywoływany po każdorazowej akcji wykonanej przez użytkownika.

Do uruchomienia aplikacji wystarczy przeglądarka internetowa wspierająca obsługę elementu *canvas* (każda zaktualizowania na przełomie ostatnich 3-4 lat). Uruchomić należy plik ***index.html***

***Komentarz do zadania:***

1. Stworzono graf – „mapę” do przeszukiwania. Mapa składa się z *n* wierszy (*rowsCount)* oraz z *m* kolumn (*boxPerRow*). Mapa opisywana jest poprze tablicę *net* składającą się z *n∙m* obiektów, których struktura przedstawia się następująco:
   * *x* - pozycja poziomo (w pikselach) obiektu na graficznych odwzorowaniu siatki
   * *y* - pozycja pionowo (w pikselach) obiektu na graficznych odwzorowaniu siatki
   * *type*  - rodzaj bloku (startowy, końcowy, przeszkoda, zwykły)
   * *neighId* – jeśli blok jest sąsiadem, zmienna przechowuje informacje, który z kolei jest to sąsiad
   * *isSearched* – czy element był brany pod uwagę przy poszukiwaniach
   * *lastSearched* – czy element występował jako ostatni przy przeszukiwaniu

Do prawidłowego działania algorytmu jest wymagana jedynie zmienna *type*. Pozostałe są użyte jedynie w celu wizualizacji wyniku.

1. Zaimplementowane zostały kolejno wszystkie z algorytmów.  
   Procedury *BFS\_search,*  D*FS\_search* jako parametr początkowy przyjmują numer bloku startu oraz numer bloku końca. Procedura *GBF\_search* z kolei wymaga podania dokładnie tych samych parametrów oraz – dodatkowo – referencji do funkcji heurystycznej A lub B. Związane jest to z tym, iż algorytm pozwalamy uruchomić w dwóch konfiguracjach, z różnymi heurystykami.   
     
   Jako wynik otrzymywane są dwie listy:
   * lista zawierająca kolejne numery bloków tworzących wyznaczoną trasę między blokami startu i końca (A i B)
   * lista składająca się z list zawierających identyfikatory sąsiadów punktów wyznaczanych w kolejnych
2. Wyniki algorytmów przedstawiane są na sąsiadujących ze sobą „mapach” – obiektach typu *canvas*. Każde kliknięcie (akcja) powoduje wywołanie funkcji uruchamiającej algorytm – *OnCanvasClick()*. Rolę pomocniczą pełni funkcja *ChangeSquareType()*, która zmienia typ ostatnio klikniętego obiektu: jeśli nie było jeszcze bloku startowego – w start; jeśli nie było końcowego – w końcowy; jeśli jest to blok zwykły – w przeszkodę; jeśli jest to blok inny niż zwykły – zamienia blok w blok zwykły.

Za rysowanie jest odpowiedzialna funkcja *DrawNet*, która rysuje siatkę na mapie zadanej jako argument. Wszelkie dane potrzebne do wyrysowywania obiektu pochodzą z danych wyjściowych algorytmu.

1. Dodatkowo skonstruowane narzędzie zostało wyposażone w następujące opcje:
   * Możliwość wyświetlenia na mapie sytuacji aktualnej dla *i*-tej iteracji. W tym celu wywoływana jest funkcja *SetIteration(i)* – wyświetlanie wyniku jest ograniczone i nie wyświetlamy wszystkich danych *N* iteracji, lecz wyświetlanie kończymy na iteracji *i.*

* + Odtwarzanie animacji przedstawiającej poszukiwanie drogi – realizowane przez funkcję *playIterations()*, która samoczynnie przestawia numer iteracji od 1 aż do N.
  + Czyszczenie siatki ze zmian wprowadzonych poprzez uruchomienie algorytmu – funkcja EraseCanvas(clearAlsoObjects) odpowiada za zerowanie ustawień do stanu początkowego. Jako parametr przyjmowana jest zmienna logiczna informująca, czy należy również wyczyścić bloki startowe, końcowe i przeszkody.
  + Zmiana rozmiaru siatki *net*

1. Powyżej ograniczono się jedynie do wymieniania najważniejszych funkcji – dużą część kodu odgrywają funkcje pomocnicze oraz odpowiadające za interakcję z użytkownikiem aplikacji.
2. Uzyskane wyniki …