Uniwersytet Jagielloński
Wydział Informatyki i Matematyki
INSTYTUT INFORMATYKI I MATEMATYKI KOMPUTEROWEJ
Studia stacjonarne

Nr albumu: 1088807 Ocena:

Damian Misiura

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

Opiekun pracy licencjackiej:

Dr Marcin Żelawski

Kraków 2015

Abstrakt

Celem tej pracy będzie ukazanie budowy aplikacji klienckiej do gry typu massively multiplayer real-time strategy, czyli do gry wieloosobowej strategicznej czasu rzeczywistego Teiru MMORTS. Aplikacja kliencka jest niezbędna do rozpoczęcia rozgrywki oraz komunikacji z serwerem gry. Dzięki niej użytkownik może rozpocząć interakcję z innymi graczami tejże gry, walczyć z nimi, tworzyć nowe zorganizowane grupy. Każdy gracz ma możliwość tworzenia nowych elementów tej gry, na przykład mapy bądź przedmiotów. Gra oferuje rozwój postaci, wykonywanie ciekawych zadań, podróż po świecie gry. Aplikacja kliencka powinna realizować wyświetlanie wszystkich z powyżej wymienionych funkcji. Musi też zagwarantować stabilność połączenia z serwerem oraz przeprowadzenie rozgrywki bez problemów.

The goal of this work is to show the construction of the client application to game named Teiru MMORTS, which type of massively multiplayer real-time strategy. The client application is necessary to start the game and communication with game server. Thanks to it, user can start interaction with another players of this game, fight with them, create new organized groups. Every player has the ability to create new elements of this game, such as new maps or new items. The game itself offers character development, interesting tasks to perform, ability to make journey through the world of the game. Client application should pursue to display all of the above functions. It has to ensure stability of connection with server and perform the game without problems.

Spis treści

1. Wstęp

Tematem pracy jest stworzenie aplikacji klienckiej dla gry typu MMORTS, czyli gry strategicznej czasu rzeczywistego dla wielu graczy.

Aplikacja ta powinna udostępniać użytkownikowi rozgrywkę w czasie rzeczywistym wraz z innymi graczami. Gracze powinni mieć możliwość odczucia dynamiki gry oraz uczucia ciągle rozwijającej się akcji. Aplikacja powinna udostępnić użytkownikowi możliwość tworzenia nowych map bądź nowych przedmiotów.

Gry MMORTS ostatnimi latami przyciągają ogromną ilość graczy z powodu dynamicznej rozgrywki, trudności polegającej na szybkości podejmowanych decyzji bądź rywalizacji pomiędzy nimi.

Celem tej pracy jest ukazanie procesu budowy tejże aplikacji, schematu jej działania oraz efektów jej pracy. Aplikacja jest budowana na bazie pakietu narzędzi Unity oraz technologii .NET. Wykorzystywane w niej są algorytmy bezpiecznego połączenia z serwerem oraz algorytmy typu znajdowanie najkrótszej drogi bądź optymalnego zarządzania obiektami w grze.

Pierwszy rozdział opisywać będzie czym jest język formalny modelowania UML oraz z jakich części jest zbudowany.

Drugi rozdział

Trzeci rozdział będzie opisywać budowę modelów aplikacji klienckiej, ich struktury oraz celu zastosowania odpowiednich modelów.

Czwarty rozdział opisywać będzie część logiczną aplikacji, strukturę modułów zbudowanych w tejże apliacji oraz ich wykorzystanie.

Piąty rozdział będzie pokazywał proces implementacji aplikacji klienckiej, ukazane będą fragmenty aplikacji które będą miały kluczowe znaczenie do działania całego systemu. (semestr 2)

Szósty rozdział opisywał będzie użyte zewnętrzne materiały do budowy aplikacji, użyte algorytmy, grafiki bądź frameworki oraz wymagania systemowe gry. (częściowo semestr 2)

Siódmy rozdział opisywał będzie zewnętrzne książki, artykuły bądź prace naukowe z który zaczerpana została wiedza do napisana tej pracy.

2. Część teoretyczna UML

2.1. Czym jest UML?

UML (ang. Unified Modeling Language) [1] jest ustandaryzowaną notacją modelowania obiektów w świecie rzeczywistym, nazywany jest też językiem formalnym do modelowania tychże obiektów. Notacja została stworzona poprzez złączenie trzech notacji opartych o obiektowość oraz metodologii analizy:

- Metodologia Grady Booch'a do opisywania zbioru obiektów i zależności między nimi
- Podejście Ivara Jacobson'a które zawierające też metodę przypadków użycia
- Object-Modeling Technique Jamesa Rumbaugh'a

Tworzenie projektu w oparciu o język UML jest pierwszym krokiem do tworzenia projektów opartych o projektowanie oparte o metodykę obiektową.

Inne pomysły również zostały włączone to standardu, co było zasługą Boocha, Rumbaugha, Jacobsona oraz innych osób pracujących dzięki sponsorowaniu przez Rational Software. UML został dopracowany i zaakceptowany przez Object Management Group (OMG). Obecnie UML jest wykorzystywany przez prawie każdego największego dostawcę oprogramowania na świecie, wliczając w to IBM lub Microsoft.

2.2. Praca w UML

W notacji UML modele bądź obiekty przedstawiane są jako różnego rodzaju diagramy. Reprezentowane są one przez obiekty które przedstawione są jako kształty bądź obrazki zawierające odpowiednią treść. Wyróżnia się wiele rodzajów diagramów [2]:

2.2.1. Diagram przypadków użycia oraz scenariusze przypadków użycia

Diagram przypadków użycia opisuje zachowanie systemu z punktu widzenia użytkownika. Dla deweloperów oprogramowania ten typ diagramów okazuje się być bardzo przydatnym narzędziem, dzięki niemu metodą prób i błędów można zobaczyć założenia użytkowników

wobec tworzonego systemu. Diagramy te służą ukazaniu tego co chce użytkownik wobec systemu.

Scenariusz przypadków użycia pokazuje oczekiwanie użytkownika względem konkretnie wykonywanej akcji, proces jej wykonywania oraz etapy w których owa akcja jest wykonywana. Przykładem jest proces kupna jachtu.

2.2.2. Diagram stanów

Diagram stanów opisuję zmianę stanu obiektów względem wykonywanej akcji. Z wykonaniem każdego etapu tej akcji zmienia się stan obiektu bądź modułu systemu, ich zmiany pokazane są na diagramie. Dla programistów jest to ważny diagram ponieważ mogą zauważyć zmianę zachowania obiektu wobec wykonywanej czynności na nim.

2.2.3. Diagram klas

Diagram klas opisuje wszystkie obiekty tworzone w obecnym module, ich powiązania, na przykład zawieranie się w sobie bądź związki dziedziczenia. Obiekty mają określone typy, funkcje oraz metody. Każdy element modułu przedstawiany na diagramie ma określoną formę, nie opisywane są konkretne obiekty ale jedynie ich abstrakcyjne formy. Dla programisty jest to szczególnie ważne ponieważ diagram ten pokazuje mu jak zbudowany jest system i jakie są w nim zależności. Również dzięki niemu programista jest w stanie łatwo zaimplementować te powiązania między obiektami.

2.2.4. Diagram czynności

Diagram czynności pokazuje jak zachowuje się system w odniesieniu do obecnie wykonującej się operacji.
Jest bardzo podobny do diagramu stanów, jednak zamiast opisywać zachowanie jednego obiektu opisuje zachowania wielu obiektów, między którymi nawiązywana jest komunikacja. Pokazuje on programistom sposób działania pewnych czynności, ułatwiając implementację opisywanej diagramem czynności.

2.2.5. Diagram sekwencji

Diagram sekwencji prezentuje kolejność wykonania etapów pewnej akcji, przepływ sterowania informacjami pomiędzy modułami systemu oraz szablon opisywanego algorytmu. Nie jest uwzględniany dostęp do danych, operacje na danych oraz innych funkcji związanych z komunikacją.

2.2.6. Diagram komunikacji

Diagram skupia się na obiektach które wchodzą w skład interakcji oraz komunikatach wymienianych przez nich. Aspekt czasowy ma małe znaczenie, z tego powodu umieszczenie obiektów na diagramie ma cel łatwego opisania relacji między nimi. Diagram opisuje w jaki sposób komponenty komunikują się ze sobą podczas wykonywania danej akcji.

2.2.7. Diagram komponentów

Diagram komponentów pokazuje podział systemu na mniejsze podsystemy logiczne. Podsystemy te są połączone między sobą poprzez wspólne funkcje bądź wspólne zapotrzebowanie na daną porcję danych. Komponent to fizyczny moduł kodu, zwykle pakiet. Musi on posiadać swoją unikalną nazwę. Można tworzyć grupy komponentów zwane pakietami, które mogą zostać podsystemami.

2.2.8. Diagram wdrożenia

Diagram wdrożeń pokazuje powiązania pomiędzy oprogramowaniem a sprzętem fizycznym. Wykorzystywane są przy modelowaniu dużych systemów informatycznych.

3. Funkcjonalna charakterystyka systemu

System Teiru MMORTS będzie składał się z trzech części:

- Aplikacji klienckiej
- Serwera gry
- Bazy danych

Wyżej wymieniona baza danych będzie podzielona między serwer a klienta. W tym rozdziale bardziej skupię się na aplikacji klienckiej będącej tematem tej pracy, jednak postaram się opisać zasadę działania całego systemu.

3.1. Charakterystyka gry

Budowany system jest grą strategiczną czasu rzeczywistego skierowaną do rozgrywek dla dużej ilości graczy. Sama gra będzie oferować tryb pojedynczego gracza (single player mode) oraz tryb wielu graczy (multi player mode). Tryb pojedynczego gracza ma za zadanie zapoznać nowego z założenia gracza z zasadami gry, pokazać mu podstawowe mechanizmy gry oraz przygotowanie do rozgrywek trybu multi player. Sam tryb multi player umożliwia stworzenie nowej postaci, dostosowanie jej statystyk do własnych potrzeb.

Zalogowanie się do systemu jest warunkiem koniecznym do rozpoczęcia jakiegokolwiek trybu rozgrywki, tworzenia postaci bądź rozmowy z innymi graczami.

Rozgrywka odbywa się na zasadzie rund, gdzie każdy ruch gracza bazowany jest na rzucie kostką. Rundy są ograniczone czasowo aby zachować płynność rozgrywki oraz zapewnić brak celowego wstrzymywania rozgrywki. W każdej rundzie gracz może przesuwać się po mapie, prowadzić interakcje z innymi graczami, statycznymi postaciami na mapie, wykonywać zadania, prowadzić walki.

Każdy gracz ma możliwość stworzenia gildii, to znaczy grupy osób zrzeszonej w tej samej grupie, które mają możliwość rozmowy w prywatnym kanale bądź braniu udziału w walkach pomiędzy innymi gildiami. Gracze mają również tworzenia listy przyjaciół w grze, dzięki której możliwe będzie sprawdzenie obecnego stanu przyjaciela lub prywatnej rozmowy z nim.

Postać stworzona przez gracza może zostać wybrana z jednej z czterech dostępnych klas. Są to Ranger, Mage, Knight oraz Mystiq. Każda klasa postaci ma swoje zalety jak i wady, każda też wyróżnia się stylem walki. Przykładowo Mage włada magią w celach ofensywnych, Knight z drugiej strony zawiera dużą ilość obrony i walczy wręcz. Postaci z czasem mogą awansować na wyższe poziomy, aby odblokować nowe zaklęcia, umiejętności oraz rozdzielić nowo dobyte punkty statystyk. Statystyk w grze jest sześć, są to:

- Strength – zwiększa możliwy ładunek gracza oraz ilość obrażeń zadawanych w walce wręcz
- Dexterity – określa skuteczność ataków dystansowych oraz ich obrażenia
- Constitution – wytrzymałość, określa ilość punktów życia postaci
- Intelligence – zwiększa ilość punktów statystyk i umiejętności przyznawanych na każdy nowy poziom
- Wisdom – mądrość, zwiększa siłę zaklęć
- Charisma - zwiększa odporność na magiczne obrażenia

Gracz pierwszopoziomowy otrzymuje wartość 8 każdej ze statystyk, aby zwiększyć każdą z
nich należy rozdysponować punkty statystyk otrzymywane za każdy nowy poziom. Kolejno 1 punkt statystyki od 9 do 14, po 14 punkcie statystyk potrzebne są 2 punkty aby awansować. Co każdy poziom postać ma możliwość wyboru nowych zaklęć bądź indywidualnych umiejętności, wyspecyfikowanych dla każdej klasy.

Postać ma możliwość tworzenia ekwipunku, zarządzania nim, zbierania nowych przedmiotów z zabitych potworów, nakładania nowych elementów ekwipunku, kupna nowych przedmiotów oraz sprzedawania niepotrzebnych przedmiotów.

Gra posiada czat, w którym wszyscy gracze mogą rozmawiać ze sobą. Czat składa się z podczatów: czatu gry, czatu ogłoszeń bądź czatu logu. Gracze mogą tworzyć własne kanały do rozmowy poprzez odpowiednią akcję w grze. Gildie również posiadają swój odrębny czat, tak samo jaki walki.

Walki w grze dzielą się na trzy rodzaje:

- PvE – gracz kontra środowisko, gdzie walka odbywa się pomiędzy stworzenia ze świata gry
- PvP – gracz kontra gracz, pojedynek przeciwko innemu graczowi
- GvG – gildia kontra gildia, walka 4 graczy na 4 graczy zwaśnionych gildii

Walki PvE oznaczają walkę przeciw stworzeń ze świata gry, walka ta może być przykładowo wyznaczona przez zadanie, rozpoczęta losowo bądź znaleziona poprzez losowe wydarzenie na mapie. Nagroda za nie jest duża ilość doświadczenia, losowa ilość przedmiotów bądź nowe elementy ekwipunku.

Walki PvP oznaczają pojedynki między graczami, odbywają się one na zasadzie jeden na jednego. Nagrodą za nie może być nowa pozycja w globalnym rankingu, złoto, przedmioty bądź jakiś element ekwipunku przeciwnika oraz wieczna chwała.

Walki GvG określają walkę pomiędzy dwiema gildiami. Toczą się na zasadzie 4 graczy na 4 graczy, przy czym nie ma możliwości aby walki posiadały mniej lub więcej graczy. Nagrodą za

walkę jest złoto, przedmioty i ekwipunek podzielony na zwycięska drużynę, nowa pozycja rankingowa gildii.

Sama walka odbywa się turowo, początkowo tworzona jest kolejka postaci wykonujących ruch względem ich inicjatywy. Następnie kolejka ruchów cyklicznie przesuwa się do momentu aż postaci jednej frakcji zostaną w tej kolejce eliminując drugą stronę.

Walka zawiera typy akcji, a więc: akcje standardową, czyli np. atak bądź ruch po mapie. Akcja ruchu oznacza poruszanie się po mapie przez postać, może być więc akcją standardową. Akcja całorundowa to skupienie całej tury postaci na jednym zadaniu zamiast przykładowo na ruchu i następującym ataku.

Podczas ataku wykonywany jest test ataku aby sprawdzić czy próba ataku się powiodła, oceniane jest to przez rzut kostką K20. Jeżeli atak się powiedzie (czyli rzut kostką będzie korzystny) następuje obliczenie otrzymanych obrażeń przez cel, kostka tutaj również ma wpływ. Minimalne obrażenia zawsze wynoszą 1, maksymalne są obliczane przez modyfikatory typu umiejętności, statystyk bądź broni lub zaklęcia. Postaci mogą być w różnych stanach w zależności od otrzymanych obrażeń, np. Zdrowy, Okaleczony, Martwy.

Rozgrywka całej gry oparta jest głównie na walkach, dzięki czemu gracz ma możliwość zaobserwowania dynamiki całej akcji mającej miejsce w grze.

3.2. Charakterystyka systemu informatycznego

System jest podzielony na część serwerową, kliencką oraz na bazę danych.

Z założenia gier typu MMORTS serwer powinien obsługiwać połączenie dużej ilości graczy. Jako przewidywana ilość osób połączonych jednocześnie uznano około 50-100 osób. Serwer ten powinien reagować na każdą zmianę stanu gry zgłoszoną przez aplikację kliencką. Musi też reagować w przypadku sytuacji wyjątkowych typu utrata połączenia, atak na serwer bądź nieoczekiwany błąd obliczeń i tym podobnych.

Baza danych w tym systemie przechowuje głównie dane dotyczące graczy, ich ekwipunku, obecnych statystyk. Posiada też strukturę wszystkich przedmiotów dostępnych w grze, typu przeciwników, zadań oraz wszystkich obiektów na mapie. Baza ta jest podzielona pomiędzy serwer i klient z zaznaczeniem że obydwie części systemu posiadają dwie kopie tej samej części bazy – o typach przeciwników, przedmiotach itp. W przypadku awarii systemu aplikacja kliencka tworzy lokalną kopie bazy którą przy wznowieniu połączenia z serwerem stara się dołączyć do bazy serwerowej.

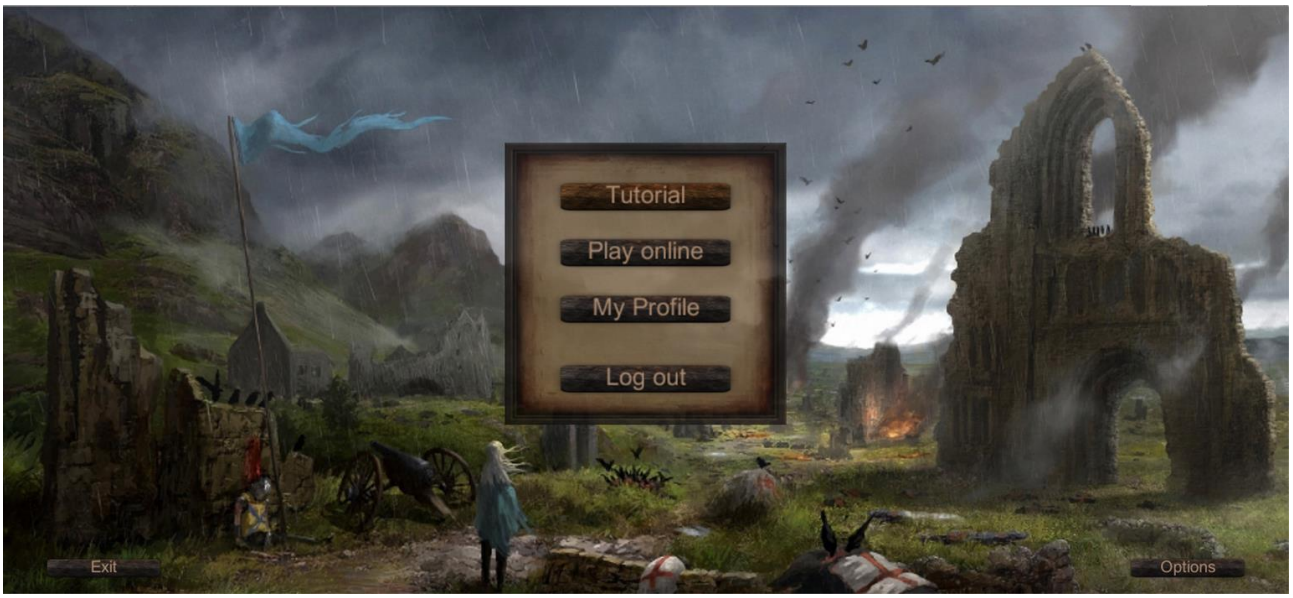
Aplikacja kliencka ma za zadanie odczytywać każde żądanie klienta, przetwarzać je oraz wykonywać odpowiednie polecenie względem wskazanego żądania. Gracz powinien mieć

możliwość doboru wybranego przez niego trybu gry wraz z opcjami występującymi w nim. Klient ma też za zadanie zaprezentować w sposób wizualny działanie silnika graficznego gry. Aplikacja kliencka powinna się również dostosować do mocy obliczeniowej komputera gracza pokazując mu odpowiednie opcje wyboru ustawień graficznych, oprawy muzycznej bądź sterowania.

Serwer wraz z klientem łączyć się będą połączeniem szyfrowanym dla zachowania poufności danych, dane klienta zarówno na serwerze jak i na aplikacji klienckiej powinny zostać zachowane w sposób zaszyfrowany. System może zostać zmodyfikowany przez użytkownika poprzez dodanie nowych map za pomocą edytora map bądź nowych przedmiotów za pomocą edytora przedmiotów.

4. Warstwa logiczna aplikacji klienckiej - projektowanie

Aplikacja kliencka została napisana w oparciu o środowisko programistyczne Unity w wersji 4.7.1., w środowisku tym została stworzona cała logika menu głównego aplikacji wraz z warstwą reprezentacji.



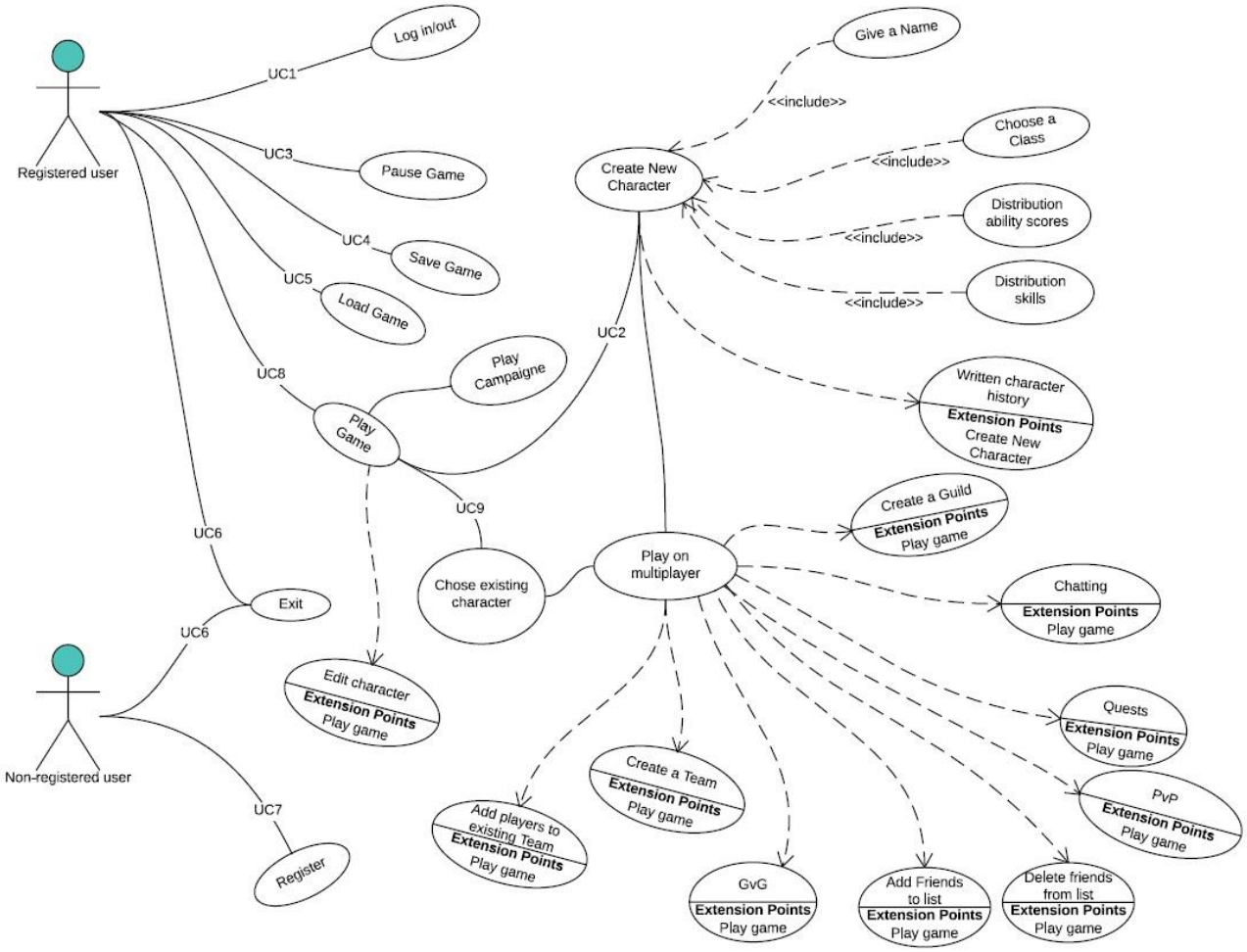
Rysunek 1- Ekran menu głównego gry

Aplikacja zawiera menu główne posiadające opcje wyboru samouczka, gry sieciowej, swojego profilu gry, wylogowania się. W dolnych rogach menu głównego można zauważyć opcję wyjścia z gry oraz zmiany opcji gry.

W opcjach gry użytkownik może zmienić swoje ustawienia gry jak na przykład ustawienia graficzne, ustawienia dźwiękowe bądź ustawienia sterowania myszką lub klawiaturą.

W ustawieniach profilu użytkownik może zmienić swój nick, swój obrazek, może edytować stworzone postaci, zarządzać gildią.

4.1. Przypadki użycia aplikacji klienckiej



Rysunek 2 - Przypadek użycia aplikacji klienckiej przez użytkownika

Użytkownik zarejestrowany bądź niezarejestrowany może prowadzić interakcje z aplikacją kliencką. Użytkownik niezarejestrowany ma możliwość rejestracji bądź wyjścia z gry, funkcje gry są zablokowane dla niego do momentu rejestracji.

Gracz który pomyślnie się zalogował może zacząć grę, wybrać tryb gry (single player lub multi player) i rozpocząć grę. Przy wyborze trybu pojedynczego gracza gracz może rozpocząć kampanie przygotowującą go do gry wieloosobowej. Jeśli wybierze tryb wieloosobowy musi posiadać do gry stworzoną postać którą może stworzyć teraz bądź wybrać istniejącą postać. Po czym rozpoczyna normalne czynności związane z grą wieloosobową na przykład rozmowa z innymi graczami, handel, polowanie na stwory na mapie gry itd..

Przykładowe scenariusze użycia gry przez użytkownika:

Scenariusz 1 – wylogowanie użytkownika z serwera

Scenariusz główny:

1. Użytkownik wybiera opcje wylogowania.
2. System prezentuje zapytanie, czy użytkownik chce dalej się wylogować.
3. System zapisuje aktualny stan gry/postaci użytkownika.
4. System wylogowuje użytkownika z gry.

Wyjątki i rozszerzenia:

1.A User chce się wylogować w tracie aktywnej walki.
- **1.A.1** System informuje gracza o problemie i wyświetla mu formularz z zapytaniem o poddanie pojedynku.
- **1.A.2** Przejdź do kroku 3.

Scenariusz 2 – stworzenie nowej postaci

Scenariusz główny:

1. Użytkownik chce stworzyć nową postać.
2. System wyświetla formularz z zapytaniem o imię postaci.
3. System sprawdza czy takie imię postaci figuruje w bazie graczy w trybie multi player.
4. User wybiera rasę, klasę oraz rozdaje punkty statystyk dla postaci.
5. System sprawdza czy gracz wydał wszystkie punkty.
6. Gracz musi wybrać umiejętności swojej nowej postaci.
7. User może napisać historię swojej postaci.

Wyjątki i rozszerzenia:

3.A User chce nadać postaci imię, które już figuruje w bazie postaci w trybie multi player.
- **3.A.1** System informuje gracza o problemie.
- **3.A.2** Użytkownik musi wybrać nowe imię dla postaci.
- **3.A.3** Przejdź do kroku 3.

5.A User nie rozdał wszystkich punktów statystyk.
- **5.A.1** System informuje gracza o problemie.
- **5.A.2** Użytkownik musi rozdać resztę punktów.
- **5.A.3** Powrót do kroku 7.

Scenariusz 3 – Rejestracja nowego użytkownika

Scenariusz główny:

1. User wybiera opcję rejestracji.
2. System prezentuje formularz rejestracji.
3. User wypełnia formularz.
4. System sprawdza, czy wszystkie wymagane pola są uzupełnione.
5. System sprawdza dane.
6. System rejestruje nowego użytkownika.

Wyjątki i rozszerzenia:

3.A Użytkownik nie uzupełnił wszystkich wymaganych pól w formularzu.
- **3.A.1** System informuje gracza o problemie.
- **3.A.2** Przejdź do kroku 3.

5.A Użytkownik próbuje zarejestrować nazwę lub adres e-mail, która już istnieje w bazie.
- **5.A.1** System informuje gracza o problemie.
- **5.A.2** Przejdź do kroku 3.

6.A Użytkownik przechodzi proces uwierzytelniania i zostaje zarejestrowany.

6.B Użytkownik nie przechodzi procesu uwierzytelnienia i nie zostaje zarejestrowany.

Scenariusz 4 – rozpoczęcie nowej gry

Scenariusz główny:

1. Użytkownik wybiera rodzaj rozgrywki.
2. Użytkownik rozpoczyna nową grę.

Wyjątki i rozszerzenia:

1.A Użytkownik rozpoczyna kampanię.

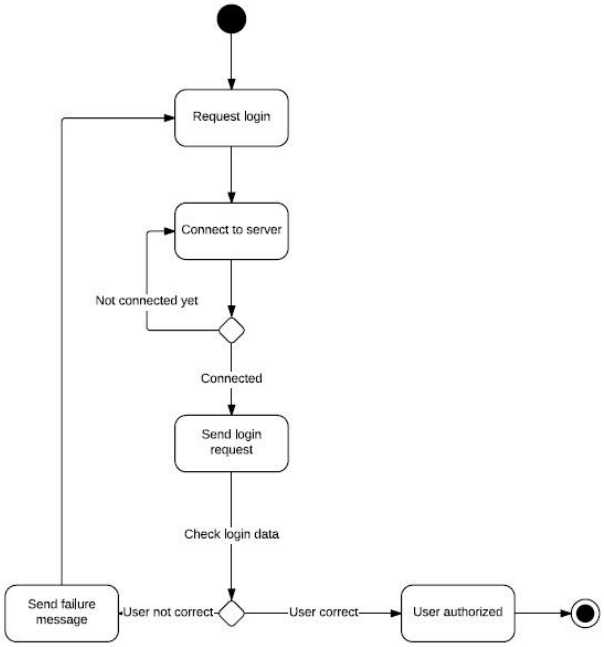
1.B Użytkownik rozpoczyna rozgrywkę w trybie multi player.
- **1.B.1** Użytkownik rozmawia z innym graczem poprzez czat.
- **1.B.2** Użytkownik tworzy drużynę.
  - **1.B.2.1** Użytkownik dodaje lub usuwa gracza z drużyny.
  - **1.B.2.2** Użytkownik proponuje pojedynki między drużynami.
- **1.B.3** Użytkownik tworzy gildię.
  - **1.B.3.1** Użytkownik dodaje lub usuwa gracz z gildii.
  - **1.B.3.2** Użytkownik proponuje pojedynki między gildiami.
- **1.B.4** Użytkownik pojedynkuje się z innym graczem.
- **1.B.5** Użytkownik pobiera lub anuluje zadanie.
- **1.B.6** Użytkownik dodaje lub usuwa gracza z listy przyjaciół.

1.C Użytkownik tworzy nową postać.

1.D Użytkownik wybiera postać już istniejącą.

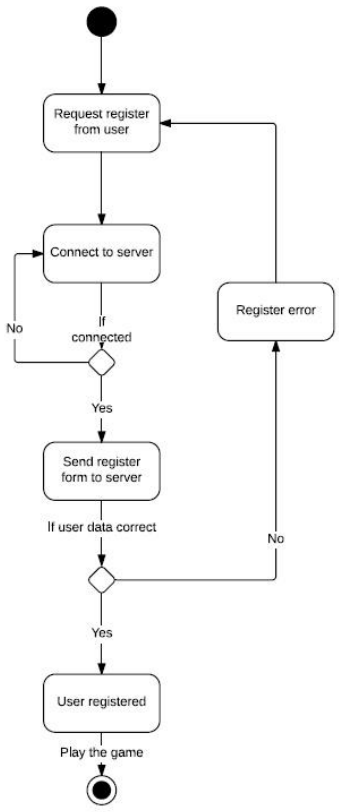
4.2. Diagramy czynności ważniejszych funkcji programu

Poniżej przedstawione diagramy opisują ważne części systemu, bez których gra wieloosobowa nie ma sensu:



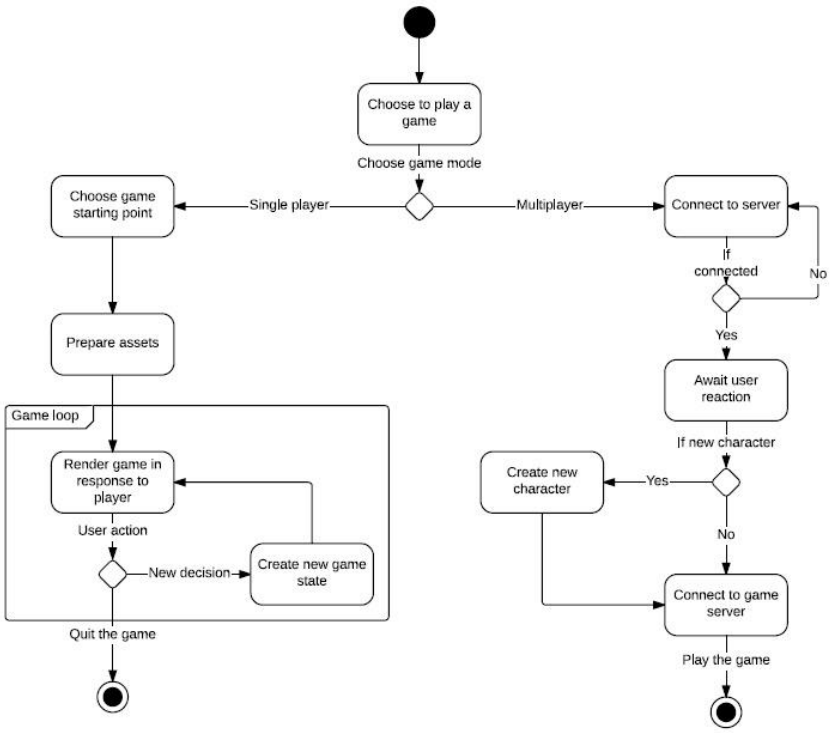
Rysunek 3 - Diagram czynności dla logowania do serwera

Diagram przedstawia proces logowania do systemu. Użytkownik żąda zalogowania, po czym aplikacja kliencka loguje się do serwera, przesyła mu żądanie logowania. Jeśli serwer odpowie błędem, operacja logowania jest przywracana do początku.



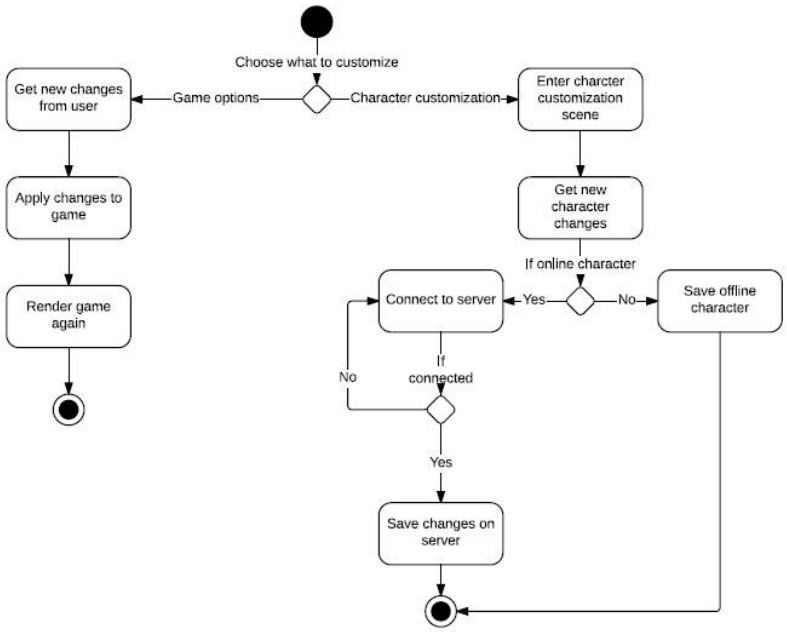
Rysunek 4 - Diagram czynności dla rejestracji nowego konta

Użytkownik żąda stworzenia nowego konta w serwisie. Aplikacja tworzy szablon żądania po czym wysyła je na serwer. Ten po odebraniu danych sprawdza czy dane sa poprawne i czy użytkownik już nie istnieje w serwisie. Jeśli nastąpił jakikolwiek błąd, serwer zwraca ten błąd i cała procedura rejestracji jest powtarzana od nowa. W przeciwnym przypadku rejestracja jest kończona sukcesem i użytkownik może korzystać z gry.



Rysunek 5 - Diagram czynności dla rozpoczęcia gry

Użytkownik ma opcję wyboru trybu gry: single player mode and multi player mode. Gdy wybierze single player mode ma możliwość wyboru punktu rozpoczęcia rozgrywki (nową kampanie bądź zapisany stan gry), następnie aplikacja kliencka przygotowuje potrzebne zasoby do stworzenia gry i tworzy pętlę gry (game loop) gdzie oczekuje na decyzje gracza i dostosowuje narysowany obraz do jego decyzji. Jeśli gracz zdecyduje się na tryb wieloosobowy, musi się połączyć z serwerem. Po nawiązaniu połączenia klient oczekuje na decyzję gracza jaką postacią rozpocznie grę. Nstępnie użytkownik jest łączony z serwerem gry i następuje rozpoczęcie rozgrywki sieciowej.

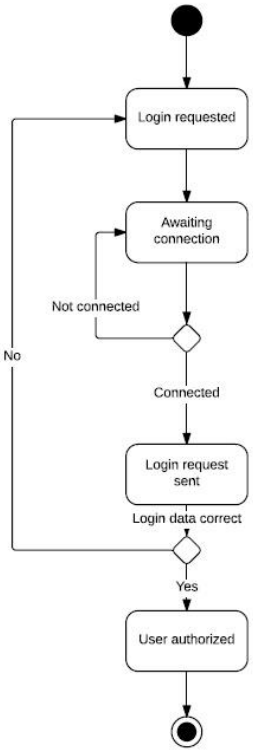


Rysunek 6 - Diagram czynności dla dostosowania elementów gry

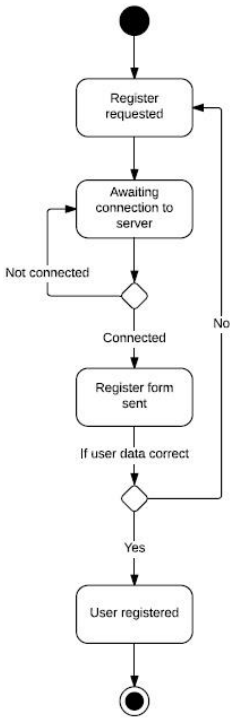
Użytkownik musi wybrać co chce dostosować w grze, jeśli to są same gry to wybiera spośród opcji w menu opcji nowe ustawienia i zapisuje je bądź anuluje. Klient wtedy odczytuje nowe ustawienia i ponownie tworzy środowisko gry dostosowane do nowych ustawień. Jeśli wybrane zostanie dostosowanie postaci wtedy użytkownik dobiera interesujące go opcje, zmienia statystyki bądź umiejętności postaci, po czym w zależności czy to postać z kampanii bądź gry wieloosobowej zmiany zostają zapisane odpowiednio na komputerze klienta bądź na serwerze gry.

4.3. Diagramy stanów ważniejszych funkcji programu

Diagramy stanów opisują stan klienta podczas wykonywania ważnych czynności w programie jest to bardzo przydatne dla programisty ponieważ pokazuje zachowanie obiektu względem czynności, co ułatwia implementację.



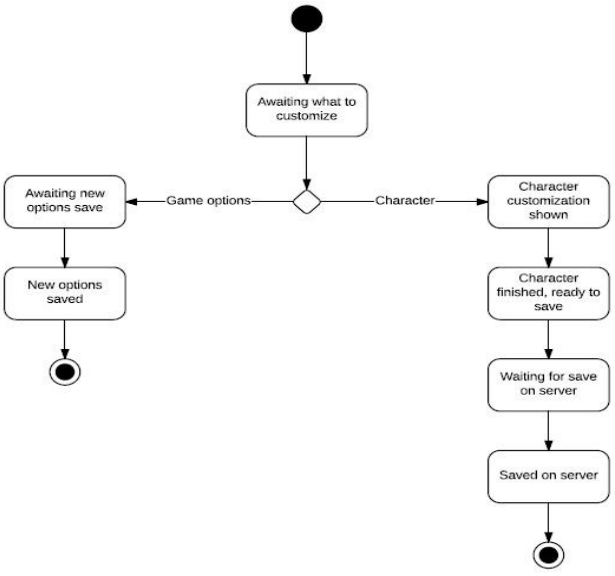
Rysunek 7 - Diagram stanów dla logowania



Rysunek 8 - Diagram stanów dla rejestracji

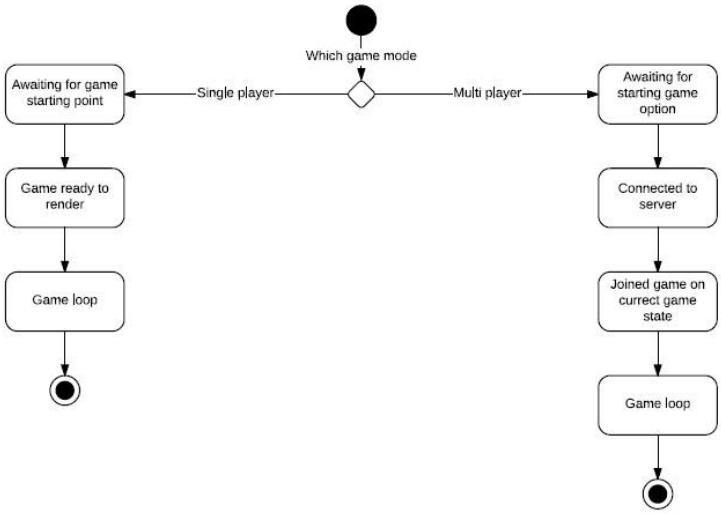
Rysunek 7 pokazuje stan klienta podczas procesu logowania. Klient pierw otrzymuje żądanie o logowanie do serwisu, po czym oczekuje na połączenie z serwerem. Aplikacja wysyła żądanie dla serwera z logowaniem i w zależności od otrzymanej odpowiedzi akceptuje użytkownika do gry bądź odrzuca go i zaczyna proces logowania od nowa.

Rysunek 8 ukazuje stan klienta podczas rejestracji nowego użytkownika. Klient otrzymuje żądanie rejestracji klienta, po czym łączy się z serwerem. Następnie zostaje wysłane żądanie o rejestrację nowego użytkownika do serwera. Jeśli się powiedzie nowy użytkownik zostaje stworzony w bazie, w przeciwnym wypadku cały proces jest ponawiany.



Rysunek 9 - Diagram stanów dla dostosowanie aplikacji

Klient oczekuje na decyzję użytkownik o tym co on chce dostosować. Jeśli wybrał opcje gry, klient oczekuje na zapisanie zmian, po czym zapisuje je i tworzy środowisko gry dostosowane do nowych ustawień. Jeśli zostało wybrane dostosowanie postaci, klient oczekuje na stworzenie zmian, po czym zapisuje zmiany na komputerze użytkownika i zapisuje zmiany na serwerze.

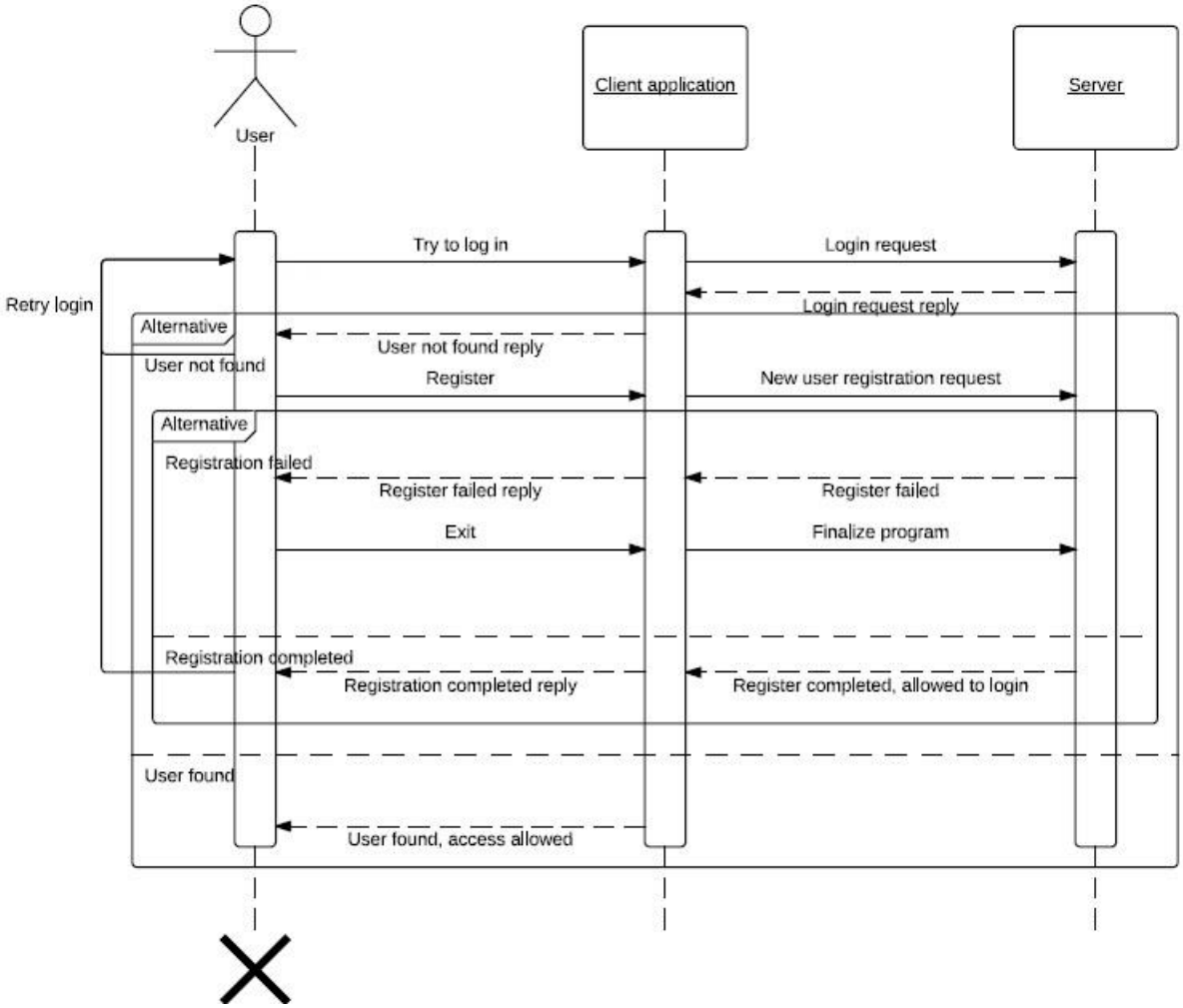


Rysunek 10 - Diagram stanu rozpoczęcia gry

Klient początkowo oczekuje na wybranie przez użytkownika trybu gry. Jeśli gracz wybrał tryb pojedynczego gracza klient oczekuje na wybranie punktu startego (nową grę lub wznowienie poprzedniej rozgrywki), po czym przygotowuje grę do działania i przechodzi do pętli gry (game loop). Jeśli wybrany został tryb wieloosobowy, gra oczekuje na wybranie postaci przez gracza, następnie łączy się z serwerem, dołącza do aktualnego stanu gry serwera. Klient przechodzi wtedy do pętli gry.

4.4. Diagramy sekwencji

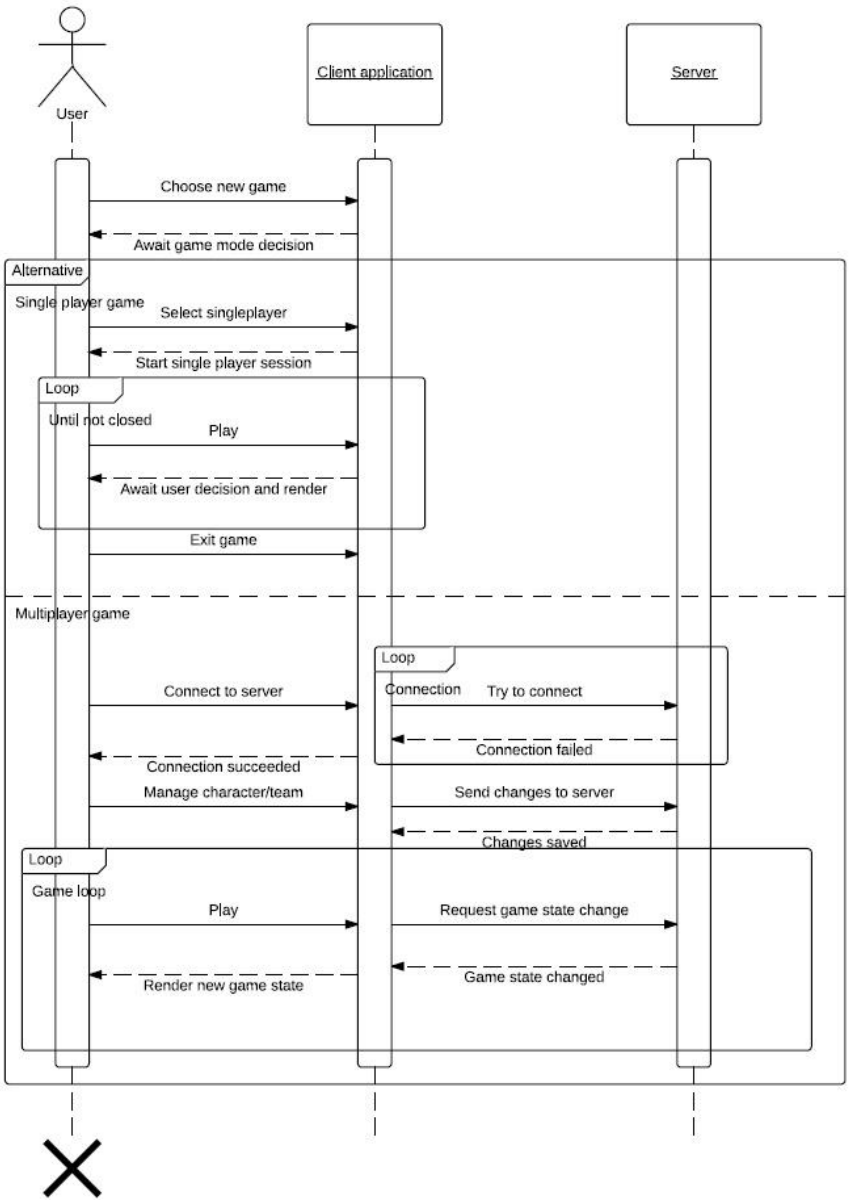
Diagramy sekwencji pokazują przepływ sterowania przez moduły programu oraz jak przepływają dane pomiędzy nimi. Programista dzięki tym diagramom potrafi odtworzyć łatwość w tworzeniu metod aby zachować opisany przepływ danych i sterowania.



Rysunek 11 - Diagram sekwencji dla logowania i rejestracji

Użytkownik najpierw próbuje zalogować się do aplikacji, co odbiera poprawnie aplikacja kliencka i przygotowuje odpowiednie żądanie dla serwera. Serwer odpowiada aplikacji, w zależności od odpowiedzi istnieją dwa przypadki rozpatrzenia sytuacji.

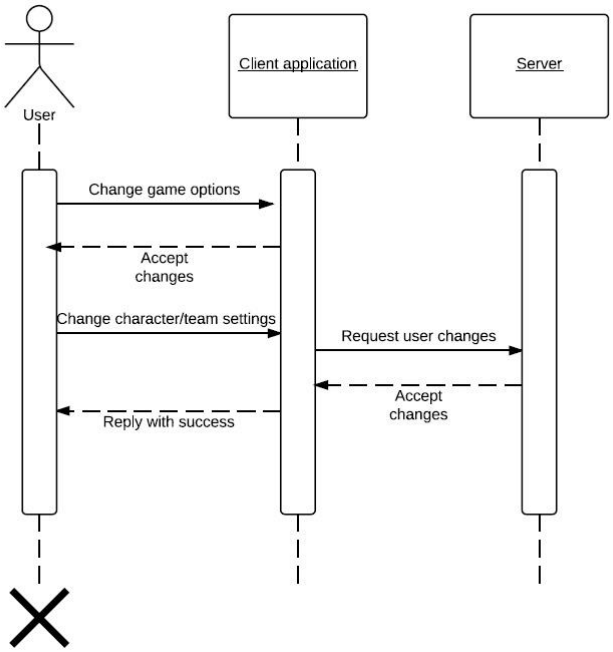
1. Użytkownik nie został znaleziony, wtedy użytkownik dostaje możliwość rejestracji na serwis. Jeśli uda mu się pomyślnie wypełnić formę rejestracyjną, aplikacja kliencka wysyła żądanie o dodanie nowego użytkownika, jeśli z jakiegoś powodu serwer odrzuci prośbę rejestracji, gracz może albo ponowić próbę lub wyjść z gry. Po pomyślnym zarejestrowaniu się użytkownik dostaje możliwość korzystania z gry na nowym koncie.
2. Użytkownik został rozpoznany i dostał możliwość korzystania z gry.



Rysunek 12 - Diagram sekwencji dla rozpoczęcia gry

Gracz początkowo wybiera nowy tryb gry, na co oczekuje aplikacja kliencka. W zależności który tryb gry wybierze klient sterowanie przeprowadzone jest w inny sposób.

1. Jeśli wybierze tryb pojedynczego gracza, aplikacja kliencka nie łączy się z serwerem a tworzy nową sesję gry single player i wykonuje pętlę gry (game loop) w której dostosowuje środowisko gry do decyzji gracza.
2. Jeśli wybrany zostanie tryb wieloosobowy aplikacja kliencka łączy gracza z serwerem i po przygotowaniu swojej postaci bądź drużyny gracz rozpoczyna rozgrywkę sieciową.

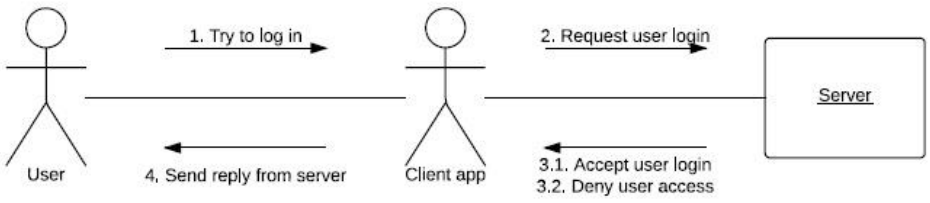


Rysunek 13 - Diagram sekwencji dla dostosowania gry

Gracz może dostosować pod siebie zarówno opcje gry jak i zmienić ustawienia swoich postaci i drużyny. W przypadku zmiany opcji gry nie dochodzi do połączenia z serwerem i zmiany są zapisywane lokalnie. W przypadku zmiany opcji drużyny bądź postaci aplikacja kliencka łączy się z serwerem aby zapisać nowe zmiany wprowadzone przez użytkownika.

4.5. Diagramy komunikacji dla ważniejszych funkcji programu

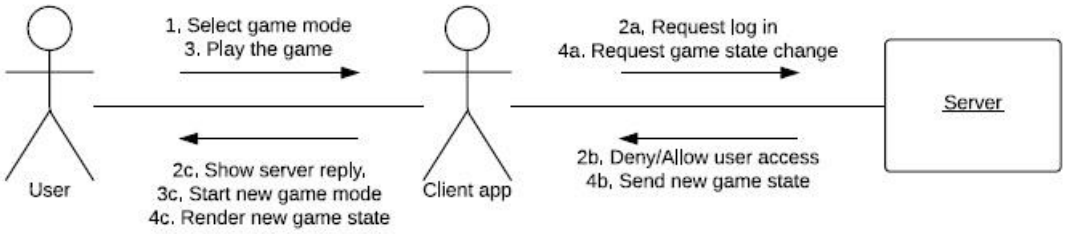
Diagramy komunikacji obrazują jak wygląda komunikacja między elementami systemu dla danej akcji. W tych diagramach zostało założone że aplikacja kliencka jest aktorem.



Rysunek 14 - diagram komunikacji dla logowania

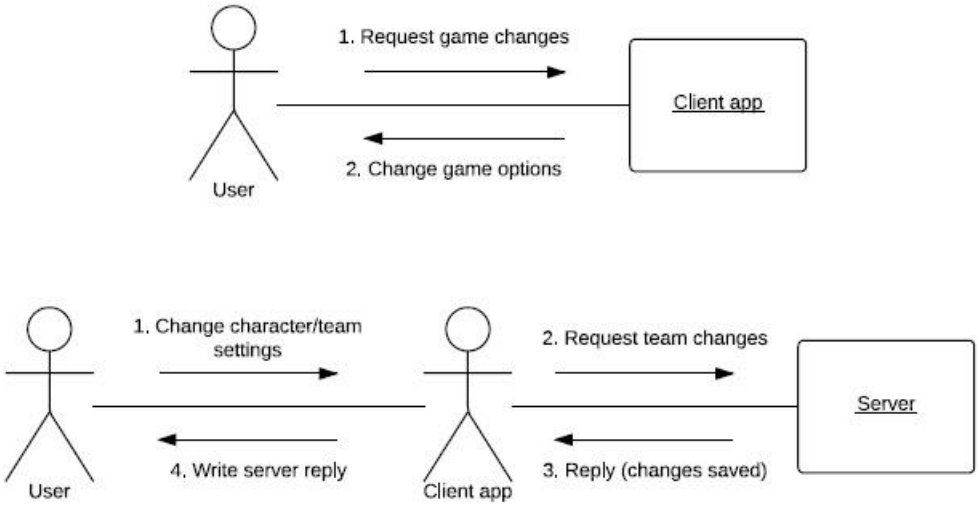
Klient przesyłając formularz logowania do klienta komunikuje się z aplikacją kliencką. Następnie zostaje stworzone żądanie logowania w programie które zostaje wy słane do

serwera, ten po odebraniu żądania wysyła odpowiedź z powrotem do aplikacji. Zostaje wyświetlony komunikat w zależności od odpowiedzi serwera.



Rysunek 15 - diagram komunikacji dla rozpoczęcia gry

Gracz komunikuje się z aplikacja poprzez wybranie trybu gry, w przypadku trybu multi player (podpunkty a,b i c) aplikacja ta łączy się z serwerem aby zalogować użytkownika. Serwer wysyła odpowiedź do programu, po czym zostaje ona wyświetlona dla użytkownika.

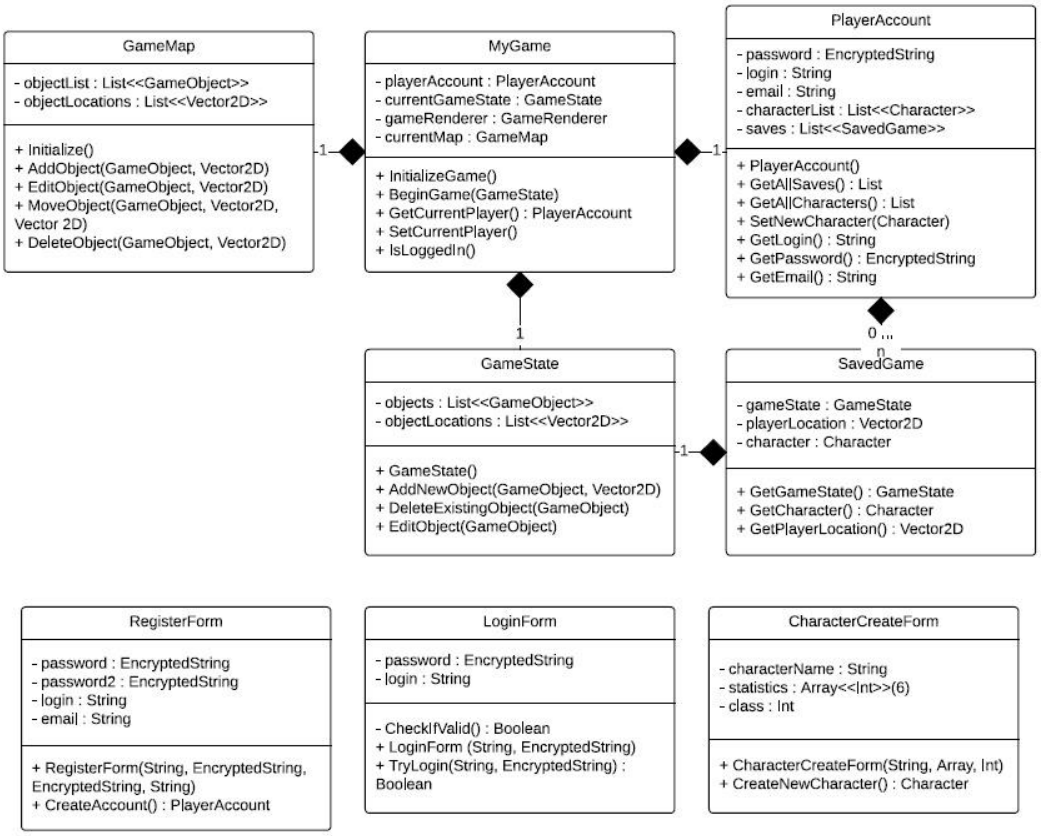


Rysunek 16 - diagram komunikacji dla dostosowania aplikacji

Gracz może zażądać zmian ustawień gry bądź jego postaci lub drużyny. W przypadku zmian opcji gry komunikacja odbywa się tylko z aplikacją gry. W przypadku zmian postaci lub drużyny gracz wprowadza zmiany w aplikacji klienckiej, a ta przesyła informację o zmianie do serwera który wprowadza je.

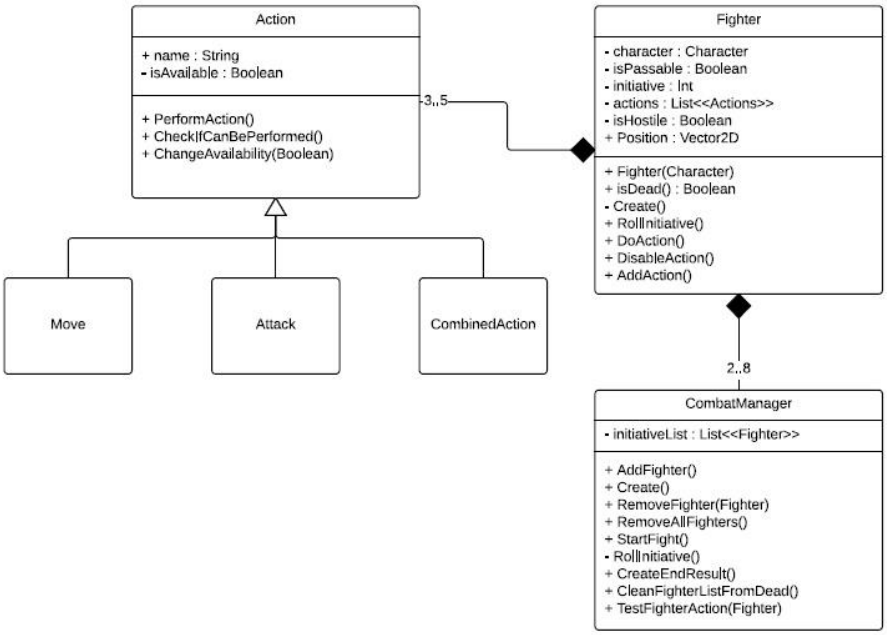
4.6. Diagramy klas

Diagramy klas pokazują podział funkcji systemu na poszczególne klasy, które odpowiadają ich odpowiednikom podczas implementacji systemu. Programista powinien śledzić te diagramy aby utworzyć żądaną strukturę programu.



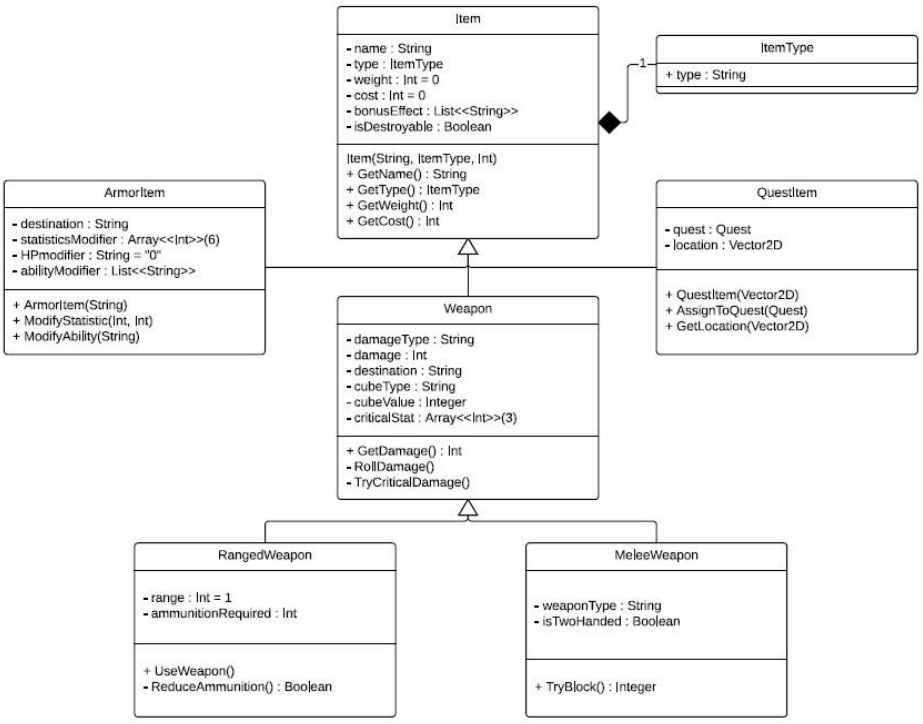
Rysunek 17 - Diagram klas dla mechanizmu gry

Klasa MyGame obsługuje cały mechanizm gry, zarządzając położeniem obiektów na mapie oznaczona klasą GameMap oraz rysowaniem ich na ekranie za pomocą renderera wbudowanego w pakiet Unity. Klasa ta może też sprawdzić czy obecny gracz ma możliwość obsługi gry i czy jest zalogowany. Sam gracz jest oznaczony klasą PlayerAccount, posiada swoje hasło, login, adres e-mail, listę postaci (mogą być zarówno lokalne bądź w trybie online) oraz listę zapisanych stanów gry. Stan gry (GameState) jest zapisany w tej liście wraz z lokalizacją gracza oraz postacią wykorzystaną na mapie. Sam stan gry to lista obiektów oraz ich lokalizacji na mapie. Można dodawać oraz usuwać obiekty z mapy oraz je edytować. Do komunikacji z serwerem klient wykorzystuje formy do kolejno: rejestracji, logowania oraz tworzenia postaci. Formy te są szyfrowane i bezpiecznie wysyłane na serwer oczekując odpowiedzi od serwera.



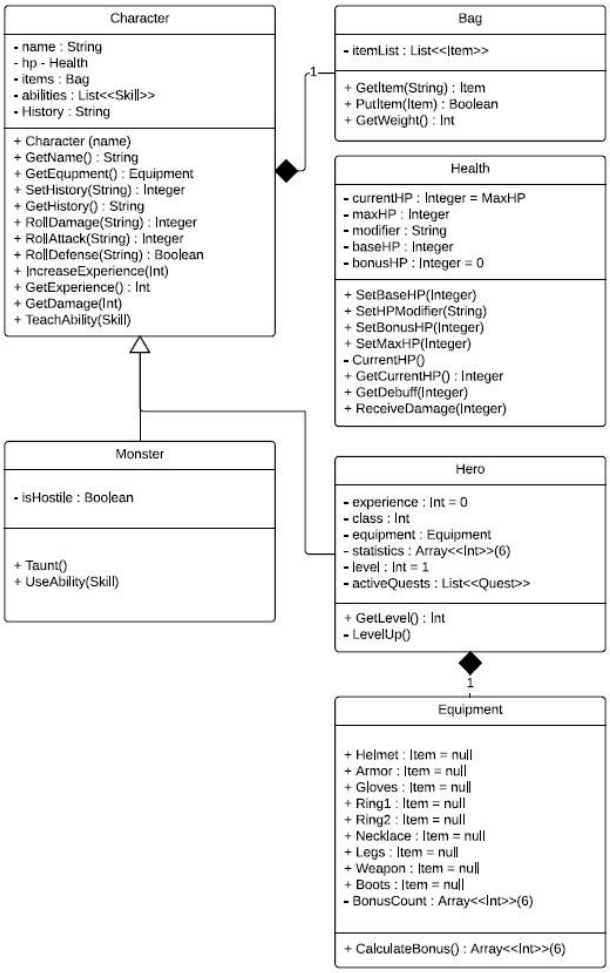
Rysunek 18 - Diagram klas dla wydarzeń podczas walki w grze

Walka startowana jest przez klasę CombatManager zawierającą listę wojowników (Fighters), zarządza listą priorytetów w walce oraz decyduje o tym, jaki rezultat jest stworzony przez dane akcje. Klasa może zarządzać wojownikami, czyścić pole walki z martwych wojowników, tworzyć rezultat walki i wyświetlać go. Wojownik zawiera daną postać, sprawdza czy można przejść przez nią, czy jest sojusznikiem bądź wrogiem. Każdy wojownik zawiera też inicjatywę – obliczaną ze statystyk oraz umiejętności postaci oraz listę akcji które może wykonać. Wojownik przy wykonywaniu tury wybiera jedną z nich, po czym sprawdzana jest poprawność akcji. Akcja jest to przykładowo ruch, atak bądź kombinacja obydwu lub akcja wspierająca swój zespół.



Rysunek 19 - Diagram klas przedmiotów używanych w grze

Klasa Item jest ogólnym pojęciem przedmiotu, każdy obiekt poza terenem, postaciami, przeszkodami i punktami kluczowymi zadań są przedmiotami. Przedmiotem może być na przykład mikstura życia regenerująca część życia użytkownikowi. Zagwarantowane jest to przez bonusowy efekt jako pole w tej klasie. Każdy przedmiot ma swoją nazwę, wagę, koszt oraz typ. Przedmioty zadaniowe (QuestItem) nie mogą być zniszczone, posiadają ich cel podany przez wektor dwuwymiarowy oraz zadanie do którego należą. Przedmiot który jest swojego rodzaju pancerzem (ArmorItem) posiada miejsce w którym powinno się znaleźć (hełm, pancerz, nogawice, buty, amulet, dwa pierścienie, rękawice). Każdy przedmiot tego typu może modyfikować statystyki gracza, jego ilość punktów życia. Rzadsze przedmioty mają też możliwość edycji umiejętności gracza. Bronie podzielone są na długodystansowe oraz bronie do walki wręcz. Każda broń ma swoje obrażenia oraz szansę na trafienie krytyczne (obliczanie przez kość). Każda broń ma szansę na nie, obrażenia wywołane przez nie może się różnić w zależności od broni (zwykle jest to 150% bazowych obrażeń). Bronie dystansowe posiadają zasięg, mogą być używane przez postaci nie znajdujące się przy przeciwniku. Używają też amunicji, jeśli gracz nie posiada takiej ten typ broni jest zablokowany. Broń do walki wręcz może być jednoręczna oraz dwuręczna. Modyfikuje to jej obrażenia. Można też zablokować cios przeciwnika jeśli obliczony rzut kostki będzie korzystny.



Rysunek 20 - Diagram klas dla typów postaci w grze

Postaci (Character) w tej grze posiadają własną nazwę, ilość punktów życia, przedmioty przy sobie, znane przez nie umiejętności oraz historię opisaną przez tekst. Postać ma też swoją bazową wartość obrażeń, którą może wylosować przez rzut kostką sześcienną. Postać może też zdobywać punkty doświadczenia, dla zwykłych postaci nie mają one znaczenia jednak dla bohaterów (Hero) mają one duże znaczenie. Worek (Bag) to przedmioty posiadane przez gracza. Worek ma limit maksymalnie 20 przedmiotów jednocześnie lub określonego udźwigu postaci określonego przez statystykę Strength. Życie (Health) to ilość punktów postaci, determinowana przez bonusy napływające z opancerzenia postaci, umiętności bądź statystyki Consitution. Po otrzymaniu obrażeń postać może dostać modyfikator życia w postaci mniejszej ilości zadawanych obrażeń, otrzymywania silniejszych ciosów bądź zmniejszonej inicjatywy. Postaci dzielą się na dwa typy:

1. Potwory, są one sterowane przez boty wbudowane w grę. Mają swoje bazowe umiejętności, określoną ilość obrażeń. Nie każdy potwór musi być wrogiem, z reguły jednak są wrogo nastawione do gracza. Za ich zabicie gracz może zdobyć przedmioty należące do ich worka.
2. Bohaterów, jest to typ postaci kierowany przez gracza. Bohater ma swój ekwipunek, klasę postaci do której należy, statystyki rozdzielane od początku gry. Każdy bohater ma swoją ilość punktów doświadczenia która wraz z wykonywaniem zadań, wygrywaniem walk bądź uczestnictwie w losowych wydarzeniach wzrasta i może osiągać nowe poziomy. Bohater posiada też listę zadań do wykonania, zostją one usunięte z listy po wykonaniu. Ekwipunek gracza składa się z broni oraz maksymalnie ośmiu unikalnych typowo przedmiotów pancerzowych.

5. Implementacja aplikacji klienckiej

Zostanie zrobione podczas następnego semestru.

6. Szczegóły techniczne dotyczące implementacji

Wykorzystane technologie w budowie aplikacji: C#, .NET, pakiet Unity 4.7.1, MySQL, DirectX 9c.

Język C# wraz z platformą .NET służył do stworzenia skryptów w aplikacji klienckiej, struktury gry oraz połączenia z serwerem. Baza danych MySQL została wykorzystana do stworzenia bazy informacji o typach obiektów w grze. DirectX w wersji 9c został wykorzystany przez samo środowisko Unity do wyświetlenia samej gry jako animacji.

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

Monitor: minimalna obsługiwana rozdzielczość 1024 x 768

HDD: minimum 1GB miejsca dostępnego na dysku

Pozostałe: dla trybu multiplayer wymagane stałe połączenie internetowe, klawiatura, myszka

7. Bibliografia

[1] Wikipedia, UML, http://pl.wikipedia.org/wiki/Unified_Modeling_Language

[2] Joseph Schmuller, „UML dla każdego”, tłumaczenie Krzysztof Masłowskiego, Gliwice 2003

[3] Beata Frączek, Notatki w Internecie 2008-2009, http://brasil.cel.agh.edu.pl/~09sbfraczek/

Disclaimer:

Poniższa praca jest w trakcie edycji, prawie każde zdanie, diagram bądź inna forma informacji może być zmieniona w trakcie tworzenia projektu.