Dokumentacja końcowa projektu z ZPR

Temat projektu:	*Losowe przeszukiwanie stanów*
Prowadzący projekt:	*dr inż. Robert Nowak*
Zespół projektowy:	*Piotr Krysik* *Kamil Zabielski*

1 Opis projektu

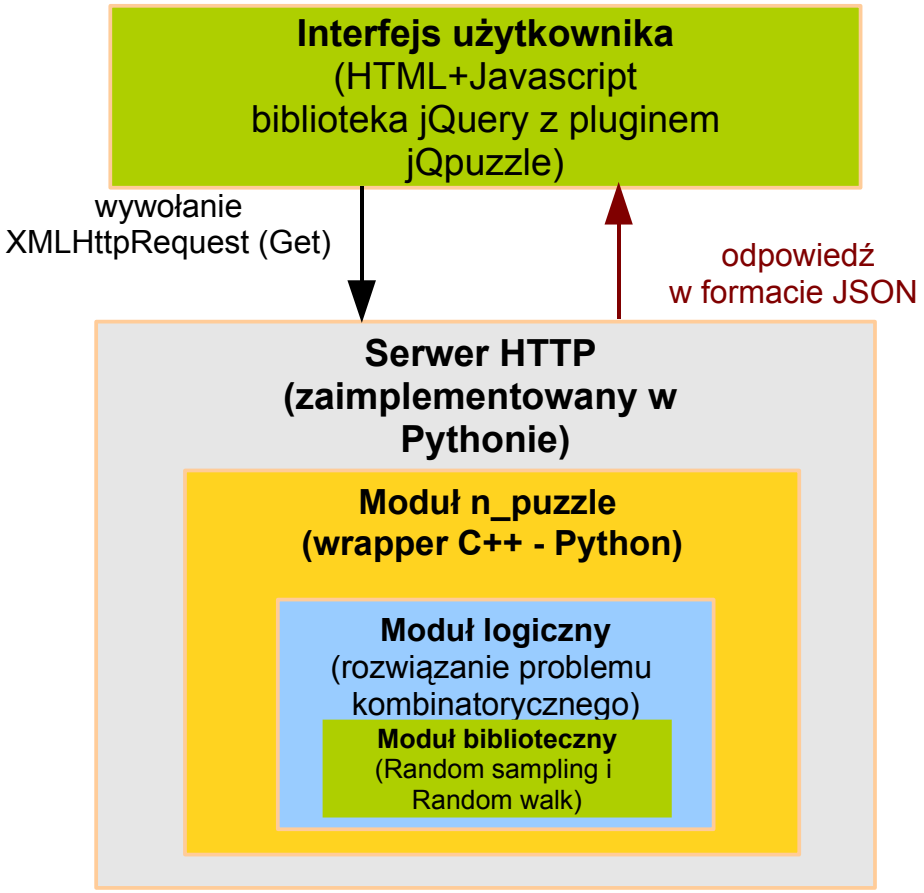
Celem projektu było zaimplementowanie algorytmów błądzenia losowego oraz próbkowania losowego. Działanie tych algorytmów zaprezentowane zostało na przykładzie problemu znajdywania rozwiązywania n-puzzli. Interakcja z użytkownikiem odbywa się poprzez aplikację internetową.

Pełna dokumentacja implementacji biblioteki jest dostępna w folderze *doc*.

2 Aplikacja internetowa - n-puzzle

Rozwiązanie problemu n-puzzli przy użyciu algorytmu błądzenia losowego (random walk) zostało wykorzystane do zbudowania aplikacji internetowej. Działanie tej aplikacji polega na znajdowaniu rozwiązania dla zadanego przez użytkownika ustawienia puzzli. Na rysunku 1 przedstawiony został schemat aplikacji. Składa się ona z:

- klienta stanowiącego interfejs użytkownika uruchamiany w przeglądarce
- serwera HTTP zaimplementowanego w języku python
- modułu Pythona n_puzzle zaimplementowanego w języku C++, w którego skład wchodzi rozwiązanie problemu n-puzzli oraz biblioteka zawierająca algorytmy random walk i random sampling



Rysunek 1: Schemat pokazujący działanie aplikacji

2.1 Opis Serwera

Serwer został zaimplementowany w całości w języku Python przy wykorzystaniu modułów BaseHTTPRequestHandler i HTTPServer. Logika aplikacji została zawarta w klasie

PuzzleServHandler obsługującej żądania do serwera i dziedziczącej po BaseHTTPRequestHandler.

Zadaniem serwera jest dostarczanie przeglądarce plików zawierających opis strony www oraz kod javascript aplikacji użytkownika. Serwer obsługuje też asynchroniczne wywołanie XMLHttpRequest generowane po stronie przeglądarki będące żądaniem rozwiązania układanki o zadanym rozmiarze (wymiarze boku układanki) oraz ustawieniu. Te parametry serwer przekazuje do funkcji obliczającej rozwiązanie. Wyniki obliczeń są zapisywane w plikach w katalogu webapp/results/. Pliki te składają się ze strzałek oznaczających kolejne przesunięcia pustego elementu układanki, które prowadzą do rozwiązania.

Serwer jest uruchamiany przez skrypt run_server.sh (run_server.bat w systemie Windows).

Po uruchomieniu serwer nasłuchuje na porcie 8080 w oczekiwaniu na wywołania ze strony aplikacji klienckich.

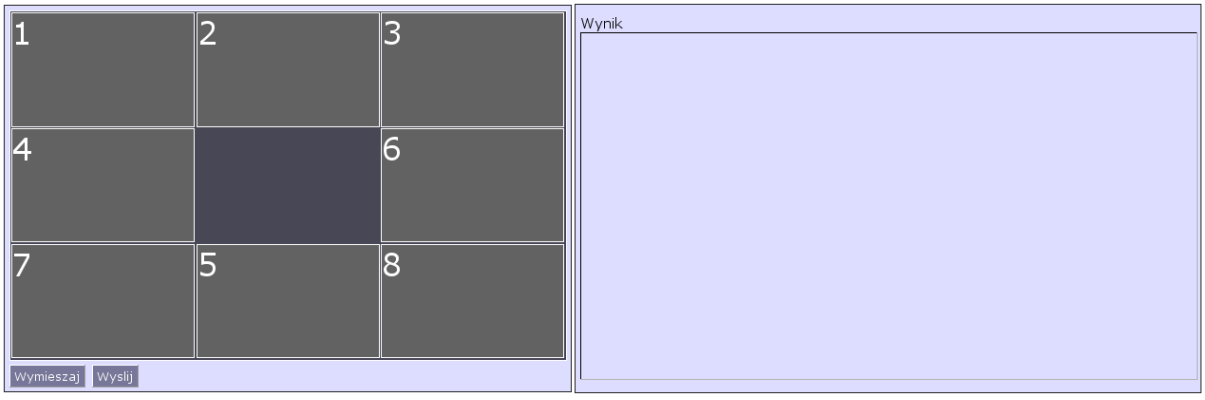
2.2 Opis interfejsu użytkownika

Aplikacja kliencka została zaimplementowana jako strona www w znacznym stopniu wykorzystująca program napisany w języku javascript. Program ten korzysta z biblioteki jQuery wraz ze zmodyfikowanym przez nas komponentem jqPuzzle. JqPuzzle odpowiada za tworzenie planszy z puzzlami oraz pozwala na zmianę przez użytkownika ustawienia puzzli. Wprowadzone przez nas zmiany polegają na:

- dodaniu możliwości odczytania ustawienia puzzli,
- dostarczeniu wywołania wysyłającego do serwera rozmiar i ustawienie puzzli,
- dołączeniu do układanki przycisku wywołującego wysłanie tego wywołania do serwera,
- umożliwieniu prezentacji wyniku.

2.3 Przykładowa sesja

Komunikacja klienta z serwem zostanie przedstawiona na przykładzie. Po uruchomieniu serwer nasłuchuje na porcie 8080. Użytkownik uruchamia aplikację kliencką wchodząc na stronę http://<adres_serwer>:8080 (w sytuacji gdy serwer uruchomiony jest na tym samym komputerze: http://localhost:8080).



Rysunek 2: Interfejs aplikacji klienckiej

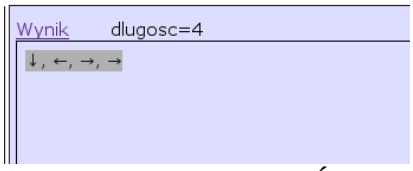
Na rysunku 1 przedstawiony jest interfejs programu wraz z przykładowym ustawieniem puzzli. Po wciśnięciu przycisku "Wyslij" do serwera zostanie wysłane żądanie metodą GET o następującej treści:

```
GET /query?x=1&x=2&x=3&x=4&x=9&x=6&x=7&x=5&x=8&size=3
```

Serwer przetworzy odebrany URL na listę *puzzle_arr = [1,2,3, 4,9,6, 7,5,8]* i rozmiar *puzzle_size = 3*. Liczba 9 reprezentuje pusty element. Te dane zostaną przesłane do obiektu typu *n_puzzle* a uzyskany rezultat zostanie zapisany w pliku out1.txt w katalogu webapp/results ("1" w nazwie oznacza pierwsze wywołanie). Do klienta zostanie odesłana w formacie JSON długość ścieżki prowadzącej do rozwiązania oraz nazwa pliku zawierającego rozwiązanie:

```
{"result_len": 4, "result_fname": "out1.txt"}
```

Po odebraniu tych informacji aplikacja kliencka pobierze i wyświetli plik z rozwiązaniem w ramce znajdującej się obok układanki (rys. 3).



Rysunek 3: Ścieżka prowadząca do rozwiązania

Strzałki oznaczają kolejne przesunięcia pustego elementu, które prowadzą do rozwiązania.

2.4 Opis modułu logicznego

Ten w tym module odbywa się rozwiązanie problemu kombinatorycznego czyli faktyczne generowanie odpowiedzi (ścieżki) metodą losową dla zadanego ustawienia planszy wysłanego przez serwer http. Cały algorytm wytyczania ścieżki znajduje się w klasie *puzzle_solution*, a dokładniej w metodzie *solve_puzzle*(…) .

W skład tego modułu wchodzą pliki *n_puzzle_solution.cpp* oraz *n_puzzle_solution.hpp*.

Więcej informacji można uzyskać w dokumentacji kodu.

2.5 Opis modułu bibliotecznego

Moduł dostarcza klas implementujących algorytmy błądzenia przypadkowego oraz próbkowania losowego: *RandomWalk*(plik *RandomWalk.hpp*) i *RandomSampling* (plik *RandomSamoling.hpp*) . Są one zaimplementowane jako szablony które wystarczy dołączyć do plików korzystających z nich poprzez *#include.*

Więcej informacji można uzyskać w dokumentacji kodu.

3 Kompilacja

Projekt do poprawnej kompilacji wymaga zainstalowania biblioteki boost(wersja przynajmniej 1.38) oraz biblioteki pyton (przynajmniej w wersji 2.6). Ponad to pod Linuxem do kompilacji używamy programu Scons oraz kompilatora G++, natomiast pod Windows'em środowiska programistycznego Visual Studio(przynajmniej wersja z roku 2005).

3.1 Windows

Pod systemem Windows do kompilacji trzeba użyć środowiska programistycznego Visual Studio 2005 albo nowszej. Uruchamiamy środowisko i ładujemy ustawienia projeku przez opcję *open project* i wyszukujemy plik *random.sln*. Po załadowaniu projektu trzeba jeszcze ustawić ścieżki do plików *.hpp i *.h, przechodzimy do *Project->properties-*

>configuration properties->C/ C++/General, w polu *Additional Includes Directories* dodajemy ścieżki do boost(np. C:\Program Files\boost_1_38\) i pyton(np. C:\Python28\include). Jeszcze tylko pozostało ustawić ścieżki do bibliotek potrzebych do kompilacji: , przechodzimy do *Project->properties->configuration properties->linker->general* w polu *Additional Library Directories* dodajemy ściezki do plików *.libboost np. " C:\Program Files\boost_1_38\stage\lib" oraz do bibliotek phyton'a (np. C:\Python31\libs)

UWAGA: Należy mieć na uwadze ze project korzysta z boost::python w związku z czym trzeba mieć bibliotekę statyczną *libboost_python.lib* skompilowaną. Zdarza się że *bjam* nie skompiluje tej części biblioteki boost, w związku z czym trzeba ręcznie skonfigurować i user_config.jam

3.2 Linux

Pod Linuxem wystarczy przejść do folderu w którym znajduje się project i wywołać polecenie *scons* z konsoli. Ewentualnie będzie potrzebna modyfikacja ścieżki do plików nagłówkowych *python'a*

4 Wnioski i uwagi.

4.1 Wielowątkowość

Zarówno moduł logiczny jak i moduł biblioteczny mogą być uruchamiane w środowisku współbieżnym. Można stworzyć osobny obiekt dla każdego wątku i nie będą zachodziły zjawiska niepożądane (wyścigi, dereferencja wskaźnika), dopóki jest zapewniony odpowiedni czas życia obiektów. Zapewnione jest to poprzez użycie generatorów liczb pseudolosowych z biblioteki *boost::random*. Ponadto do klas *puzzle_solution* jak i *RandomSampling* jest również prawdzie stwierdzenie iż wiele wątków może korzystać z tych samych obiektów tych klas. Ponieważ obiekty klasy *RandomWalk* przechowują obiekty z których losują użycie jednego obiektu w wielu wątkach jest dosyć niebezpieczne.

4.2 Wydajność

Ponieważ np macierz 3x3 ma 9!(ok. 350.tyś)możliwych kombinacji a macierz 4x4 to już 16! (ok. 20*10^12), wytyczanie ścieżki może być trwać dosyć długo(możemy oczekiwać ścieżek nawet na poziomie n=9! dla macierzy 3x3). Dlatego też moduł logiczny oraz moduł biblioteczny zostały zaimplementowane w sposób prosty ale wydajny.

4.3 Poprawność rozwiązania

Jak się okazuje dla macierzy 4x4 i większych stany można podzielić na 2 grupy. Od jednego stanu do innego da się przejść tylko wtedy gdy należą do tej samej grupy. Wiec z części stanów nie da się dojść do stanu końcowego(uporządkowanego.)