

Dokumentacja techniczna

Projekt Yet Another Dune II

2007-11-07



SPIS TREŚCI

1. Wstęp	5
2. Technologie	5
2.1. Grafika	5
2.2. Dźwięk	5
2.3. Sieć	5
3. Interfejs użytkownika	5
3.1. Menu	5
3.1.1. Menu główne gry	5
3.1.2. Ekran opcji	6
3.1.3. Opcje dostępne w trakcie gry	7
3.1.4. Ekran pauzy	8
3.2. Panel gry sieciowej	8
3.2.1. Ekran logowania	8
3.2.2. Ekran rejestracji użytkownika	9
3.2.3. Ekran czatu	10
3.2.4. Ekran informacji o użytkowniku	11
3.2.5. Ekran rozgrywki	11
3.2.6. Ekran tworzenia gry	12
3.2.7. Ekran oczekiwania na graczy	13
3.3. Ekran rozgrywki	13
3.4. Drzewo przejść pomiędzy elementami menu	15
4. Implementacja silnika gry	15
4.1. Logika ładowania elementów – danych gry	16
4.2. Logika budowania budynków	18
4.3. Logika tworzenia jednostek	20
4.4. Logika ruchu jednostek	21
4.5. Logika walki jednostek	23
4.6. Logika jednostek specjalnych	25
4.7. Logika pracy zegara z naciskiem na animację elementów	25
5.1. Logika dźwięków	26
5.2. Schemat modułów	27
5.3. Schemat klas z opisem	27
6. Implementacja gry sieciowej	29
6.1. Opis realizacji gry sieciowej	29
6.1.1. Serwer	29
6.1.2. Struktury danych wykorzystywane w aplikacji serwerowej	35
6.1.3. Rozpoczęcie rozgrywki	35
6.1.4. Rozgrywka sieciowa	37
6.1.5. Wiadomości rozgrywki	38
6.1.6. Wiadomości gry	39
6.1.7. Obsługa zakończenia tury i pauzy	39
6.1.8. Przetwarzanie wiadomości	39
6.1.9. Przetwarzanie komunikacji w silniku gry	40
7. Sytuacje wyjątkowe	41

Piotr Witosławski Adam Nowacki Paweł Rokoszyński Kamil Ślesieński Radosław Stankiewicz	<h1>Yet Another Dune II</h1> <p>Identyfikator Projektu: yad2</p>	
---	--	---

7.1.	Modyfikacja zasobów	41
7.2.	Brak połączenia z serwerem	42
7.3.	Zerwanie połączenia podczas gry – strona klienta	42
7.4.	Zerwanie połączenia podczas gry – strona serwerowa	42
7.5.	Długotrwałe opóźnienie podczas gry – strona serwerowa	42
8.	Podział prac i fazy projektu.....	42
8.1.	Zadania	42
8.1.1.	Zadania pierwszego tygodnia	42
8.1.2.	Zadanie drugiego tygodnia	42
8.1.3.	Zadania trzeciego tygodnia	43
8.1.4.	Zadania czwartego tygodnia	43
8.1.5.	Zadania piątego tygodnia	43
8.1.6.	Zadania szóstego tygodnia	43
8.1.7.	Tydzień siódmy	43
8.1.8.	Tydzień ósmy	43
8.2.	Fazy projektu	43
8.2.1.	Wersja wstępna	44

Piotr Witosławski Adam Nowacki Paweł Rokoszný Kamil Ślesieński Radosław Stankiewicz	<h1>Yet Another Dune II</h1> <p>Identyfikator Projektu: yad2</p>	
--	--	---

Dane Dokumentu

Nazwa Projektu:	YAD II	Nr Wersji:	0.9.3
Opracowany Przez:	Piotr Witosławski	Data Wersji:	2007-11-07
Rola:	Szef zespołu	Data Przeglądu:	
Sprawdzony Przez:			

Historia Zmian

Nr Wersji	Data Wersji	Zmiany Wprowadził(a)	Opis Zmian	Recenzent	Status
0.1.0	2007-10-20	Piotr Witosławski	Utworzenie dokumentu		
0.1.1	2007-10-23	Adam Nowacki	Organizacja dokumentu (spis treści, numeracja)		
0.2.0	2007-10-23	Paweł Rokoszný	Dodanie: interfejs użytkownika		
0.3.0	2007-10-24	Radek Stankiewicz	Dodanie: logika jednostek i wzorce projektowe		
0.4.0	2007-10-24	Adam Nowacki	Dodanie: technologie, logika ładowania elementów, animacja		
0.5.0	2007-10-24	Piotr Witosławski	Dodanie: implementacja gry sieciowej, fazy projektu		
0.6.0	2007-10-25	Radek Stankiewicz	Dodanie: opis klas		
0.7.0	2007-10-25	Kamil Ślesieński	Modyfikacja: technologie: dźwięk, logika komunikatów, dodanie: logika budowania budynków, logika budowania jednostek		
0.7.1	2007-10-25	Adam Nowacki	Poprawa diagramu XSD		
0.7.2	2007-10-27	Paweł Rokoszný	Wstępna poprawka w interfejsie		
0.7.3	2007-10-27	Paweł Rokoszný	Poprawka w interfejsie		
0.7.4	2007-10-30	Kamil Ślesieński	Poprawki w logice budowania budynków i jednostek		
0.8.0	2007-10-31	Adam Nowacki	Dodanie schematu modułów		
0.9.0	2007-11-03	Radek Stankiewicz	Dodanie sytuacji wyjątkowych		
0.9.1	2007-11-06	Paweł Rokoszný	Małe poprawki interfejsu		
0.9.2	2007-11-06	Piotr Witosławski	Dodanie diagramów		
0.9.3	2007-11-07	Radek Stankiewicz	Drobne poprawki opisu klas		



1. Wstęp

Dokument stanowi dokumentację techniczną realizacji gry Yet Another Dune II i stanowi podstawę implementacji projektu realizowanego w ramach przedmiotu „Pracownia programowania”

2. Technologie

2.1. Grafika

Grafika w grze zostanie wykonana w technologii *.NET Framework 2.0* w połączeniu z biblioteką *TAO Framework*. Interfejs gry zostanie wykonany przy użyciu *.NET Windows Forms*, jednak z uwzględnieniem grafik użytych w oryginalnym „Dune II”, tak by nie utracić pierwotnego charakteru gry.



Część grafiki odpowiedzialna za wyświetlanie świata gry zostanie zaimplementowana z wykorzystaniem biblioteki *TAO Framework*. Technologia ta umożliwia użycie w *.NET Framework* dobrodziejstw jednej z najpopularniejszych bibliotek do obsługi grafiki – *OpenGL*.

Takie rozwiązanie pozwoli na szybkie wyświetlanie grafiki związanej ze światem gry, przy jednoczesnym zachowaniu prostoty rozbudowy interfejsu użytkownika.

2.2. Dźwięk

Do odtwarzania dźwięków i muzyki w grze zostanie użyta biblioteka *OpenAL*, również wchodząca w skład *TAO Framework*. Decyzję o skorzystaniu z tej biblioteki podjęto, aby jeszcze bardziej zintegrować kod projektu z i tak już wybraną do obsługi grafiki biblioteką *TAO Framework*. Dźwięki i muzyka w oryginalnym „Dune 2” były przechowywane w formacie MIDI oraz WAV i takie formaty wykorzystamy w swoim projekcie.

2.3. Sieć

Do implementacji rozgrywki sieciowej zostaną wykorzystane możliwości udostępniane przez platformę *.NET Framework* (w tym klasy *TcpClient* i *TcpListener* z przestrzeni nazw *System.Net*). Klasy te implementują podstawowe mechanizmy do połączenia, wysyłania i odbierania danych za pomocą protokołu TCP.

Zdecydowano się na użycie tego protokołu, gdyż z charakteru rozgrywki sieciowej wynika, że nie można sobie pozwolić na utratę danych przesyłanych po sieci. Dlatego też zastosowanie UDP jest niemożliwe, a wybrano nieco wolniejszy protokół ze względu na jego niezawodność.

Dodatkowo zastosowanie na serwerze wielowątkowości umożliwi obsługiwanie wielu klientów jednocześnie (szczegółowy opis patrz 5.1).

3. Interfejs użytkownika

3.1. Menu

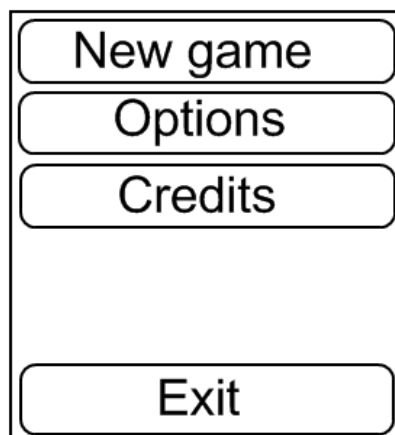
Poniżej przedstawione są ekrany menu, które będą się pojawiać w trakcie gry.

Menu gry jest nastawione na poruszanie się po nim kursorem myszy. Wybranie odpowiedniej opcji odbywa się poprzez pojedyncze kliknięcie na niej. Kolorystyka ekranów oraz ich otekstowanie zostało pominięte celem skupienia się na funkcjonalności

Należy też już na wstępie nadmienić, że wszędzie tam, gdzie występują listy (chat, nazwy użytkowników) będą wyświetlane suwaki do przewijania tych list.

3.1.1. Menu główne gry

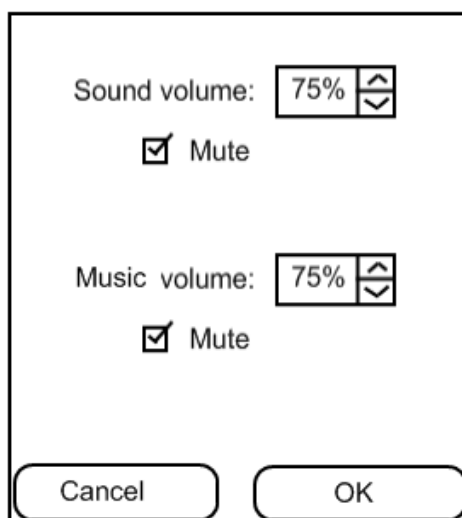
Menu główne gry będzie miało wygląd, jak na poniższym rysunku:



Rysunek 1 Menu główne gry

- New Game* – powoduje przejście do ekranu rozgrywki sieciowej (opisany później)
Options – przejście do ekranu opcji gry
Credits – wyświetla informację o autorach gry
Exit – opuszczenie gry

3.1.2. Ekran opcji



Rysunek 2 Ekran opcji

Użytkownik ma do dyspozycji opcje regulowania głośności dźwięków gry oraz muzyki tła. Może też całkiem wyciszyć dźwięki.

Sound volume służy go ustawiania dźwięków pochodzących z gry, odgłosów budowania, walki itd. Natomiast *Music volume* służy do regulacji natężenia muzyki będącej tłem rozgrywki.

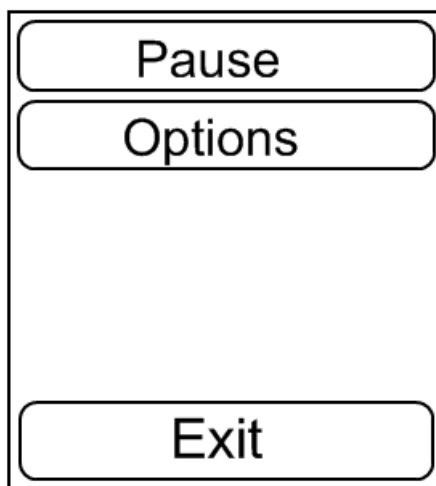
Klikając na kwadrat znajdujący się po lewej stronie słowa *Mute* użytkownik wycisza lub na nowo włącza regulowany właśnie typ dźwięków. Jeśli kwadrat jest pusty, dźwięk nie jest wyciszony i jest dla użytkownika słyszalny. W przeciwnym przypadku dźwięk jest wyciszony i jest niesłyszalny.



Regulacja głośności odbywa się poprzez klikanie na guziki z sygnaturkami trójkątów znajdujące się obok pola wskazującego procentowe natężenie dźwięku. Klikając na trójkąt skierowany do góry użytkownik podniesie głośność dźwięku o 5%. Klikając na trójkąt skierowany ku dołowi – 5% ciszej.

W przypadku, kiedy użytkownik zmienił ustawienia dźwięków, ale wcisnął klawisz *Cancel*, zostanie wyświetlony wcześniejszy ekran, a dane o zmianach zostaną utracone. Aby zapisać te dane, wymagane jest naciśnięcie klawisza *OK*. Wtedy również nastąpi powrót do poprzedniego ekranu, ale ze zmienionymi ustawieniami dźwięków.

3.1.3. Opcje dostępne w trakcie gry



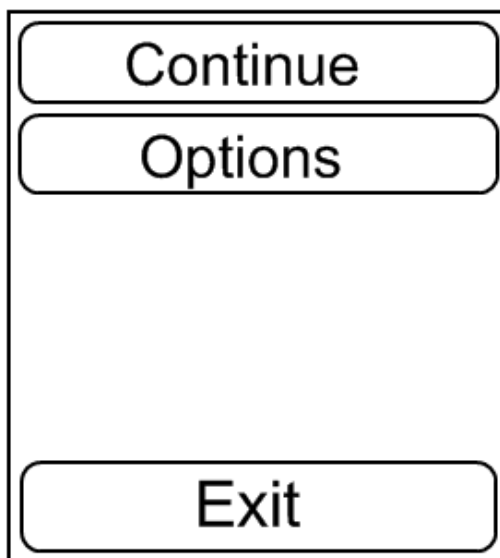
Rysunek 3 Menu główne w trakcie gry

- Pause* - zapauzuje grę; użytkownik ma ograniczone możliwości pauzowania gry, ponieważ gra odbywa się przez sieć z innymi użytkownikami
- Options* - ekran opcji (jak w głównym menu gry)
- Exit* - zakończenie gry i wyjście od razu do systemu operacyjnego.

Menu to zostanie wyświetlone użytkownikowi, gdy w trakcie gry kliknie on raz na przycisk *Options* na górze ekranu. Wówczas jednak nie przzerwana, ani nie wstrzymana zostanie gra, a jedynie wyświetlona plansza menu. Może to doprowadzić do tego, że gracz włączy na dłuższy czas ten ekran, a gdy wróci, gra będzie na znacznie dalszym etapie, niż przed włączeniem ekranu.



3.1.4. Ekran pauzy



Rysunek 4 Menu dostępne w trakcie pauzy

Ekran wyświetlany jest, jeśli gra jest zapauzowana. Widoczny jest dla każdego użytkownika (niezależnie od tego kto uaktywnił pauzę), przez co uniknie się sytuacji, gdzie jeden użytkownik jest nieobecny, a reszta umacnia swoje siły w grze. Dodatkowo, dowolny użytkownik może w dowolnym momencie przerwać pauzę wznowiając grę.

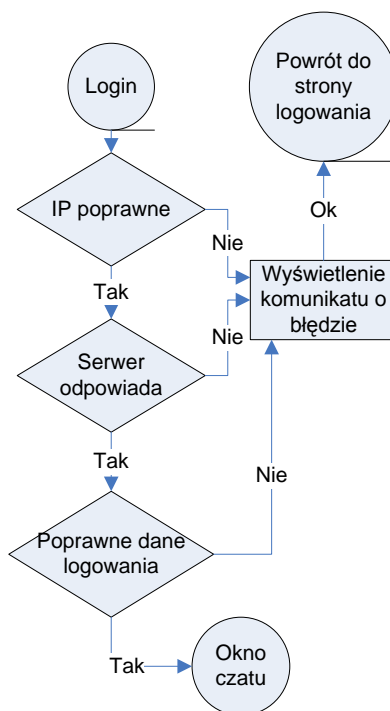
3.2. Panel gry sieciowej

Po wybraniu opcji „Nowa Gra” oczom gracza ukazuje się ekran logowania:

3.2.1. Ekran logowania

Rysunek 5 Ekran logowania

Logowanie będzie się odbywać według następującego schematu:



3.2.2. Ekran rejestrowania użytkownika

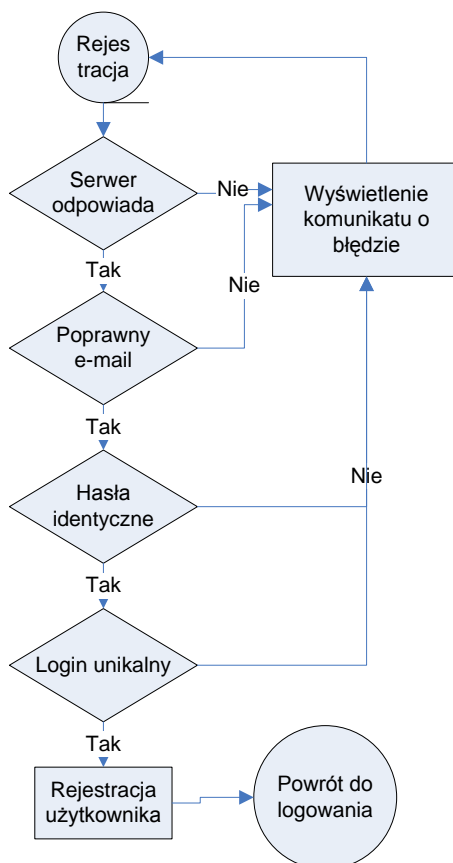
Umożliwia on rejestrację i zalogowanie na odpowiednim serwerze, a także przypomnienie hasła przy pomocy poczty elektronicznej. Adresem serwera jest jego numer IP i port, na którym nasłuchuje. Hasło będzie maskowane gwiazdkami.

Gdy wybrana zostanie opcja Register, pojawi się okno rejestracji:

Login:	<input type="text"/>
Password:	<input type="password"/>
Repeat password:	<input type="password"/>
E-mail:	<input type="text"/>
<div>Back Register</div>	

Rysunek 6 Rejestracja użytkownika

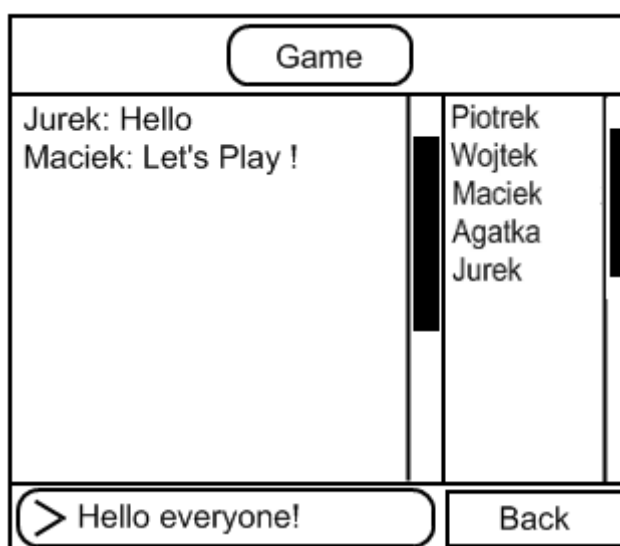
Tworzenie konta użytkownika będzie się odbywało według następującego schematu (po naciśnięciu klawisza „Register”):



Po wpisaniu loginu, dwukrotnie hasła oraz adresu e-mail i wciśnięciu przycisku Register użytkownik zostanie zarejestrowany na serwerze pod warunkiem, że login, który podał jest unikatowy. Adres e-mail jest niezbędny do przywrócenia hasła do konta.

Po zalogowaniu ukazuje się główne okno:

3.2.3. Ekran czatu

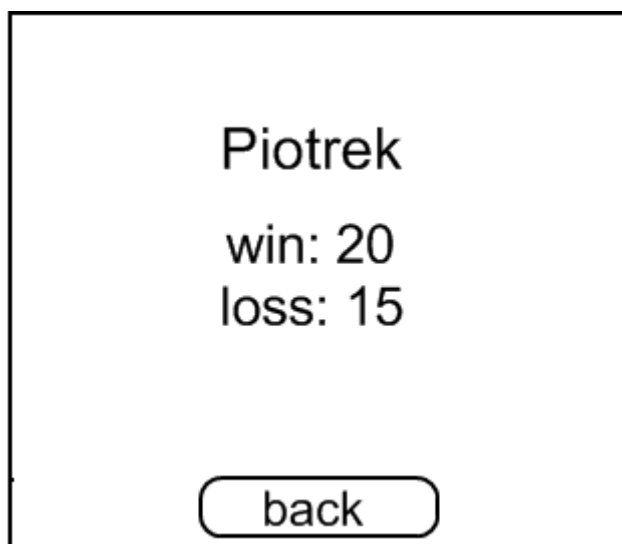


Rysunek 7 Okno czatu



Lewa część okna odpowiada za czat między użytkownikami. Po prawej stronie znajduje się lista użytkowników – po wciśnięciu imienia dowolnego z nich otworzy się okno ze statystykami: loginem, liczbą zwycięstw i porażek. Wciśnięcie *Game* powoduje rozpoczęcie przejście do panelu definicji rozgrywki

3.2.4. Ekran informacji o użytkowniku



Rysunek 8 Okno informacji o użytkowniku

Ekran informacji o użytkowniku zawierał będzie jego login, Liczbę wygranych oraz liczbę przegranych gier.

Wciśnięcie klawisza *back* spowoduje powrót na ekran czatu.

3.2.5. Ekran rozgrywki

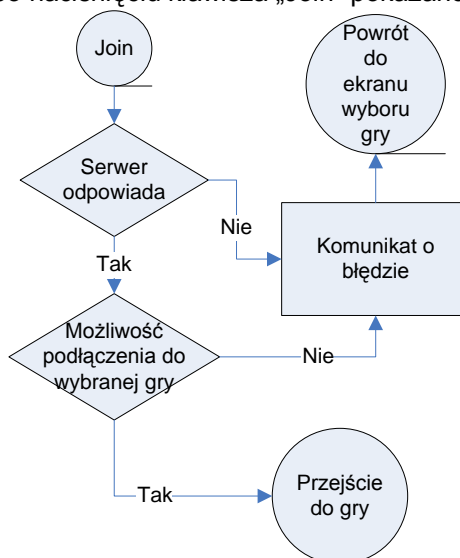
Ekran rozgrywek przedstawia się następująco:



Rysunek 9 Ekran gry dowolnej



Schemat łączenia do gry po naciśnięciu klawisza „Join” pokazano poniżej:



W panelu wyboru gry gracz będzie miał możliwość wyboru, do jakiej publicznej gry chce dołączyć.

Gry identyfikowane będą poprzez nazwę – musi być ona unikatowa. Jeśli użytkownik chce dołączyć do gry prywatnej, będzie musiał wpisać jej nazwę przy pomocy klawiatury do odpowiedniego okienka. Wciśnięcie przycisku Create spowoduje przejście do ekranu tworzenia gry:

3.2.6. Ekran tworzenia gry

2.Textbox nazwy gry

game_name

private

public

Sandstorm

The Last Battle

Three hills

1. Wybór mapy

Create

Cancel

Rysunek 10 Ekran tworzenia gry

Użytkownik przy pomocy kontrolki wybiera odpowiednią mapę, wpisuje nazwę gry, a także zaznacza, czy jest to gra prywatna, czy też publiczna. Gry prywatnie nie zostają wyświetlone w oknie dostępnych gier w oknie rozgrywek dowolnych. Po wciśnięciu przycisku Create otwiera się okno oczekiwania na graczy:



3.2.7. Ekran oczekiwania na graczy

Piotrek	Ordos ▼	Team 1 ▼
Agatka	Ordos	Team 2
Maciek	Ordos	Team 2

Name:
MyGame
Map:
High hills

1. Okno graczy

2. Okno informacji

Canel

Start game

Rysunek 11 Ekran oczekiwania na graczy

Wykorzystywane jest ono przy dołączaniu do gry. Po jego lewej stronie znajdują się gracze, którzy dołączyli do gry. Mają oni możliwość wyboru rasy i drużyny w której będą grać. Gdy każdy z graczy naciśnie przycisk „Start game” rozgrywka się rozpoczyna (według schematu przedstawionego w opisie ekranu rozrywki).

3.3. Ekran rozgrywki

Poniżej widoczny jest przykładowy ekran rozgrywki w trakcie gry (ekran w ostatecznej wersji gry może odbiegać od poniższego ekranu, jednak opisane elementy zostaną zachowane):



Rysunek 12 Ekran gry

- 1 – przycisk otwierający menu dostępne dla użytkownika w trakcie gry (patrz 3.1.4)
- 2 – ilość dostępnych w grze kredytów, używanych m.in. do budowy nowych obiektów i jednostek (patrz 4.2 i 4.3)
- 3 – wybór obiektu do zarządzania.
Zmiana w tym panelu powoduje zmianę widocznych budowli/jednostek do wybrania w panelu 4 (związanych z danym wybranym z tego menu obiektem). Dostępne budynki do zarządzania to - Contruction Yard, Barracs, Light Factory, Heavy Factory, House of IX, Starport. Jeśli jakiś budynek jest niedostępny dla gracza (tzn. nie został jeszcze wybudowany) to przycisk do jego obsługi jest wyszarzony.
- 4 – wybór budowli/jednostki do postawienia.
Widoczne są tylko budowle/jednostki, które gracz może zbudować. Wybór opcji LPM oznacza rozpoczęcie budowy danej jednostki/budowli. Jest ona stopniowo odszarzana, a następnie w przypadku budynków pojawia się na niej napis „PLACE ON MAP”. Wybór wtedy tej opcji LPM i następnie kliknięcie LPM w miejsce na mapie, w którym budowa obiektu jest dozwolona oznacza pojawienie się tam danego budynku. W przypadku jednostek bojowych po wyszkoleniu jednostka pojawia się od razu w pobliżu obiektu, w którym może być tworzona (jeśli jest więcej niż jeden takich obiektów, pojawia się przy obiekcie wybudowanym na mapie w pierwszej kolejności)
- 5 – podgląd całej mapy rozgrywki z zaznaczonymi budynkami i jednostkami, tzw. radar
Na mapie odzwierciedlony jest stan odkrytego terenu walki (nie odkryte tereny zaznaczone są na czarno). Widoczna jest również ramka, która wskazuje, jaki obszar aktualnie jest obserwowany



na ekranie pola rozgrywki. Mapa używana jest także przy nawigacji. Wybranie LPM danego punktu na mapie automatycznie przesuwa obszar widoczności na ten punkt.

6 – właściwa plansza rozgrywki (pole walki)

Na ekranie pola walki można obserwować wycinek planszy, na której prowadzone są działania. Widoczny jest na nim teren, budynki, jednostki oraz wszelkie zjawiska i efekty graficzne. Na ekranie można nie tylko śledzić przebieg potyczki, ale także zaznaczać jednostki i budynki. Po wybraniu budowli/jednostki LPM nad jednostką/budowlą pojawi się pasek stanu reprezentujący ilość energii, jaka pozostała jednostce/budowli. Nie odkryte obszary na ekranie są zasłonięte czarną mgłą (czarny teren). Jeśli na jakiejś jednostce zostanie wykonany dwuklik, wykonana będzie akcja specjalna właściwa dla tej jednostki (na dla mcv spowoduje rozłożenie go w pełni funkcjonalną bazę).

3.4. Drzewo przejść pomiędzy elementami menu

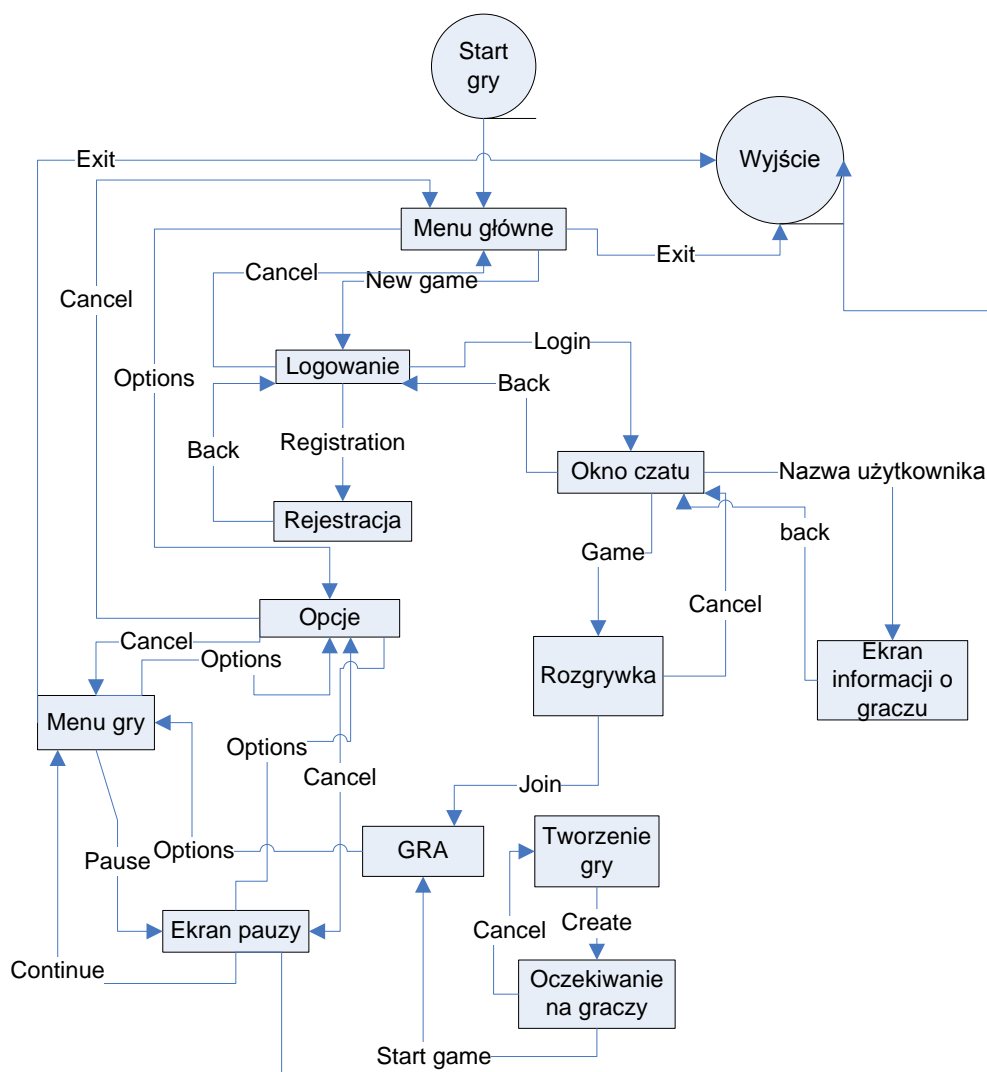
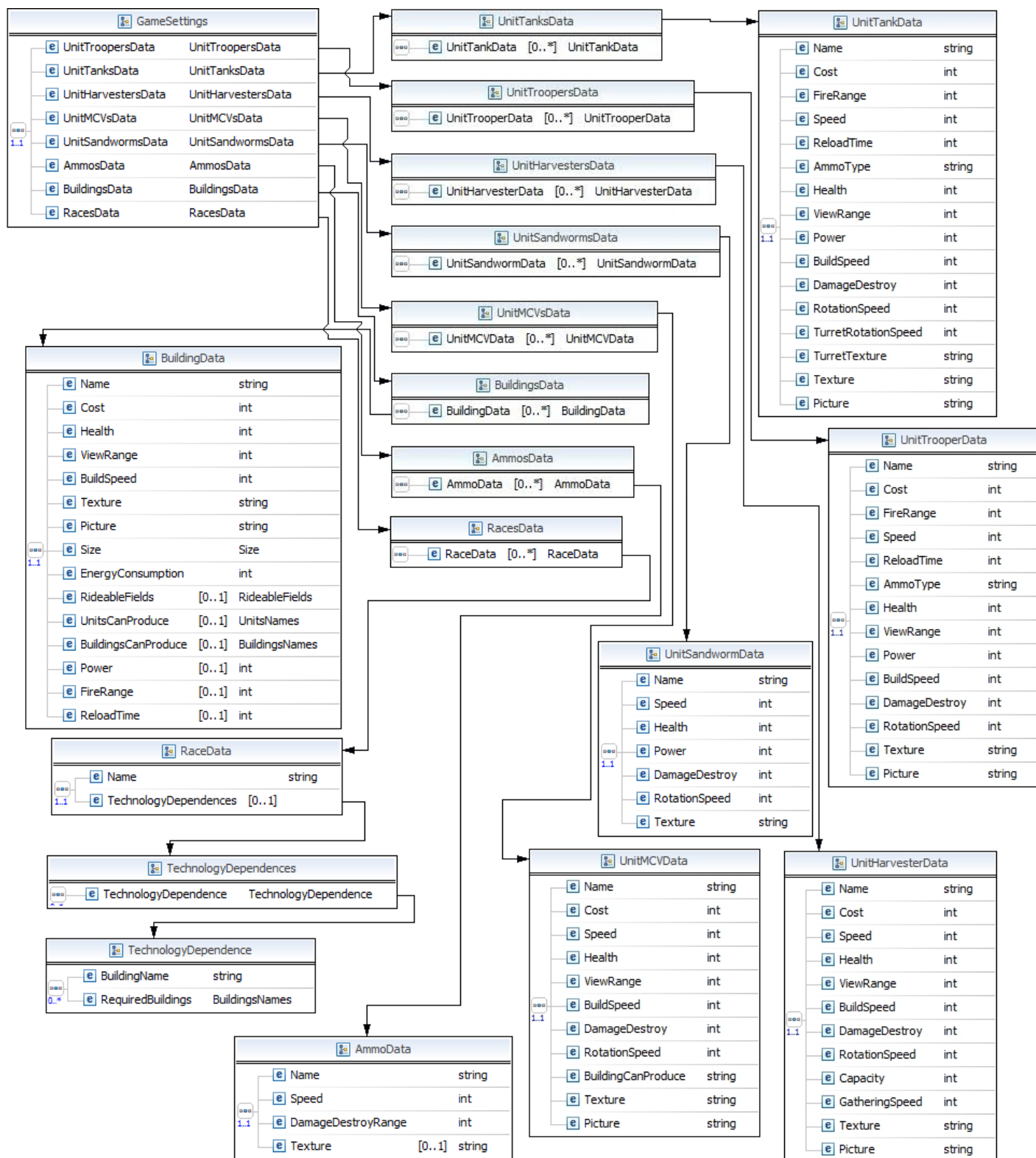


Diagram ten przedstawia wszystkie możliwe przejścia i połączenia między poszczególnymi oknami menu. Strzałki opisane są nazwami przycisków, których naciśnięcie skutkuje wykonaniem akcji.

4. Implementacja silnika gry



4.1. Logika ładowania elementów – danych gry



Rysunek 13 Schemat pliku XSD z ustawieniami gry



Przy uruchamianiu gry z pliku XML zostaną wczytane dane opisujące charakterystykę gry. Na rysunku 15 przedstawiony został schemat pliku XSD do walidacji wczytywanego pliku konfiguracyjnego.

Najważniejsze elementy w pliku XML zawierają:

- **UnitTanksData** – opis czołgów występujących w grze. Pojedynczy czołg jest opisywany przez następujące elementy:
 - Name – nazwa jednostki
 - Cost – koszt produkcji jednostki
 - FireRange – zasięg strzału jednostki
 - Speed – prędkość poruszania się jednostki
 - ReloadTime – czas przeładowywania po strzale
 - AmmoType – typ amunicji używanej przez jednostkę
 - Health – wytrzymałość jednostki
 - ViewRange – zakres widzenia
 - Power – siła strzału
 - BuildSpeed – szybkość budowania
 - DamageDestroy – straty zadawane jednostkom na sąsiednich polach po zniszczeniu jednostki
 - RotationSpeed – szybkość obrotu jednostki
 - TurretRotationSpeed – szybkość obrotu działa czołgu
 - TurretTexture – ścieżka do pliku z teksturą działa czołgu
 - Texture – ścieżka do pliku z teksturą jednostki
 - Picture – obrazek przedstawiający jednostkę, umieszczany w interfejsie użytkownika
- **UnitTroopersData** – opis jednostek piechoty występujących w grze. Znaczenie pól jest podobne jak w przypadku opisu czołgu
- **UnitHarvestersData** – opis żniwiarek występujących w grze. Znaczenia pól jest podobne jak w przypadku opisu czołgu, poza dodatkowymi polami:
 - Capacity – pojemność żniwiarki na zbieraną przyprawę
 - GatheringSpeed – prędkość zbierania przyprawy
- **UnitMCVsData** – opis jednostek MCV występujących w grze. Znaczenie pól jest podobne jak w przypadku czołgu, poza dodatkowym polem:
 - BuildingCanProduce – budynek, jaki może powstać przy pomocy MCV
- **UnitSandwormData** – opis czerwii występujących w grze. Znaczenie pól jest podobne jak w przypadku czołgu.
- **AmmosData** – opis typów amunicji występującej w grze. Pojedynczy typ opisywany jest przez następujące elementy:
 - Name – nazwa typu amunicji
 - Speed – prędkość przemieszczania się po strzale
 - DamageDestroyRange – zasięg wybuchu amunicji
 - Texture – ścieżka do pliku z teksturą amunicji
- **BuildingsData** – opis budynków występujących w grze. Pojedynczy budynek opisywany jest przez następujące elementy:
 - Name – nazwa budynku
 - Cost – koszt produkcji budynku
 - Health – wytrzymałość budynku
 - ViewRange – zakres widzenia budynku
 - BuildSpeed – szybkość budowania
 - Texture – ścieżka do pliku z teksturą budynku
 - Picture – obrazek przedstawiający budynek, umieszczany w interfejsie użytkownika
 - Size – rozmiary budynku wszerz i wzdłuż
 - EnergyConsumption – zużycie energii przez budynek
 - RideableFields – numery pól budynku, po których mogą poruszać się jednostki (numerowanie od lewego górnego rogu budynku, wierszami)
 - UnitsCanProduce – jednostki, jakie mogą być tworzone przez ten budynek (znajduje to odzwierciedlenie w interfejsie użytkownika)
 - BuildingsCanProduce – budynki, jakie mogą być tworzone przez ten budynek (znajduje to odzwierciedlenie w interfejsie użytkownika)



W przypadku budynków mogących strzelać (np. bunkry) można jeszcze ustawić następujące pola w pliku konfiguracyjnym:

- Power – siła strzału
- FireRange – zasięg strzału
- ReloadTime – czas przeładowywania
- RacesData – dane na temat każdej z ras występujących w grze:
 - Name – nazwa rasy
 - TechnologiesDependences – opis drzewa technologii (patrz 4.2), dla każdej z ras (spis budynków potrzebnych do wybudowania danego typu budynku)

Wszystkie dane dotyczące prędkości zostały podane w ilości tur gry, jakie zajmują tą czynność, natomiast jednostką odległości jest jedno pole planszy.

Schemat pliku XSD może w miarę potrzeb ulec niewielkim modyfikacjom, jeśli na etapie implementacji okaże się ważne przechowywanie pewnych danych dotyczących gry w osobnym pliku, jednak jego logika i układ nie będą przebudowywane.

Dane z pliku XML, są wczytane do klasy **GameSettings** (*static*), której statyczne właściwości odzwierciedlają strukturę pliku z ustawieniami. Dzięki temu dane te w każdej chwili gry (np. przy tworzeniu nowej jednostki) mogą zostać pobrane, bez potrzeby ponownego przeglądania drzewa XML.

Zależności zdefiniowane w pliku pomiędzy budynkami i jednostkami znajdują odzwierciedlenie w interfejsie użytkownika, tzn. jeśli jakiś budynek może budować tylko konkretne typy jednostek, tylko te typy są prezentowane użytkownikowi za pomocą obrazków, do których ścieżki również zdefiniowane są w pliku. Oznacza to oczywiście generowanie menu przy każdym uruchamianiu gry, na podstawie danych w pliku (patrz 3.3).

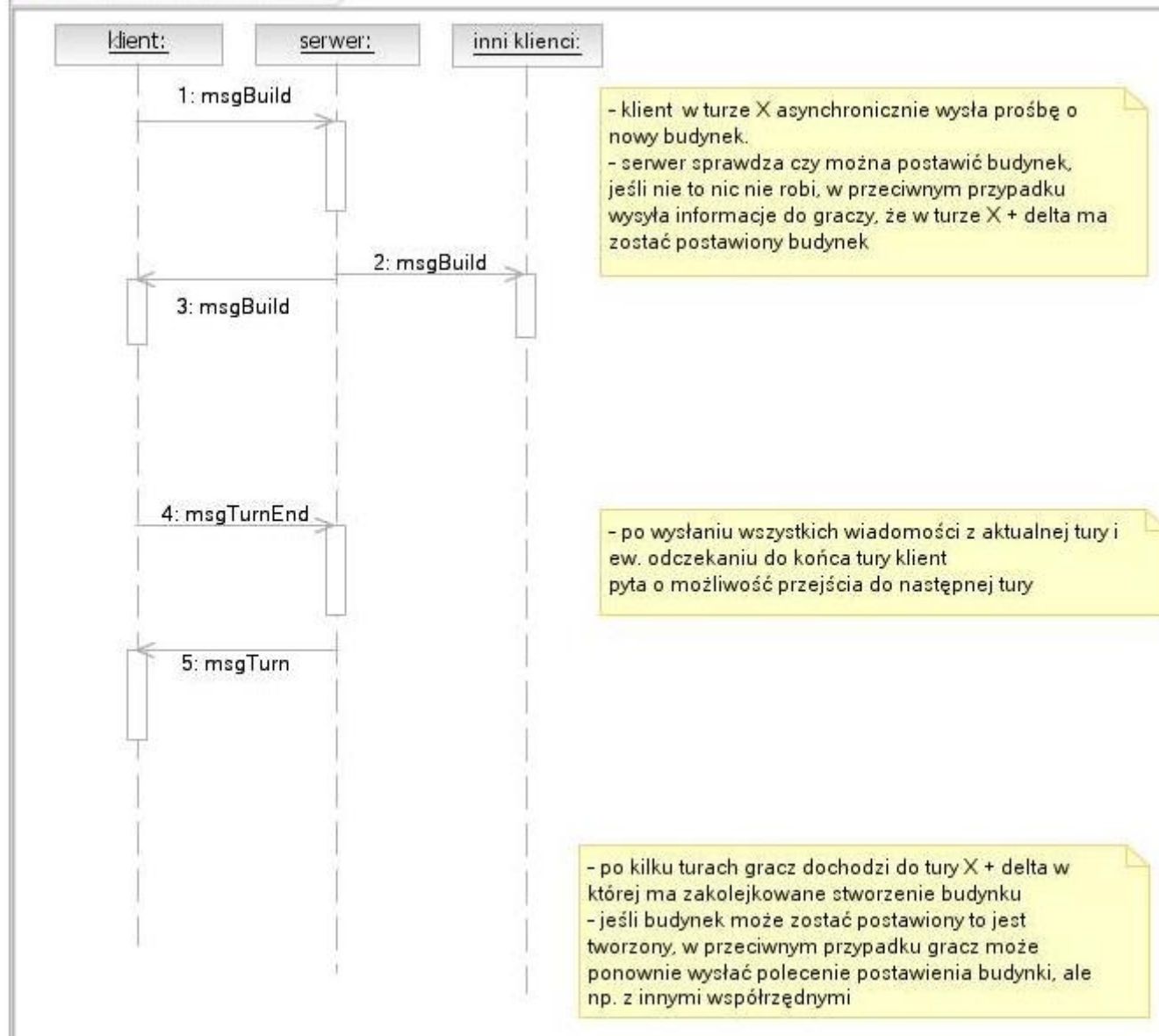
4.2. Logika budowania budynków

Tworzenie nowych budynków będzie polegało na kliknięciu w UI na ikonkę reprezentującą Construction Yard i wybranie nowego budynku z listy dostępnych budynków. Po tej akcji przez pewien czas (określony w turach), zdefiniowany w pliku konfiguracyjnym, lista z budynkami będzie nieaktywna, a nowowybrany budynek będzie w trakcie budowania. Wszystko to odbywać się będzie lokalnie po stronie klienta gry, bez udziału serwera. Po upływie czasu potrzebnego na zbudowanie budynku użytkownik będzie mógł ponownie kliknąć na dany budynek na liście i umieścić go na mapie. W tym momencie aplikacja klienta wygeneruje wiadomość proszącą serwer o postawienie budynku w danym miejscu na mapie. Serwer może odrzucić taką wiadomość, jeśli dane miejsce na mapie jest już zajęte (w takim wypadku gracz będzie mógł wybrać inne miejsce), bądź zezwolić na postawienie budynku, co zaowocuje wysłaniem przez serwer wiadomości tworzącej dany budynek do wszystkich graczy. Oczywiście, jeśli klient wygenerował prośbę postawienia budynku w turze X to budynek zostanie wybudowany dopiero w turze $X + \Delta$. Przy dużej Δ może wprowadzić to znaczne opóźnienie między postawieniem budynku (kliknięciem na mapie), a ujrzeniem budynku, niemniej jednak jest to niezbędny mechanizm zapobiegający rozsynchronizowaniu się gry. Ważną rzeczą jest to, iż klient w trakcie stawiania budynku w turze $X + \Delta$ również sprawdza czy aby na pewno możliwe jest postawienie budynku w danym miejscu. Dopiero, gdy budynek zostanie postawiony kolejka budowania zostanie zwolniona i gracz będzie mógł rozpocząć stawianie kolejnej budowli.

Schemat pokazujący transmisję wiadomości pomiędzy klientem, a serwerem:

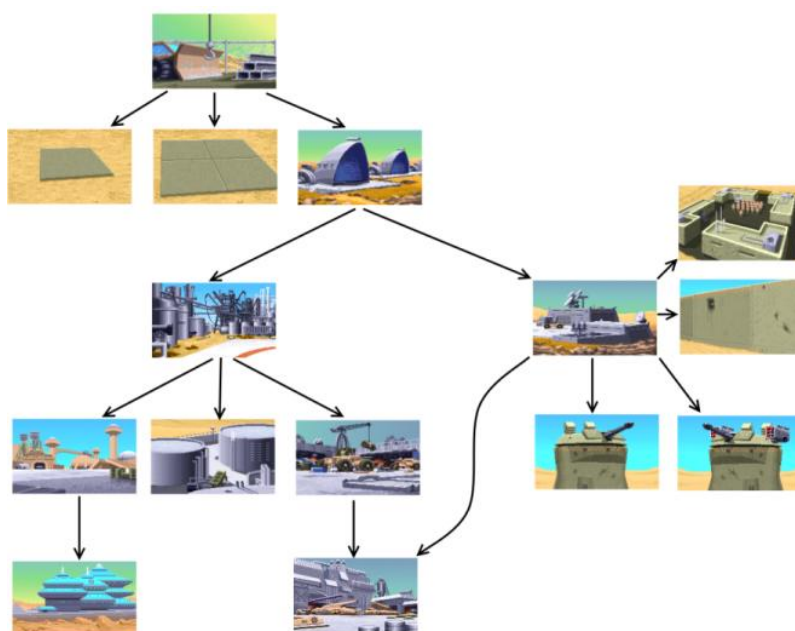


Komunikacja klient-serwer



Nie wszystkie budynki będą możliwe do wybudowania na samym początku gry. W grze, między budynkami wystąpią pewne zależności. Oznacza to, że niektóre budynki będą niedostępne do czasu wybudowania jakichś innych budynków. Dla każdego budynku będzie utrzymywana lista wymaganych budynków wraz z informacją ile danych budynków już istnieje. W momencie gdy któryś budynek na liście nie będzie występował przynajmniej w jednym egzemplarzu to docelowy budynek nie będzie mógł być wybudowany, np. zniknie z listy dostępnych budowli.






W naszym projekcie użyjemy zależności przedstawionych na poniższym wykresie. Należy pamiętać przy tym, że struktura pliku konfiguracyjnego umożliwi późniejsze modyfikowanie tych zależności.



4.3. Logika tworzenia jednostek

Jednostki będą budowane analogicznie do budynków (patrz punkt 4.2) z tą różnicą, że jednostki budowane są w budynkach: Barracs, Light Factory i Heavy Factory oraz, że położenie nowoutworzonych jednostek na mapie może zostać automatycznie wybrane przez aplikację.

Każdy budynek będzie miał zdefiniowaną listę możliwych do wytworzenia jednostek. Jednostki typu „Troopers” (piechota) będą tworzone w budynku „Barracs”, lekkie pojazdy (m.in. Trike, Quad, Raider Trike) będą produkowane w „Light Factory”. Pozostałe jednostki (ciężkie) wytwarzane będą w „Heavy Factory”. Jednostki w poszczególnych budynkach będą mogły być tworzone niezależnie, tj. będzie można na raz budować piechotę i czołg, ale już nie np. dwa Quad’y na raz.

Nazwa	Obrazek	Miejsce produkcji
Light Infantry		Barracs
Heavy Troopers		Barracs
Trike		Light Factory
Raider Trike		Light Factory
Quad		Light Factory



Combat Tank		Heavy Factory
Devastator		Heavy Factory
Siege Tank		Heavy Factory
Sonic Tank		Heavy Factory
Deviator		Heavy Factory
Rocket Launcher		Heavy Factory
MCV		Heavy Factory
Harvester		Heavy Factory

Należy przy tym pamiętać, że miejsce produkcji jednostek jest zdefiniowane w pliku konfiguracyjnym gry i w dowolnym momencie może zostać zmodyfikowane.

4.4. Logika ruchu jednostek

Do wyszukiwania drogi między obiektami wykorzystany zostanie klasyczny algorytm A*. Jeśli okaże się on nieefektywny – zostanie on rozbudowany o implementację algorytmu HPA*, który przedstawia się następująco:

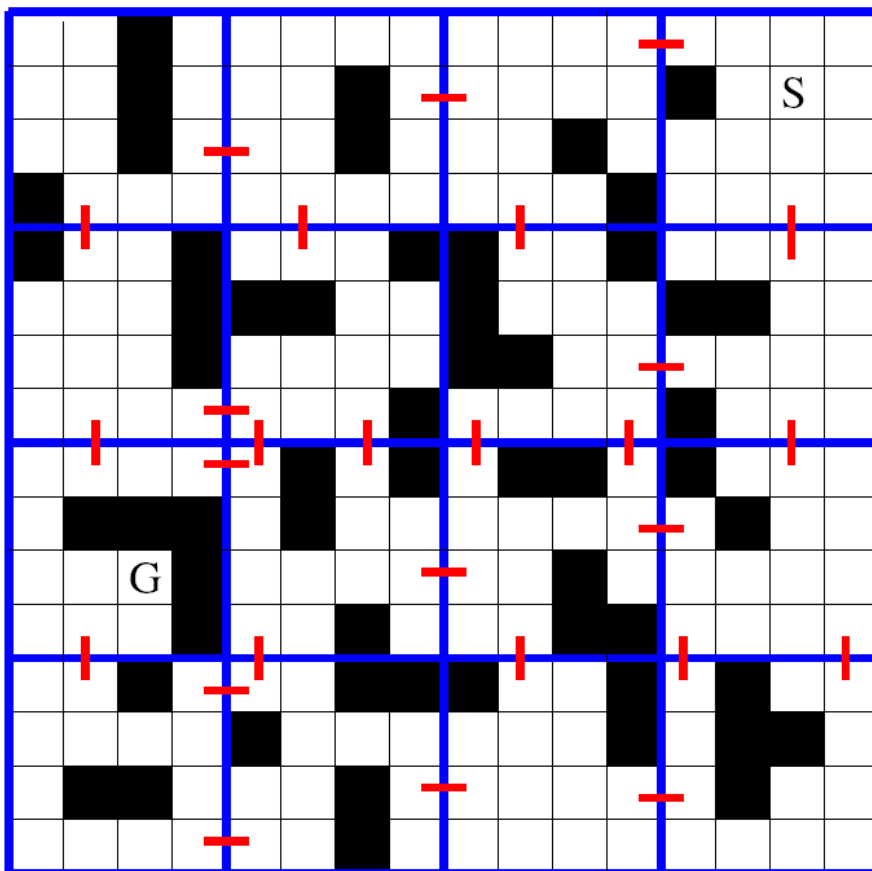
4.4.1. Logika ruchu jednostek (HPA*)

Naturalną, jednak nie banalną rzeczą w każdej grze typu RTS jest logika ruchu jednostek (żołnierzy, pojazdów...). Problem wydaje się nie istnieć, gdy mamy do czynienia z ruchem po prostej. W momencie jednak, gdy pojawią się przeszkody terenowe, przez które nie da się przejść/przejechać, pojawia się problem. Rozwiązanie należy zaczerpnąć z podstaw sztucznej inteligencji. Bardzo często wykorzystywanym algorytmem jest A*, jednak dla dużych map używanie tego algorytmu do poruszania się pomiędzy dwoma punktami z uwzględnieniem przeszkód, powoduje eksponencjalny wzrost złożoności.

Lekarstwem jest podzielenie planszy gry na podobszary i zastosowanie metody *divide&conquer*. Dla małych obszarów A* jest wystarczająco szybki. Algorytmem, który wykorzystuje taki podział

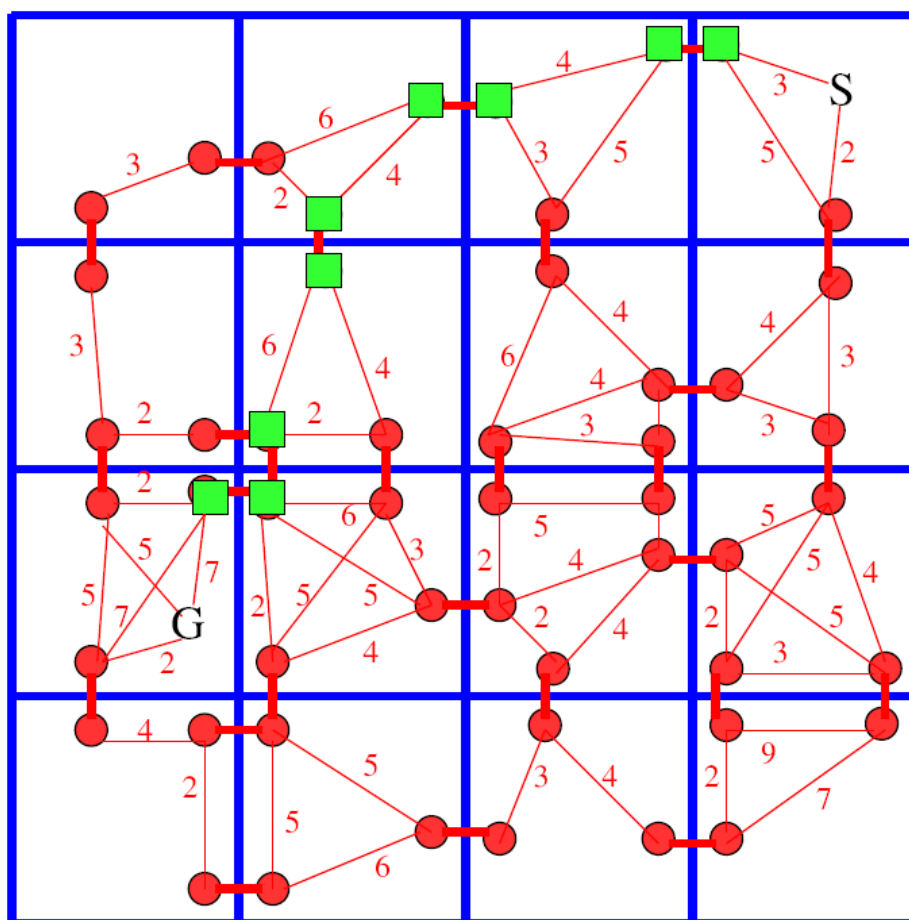


planszy jest *HPA** (*Hierarchical Pathfinding A**). Przykładowo mając mapę, jak poniżej, możemy ją podzielić na 16 obszarów.



Rysunek 14 Przykładowa mapa

Czarne pola to przeszkoda terenowa. Dodatkowo na krańcach przedziałów, w miejscach, gdzie są przeszkody (lub na środku w przeciwnym przypadku) wyznaczamy miejsca *wejścia*. Na planszy są one zaznaczone czerwonymi kreseczkami. Punkty te służą do zbudowania mapy wyższego poziomu. Między tymi punktami są liczone odległości, jak poniżej:



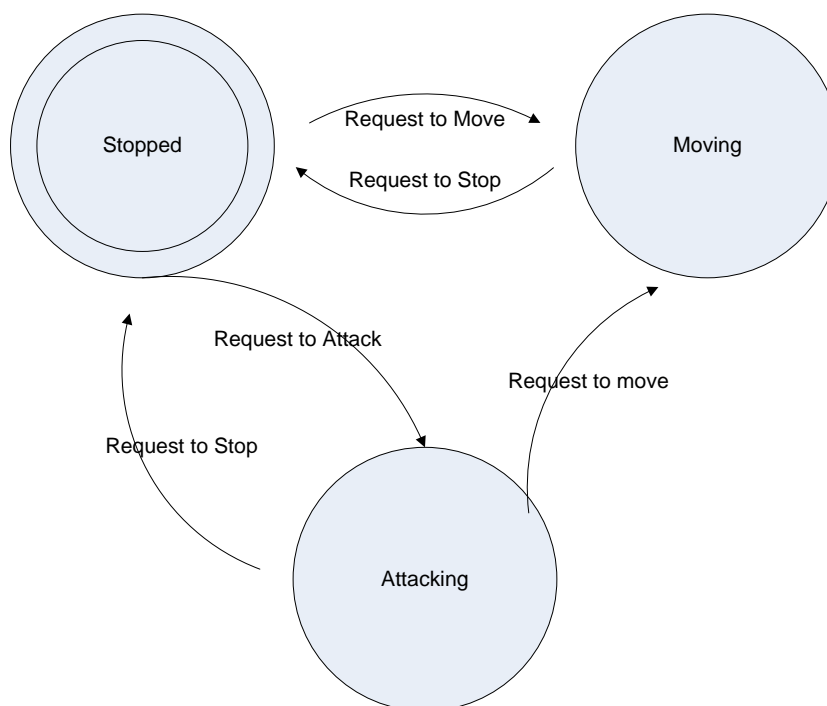
Rysunek 15 Zredukowana mapa

Zbudowanie ścieżki pomiędzy punktami S i G polega na szukaniu tylko wśród wierzchołków zredukowanych, co znacznie zmniejsza złożoność. Następnie mając listę takich wierzchołków, kierowana jednostka może między kolejnymi budować sobie lokalną ścieżkę i według niej się poruszać.

Dużą zaletą tego rozwiązania jest hierarchiczność. W konsekwencji poruszająca się jednostka nie musi od razu budować całej ścieżki, po której będzie się poruszała, a jedynie jej najbliższy fragment. Możliwe jest zatem rozłożenie obliczeń na kilka kolejnych klatek gry, nie powodując opóźnień.

4.5. Logika walki jednostek

Poniżej widoczna jest maszyna stanowa jednostki. Większość jednostek zawiera trzy podstawowe stany – stanie, poruszanie się i atakowanie. Te trzy stany się wykluczają, więc jeśli jednostka, która się porusza zauważy jednostkę wroga, to musi najpierw się zatrzymać, by następnie przejść do stanu atakowania.



Rysunek 16 Maszyna stanowa jednostki

Następujące czynności są wykonywane w odpowiednich stanach:

1. Stopped – jednostka w tym stanie czeka na rozkazy.
2. Moving – jednostka porusza się po ścieżce do celu.
3. Attacking – jednostka atakuje wyznaczoną jednostkę dokonując co określony interwał czasu strzału.

Następujące czynności powodują zmianę odpowiednich stanów:

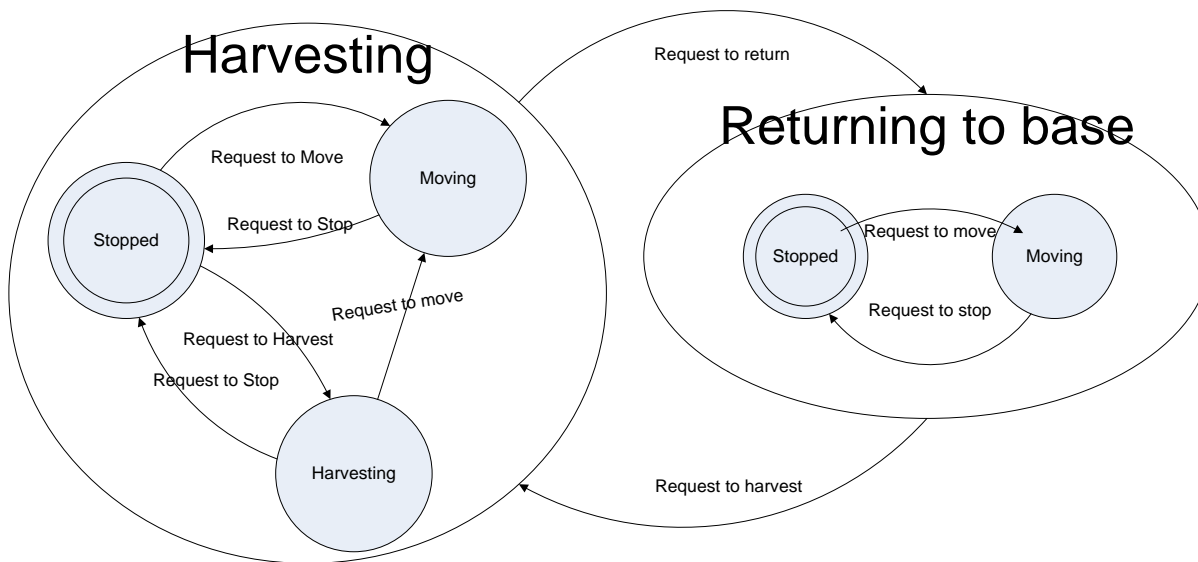
1. **Stopped**
 - a. Do stanu Moving
 - i. Rozkaz przesunięcia jednostki
 - ii. Jednostka wroga poza zasięgiem strzału
 - b. Do stanu Attacking
 - i. Najbliższa jednostka wroga w zasięgu strzału
 - ii. Rozkaz ataku jednostki będącej w zasięgu strzału
2. **Moving**
 - a. Do stanu Stopped
 - i. Jednostka w zasięgu strzału
 - ii. Jednostka dotarła do celu
3. **Attacking**
 - a. Do stanu Stopped
 - i. Jednostka zniszczona
 - b. Do stanu Moving
 - i. Rozkaz poruszania się
 - ii. Jednostka poza zasięgiem strzału

Jednostki strzelają różnymi typami amunicji. Wyróżniamy pociski(Bullet), rakiety(Rocket) oraz fale soniczną(Sonic). Różnią się one zachowaniem. Pocisków nie widać, rakiety zadają obrażenia wokół celu wybuchu, a fala soniczna zadaje obrażenia na całej długości od jednostki strzelającej do celu. W momencie zniszczenia jednostki, jednostka ta wybuchając zadaje obrażenia jednostkom naokoło.



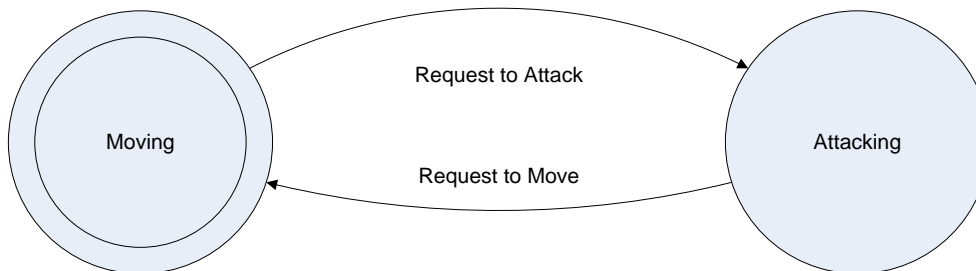
4.6. Logika jednostek specjalnych

Poniżej widoczna jest maszyna stanów żniwiarki. Żniwiarka różni się tym od innych jednostek, że jej zadaniem jest zbieranie przyprawy. Gdy zbierze odpowiednią ilość tej przyprawy, wraca do rafinerii i zostaje opróżniona. Maszyna stanów jest zatem bardziej skomplikowana.



Rysunek 17 Maszyna stanowa żniwiarki

Inną jednostką specjalną jest czerw. Porusza się on po mapie niezauważony poszukując dużych zgrupowań pojazdów. W chwili gdy znajdzie takie zgrupowanie, atakuje wyłaniając się z piasku i niszcząc jednostkę pod którą się wyłonił. Czerwom mogą się poruszać tylko po piasku.



Rysunek 18 Maszyna stanów Czerwia

4.7. Logika pracy zegara z naciskiem na animację elementów

Podstawową jednostką czasu w grze jest jedna tura gry. Przejście do następnej tury, kontrolowane jest przez serwer (patrz 5.1). Niezależnie od tur gry przebiega wyświetlanie stanu gry. Jest ono robione tak często jak pozwalają nam na to parametry komputera użytkownika.

W przypadku, kiedy jednostka jest w stanie *Stopped* (patrz 4.5) wyświetlana jest ona w bezruchu na planszy, jednak kiedy podejmie ona jakąś akcję (obróć, ruch, itp.) musi ona zostać odpowiednio animowana. Klasą odpowiedzialną za animacje jest **Animation**, zawiera ona pozycję animacji na mapie oraz aktualną klatkę animacji do wyświetlenia (numerując od 0).

Przy wyświetlaniu stanu rozgrywki aktualna klatka animacji do wyświetlenia wyliczana jest ze wzoru



5. $\text{Round}(\frac{(\text{frames} - 1) \cdot \text{turn}}{\text{speed} - 1})$, gdzie:

frames – całkowita liczba klatek w tej animacji

turn – aktualna tura dla tej animacji

speed – prędkość podejmowanej akcji (liczba tur w jakiej powinna zostać wykonana)







Jednocześnie animacja zmienia swoje położenia o

$\frac{\text{tile}}{\text{speed}}$, gdzie

tile – rozmiar w pikselach pojedynczego segmentu mapy

speed – prędkość podejmowanej akcji (liczba tur w jakiej powinna zostać wykonana)

Prześledźmy zaproponowany algorytm na podstawie jednostki piechoty. Załóżmy, że posiada ona 2 klatki animacji na ruch i jedną na spoczynek oraz, że porusza się z prędkością 4 tur na ruch. Akcja ruchu będzie przedstawiała się, zatem następująco:

Rysunek	Tura ruchu (numerowana od 0)	Opis
	Przed ruchem	Jednostka w spoczynku
	0	Zerowa klatka animacji, jednostka przesunięta o 0.25 segmentu mapy
	1	Zerowa klatka animacji, jednostka przesunięta o 0.5 segmentu mapy
	2	Pierwsza klatka animacji, jednostka przesunięta o 0.75 segmentu mapy
	3	Pierwsza klatka animacji, jednostka przesunięta o 1.0 segmentu mapy
	Po ruchu	Jednostka po ruchu

5.1. Logika dźwięków

Bardzo ważnym elementem każdej gry jest oprawa dźwiękowa. W „Yet Another Dune 2” rozróżniamy dwa rodzaje efektów dźwiękowych:

1) Muzyka

Towarzyszyć będzie graczowi od początku gry. Można wyróżnić kilka typów ścieżek muzycznych:

- odtwarzane w trakcie pokoju – gdy nie ma walczących jednostek
- odtwarzane w trakcie walki – gdy na mapie są walczące jednostki
- odtwarzane po wygranej grze
- odtwarzane po przegranej grze

Muzyka będzie odtwarzana cały czas i będzie się zmieniać dynamicznie w zależności od zmieniających się warunków na polu bitwy.

2) Efekty dźwiękowe

Towarzyszą takim typom zdarzeń jak:

- zniszczenie budynku
- zaznaczanie i wydawanie rozkazów jednostkom
- strzały jednostek
- stawianie nowych budynków

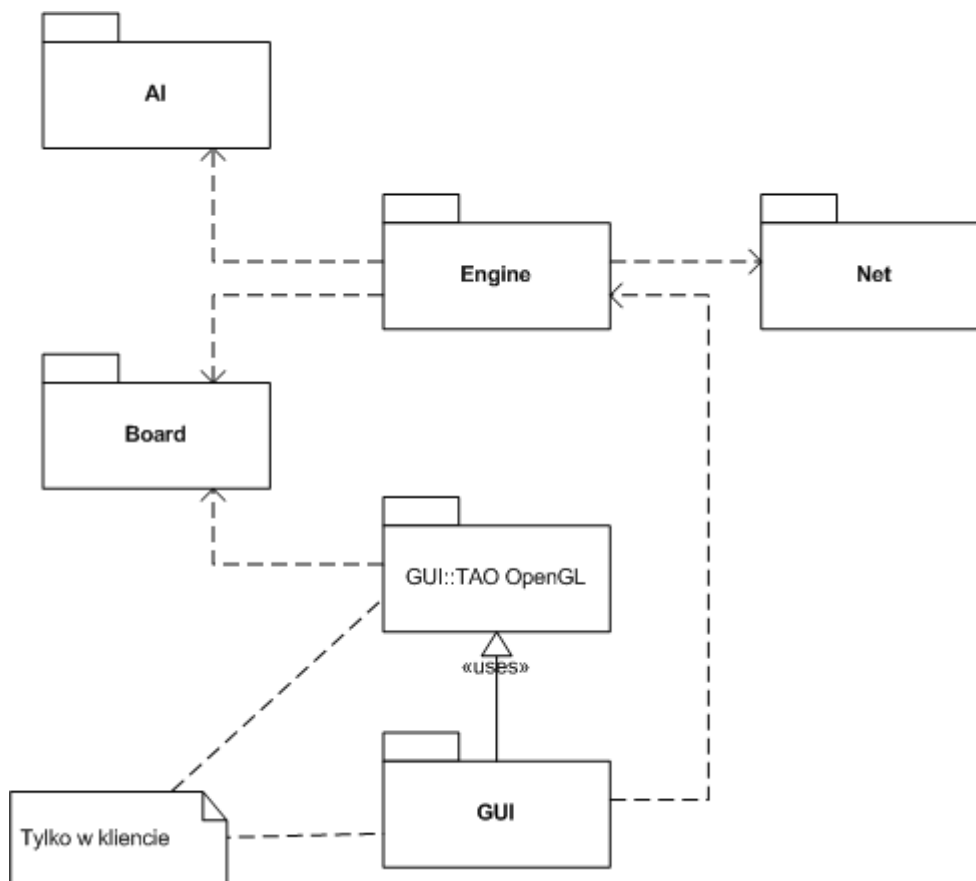


- śmierć żołnierzy

Efekty dźwiękowe będą odgrywane jednorazowo podczas każdego wystąpienia zdarzenia dla którego przypisany jest efekt dźwiękowy.

Dla każdego typu zdarzeń dźwiękowych oraz dla każdego typu ścieżki muzycznej będzie można zdefiniować kilka plików dźwiękowych, które będą odtwarzane zamiennie w sposób losowy.

5.2. Schemat modułów



Rysunek 19 Schemat modułów

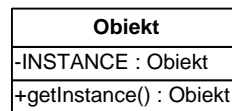
Sercem całego programu jest moduł **Engine**, który odbiera wiadomości z sieci (moduł **Net**) oraz z **GUI**, a następnie jest odpowiedzialny za ich przetwarzanie – poprzez m.in. modyfikację struktur danych udostępnianych przez interfejsy planszy (Board). Może przy tym korzystać z modułu **AI**, do wyznaczania np. tras ruchu jednostek. Za rysowanie planszy odpowiedzialny jest moduł **GUI::TAO OpenGL** będący częścią modułu **GUI**. Przy odrysowywaniu sytuacji na planszy korzysta on oczywiście z modułu **Board**.

5.3. Schemat klas z opisem

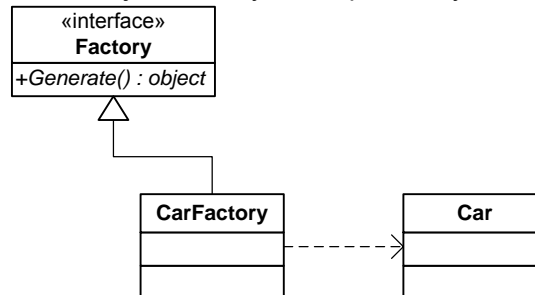
W załączeniu do dokumentacji dołączony jest schemat bazowych klas dla projektu.

W projekcie będzie można zauważyć następujące wzorce projektowe:

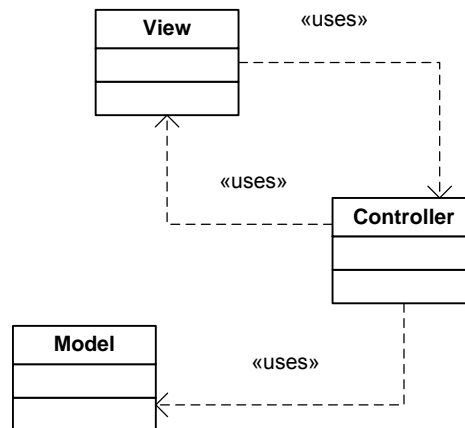
- Singleton – podstawowy wzorec projektowy służący do przypilnowania, że tylko jedna instancja danej klasy powstanie w aplikacji.



- Factory – wzorec projektowy oddelegowujący tworzenie obiektu fabryce – przydatne, gdy proces tworzenia obiektu jest złożony i skomplikowany



- MVC – model, view, controller – wzorec oddzielający warstwę interfejsu od logiki aplikacji. Pozwala to na zachowanie logiki aplikacji niezależnie od tego jak wygląda interfejs



Część klas dotycząca komunikacji sieciowej oraz strony serwera jest opisana dalej, natomiast Następujące klasy wymagają dodatkowego opisu:

- **XMLLoader**
 - Klasa obsługująca ładowanie ustawień gry i zapisywanie ich do GameSettings.
- **ErrorLog**
 - Jest to dziennik zdarzeń aplikacji.
 - Służy do zapisywania komunikatów i raportów o błędach do pliku.
- **IRenderer**
 - Interfejs do renderowania obiektów.
- **BoardObject**
 - Reprezentuje podstawowy obiekt umieszczany na planszy.
 - W najprostszym przypadku może być to animacja rakiety bądź wybuchu.
- **Animation**
 - klasa opakowująca grafikę.
 - Umożliwia zmianę klatki w zależności od czasu.
- **Building**
 - Klasa reprezentująca budynek.
 - Budynki mają możliwość pozwolenia na wybudowanie innych budynków
 - Budynki mają możliwość pozwolenia na wybudowanie jednostek
- **Unit**
 - Klasa abstrakcyjna reprezentująca jednostkę. Jest to część bazowa dla wszystkich jednostek.
- **UnitTank**



- Klasa reprezentująca czołgi.
 - Czołg ma wierzyczkę, która się obraca niezależnie od korpusu
- **UnitTrooper**
 - Klasa reprezentująca jednostki biegające.
- **UnitHarvester**
 - Klasa reprezentująca żniwiarkę.
 - Obiekty tej klasy zachowują się jak żniwiarka tj szukają przyprawy.
- **UnitMCV**
 - Klasa reprezentuje MCV.
 - Jednostki MCV transformują się w odpowiedni budynek
- **UnitSandWorm**
 - Klasa reprezentuje czerwie.
 - Czerwie szukają zgrupowań pojazdów.
- **Map**
 - Klasa zawierająca informacje na temat całej planszy na której jest toczona rozgrywka. Jest na niej informacja o wszystkich polach oraz o wszystkich jednostkach w grze.
- **Simulation**
 - Mechanizm obsługi wiadomości dotyczących samej gry. Posiada kolejke wiadomości.
- **IONGameMessage**
 - Interfejs do obsługi wiadomości. Interfejs ten jest podawany instancji symulacji.
 - Służy do tego, aby można było w różny sposób obsługiwać te same wiadomości po stronie serwera jak i klienta.

6. Implementacja gry sieciowej

6.1. Opis realizacji gry sieciowej

6.1.1. Serwer

Do realizacji funkcjonalności serwera zrealizowany zostanie osobny program „Serwer YAD 2”. Do jego zadań należeć będzie:

- Obsługa połączeń użytkowników:
 - Logowanie
 - Rejestracja
 - Prowadzenie chatu
 - Zakładanie gier
 - Dołączanie do gier
 - Dobieranie użytkowników do gier
- Obsługa gier:
 - Prowadzenie rozgrywek między graczami
 - Obsługa utraty połączenia
 - Obsługa zakończenia gry

Podstawowe przetwarzanie serwera przedstawione jest na schemacie.

Dane użytkowników przechowywane będą w bazie danych w następującej postaci:

Kolumna	Typ Danych	Opis
ID	Integer	Klucz główny
LOGIN	Varchar	Login użytkownika
PASSWORD	Varchar	Hasło
E-MAIL	Varchar	E-mail użytkownika



WIN_NO	Integer	Liczba zwycięstw
LOST_NO	Integer	Liczba porażek

Przy logowaniu użytkownika jego dane są ładowane do pamięci i przechowywane do czasu utraty połączenia – przechowywane dane to: id, login, win_no, lost_no, ponits. Gdy użytkownik się rozłączy, następuje uaktualnienie rekordu użytkownika. Gdy użytkownik nie jest zalogowany, może wysłać do serwera następujące wiadomości:

Nazwa	Dane	Opis
msgLogin	Login i Hasło	Wiadomość służąca do logowania
msgRegister	Login i hasło	Wiadomość służąca do rejestracji
msgRemind	Login	Wiadomość służąca do przypomnienia loginu

Odpowiedzi serwera:

Nazwa	Dane	Opis
msgLoginSuccessful	Brak	Logowanie zakończone pomyślnie
msgLoginUnsuccessful	Tekst: przyczyna	Logowanie zakończone niepomyślnie
msgRegisterSuccessful	Brak	Rejestracja zakończona pomyślnie
msgRegisterUnsuccessful	Tekst: przyczyna	Rejestracja zakończona niepomyślnie
msgRemindSuccessful	Brak	Przypomnienie zostało zrealizowane
msgRemindUnsuccessful	Tekst: przyczyna	Przypomnienie zostało niezrealizowane

Gdy użytkownik jest zalogowany, może wysłać do serwera wiadomości. Typy wiadomości:

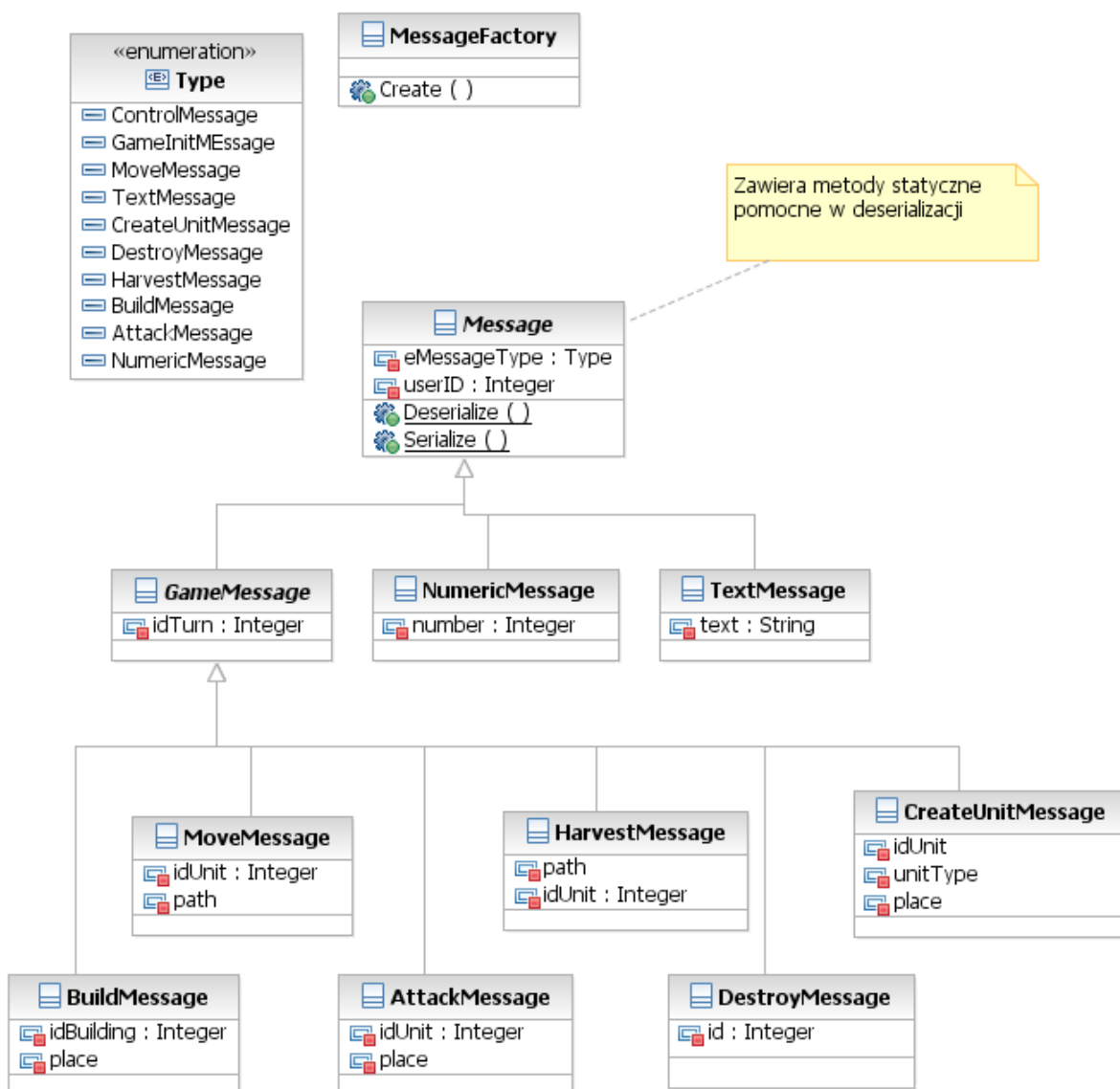
Nazwa	Dane	Opis
msgChatEntry	Brak	Informacja o wejściu na czat
msgChatExit	Brak	Informacje o wyjściu z czatu
msgChatText	Treść wiadomości	Treść wiadomości
msgChooseGameEntry	Brak	Użytkownik wchodzi w tryb dołączania do gry
msgJoinGameEntry	Nazwa gry	Prośba o dołączenie do gry
msgJoinGameExit	Brak	Informacja o opuszczeniu przez użytkownika
msgLogout	Brak	Wiadomość rozłączenia z serwerem. Po jej otrzymaniu można zakończyć wątek klienta i zaktualizować bazę danych
msgGameCreate	Rodzaj mapy, nazwa gry, czy gra prywatna \ publiczna,, ilość graczy	Informacja o założeniu gry przez użytkownika

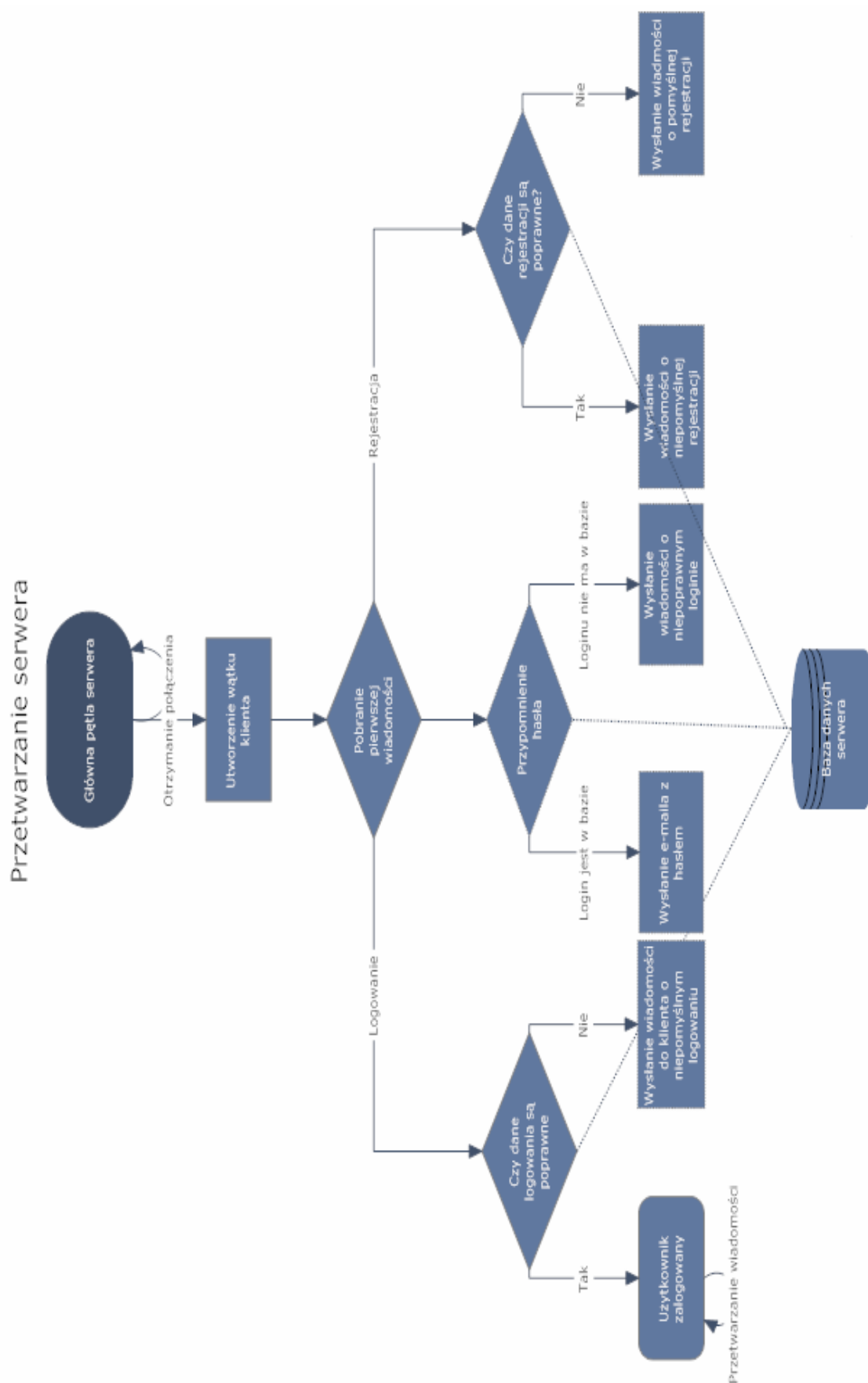
Odpowiedzi serwera

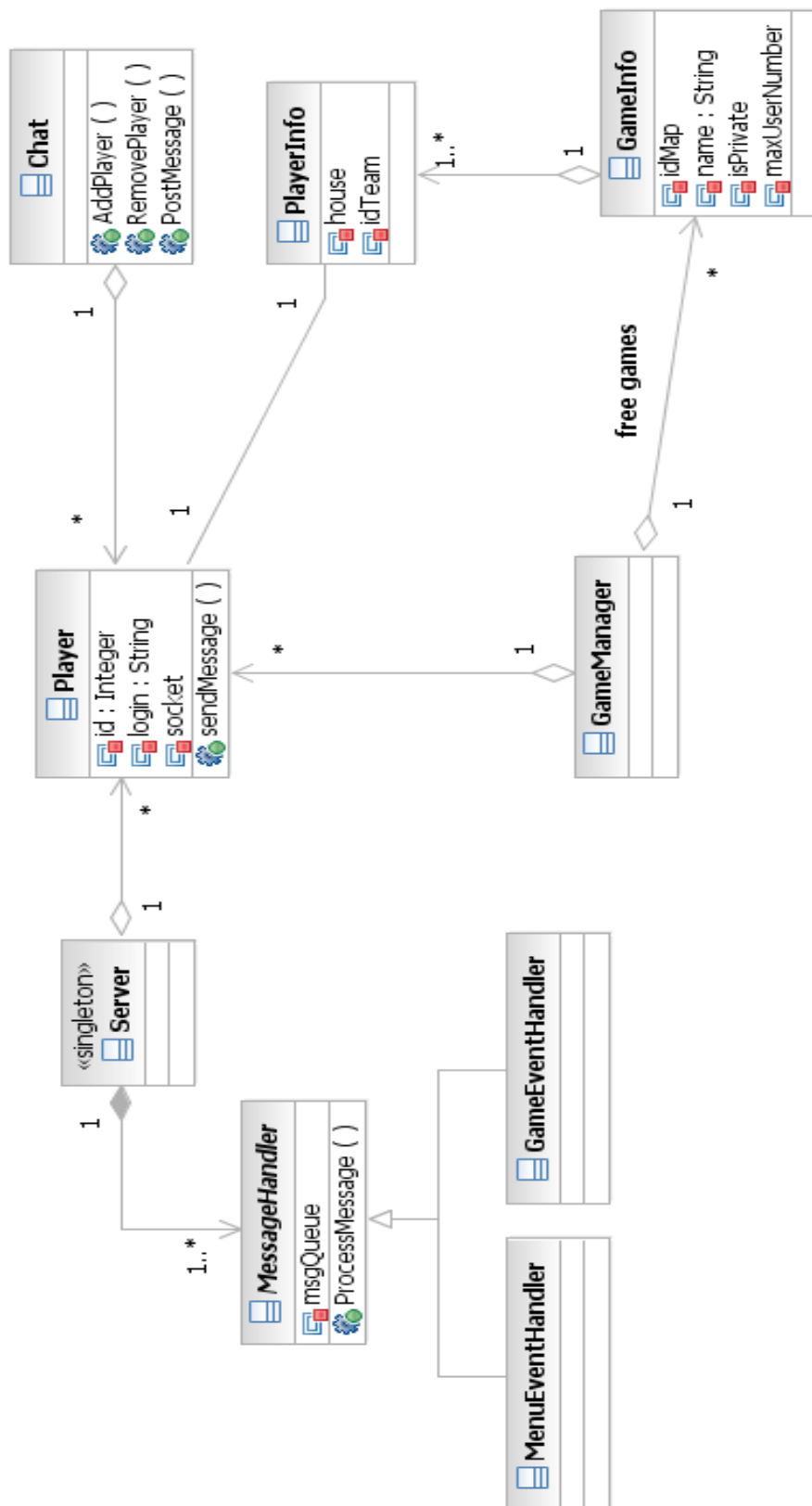
Nazwa	Dane	Opis
msgAddChatUser	Dane użytkownika: id, login,	Wysłanie danych potrzebnych



	poziom	na dodanie użytkownika do czatu
msgDeleteChatUser	Id użytkownika	Wysłanie danych potrzebnych do usunięcia użytkownika z czatu
msgNewChatText	Login użytkownika, tekst	Wysłanie informacji tekstowej czatu
msgNewGame	Nazwa gry, typ, id mapy, ilość graczy	Dodanie gry do listy gier
msgGameParams	Typ gry, id mapy, ilość graczy	Wysłanie informacji o grze, gdy jest ona prywatna
msgDeleteGame	Usunięcie gry z listy gier	Usunięcie gry z listy gier
msgJoinGameNewPlayer	Dane użytkownika który dołączył do gry: login, drużyna, rasa	Informacja o zmianie danych użytkownika \ dołączeniu użytkownika do gry
msgJoinGameDeletePlayer	Login użytkownika, który opuścił grę	Wiadomość przesyłana do graczy którzy dołączyli do gry
msgJoinGameUnsuccessful	Tekst: przyczyna	Dołączenie do gry niemożliwe
msgJoinGameSuccessful	Brak	Dołączenie do gry możliwe
msgStartGameUnsuccessful	Tekst: przyczyna	Rozpoczęcie gry niemożliwe
msgStartGameSuccessful	Dane wszystkich graczy w grze	Dołączenie do gry pomyślne
msgPlayerInfo	Dane: informacja o pojedynczym graczu – login, liczba zwycięstw \ liczba porażek	Służy poinformowaniu użytkownika o statystykach gracza









6.1.2. Struktury danych wykorzystywane w aplikacji serwerowej

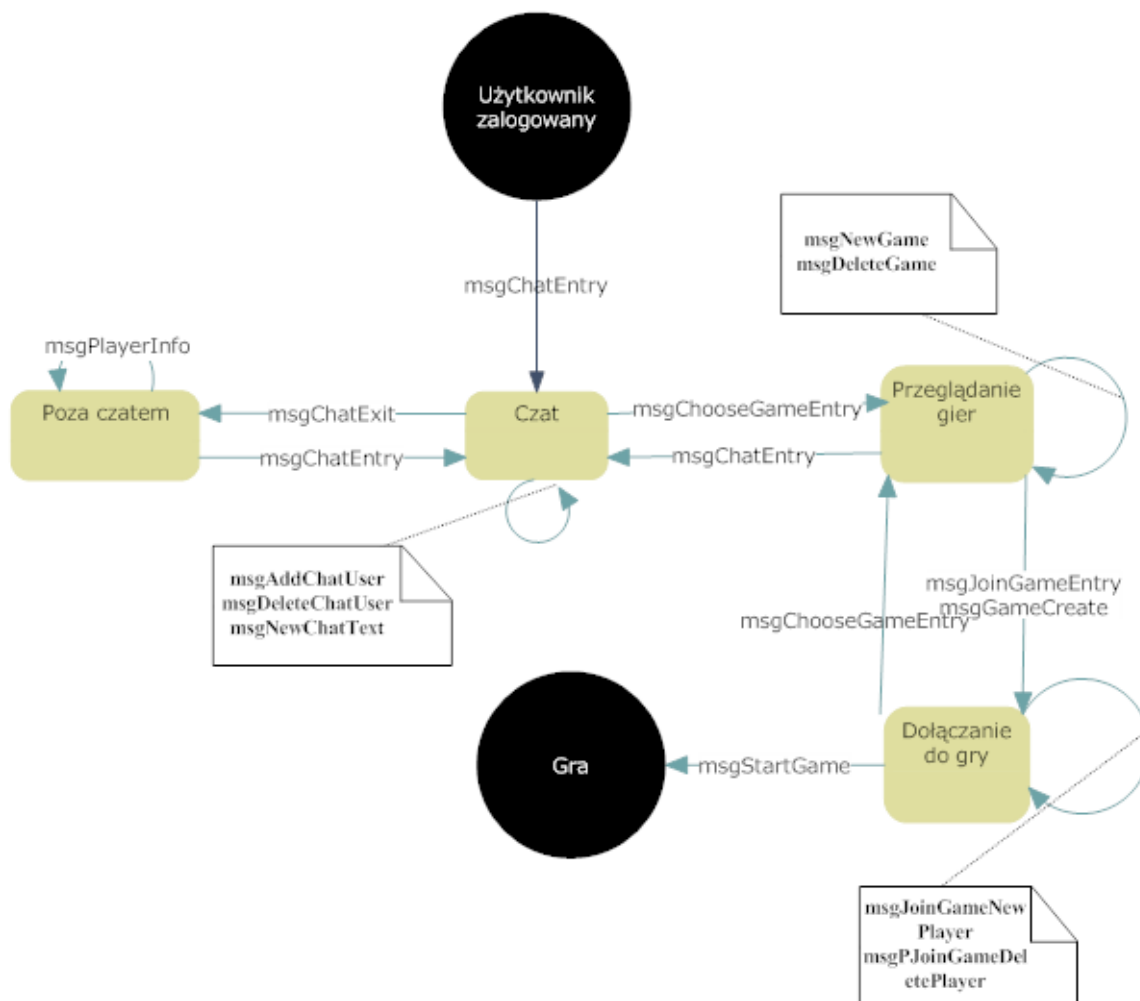
- Klasa **Server**:
 - Jest to główna klasa zarządzająca. Zawiera ona dane o wszystkich zalogowanych klientach, przyjmuje połączenia i przetwarza wiadomości
- Klasa **Chat**:
 - Odpowiada za wymianę komunikatów między użytkownikami
- Klasa **Player**:
 - Zawiera wszystkie dane o zalogowanym graczu
 - Odpowiada za odbiór wiadomości – zawierać będzie klasę MessageReceiver – podobną jak u klienta, która w będzie przekierowywać wiadomości do kolejki wiadomości.
- Klasa **MessageHandler**:
 - Klasa bazowa odpowiada za przetwarzanie i wysyłanie wiadomości. W ramach klasy pracować będą dwa wątki: pierwszy będzie odpowiadał za przetwarzanie wiadomości, drugi za wysyłanie ich do odpowiedniego klienta
- Klasa **MenuMessageHandler**:
 - Klasa przetwarza wiadomości menu – istnieje dokładnie jedna taka klasa w serwerze
- Klasa **GameMessageHandler**:
 - Klasa przetwarza wiadomości gry – w danym momencie jest tyle takich klas, ile jest prowadzonych rozgrywek
- Klasa **GameManager**:
 - Zarządca gier: odpowiada przechowywanie gier w stanie nierozpoczętym, dobieranie graczy, rozpoczynanie rozgrywek
- Klasa **GameInfo**:
 - Przechowuje informacje o grze dowolnej

6.1.3. Rozpoczęcie rozgrywki

Aby rozpocząć rozgrywkę, gracz musi przejść do ekranu rozgrywki. Tam będzie mógł wybrać grę z puli gier publicznych, dołączyć do gry prywatnej jeśli zna jej nazwę, lub założyć własną. Po wybraniu jakiegokolwiek z tych opcji zostanie przeniesiony do ekranu oczekiwania na graczy. Gdy dołączy do niego wymagana przez grę ilość, każdy z nich wybierze odpowiednią rasę i drużynę, oraz wciśnie przycisk „Start Game”, gra się rozpocznie. Jakiegokolwiek zmiany wprowadzane przez innego z graczy (zmiana parametrów gry) spowodują, że wciśnięcie przycisku „Start Game zostanie anulowane.

Rozpoczęcie gry polega na utworzeniu obiektu GameMessageHandler i przekierowaniu wiadomości graczy do jego kolejki. Od tego momentu zaczyna się etap gry zwany rozgrywką sieciową. Tworzony zostaje obiekt ServerGame i dodany do klasy GameManager. Opis klas:

- Klasa **ServerGame**
Jest to klasa zarządzająca rozgrywką gry
- Klasa **GameMessageHandler**
Wszystkie wiadomości dotyczące gry z wątków klienta trafiają do kolejki znajdującej się w GameMessageHandler. Funkcja Process w tej klasie odpowiedzialna jest za przetwarzanie kolejki – np. dla żądania ruchu oblicza odpowiednią ścieżkę, oblicza w której turze powinno ruch wystąpić, następnie zleca wysłanie wiadomości w MessageSender i dodaje wiadomość do swojej klasy ServerSimulation
- Klasa **MessageSender**
Klasa pracuje w osobnym wątku – wysyła wiadomości do poszczególnych graczy
- Klasa **ServerSimulation** – pracuje w osobny wątku i odpowiada za przetwarzanie symulacji gry, zajmuje się również rozstrzygnięciem konfliktów

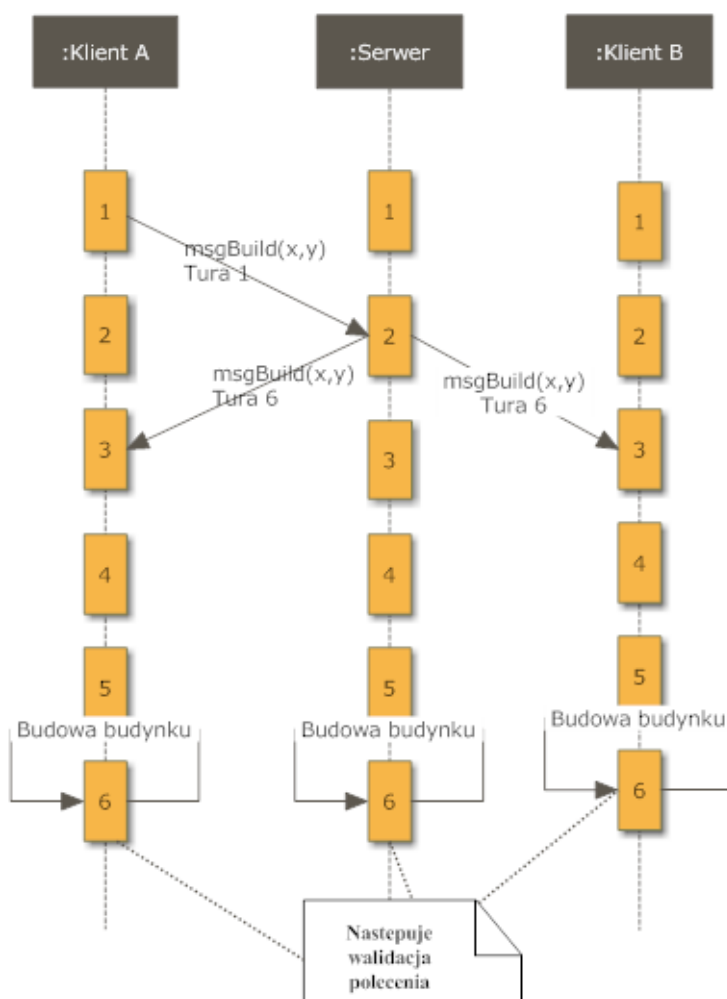




6.1.4. Rozgrywka sieciowa

Rozgrywka sieciowa w grze oparta będzie na technice blokady tur. Gra przebiega w jednostkach zwanych turami – jest to okres w którym na każdym obiekcie gry wywoływana jest operacja `.doTurn()`. W normalnych warunkach тура trwa ściśle określony czas. Dla wszystkich graczy biorących udział w rozgrywce gra zaczyna się w tym samym stanie. Gracze wykonują czynności w rozgrywce – budują budowle, przesuwiają jednostki. O wszystkich czynnościach, które mogą powodować konflikty, informowany jest serwer: przesyłane są odpowiednie dane wraz z numerem tury, w jakiej zostały wykonane. Na programie serwera również toczona jest rozgrywka w systemie turowym – program aktualizuje swoje dane na podstawie danych klientów. Jeśli otrzyma informacje o tym, że pewna akcja powinna być wykonana w turze X , przetwarza ją, a następnie wysyła pozostałym graczom z poleceniem wykonania w turze $X + \Delta$. Oprócz zwykłych wiadomości, serwer otrzymuje od klientów wiadomości przejścia do następnej tury. Dzięki temu serwer może kontrolować postępy każdego z graczy i uniemożliwić sytuację, gdy jeden z graczy wyprzedza drugiego o Δ tur. Gdy któryś z graczy osiągnie przewagę $\Delta - 1$ tur, jego gra zostanie wstrzymana do czasu przetworzenia kolejnych tur przez najwolniejszego z graczy. Gdy serwer wykryje sytuację, że pewien z graczy przetwarza grę wolniej od innych (np. ze względu na chwilowe obciążenie łącza), może nakazać przyspieszenie rozgrywki – wtedy tury tego gracza będą rozgrywać się jedna po drugiej, bez uwzględnienia przedziału czasowego.

Takie rozwiązanie powoduje, że za przetwarzanie części elementów odpowiada serwer. Musi on także informować użytkowników o pewnych sytuacjach, które nastąpiły bez ich bezpośredniej interwencji – np. o zniszczeniu jednostki. Serwer ma za zadanie prowadzić własną symulację, aby móc skutecznie zarządzać grą sieciową. Proces komunikacji z serwerem dla prostej komendy budowy został pokazany na schemacie. Ponieważ gry klientów nie znajdują się na tym samym etapie rozgrywki, lecz różnym od siebie maksymalnie o Δ tur, mogą pojawić się konflikty. Jeśli pewien gracz X znajdujący się w turze 5 wykona polecenie budowy budynku w punkcie (A,B), serwer roześle do wszystkich graczy polecenie wybudowy budynku w miejscu (A,B) w turze $\Delta + 5$, sam również w swojej grze ustanowi wykonanie tego polecenia w obliczonej turze. Problem pojawi się, jeśli gracz Y , który jest w turze 3 wyśle zaraz potem informacje o budowie budynku w tym samym miejscu (A,B). Wtedy zgodnie z logiką gry, serwer wyśle pozwolenie na budowę budynku w miejscu (A,B) w turze $\Delta + 3$, gdyż sam nie przetworzył jeszcze informacji od gracza X . Aby uniknąć sytuacji, że gracze zbudują budynki w turach $\Delta + 3$ i $\Delta + 5$ wprowadzony został mechanizm weryfikacji rozkazów. Każdy z uczestników rozgrywki oraz serwer musi przed wykonaniem rozkazu dokonać ich weryfikacji – jeśli jest ona niepomysłna – rozkaz jest anulowany. W przypadku symulacji serwera, może on wysłać zaktualizowaną wiadomość do użytkownika, która rozwiązuje konfliktową sytuację.



6.1.5. Wiadomości rozgrywki

6.1.5.1. Wiadomości wysyłane przez gracza

Nazwa wiadomości	Dane	Opis wiadomości
msgGameStartReady	brak	Wiadomość zostaje wysłana gdy gracz załadował wszystkie dane do pamięci i jest gotowy na rozpoczęcie rozgrywki
msgTurnEnd	numer tury	Wiadomość zostaje wysłana po zakończeniu tury użytkownika – informuje serwer o stanie rozgrywki oraz jest formą zapytania o przejście do następnej
msgPause	brak	Wiadomość o dokonaniu pauzy przez gracza
msgResume	brak	Wiadomość o przerwaniu pauzy



6.1.5.2. Wiadomości wysyłane przez serwer

Nazwa wiadomości	Dane	Opis wiadomości
msgGameInit	Początkowe ustawienia jednostek MCV (budujących bazę)	Wiadomość zostaje wysłana zaraz po rozpoczęciu gry
msgGameStart	brak	Początek rozgrywki
msgTurn	Informacja czy należy przyspieszyć przetwarzanie następnych Δ tur	Wiadomość wysyłana jako odpowiedź na wiadomość msgTurnEnd
msgGameEnd	Brak	Wiadomość jest wysyłana przy zakończeniu gry

6.1.6. Wiadomości gry

Każda z nich musi zawierać informacje o turze – wiadomości wysyłane do klienta zawierają turę, w której wykonany będzie rozkaz oraz identyfikator gracza który dokonał akcji, wiadomości wysyłane do serwera – aktualną turę gracza i identyfikator gracza, który dokonał akcji:

Nazwa wiadomości	Dane	Opis wiadomości
msgMove	Identyfikator jednostki, punkt docelowy	Wiadomość zostaje wysłana, gdy gracz chce wykonać ruch na planszy
msgBuild	Identyfikator budowli, typ budowli, miejsce	Wiadomość wysyłana, gdy gracz wykonał komendę budowy jednostki
msgNewUnit	Identyfikator jednostki, miejsce, w którym się ona pokazała	Wiadomość zostaje wysłana, gdy gracz stworzył nową jednostkę
msgDestroy	Identyfikator jednostki	Wiadomość została wysłana gdy jednostka została zniszczona
msgHarvest	Identyfikator jednostki, miejsce zbierania przyprawy	Wiadomość zostanie wysłana, gdy gracz dokonuje operacji zbierania przyprawy

6.1.7. Obsługa zakończenia tury i pauzy

Klienci informują serwer o zakończeniu tury – jeśli numer tury, do której chcieli przejść nie odbiega od najmniejszego numeru tury wśród uczestników gry o Δ , wysyłana zostaje odpowiedź informująca o możliwości przejścia do następnej. Wiadomość tego typu zawiera również informację o ewentualnej pauzie – jeśli użytkownik dowie się o pauzie, zatrzymuje przetwarzanie, podobnie jak w przypadku braku pozwolenie na przejście do kolejnej tury

6.1.8. Przetwarzanie wiadomości

Diagram wiadomości przedstawia podstawowe klasy, pokazanie wszystkich pod obiektów zajęłoby zbyt dużo miejsca i niepotrzebnie zaciemniłoby obraz. Wiadomości przetwarzane są po stronie serwera w sposób następujący:

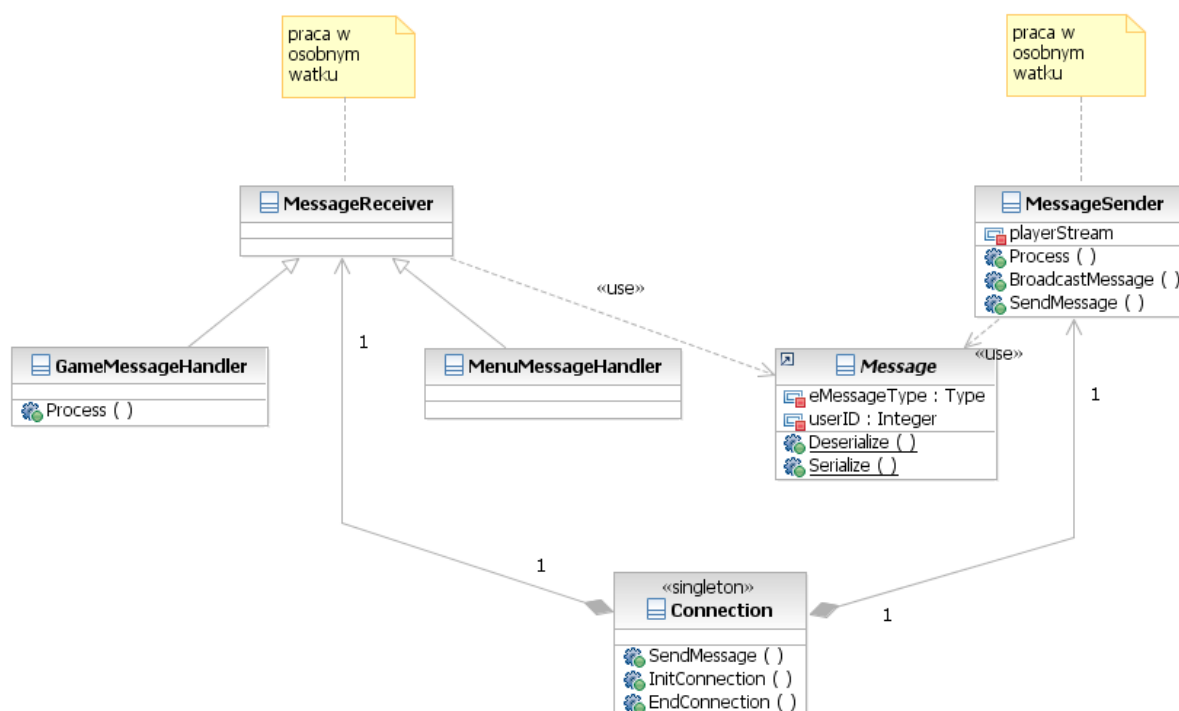


- W wątku nasłuchu klienta sprawdzany jest typ wiadomości w strumieniu – na jego podstawie klasa MessageFactory tworzy odpowiedni obiekt wiadomości
- Obiekt wiadomości wczytuje swoje dane ze strumienia
- Obiekt wiadomości dołączany jest do kolejki odbioru
- Kolejka odbioru przetwarza wiadomość – dla GameMessage wywoływana jest metoda Process, która przetwarza klasę zmieniając ją do postaci nadającej się do wysłania do klientów i uruchomienia przez serwer – dla klasy ControlMessage – wywoływana jest metoda CreateResponse, która zwróci nową wiadomość, która jest umieszczana w kolejce wiadomości do wysłania.

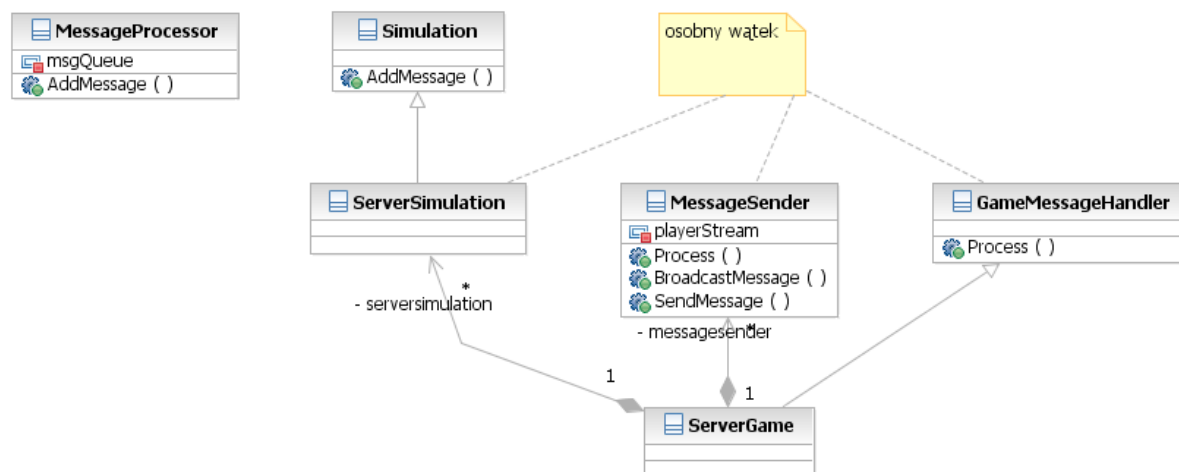
6.1.9. Przetwarzanie komunikacji w silniku gry

Klient będzie posiadał dwa wątki używane do komunikacji z serwerem. Jeden z wątków będzie odczytywał komunikaty z kolejki komunikatów wyjściowych i przysyłał je strumieniem danych do serwera przy użyciu protokołu TCP/IP. Drugi wątek będzie odczytywał ze strumienia danych komunikaty i dokonywał wstępnej selekcji komunikatów i albo od razu je obsługiwał (np. wysłał komunikaty zwrotne do serwera), albo wstawił komunikaty do kolejki tur, której elementy (identyfikujące konkretną turę) zawierać będą listę wszystkich komunikatów, które należy przetworzyć w danej turze. Warto zauważyć, iż kolejka tur nie będzie miała więcej niż $2\Delta - 1$ elementów. Do przetworzenia komunikatów z konkretnej tury używany będzie trzeci wątek, który, w razie potrzeby, może być wstrzymywany i wznowiany. Sterowanie przebiegiem symulacji odbywać się będzie za pośrednictwem wątku odbierającego komunikaty z serwera. Opis klas pakiet Net dla klienta przedstawia się następująco:

- Klasa Connection
Jest to główna klasa odpowiedzialna za połączenia sieciowe zrealizowana za pomocą wzorca projektowego Singleton. Klasa będzie odpowiedzialna za rozpoczynanie połączenia i zakończenie go.
- Klasa MessageReceiver
Klasa odpowiedzialna będzie za odbieranie wiadomości ze strumienia – pracować będzie w osobnym wątku. Pobierane dane zamieniane będą na klasy wiadomości przy pomocy metody Deserialize w nich zawartej. Następnie dla danej wiadomości uruchamiana będzie metoda Process w klasie MessageHandler
- Klasa MessageHandler
Jest to abstrakcyjna klasa występująca zarówno w kliencie jak i w serwerze. Jest pierwszą instancją przetwarzającą wiadomości
- Klasa MenuMessageHandler
Jest to klasa grupująca zdarzenia, odpowiedzialna za interakcje z menu. Jeśli np. znajduje się w oknie czatu, to przez wywołanie zdarzenia w tej klasie nowa wiadomość od użytkownika zostanie wpisana w odpowiednim oknie
- Klasa GameMessageHandler
Jest to klasa odpowiedzialna za przetwarzanie wiadomości, gdy użytkownik jest w stanie gry – przetwarza wiadomości kontrolne, wiadomości gry umieszcza w odpowiednim miejscu klasy Simulation.
- Klasa MessageSender
Jest to klasa odpowiedzialna jedynie za wysyłanie wiadomości. Zawiera ona bufor, do którego kierowane będą wiadomości dodawane przez klasę Connection



Przetwarzanie komunikatów w silniku serwera odbywa się analogicznie jak w przypadku klienta – w osobnych wątkach klienta obierane są wiadomości i umieszczane w kolejce wiadomości. Kolejka wiadomości przetwarzana jest przez osobny wątek. Osobny wątek odpowiada za wysyłanie wiadomości z kolejki wiadomości wychodzących i osobny za przetwarzanie gry sieciowej.



7. Sytuacje wyjątkowe

7.1. Modyfikacja zasobów

Aplikacja będzie kontrolować spójność zasobów. Gdy modyfikacja zagrazi poprawnemu działaniu aplikacji – aplikacja jest zamykana a użytkownik jest powiadamiany o błędzie. Dotyczy to modyfikacji grafiki, dźwięków, bibliotek zewnętrznych oraz plików tekstowych.



7.2. Brak połączenia z serwerem

W momencie braku połączenia z serwerem podczas rejestracji bądź uwierzytelniania użytkownik zostaje poinformowany o błędzie i wraca na początek procesu rejestracji bądź logowania.

7.3. Zerwanie połączenia podczas gry – strona klienta

Aplikacja kliencka potrafi wychwycić zerwanie połączenia, poinformować o tym gracza i przejść do menu wyboru gry.

7.4. Zerwanie połączenia podczas gry – strona serwerowa

Część aplikacji odpowiedzialna za połączenie z graczami potrafi wychwycić zerwanie połączenia. W takiej sytuacji informuje graczy o opuszczeniu gry przez danego gracza, a gra jest kontynuowana o ile jest więcej niż 2 graczy.

7.5. Długotrwałe opóźnienie podczas gry – strona serwerowa

W momencie, gdy na jednego z graczy serwer długo oczekuje komunikatów, serwer zrywa połączenie z danym graczem i informuje o przyczynie wyjścia gracza resztę graczy.

8. Podział prac i fazy projektu

Tydzień	Tydzień	Adam Nowacki	Paweł Rokoszný	Radosław Stankiewicz	Kamil Ślesiński	Piotr Witosławski
I	25 X - 1 XI	3	2	5	4	1
II	1 XI – 8 XI	3	2	1	1	4
III	8 XI - 15 XI	3	2	1	1	4
IV	15 XI – 22 XI	4	3	2	1	5
V	22 XI – 29 XI	4	2	3	1	5
Wersja wstępna						
VI	29 XI – 6 XII	4	2	3	1	2
VII	6 XII – 13 XII	2	1	2	2	1
Wersja beta						
VIII	13 XII – 20 XII	1	2	2	2	2
Wersja ostateczna						

8.1. Zadania

8.1.1. Zadania pierwszego tygodnia

1. Założenie projektu + ErrorLog
2. Wczytywanie XML
3. Budowa interfejsu użytkownika (menu)
4. Silnik graficzny gry
5. Stworzenie klas logiki gry na podstawie UML

8.1.2. Zadanie drugiego tygodnia



1. Zaimplementowanie Simulation, ClientSimulation, rysowania planszy
2. Ładowanie planszy, generowanie bitmapy planszy
3. Implementacja silnika sieciowego klienta: umożliwiające rozpoczęcie gry dowolnej
4. Implementacja silnika serwera umożliwiające rozpoczęcie gry dowolnej

8.1.3. Zadania trzeciego tygodnia

1. Zaimplementowanie Simulation, ClientSimulation, rysowania planszy
2. Podstawowa implementacja logiki budowy jednostek i budowli
3. Implementacja silnika sieciowego klienta – prowadzenie gry
4. Implementacja silnika sieciowego serwera – prowadzenie gry

8.1.4. Zadania czwartego tygodnia

1. Obsługa ruchu jednostek
2. Obsługa ataku jednostek
3. Obsługa mgły wojny i kolorów graczy
4. Implementacja silnika sieciowego klienta – prowadzenie gry
5. Implementacja silnika sieciowego serwera – prowadzenie gry

8.1.5. Zadania piątego tygodnia

1. Animacja (początek)
2. Implementacja zachowania jednostek specjalnych
3. Poruszanie się jednostek z uwzględnieniem przeszkód i A*
4. Implementacja pozostałej funkcjonalności klienta (przechodzenie po menu, logowanie, czat, zakończenie)
5. Implementacja pozostałej funkcjonalności serwera (chat, logowanie)

8.1.6. Zadania szóstego tygodnia

1. Animacja
2. Poruszanie się jednostek z uwzględnieniem przeszkód i A*
3. Poruszanie się żniwiarki
4. Dźwięki i opcje menu

8.1.7. Tydzień siódmy

1. Poruszanie się jednostek z uwzględnieniem przeszkód i A*
2. Czynności finalizacyjne i testy

8.1.8. Tydzień ósmy

1. Instalator
2. Dokumentacja powykonawcza

8.2. Fazy projektu



8.2.1. Wersja wstępna

Menu klienta:

- Funkcjonalność
 - Umożliwia założenie gry dowolnej
- Dźwięki nie są obsługiwane

Gra:

- Prowadzenie gry – pełna funkcjonalność z wyjątkiem
 - Żniwiarki nie poruszają się bez ingerencji użytkownika
 - Jednostki przechodzą przez budynki i najeżdżają na siebie
 - Brak dźwięków
 - Wyświetlanie w wersji podstawowej (brak animacji)

Serwer:

- Umożliwia prowadzenie gry dowolnej