Sprawozdanie końcowe
Projekt indywidualny

Łukasz Knigawka

29 listopada 2018

Spis treści

1 Wprowadzenie

Dokument opisuje działanie programu w momencie zakończenia pracy nad projektem. Porównano założenia projektowe z faktycznym sposobem zachowania programu. Dokonano krytycznej analizy realizacji wymagań a także przedstawiono krótką retrospekcję. Zakładane funkcje programu przedstawiono w specyfikacji funkcjonalnej. Szkic realizacji problemu przedstawiono w specyfikacji implementacyjnej.

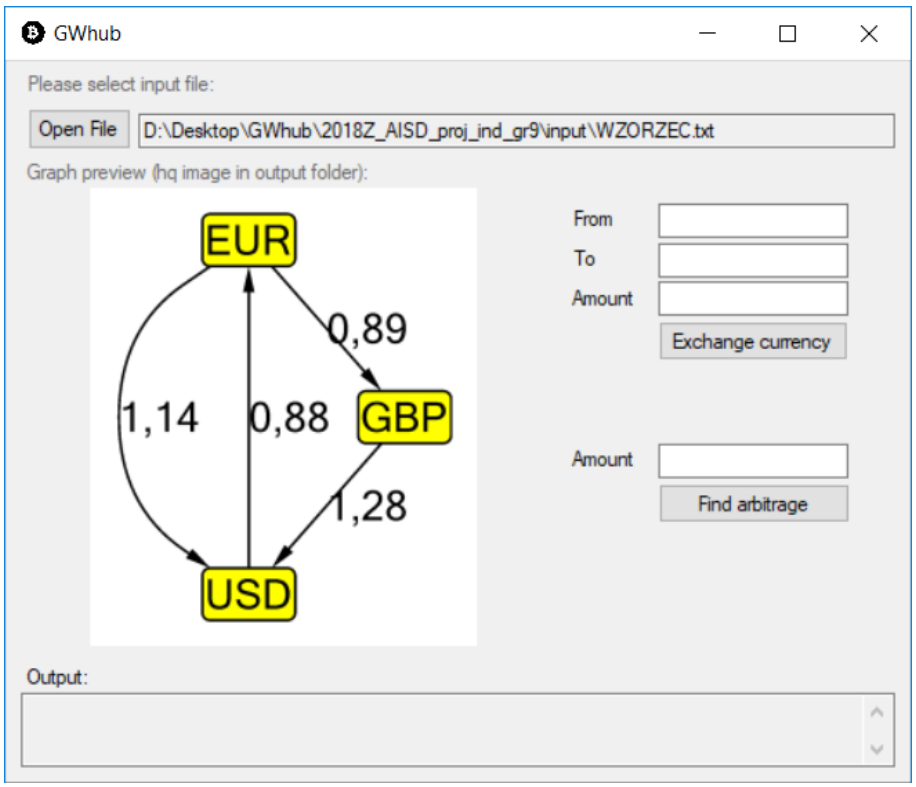
2 Efekty działania programu

W tej sekcji przyjrzymy się zarówno zachowaniu programu w przypadku podania przez użytkownika bezbłędnych danych wejściowych, na podstawie których da się wyświetlić ścieżkę wynikową, a także reakcji programu na otrzymanie danych, na podstawie których niemożliwe jest wyznaczenie ścieżki wynikowej. Sposób edycji danych wejściowych zaprezentowany w specyfikacji funkcjonalnej nie różni się poza jedną, lecz dość istotną różnicą: dane liczbowe - czy to kursy wymiany, czy to wartości opłat - należy podawać stosując jako separator dziesiętny bezwzględnie przecinek. Omówmy więc sposób zachowania programu dla różnych danych wejściowych.

Załóżmy dane wejściowe jak poniżej. Należy zwrócić uwagę, iż linia, w której znajduje się drugi komentarz, czyli druga linia, która rozpoczyna się od znaku #, kończy się dopiero po słowie *opłata*. W podanym przykładzie jedynym powodem, dla którego można byłoby odnieść inne wrażenie, jest fakt, że szerokość strony dokumentu jest ograniczona.

 # Waluty (id | symbol | pełna nazwa)
0 EUR euro
1 GBP funty brytyjski
2 USD dolar amerykański
 # Kursy walut (id | symbol (waluta wejściowa) | symbol (waluta wyjściowa) | kurs | typ opłaty | opłata
0 EUR GBP 0,8889 PROC 0,0001
1 GBP USD 1,2795 PROC 0
2 EUR USD 1,137 STAŁA 0,025
3 USD EUR 0,8795 STAŁA 0,01

W przypadku takich danych program stwierdzi, iż są one poprawne - zostanie utworzony odpowiadający tym danym obraz. Użytkownik, po wybraniu pliku tekstowego (z zawartością taką jak powyższy przykład) poprzez kliknięcie przycisku *Open File* oraz wybranie odpowiedniego pliku w eksploratorze, otrzyma okno programu takie jak przedstawia *Rysunek 1.*



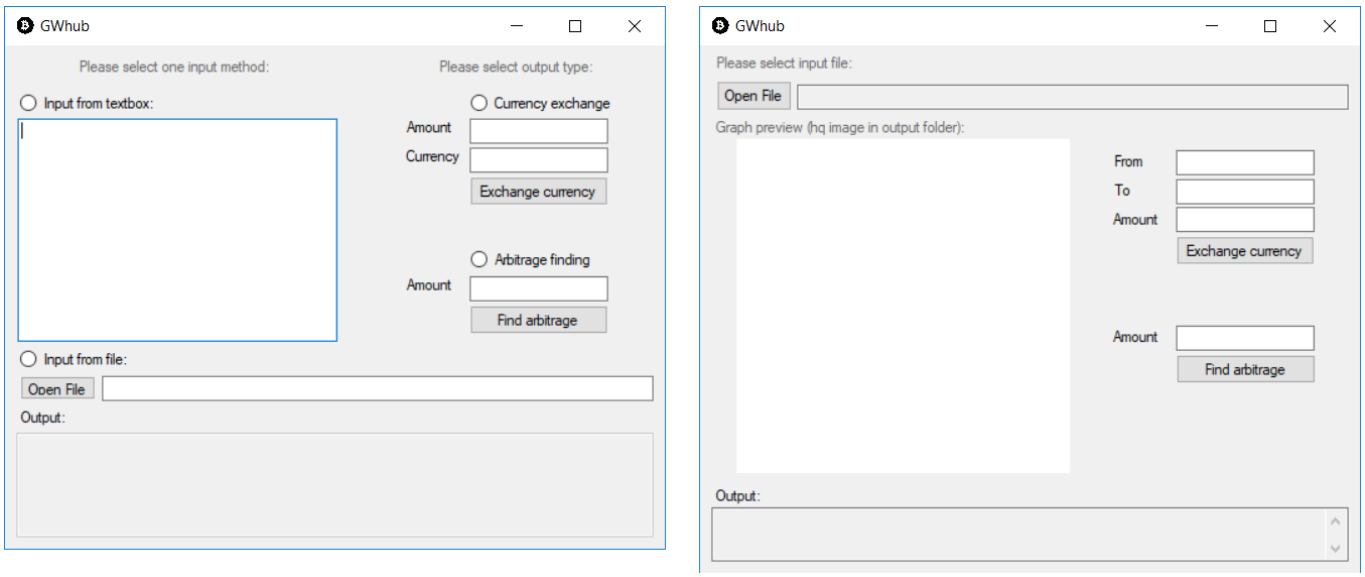
Rysunek 1: Okno główne programu po wprowadzeniu poprawnych danych wejściowych

3 Zmiany względem specyfikacji

3.1 Zmiany względem specyfikacji funkcjonalnej

Nie powstało wiele różnic pomiędzy założeniami specyfikacji funkcjonalnej a faktyczną realizacją projektu. Nieco inaczej wygląda okno główne programu, przy czym drobne zmiany były przewidywane, stąd też wygląd okna głównego w specyfikacji funkcjonalnej został przedstawiony jako prototyp. Na *Rysunku 1* przedstawiono porównanie wersji okna. Najbardziej istotną zmianą jest zrezygnowanie z możliwości wprowadzania danych wejściowych poprzez pole tekstowe będące częścią okna głównego. W zamian w tym samym miejscu umieszczono pole, w którym wyświetlana może być graficzna reprezentacja wczytanego grafu.

Pozostałe zmiany dotyczące wyglądu graficznego interfejsu użytkownika programu są kosmetyczne. Dodano pole, w które należy wpisać walutę początkową przy wybraniu opcji konwersji walut. Pole, w którym pojawiają się dane wyjściowe, zostało wzbogacone o pasek boczny, który umożliwiłby odczytanie całego wyniku działania programu, gdyby ten nie zmieścił się w polu. Zrezygnowano z jakichkolwiek zsynchronizowanych przycisków (jak te przy opcji wymiany walut oraz opcji arbitrażu w prototypie okna głównego), gdyż zastosowanie takiego rozwiązania wydaje się być bezzasadne. Poszczególne komponenty mogły zostać przemieszczone w obrębie okna.



Rysunek 2: Porównanie okna głównego programu w wersji prototypowej oraz ostatecznej

Zaszły liczne zmiany w stosunku do *specyfikacji funkcjonalnej*, jeśli chodzi o sposób obsługi sytuacji wyjątkowych. Jakiekolwiek komunikaty o błędach, czy to związanych z wadliwym plikiem wejściowym, czy też z niepoprawnymi wartościami wpisywanymi w pola graficznego interfejsu, zdecydowano się wyświetlać w tym samym polu, w którym wyświetlane są dane wynikowe. Wszystkie przewidziane w *specyfikacji funkcjonalnej* sytuacje wyjątkowe udało się obsłużyć, generując inne komunikaty, lecz tożsame w sensie znaczeniowym. Dodano więcej przewidzianych sytuacji wyjątkowych, a wszystkie zostały krótko omówione na przykładzie w sekcji *Efekty działania programu*.

3.2 Zmiany względem specyfikacji implementacyjnej

Zdecydowanie więcej różnic niż pomiędzy wersją ostateczną projektu a założeniami specyfikacji funkcjonalnej można odnaleźć pomiędzy ostateczną wersją programu a założeniami specyfikacji implementacyjnej.

Klasę *Visionary*, która pierwotnie miała zawierać metodę odnajdującą arbitraż, oraz metodą odnajdującą najkorzystniejszą ścieżkę pomiędzy danymi walutami, zdecydowano się podzielić na dwie klasy, obarczone węższym zakresem odpowiedzialności. Idąc tym tropem, klasa *Arbitrage* umożliwia odnalezienie ścieżki arbitrażu, a klasa *Best Exchange* wspiera wyszukiwanie najkorzystniejszej ścieżki wymiany. Obie te klasy posiadają swoje metody do wygenerowania napisu będącego efektem wyszukiwania - sklejane są symbole kolejnych walut, a także na początku dodawana jest wartość pieniężna w walucie początkowej, a na końcu wartość pieniężna w walucie końcowej.

W klasie *Parser* zrezygnowano z wykorzystania typu wyliczeniowego przy rozróżnianiu rodzaju danej linii. Ze względu na nieskomplikowaną strukturę plików wejściowych, zdecydowano się problem rodzaju linii rozwiązać znacznie prościej, wykorzystują fakt, iż linie pomiędzy pierwszym a drugim komentarzem to linie przedstawiające waluty, a linie po drugim komentarzu przedstawiają połączenia między tymi walutami - kursy wymiany. Sama metoda wczytująca dane wejściowe w pierwotnej wersji nie zwracała danych, natomiast w wersji ostatecznej zwraca obiekt typu *Digraph*. Obiekt ten jest reprezentacją grafu skierowanego.

Wspomniana powyżej klasa *Digraph* jest odpowiednikiem klasy *Graph* ze specyfikacji funkcjonalnej. Pierwotna nazwa klasy była identyczna jak nazwa klasy reprezentującej graf w bibliotece wspomagające rysowanie grafu, stąd też zmiana. W celu umożliwienia rysowania grafu, do

klasy reprezentującej graf skierowany w omawianym projekcie zostały dodane metody: *Draw*, *AddEdges*, *AddNodes* oraz *SaveGraphAsImg*. Dzięki zaimportowaniu *Microsoft.Msagl.Drawing* oraz *Microsoft.Msagl.GraphViewerGdi* tworzenie graficznych reprezentacji grafów nie zajęło ani dużo czasu, ani nie zajmuje wielu linijek w kodzie.

4 Wnioski

1. Dodanie do programu interfejsu graficznego można uznać za udany eksperyment. Z pewnością takie rozwiązanie dla wielu użytkowników końcowych byłoby wygodniejsze niż konieczność posługiwania się oknem konsoli.

2. Przygotowanie specyfikacji funkcjonalnej oraz specyfikacji implementacyjnej przed właściwym tworzeniem kodu aplikacji znacznie usprawniło proces pisania kodu.

3. Pomimo wielu nieznacznych różnic obecnej implementacji z założeniami specyfikacji implementacyjnej, udało się podtrzymać wszystkie główne idee z obu specyfikacji.

4. Kosztem stosunkowo niewielkiego wysiłku program można rozszerzyć o funkcję zaznaczania na podglądzie grafu (tym samym także na obrazach znajdujących się w folderze z obrazami wynikowymi) wierzchołków lub krawędzi obecnych w wynikowej ścieżce - czy to symbolizującej wierzchołki arbitrażu, czy to kolejne wierzchołki biorące udział w wymianie.

5. Dzięki dodaniu funkcji graficznego przedstawienia danych wejściowych można szybko upewnić się, czy nie popełniło się błędu przy wprowadzaniu danych