Politechnika Wrocławska
Wydział Elektroniki

Kierunek: Informatyka (INF)
Specjalizacja: Inżynieria internetowa (INT)

Projekt Inżynierski

Gra internetowa dla wielu graczy z interfejsem webowym z elementami grafiki 3D

Multiplayer internet game with web based interface and 3D graphics

AUTOR:
Kamil Markuszewski

PROWADZĄCY PRACĘ:
dr inż. Tomasz Walkowiak, I-6

OCENA PRACY:

Wrocław 2011

Spis Treści

Zawartość

1. Wprowadzenie

1.1 Wstęp

Odkąd pojawiły się prywatne komputery, gry stały się ich nieodłącznym elementem. Spowodowało to ogromny popyt w tej dziedzinie oprogramowania. Dzięki temu powstało wiele gatunków gier komputerowych, charakteryzujących się odrębnymi cechami. Jednym z najbardziej zachęcających do gry jest gatunek **Role Playing Game**, w którym gracz wciela się w jakąś postać oraz rozwija ją. Rozgrywka zazwyczaj jest bardzo długa, a możliwość rozwoju zdecydowanie zachęca do dalszej zabawy.

Dynamiczny rozwój Internetu pozwolił na tworzenie sieciowych rywalizacji. Początkowo były to jedynie krótkie rozgrywki. Jednak połączenie zalet gatunku RPG oraz rozgrywki sieciowej zrodziło nowy gatunek, który pokochały miliony graczy na całym świecie. **Massive Multiplayer Online Role Playing Game** charakteryzuje się zarówno możliwością interakcji z innymi graczami, rozwojem własnej postaci, jak i niekończącą się rozgrywką. W jednym świecie gry znajdują się czasami ogromne ilości graczy, dzięki czemu rozgrywka staje się o wiele bardziej pasjonująca. Twórcy tego typu gier odkryli, że gracze są wstanie kupić nie tylko samą grę, ale także za opłatą ulepszać swoją postać. W przypadku niektórych produkcji stało się to nawet głównym źródłem dochodu. Tego typu produkcje charakteryzują się również często niesamowitymi światami gry z własną historią. Jako autor pracy uważam za bardzo ciekawe i przyjemne stworzenie własnego świata gry. Praca ta powstała z powodu fascynacji możliwością stworzenia własnej gry i rozwijania jej.

Bardzo często gracze wybierają proste gry stworzone w technologii Flash, osadzone w przeglądarce internetowej, ponieważ nie muszą ich kupować, pobierać ani instalować. Zastosowanie takiego rozwiązania bardzo zachęciłoby potencjalnych użytkowników. Dlatego też postanowiłem, że aplikacja zostanie osadzona w przeglądarce internetowej.

Dzięki istnieniu narzędzi pozwalających, w prosty sposób na implementację takiej gry udało się stworzyć w pełni działającą aplikację. Pozwala ona na zaspokojenie wizualnych wymagań użytkowników, ponieważ wykorzystuje grafikę 3D. Wykorzystana technologia pozwoliła na uruchamianie aplikacji w przeglądarce internetowej. Dzięki wykorzystaniu serwera, w rozgrywce jednocześnie może uczestniczyć wielu graczy, a stan rozgrywki po wylogowaniu gracza jest zapisywany. Udało się więc połączyć zalety gier RPG, rozgrywki sieciowej oraz prostych gier osadzonych w przeglądarce internetowej.

1.2 Cel pracy

Celem pracy jest zaplanowanie oraz implementacja gry, która umożliwia rozgrywkę sieciową dla wielu graczy. Projekt składa się z **serwera** oraz **aplikacji klienckiej**. Po stronie użytkownika dostępna jest możliwość uczestnictwa w rozgrywce przy pomocy przeglądarki internetowej. Do osiągnięcia celu wykorzystany został silnik graficzny **Unity 3D** rozbudowany o narzędzia do tworzenia gier, a także **Smart Fox Server**. Stworzona aplikacja kliencka wykorzystuje trójwymiarowe elementy graficzne.

1.3 Zakres pracy

Tekst pracy został podzielony na pięć rozdziałów, z których najważniejszymi są projekt bazy danych, serwer, oraz aplikacja kliencka. We wprowadzeniu, czyli rozdziale pierwszym została przybliżona tematyka projektu, oraz przedstawiono użyte technologie, omówiono także funkcjonalności jakie realizuje projekt.

Rozdział drugi omawia projekt bazy danych wykorzystanej w projekcie. Choć przedstawione tam schematy nie są skomplikowane, to jednak ich znajomość jest kluczowa dla zrozumienia funkcjonalności aplikacji.

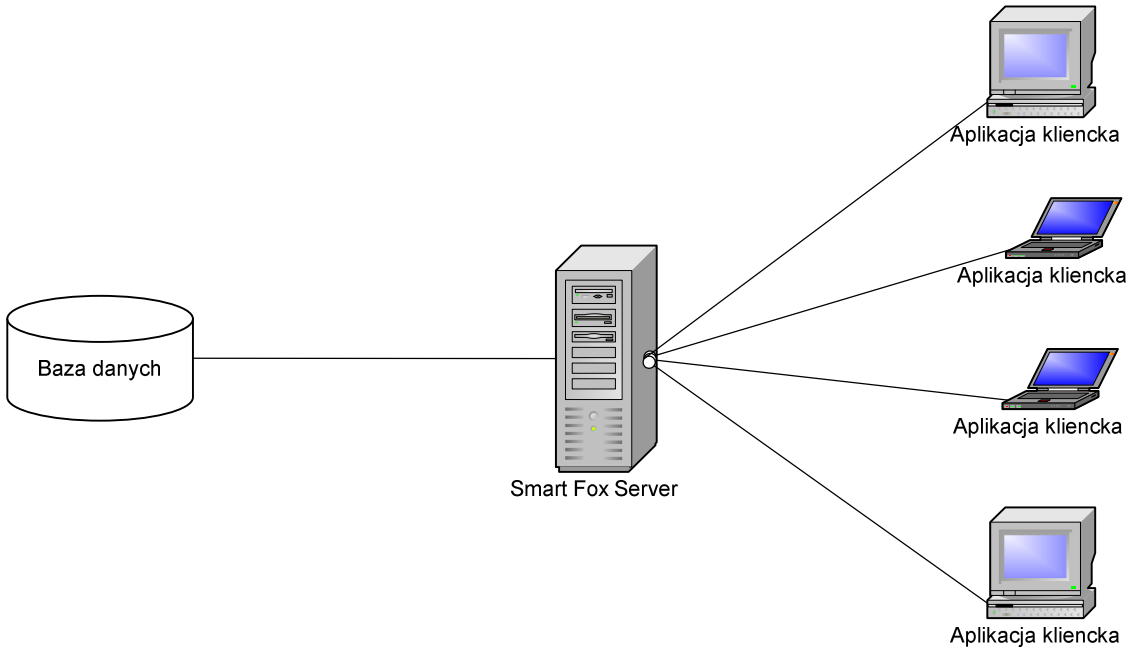
Rozdział trzeci opisuje działanie serwera, jego komunikacje z bazą danych, komunikację serwera z aplikacją kliencką, a także kwestie bezpieczeństwa ze względu na działania użytkownika. Problemy techniczne rozwiązane za pomocą użycia gotowej technologii nie zostały omówione.

Najobszerniejszy z rozdziałów, rozdział czwarty, ma za zadanie wyjaśnić działanie aplikacji klienckiej. Opisane w nim są funkcjonalności aplikacji klienckiej pod względem sposobów implementacji, oraz użytych elementów wybranej do aplikacji technologii. Zaprezentowane zostały tam elementy graficzne stworzone na potrzeby projektu, a także końcowy wygląd stworzonej gry.

W rozdziale ostatnim omówiono wady i zalety gry, dotyczy to zarówno wybranych technologii jak i sposobów implementacji. Przedstawione zostaną również perspektywy rozwoju i wdrożenia projektu.

1.4 Założenia techniczne

Projekt umożliwia wspólną grę sieciową, zatem użyty został serwer, z którym użytkownicy łączą się za pomocą aplikacji klienckich. Zapewniona została również trwałość danych. Działania użytkowników są zapisywane i wpływają na dalszą rozgrywkę nawet po opuszczeniu gry. Można było w tym celu wykorzystać pliki XML jak w wielu innych tego typu grach jednak tu została wykorzystana **baza danych**.



Rys. 1.1 Ogólny schemat połączeń sieciowych

Do podstawowych założeń technicznych należy również dodać dbałość o wydajność zarówno aplikacji klienckiej jak i serwera. Wydajność serwera wpłynie na ogólną możliwość wdrożenia gry, a wydajność aplikacji klienckiej na łatwość interakcji ze światem gry na stacji roboczej.

Ważnym aspektem jest również bezpieczeństwo. Tego typu gry narażone są na różnego rodzaju hacking oraz możliwość oszukiwania serwera. Dlatego też należało zaplanować przesyłanie komunikatów tak, aby użytkownik nie miał możliwości za pomocą zmienionej aplikacji klienckiej wysłać do serwera nieodpowiednich danych. Zdecydowanie nie udało się przewidzieć wszystkich takich sytuacji, jednak zarówno aplikacja kliencka oraz wtyczki serwera pisane były w taki sposób, by mieć na uwadze bezpieczeństwo. Z tego powodu serwer powinien kontrolować większość akcji.

Zadaniem serwera jest kontrolowanie rozgrywki wielu graczy, a także zapisywanie stanu postaci gracza w bazie danych. Do serwera zostały zaimplementowane odpowiednie wtyczki realizujące te założenia.

Silnik Unity 3D umożliwił implementację aplikacji klienckiej z trójwymiarowym interfejsem graficznym. Wykorzystanie tej technologii pozwoliło również na osadzenie gry w przeglądarce internetowej.

Aplikacja kliencka została wygenerowana w dwóch wersjach:

- aplikacja wolnostojąca
  - plik wykonywalny *.exe
- skrypt
  - wykonywany przez aplikację Unity Web Player
  - osadzony w przeglądarce internetowej

Do stworzenia gry przygotowane zostały także elementy graficznego interfejsu użytkownika:

- elementy graficzne 2d
  - ikony przycisków
  - ikony przykładowych przedmiotów
  - tekstury
- modele 3D

Zostały jednak wykorzystane również elementy dostarczane wraz ze środowiskiem Unity 3D, a także umieszczone na stronie internetowej z darmowymi modelami. [1]

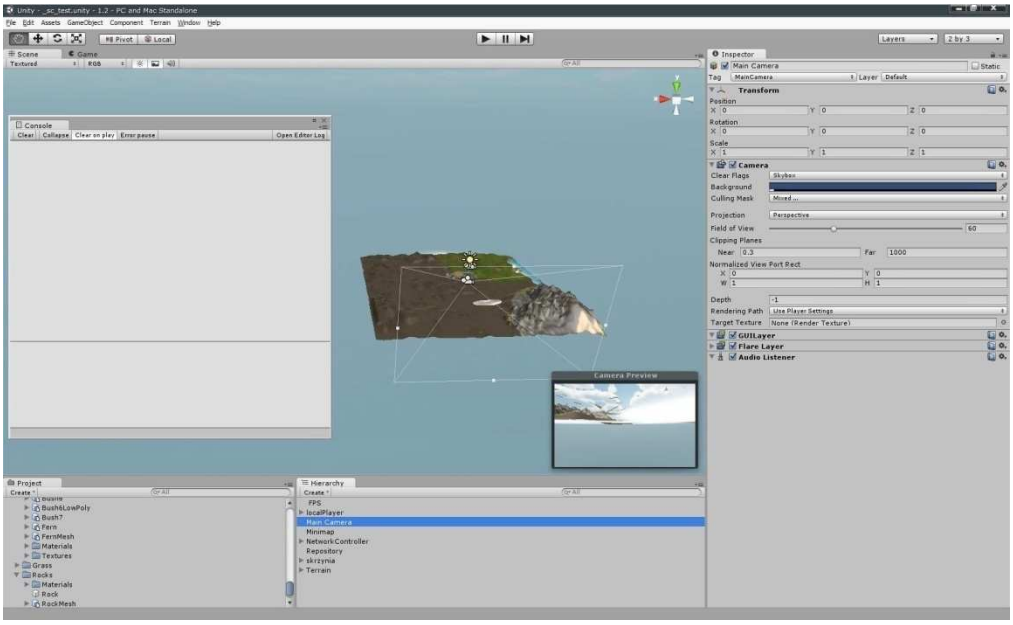
1.5 Użyte technologie

1.5.1 Unity 3D

Unity 3D [5] jest silnikiem graficznym do tworzenia gier i dostarcza narzędzia wystarczające do stworzenia w pełni funkcjonalnej gry z trójwymiarową grafiką. Bezpłatna wersja silnika nie pozwala na komercyjne wykorzystanie, a także nie zawiera wielu dodatkowych narzędzi dostępnych w wersji Unity PRO.

Wraz z silnikiem dostarczane jest aplikacja Unity Editor wspomagająca tworzenie scen 3D. W edytorze można tworzyć obiekty, przypisywać do nich skrypty czy trójwymiarowe modele, a następnie zbudować gotową już aplikację. Środowisko łatwo radzi sobie z importowaniem do projektu trójwymiarowych modeli stworzonych w takich programach jak Blender, Maya czy 3D Studio Max. Potrafi użyć również plików graficznych czy dźwiękowych. Obsługuje on takie języki jak JavaScript, Boo oraz C#. W projekcie po stronie aplikacji klienckiej przeważa zdecydowanie **C#**, niektóre elementy oprogramowane są przy użyciu **JavaScript**.

Unity Editor można rozbudowywać na podstawie dostarczonego interfejsu. Każda właściwie zaimplementowana klasa, która jest odpowiednio przyporządkowana do obiektu w edytorze jest przez niego wyświetlana tak, że bez pomocy edytora tekstowego można zmieniać jej publiczne argumenty. Możliwe jest tworzenie komponentów edytora które będą pozwalały na szybsze tworzenie gier. Edytor obsługuje takie języki jak JavaScript, Boo oraz C#. W projekcie po stronie aplikacji klienckiej przeważa zdecydowanie **C#**, niektóre elementy oprogramowane są przy użyciu **JavaScript**.



Rys. 1.2 Unity Editor dostarczany wraz z silnikiem Unity 3D

1.5.2 Smart Fox Server

Smart Fox Server [2] jest serwerem napisanym w języku **Java** przeznaczonym głównie do obsługi gier napisanych w technologii Flash, wspierających obsługę dużej liczby komunikujących się użytkowników. Pozwala on na rozszerzanie jego możliwości w oparciu o załączanie zewnętrznych bibliotek lub pisanie wtyczek. Mogą być one pisane w takich językach jak Java, ActionScript czy Python. Darmowa wersja serwera pozwala na połączenie z maksymalnie 20 użytkownikami. Aby połączyć się z większą liczbą osób należy wykupić licencje na określony limit połączeń.

Mimo iż Smart Fox Server tworzono głównie z myślą o wspieraniu aplikacji flaszowych, powstało API pozwalające za pomocą języka **C#** oraz technologii.**NET** komunikować się z serwerem. Pozwala to połączyć wybrany serwer z grą napisaną w środowisku Unity 3D. Ustanowienie połączenia między serwerem, a grą uruchamianą w środowisku Unity Web Player w przeglądarce internetowej wymaga uruchomienia dodatkowego programu - **policy-server** dostarczanego wraz z aplikacją Smart Fox Server. Policy-server stworzy dla przeglądarki internetowej osobne gniazdko i skomunikuje się z samym serwerem.

Początkowo wtyczki rozszerzające funkcjonalność serwera pisane były w projekcie przy użyciu języka **ActionScript**. Zapytania wykonywały się asynchronicznie, jako osobne skrypty i bardzo często zapisywały informacje do bazy danych. Rozwiązanie to było wyjątkowo dalekie od optymalnego, zbyt obciąża bazę danych, nie pozwala na dobrą kontrolę działań użytkowników i daje możliwość oszukiwania serwera przy użyciu własnej aplikacji klienckiej. Wiosek ten skłonił mnie do napisania wtyczki do serwera od początku w języku **Java**. Była to jedna wtyczka kontrolująca wszystkie zapytania wysyłane od użytkowników skompresowana do pliku z rozszerzeniem *jar*.

1.6 Funkcjonalności

System kont	W celu uczestniczenia w rozgrywce użytkownik musi posiadać konto w grze. Projekt nie zakłada możliwości rejestracji, zatem konta zakładane będą administracyjnie.
Logowanie	Po uruchomieniu aplikacji klienckiej użytkownik proszony jest o podanie loginu oraz hasła. Dane te wysyłane są do serwera. Autoryzacja odbywa się w oparciu o informacje zapisane w bazie danych. Dla zapewnienia bezpieczeństwa hasło jest hashowane.
Tworzenie postaci	Po zalogowaniu użytkownik ma możliwość stworzenia nowej postaci. Wymagane jest podanie nazwy oraz wybór klasy. Po wciśnięciu przycisku „stwórz” do bazy danych zostanie dodana postać użytkownika z początkowymi parametrami.
Wybór postaci	Po zalogowaniu użytkownik ma możliwość wyboru jednej prezentowanych na trójwymiarowej scenie postaci. Dla polepszenia wyglądu aplikacji, za postacią widoczny jest trójwymiarowy świat gry. Po wciśnięciu przycisku „wejdź” gracz przechodzi do świata gry.
Postać	Gracz na pojedynczym koncie może posiadać do 10 postaci. Każda z nich charakteryzować się może innymi parametrami, takimi jak posiadane przedmioty, statystyki czy położenie. Jeden gracz może kontrolować losy jednej postaci jednocześnie.
Widok trzeciej osoby	Postać widziana jest z tyłu przez użytkownika. Ma on możliwość posługując się suwakiem myszki po łuku oddalać oraz przybliżać kamerę, tak by możliwy był widok zza postaci jak i od góry.
Sterowanie postacią	Gracz ma możliwość sterowania akcjami oraz położeniem postaci na płaszczyźnie. Za pomocą kursorów gracz może przemieszczać postać. Wciskając Shift przełączamy pomiędzy chodzeniem i bieganiem. Kliknięcie myszą na płaszczyznę ustala punkt, w którym będzie podążać postać. Mysz służy również do atakowania przeciwników po kliknięciu na nich prawym przyciskiem myszy, a także umożliwia interakcje z obiektami jak skrzynia czy kapliczka.

Graficzny interfejs użytkownika	Graficzny interfejs użytkownika składa się z panelu po lewej stronie ekranu oraz panelu szybkiego dostępu u dołu ekranu. Dla wygody użytkownika, po kliknięciu prawym przyciskiem na elementy graficznego interfejsu, pokazują się podpowiedzi na temat ich przeznaczenia. Ważnym elementem są zakładki uruchamiane następującymi przyciskami: ekwipunek, plecak, postać, opcje.
Mapa	Mapa znajdująca się w lewym górnym rogu monitora wyświetla widziane z góry, aktualne położenie postaci w świecie gry wraz z pobliskimi obiektami. Znajdujące się obok przyciski pozwalają przybliżać oraz oddalać punkt, z którego widoczny jest prezentowany świat.
Prezentacja punktów zdrowia i many	Pod mapą znajdują się dwa paski prezentujące punkty życia oraz many. Wskaźniki te zmieniają się, gdy postać otrzymuje obrażenia, lub używa czaru.
Prezentacja przedmiotów, magii oraz ekwipunku postaci	Postać może posiadać przedmioty. Przedmioty mogą znajdować się w trzech obszarach: ekwipunku, plecaku oraz panelu szybkiego dostępu. Możliwość umieszczenia przedmiotu w danym obszarze zależy od jego typu. W ekwipunku znajdują się przedmioty, które można założyć, wpływają one na statystyki gracza. W plecaku znajdują się wszystkie przedmioty, które nie są czarami. W panelu szybkiego dostępu znajdują się czary oraz eliksiry. Wykorzystać je można za pomocą skrótów klawiszowych. W grze przedmioty reprezentowane są poprzez odpowiednie dwuwymiarowe grafiki. Dokładne informacje na temat przedmiotu można uzyskać poprzez kliknięcie na nim prawym przyciskiem myszy.
Prezentacja statystyk postaci	W zakładce postać można zobaczyć nazwę postaci, jej poziom, doświadczenie oraz statystyki. Po zdobyciu poziomu gracz może rozdawać statystyki, co wpływa na siłę postaci.
Opcje	W zakładce można przełączać aplikację między trybem okienkowym, a pełnoekranowym. Wyświetlana jest również ilość klatek na sekundę.

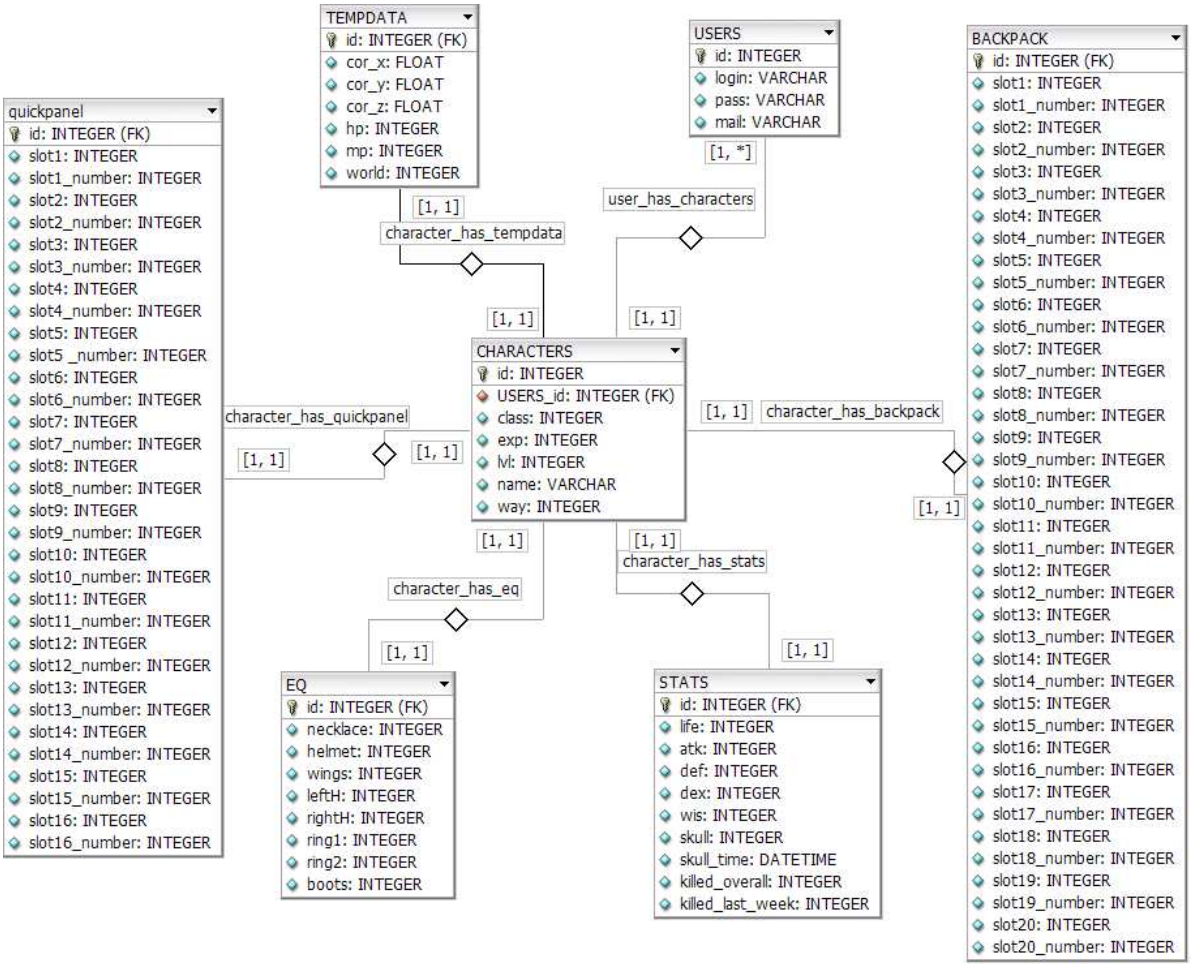
Doświadczenie	Za pokonanie przeciwnika postać otrzymuje doświadczenie. Po zdobyciu jego określonej liczby postać otrzymuje poziom, co pozwala na rozdanie nowych statystyk.
Walka	Gracze mogą walczyć między sobą. Polega to na uderzaniu postaci przeciwnika, przez co traci ona punkty zdrowia. Po utracie wszystkich postać umiera. W walce można używać czarów, które służą do leczenia się oraz zadawania przeciwnikowi większych obrażeń. Wpływ na zadawane obrażenia ma odległość, statystyki obu postaci oraz ich przedmioty znajdujące się w ekwipunku. Dodatkowo postać może otrzymać obrażenia podczas upadku z dużej wysokości.
Automatyczny atak oraz podążanie	Atakowany przeciwnik może nam uciekać, w celu ułatwienia rozgrywki dostępne są domyślnie włączone przyciski automatycznego ataku oraz podążania. Podążanie polega na śledzeniu atakowanej postaci, a automatyczny atak na atakowaniu jej, co sekundę przez określony czas.
Trwałość parametrów postaci	Poczynania gracza mają wpływ na dalszą rozgrywkę i zostaną zapisane po jego wylogowaniu i utracie połączenia Zapamiętane będą między innymi poziom, statystyki, przedmioty, położenie.

2. Projekt bazy danych

Do uzyskania trwałości danych dotyczących rozgrywki została wykorzystana baza danych. Jej schemat jest wyjątkowo prosty i da się go sprowadzić do dwóch tabel USERS (użytkownicy) oraz CHARACTERS (postacie) z relacją jeden do wielu. Zostały jednak one rozdzielone z powodu innego przeznaczenia funkcjonalnego, na tabele z relacją jeden do jednego.

Tabela:	**Opis:**
Users	Przechowuje użytkowników. Każdy z nich może mieć wiele postaci. Login – login użytkownika wymagany do autoryzacji Pass – hasło użytkownika wymagany do autoryzacji Mail – adres e-mail użytkownika, stworzone na potrzeby dalszego rozwoju aplikacji
Characters	Przechowuje postacie. Jedna postać należy do jednego użytkownika. Class – klasa postaci, inaczej profesja Exp – punkty doświadczenia Lvl – poziom postaci Name – nazwa postaci Way – ścieżka postaci (dobra, zła, neutralna)
Tempdata	Dane tymczasowe na temat postaci. Cor_x, Cor_y, Cor_z – współrzędne położenia postaci w trójwymiarowej przestrzeni. Hp, Mp – punkty życia oraz many postaci World – świat, w którym znajduje się postać, stworzone na potrzeby dalszego rozwoju aplikacji
Backpack	Plecak przechowujący przedmioty. Użyty tu, na pozór zły sposób umieszczenia danych w tabeli pozwala zabezpieczyć się przed niespójnością danych. Slot – przechowuje identyfikator przedmiotu Slot_number – przechowuje liczbę przedmiotów w slocie, stworzone na potrzeby dalszego rozwoju aplikacji

Quickpanel	Panel szybkiego dostępu przechowujący przedmioty oraz czary.
Eq	Ekwipunek przechowujący identyfikatory przedmiotów będące ubiorem postaci.
Stats	Statystyki postaci dotyczące zabójstw oraz wpływające na jego siłę. Life – wpływa na liczbę punktów życia Atk – wpływa na siłę ataku Def – wpływa na umiejętność obrony Dex - stworzone na potrzeby dalszego rozwoju aplikacji, wpływa na szybkość Wis – wpływa na liczbę punktów many Skull, Skull_time, Killed_overall, Killed_last_week - stworzone na potrzeby dalszego rozwoju aplikacji



Rys. 2.1 Schemat projektu bazy danych

3. Serwer

3.1. Struktura plików serwera

Aby uruchomić serwer, tak by współdziałał z aplikacją kliencką, należy załączyć do niego odpowiednie biblioteki oraz rozszerzenia, a także poprawnie go skonfigurować. Opis struktury plików serwera znajduje się w **załączniku**. Najważniejszymi kwestiami są rozszerzenie możliwości serwera o korzystanie z wybranej bazy danych, a także dodanie rozszerzeń stworzonych na potrzeby projektu.

3.2. Konfiguracja serwera

Smart Fox Server można konfigurować za pomocą pliku config.xml [3] zapisanego w języku XML. Odpowiednia konfiguracja jest niezbędna do prawidłowego funkcjonowania aplikacji, ponieważ zakłada ona miedzy innymi dodanie odpowiednich rozszerzeń oraz zdefiniowanie stref. Jest to bezpośrednio związane z samą budową projektu. Ponadto należy określić elementy niezbędne do komunikacji z bazą danych. Opis pliku konfiguracyjnego znajduje się w **załączniku**.

3.3. Komunikacja klient – serwer

W celu umożliwienia komunikacji ze strony aplikacji klienckiej została wykorzystana dodatkowa biblioteka **SmartFoxClient.dll** zawierająca interfejsy wystarczające do zapewnienia wymiany danych. Po stronie klienta [7] utworzenie połączenia sprowadza się do stworzenia obiektu typu *SmartFoxClient*, oraz wywołania funkcji *Connect*. W przypadku aplikacji osadzonej w przeglądarce, wysyłane jest zapytanie dotyczące pozwolenia na połączenie z niestandardowym portem Policy-servera, przy pomocy funkcji *PrefetchSocketPolicy*.

Smart Fox Server umożliwia definiowanie stref oraz pokoi, w których mogą znajdować się użytkownicy. Do każdej strefy oraz pokoju może być przypisane rozszerzenie pozwalające na obsługę zapytań. Po ustanowieniu połączenia użytkownik dodawany jest do odpowiedniej strefy oraz pokoju. Pozwala to na komunikację z rozszerzeniami oraz sprawne zarządzanie użytkownikiem. Użytkownik znajdujący się w strefie może wysłać wiadomość do rozszerzenia tej strefy przy pomocy funkcji *SendXtMessage*, w której należy określić rozszerzenie serwera, komendę wiadomości, obiekt zawierający przesyłane dane, a także format przesyłania komunikatu. Dostępne formaty komunikatu to ciąg znaków, XML, oraz JSON – wybrany w projekcie.

Wysłany od klienta komunikat powinien być na serwerze odebrany i obsłużony. Pozwala na to metoda *handleRequest* dostępna w klasie *AbstractExtension*, po której należy dziedziczyć, aby zaimplementować rozszerzenie. Argumentami funkcji są komenda komunikatu, obiekt reprezentujący przesyłane dane, obiekt typu *User* reprezentujący użytkownika wysyłającego oraz identyfikator pokoju. Obiekt typu *User* jest ważnym elementem pozwalającym na późniejszą identyfikację użytkownika dzięki dostępie do gniazdka łączącego serwer z aplikacją kliencką oraz unikalnemu identyfikatorowi użytkownika. Identyfikator przypisywany jest podczas tworzenia połączenia między serwerem i aplikacją kliencką.

Serwer odpowiada użytkownikowi za pomocą funkcji *sendResponse*, która jako argumenty przyjmuje wysyłane dane, identyfikator pokoju, oraz listę odbiorców.

Serwer otrzymuje od użytkowników informacje o ich wszystkich ważnych akcjach wpływających na dalszą rozgrywkę. Stworzony sposób kontroli pozwala na rozbudowanie funkcji serwera obsługujących odbieranie tych komunikatów o dodatkową kontrolę poprawności względem stanu rozgrywki.

Wysyłany komunikat składa się z dwóch głównych części: komendy komunikatu oraz wysyłanych danych. Komenda decyduje o sposobie obsłużenia go.

Komenda:	**Opis:**	**Wysyłane:**
getAuth	Autoryzacja przy pomocy loginu i hasła.	Po wciśnięciu przycisku zaloguj.
getChars	Wysłanie listy postaci do gracza wraz z podstawowymi informacjami o nich takich jak poziom, doświadczenie, ścieżka, nazwa.	Od razu po autoryzacji.
loadCharacter	Wysłanie pełnej informacji o danej postaci do gracza.	Po wybraniu postaci.
createCharacter	Stworzenie w bazie danych nowej postaci z początkowymi parametrami dla użytkownika.	Po wciśnięciu przycisku stwórz.

saveCharacter	Wysyłanie głównych informacji o postaci na serwer.	Po wylogowaniu gracza z postaci.
savePeriodCharacter	Wysyłanie a serwer tymczasowych informacji o postaci takich jak liczba punktów życia oraz many, a także współrzędnych położenia.	Co 10 sekund oraz przy otrzymywaniu dużych obrażeń.
saveCharacterStats	Wysyłanie statystyk postaci po dodaniu jakiegoś punktu.	Podczas rozdawania statystyk.
saveCharacterWay	Ustawienie ścieżki postaci. Powinno być możliwe tylko raz dla jednej postaci.	Stworzone na potrzeby dalszego rozwoju aplikacji
saveCharacterItems	Wysyłanie przedmiotów postaci z plecaka, panelu szybkiego dostępu oraz ekwipunku.	Po zmianie położenia przedmiotu.
atkCharacter	Wysyłanie informacji o zadaniu obrażeń. Zadawane obrażenia obliczane są przez serwer.	Podczas ataku.
characterDied	Wysyła informację o śmierci atakowanego przeciwnika.	Po walce.
saveCharacterItemsStats	Dodatkowe właściwości założonych przedmiotów wysyłane do serwera w celu dokładnego obliczenia obrażeń.	Przed atakiem.
disconnect	Informacja dla serwera o zapisaniu postaci do bazy danych oraz usunięcia jej z tymczasowej listy na serwerze.	Po wylogowaniu z postaci, lub wewnętrznym zdarzeniu serwera dotyczącym utraty połączenia.

3.4. Komunikacja z bazą danych

Jednym z głównych zadań serwera jest zapewnienie trwałości danych, czyli zapisywanie stanu gry w bazie danych. Odpowiednia konfiguracja zapewnia dostęp do bibliotek pozwalających na połączenie serwera z bazą MySQL. Z powodów bezpieczeństwa cała komunikacja z bazą danych wykonywana jest na serwerze. Rozszerzenie korzysta z bazy danych używając interfejsów będących elementami serwera, pozwala to na większą kontrolę oraz wygodną konfigurację połączenia. Ilość zapytań została możliwie zminimalizowana, tak by nie obciążać bazy danych.

Początkowo do tego celu użyty miał być framework hibernate. Udało się umożliwić jego wdrożenie. Okazał się jednak zbyt skomplikowany przez narzut dodatkowych zapytań oraz dość pracochłonną konfigurację. Dlatego zostało to zrealizowane przy pomocy prostych zapytań SQL.

W funkcji inicjalizującej *init* rozszerzenia obiektu reprezentującego aktualną strefę pobierany jest menadżer bazy danych. Z jego pomocą, używając prostych funkcji prepareStatement, executeQuery oraz executeUpdate wykonywane są operacje na bazie danych takie jak: SELECT, INSERT oraz UPDATE.

Zapytania obejmują:

- autoryzację
- pobranie listy wszystkich postaci gracza
- tworzenie postaci
- pobranie pełnej postaci
- zapisanie postaci po opuszczeniu gry

3.5. Autoryzacja użytkownika

Gatunek gier MMORPG zakłada, iż gracz posiada własna postać, do której tylko on ma dostęp. Potrzebny jest sposób autoryzacji użytkownika zapewniający bezpieczeństwo. W bazie danych stworzona została tabela USERS zawierająca loginy oraz zahashowane hasła użytkowników. W celu uwierzytelnienia, aplikacja kliencka wysyła komunikat z loginem oraz zahashowanym hasłem. Jeśli autoryzacja powiedzie się, rozszerzenie serwera tymczasowo przechowywać będzie niezbędne do identyfikacji dane o użytkowniku. Przechowywane są zarówno elementy wczytane z bazy danych jak login, identyfikator użytkownika, a także obiekt typu *SocketChannel* reprezentujący gniazdko połączenia. Z pomocą tych danych będzie możliwa weryfikacja użytkownika bez ponownego podawania hasła. Usunięcie tych informacji z pamięci serwera nastąpi, gdy gracz wyloguje się z gry lub połączenie zostanie utracone.

3.6. Kontrolowanie aplikacji klienckiej

Po autoryzacji, do gracza wysyłana jest lista jego postaci. Gracz wybiera jedną z nich i wchodzi do gry. Na serwerze zapisywany jest pełny stan wybranej postaci, która została pobrana z bazy danych. Ten sam stan wysyłany jest do użytkownika. Serwer może kontrolować zmiany, jakich próbuje dokonać użytkownik. Aby zabezpieczyć się przed zmodyfikowanymi aplikacjami klienckimi ułatwiającymi grę, można rozszerzyć możliwości serwera o weryfikację wysyłanych danych. W przypadku wykrycia oszustwa gracza można automatycznie korygować dane, rozłączać gracza, lub dodawać go na listę zbanowanych.

3.7. Wylogowanie

W projekcie wylogowanie rozumiane jest jako opuszczenie postaci, ale nie wylogowanie z konta. Po wyjściu z postaci użytkownik nie musi ponownie przechodzić autoryzacji i może wejść do gry na innej postaci. W tym czasie do serwera wysyłane są komunikaty dotyczące aktualizacji pełnego stanu postaci oraz chęci zapisania jej do bazy danych, a usunięcia z tymczasowej listy na serwerze. Pełne wylogowanie użytkownika oraz usunięcie go z tymczasowej listy na serwerze następuje po utracie połączenia, czyli wyłączeniu aplikacji lub zamknięciu zakładki w przeglądarce.

4. Aplikacja kliencka

4.1. Możliwości silnika Unity 3D

Aplikacja kliencka została stworzona w programie Unity Editor [6]. Pozwala on na łatwe budowanie trójwymiarowych scen oraz rozszerzanie ich możliwości za pomocą języków programowania C#, JavaScript, Boo. Gotowy projekt można skompilować, co pozwoli na stworzenie pełnej działającej aplikacji w wybranej wersji. Może być to aplikacja działająca w systemie Windows, Mac OS X, uruchamiana aplikacją Web Player, a także działająca na platformach Wii, PS3, Xbox 360, Android, iOS.

Projekt w Unity Editor dzieli się przede wszystkim na sceny zawierające obiekty. Obiekt nie zawsze jest figurą w przestrzeni, ale każdy posiada współrzędne położenia, rotację oraz skalę. Te podstawowe informacje zapisane są w klasie *Transform*. Do obiektu można przypisywać dodatkowe obiekty, klasy zaimplementowane w językach programowania oraz inne komponenty. Środowisko dostarcza ogromną ilość przydatnych komponentów jak na przykład klasy renderujące siatki trójwymiarowych modeli, pozwalające na odtwarzanie dźwięku, obsługujące kolizje. Dostępne są również podstawowe trójwymiarowe modele, tekstury, a nawet proste efekty specjalne jak płonący ogień.

4.2. Sceny w aplikacji klienckiej

Aplikacje powstające w Unity Editor muszą posiadać przynajmniej jedną scenę zawierającą obiekty. Zmiana aktualnie aktywnej sceny jest możliwa za pomocą funkcji *LoadLevel* znajdującej się w klasie *Application*, która jako argument przyjmuje nazwę sceny.

Scena	Opis
_sc_login	Scena z formularzem logowania.
_sc_auth	Scena obsługująca komunikaty po logowaniu w celu autoryzacji.
_sc_choseChar	Scena powalająca na wybór lub tworzenie postaci.
_sc_world1	Scena reprezentująca świat gry. Można stworzyć więcej scen świata gry i poruszać się między nimi.

4.3. Scena logowania

W scenie logowania nie są wyświetlane elementy trójwymiarowego świata. Widoczny jest jednak graficzny interfejs użytkownika składający się z:

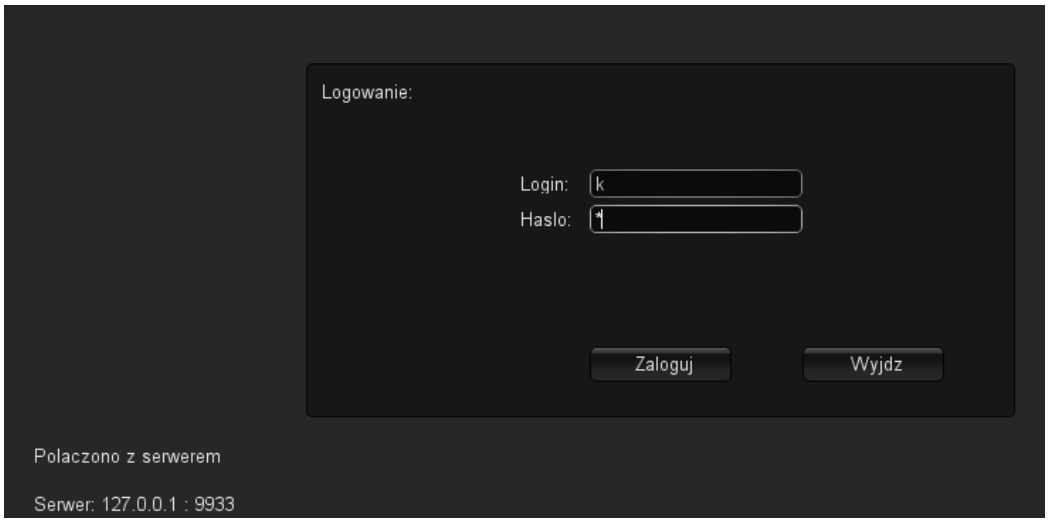
- pól tekstowych – *GUI.TextField*
- tekstów zachęty – *GUI.Label*
- przycisków – *GUI.Button*
- obszarów – *GUI.Box*

W celu utworzenia tych elementów do obiektu sceny została przypisana klasa dziedzicząca po *MonoBehaviour*. Po klasie *MonoBehaviour* musi dziedziczyć każda klasa języka C# przypisana do obiektu. Pozwala to na użycie przydatnych metod, z których najważniejsze to:

Nazwa metody	Opis
Awake	Wywoływana podczas ładowania instancji skryptu.
Start	Wywoływana przed pierwszym wywołaniem funkcji Update.
Update	Wywoływana z każdą ramką animacji, jeśli scena jest już załadowana.
onGUI	Wywoływana dla zdarzenia odświeżenia graficznego interfejsu użytkownika.
OnMouseOver	Wywoływana z każdą ramką animacji, jeśli kursor myszy jest wewnątrz obiektu.

Wszystkie metody tworzące graficzny interfejs użytkownika muszą być wywoływane w funkcji *onGUI*. W większości funkcje graficznego interfejsu użytkownika, jako argumenty, przyjmują współrzędne w dwuwymiarowej przestrzeni reprezentującej ekran oraz ciąg znaków, lub obiekt typu *GUIStyle* opisujący wygląd danego elementu.

Obiekt typu *GUIStyle* jest ważny w przypadku teksturowania oraz zmiany koloru elementu, co również wymaga stworzenia tekstury. Tekstura ta może być jednak wygenerowana za pomocą wypełniania kolejnych pikseli kolorami określanymi w trybie RGB.



Rys. 4.1 Scena logowania do aplikacji

```
void Awake()
  {

      Application.runInBackground = true;
      if (SmartFox.IsInitialized())
      {
          Debug.Log("SmartFox zostal zainicjalizowany, ale odrzuca polaczenie");
          smartFox = SmartFox.Connection;
      }
      else
      {
          if (Application.platform == RuntimePlatform.WindowsWebPlayer)
          {
              Security.PrefetchSocketPolicy(serverIP, serverPort, 3);
          }
          try
          {
              Debug.Log("Tworze nowego SmartFoxClient");
              smartFox = new SmartFoxClient(debug);
              smartFox.runInQueueMode = true;
          }
          catch (Exception e)
          {
              Debug.Log(e.ToString());
          }
      }

      SFSEvent.onConnection += OnConnection;
      SFSEvent.onConnectionLost += OnConnectionLost;
      SFSEvent.onLogin += OnLogin;
      SFSEvent.onRoomListUpdate += OnRoomList;
      SFSEvent.onDebugMessage += OnDebugMessage;
      Debug.Log("Proba polaczenia z SmartFoxServer");
      smartFox.Connect(serverIP, serverPort);

    }
```

Rys. 4.2 Ustanowienie połączenia z serwerem

Scena logowania odpowiada także za rozpoczęcie połączenia z serwerem. Sposób ustanowienia połączenia widoczny jest na **rysunku 4.2**. Połączeniem zarządza obiekt typu *SmartFoxClient* dostarczany wraz z biblioteką **SmartFoxClient.dll.** Do stworzenia połączenia wystarcza użycie funkcji *Connect*.

Dodatkowo rejestrowane są zdarzenia pomocne przy korzystaniu z połączenia. Zdarzenia muszą być zarejestrowane w funkcji Awake oraz wyrejestrowane podczas zamykania aplikacji. Używane w projekcie zdarzenia serwera to:

- *SFSEvent.onJoinRoom* – Wywołuje metodę, gdy lokalny użytkownik wejdzie do pokoju na serwerze.
- *SFSEvent.onUserEnterRoom* – Wywołuje metodę, gdy inny użytkownik wejdzie do pokoju na serwerze.
- *SFSEvent.onUserLeaveRoom* - Wywołuje metodę, gdy inny użytkownik opuści pokój na serwerze.
- *SFSEvent.onObjectReceived* – Wywołuje metodę, gdy zostanie odebrana wiadomość od użytkownika za pośrednictwem serwera.
- *SFSEvent.onPublicMessage* - Wywołuje metodę, gdy zostanie odebrana publiczna wiadomość od użytkownika za pośrednictwem serwera wysłana do wszystkich użytkowników.
- *SFSEvent.onExtensionResponse* - Wywołuje metodę, gdy zostanie odebrana wiadomość od rozszerzenia serwera.
- *SFSEvent.onConnection* - Wywołuje metodę, gdy nawiązane zostanie połączenie z serwerem.
- *SFSEvent.onConnectionLost* – Wywołuje metodę, gdy połączenie z serwerem zostanie zerwane.
- *SFSEvent.onLogin* - Wywołuje metodę, gdy użytkownik zostanie poprawnie zalogowany na serwerze. Nie jest to równoważne z autoryzacją zaimplementowaną w projekcie.
- *SFSEvent.onRoomListUpdate* - Wywołuje metodę, gdy serwer odpowie na pytanie o listę pokoi.
- *SFSEvent.onDebugMessage* - Wywołuje metodę, gdy serwer wyśle komunikat związany z debugowaniem.

4.4. Scena tworzenia oraz wyboru postaci

Scena tworzenia oraz wyboru postaci jest sceną bardziej rozbudowaną od poprzedniej. Zawiera ona elementy graficznego interfejsu użytkownika, ale wyświetla również trójwymiarowy animowany świat.

Aby wyświetlić trójwymiarowy model wykorzystywany jest komponent *Camera*. Pozwala on określić pole widzenia, umiejscowienie obrazu na ekranie, maksymalnie widzianą odległość czy typ projekcji perspektywiczny lub ortogonalny. W aplikacji wykorzystana jest również druga kamera wyświetlająca mapę świata, jako dodatkowa kamera umieszczona nad graczem. Jej obraz widoczny jest w lewym górnym rogu ekranu.

Rys. 4.3 Scena tworzenia i wyboru postaci

Tło sceny zmienia się w zależności od parametrów postaci. Wymagało to utworzenia tak zwanych prefabów, czyli wcześniej przygotowanych złożonych obiektów, interpretowanych przez środowisko jako jeden obiekt. Elementy takie mogą być w prosty sposób umieszczane na scenie używając funkcji *Instantiate* zawartej w klasie *MonoBehaviour*. Równie proste jest usunięcie takiego obiektu ze sceny, wystarczy wywołanie funkcji *Destroy* z obiektem jako argument. W podobny sposób umieszczana jest postać widoczna na **rysunku 4.3**.

4.5. Właściwa scena gry

Rys. 4.4 Scena gry

Na **rysunku 4.4** widoczny jest wygląd sceny gry. Największy obszar zajmowany jest przez obraz wyświetlany z głównej kamery umieszczonej za postacią. Ważnym elementem jest również panel po lewej stronie będący głównym elementem graficznego interfejsu użytkownika, oraz panel szybkiego dostępu znajdujący się u dołu.

Na rysunku widoczne są także elementy graficzne zaprojektowane specjalnie do aplikacji. Są to białe oraz czerwone przyciski, a także grafika reprezentująca przedmioty oraz „czary". Grafika reprezentująca przedmioty musiała mieć oczywiście przeźroczyste tło.

Widoczny świat składa się z głównie obiektu typu *Terrain*, przypisanych do niego obiektów typu *Tree*, oraz trójwymiarowych obiektów. Dzięki narzędziu *Terrain Creator* w środowisku Unity Editor można w łatwy sposób modyfikować teren. Polega to na nadawaniu tekstur, modyfikowaniu siatki poprzez wybrzuszenia, a nawet przypisywaniu w odpowiednich miejscach obiektów typu *Tree*, reprezentujących kamienie, drzewa i krzewy. Widoczne tu modele drzew, krzewów i kamieni są elementem środowiska Unity 3D, nie zostały przygotowane specjalnie na potrzeby projektu.

4.5.1 Kontrolowanie postaci

. W samym centrum ekranu widoczna jest postać, która może być sterowana przez gracza. Do obsługi sterowania postacią wykorzystany został gotowy moduł *Character Controller* oraz skrypty obsługujące kamerę (*ThirdPersonCamera*) i poruszanie postacią (*ThirdPersonController*).

ThirdPersonCamera ma za zadanie obracać kamerę wraz z obrotem postaci. Dodatkowo pozwala na kontrolowanie kamery za pomocą suwaka myszy. *ThirdPersonController* odpowiada za kontrolę prędkości, skoki oraz animację modelu reprezentującego postać.

Model posiadający postać posiada animację. Polega ona na przypisaniu wartości rotacji, skali oraz przesunięcia częściom obiektu w zależności od klatki animacji., reprezentującej czas. Chcąc użyć takiej animacji w środowisku Unity należy zdefiniować które klatki odpowiadają za jaką część animacji. Dzięki czemu można ustawiać dowolnie wybraną przez siebie animację używając funkcji Play z klasy *Animation*.

Gracz kontroluje postać przy pomocy kursorów, lub myszy. Kontrolowanie jej ruchów za pomocą myszy polega na kliknięciu na płaszczyznę oraz znalezieniu punktu który został kliknięty. Następnie postać podąża w jego stronę. Oczywiście po natrafieniu na przeszkodę nie potrafi jej ominąć. Podążanie w kierunku punktu jest zatrzymywane, gdy postać osiągnie cel, natrafi na przeszkodę której nie potrafi ominąć, lub gdy gracz zmieni punkt docelowy. Funkcjonalność ta w przyszłości mogłaby być rozbudowana o mechanizm znajdywania drogi w oparciu o graf. Wymagałoby to zdefiniowania na płaszczyźnie punktów w których stać może postać oraz połączeń między nimi.

W celu wykrywania kliknięcia na płaszczyznę wykorzystana została klasa *RaycastHit* pozwalająca na śledzenie przecięcia promienia. Za pomocą funkcji kamery *ScreenPointToRay* współrzędne na ekranie zostają zamienione na wektor w przestrzeni. Dzięki temu funkcje klasy *RaycastHit* pozwoliły na wykrycie przecięcia tego wektora z interesującą nas obiektem typu *Terrain*, reprezentującym płaszczyznę. Algorytm znajdywania punktu przecięcia nie jest oparty o proste wyliczenie przecięcia prostej i płaszczyzny. Stworzona płaszczyzna może podlegać modyfikacjom, powodującym wybrzuszenie. Szukanie punktu przecięcia po takich modyfikacjach wciąż działa prawidłowo, ponieważ interpretuje *Terrain* jako złożony obiekt.

4.5.2 Postacie innych graczy

Każda zmiana położenia postaci powinna być widoczna dla innych graczy. W tym celu stworzona została komunikacja między aplikacjami klienckimi za pośrednictwem serwera. Aplikacja kliencka może odbierać komunikaty wysyłane przez innego użytkownika za pomocą funkcji obsługującej zdarzenie *SFSEvent.onObjectReceived*. Wywoływana funkcja otrzymuje jako argumenty obiekt typu *SFSObject*, reprezentujący przesłane dane oraz obiekt typu *User*, przechowujący dane identyfikujące nadawcę. W ten sposób można odebrać informacje o położeniu, stanu aktualnej animacji modelu, lub stanu punktów życia postaci innego gracza. Za pomocą tego mechanizmu wysyłane są również prośby o wysłanie aktualnych danych.

W celu wysłania komunikatów do innych graczy wykorzystywane są funkcje obsługi zdarzeń. Gdy lokalny gracz dołączy do pokoju, czyli gdy pojawi się w grze wywoływane jest zdarzenie *SFSEvent.onJoinRoom*. Tworzone są graficzne reprezentacje postaci innych graczy, do których utworzenia wykorzystywane są stworzone wcześniej prefaby zawierające animowaną postać wraz z potrzebnymi klasami. W tym samym czasie u innych graczy wywoływane jest zdarzenie *SFSEvent.onUserEnterRoom* informujące, iż nowy użytkownik wszedł do pokoju. Gracze wysyłają mu prośbę o podanie nicku oraz współrzędnych postaci, a także tworzą reprezentację jego postaci w świecie gry. Po wywołaniu zdarzenia *SFSEvent.OnUserLeaveRoom* informującym o opuszczeniu pokoju przez jakiegoś użytkownika usuwana jest wizualizacja jego postaci oraz wszystkie dane na jego temat.

Na **rysunku 4.5** można zobaczyć przykład wykorzystania takiego mechanizmu. Za pomocą obsługi zdarzenia *SFSEvent.onUserEnterRoom* odbierana jest informacja, że do rozgrywki dołączył nowy gracz. Jako argument funkcji obsługującej zdarzenie przekazywany jest obiekt reprezentujący tego gracza. Obiekt ten zawiera między innymi jego identyfikator na serwerze oraz gniazdko połączenia. W funkcji *SpawnRemotePlayer* tworzona jest instancja stworzonego wcześniej prefabu. Nadawana jest też unikalna nazwa, zawierająca jego identyfikator na serwerze. Użycie funkcji *SendMessage("StartReceiving")* powoduje wywołanie funkcji w klasie utworzonego właśnie obiektu reprezentującego gracza. Funkcja ta ma zadanie rozpoczęcie komunikacji z aplikacją kliencką gracza w celu synchronizacji animacji oraz położenia gracza. Następnie do aplikacji klienckiej gracza wysyłany jest komunikat z prośbą o podanie aktualnych współrzędnych, w których znajduje się jego postać. W przypadku gdy wszystkie współrzędne wynoszą 0 postać ustawiana jest w jednym z kilku domyślnych punków startowych. W rozgrywce nie ma możliwości by współrzędne postaci wynosiły 0. W ten sposób funkcjonuje cała komunikacja między graczami, dotycząca położenia, punktów zdrowia, oraz aktualnej animacji.

```
    private void UserEnterRoom(User user)
    {
      SpawnRemotePlayer(user);
      remoteUser = user;
    }


    private void SpawnRemotePlayer(User user)
    {
      UnityEngine.Object remotePlayer = Instantiate(remotePlayerPrefab, new
      Vector3(-10000, -10000, -10000), new Quaternion(0, 0, 0, 1));

      remotePlayer.name = "remote_" + user.GetId();

      (remotePlayer as Component).SendMessage("StartReceiving");
      ForceRemotePlayerToSendTransform(user);
    }

    void ForceRemotePlayerToSendTransform(User user)
    {
        SmartFoxClient client = NetworkController.GetClient();
        SFSObject data = new SFSObject();
        data.Put("_cmd", "f");
        data.Put("to_uid", user.GetId());
        client.SendObject(data);
    }
```

Rys. 4.5 Tworzenie reprezentacji zdalnego gracza w grze

Na **rysunku 4.5** widać wykorzystanie funkcji *SendMessage*. Pozwala ona na wywołanie innej funkcji danego obiektu dziedziczącego po *MonoBehaviou*, lub *Component*, . Pierwszym argumentem tej funkcji jest nazwa wywoływanej funkcji, a kolejnymi są argumenty wywoływanej funkcji. Pozwala to na połączenie programów napisanych w języku C# oraz skryptów w języku JavaScirpt.

Widoczny jest również sam proces wysyłania. Polega on na umieszczeniu odpowiednich danych w obiekcie typu *SFSObject*. Następnie obiekt wysyłany jest przy użyciu klasy *SmartFoxClient*. W podobny sposób wysyłane są informacje do rozszerzeń serwera. Jednak potrzebne informacje pakowane są do obiektu typu *Hashtable*, następnie obiekt ten jest wysyłany przy pomocy funkcji *SendXtMessage* znajdującej się także w klasie *SmartFoxClient*. Wewnątrz tej funkcji tworzony jest dopiero obiekt typu *SFSObject*, wysyłany do serwera.

Nad postaciami graczy wyświetlane są dodatkowe informacje, takie jak ich nicki oraz punkty zdrowia. Nick jest prezentowany za pomocą napisu *GUI.Label*, Punkty zdrowia zaś w sposób graficzny, za pomocą Ramki *GUI.Box* z odpowiednią teksturą. Samo wyświetlenie tych informacji nie jest problemem, ale problemem jest wyświetlenie ich w odpowiednim miejscu, tak aby widniały zawsze nad postacią gracza. *GUI.Label* oraz *GUI.Box* są elementami graficznego interfejsu użytkownika, oznacza to, że używają dwuwymiarowych współrzędnych oznaczających położenie na ekranie. Postać zaś znajduje się w trójwymiarowej przestrzeni. Należało więc znaleźć punkt przestrzeni dwuwymiarowej odpowiadający punktowi w przestrzeni trójwymiarowej znajdujący się nad postacią.

W tym celu została użyta funkcja *WorldToScreenPoint* klasy *Camera*. Funkcja ta wywołana została na głównej kamerze znajdującej się za postacią lokalnego gracza. Jako argumenty otrzymała trójwymiarowy, reprezentujący położenie zdalnego gracza zmodyfikowane o wysokość. Wartości zwrócone przez funkcję to dwuwymiarowy wektor, reprezentujący położenie na ekranie. Dzięki temu można wyświetlić informacje zawsze w odpowiednim miejscu. Efekt funkcjonalności widoczny jest na **rysunku 4.6**.

Rys. 4.6 Gra z widoczną postacią innego gracza

4.5.3 Przedmioty

Według założeń funkcjonalnych każda postać może posiadać przedmioty oraz magię. Magię można użyć, co skutkuje różnymi efektami. Przedmioty można założyć co wpływa na umiejętności ataku oraz obrony. Graficzna reprezentacja przedmiotów i magii została zrealizowana poprzez dwuwymiarowe tekstury. Tekstury wyświetlane są za pomocą funkcji *GUI.DrawTexture*, przyjmującej jako argumenty współrzędne na ekranie oraz podaną grafikę. Dodatkowo miejsca, w których mogą znajdywać się przedmioty charakteryzują się odmienną teksturą tła. Dla obsługi przedmiotów przechwytywane jest zdarzenie dotyczące użycia przycisku myszy. Jeśli nastąpiło użycie przycisku myszy w miejscu, gdzie może znajdować się przedmioty uruchamiane są odpowiednie funkcje stworzone na potrzeby projektu. Mają one za zadanie obsługiwać przekładanie przedmiotów gracza, a także umożliwić zobaczenie opisu przedmiotu.

Przedmioty posiadane przez gracza mają określony typ oraz rozmiar. Przedmioty takie jak tarcze, pancerze, topory i miecze mogą mieć podwójną wysokość i zajmować 2 miejsca. Jak widać na **rysunku 4.7** przedmioty mogą być umieszczone w trzech strefach. U góry widać ekwipunek. Znajdują się tam przedmioty założone, znajdując się tam wpływają pasywnie na atak i obronę postaci. Miejsca w których znajdują się przedmioty mogą wymagać konkretnego typu przedmiotu do umieszczenia. Poniżej widać plecak, mogą się tam znajdywać dowolne przedmioty. Na samym dole widnieje panel szybkiego dostępu, przeznaczony na czary.

Rys. 4.7 Gra z widocznymi przedmiotami

Przedmioty znajdujące się w panelu szybkiego dostępu u dołu ekranu mogą być użyte. W celu użycia przedmiotu należy użyć odpowiedniego klawisza numerycznego. Na **rysunku 4.8** przedstawiony jest sposób obsługi zdarzeń związanych z myszą i klawiaturą. Przedstawiony kod został uproszczony w celu przejrzystego zaprezentowania go.

Funkcja *Update*, wywoływana wraz z każdą klatką animacji uruchamia sprawdzanie aktualnych zdarzeń, znaczy to, że sprawdzanie zdarzeń następuje nieustannie. Jeśli zdarzenie istnieje, za pomocą jego atrybutów *e.isKey* oraz *e.isMouse* sprawdzany jest typ. W przypadku zdarzeń związanych z obsługą klawiatury sprawdzany jest wciśnięty przycisk (*e.keyCode == KeyCode.Alpha1*). Jeśli zdarzenie zostało wywołane przez mysz aktywny przycisk testowany jest z użyciem atrybutu *e.button*, dodatkowo *e.type* informuje czy przycisk został wciśnięty, czy zwolniony. W ten sposób następuje złożona obsługa przedmiotów.

```
void Update() {
      checkClick();
}


public void checkClick()
{
      Event e = Event.current;
      if (e != null)
      {
            if (e.isKey)
            {
                  if (e.keyCode == KeyCode.Alpha1)
                  {
                        // funkcja
                  }
            }else if(e.isMouse){
                  if (e.type == EventType.MouseDown && e.button == 0)
                  {
                        // funkcja
                  }
            }
      }
}
```

Rys. 4.8 Obsługa myszy i klawiatury

Graficzny interfejs użytkownika został wyposażony w podpowiedzi. Aby uzyskać podpowiedź, należy użyć prawego przycisku myszy na wybranym elemencie. Elementem tym może być każdy przycisk, mapa, przedmiot, lub czar. Spowoduje to pojawienie się przeźroczystej ramki z tekstem. Mechanizm ten został zaimplementowany w oparciu o obsługę zdarzeń związanych z myszą oraz funkcje *GUI.Box* i *GUI.Label*. Efekt działania funkcjonalności można zobaczyć na **rysunkach 4.9** i **4.10**

Rys. 4.9 Element graficznego interfejsu użytkownika z włączoną podpowiedzią

Rys. 4.10 Przedmiot z włączoną podpowiedzią

4.5.4 Interakcja z obiektami gry

Gracz może wchodzić w interakcję z obiektami w grze. W tym celu niezbędne jest wykorzystanie komponentu *Collider* dostarczanego wraz ze środowiskiem. *Collider* może być przypisany do obiektu posiadającego siatkę modelu. Przede wszystkim służy on to symulowania kolizji. Postać gracza posiada taki komponent dzięki czemu nie może przejść przez inny obiekt posiadający ten komponent. *Collidery* przypisane są takich obiektów jak: skały, drzewa, mury, most, budowle, gracze. Dodatkowo mechanizm ten pomaga w implementacji interakcji za pomocą myszy. Najechanie kursorem myszy na obiekt posiadający *Collider* pozwala wywołać funkcję *OnMouseOver* klasy *MonoBehaviour*. Funkcję tę można przeciążyć w klasie , przypisanej do obiektu, dziedziczącej po *MonoBehaviour*. Pozwala to na obsługę zdarzenia polegającego na najechanie na obiekt kursorem myszy, lub na kliknięciu w obiekt. Dzięki temu mechanizmowi została zaimplementowana walka między postaciami, przydał się on również do stworzenia interaktywnej skrzyni z przedmiotami oraz kapliczki z magią. Elementy te pozwalają na znaczną rozbudowę aplikacji w przyszłości.

4.5.5 Walka pomiędzy graczami

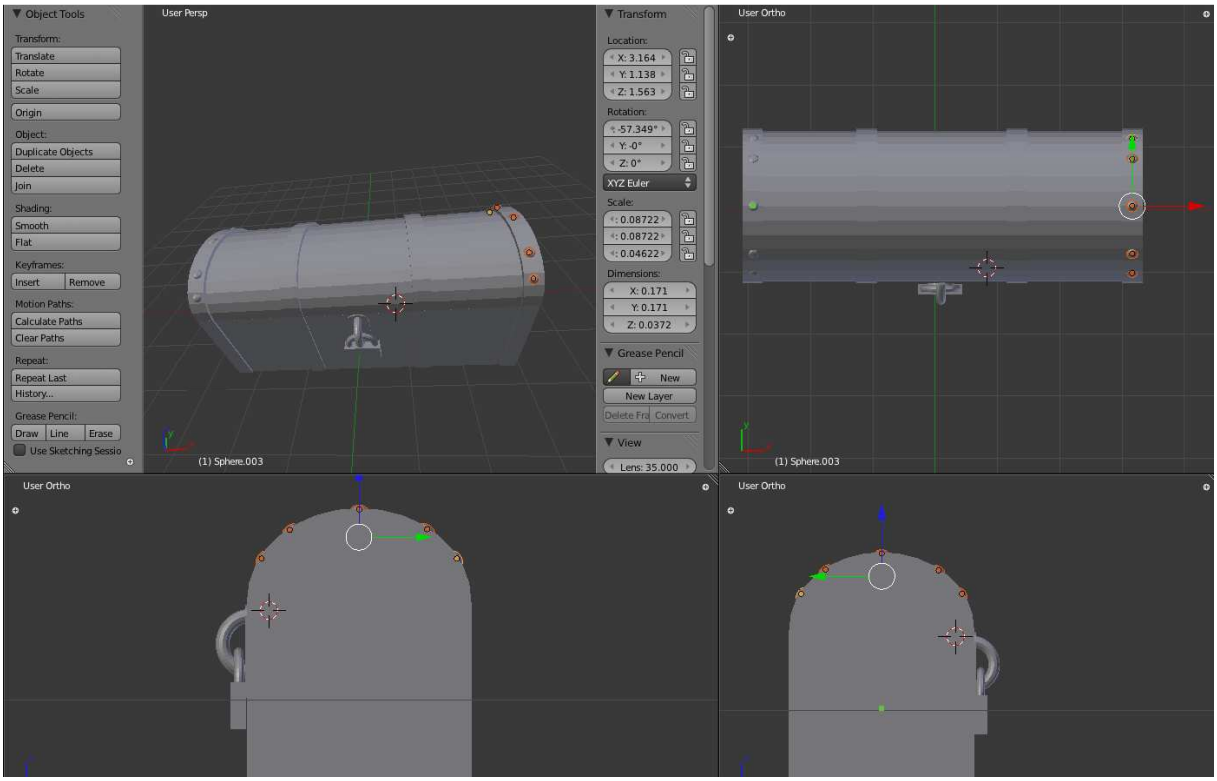
Jedną z głównych funkcjonalności gry jest możliwość walki pomiędzy postaciami. Aby zaatakować przeciwnika należy kliknąć na nim prawym przyciskiem myszy. Powoduje to wywołanie funkcji w sposób opisany w rozdziale „Interakcja z obiektami gry”. Na serwer wysyłana jest informacja o ataku, rozszerzenie serwera sprawdza, czy odległość między postaciami nie jest zbyt duża, a następnie oblicza zadawane obrażenia oraz liczbę punktów zdrowia. Obrażenia zależą od posiadanych przedmiotów, statystyk postaci, a także różnicy poziomów postaci. Informacje o zadanych obrażeniach wysyłane są do aplikacji zaatakowanego gracza. Ona zaś informuje aplikacje innych graczy o liczbie punktów zdrowia.

Jeśli punkty zdrowia postaci spadną do wartości zerowej postać umiera. Do wszystkich graczy w pokoju wysyłany jest komunikat o śmierci. U wszystkich graczy wizualizacja postaci poległego gracza staje się niewidoczna, pojawia się za to model reprezentujący zmarłego. Model ten od początku jest przypisany do obiektu i porusza się wraz z nim, jednak jest nieaktywny.

W walce przydatne są dwa przyciski graficznego interfejsu użytkownika. Przycisk automatyczne podążanie sprawia że atakując postać porusza się w stronę atakowanego. Przycisk automatyczny atak sprawia, że po zaatakowaniu postać będzie atakowana co sekundę w ciągu minuty. Użycie obu tych przycisków naraz sprawia, że postać jest atakowana co sekundę, a postać atakujaca przez cały ten czas za nią podąża.

4.5.6 Trójwymiarowe modele

Projekt wymagał stworzenia trójwymiarowych modeli, niestety było to bardzo trudne zadanie i nie w pełni udało się je wykonać. Większość modeli wykorzystanych modeli zostało dostarczone wraz ze środowiskiem Unity 3D. Jednak należało stworzyć animowany model postaci pasujący do świata gry, zadanie to okazało się zbyt trudne. Stworzony został model statyczny, który przydał się do pewnej funkcjonalności. Modelem tym jest skrzynia którą można zobaczyć na **rysunku 4.11** i **4.12.**



Rys. 4.11 Model skrzyni w programie Blender

Rys. 4.11 Model skrzyni w aplikacji klienckiej

5. Podsumowanie

5.1. Wady i zalety aplikacji

Głównymi zaletami projektu są te związane z wybranymi technologiami. Dzięki wykorzystaniu serwera Smart Fox Server powstała gra, w której uczestniczyć może wielu graczy, a stan rozgrywki może być zapisywany do bazy danych. Użycie silnika Unity 3D pozwoliło w prosty sposób stworzyć aplikację o efektownej trójwymiarowej grafice z ciekawymi efektami i animacjami. Niestety przyszły rozwój tego typu aplikacji wymaga stworzenia trójwymiarowych modeli, co nie jest prostym zadaniem.

Wykorzystany silnik graficzny ma jednak swoje wady. Pisanie gier w których bezpieczeństwo danych nie jest ważne byłoby przy użyciu tego rozwiązania o wiele prostsze. Jednak w przypadku tego projektu, bezpieczeństwo danych jest ważne. A inżynieria wsteczna aplikacji klienckiej nie jest bardzo trudna. W przypadku aplikacji osadzonej w przeglądarce internetowej istnieje możliwość ograniczenia adresów z jakich można łączyć się z serwerem. Jednak jeśli udostępniona jest możliwość połączenia się w z serwerem z pośrednictwem wolnostojącej aplikacji, można stworzyć własną zmodyfikowaną wersję.

Rozszerzenie serwera miało w taki sposób kontrolować rozgrywkę by nie dało się go oszukać. Jest ono napisane dobrze i przy odpowiednim rozbudowaniu go będzie weryfikować przesyłane mu dane co jest niewątpliwą zaletą. Dzięki temu próby oszukania serwera za pomocą własnej aplikacji klienckiej zostaną udaremnione.

Stworzenie własnej zmodyfikowanej aplikacji klienckiej daje możliwość oszukiwania innych graczy, poprzez wysyłanie im błędnych komunikatów dotyczących swojego położenia, nicku, lub liczby punktów życia. Przed tym aplikacja nie jest zabezpieczona, problem można rozwiązać poprzez wysyłanie całej komunikacji do rozszerzenia, jednak weryfikowanie danych zajmowałoby zbyt wiele czasu z powodu generowania obliczeń, zatem należy odrzucić takie rozwiązanie. Zatem zastosowano takie metody kontroli poczynań gracza by zbytnio nie obciążać serwera, ale jednocześnie mieć wystarczającą kontrolę.

Niewątpliwą zaletą projektu jest możliwość dołączenia do rozgrywki zarówno za pośrednictwem przeglądarki internetowej jak i aplikacji wolnostojącej. W obu przypadkach wydajność aplikacji jest na wysokim poziomie.

5.2. Możliwości rozwoju

Temat projektu jest szeroki, a samą aplikację można w nieskończoność rozszerzać. Pomysły na rozbudowę można opierać między innymi na grach tego typu. Wprowadzenie elementów, które sprawiłyby, iż można by wdrożyć projekt polega głównie na stworzenie trójwymiarowych modeli postaci oraz dodatkowej grafiki.

Przede wszystkim projekt można rozwinąć o stronę internetową korzystającą z bazy danych gry. Umożliwiałaby ona rejestrację nowego gracza, pozwalała przeglądanie elementów rozgrywki, lub wyświetlanie statystyk postaci. Strona służyłaby również jako źródło informacji o grze oraz element reklamowy.

Projekt można również rozbudować tworząc dodatkowe modele jak i grafikę. Mogłaby być to grafika przedmiotów oraz „magii", a także modele postaci, elementów świata i potworów występujących w grze. Pozwoliłoby to na dodanie nowych przedmiotów i rozbudowanie „magii".

Gra została przygotowana w taki sposób aby można było rozszerzyć rozgrywkę o dodatkowe światy. Należy w tym celu stworzyć kolejną scenę, za określenie świata w którym przebywa gracz odpowiada w bazie danych kolumna *world* tabeli *Tempdata*. W aplikacji klienckiej przygotowane zostały funkcje pozwalające operować scenami. Przejście do innego świata mogłoby opierać się na podejściu do określonego obiektu i kliknięciu na nim. Mogłaby to być na przykład pokryta ciemną teksturą figura symbolizująca wejście do jaskini.

Ważnym aspektem jest poprawienie kilku szczegółów mało ważnych ze względu na sprawy techniczne, ale ważnych ze względu na wygodę gracza. Jedną z takich rzeczy jest sterowanie postacią i obroty kamery. Dopracowanie polegałoby na ustawieniu innych wartości liczbowych w kilku miejscach kodu.

Ciekawym pomysłem rozbudowy aplikacji jest mechanizm wyszukiwania drogi. Miejsca na planszy powinny zostać odwzorowane do postaci grafu. Wierzchołki grafu symbolizowałyby możliwe położenie gracza, a krawędzie możliwe przejścia między połączeniami. W ten sposób, wykorzystując na przykład algorytm Forda-Bellmana, byłaby możliwość znajdywania najkrótszej ścieżki między wybranymi miejscami. Gracz, przy pomocy myszki, mógłby poprowadzić postać przez skomplikowany teren tak, by zawsze dotarła do celu. Ponieważ wymagałoby to dodatkowych obliczeń, powinny one być wykonywane w osobnym wątku o niskim priorytecie, aby obliczenia nie obciążały zbytnio aplikacji.

Należałoby również dokończyć funkcjonalności nie zaplanowane jako elementy projektu, ale zaplanowane w interfejsie użytkownika:

- Zakładka „znajomi”: lista znajomych
- Czat: globalny, graczy w pobliżu, pokoju na serwerze, prywatny.
- Menu kontekstowe po kliknięciu na gracza z pozycjami: atak, handel, rozmowa.
- Zakładka „bitwa”: Lista graczy znajdujących się w pobliżu z możliwością użycia menu kontekstowego.
- Zakładka „magia”: Lista czarów jakie zna postać.
- Zakładka „opcje”: dodatkowe ustawienia gry, dotyczące na przykład szczegółowości grafiki, ustawień dźwięku i ustawień gry.
- Skrzynia służąca za depozyt w której można pozostawić przedmioty i pieniądze.

Aplikacja powinna zostać również rozbudowana o potwory pojawiające się na mapie. Powinny one się znajdować u każdego gracza w tym samym miejscu i wykonywać te same akcje. Atakować graczy w zależności od odległości. Po zabiciu potwora gracz powinien otrzymać doświadczenie. A z martwego ciała powinno dać się wyjąć jakieś przedmioty.

5.3. Zakończenie

Pomysłów na dalszą rozbudowę może być bardzo wiele, a ponieważ wraz z tworzeniem aplikacji wciąż ich przybywa, jednak cel pracy został w pełni osiągnięty. Projekt został doprowadzony do końca. Wybrane technologie spełniły wymagane oczekiwania. Powstała gra dla wielu graczy, która choć oczywiście nie dorównuje rynkowym produktom jest w pełni działającą aplikacją. Aplikacja działa poprawnie i wydajnie zarówno w wersji wolnostojącej jak i w wersji osadzonej w przeglądarce internetowej.

6. Bibliografia

[1] Archive 3D. Free 30 000+ 3D models, 2007-2011, data dostępu 7.09.2011-5.12.2011, http://archive3d.net

[2] GotoAndPlay. SmartFoxServer Massive Multiplayer Platform, 2004-2010 , data dostępu 21.06.2011-5.12.2011, http://www.smartfoxserver.com/

[3] GotoAndPlay. SmartFoxServer User Documentation, 2004 – 2006, data dostępu 21.06.2011-5.12.2011, http://www.smartfoxserver.com/docs/

[4] Ryan Henson Creighton. Unity 3D Game Development by Example Beginner's Guide, 2010.

[5] Unity Technologies. UNITY: Game Development Tool, 2011, data dostępu 13.05.2011-5.12.2011, http://unity3d.com/

[6] Unity Technologies. Unity Manual, 2011, data dostępu 13.05.2011-5.12.2011,, http://unity3d.com/support/documentation/Manual/index.html

[7] Unity Technologies. Unity Script Reference, 2011, data dostępu 13.05.2011-5.12.2011,, http://unity3d.com/support/documentation/ScriptReference/index.html

[8] Will Goldstone. Unity Game Development Essentials, 2009.

7. Załącznik

7.1. Struktura plików serwera

Najważniejszymi folderami w strukturze plików serwera są:

- Folder SmartFoxServerPRO_1.6.6\jre\lib\ext
  - Dodatkowe **pliki jar** rozszerzające funkcjonalność serwera
  - Pliki, które zostały tam dodane w projekcie:
    - **mysql-connector-java-5.0.8-bin.jar**
    - **mm.mysql-2.0.14-bin.jar**

- Folder SmartFoxServerPRO_1.6.6\Server
  - Pliki konfiguracyjne - **config.xml**
  - Pliki *.bat uruchamiające Smart Fox Server oraz **policy-server**

- Folder SmartFoxServerPRO_1.6.6\Server\sfsExtensions
  - Rozszerzenia w języku Action Script
  - Używane rozszerzenia należy dopisać również do plików konfiguracyjnych
  - Pliki które zostały tam dodane w projekcie:
    - auth.as
    - chars.as
    - createCharacter.as
    - loadCharacter.as
    - saveCharacter.as
    - Pliki te nie zostały jednak użyte w końcowej wersji

- **Folder SmartFoxServerPRO_1.6.6\Server\javaExtensions**
  - Rozszerzenia w języku Java
  - Pliki w postaci *.class lub *.jar
  - Pliki które zostały tam dodane w projekcie:
    - **gameServ.jar**

- Folder SmartFoxServerPRO_1.6.6\Server\logs
  - Zawiera logi serwera
  - Pomocne przy debugowaniu

7.2. Konfiguracja serwera

Smart Fox Server można konfigurować za pomocą pliku config.xml zapisanego w języku XML.

Najważniejsze elementy konfiguracji:

- smartFoxConfig
    - serverSetup
        - serverIP –adresy z których można odwołać się do serwera
        - serverPort – port na którym można komunikować się z serwerem
        - policyAllowedDomains – domeny z których można się łączyć z serwerem za pośrednictwem aplikacji umieszczonej w przeglądarce internetowej
        - maxSocketIdleTime – maksymalny czas bezczynności gniazdka, mierzony w sekundach, po tym czasie gniazdko zostanie rozłączone
        - maxUserIdleTime – maksymalny czas bezczynności użytkownika, mierzony w sekundach, po tym czasie użytkownik zostanie odłączony
        - maxWriterQueue – maksymalna liczba komunikatów w kolejce oczekujących na wysłanie

- zones – definiowanie stref
  - zone
    - rooms – definiowanie pokoi
      - name – nazwa pokoju
      - pwd – hasło do pokoju
      - maxUsers – maksymalna liczba użytkowników
      - isPrivate – określa czy pokój jest prywatny, jeśli tak wymagane jest hasło
      - autoJoin – określa czy użytkownik ma być automatycznie dołączany do tego pokoju za pośrednictwem funkcji autoJoin

    - extensions – rozszerzenia serwera
      - name – nazwa rozszerzenia, za pomocą tej nazwy można odwołać się do rozszerzenia
      - className – nazwa pliku zawierającego rozszerzenie, plik musi być umieszczony w odpowiednim katalogu, różnym w zależności od języka w którym został napisany
      - type – określa język w którym zostało napisane rozszerzenie, może przyjmować wartości „Scripts”, „Python”, „Java”

    - databaseManager – dostęp do bazy danych
      - connectionString – pełny adres bazy danych
      - driver – pełna nazwa sterownika bazy danych
      - userName – nazwa użytkownika bazy danych potrzebna do autoryzacji
      - password – hasło użytkownika bazy danych potrzebne do autoryzacji
      - testSQL – testowe zapytanie to bazy danych pozwalające sprawdzić połączenie podczas uruchamiania serwera
      - MaxActive – maksymalna liczba aktywnych połączeń z bazą danych
      - OnExhaustedPool – określa co zrobić w przypadku przeciążonej bazy danych