Uniwersytet Jagielloński w Krakowie
Instytut Informatyki i Matematyki Komputerowej

Oleksandr Kuzhel
Nr albumu: 1086931

Teiru MMORTS
(Massively multiplayer online real-time strategy)
Projekt i implementacja części serwerowej

Praca licencjacka
na kierunku Informatyka Stosowana

Praca wykonana pod kierunkiem
<tytuł/stopień naukowy Imię Nazwisko>
<Instytut/Zakład>

Kraków 2015

Opis bibliograficzny pracy :
Kuzhel Oleksandr(2015). Teiru MMORTS (Massively multiplayer online real-time strategy). Projekt i implementacja części serwerowej.

Abstrakt

Praca licencjacka jest poświęcona projektowaniu oraz implementacji serwera dla wieloosobowej gry z rozgrywką czasu rzeczywistego. Serwer jest jedną z najważniejszych części każdej gry wieloosobowej. Programy tego typu przeważnie wykorzystują architekturę typu klient-serwer. Takie rozwiązanie ma zarówno wady jak i zalety które są są później szczegółowo rozpatrzone. Pisząc tą pracę skupiłem uwagę przede wszystkim na szczegółowym opisywaniu omawianych zagadnień. Stworzenie szybkiego, bezpiecznego i wydajnego serwera jest jedną z głównych inspiracji tego projektu. Poniższa praca powstała podczas tworzenia projektu zespołowego i ma na celu implementację odpowiedniego serwera dla gry wieloosobowej Teiru MMORTS.

Bachelor thesis is dedicated to a projecting and implementation of online game server. Server is one of the most important part of each multiplayer online game. This type programs for the most part using client-server architecture. This solution has it pros and cons, which I described in my work. Writing this work I elaborate on discussed issues. Creating of fast, secure and productive server is one of my major intentions.
Work mentioned below rised during creation of team project and the purpose of my work is to implement server for inline multiplayer game Teiru MMORTS.

Słowa kluczowe
GRA KOMPUTEROWA , UNITY , JAVA , MYSQL , CZĘŚĆ SERWEROWA

COMPUTER GAME, UNITY , JAVA , MYSQL , SERVER PART

SPIS TREŚCI

Wstęp

Gry czasu rzeczywistego z roku na rok zyskują coraz większą popularność, dlatego implementacja serwera wymaga, aby dużą część uwagi poświęcić optymalizacji wydajności i bezpieczeństwu serwera.

Najczęściej spotykana architektura w grach komputerowych jest architektura typu klient-serwer. Klient łączy się z serwerem za pomocą określonych protokołów komunikacyjnych. Komunikacja między serwerem, a klientem odbywa się w następujący sposób : klient nawiązuje połączenie z serwerem , po czym wysyła żądanie w określonym formacie do serwera i oczekuje na odpowiedź. Serwer ciągle oczekuje na żądania klientów i w momencie otrzymania żądania od jednego z klientów przetwarza je, po czym wysyła odpowiedź.

Celem projektu jest implementacja serwera dla gry wieloosobowej czasu rzeczywistego Teiru MMORTS. Serwer został napisany w języku Java z wykorzystaniem bibliotek Netty[1], wersja 3.6.10 i MySql Connector[2] , wersja 5.1.34. Komunikacja po stronie klienta została napisana w języku C# z wykorzystaniem biblioteki graficznej Unity[3] , wersja 4.6. Diagramy przedstawione w pracy zostały stworzone za pomocą Lucidchart[5], a model bazy danych za pomocą Vertabelo[6].

Stworzony serwer oczekuje na żądania klientów, opracowuje je i wysyła odpowiedzi (np. odbiera informacje o przebiegu gry od jednego gracza i wysyła je do każdego z osobna). Zmienia, usuwa i dodaje informację do bazy danych. Serwer pozwała graczom na wspólną rozrywkę w świecie gry, pozwala też na swobodną wymianę wiadomości. Podczas pisania serwera skupiałem uwagę przede wszystkim na jego wydajności. Aby ją zwiększyć próbowałem wykorzystując oszczędny protokół komunikacyjny i zostawić większą część funkcjonalności po stronie klienta (np. zmianę pozycji gracza).

W pierwszym rozdziale opiszę nasz projekt, omówię UML i bazy danych.
W drugim rozdziale przedstawię model serwera i bazy danych.

1. Opis projektu

Celem projektu jest napisanie gry wieloosobowej czasu rzeczywistego. Podzieliliśmy projekt na 3 części : część kliencka , część serwerowa i interfejs graficzny. W tej pracy jest opisane tworzenie i implementacja części serwerowej. Różne procesy łączą część kliencką i część serwerową, ale chociaż czasem są to te same procesy, serwer i klient widzą je inaczej. Na przykład podczas logowania klient skupia się na wprowadzanych danych i na przesłaniu tych informacji dalej do serwera , który skupia się na poprawnym odczycie danych otrzymanych od klienta , sprawdzaniu ich poprawności (opcjonalnie łączeniu się z bazą danych) i wysyłaniu opracowanej odpowiedzi do klienta.

Gracz ma możliwość zapoznać się z grą za pomocą samouczka(ang. Tutorial), albo zacząć przygodę w świecie gry z innymi graczami. Zaczynając grę online gracz musi stworzyć postać. Tworząc postać gracz wybiera klasę swojej postaci, wybiera umiejętności dostępne tylko tej klasie i rozdaje statystyki(siła, zręczność, budowa, inteligencja, roztropność, charyzma). Niezależnie od klasy gracz zaczyna grę na 1 poziomie . Gracz ma możliwość rozwoju postaci. Dostępne są zlecenia(ang. Quests) , które mogą być fabularne lub polegające na zwykłych zadaniach. W części klienckiej są zrealizowane moby(jednostki NPC) przeznaczone do zabicia dzięki czemu postać zdobywa doświadczenie i złoto. Gracz ma też możliwość sprzedaży lub kupna zdobytych przedmiotów(ang. Items). W grze jest zrealizowana możliwość wymiany informacji między graczami, czat prywatny lub publiczny. Gracz ma możliwość tworzenia drużyn wieloosobowych (maksymalnie 4 osoby), które pozwalają pojedynkować się z inną dowolną grupą. Każdy gracz posiada konto , więc istnieje możliwość dodawania innych graczy jako przyjaciół. Co pozwala szybko nawiązać z nimi kontakt w przyszłości.

Walki toczą się według ustawionych reguł. Rozgrywka jest turowa. Istnieje kilka rodzajów pojedynków : PvP(Player versus Player) walki pomiędzy graczami, GvG(Guild versus Guild) walki pomiędzy zespołami 4 osobowymi. Walka jest podzielona na 3 etapy. Etap ustawienia - ustalenie kolejki według inicjatyw graczy. Etap działania – walczący działają w kolejności inicjatyw(od większej do mniejszej). Etap końcowy – po tym jak każdy z walczących rozegra turę , gracz z większą inicjatywą działa jeszcze raz, a inne etapy powtarzają się aż do końca starcia. W trakcie tury gracz może wykonać ruch , akcję całorundową , akcję darmową lub wykonać akcję standardową.

Sukces w walce zależy w dużej mierze od współczynników bojowych. Krótka lista współczynników bojowych jest przestawiona poniżej.

Test ataku – próba trafienia przeciwnika , podejmowana jest każdym graczem w swojej turze. Jeśli posiadana przez gracza premia do ataku razem z wynikiem rzutu kostką k20 przekracza klasę pancerza przeciwnika to przeciwnik otrzymuje obrażenia.

Premia do ataku – składa się z bazowej premii do ataku(BAB) i z modyfikatora z Siły(dla ataki bronią do walki wręcz) . Bazowa premia do ataku + kara z zasięgu + modyfikator ze Zręczności (dla ataki bronią do walki dystansowej).

Premia z Siły – do obrażeń zadanych w wyniku trafienia przeciwnika bronią do walki wręcz stosuje się modyfikator z Siły.

Klasa Pancerza – to 10 + premia z tarczy + premia z pancerze + modyfikator ze Zręczności. Pokazuje to jak trudno wyprowadzić cios który zada graczowi obrażenia.

Punkty wytrzymałości(ang. Heath Points) – przy -10 punktach wytrzymałości postać

ginie. Jeśli postać ma nie mniej niż -9 punktów wytrzymałości i nie więcej niż -1 ona zostaje umierająca. Kiedy postać ma 0 punktów wytrzymałości to jest okaleczona.
Okaleczony – (0 punktów wytrzymałości)gracz traci możliwość wykonywania akcji całorundowej i nie może wykonać akcję ruchu i akcję standardową w jednej turze(powinien wybrać jedną z nich). Po zakończeniu tury postać traci jeden punkt wytrzymałości i zostaje umierający.
Umierający – (od -1 do -9 punktów wytrzymałości)umierający bohater nie ma prawa do żadnych działań i każdą rundę traci jeden punkt wytrzymałości , dopóki nie umrze lub dopóki ktoś go nie wyleczy.
Martwy – (-10 i mnie punktów wytrzymałości)grać traci możliwość być wyleczonym i już nie może wrócić do pojedynku.
Leczenie – jeśli podczas leczenia postaci punkty wzrosną powyżej 0 , postać może funkcjonować tak jakby nigdy nie była okaleczona lub umierająca.
Rzuty obronne – postać może być zaatakowana magią lub innym niezwykłym atakiem. Wtedy postać ma prawo do rzutu obronnego który pozwoli jej częściowo lub w całości unicestwić efekt ataku. Wynik rzutu obronnego to rzut kostką k20 + premia (w zależności od klasy, poziomu i atrybutu) .

Rodzaje akcji są podzielone na 3 różne klasy. Podział na klasy odbywa się według długości akcji.

Akcja standardowa – to każde wykonane przez gracza działanie , takie jak atak wręcz lub dystansowy lub jedno z dostępnych zaklęć.
Akcja ruchu – postać zmienia swoją pozycję.
Akcja całorundowa – to działanie które postać wykonuje przez całą rundę.

1.1 UML

UML (ang. Unified modeling language) – jest językiem modelowania wizualnego i jest wykorzystywany przede wszystkim do opisywania systemów. UML jest zestawem różnorodnych pojęć oraz diagramów, które dają możliwość szczegółowego opisania każdego programu, koncepcji projektowanego programu, oraz wszystkich kluczowych fragmentów. UML wykorzystuje wiele rozmaitych elementów graficznych dla opisu systemu, Dokładnie określone zasady łączenia ich pozwalają na wszechstronne i precyzyjne przedstawienie projektu. Diagramy UML opisują co system powinien robić, ale nie przedstawiają jak on to robi. W następnych podroździałach krótko opiszę używane w tej pracy diagramy UML-owe.

1.1.1 Diagram klas

Diagram klas prezentuje klasy obiektów w programie i opisuje wszystkie jej atrybuty (właściwości, zmiennie i funkcje). Atrybuty klasy mogą być: a) publiczne(ang. public) „+” b) prywatne(ang. private) „ -” c) chronione(ang. protected) „#” . Diagramy klas również ułatwiają analizę projektu i reprezentują statyczną część programu.

1.1.2 Diagram przypadków użycia.

Diagram przypadków użycia opisuje działania systemu z punktu widzenia potencjalnego użytkownika. Jest to zbiór scenariuszy wywoływany przez aktora(użytkownika). Każdy przypadek użycia może być wykorzystany wielokrotnie. Pojedynczy przypadek użycia zawiera pewną ilość scenariuszy opisujących różne możliwości korzystania z systemu.

1.1.3 Diagram stanów

W wyniku różnych zdarzeń w systemie różne obiekty zmieniają swoje stany. Demonstrować te zmiany możemy za pomocą diagramów stanów. Opisują one zmiany stanu jednego z obiektów. Diagramy te są niezbędne dla programistów gdyż opisują założone zachowanie obiektów, co ułatwia implementowanie ich.

1.1.4 Diagram sekwencji

Diagram sekwencji składa się z obiektów(prostokąty z nazwami), komunikatów(linie) i czasu(przesunięcie wzdłuż pionowej osi). Obiekty znajdują się w górnej części diagramu, od obiektów w dół ciągną się pionowo linie(przesunięcie po takie linii w dół odpowiada upływowi czasu). Od każdego obiektu biegnie w dół jego linia życia. Prostokąty umieszczane na liniach życia reprezentują wykonywane przez obiekty zadania. Zdarzają się sytuacje kiedy obiekt może wywoływać sam siebie(rekurencja). Diagram sekwencji pozwała nam wykorzystać takie narzędzia jak rozgałęzienie(if) i pętle(while). Diagram sekwencji pokazuje , jak , w zależności od czasu, odbywa się komunikacja między obiektami.

1.1.5 Diagram komunikacji

Diagram komunikacji opisuje metodę wymiany informacją pomiędzy uczestnikami interakcji. Strzałki między obiektami prezentują przesalaną informację(komunikaty). Obok każdego komunikatu znajduje się liczbą wskazująca na kolejność ich wysyłania. Największą uwagę w diagramach komunikacji skupia się na interakcji między obiektami i przesyłanymi między nimi komunikatami, więc możemy powiedzieć , że diagramy komunikacji określają rzeczywiste powiązania między obiektami co ułatwia zrozumienie interakcji.

1.1.6 Diagram czynności

Diagram czynności jest rozwinięciem diagramu stanów. Co wynika z nazwy , w diagramie czynności szczególna uwaga jest skupiona na czynnościach(prostokąty z zaokrąglonymi kątami). W ten sam sposób co w diagramie stanów przedstawione są końcowe i

początkowe punkty. Przede wszystkim diagram czynności pozwala opisać to co dzieje się podczas wykonywania operacji.

1.1.7 Diagram komponentów

Diagram komponentów zawiera komponenty , interfejsy i związki. Zamiast skupiać się na stanach lub czynnościach diagram komponentów przedstawia nam elementy że świata rzeczywistego. Zbiór operacji(interfejs) daje nam dostęp do do komponentu. Komponent to element oprogramowani – fizyczna część systemu(np. Baza danych, wykonywany plik, dokument, etc).

1.2 Baza danych.

Baza danych to zbiór danych zapisanych zgodnie z określonymi regułami. W naszym przypadku to dane cyfrowe zapisywane zgodnie z zasadami przyjętymi dla danego programu komputerowego specjalizowanego do gromadzenia i przetwarzania danych.

1.2.1 Relacyjne bazy danych.

W bazach relacyjnych wiele tablic danych może współpracować ze sobą (są między sobą powiązane). Bazy relacyjne posiadają wewnętrzne języki programowania, wykorzystujące zwykle SQL do operowania na danych, za pomocą których tworzone są zaawansowane funkcje obsługi danych. Relacyjne bazy danych (jak również przeznaczony dla nich standard SQL) oparte są na kilku prostych zasadach:
1.Wszystkie wartości danych oparte są na prostych typach danych.
2.Wszystkie dane w bazie relacyjnej przedstawiane są w formie dwuwymiarowych tabel(w matematycznym żargonie noszących nazwę „relacji”). Każda tabela zawiera zero lub więcej wierszy i jedną lub więcej kolumn („atrybutów”). Na każdy wiersz składają się jednakowo ułożone kolumny wypełnione wartościami, które z kolei w każdym wierszu mogą być inne.
3.Po wprowadzeniu danych do bazy, możliwe jest porównywanie wartości z różnych kolumn, zazwyczaj również z różnych tabel, i scalanie wierszy, gdy pochodzące z nich wartości są zgodne. Umożliwia to wiązanie danych i wykonywanie stosunkowo złożonych operacji w granicach całej bazy danych.
4.Wszystkie operacje wykonywane są w oparciu o algebrę relacji, bez względu na położenie wiersza tabeli. Wiersze w relacyjnej bazie danych przechowywane są w porządku zupełnie dowolnym – nie musi on odzwierciedlać ani kolejności ich wprowadzania, ani kolejności ich przechowywania.
5.Z braku możliwości identyfikacji wiersza przez jego pozycję pojawia się potrzeba obecności jednej lub więcej kolumn niepowtarzalnych w granicach całej tabeli, pozwalających odnaleźć konkretny wiersz. Kolumny te określa się jako „klucz podstawowy” (ang. primary key) tabeli.

1.2.2 Klucze.

Klucze to zbiory atrybutów mających określoną właściwość. Dzięki nim, możemy jednoznacznie identyfikować każdy pojedynczy wiersz. Znajomość pojęć kluczy podstawowych i obcych jest niezbędna do tworzenia zapytań, odwołujących się do wielu tabel. Dalej przedstawię najczęściej spotykane typy klucze.

Superklucz to dowolny podzbiór atrybutów, identyfikujący jednoznacznie każdy wiersz. Każda RELACJA (tabela) może zawierać wiele takich kluczy. Szczególnym przypadkiem jest superklucz składający się ze wszystkich atrybutów (kolumn) danej tabeli.

Klucz kandydujący to dowolny z superkluczy, mogący zostać kluczem podstawowym. W implementacji bazy danych, w praktyce nie istnieje jako niezależny osobny (zmaterializowany byt) jako taki. Jest to tylko założenie teoretyczne.

Klucz podstawowy(primary key) to wybrany (zazwyczaj najkrótszy), jednoznacznie identyfikujący każdy, pojedynczy wiersz, zbiór atrybutów (kolumn) danej relacji (tabeli). Jest to pierwszy z wymienionych do tej pory kluczy, który ma faktyczne, fizyczne odwzorowania w implementacji bazy danych. Każda tabela może mieć tylko jeden taki klucz.

Klucz obcy to atrybut lub zbiór atrybutów, wskazujący na klucz główny w innej relacji. Klucz obcy to nic innego jak związek, relacja między dwoma tabelami. Definicja klucza obcego, pilnuje aby w tabeli powiązanej, w określonych atrybutach, znaleźć się mogły tylko takie wartości które istnieją w tabeli docelowej jako klucz główny. Klucz obcy może dotyczyć również tej samej tabeli.

1.2.3 Powiązania pomiędzy tabelami.

Związek 1:1(ang. One to one) - każdy wiersz z tabeli A może mieć tylko jednego odpowiednika w tabeli B (i na odwrót). Ten rodzaj relacji może być postrzegany jako podzielenie tabeli na dwie (bo relacja jest jeden do jeden). Stosowany np. wtedy, gdy zbiór dodatkowych atrybutów jest określony tylko dla wąskiego podzbioru wierszy w tabeli podstawowej.

Związek 1:N(ang. One to many) - jest to najczęściej spotykana relacja. Określamy w niej że każdy element ze zbioru A (wiersz tabeli A), może być powiązany z wieloma elementami zbioru B.

Związek N:M(ang. Many to many) - realizowana jest zawsze jako dwie relacje 1:N. Zatem jeśli chcemy między dwoma tabelami zamodelować związek N:M **potrzebujemy trzecią tabelę – łącznikową**.

1.2.4 MySQL.

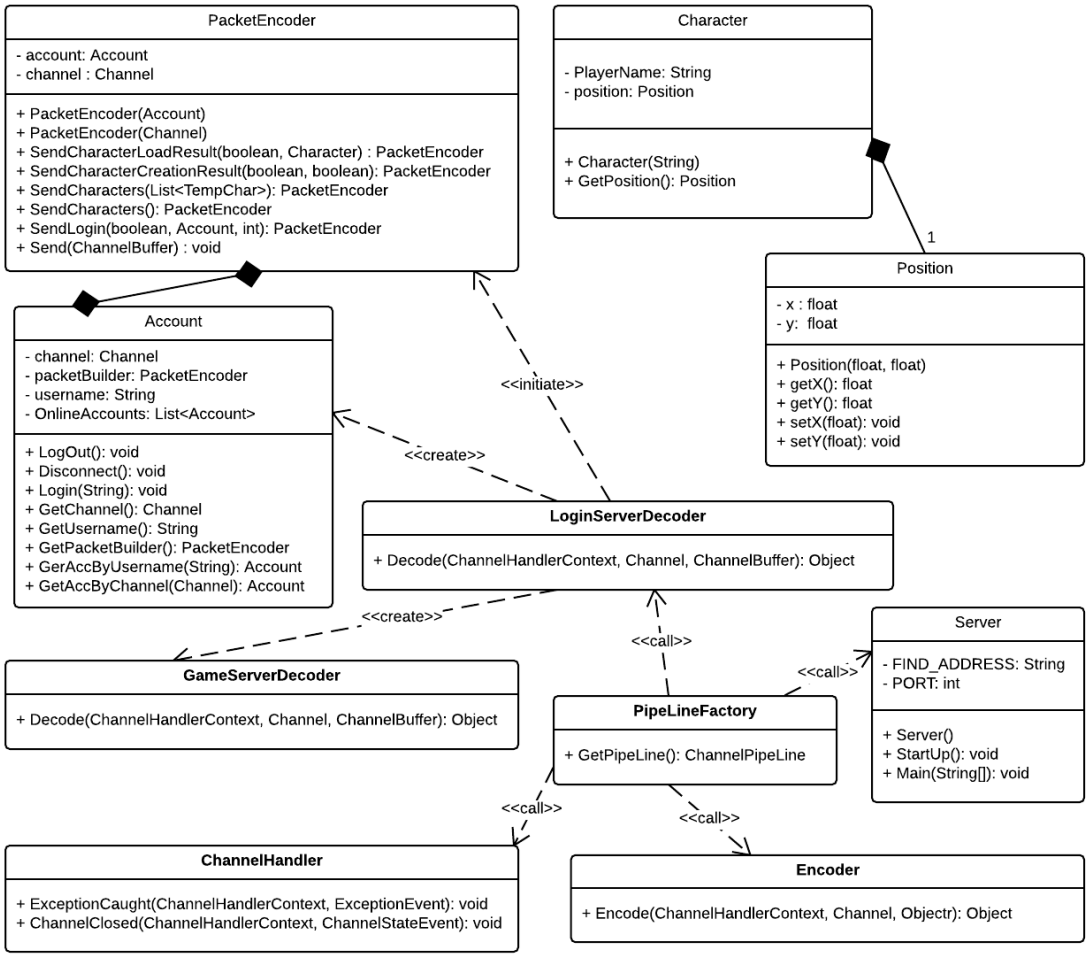
MySQL –wolnodostępny system zarządzania relacyjnymi bazami danych. MySQL rozwijany jest przez firmę Oracle. Wcześniej przez większość czasu jego tworzeniem zajmowała się szwedzka firma MySQL AB. MySQL AB została kupiona 16 stycznia 2008 roku przez Sun Microsystems, a ten 27 stycznia 2010 roku przez Oracle. W międzyczasie Monty Widenius (współtwórca MySQL) stworzył MariaDB –alternatywną wersję opartą na licencji GPL. MariaDB jest oparta na tym samym kodzie bazowym co MySQL i dąży do utrzymania kompatybilności z jej poprzednimi wersjami.

2.Model części serwerowej

2.1.Opis serwera

Serwer w Teiru MMORTS powinien pełnić kilka bardzo ważnych funkcji. Przedewszystkim powinien pozwalać użytkownikom się logować/rejestrować , walczyć i wspólnie swobodnie zwiedzać świat gry. Niżej , za pomocą diagramów spróbuje przedstawić to jak działa serwer.

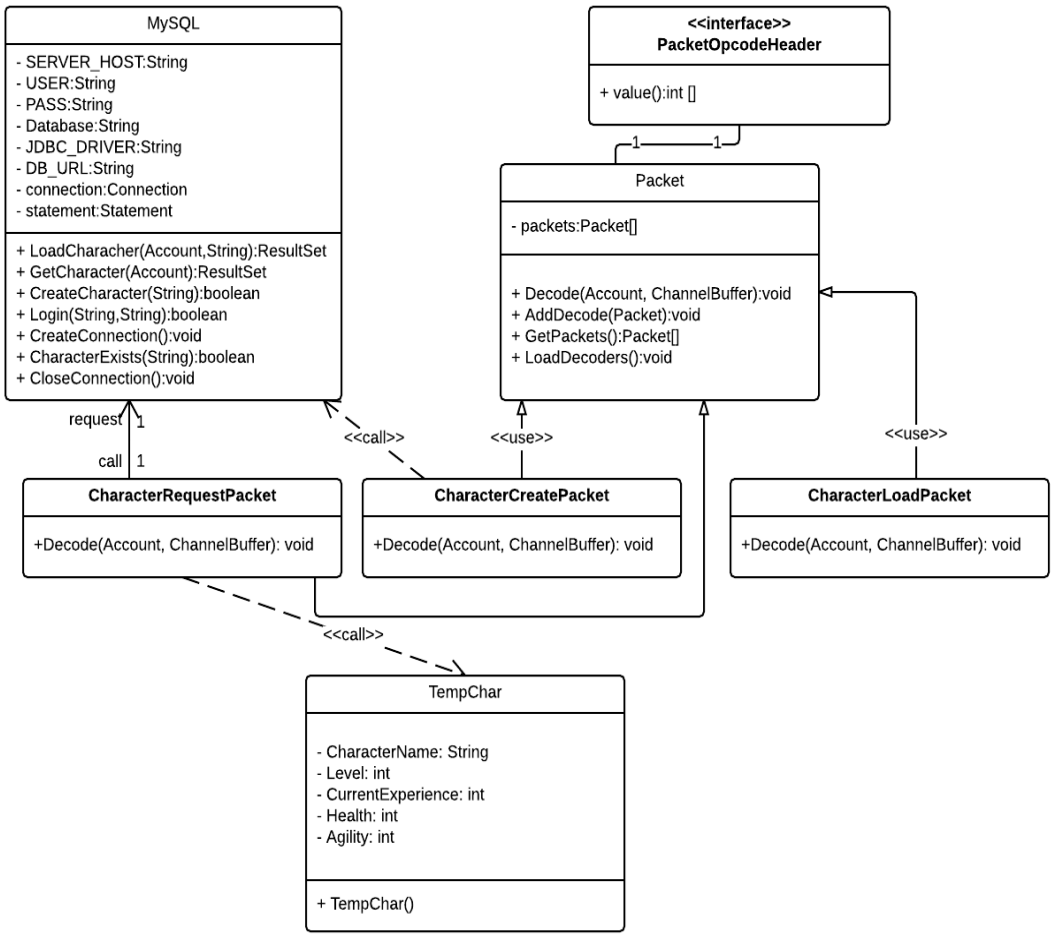
2.1.1 Diagram klas – serwer



Rys. 1: Diagram klas - część 1

Powyżej została przedstawiona część diagramu klas odpowiadająca za logowanie. Serwer ciągle oczekuje na żądania użytkowników . Kiedy klient wysyła żądania serwer odszyfrowuje otrzymany komunikat i sprawdza w bazie danych wprowadzoną przez użytkownika informacje. Proces rejestracji odbywa się podobnie , jedyną różnicą jest to że serwer sprawdza dostępność wprowadzonego identyfikatora użytkownika(ang. Username) i wprowadzonej poczty elektronicznej(ang. Email). Po udanym zalogowaniu, żądania użytkownika przestaje przyjmować

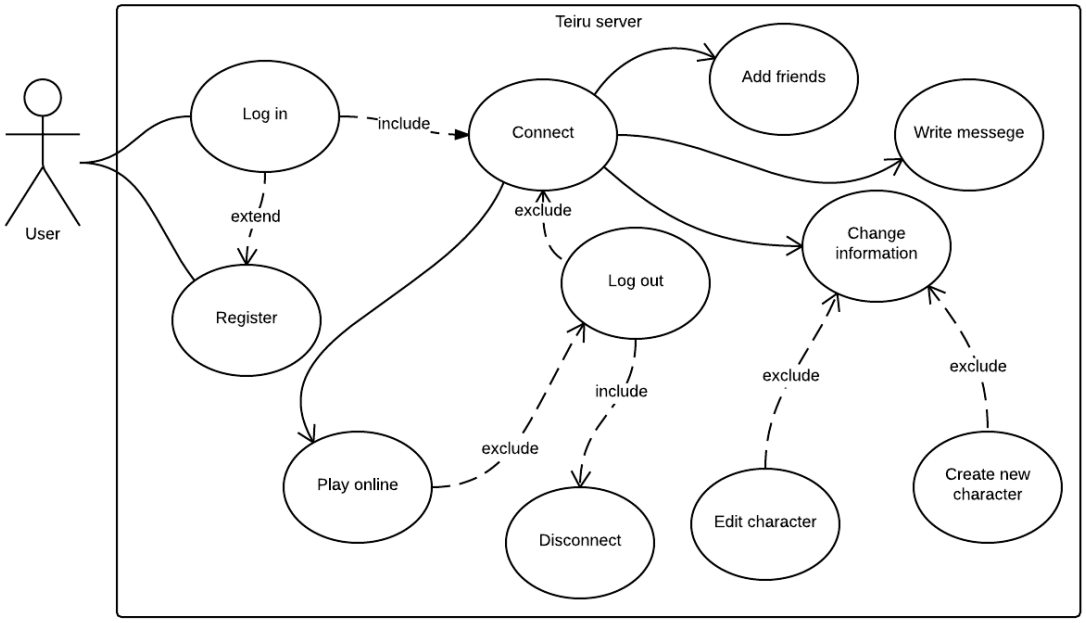
i opracowywać LoginServerDecoder i zaczyna GameServerDecoder, który odpowiada już za proces gry.



Rys. 2: Diagram klas - część 2

Obsługa gracza podczas gry w większej części odbywa się po stronie klienta. Serwer tylko odpowiada na żądania klientów, aktualizuje i przechowuje dane , takie jak pozycja każdej postaci gracza, poziom, punkty wytrzymałości i inne.

2.1.2 Diagram przypadków użycia – serwer



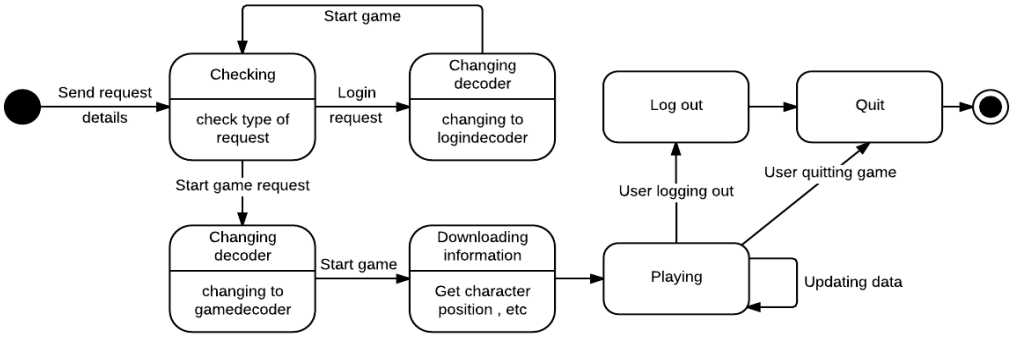
Rys. 3: Diagram przypadków użycia

Jak już pisałem wcześniej diagram przypadków użycia opisuje system z punktu widzenia użytkownika. Na diagramie możemy zobaczyć dostępne dla użytkownika działania związane z serwerem. Następne kilka scenariuszy przypadków użycia przedstawiają parę możliwych dla użytkownika scenariuszy działania.

Nazwa: Zaloguj się **Aktorzy:** Niezalogowany użytkownik	**Główny scenariusz:** 1. Użytkownik wypełnia formularz logowania. 2. Użytkownik potwierdza wpisane dane. 3. System sprawdza otrzymane dane. 4, Użytkownik uzyskuje dostęp do swojego konta. **Rozszerzenia:** 2a. Nieodpowiednie dane(zabronione znaki, zbyt krótkie hasło , etc) a. Użytkownik wpisuje poprawne dane. b. Użytkownik potwierdza wpisane dane. 4a. Nieprawidłowa nazwa użytkownika lub hasło. a. Użytkownik ponownie wpisuje dane. b. System sprawdza otrzymane dane.

Nazwa: Stwórz nową postać **Aktorzy:** Zalogowany użytkownik	**Główny scenariusz:** 1. Użytkownik poprawnie się loguje 2. Użytkownik wybiera opcję „Create new character” 3. Użytkownik wybiera klasę postaci i rozdaje dostępne punkty umiejętności. 4. Użytkownik potwierdza swój wybór. 5. Nowa postać zostaje stworzona i odpowiednia informacja zapisana w bazie danych. **Rozszerzenia:** 1a. Wystąpił problem z logowaniem a. Użytkownik ponownie próbuję się zalogować. 2a. Zbyt dużo postaci a. Użytkownik usuwa jedną z już istniejących postaci. 5a . Wystąpił problem z połączeniem a. Użytkownik zostaje poproszony o ponowne potwierdzenia wyborów.
Nazwa: Edytuj istniejącą postać **Aktorzy:** Zalogowany użytkownik	**Główny scenariusz:** 1. Użytkownik poprawnie się loguje 2. Użytkownik wybiera jedną ze swoich postaci 3. Użytkownik zmienia dane. 4. Użytkownik potwierdza zmiany. 5. Zmiany zostają zapisane w bazie danych. **Rozszerzenia:** 1a. Wystąpił problem z logowaniem a. Użytkownik ponownie próbuję się zalogować. 2a. Nie ma żadnej postaci. a. Użytkownik tworzy nową postać (patrz scenariusz „Stwórz nową postać”) 5a . Wystąpił problem z połączeniem a. Użytkownik zostaje poproszony o ponowne potwierdzenia zmian
Nazwa: Zacznij grać **Aktorzy:** Zalogowany użytkownik	**Główny scenariusz:** 1. Użytkownik poprawnie się loguje 2. Użytkownik wybiera tutorial lub online grę 3. Użytkownik zaczyna grać. **Rozszerzenia:** 2a. Użytkownik wybiera tutorial. a. Bezzwłocznie zaczyna grę. 2b. Użytkownik wybiera online. a. Tworzy lub wybiera nową postać. b. Zaczyna grę.

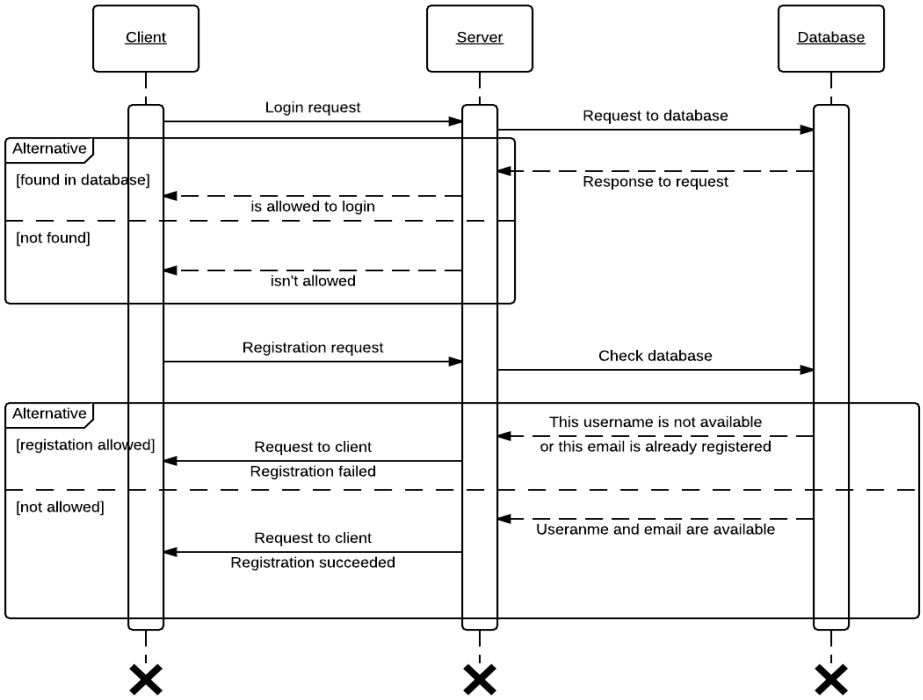
2.1.3 Diagram stanów - serwer



Rys. 4: Diagram stanów

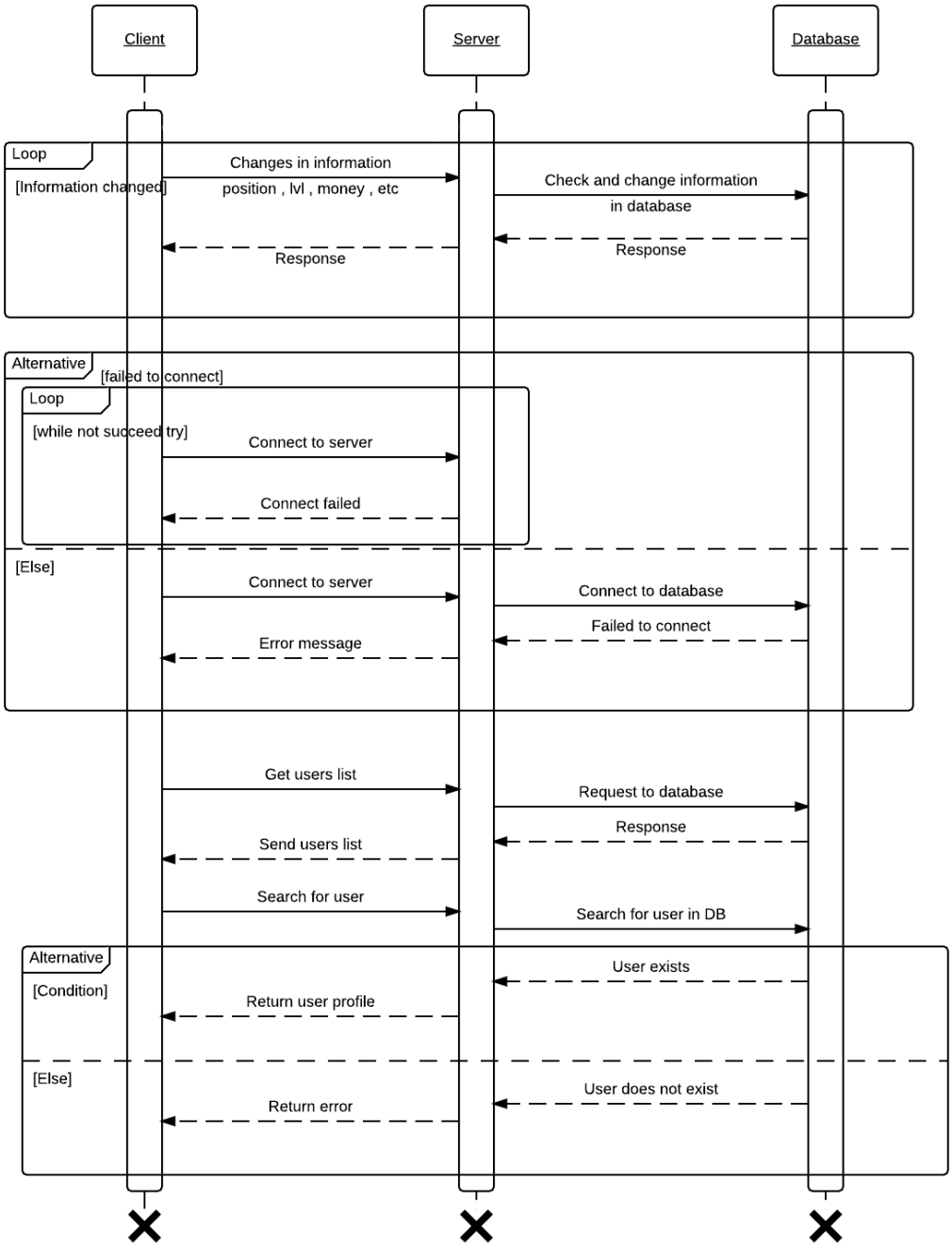
Najlepszym sposobem na scharakteryzowanie tego że w systemie zachodzą zmiany jest to że obiekty systemu zmieniają stan. Zmiana stanów obiektów jest pokazana na Rys.4. Na diagramie są opisane prawie wszystkie możliwe zmiany stanów, które są związane z serwerem.
W zależności od tego czy użytkownik chce się zalogować, czy chce zacząć grę, serwer zmienia swój stan w odpowiedzi na te zdarzenia. Podczas gry żaden obiekt nie zmienia swojego stanu, tylko zaktualizowane dane automatycznie zapisują się do bazy danych.

2.1.4 Diagram sekwencji - serwer



Rys. 5: Diagram sekwencji - Logowanie i rejestracja, część 1

Diagram sekwencji(przebiegu) pokazuje jak , w zależności od czasu, odbywa się komunikacja między obiektami. Na przedstawionym powyżej diagramie opisane są komunikacje odbywające się między klientem , serwerem i bazą danych podczas logowania i rejestracji. Diagram opisuje to co już było wcześniej rozpatrzone, ale pod innym kątem co pozwala lepiej zrozumieć działanie systemu(w tym przypadku serwera).

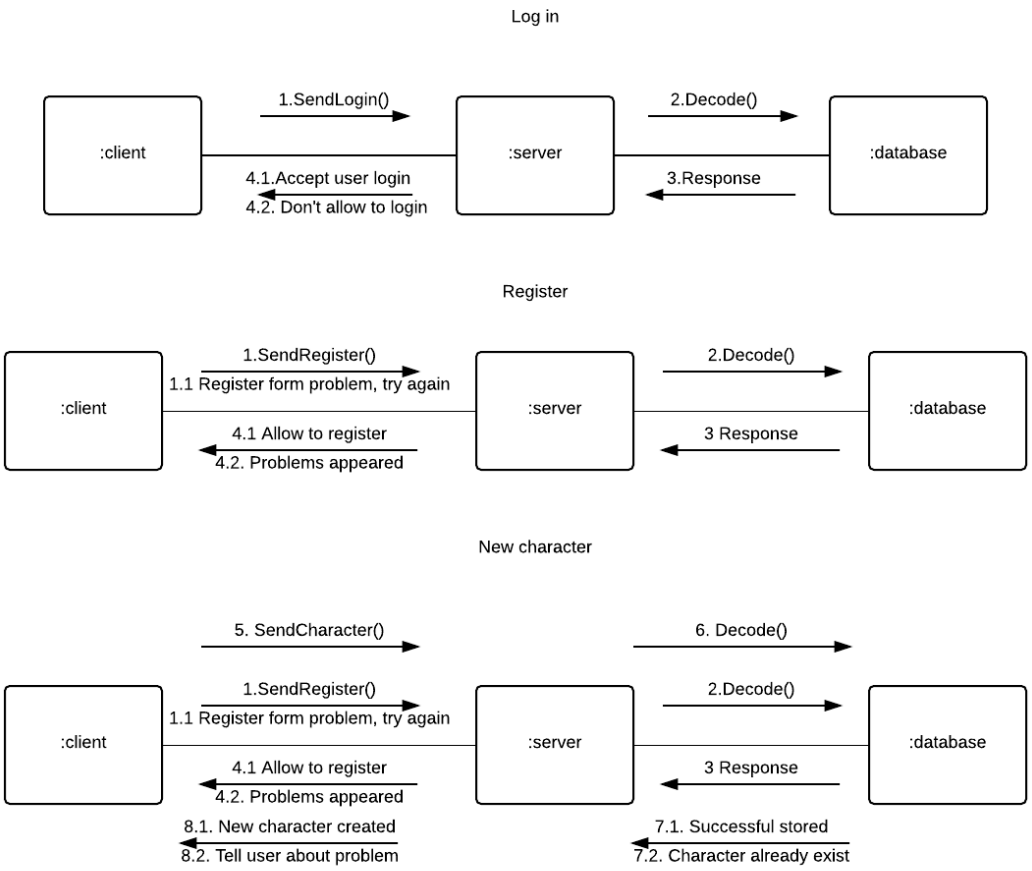


Rys. 6: Diagramy sekwencji - łączenie się z bazą danych, zmiana informacji i dodawanie przyjaciół, część 2

Diagram przedstawiony na Rys.6 przedstawia komunikacje odbywające się podczas edytowania

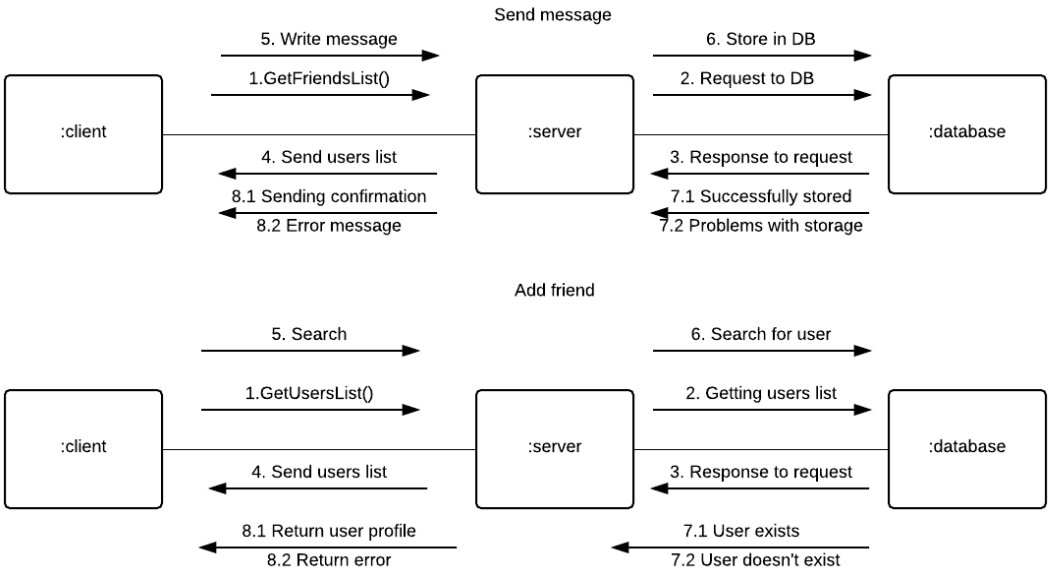
informacji, łączenia się z serwerem i dodawania nowych przyjaciół. Łączenie się z serwerem i bazą danych odbywa się bezpośrednio, po każdym niepowodzeniu występuje próba ponownego łączenia się z serwerem/bazą danych. Po pewnym czasie klient/serwer przestaje próbować się połączyć z serwerem/bazą , co nie pozwala programowi się zapętlić. Dodając przyjaciół użytkownik wyszukuje ich wpisując imię użytkownika, po czym serwer zwraca wynik wyszukiwania. Jeśli użytkownik o takim imieniu użytkownika istnieje to może być dodany do przyjaciół

2.1.5 Diagram komunikacji - serwer



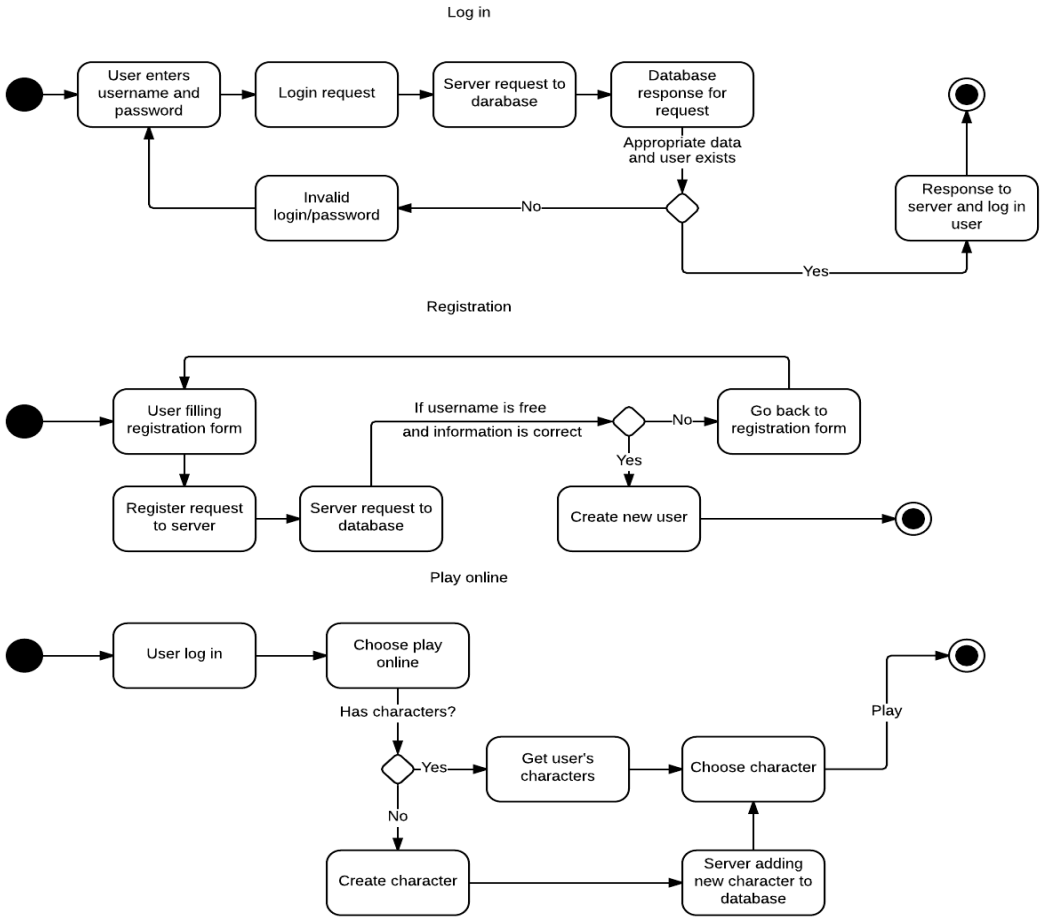
Rys. 7: Diagramy komunikacji - logowanie , rejestracja i tworzenie nowej postaci, część 1

Diagram komunikacji oprócz powiązań między obiektami pokazuje jeszcze komunikaty przesyłane między nimi. Diagramy te pozwalają na sortowanie komunikatów otrzymywanych przez obiekt. Na diagramach powyżej możemy zobaczyć jak obiekty komunikują się między sobą podczas logowania , rejestracji lub tworzenia nowej postaci. Wszystkie komunikaty są oznaczone liczbami w kolejności ich przesyłania między obiektami. Najwięcej komunikatów widzimy między klientem, a serwerem, gdyż baza danych uczestniczy w opisanych komunikacjach tylko poprzez przesyłanie danych do serwera. Na Rys. 8 przedstawiłem komunikowanie się obiektów podczas wysyłania wiadomości lub dodawania przyjaciół.



Rys. 8: Diagramy komunikacji - wysyłanie wiadomości i dodawanie przyjaciół, część 2

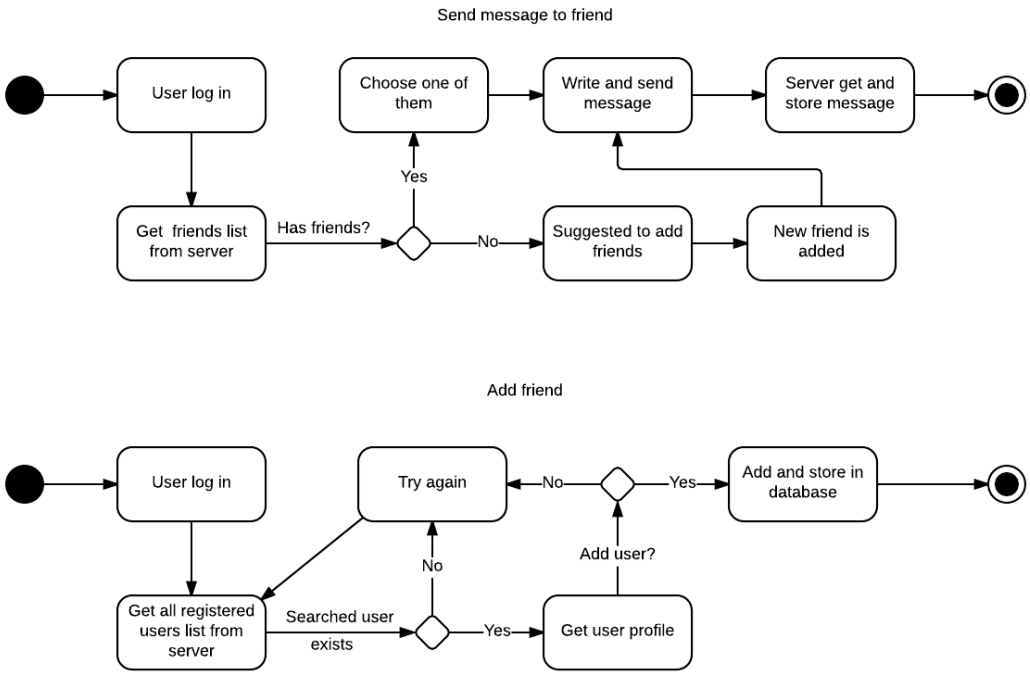
2.1.6 Diagram czynności – serwer



Rys. 9: Diagramy czynności - część 1

Diagram czynności opisuje kroki, punkty rozgałęzienia i podejmowania decyzji. Diagram czynności to rozwinięcie diagramu stanów, gdzie czynności były opisane tylko napisami na strzałkach. Więc dla lepszego zrozumienia tego jak działa system, diagramy czynności są niezbędne.

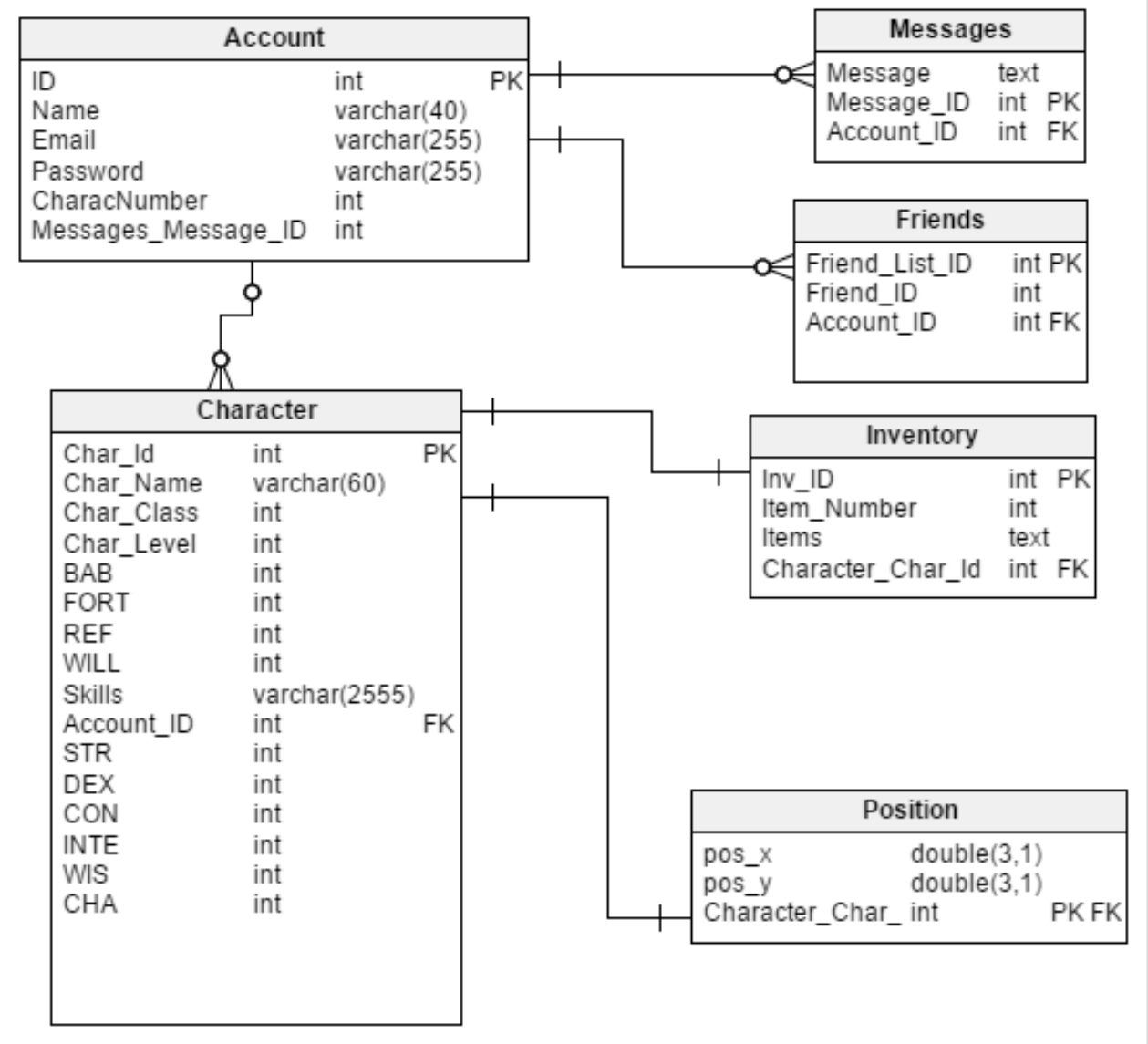
Na diagramach powyżej są opisane rejestracja, logowanie i gra. Możemy zobaczyć jakie decyzje może podjąć użytkownik i jakie możliwości on otrzymuje po podjęciu tej czy innej decyzji.



Rys. 10: Diagramy czynności - część 2

Na Rys. 10 opisałem wysyłanie wiadomości do przyjaciół i dodawanie nowych przyjaciół. Tak jak potrzebne dane(wszyscy zarejestrowani użytkownicy i lista przyjaciół użytkownika) są zapisane w bazie danych to każdy raz klient otrzymuje wszystkie dane z serwera.

2.2 Model bazy danych



Rys. 11: Model bazy danych

W bazie przechowujemy informację o koncie użytkownika. Razem z tym utrzymujemy w bazie informację o otrzymanych i wysłanych wiadomościach i listę przyjaciół użytkownika. Jak widać na Rys. 11 każdy użytkownik ma postać . Przechowujemy w bazie informacje o postaciach (charakterystyki, pozycję i przedmioty postaci). W bazie przechowujemy tylko niezbędną informację co pozwala bazie obsługiwać więcej żądań jednocześnie(tak jak część informacji zostaje po stronie klienta i klient nie musi łączyć się z bazą dla jej otrzymania).

3.Szczegóły techniczne

Podczas pisania tej pracy korzystałem z Unity 4.6(obiekty i interakcje w grze). Dla tworzenia bazy wykorzystałem MySql. Serwer został napisany w języku programowania Java. Dla nawiązywania połączenia z bazą wykorzystałem biblioteki Netty i JDBC connector. Gra ma bardzo niskie wymagania sprzętowe.

Wymagania sprzętowe aplikacji:
Procesor: dowolny procesor Dual Core, minimum 1,2 GHz
RAM: minimum 512 MB
Grafika: karta graficzna z minimum 256 MB pamięci RAM oraz obsługująca DirectX w wersji minimum 9c
HDD: minimum 1GB miejsca dostępnego na dysku
Pozostałe: dla trybu multiplayer wymagane stałe połączenie internetowe, klawiatura, myszka

Bibliografia

[1] Netty http://netty.io/
[2] MySql Connector http://www.mysql.com/products/connector/
[3] Unity http://unity3d.com/
[4] Joseph Schmuller „UML dla każdego” tłumaczenie Krzysztofa Masłowskiego , Gliwice 2003
[5] Lucidchart https://www.lucidchart.com/
[6] Vertabelo https://my.vertabelo.com/