Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Zaawansowane programowanie w C++

Dokumentacja projektu CoreWars

Kamil Gabryjelski, Antoni Różański

Prowadzący: Konrad Grochowski

Warszawa, 2017

1. Dokumentacja użytkownika

Celem gry jest napisanie programu komputerowego w języku RedCode, który walczyć będzie o zasoby w symulowanym środowisku z programem napisanym przez innego gracza. Zasobem jest pamięć podzielona na 400 komórek. W każdej komórce mieści się maksymalnie jedna instrukcja programu. Zwycięża gracz, którego program wyeliminuje przeciwnika lub zajmie całą dostępną pamięć. Dostępne instrukcje języka RedCode:
— DAT - zabija proces
— MOV - przenosi dane z jednego adresu na drugi
— ADD - dodaje 2 liczby
— JMP - kontynuuje wykonanie programu od danego adresu
Możliwe jest komentowanie kodu poprzez rozpoczęcie komentowanej linii znakiem ";". Wspierane są tryby adresowania natychmiastowy(immediate), pośredni(direct) i bezpośredni(indirect). Szczegółowe informacje, samouczek i przykładowe programy języka RedCode dostępne są na stronie internetowej.

Przed uruchomieniem wizualizacji gry w przeglądarce należy włączyć serwer oraz klienta C++. Po rozpoczęciu gry dostępne są pola tekstowe, w które należy wpisać kod programu, nazwę programu i nazwę gracza a nastepnie wcisnąć przycisk "Send code". Jeśli wystąpią błędy składniowe w programie gracza, zostanie wyświetlony komunikat zawierający informację o linijce, w której pojawił się błąd. W przeciwnym wypadku kod zostanie przyjęty, a gra poprosi o wpisanie kodu drugiego gracza. Jeżeli kod zostanie wczytany pomyślnie, rozpocznie się symulacja "walki" między dwoma programami, która zostanie zwizualizowana w postaci kolorowej tablicy. Każda komórka tablicy oznacza adres pamięci, a kolor symbolizuje instrukcję, która zapisana jest w danej komórce.

Poszczególne kolory w wizualizacji odpowiadają kolejno:
— biały - proces gracza 1
— czarny - proces gracza 2
— niebieski - instrukcja DAT
— czerwony - instrukcja MOV

2. Funkcjonalności

Udało nam się zrealizować następujące funkcjonalności:
— pisanie kodu wojownika z poziomu przeglądarki
— wczytywanie wojownika z pliku
— analiza składniowa wprowadzonego programu
— wizualizacja przebiegu gry i statystyk
Nie udało nam się zrealizować następujących funkcjonalności:
— integracja z bazą danych
— zmiana ustawień gry (takich jak prędkość symulacji) z poziomu przeglądarki

3. Technologie

W celu realizacji połączenia między serwerem, klientem C++, a wizualizacją w JavaScript, użyliśmy frameworka Apache Thrift. Do napisania logiki aplikacji użyty został standard C++14.

Do przeprowadzenia testów użyty został framework Catch (nie trzeba go instalować, mieści się w jednym pliku nagłówkowym).

Przy tworzeniu aplikacji użyliśmy systemu kontroli wersji git i serwisu GitHub (repozytorium).

4. Uruchamianie i kompilacja

Aby możliwe było skompilowanie aplikacji, niezbędne są skompilowane biblioteki Boost (używana przez nas wersja to 1.64). Ponadto niezbędne jest posiadanie skompilowanego frameworka Apache Thrift, którego instrukcja instalacji dostępna jest na stronie internetowej. Używanym przez nas systemem budowania jest SCons, do którego działania konieczny jest interpreter Pythona w wersji 2.7. Kod serwera napisany jest w języku C++14, dlatego należy użyć kompilatora wspierającego ten standard (na systemie Linux jest to kompilator g++ w wersji co najmniej 4.9).

Aby uruchomić aplikację, najpierw należy uruchomić serwer (Server), następnie klienta C++ (CoreWars), a na końcu wizualizację w przeglądarce (jsClient.html). Prosimy uruchamiać jsClient.html w przeglądarce **Mozilla Firefox**, gdyż z nieznanych nam powodów na innych przeglądarkach aplikacja nie działa tak, jak powinna.

Testy uruchamiane są z pliku wykonywalnego tests.

5. Informacje o kodzie

W tabeli 5.1 został przedstawiony udział języków w projekcie. W obliczaniu linii kodu nie zostały uwzględnione pliki wygenerowane przez kompilator thrift.

5.1. Linie kodu

Język	Liczba linii kodu
C++	2257
C++ testy	1304
JavaScript	141

Tab. 5.1: Liczba linii kodu poszczególnych języków

5.2. Pokrycie testami

Przeprowadzane zostają 23 testy, zawierające 235 asercji.

6. Napotkane problemy

Podczas tworzenia projektu napotkaliśmy na szereg problemów. Jednym z poważniejszych była skąpa dokumentacja frameworku Apache Thrift na C++ i JavaScript, w wyniku czego wiele czasu poświęciliśmy na szukanie rozwiązań metodą prób i błędów. Okazało się również, że Thrift domyślnie przekazuje zapytania między klientem JavaScript a serwerem w trybie synchronicznym. Nie udało nam się ustawić trybu asynchronicznego, w wyniku czego podczas symulacji walki między programami przeglądarka nie jest responsywna. Ponadto problematyczna okazała się konfiguracja Apache Thrifta na systemie Windows. Doprowadziliśmy do jego działania, choć zajęło nam to wiele godzin. Kolejnym problemem jest to, że klient JavaScript działa poprawnie jedynie na przeglądarce Mozilla Firefox, czego przyczyny nie znaleźliśmy (choć może to wynikać z naszego braku wiedzy o front-endzie).

Jednym z naszych podstawowych błędów było niedokładnie przeprowadzona faza planowania, co wynikało z braku doświadczenia. Skutkiem tego były zmiany w strukturze kodu w trakcie trwania projektu, co doprowadzało do przedłużania i opóźniania pracy. Kolejnym poważnym błędem było zbyt późne rozpoczęcie pracy nad projektem, skutkiem czego był pośpiech i brak realizacji części założeń.

Każdy z nas spędził nad projektem około 100 godzin. Sądzimy, że posiadając doświadczenie zdobyte w trakcie tego projektu, czas ten wykorzystalibyśmy znacznie bardziej produktywnie i zdążylibyśmy zaimplementować dodatkowe funkcjonalności, takie jak połączenie z bazą danych lub implementacja większej ilości komend języka RedCode.