

Wydział Matematyki i Nauk Informatycznych Politechniki  
Warszawskiej



# Project Game

Sebastian Jakubiak, Mikołaj Karaś, Tomasz Koter

v1.2

*27 listopada 2016r.*

# Spis treści

<b>1</b>	<b>Specyfikacja</b>	<b>2</b>
1.1	Opis biznesowy . . . . .	2
1.2	Wymagania funkcjonalne . . . . .	2
1.3	Wymagania нефункционалне . . . . .	6
1.4	Architektura rozwiązania . . . . .	7
1.5	Opis obiektów . . . . .	9
1.6	System komunikacji . . . . .	10
1.6.1	Mechanizm . . . . .	10
1.6.2	Protokół . . . . .	11

Tablica 1: Lista zmian

Data	Autor	Opis zmiany	Wersja
Nov 27, 2016	Sebastian Jakubiak, Mikołaj Karaś, Tomasz Koter	Rewizja i poprawa spójności doku- mentu. Pomniejsza korekcja tekstu.	1.2
Nov 27, 2016	Sebastian Jakubiak	Dodano opis klas. Poprawiono for- matowanie i wygląd tabel. Dodano diagramy w wyższej jakości.	1.1
Nov 26, 2016	Tomasz Koter	Pierwsza wersja dokumentu	1.0a

# 1. Specyfikacja

## 1.1. Opis biznesowy

*Project Game* jest grą mającą na celu symulację realizacji pewnego abstrakcyjnego projektu. Grać w nią może w ramach jednej sesji jednocześnie dowolna liczba ludzkich bądź komputerowych graczy w dwóch przeciwnych drużynach, zależna jedynie od upodobań założyciela gry oraz możliwości sprzętowych.

Jako dwie konkurujące drużyny gracze mają za zadanie zrealizować swój projekt jako pierwsi. Drużyna, która jako pierwsza skończy projekt, wygrywa, natomiast ta druga przegrywa.

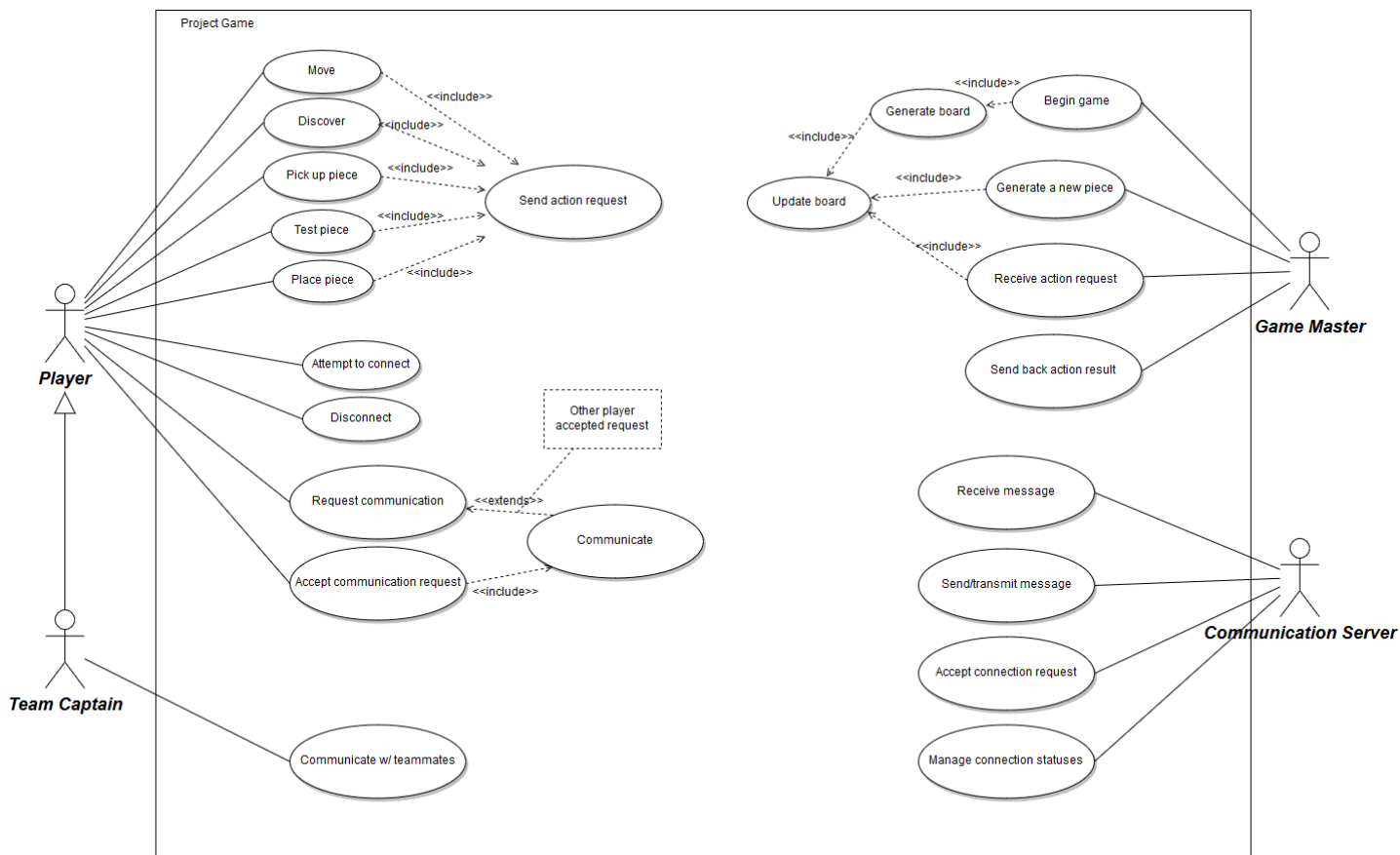
W ramach jednego projektu wyznaczone są cele, które dana drużyna musi osiągnąć, by zrealizować projekt. Każdy cel osiąga się poprzez podjęcie się zadania i wykonanie go w ramach tego celu. Może się jednak okazać, że wykonane zadanie nijak nie było pomocne w osiągnięciu celu, w związku z czym po wykonaniu go cel pozostaje nieosiągnięty.

Cele i zadania reprezentowane są za pomocą prostokątnej planszy składającej się z kwadratowych pól. Jedno pole może należeć do jednego z trzech obszarów: obszaru zadań, obszaru celów pierwszej drużyny lub obszaru celów drugiej drużyny. Gracze poruszają się po tej planszy w czterech kierunkach, z pola na pole. Zadania generowane są i umieszczane w polach obszaru zadań. Następnie gracze mogą podejmować się tych zadań i wypełniać je poprzez zanieśenie ich z obszaru zadań do obszaru celów swojej drużyny. Gracze nie wiedzą, które pola celów zawierają cel, który przysłuży się zrealizowaniu projektu, a które nie są celem projektu. Odkrywają je dopiero realizując pewne użyteczne zadanie w ramach tego pola. Gracz może upewnić się, czy zadanie jest użyteczne dowiadując się więcej na jego temat, lecz zajmuje to trochę czasu.

Gracze mogą komunikować się między sobą w ramach drużyny i w ten sposób współpracować.

## 1.2. Wymagania funkcjonalne

Poniższy diagram oraz tabela prezentują wszystkie możliwe przypadki użycia systemu gry, co jednocześnie pokrywa postawione w ramach tego projektu wymagania funkcjonalne.



Tablica 2: Przypadki użycia

Aktor	Lp	Nazwa	Opis
Gracz/Kapitan Drużyny	1	Move	Rusz graczem w kierunku góra/dół/lewo/prawo. Aby ruch został zrealizowany, Game Master musi zostać powiadomiony o chęci wykonania tego ruchu i go zatwierdzić (patrz # 6 <i>Send action request</i> ).
	2	Discover	Odkryj zawartość otaczających pól oraz poznaj ich odległości od najbliższego zadania. Aby akcja została zrealizowana, musi zostać wykonany # 6 <i>Send action request</i> .

Tablica 2: Przypadki użycia

Aktor	Lp	Nazwa	Opis
	3	Pick up piece	Podnieś zadanie z pola, na którym stoisz. Jeśli gracz stoi na polu z zadaniem, zadanie znika z planszy; w przeciwnym przypadku nic się nie dzieje. Akcja musi zostać zatwierdzona przez Mistrza Gry (patrz # 6 <i>Send action request</i> ).
	4	Test piece	Sprawdź, czy zadanie jest użyteczne. Jeśli jest bezużyteczne, zostaje wyrzucone. Akcja musi zostać zatwierdzona przez Mistrza Gry (patrz # 6 <i>Send action request</i> ).
	5	Place piece	Użyj zadania do osiągnięcia celu, na którym stoisz. Jeśli zadanie nie było użyteczne, nie dzieje się nic. Jeśli zadanie było użyteczne, odkrywa cel jako osiągnięty i informuje, czy był to cel służący do zrealizowania projektu. Akcja musi zostać zatwierdzona przez Mistrza Gry (patrz # 6 <i>Send action request</i> ).
	6	Send action request	Wyślij zlecenie wykonania akcji (dowolnego przypadku użycia z przedziału 1-5) przez gracza do Mistrza Gry, by mógł potwierdzić poprawność operacji oraz zaktualizować planszę. Wykonywane bez wywołania przez gracza w momencie zlecenia dowolnej z akcji 1-5. Gracz nie posiada żadnych danych poza tymi, które są mu znane. Stąd każda akcja musi być zatwierdzona przez GM, który odsyła zaktualizowaną planszę po każdej zmianie. To zapobiega kolizjom, desynchronizacji i próbom hackowania gry.
	7	Attempt to connect	Spróbuj połączyć się z serwerem gry. W wypadku błędu gracz będzie próbował aż do skutku bądź osiągnięcia limitu prób. Potem dołączy do aktualnego stanu gry (powodzenie) lub się wyłączy (niepowodzenie).
	8	Disconnect	Rozłącz się z serwerem gry (opuść grę). Natychmiast po tym wyłącza aplikację gracza. Gracz rozłącza się z własnej woli jedynie w przypadku zakończenia gry, chyba że pewna implementacja sztucznej inteligencji w przyszłości będzie zamierzenie rozłączać gracza.

Tablica 2: Przypadki użycia

Aktor	Lp	Nazwa	Opis
	9	Request communication	Wyślij do innego gracza prośbę o rozpoczęcie korespondencji. Skutkuje otwarciem kanału komunikacji między graczami w wypadku zaakceptowania prośby; w przypadku odrzucenia - informacja o tym; w przypadku braku odpowiedzi, po upływie określonego czasu - równoznaczne z odrzuceniem. Od tego momentu gracze mogą fizycznie komunikować się ze sobą.
	10	Accept communication request	Zaakceptuj prośbę komunikacji od innego gracza. Otwiera kanał komunikacji między graczami.
	11	Communicate	Komunikuj się z innym graczem z drużyny. Wyślij/przeczytaj wiadomość od innego gracza. Wymaga zaakceptowania prośby o komunikację przez docelowego członka drużyny (# 10).
Kapitan Drużyny	12	Communicate with teammates	Komunikuj się z innym członkiem drużyny. Kapitan drużyny ma większe kompetencje od zwykłego gracza i nie musi prosić o komunikację. Automatycznie otwiera dwustronne połączenie do komunikacji między kapitanem a docelowym graczem.
Game Master	13	Begin game	Wyzwala akcję generowania planszy (# 14). Ustawia stan gry na rozpoczęty. Informuje o tym graczy za pośrednictwem serwera komunikacji.
	14	Generate board	Generuje planszę po raz pierwszy według podanych w pliku konfiguracyjnym lub w programie danych wejściowych. Wymusza aktualizację planszy i powiadomienie o tym graczy (akcja # 17).
	15	Generate a new piece	Generuje nowy element <i>piece</i> na planszy w obszarze <i>task area</i> . Wymusza aktualizację planszy (# 17).
	16	Receive action request	Odbiera zlecenie wykonania akcji przez gracza (wysłane przez # 6 <i>Send action request</i> ). Wykonanie zleconej akcji wymusza aktualizację planszy (# 17) oraz odesłanie informacji o powodzeniu akcji (# 18).
	17	Update board	Wprowadza zmiany w planszy i rozsyła odpowiednie informacje o stanie planszy przeznaczone dla graczy.

Tablica 2: Przypadki użycia

Aktor	Lp	Nazwa	Opis
	18	Send back action result	Odsyła informację o powodzeniu zleconej akcji (odebranej w # 16).
Communication Server	19	Receive message	Odbiera wiadomość będącą na wejściu danego połączenia.
	20	Send/transmit message	Przekierowuje odebraną wcześniej wiadomość (# 19) do odpowiedniego odbiorcy lub wysyła własny komunikat.
	21	Accept connection request	Akceptuje próbę połączenia przez danego klienta.
	22	Manage connection statuses	Kontroluj stany połączeń. Tworzy nowe tunele dla konwersacji między graczami. Zamyka tunele zakończonych konwersacji. Sprawdza i informuje o nagłych rozłączeniach.

Tablica 3: Aktorzy korzystający z systemu

Nazwa	Opis
Gracz	Gracz uczestniczący w rozgrywce. Jest programem uruchamianym automatycznie na potrzeby gry.
Kapitan Drużyny	Wyróżniony gracz, który może bez wysyłania prośby komunikować się z innymi graczami z drużyny. Każda drużyna ma dokładnie jednego kapitana.
Game Master	Aplikacja kontrolująca przebieg gry. Generuje planszę, przekazuje informacje o niej graczom, aktualizuje ją i decyduje o rozpoczęciu i zakończeniu rozgrywki.
Communication Server	Aplikacja służąca do ustanowienia komunikacji między graczami i Game Masterem. Jej jedynym celem jest przesyłanie danych i zarządzanie połączeniami.

### 1.3. Wymagania niefunkcjonalne

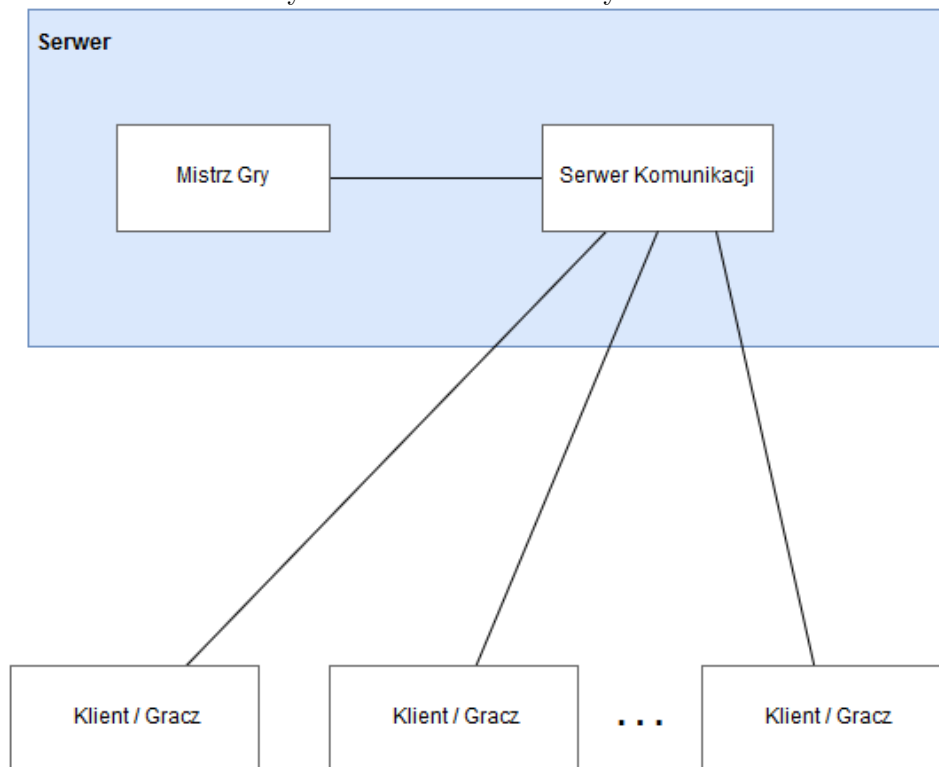
Żadne wymagania niefunkcjonalne nie zostały określone dla tego projektu.

#### 1.4. Architektura rozwiązania

Program zostanie zaimplementowany w strukturze klient-serwer, gdzie klientami będą poszczególni gracze korzystający z aplikacji klienckiej (możliwe będzie korzystanie z jednej maszyny/aplikacji przez kilku graczy jednocześnie z opcją podziału ekranu). Po stronie serwera działać będzie serwer komunikacji oraz jednostka Mistrza Gry.

Wszelka komunikacja gracz-gracz oraz gracz-Mistrz Gry będzie odbywać się za pośrednictwem serwera komunikacji, nawet w wypadku skonfigurowania całego systemu lokalnie na jednej maszynie.

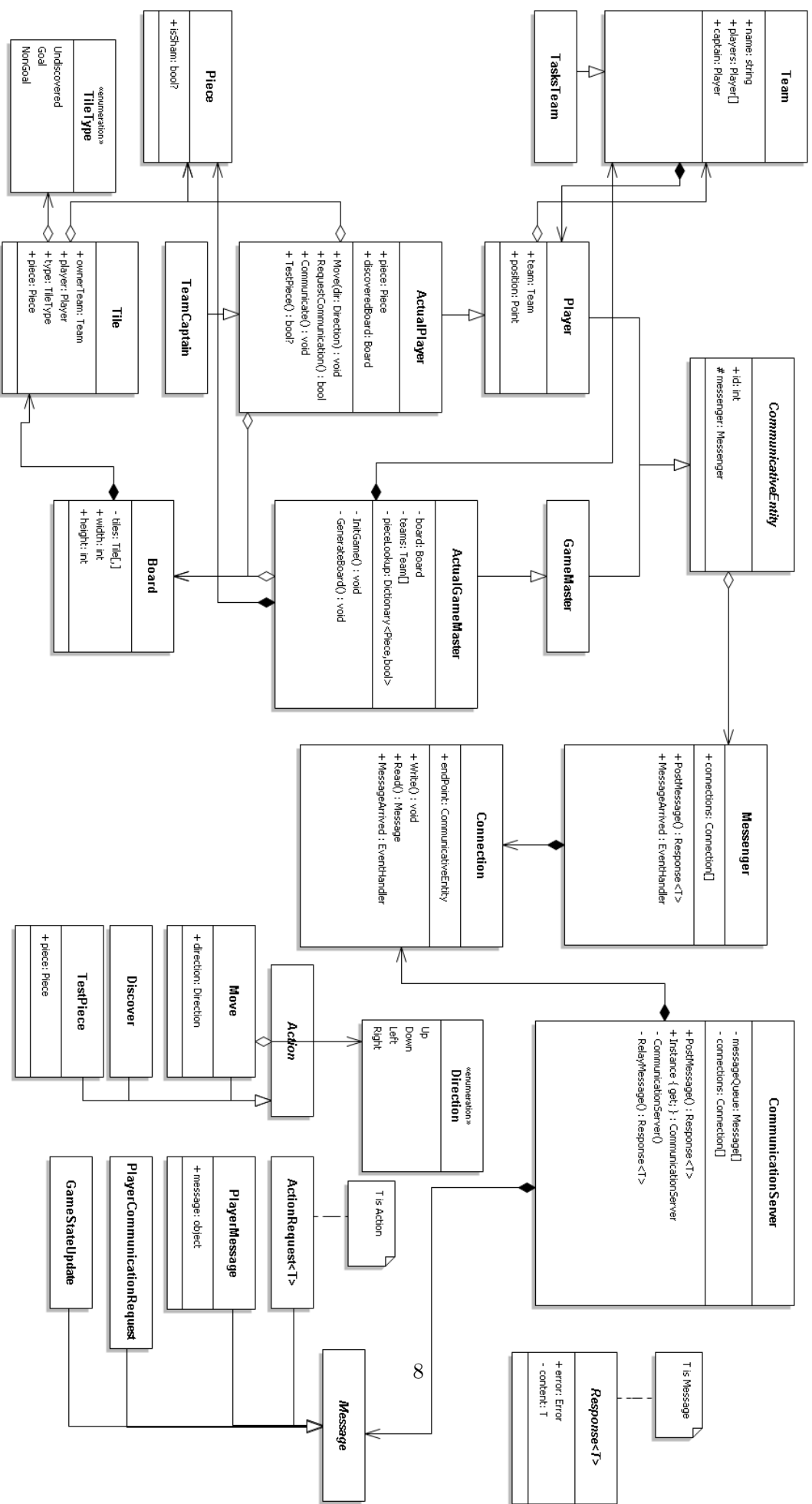
Rysunek 1: Architektura systemu



Zależności między obiektami powinny zachodzić jak na poniższym diagramie klas.



Rysunek 2: Diagram klas



## 1.5. Opis obiektów

**Player** Podstawowe informacje o gracz: drużyna, pozycja na planszy.

**GameMaster** Reprezentacja mistrza gry.

**ActualPlayer** Bieżący użytkownik systemu, jeśli jest on graczem. Posiada operacje wykonywania ruchów i akcji w grze oraz komunikacji z innymi graczami. Przechowuje zdobyte informacje o stanie gry (planszę) oraz o zadaniu, jeśli jakieś znalazł w trakcie gry.

**ActualGameMaster** Bieżący użytkownik systemu, jeśli jest on mistrzem gry. Posiada operacje inicjalizacji gry i generowania planszy. Przechowuje pełną informację o planszy i stanie gry oraz o tym, które zadania są użyteczne, a które nie.

**Team** Drużyna. Są dwie faktyczne drużyna oraz jedna sztuczna, która jest „właścicielem” pól w obszarze zadań.

**Board** Plansza, czyli stan gry — faktyczny (w przypadku mistrza gry) lub widziany przez gracza.

**Field** Pojedyncze pole na planszy. Posiada odwołanie do drużyny-właściciela; w przypadku pola w obszarze celów będzie to prawdziwa drużyna, a w przypadku pola w obszarze zadań — instancja `TasksTeam`. Pole posiada informację o gracz, który na nim stoi. Ponadto dla pól w obszarze zadań jest informacja o zadaniu, które może się znajdować na danym polu, a dla pól w obszarze celów informacja, czy na polu znajduje się cel (o ile użytkownik to wie).

**Piece** Zadanie. Może być użyteczne lub nie (użytkownik może nie wiedzieć).

**Action** Akcja wykonywana przez gracza, taka jak ruch, test zadania, rozpoczęcie komunikacji z innym graczem.

**CommunicativeEntity** Użytkownik systemu mogący komunikować się z innymi użytkownikami. Do komunikacji używa instancji klasy `Messenger`.

**Messenger** Obiekt wykorzystywany przez użytkownika do komunikacji z innymi. Umożliwia wysyłanie komunikatów i reagowanie na komunikaty przychodzące. Utrzymuje listę połączeń do innych użytkowników (tunelowanych przez serwer komunikacji).

**Message** Komunikat nadawany lub odbierany przez użytkownika. Wyróżnia się kilka rodzajów:

**ActionRequest<T>** Żądanie zatwierdzenia pewnej akcji gracza, wysyłane przez niego do mistrza gry.

**PlayerMessage** Wiadomość od jednego gracza do drugiego.

**PlayerCommunicationRequest** Prośba o rozpoczęcie wymiany informacji, kierowana przez jednego gracza do drugiego.

**GameStateUpdate** Informacja (być może częściowa) o zmianach stanu gry (czyli o zmianach na planszy), wysyłana przez mistrza gry do gracza (np. skutek wykonania przez gracza poprawnej akcji).

**Connection** Połączenie z użytkownikiem. Obiekt odpowiadający za komunikację niskopoziomową.

**CommunicationServer** Serwer komunikacji przekazujący wiadomości między użytkownikami. Singleton. Posiada listę bezpośrednich połączeń do użytkowników. Nadchodzące wiadomości są gromadzone w kolejce, z której serwer je pobiera i przekazuje do adresatów.

## 1.6. System komunikacji

### 1.6.1. Mechanizm

Wszelka komunikacja może występować w następujących płaszczyznach:

**Mistrz Gry - Serwer Komunikacji** Służy ustanowieniu połączenia. Ponadto serwer komunikacji informuje mistrza gry o zaburzeniach toku gry w postaci rozłączeń.

**Gracz - Serwer Komunikacji** Służy ustanowieniu i utrzymaniu połączenia.

**Gracz - Mistrz Gry** Mistrz Gry informuje graczy o zmianach w układzie planszy, przekazuje im konsekwencje ich działań oraz o statusie gry. Gracze zlecają Mistrzowi Gry wykonanie operacji na rzecz gracza.

**Gracz - Gracz** Gracze mogą komunikować się między sobą, by przekazać informacje o planszy i stanie gry z ich perspektywy.

Niezależnie od płaszczyzny, **serwer komunikacji** zawsze jest ogniwem w drodze przekazywania komunikatów. Albo on sam komunikuje się (MG - Serwer lub Gracz - Serwer), albo przekazuje komunikaty między graczami a mistrzem gry lub graczami a graczami. Nie istnieją połączenia bezpośrednie, omijające **serwer komunikacji**.

Pomimo tego, że serwer komunikacji pośredniczy w każdej wymianie informacji, każda jednostka zdolna komunikować się (*CommunicativeEntity*) posiada moduł (*Messenger*), który wykonuje wszystkie działania potrzebne do komunikacji. Z punktu widzenia takiej jednostki komunikacja jest bezpośrednia, ponieważ dla każdej pary jednostek komunikujących się **serwer komunikacji** tworzy połączenie w postaci wirtualnego tunelu. Tunel posiada dwa punkty wejścia/wyjścia, to jest *Messenger*y dwóch jednostek komunikujących się ze sobą. Z zewnątrz nie widać żadnych mechanizmów wewnętrznych, można jedynie pisać do tunelu i odczytywać wiadomości z niego. W rzeczywistości jednak tunel prowadzi przez **serwer komunikacji**, który przekazuje wiadomości na bieżąco do *Messengerów*.

### 1.6.2. Protokół

Wszystkie wiadomości zostają przekonwertowane na format JSON. Wybór ten jest motywowany niezależnością od języka programowania, powszechnością parserów oraz choćby tym, że format JSON jest bardziej zwężły niż XML. Następnie łańcuch znaków jest pakowany w pakiety TCP/IP i transmitowany. TCP, mimo że powolniejszy niż UDP, bo wymaga potwierżeń i uwzględnia retransmisję, jest wygodniejszy w użyciu. Ponadto prędkość UDP nie zostałaby w pełni wykorzystana, ponieważ gra nie jest na tyle złożona.

Przykładowy komunikat - zlecenie ruchu o jedno pole w górę - wygląda następująco:

```
{
    "type": "ActionRequest<Move>",
    "data": {
        "dir": "0"
    }
}
```