16.01.2012

Dokumentacja końcowa projektu z przedmiotu ZPR

Piotr Jastrzębski
Łukasz Tomczak

Temat projektu: Gra Warcaby

Zrealizowanie koncepcji i założonej funkcjonalności:

Koncepcja zakładała stworzenie implementacji dwuosobowej gry w warcaby, gdzie rozgrywka odbywałaby się zdalnie(Internet, sieć lokalna), a stan rozgrywki prezentowałby graficzny, okienkowy interfejs. Program został zaimplementowany w technice „grubego klienta" z pomocą C++ i biblioteki Qt, która to przede wszystkim zapewnia graficzny interfejs użytkownika oraz łatwą komunikację między metodami w odpowiedzi na zdarzenia (sygnał – slot). Komunikacja oparta jest na protokole TCP i korzysta z klas QTcpServer i QtcpSocket.

Aplikacja posiada prosty, lecz czytelny interfejs, a zastosowane mechanizmy pozwalają na szybkie uruchamianie serwera na zadanym adresie IP i porcie oraz przyłączanie klienta.

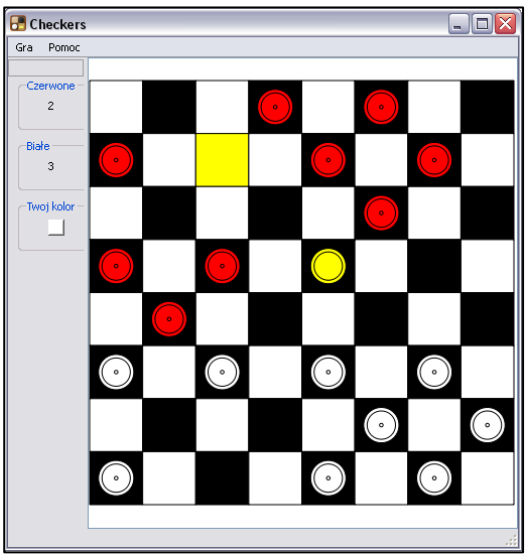
Zaimplementowaliśmy mechanizmy kontroli poprawności m.in. sprawdzanie adresu IP pod kątem poprawności semantycznej, informowania użytkownika o zerwanym połączeniu lub próbie nawiązania komunikacji z nieistniejącym serwerem itp.

Rozrywkę ułatwia system kalkulacji punktów, oznaczania koloru, którym gra gracz oraz wyszarzenie planszy, gdy ruch wykonuje przeciwnik a także oznaczanie dozwolonych pól ruchu.

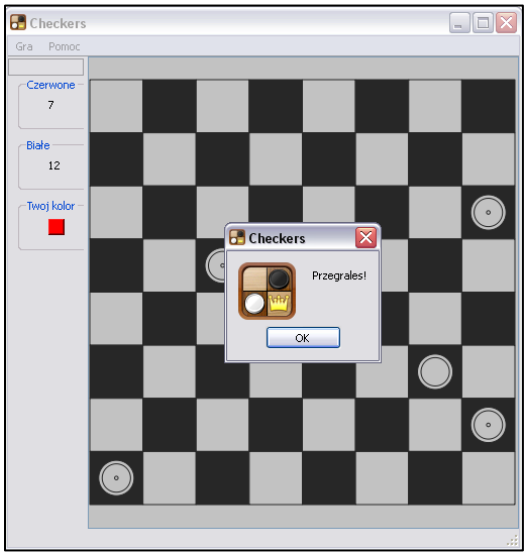
Zasady gry

Sama gra przebiega właściwie zgodnie z zasadami Warcabów Niemieckich "Dame":

- Gra rozgrywana jest na ciemnych polach planszy o rozmiarze 8 × 8 pól.
- Każdy gracz rozpoczyna grę z dwunastoma pionami (jeden koloru białego, drugi - czerwonego).
- Jako pierwszy ruch wykonuje grający pionami białymi, po czym gracze wykonują na zmianę kolejne ruchy.
- Celem gry jest zbicie wszystkich pionów przeciwnika (w tym damek).



- Piony mogą poruszać się o jedno pole do przodu na ukos na wolne pola.
- Bicie pionem następuje przez przeskoczenie sąsiedniego pionu (lub damki) przeciwnika na pole znajdujące się tuż za nim po przekątnej (pole to musi być wolne). Zbite piony są usuwane z planszy po zakończeniu ruchu. Bić można jedynie do przodu i są one nieobowiązkowe.
- W jednym ruchu wolno wykonać więcej niż jedno bicie tym samym pionem, przeskakując przez kolejne piony (damki) przeciwnika.
- Pion, który dojdzie do ostatniego rzędu planszy, staje się damką, wtedy kolej ruchu przypada dla przeciwnika.
- Damki mogą poruszać się w jednym ruchu o dowolną liczbę pól do przodu lub do tyłu po przekątnej, zatrzymując się na wolnych polach.
- Bicie damką jest możliwe z dowolnej odległości po linii przekątnej i następuje przez przeskoczenie pionu (lub damki) przeciwnika, za którym musi znajdować się co najmniej jedno wolne pole – damka przeskakuje na to pole.



Struktura projektu:

Klasy CheckableField i UncheckableField dziedziczące z klasy Field która z kolei jest klasą pochodną od QGraphicsObject, klasy odpowiadają za reprezentację zdarzeń pól na planszy, implementują reakcję na ruch/kliknięcia myszy, oraz to jak mają być malowane.

Klasa Board będąca klasą pochodną QGraphicsScene jest odpowiedzialna za odbieranie zdarzeń od obiektów, i ich przechowywania oraz relacji między nimi. Jest modelem planszy.

Klasa Piece dziedzicząca z klasy QGraphicsItem jest reprezentacją pionka na planszy, implementują reakcję na kliknięcia myszy, oraz to jak ma być malowany.

Klasa NetworkManager oraz jej klasy pochodne NetworkServer i NetworkClient reperezentują obiekty zarządzające połączeniem sieciowe z drugim graczem, pozwalaja na połączenie, prtzesyłanie danych pomiędzy obiema stronami połączenia.

Klasa GameState reprezentująca stan gry i punkty za bicia.

Klasa Test odpowiedzialna jest za implementację testów.

Klasa CreateGameDialog to klasa reprezentują dialog do dołączania / tworzenia nowej gry, pozwala na pobranie wprowadzanych danych, wartości w kontrolkach są walidowane podczas wprowadzania za pomocą obiektów klas QValidator , a dokładnie QIntValidator i QRegExpValidator.

Klasa Move implementuje ruch na planszy ma metody sprawdzające poprawność ruchu, możliwość podwójnych bić itd.

Klasa MainWindow łączy pozostałe elementy ze sobą.

Nietrywialne techniki:

Interfejs graficzny oraz komponenty sieciowe korzystają w głównej mierze z biblioteki Qt. Biblioteka ta dla rozwiązań niekomercyjnych ma otwarto źródłową licencję oraz jest tworzona przez osoby i firmy związane z oprogramowaniem open source co zapewnia szerokie wsparcie oraz obszerne jej udokumentowanie. Dodatkowym atutem jest jej przenośność i zgodność zarówno z kompilatorem g++ i Visual C++.

Do wykonania planszy do gry użyliśmy Graphics View Framework który jest częścią biblioteki odpowiedzialną za grafikę dwuwymiarową. Do tworzenia obiektów planszy użyliśmy klas bazowych Graphics View takich jak: QGraphicsScene do stworzenia modelu planszy, QGraphicsItem do pól planszy oraz pionków oraz GraphicsView jako Widgetu który pozwolił wyświetlić planszę i jej obiekty w głównym oknie programu. Wykorzystaliśmy również wszelkie wbudowane możliwości tych obiektów do śledzenia ruchów i kliknięć myszy w ich obrębie.

Użyliśmy również mechanizmu budowy interfejsu użytkownika za pomocą plików xml (*.ui) który jest tworzony w graficznym kreatorze. Pliki te w procesie budowania są przekształcane automatycznie w odpowiedni kod języka aplikacji. Za pomocą tej techniki jest stworzony dialog dołączania do gry/ tworzenia gry. Mechanizm ten w znaczny sposób upraszcza kod i pozwala na oddzielenie logiki danego okna bądź dialogu od jego wyglądu.

Kolejnym wykorzystanym elementem jest mechanizm sygnałów i slotów Qt będący implementacją wzorca projektowego obserwator. Jako, że wszelkie gry należą do aplikacji interakcyjnych, mechanizm ten okazał się wysoce przydatny w naszej implementacji warcab. Wszelkie zdarzenia pochodzące od gracza który ma możliwość sterowania myszą, zdarzenia sieciowe oraz związane ze stanem rozgrywki wykorzystują ten mechanizm. Wadą tego rozwiązania okazało się częściowe zaciemnienie relacji zachodzących pomiędzy klasami programu.

Do obsługi sieci zostały wykorzystane klasy QTcpServer i QTcpSocket z modułu QtNetwork. Klasa serwera jest wykorzystywana przy tworzeniu rozgrywki i po podłączeniu do rozgrywki drugiego gracza serwer nie jest dłużej używany.

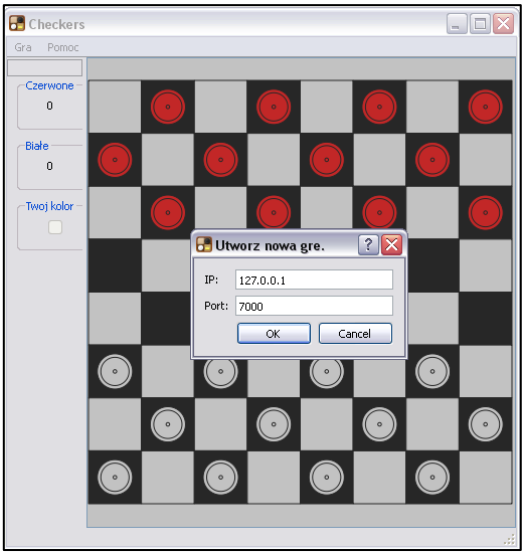
Do budowania projektu użyliśmy również mechanizmu wywodzącego się z Qt tj. qmake, lecz może on być również używany do budowania programów nie korzystających z tej biblioteki. Jego zaletą jest prostota wszystkie opcje plików projektu i dołączonych bibliotek są ustawiane w pliku *.pro na którego podstawie qmake generuje plik makefile. Dodatkowym plusem jest również przenośność na inne platformy.

Testy jednostkowe:

Testy jednostkowe pokrywają zarówno poprawność implementacji co do przyjętych zasad gry jak również zachowanie komunikacji serwer – klient.

- Prawidłowość ruchów pionem (CTManMove) – sprawdza poprawność przejścia na dozwolone pola oraz niemożność wykonania ruchu na inne pola czarne
- Prawidłowość ruchów damą (CTKingMove) – j.w tylko z zachowaniem właściwości damki (długiego skoku)
- Prawidłowość bicia pionem (CTManCaptures) – sprawdza wykonalność bicia przeciwnika i testuje, czy niemożliwe jest zbicie swojej figury
- Prawidłowość bicia damą (CTKingCaptures) – j.w tylko dla damki
- Zamiana z piona na damę (CTMakeKing) – testuje przejście piona w damkę, oraz niemożność powtórnej przemiany

- Wygrana „białych” (CTWhiteWins) – symuluje stan osiągnięcia 12 punktów dla gracza „białego”
- Wygrana „czarnych” (CTBlackWins) – j.w tylko dla gracza „czarnego”
- Uruchomienie serwera (NTStartServer) – uruchamia serwer oraz testuje, czy nasłuchuje
- Podłączenie klienta (NTConnectClient) – próbuje podłączyć klienta do zdalnego serwera
- Wysłanie wiadomości przez klienta ku serwerowi (NTClientSends) – wysyła wiadomość od klienta do serwera
- Wysłanie wiadomości przez serwer do klienta (NTServerSends) – wysyła wiadomość w odwrotnym kierunku: od serwera do klienta



Wcześniejsza implementacja testów pozwoliła szybko i sprawnie testować zaimplementowane rozwiązania i pomagała wyłapywać błędy logiczne gry przy modyfikacjach kodu.

Narzędzia:

W naszym projekcie użyliśmy następujących narzędzi:

- Qt Creator – jako aplikacje do projektowania interfejsu i wygodne, podczas pisania z biblioteką Qt, IDE
- Git – rozproszony system kontroli wersji
- doxygen – tworzenie dokumentacji na podstawie kodu i systemu komentarzy

Budowanie:

Aby zbudować projekt należy uruchomić qmake z odpowiednim plikiem jako parametr w katalogu ze źródłami który wygeneruje odpowiedni plik Makefile. I tak dla wygenerowania konfiguracji testowej wykonujemy qmake Test.pro, a dla użytkowej qmake Checkers.pro, a następnie polecenie make buduje program odpowiednio Test i Checkers.

- aplikacja: qmake Checkers.pro && make && ./Checkers
- testy: qmake Test.pro && make && ./Test

Problemy:

Jedyną niezrealizowaną z początkowo założonych funkcjonalności jest przywracanie rozgrywki po przerwanym połączeniu sieciowym jeżeli drugi z graczy nie zamknął programu.

Nie pozwoliły na to uproszczenia zawarte w formacie przesyłanych prze sieć ruchów (założenie , że przesyłany ruch jest już sprawdzony pod kątem poprawności przed wysłaniem) oraz zdecentralizowany model planszy przechowywany u obu graczy.

Jednym z aspektów które mogłyby być lepiej zaprojektowane, jest model planszy który w naszej implementacji łączy w sobie funkcję widoku modelu i widoku aplikacji, co przy implementacji interakcji z użytkownikiem okazało się sporym ułatwieniem, lecz przy wprowadzaniu bardzo złożonej logiki prowadziło czasem do nieeleganckich rozwiązań.

Sadzimy jednak, że udało nam się zrealizować większość założeń projektowych i otrzymaliśmy funkcjonalną implementację gry w warcaby.