Raport z zadania programistycznego 1; SKJ_1.2~

Filip Rajzer s17149

Ogólny opis rozwiązania:

- Aplikacja składa się z dwóch głównych modułów :
  - GUI opartego w dużej mierze o FX & bibliotekę zewnętrzną contolsfx-9.0.0
    - Odpowiada za całość interakcji z użytkownikiem, wszystko wypisywane na konsoli (poza końcowym wynikiem) ma formę informacyjną i służy monitorowaniu funkcjonowania Comms.
    - Udostępnia takie funkcjonalności jak:
      - Rozgrywka w wielki turniej (cel projektu)
      - Czat pełniący również funkcje terminala poleceń z następującymi funkcjami:
        - /Man => manual
        - /StN @Nick => zmiana nick'a gracza; zmiana czysto kosmetyczna, widoczna dla wszystkich
        - /Con @ip @port => podłączenie do sieci w której toczy się rozgrywka, umożliwia odbieranie komunikatów zawierających wyniki
        - /LAP => wypisanie danych:
          - Znanych graczy
          - Aktualnych bram komunikacyjnych
          - Uszkodzonych/czekających pakietów
          - Wszystkich dotychczas otrzymanych wyników o niewątpliwej reputacji (brak podejrzeń oszustwa)
        - /kill => zabija aplikacje; tworzy ducha w sieci do której był przyłączony
          - Duch sam zniknie gdy którykolwiek z klientów wykryje że nie da się połączyć; w imieniu umarłego pożegna się z pozostałymi, tym samym przywracając integralność sieci
      - Monitor „http" ograniczający funkcjonalność aplikacji do czatu/konsoli
  - Comms opartego o tradycyjną Javę, z elementami FX celem komunikacji z GUI.
    - Box: => Klasa odpowiedzialna za całość komunikacji z resztą sieci
    - Gate: => Klasa reprezentująca kanał łączności z danym graczem
    - Player: => opakowanie dla danych jednoznacznie identyfikujących odbiorcę;
      - Ze względów praktycznych sieć NIE wymusza różnych nick'ów i zamiast tego identyfikuję graczy po IP & port;

Przykład funkcjonowania sieci; ze względów praktycznych będzie to znacząco uproszczony i okrojony „log" normalnie generowany przez Comms:

//(K-klijent; G-gracz; X - numer jednoznacznie identyfikujący: K0+ZS(1)=>G0 gdzie ZS = > zmiana statusu; (1) //dołączenie do rozgrywki; <>==lista)

//powitanie

K0: {nadawca:K0,odbiorca ?0,{Lista graczy znanych K0}};

G3:{nadawca:G3,odbiorca <Lista graczy znanych G3\K0>,{Lista graczy znanych K0 + K0}}

G3:{nadawca:G3,odbiorca <Lista graczy znanych K0 + K0>,{Lista graczy znanych G3 + G3}}

//dołączenie do gry

G0:{nadawca:G0,odbiorca<Lista graczy znanych G0>, zmiana statusu (1) => dołączam do rozgrywki}

//gra

G2:{nadawca:G2,odbiorca<G0>, zaproszenie do gry}

//GUI zaproszenie

G0:{nazawca:G0,odbiorca G2,akceptuje}

G2:{nadawca:G2,odbiorca<G0>,<Lista graczy w TEJ grze + G0> + uniqid rozgrywki}

//odpytanie gracza o wartość

<>:{nadawca<Wszyscy w rozgrywce>,odbiorca<wszyscy w rozgrywce>,<???>} => wrzuca się na stos zamkniętych pudełek

//oczekiwanie na sieć

<>:{nadawca<Wszyscy w rozgrywce>,odbiorca<wszyscy w rozgrywce>,klucz} => odnajduję na stosie pudełko wysłane przez tę samą osobę co klucz, otwiera

//oczekiwanie na sieć & obliczanie wyniku

<>:{nadawca<Wszyscy w rozgrywce>,odbiorca<wszyscy znani gracze, bool wygrałem + uniqid rozgrywki}

//rozgrywka zakończona==========

//przykład samo-naprawy:

G5:=>awaria wyłącza klienta nie informuje sieci o odejściu z rozgrywki

G2:{nadawca:G2,odbiorca<Wszyscy znani gracze>, wiadomość:"GG"}

G2:=>Box:=>pack:=>klient:G5 nie odpowiada na ponawianą próbę wysłania wiadomości;

=>wiadomość do G5 skasowana;

=>naprawa sieci:

//podszywa się pod nieobecnego celem naprawy sieci

G2:{nadawca:G5,odbiorca<Wszyscy znani gracze>, kończę rozgrywkę usuńcie mnie z baz danych}

//G0 ma dość i chcę wyjść z gry:

G0:=>podjęto próbę zakończenia rozgrywki;

=>tryb miękki:

=>exitFun: sprawdzenie warunków wyjściowych:

=>niespełnione: próba nawrócenie gracza do rozgrywki:

=>niepowodzenie: gracz opuszcza grę na siłę:

//dla aplikacji ważniejsze jest utrzymanie spójniej wiedzy globalnej niż utrzymanie gracza w grze

G0:{nadawca:G0,odbiorca<Wszyscy znani gracze>, kończę rozgrywkę usuńcie mnie z baz danych}

G2:=>podjęto próbę zakończenia rozgrywki;

=>tryb miękki:

=>exitFun: sprawdzenie warunków wyjściowych:

=>spełnione

G0:{nadawca:G0,odbiorca<Wszyscy znani gracze>, kończę rozgrywkę usuńcie mnie z baz danych}

//==========

Co nie zostało przedstawione a może występować to obsługa sieci; jest ona ukryta przed aplikacją. Połącznie TCP i jego utrzymanie/zamknięcie spoczywa na Comms i nie wpływa w żaden sposób na rozgrywkę.

Szczegółowy opis rozwiązania:

Dane NT sieci:

Dane o sieci w której znajduję się gracz są przechowywane w Klasie „Player” na listach:

App.players => wszyscy znani gracze

App.toPlayPlayers => gracze z którymi dany klient musi jeszcze zagrać

Box.playersToInv => gracze zaproszeni do danej rozgrywki

Box.inThisGame => gracze w TEJ rozgrywce

Klasa Player: String nick; String IP; int port;

Rozgrywki po dołączeniu nowego gracza do sieci i wysłaniu JOIN:

W momencie wysłania komunikatu JOIN wszyscy otrzymują komunikat że dana osoba jest w rozgrywce, zostaje ona też (dla graczy którzy mają status G) wrzucona na listę graczy playersToPlayWith. Poza tym rozgrywki przebiegają tak samo jak wcześniej, nie ma to żadnego istotnego wpływu na zachowanie się sieci.

Uczciwość:

Ta jest zapewniona przez niemożność uzyskania przez danego klienta dostępu do zawartości pakietu zawierającego wartość wysłaną przez innego gracza w inny sposób niż manipulacja oprogramowania rozgrywki /użycie debugera z znajomością programu gdzie szukać (a nie jest to oczywiste).

Aby wykraść liczbę należy uzyskać dostęp do zawartości pakietu box; Czy to ze strumienia danych czy z publicznego stosu inbox (tu trzeba się spieszyć; paczki są przetwarzane do 60 paczek/sec!), prywatnego stosu locked (tu ma się całe 9 sek~ czasu na odnalezienie się sytuacji) i odczytaniu zawartości paczki (właściwej wartości z prywatnego stosu paczki). Cokolwiek trudne zwłaszcza że klienci objęci są ramami czasowymi (egzekwowanymi zewnętrznie) których naruszenie skutkować może wyrzuceniem z odebraniem prawa do wysłania własnej liczby (wysłane zostaje wtedy 0). W przypadku metod programowych z kolei należy zagadnąć hasło aby uzyskać liczbę która i tak nie zostanie przekazana wywołującemu metodę ale dodana do sumy wszystkich liczb. Istnieje tu możliwość zaimplementowania dodatkowego mechanizmu powodującego zamknięcie aplikacji w przypadku podaniu nieprawidłowego hasła aby zabezpieczyć się przed BruteForce.

Klucz stanowi kwadratowa tablica byte o losowych wymiarach 10-99 i losowych wartościach pól 0-255, co daje 89 możliwych rozmiarów kluczy z ilością kombinacji sięgającą ok. 2.5*10^24065 w sumie

Obserwację i wnioski:

Obserwacje:

Sieć: odłączenie hosta w czasie rozgrywki powoduje konieczność restartu gry przez wszystkich uczestników danej rozgrywki; niektóre pakiety wysyłają się naprawdę długo, poniekąd wynika to ze struktury komunikacji opartej o ObjectStream; sieć dopuszcza możliwość podszywania się pod siebie poszczególnych graczy; nie została zaimplementowana walidacja tego kto wysyła wiadomość.

Aplikacja: całość aplikacji powinna zostać przepisana z wykorzystaniem HashMap celem optymalizacji; zamiast AnimationTimera powinno się wykorzystywać Thread na stanowisku „listonosza”; Zamiast wątków powinno się wykorzystać Exec (nie wszędzie) & try{} z możliwością przerwania; duża zależność funkcjonowania aplikacji od kaprysów FX; w tym możliwe opóźnienia w komunikacji;

Możliwe implementacje (porzucone): Okno „Ustawienia”; Czat grupowy/selektywny a nie globalny; routing (w Box prefix whatdoicarry >200); Automatyczne ping celem wykrycia możliwej awarii klienta; automatyczne oznaczanie oszustów & banowanie danego IP na okres istnienia danej sieci (sieć przestaje istnieć w momencie gdy każdy klient który jest do niej przyłączony, był do niej przyłączony, lub do kogokolwiek kto był do niej przyłączony, zostaje wyłączony (tak aby nie dało się przekazać informacji));

Wyniki eksperymentów skalowanych: ze względu na ograniczone środowisko testowania (tylko jeden laptop) testy były ograniczone do max. 3 klientów na raz. Wszystkie uzyskane obserwacje zostały opisane powyżej.