

SPRAWOZDANIE

Dotyczące projektu zaliczeniowego z przedmiotu „Sieci Komputerowe II Laboratoria”,

autorstwa Witold Janika, nr indeksu 106637.

1. Treść zadania

- a. Jako temat projektu zaliczeniowego wybrany został temat pod nr 1, pt. „Sieciowa turowa gra logiczna, np.: reversi, szachy, warcaby itp. (1 osoba).”
- b. Wybraną przeze mnie do realizacji grą logiczną zostały warcaby.

2. Przyjęta metoda rozwiązania problemu + opis protokołu komunikacyjnego.

- a. Jako rozwiązanie problemu zaprojektowano i zaimplementowano dwie aplikacje w języku C++, oparte o technologię QT 5, działające w architekturze „klient-serwer”. Nie ma możliwości komunikowania się klientów „peer-to-peer” z pominięciem serwera.
- b. Stworzono: aplikację kliencką [nazwa w projekcie: „Client”] oraz serwer [nazwa w projekcie: „Server”]. Aplikacje umieszczone zostały w repozytorium git na stronie github.com, jako repozytorium prywatne. Zostały również przesłane prowadzącemu jako archiwum rar zawierające kody źródłowe oraz pochodne pliki projektowe stworzone przez aplikację „QT Creator”.
- c. Do komunikacji przez sieć aplikacje używają rozwiązania opartego na modelu TCP/IP, zrealizowanego na poziomie „warstwy aplikacji” przy pomocy mechanizmu gniazd (ang. „sockets”). Wszystkie „niższe” warstwy komunikacji sieciowej obsługiwane są „transparentnie dla programisty”, tj. zajmuje się nimi np. system operacyjny, w związku z tym nie jest to zagadnienie będące istotą niniejszego sprawozdania.
- d. Dla zrealizowanego projektu, do obsługi ww. rozwiązania użyte zostały następujące klasy pochodzące z biblioteki „QT”:
 - i. w obu aplikacjach „QTcpSocket” [<http://doc.qt.io/qt-5/qtcpsocket.html#details>], dziedzicząca z klasy „QAbstractSocket” [<http://doc.qt.io/qt-5/qabstractsocket.html#details>].
 - ii. w aplikacji „Server” dodatkowo „QTcpServer” [<http://doc.qt.io/qt-5/qtcpserver.html#details>].
- e. Najważniejszy z punktu widzenia realizacji projektu – jest autorski „protokół” komunikacyjny, realizowany w warstwie aplikacji. Najważniejszym składnikiem niniejszego protokołu jest struktura (klasa) o nazwie „MsgAboutGame”, dzięki której zrealizowano „merytoryczną” stronę komunikacji w protokole.
 - i. Zawartość struktury wygląda następująco:

```
class MsgAboutGame{
public:
    char x_old;    // old 'x' position of draught
    char y_old;    // old 'y' position of draught
    char x_new;    // new 'x' position of draught
    char y_new;    // new 'y' position of draught
    char x_beat;   // 'x' position of beated draught, if none=-1    /* FOR FUTURE USE */
    char y_beat;   // 'y' position of beated draught                /* F.E. LEAPS    */
    info happened; // 2 server about client's state | definition above^
    /* standard konstruktor */
    MsgAboutGame()
    {
        x_old = y_old = x_new = y_new = x_beat = y_beat = 0;
        happened = MOVE_MAKE;
    }
};
```

- ii. Konkretną informację o zdarzeniu (przyczynie komunikacji) zawiera typ enumerowany „*info*” (zrealizowany jako pole klasy „*MsgAboutGame*”):

```
/* enum to send and receive about type of msg to/from the server */
enum info{
    MOVE_MAKE = 0,
    LOST_GAME = 1,
    ELSE_DISCONNECT = 2,
    FULL_SERVER = 3,
    CLIENT_CONNECTED = 4,
    CLIENT_SECOND_CONNECTED = 5,
    YOUR_TURN_IS = 6
    //TODO - add/edit if necessary in development, remember to add this to server heading too
};
```


- iii. W zależności od konkretnego zdarzenia (jakiego dokładnie – tę informację przekazuje zmienna enumerowana „*info*”) przesyłany gniazdem obiekt klasy „*MsgAboutGame*” przyjmuje różnorakie wartości dla swoich pozostałych pól – są one typu znakowego [„*char*”] i domyślnie przyjmują [przy tworzeniu nowego obiektu klasy] wartość liczbową 0). Motywacja istnienia właśnie takich zmiennych w klasie „*MsgAboutGame*” została w sposób wyczerpujący opisana w kodzie, poprzez komentarz przy każdej z nich.

3. Sposób realizacji i implementacji

- a. Aplikację zarówno kliencką oraz serwerową tworzone w środowisku programistycznym „*QT Creator 3.3.1*”. Wersja biblioteki „*QT 5.4.1*” (najnowsza stabilna w momencie tworzenia niniejszego sprawozdania). Na wcześniejszym etapie tworzenia projektu korzystano też z wersji: „*QT Creator 3.3.0*” i „*QT 5.4.0*”
- b. Projekt kompilowano (zgodnie z wymaganiami - bez ostrzeżeń) za pomocą wbudowanego w program „*QT Creator 3.3.1*” zestawu narzędzi „*Desktop QT {version} MinGW 32bit*” (port kompilatora *GCC* i narzędzi pochodnych) pod kontrolą systemu operacyjnego „*Windows 8.1 Pro x64*”, konkretne ustawienia budowania pokazane są poniżej:

Desktop Qt 5.4.1 MinGW 32bit

BudowanieUruchamianie

 **Ustawienia budowania**

Konfiguracja budowania: Debug Dodaj Usun Zmień nazwę...

Ogólne

Kompilacja w innym miejscu: ☒

Katalog wersji: DD\Documents\GitHub\SKII_Project-Internet_Draughts\build-Server-Desktop_Qt_5_4_0_MinGW_32bit-Debug [Przeglądaj...](#)

Kroki procesu budowania

qmake: qmake.exe Server.pro -r -spec win32-g++ "CONFIG+=debug" [Szczegóły](#)

Make: mingw32-make.exe in C:\Users\3c0nDD\Documents\GitHub\SKII_Project-Internet_Draughts\build-Server-Desktop_ [Szczegóły](#)

Dodaj krok budowania

Kroki procesu czyszczenia

Make: mingw32-make.exe clean in C:\Users\3c0nDD\Documents\GitHub\SKII_Project-Internet_Draughts\build-Server-Desktop_Qt_ [Szczegóły](#)

Dodaj krok czyszczenia

Środowisko budowania

Użyj Środowisko systemowe [Szczegóły](#)

4. Krótka informacja na temat obsługi programu.

- a. Uruchomić aplikację „*Server*” i dwie instancje aplikacji „*Client*”.
- b. W aplikacjach „*Client*” wpisać w polu tekstowym adres IP serwera. Domyślnie jest ono wypełnione adresem „127.0.0.1” (local loopback).
- c. W aplikacjach „*Client*” wcisnąć przycisk „*Connect to the server above*”. Nastąpi wtedy próba podłączenia do aplikacji serwera pod podany adres. W przypadku sukcesu – przejść do pkt. „4d.”, w przeciwnym wypadku – przejść do pkt. „4h”.
- d. W aplikacji „*Client*” która aktualnie posiada „turę” - zaznaczyć figurę na planszy oraz pole w które chcemy ją przenieść (pierwsze kliknięcie na figurze która chcemy przesunąć, następnie analogiczne kliknięcie na polu docelowym dla ruchu, który figura ma wykonać). W przypadku gdy ruch jest dozwolony - zostanie on wykonany, a „tura” przekazana drugiemu graczowi. W przeciwnym przypadku wyświetlony zostanie stosowny komunikat (np. o niedozwolonym ruchu), po którego wyłączeniu aplikacja będzie czekać na ponowną próbę wykonania ruchu.
- e. Wykonywać naprzemiennie (w dwóch podłączonych do serwera aplikacjach „*Client*”), tak długo „ruchy pionów” - aż któryś z graczy nie wygra (poprzez zabicie wszystkich figur przeciwnika) lub nie nastąpi „nieplanowane zerwanie połączenia” (np. z powodu awarii sieci).
- f. W przypadkach opisanych w punkcie „4e.” aplikacja „*Server*” wyśle odpowiednią wiadomość do pozostałych podłączonych klientów. W następstwie tego, w takich aplikacjach „*Client*” - stan programu zmieni się na ten z pkt. „4b.”, po wyłączeniu okna informującego o zdarzeniu.
- g. W przypadku gdy do aplikacji „*Server*” podłączone są już 2 aplikacje „*Client*” – następne próby podłączenia będą odrzucane i pojawi się komunikat informujący o błędzie. Podobnie będzie w innych sytuacjach, które nie skutkują połączeniem z serwerem, wywołując komunikat o przyczynie niepowodzenia. Po wyłączeniu okna informacyjnego – stan programu „*Client*” wróci do tego z pkt. „4b”.
- h. W aplikacji „*Server*” wyświetlane są informacje tekstowe w reakcji na ważniejsze zdarzenia, m.in.: podłączenie, odłączenie klienta / odebranie istotnych informacji od aplikacji klienta (np. o ruchu figury) / próba podłączenia się więcej niż dwóch klientów / etc.