Tomasz Bocheński
Tomasz Dakowicz
Marcin Paszkiewicz

2.06.2015 r.

Prowadzący: dr inż. Robert Nowak

# Zaawansowane programowanie w C++
# Poker z interfejsem na przeglądarkę
# Dokumentacja

## *Temat projektu*

Celem projektu było napisanie przeglądarkowej gry wieloosobowej – Poker Texas Hold'em

## *Zrealizowane funkcjonalności:*

W całości zrealizowaliśmy wszystkie podstawowe funkcjonalności, a także dodaliśmy kilka dodatkowych podwyższających atrakcyjność gry.

Gra zawiera:

- wiele stołów przy których może się odbywać gra – są one tworzone i usuwane wraz za zmianą ilości graczy
- możliwość gry z więcej niż jednym przeciwnikiem – maksymalna liczba graczy przy jednym stole wynosi 6
- kalkulator prawdopodobieństwa wygranej
- chat

## *Przebieg gry:*

Nowy gracz jest automatycznie dobierany do stołu, na którym rozgrywka jeszcze się nie rozpoczęła. Gra rozpoczyna się w chwili, gdy przy

stole znajduje się co najmniej dwóch graczy i każdy z nich potwierdzi gotowość. Każdy z graczy rozpoczyna grę z taką samą pulą żetonów. Gra przebiega zgodnie z zasadami gry Poker Texas Hold'em, bez górnego limitu podbicia, a także bez ograniczenia co do liczby podbić podczas jednej rundy licytacji (tzn. no limit Texas Hold'em). Wraz z postępem gry rosną stawki w ciemno. Gracze odpadają z gry, gdy skończą się im żetony. Gra kończy się, gdy przy stole zostanie jeden gracz. W dowolnej chwili można opuścić grę, serwer dba o to by gracz został usunięty z rozgrywki w odpowiednim momencie - na koniec rozdania, do tej chwili symuluje rezygnację z udziału w licytacji.

### *Architektura i implementacja:*

Zastosowaliśmy architekturę klient-serwer. Do implementacji serwera wykorzystaliśmy język Python. Serwer odpowiada za komunikację pomiędzy graczami, a także jest odpowiedzialny za logikę gry. Kosztowne obliczeniowo operacje zostały zaimplementowane w C++ i połączone z Pythonem przy użyciu biblioteki boost::python. Operacjami tymi są porównywanie układów kart oraz kalkulator prawdopodobieństwa wygranej.

Przy tworzeniu kalkulatora braliśmy pod uwagę wiele różnych możliwości. Mogliśmy zaimplementować kalkulator, który sprawdza wszystkie możliwe układy i w ten sposób określa prawdopodobieństwo wygranej, przegranej i remisu. Kolejnym sposobem było stworzenie bazy, w której przechowywane byłyby wszystkie, już obliczone, prawdopodobieństwa. Jednak oba te sposoby miały swoje wady. Wyznaczanie wszystkich możliwych przykładów zajmowałoby zbyt dużo czasu. Tworzenie bazy wymagałoby zapisania wszystkich możliwych prawdopodobieństw.

Zdecydowaliśmy się zastosować Metodę Monte Carlo. Tworzymy n losowych układów i porównujemy je z kartami gracza. Określamy relacje między nimi i na tej podstawie wyznaczamy poszczególne prawdopodobieństwa. Metoda ta działa poprawnie. Dla n = 10000

poszczególne wyniki różnią się o mniej niż 0.5%.

W serwerze zminimalizowaliśmy ilość wątków do minimum, co praktycznie wyeliminowało problemy z synchronizacją. Jeden z wątków odpowiada za nasłuchiwanie na nowych graczy. Jeśli takowy się pojawi, uruchamiany jest wątek który akceptuje gracza i pobiera od niego nick. Jeśli cały proces przebiegł poprawnie, gracz jest dodawany do stołu. Pozostałe to wątki związane ze stołami i toczącymi się na nich grami, które w pętli odczytują kolejno wejście od każdego z graczy. Aby gra toczyła się płynnie, czas na podjęcie decyzji jest ograniczony.

Do komunikacji między klientem a serwerem wykorzystaliśmy WebSockety oraz obiekty JSON.

Graficzny interfejs został zrealizowany przy pomocy HTML5, CSS3 oraz JavaScipt. Został oparty o popularny framework AngularJS, który między innymi zapewnia zachowanie modelu MVC. Widok wysyła i odbiera dane od serwera, które następnie aktualizuje i odpowiednio wyświetla. Interfejs użytkownika składa się z ze stołu (z miejscami na 6 graczy), panelu z buttonami do obsługi gry oraz panelu z 'siłą ręki' gracza. W zależności od akcji, interfejs zmienia się w czasie rzeczywistym. Schemat w CSS został rozplanowany na podstawie przeglądarki w rozdzielczości 1920x1080, a więc zalecane jest używanie tej rozdzielczości. W innych również wygląda dobrze, grafiki są w wysokiej rozdzielczości. Poza tym, zostały zastosowane elementy z Bootstrapa.

Walidacja przycisków w trakcie tury użytkownika (co objawia się żółtym elementem obok kart gracza) jest w pełni zastosowana (niepoprawny input powoduje wyrzucenie z rozgrywki). Fioletowy znacznik w trakcie rozgrywki oznacza, że dany gracz spasował w danej turze.

W trakcie realizacji projektu nacisk położyliśmy na organizację i jakość kodu. Naszym zamierzeniem było stworzenie aplikacji o jak największej modularności i elastyczności przyjętych rozwiązań. W tym celu zastosowaliśmy powszechnie stosowane techniki i wzorce projektowe.

Projekt został podzielony na moduły. Kod został napisany zgodnie z powszechnie przyjętymi zasadami i właściwie okomentowany. Wykorzystaliśmy wzorzec MVC (model – view – controller). Widok odpowiada za reprezentację graficzną rozgrywki - wizualizacje modelu i odbieranie zdarzeń od użytkownika (aplikacja kliencka). Model realizuje logikę gry, a kontroler odpowiada za interpretację zdarzeń odbieranych od użytkownika (serwer). Przy implementacji modelu wykorzystaliśmy wzorzec obserwatora. Stół na którym toczy się gra jest obiektem obserwowanym, a gracze są obserwatorami. Jakakolwiek zmiana stanu gry powoduje, wysyłanie powiadomienia wszystkim zainteresowanym graczom.


***Stan aplikacji:***

Aplikacja jest w pełni funkcjonalna. Zadbaliśmy o to by struktura projektu była prosta i przejrzysta. Wykorzystaliśmy popularne wzorce oraz stosowaliśmy się do zasad czystego kodu. Skutkiem czego, projekt mógłby być z łatwością rozwijany przez inny zespół programistów. Aplikacja jest przenośna. Została przetestowana na najpopularniejszych przeglądarkach (Firefox, Chrome) i działa zarówno pod systemem Linux jak i Windows.

Projekt okazał się być bardzo ciekawy i pouczający. Udało nam się zrealizować niezawodną aplikację z prostym i przyjaznym interfejsem graficznym.

### *Kompilacja i uruchomienie:*

Kompilacji podlega tylko część projektu zaimplementowana w C++. Do projektu został dołączony odpowiedni *makefile* Serwer uruchamia skrypt */core/run_server.py,* a aplikacje kliencką */app/index.html.*

### *Napotkane problemy i popełnione błędy:*

Najwięcej problemów przysporzyła komunikacja między klientem a serwerem. Wykorzystaliśmy WebSockety i sami zaimplementowaliśmy protokół – handshaking, kodowanie i odkodowanie nagłówków. To rozwiązanie okazało się to pracochłonne oraz trudne do przetestowania i zdebuggowania. Finalnie, udało nam się to wykonać, było to bardzo pouczające, ale na przyszłość będziemy wykorzystywać już gotowe i przetestowane rozwiązania. Poza tym, wiele problemów spowodowała sprawna aktualizacja widoku w cliencie – również i to zostało rozwiązane, dzięki czemu dogłębnie zapoznaliśmy się z działaniem frameworku AngularJS oraz z budową programu client-side.