The Project Game

Paweł Szymkiewicz
Krzysztof Tabeau
Kamil Todorowski
Aleksander Wiśniewski
Krzysztof Woźniak

Styczeń 2020

1 Zasady gry

Gra odbywa się na prostokątnej planszy podzielonej na kwadratowe pola. Plansza podzielona jest na trzy części, każda o szerokości równej szerokości planszy: u góry i u dołu planszy znajdują się dwie *Goal Area*, między nimi znajduje się *Task Area*. Obie *Goal Area* są tych samych rozmiarów.

Po całej planszy poruszają się gracze. W *Goal Area* znajdują się wyróżnione pola typu *Goal*. W *Task Area* pojawiają się obiekty Piece, z którymi gracze mogą wchodzić w interakcje. Pełny, prawdziwy stan planszy w danym momencie gry znany jest tylko *Game Masterowi*.

Gracze podzieleni są na dwa równoliczne zespoły. Każdemu zespołowi przypisana jest jedna z *Goal Area*.

Przed rozpoczęciem gry, *Game Master* wskazuje w każdym zespole jednego gracza, który zostaje Liderem.

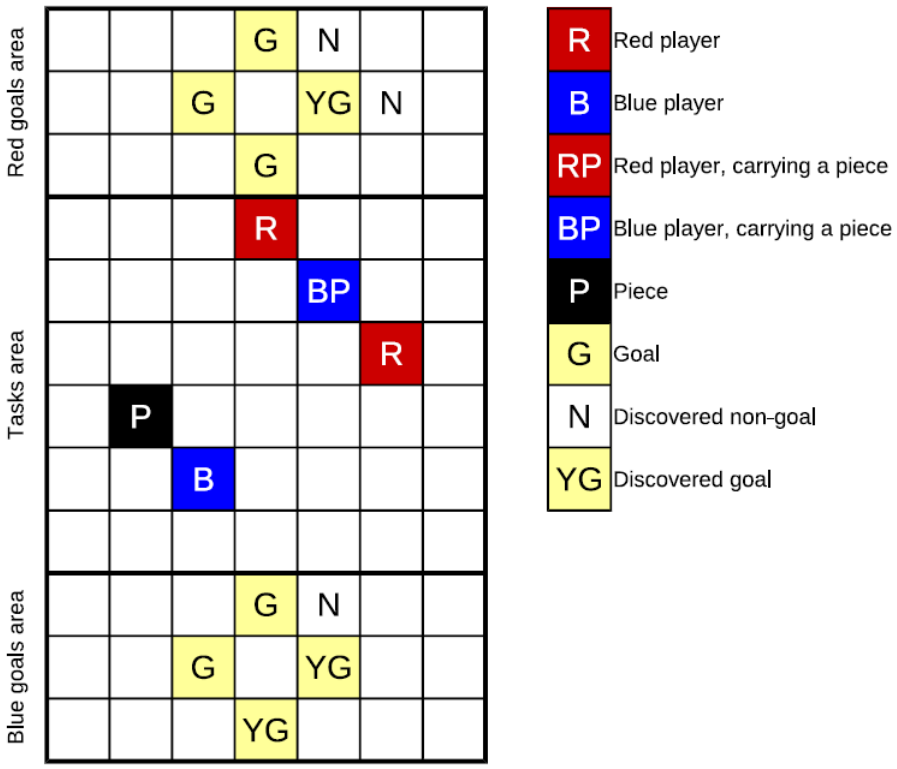
Gracze rozpoczynają grę w losowym polu *Task Area*. Gracz początkowo ma wiedzę o rozmiarze planszy, położeniu Goal Area swojego zespołu, odległości wyrażonej w metryce miejskiej do najbliższego *Piece* i swojej lokalizacji na planszy.

W każdym momencie gry w *Task Area* znajduje się stała liczba *Piece*. Piece może być automatycznie podniesiony przez jednego z graczy, który nie ma aktualnie *Piece*, gdy wejdzie on na pole na którym znajduje się *Piece*. Wówczas *Game Master* w losowym miejscu *Task Area* niezajętym przez żadnego z graczy umieszcza kolejny *Piece*. Gracz może trzymać jeden *Piece* na raz.

Gracz może położyć trzymany *Piece* na polu, na którym się znajduje. Gdy gracz położy *Piece*, otrzymuje informację, czy pole na którym je położył było *Goal* jego drużyny. Jeśli na pole *Goal* zostaje położone *Piece*, zamienia się ono w zwykłe pole.

Każdy *Piece* ma określoną szansę na bycie *Sham Piece*. Gdy gracz położy *Sham Piece*, nie otrzymuje żadnej informacji po jej położeniu. *Sham Piece* po położeniu znika i nie ma wpływu na stan pola *Goal*. Gracz ma możliwość sprawdzenia, czy *Piece* który trzyma jest *Sham Piece*.

Celem gry jest umieszczenie przez graczy *Piece* w każdym *Goal* w *Goal Area* swojej drużyny. Zespół, który pierwszy osiągnie cel, wygrywa grę. Gra zostaje przerwana gdy jeden z graczy



Rysunek 1: Rzeczywisty stan gry

przedwcześnie się rozłączy.

Gra nie ma limitu czasowego.

Każda akcja wykonywana przez gracza powoduje, że gracz staje się nieaktywny na określony czas (timeout).

Gracz może poruszać się po planszy o jedno pole w jednym z czterech kierunków: góra, dół, lewo, prawo. Gracz nie może wyjść poza planszę i wejść na pole zajęte przez innego gracza.

Po każdym ruchu gracz otrzymuje informację o odległości od najbliższego niepodniesionego *Piece*, wyrażoną w metryce miejskiej.

Gracz może poprosić *Game Mastera* o informację o odległości od najbliższego niepodniesionego *Piece* dla wszystkich pól mających wspólną krawędź lub wierzchołek z polem na którym znajduje się gracz.

Gracze z jednego zespołu mogą wymieniać między sobą informacje o planszy, o swoim położeniu, posiadaniu *Piece*, a jeśli posiadają *Piece*, to o polu, na którym chcą je położyć.

Gracz może poprosić o informacje od innego gracza. Gracz nie musi udzielić odpowiedzi, chyba że pytający gracz jest Liderem.

Rozmiar planszy, rozmiar *Goal Area*, liczba graczy, liczba niepodniesionych kawałków znajdujących się na planszy, liczba pól *Goal* w jednej *Goal Area*, timeouty akcji, szansa *Piece* na bycie *Sham Piece* mogą być konfigurowane. Muszą one jednak spełniać następujące warunki:

1. Szerokość planszy musi być liczbą naturalną i wynosić nie mniej niż 2.

2. Wysokość planszy musi być liczbą naturalną nie mniejszą niż szerokość.

3. Wysokość *Goal Area* musi być liczbą naturalną nie większą niż ((wysokość planszy − 1) / 2).

4. Liczba graczy w drużynie musi być liczbą naturalną mniejszą niż szerokość planszy.

5. Liczba pól *Goal* musi być liczbą naturalną nie mniejszą niż 1 i nie większą niż (wysokość *Goal Area* · szerokość planszy).

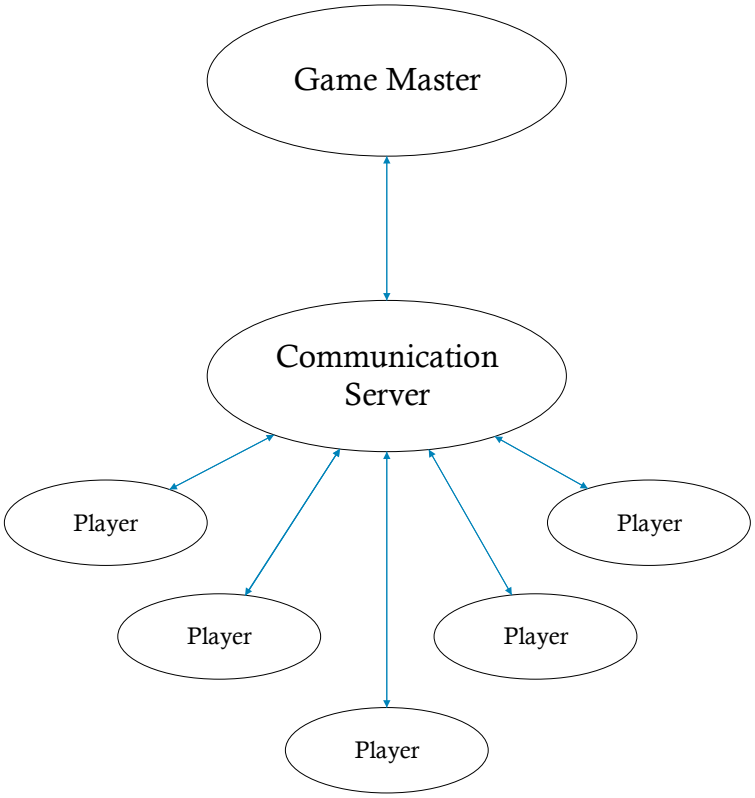
6. Liczba niepodniesionych kawałków znajdujących się na planszy musi być liczbą naturalną nie mniejszą niż 1 i nie większą niż (liczba pól *Task Area* − liczba graczy w obu drużynach).

7. Timeouty muszą być liczbami naturalnymi nie mniejszymi niż 0.

8. Szansa *Piece* na bycie *Sham Piece* musi być liczbą rzeczywistą nie mniejszą niż 0 i mniejszą niż 1.

2 Architektura

Cała gra opiera się na trzech niezależnych komponentach: *Game Master*, *Communication Server* oraz *Player*. Całą grą i jej przebiegiem zarządza *Game Master*. Do *Game Master* muszą podłączyć się gracze, by móc wziąć udział w grze. To on kontroluje każdy ruch i ocenia ich legalność. Cała komunikacja odbywa się poprzez *Communication Server*. Niemożliwa jest bezpośrednia komunikacja *Player - Player* lub *Player - Game Master* bez udziału *Communication Servera*. Aktywnymi uczestnikami gry są *Playerzy*. Ich celem jest zebranie *Piece'ów* leżących na planszy i dostarczenie ich na *Goal Area* swojej drużyny. W tym projekcie *Playerzy* są autonomicznymi botami.



3 Wymagania

3.1 Funkcjonalność

- System - Windows 10, 2GB RAM, 2GB miejsca na dysku twardym
- Logi zapisywane w formie pliku tekstowego w katalogu głównym gry
- Komunikacja poprzez TCP
- Język - C#, wymagany .NET Framework 4.7.1 lub nowszy
- Program nierozszerzalny

3.2 Użyteczność

- Gra w języku polskim
- Widok planszy dla obserwatora
- Edytowanie konfiguracji w pliku tekstowym
- Plansza - ciemne tło z białą kratką
- Drużyna - czerwona i niebieska
- Goal w przyciemnionych kolorach drużyn
- Piece: nie Sham - zielony, Sham - żółty
- Gracz posiadający Piece ma jego małą ikonkę obok swojej

3.3 Niezawodność

- Przewidywane opóźnienie wiadomości do 20-100ms
- W przypadku otrzymania przez którykolwiek komponent systemu wiadomości niepoprawnej, dany komunikat jest ignorowany
- Jeżeli jeden z uczestników gry rozłączy się podczas rozgrywki, gra zostaje zakończona z wynikiem na niekorzyść drużyny rozłączonego agenta
- Utrata połączenia:
- Utrata łączności z agentem: Gdy od serwera komunikacyjnego rozłączy się gracz, to ten przesyła do GameMastera informację o rozłączeniu gracza. W konsekwencji GameMaster wywoła metodę EndGame() i prześle do każdego gracza informację o zakończeniu gry, wraz z wynikami. Wyniki nie zawsze będą prawdziwe: drużyna, z której rozłączył się gracz, będzie miała 0 punktów, natomiast druga drużyna: 1 punkt.
- Utrata łączności z CS: Gdy GameMaster lub Player nie będą w stanie połączyć się z serwerem komunikacyjnym (zawiesi się połączenie TCP), to będą próbowali ponowić połączenie. Jeśli w ciągu 10 sekund nie odzyskają połączenia, gra się kończy bez wyłonienia zwycięzcy.

- Utrata łączności z GameMaster: Gdy CS nie będzie w stanie połączyć się z GameMasterem (zawiesi się połączenie TCP), będzie próbował ponowić połączenie. Jeśli w ciągu 10 sekund nie odzyska połączenia, gra się kończy bez wyłonienia zwycięzcy.

3.4 Wydajność

- Minimalna precyzja GM 1ms
- Liczba graczy, przy której gra gwarantuje poprawne działanie:
  - Minimalna: 1 na drużynę
  - Domyślna i zalecana: 5 na drużynę
  - Maksymalna: 8 na drużynę
- Wielkość planszy gry gwarantująca poprawne działanie:
  - Minimalna: 9 pól (3x3)
  - Domyślna i zalecana: 100 pól (10x10)
  - Maksymalna: 512 pól (32x16)

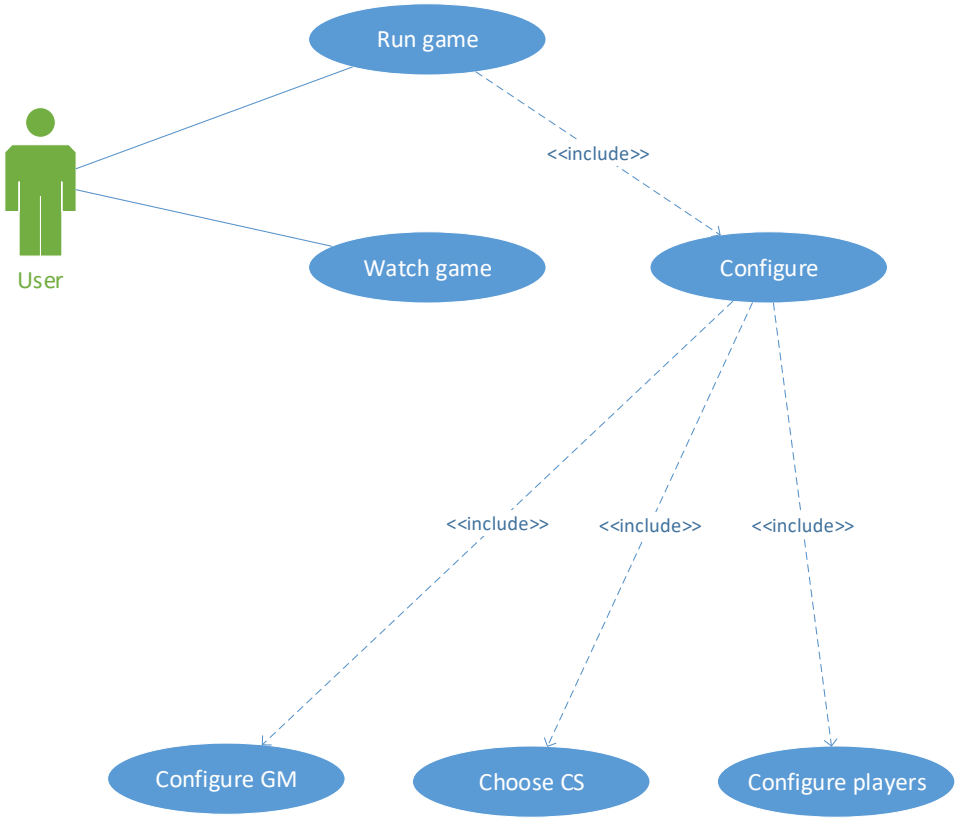
3.5 Wsparcie

- Brak potrzeby instalacji
- Brak możliwości zapisu gry
- Możliwość podłączenia własnego GM, Usera lub CS.

4 Diagramy użycia

4.1 User

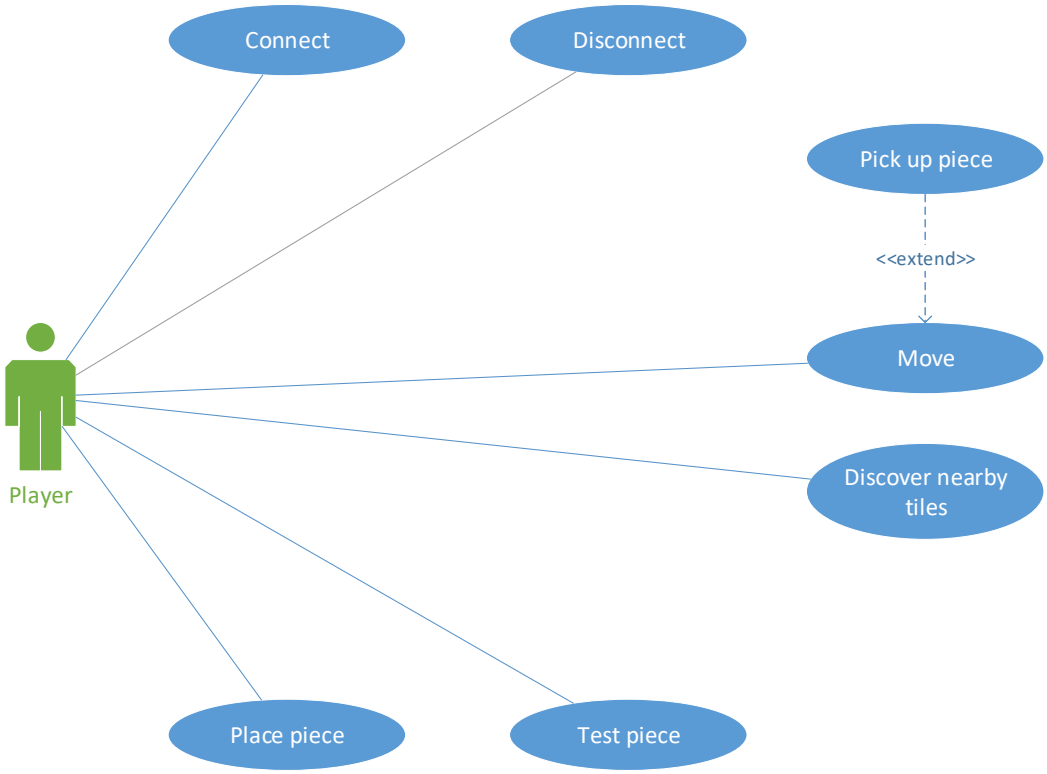
- Użytkownik może uruchomić Communication Server, Game Mastera lub Agenta, w których konfiguruje parametry uruchomienia
- Możliwa jest obserwacja trwającej rozgrywki z punktu widzenia Game Mastera lub Agenta

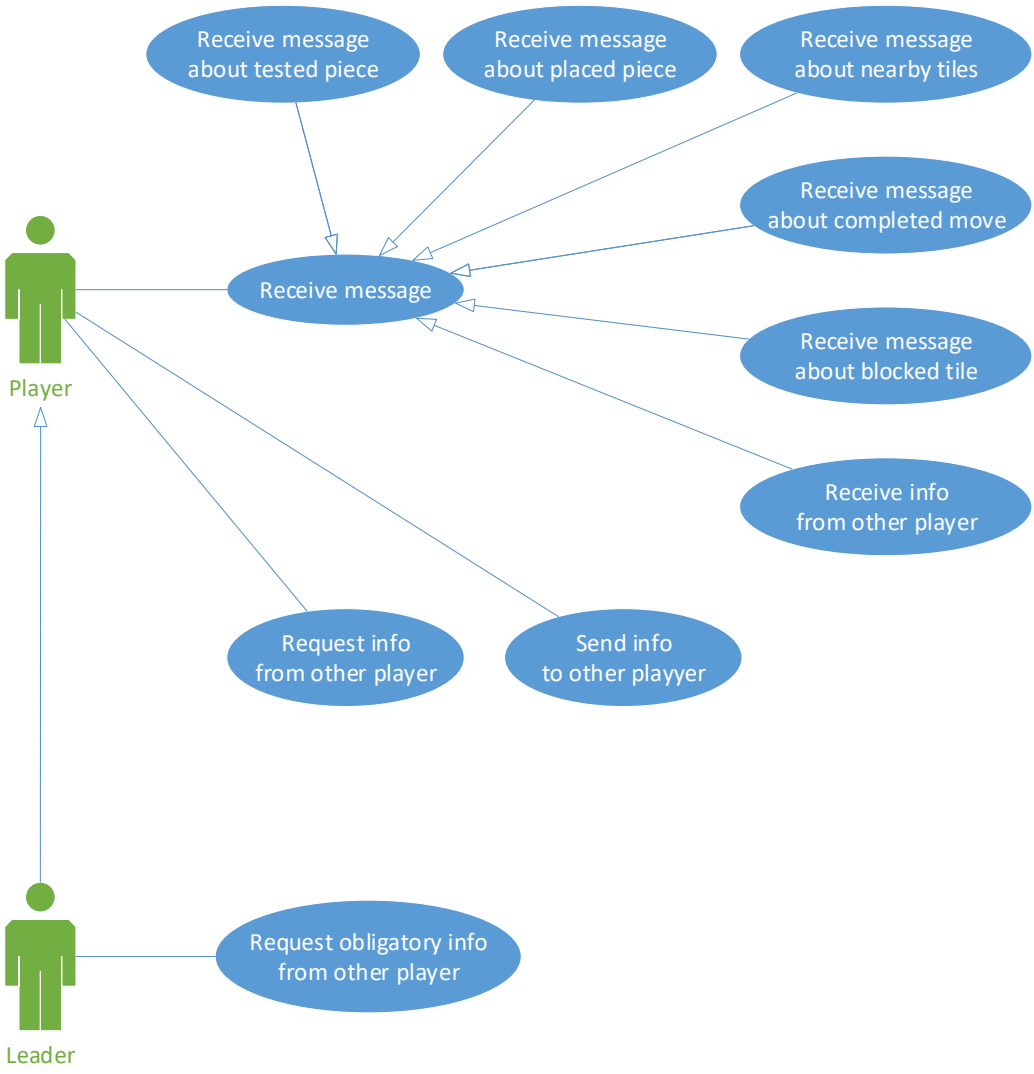


4.2 Player

- Player może połączyć się z grą, a po dołączeniu w dowolnym momencie się rozłączyć
- Podczas rozgrywki dostępne dla Playera akcje to ruch (oraz automatycznie podniesienie kawałka, jeżeli Player stanął na polu z leżącym Piece), odkrycie odległości do najbliższych Piece z ośmiu otaczających pól, testowanie prawdziwości trzymanego Piece oraz położenie go
- Komunikację z Agentami z drużyny Player odbywa wysyłając swoją wiedzę na temat stanu planszy i prosząc o te informacje innych graczy. Jeżeli Player został desygnowany przez Game Mastera na lidera danej drużyny, ma on możliwość wymuszenia odpowiedzi od sojuszników
- Player odbiera wiadomości dotyczące informacji o odległości do najbliższego Piece z otaczających pól, informacji o spełnieniu Goala po położeniu kawałka, nieprawidłowości żądanego

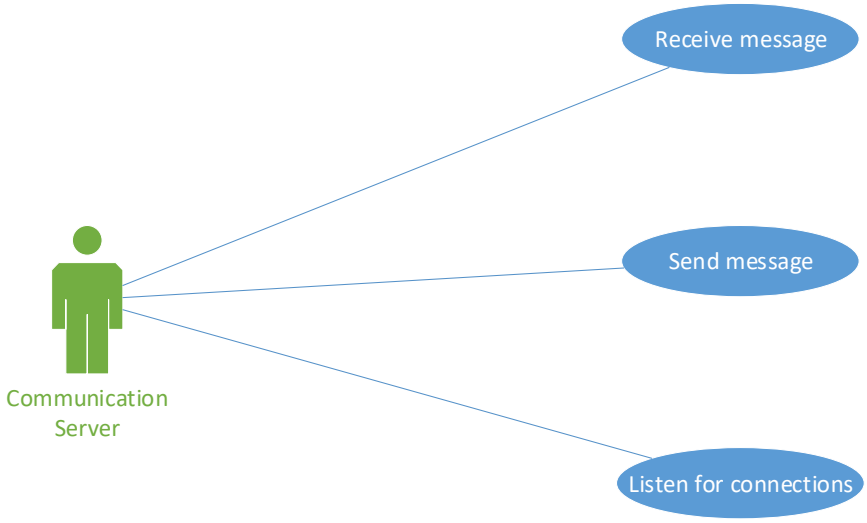
ruchu z uwagi na stojącego Agenta na docelowym Tile oraz wiedzy sojuszników na temat stanu planszy





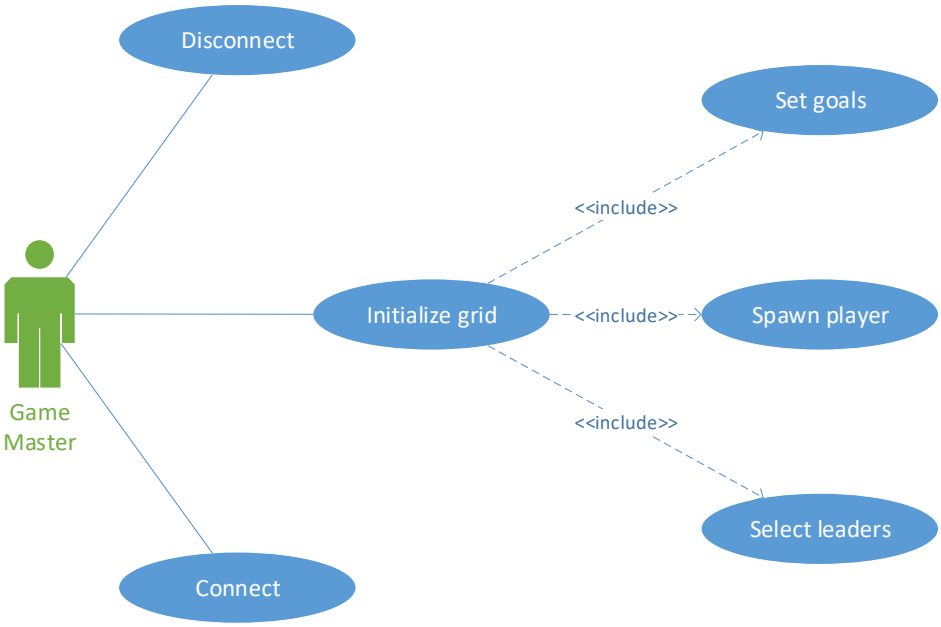
4.3 Communication Server

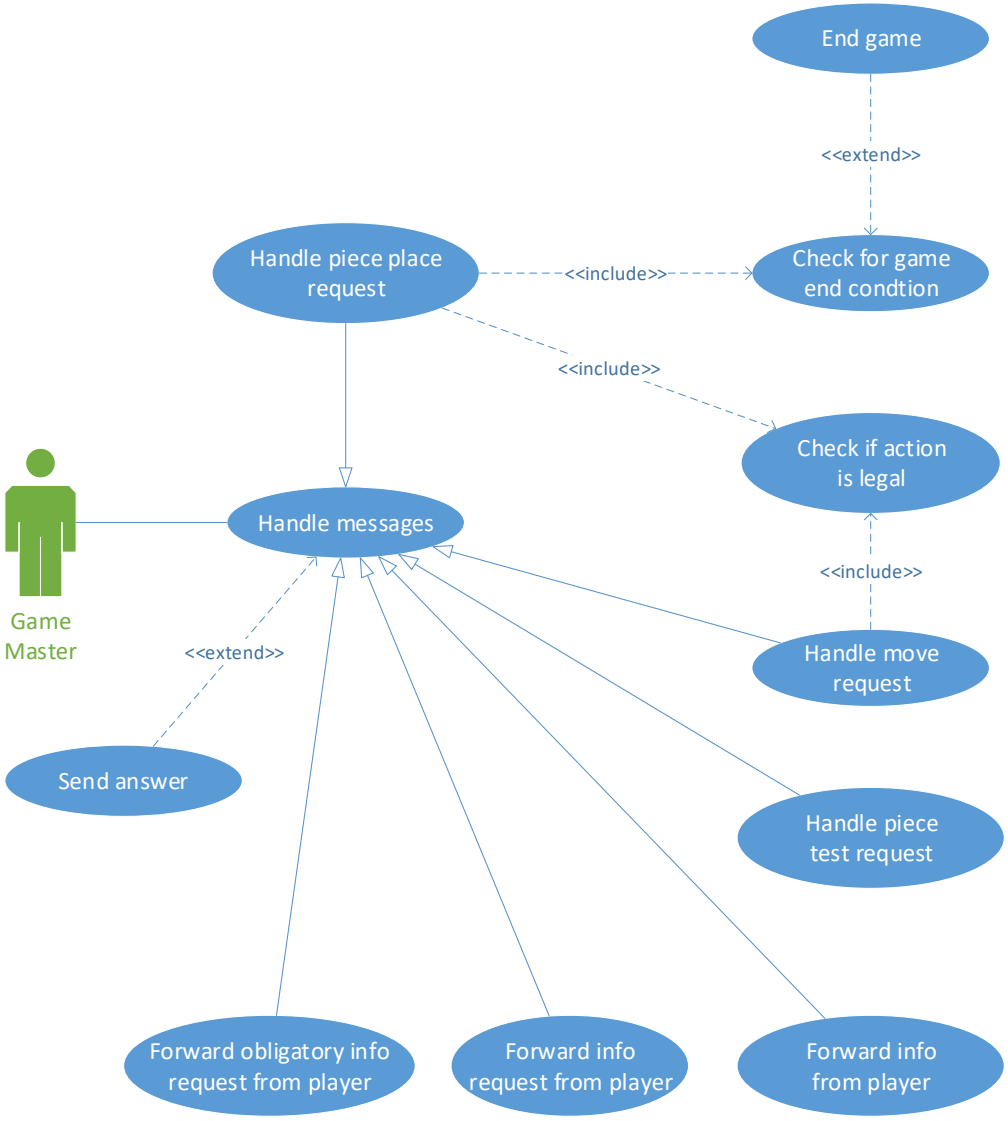
- Zadania Communication Server to nasłuchiwanie przychodzących połączeń oraz przekazywanie wiadomości pomiędzy Playerami a Game Masterem



4.4 Game Master

- Game Master może się połączyć i rozłączyć z Communication Server
- Podczas rozpoczynania rozgrywki, inicjując planszę, Game Master losuje pozycje Goali, Playerów oraz wybiera liderów drużyn
- Zadaniem Game Mastera jest obsługiwanie żądań graczy: testowanie prawdziwości Piece, przekazywanie informacji o stanie gry między graczami, obsługa ruchu graczy i próśb o położenie Piece. Niektóre z akcji wymagają sprawdzenia ich poprawności
- W przypadku położenia Piece na Goal, Game Master sprawdza warunek zakończenia gry: spełnienie wszystkich Goali danej drużyny. Gdy wszystkie Goale jednego zespołu zostaną spełnione, Game Master kończy rozgrywkę.


Disconnect
Set goals
<<include>>
Initialize grid
<<include>>
Spawn player
<<include>>
Select leaders
Connect
Game
Master



5 Diagramy klas i komentarze

5.1 Player

Moduł *PLAYER* opisuje pojedynczego gracza. *PLAYER* agreguje dwie klasy: *GRID* oraz *TEAMMATE MODEL*. Pierwsza z nich reprezentuje planszę gry widzianą przez gracza. Zawiera ona informacje o wielkości planszy oraz wysokości *GOAL AREA*. Klasa *GRID* agreguje wiele obiektów klasy *TILE*; **każdy *TILE* przechowuje poniższe informacje**:

- **IsPossibleGoal**: przyjmuje wartość true, jeśli *TILE* jest w *GOAL AREA* oraz nie został on wcześniej uznany jako *GOAL*/nie *GOAL* (to znaczy, jest sens z perspektywy gracza by postawić tam *PIECE*). W innym wypadku przyjmuje wartość false.
- **DistToPiece**: określa odległość od zadanego *TILE'A* do najbliższego *PIECE'A*, w metryce

Manhattan. Dowolny ruch gracza powoduje, że *GAME MASTER* zwraca do ruszającego się gracza informacje o DistToPiece na nowej pozycji, po czym aktualizowane jest to pole i ustawiany Timestamp na obecnę chwilę.

- **Timestamp**: określa czas, w którym informacje o zadanym *TILE'U* zostały określone (każda aktualizacja zawartości *TILE'A* powoduje, że Timestamp jest aktualizowany).

Klasa *GRID* posiada też pola:

- **Width**: szerokość planszy.
- **Height**: wysokość planszy.
- **GoalAreaHeight**: wysokość Goal Area.

PLAYER agreguje kolekcję *TEAMMATE MODELI*. Każdy taki model ma reprezentować wiedzę gracza o swoich sojusznikach:

- **PlayerId**: unikalny identyfikator gracza, nadany mu przez *GM*. Umożliwi to wybieranie gracza do którego ma zostać wysłany RequestInfo().
- **Location**: położenie gracza.
- **HasPiece**: czy sojusznik posiada *PIECE*.
- **Direction**: w jakim kierunku porusza się gracz.

W ramach wymian informacji między graczami (RequestInfo(), SendInfo()), gracze przesyłają swój widok planszy, a także wszystkie informacje potrzebne do uzupełnienia *TEAMMATE MODELU*.

Oprócz wymienionych wyżej zagregowanych klas, PLAYER posiada pola:

- **Direction**: wynika ze strategii objętej przez gracza i określa jaki jest obecnie cel gracza (tj. pole na planszy o jakich współrzędnych). Ta informacja jest istotna przy wymianie informacji przez graczy, ponieważ jest podstawą do określania strategii drużyny (a nie pojedynczego gracza).
- **IsSleeping**: przyjmuje wartość true, jeśli gracz jest obecnie stimeoutowany i powinien spać, czekając aż *GAME MASTER* go wybudzi.
- **Requests**: jest to priorytetowa kolejka par (int, bool), która ma reprezentować informację o zapytaniach, które otrzymał dany gracz, ale których jeszcze nie przetworzył. Liczba całkowita jest indeksem pytającego gracza, a bool określa, czy zadany gracz jest liderem. Priorytetem kolejki jest to, czy dany pytający *PLAYER* jest liderem.
- **Socket**: gniazdo związane z *COMMUNICATION SERVER*.
- **PieceState**: wyliczenie, którego wartość określa wiedzę gracza o posiadanym kawałku. NotPicked oznacza, że gracz nie ma podniesionego kawałka. UnChcked, że kawałek nie został jeszcze sprawdzony. NotSham oznacza, że kawałek został sprawdzony i nie jest shamem. Jeżeli kawałek został sprawdzony i był shamem, to wyliczenie przyjmuje wartość NotPicked: gracz traci kawałek.

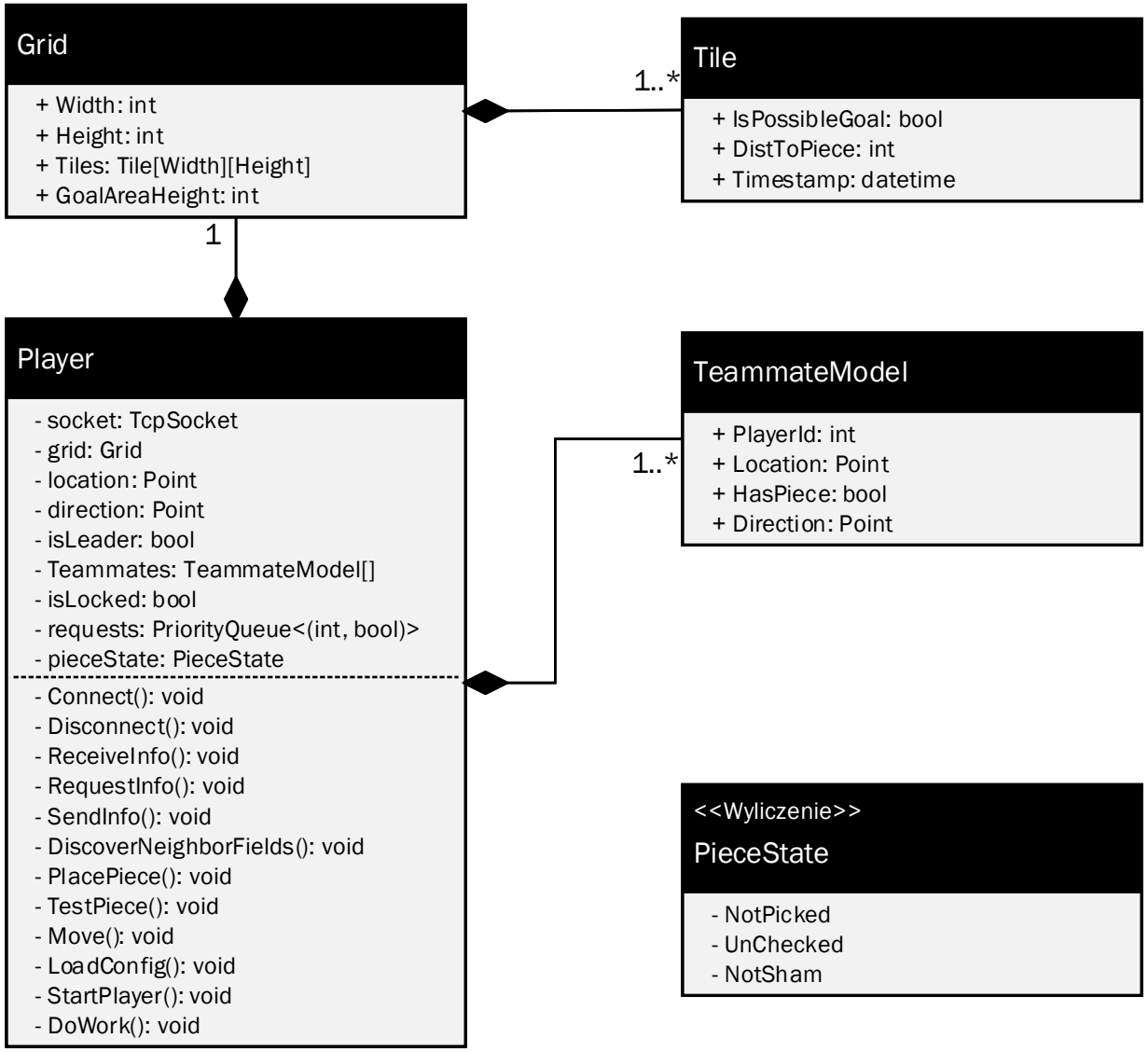
- **Location**: aktualne położenie gracza.
- **IsLeader**: czy gracz jest liderem.

Działanie *PLAYERA* zaczyna się od wywołania metody **StartPlayer()**. Ta metoda ma na celu inicjację oraz połączenie. Po pierwsze, wywoływany jest **LoadConfig()**, który ustawia startowe wartości pól gracza z pliku konfiguracyjnego: informację o drużynie oraz o adresie/porcie serwera komunikacyjnego. Sprawdzana jest poprawność konfiguracji. Następnie wywoływana jest funkcja **Connect()**, która łączy gracza z *COMMUNICATION SERVER* oraz ustawia socket na nieblokujący. Potem następuje oczekiwanie na wiadomość od serwera z danymi: wielkość siatki, położenie gracza, czy dany gracz jest liderem. Gdy *GAME MASTER* wyśle dane i rozpocznie rozgrywkę, gracz wywołuje metodę **DoWork()**, która jest główną pętlą działania gracza i określa jego strategię. **W ramach przyjętej strategii gracz będzie wykonywał odpowiednie działania w trakcie rozgrywki, tj. wywoływał metody:**

- **RequestInfo():** prośba/żądanie o informacje od innego gracza
- **SendInfo():** obsługuje jedno zapytanie z kolejki Requests
- **DiscoverNeighborFields():** odkrycie odległości przyległych TILE'ÓW od najbliższego *PIECE'A* w metryce Manhattan
- **PlacePiece():** położenie *PIECE'A*
- **TestPiece():** sprawdzenie, czy trzymany *PIECE* jest fałszywy
- **Move():** ruch w jednym z 4 kierunków

Dodatkowo, metoda **ReceiveInfo()** ma asynchronicznie obsługiwać otrzymywane przez gracza komunikaty: komunikaty od serwera natychmiast wpływają na stan gracza (przykładowo, budzą go lub modyfikują stan siatki), podobnie jak informacje o stanie wiedzy innego gracza; natomiast prośby o wymianę informacji od innych graczy są dodawane do kolejki priorytetowej Requests.

Istotnym jest, że każde działanie gracza wywołuje funkcję Sleep lub analogiczną oraz wysyła wiadomość do *GAME MASTERA* (przez *COMMUNICATION SERVER*). Tylko serwer może teraz obudzić gracza.


Grid
+ Width: int
+ Height: int
+ Tiles: Tile[Width][Height]
+ GoalAreaHeight: int
1..*
Tile
+ IsPossibleGoal: bool
+ DistToPiece: int
+ Timestamp: datetime
1
Player
- socket: TcpSocket
- grid: Grid
- location: Point
- direction: Point
- isLeader: bool
- Teammates: TeammateModel[]
- isLocked: bool
- requests: PriorityQueue<(int, bool)>
- pieceState: PieceState
- Connect(): void
- Disconnect(): void
- ReceiveInfo(): void
- RequestInfo(): void
- SendInfo(): void
- DiscoverNeighborFields(): void
- PlacePiece(): void
- TestPiece(): void
- Move(): void
- LoadConfig(): void
- StartPlayer(): void
- DoWork(): void
1..*
TeammateModel
+ PlayerId: int
+ Location: Point
+ HasPiece: bool
+ Direction: Point
<<Wyliczenie>>
PieceState
- NotPicked
- UnChecked
- NotSham

5.2 Communication Server

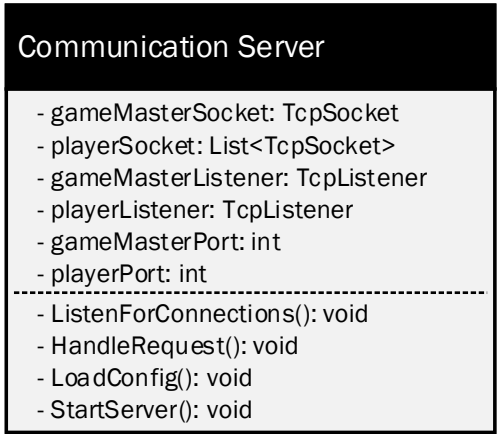
Moduł *COMMUNICATION SERVER* odpowiada za komunikację między graczami oraz między graczami a *GAME MASTEREM* (tak naprawdę komunikacja między graczami także odbywa się z użyciem *GAME MASTERA*). Polami serwera komunikacyjnego są:

- **GameMasterSocket**: socket do komunikacji z *GAME MASTEREM*.
- **PlayerSocket**: sockety do komunikacji z graczami.
- **GameMasterListener**: listener nasłuchujący połączenia od *GAME MASTERA*.
- **PlayeListener**: listener nasłuchujący połączenia od graczy.
- **GameMasterPort**: załadowany z pliku konfiguracyjnego port, na którym *CS* ma nasłuchiwać połączenia od *GAME MASTERA*.
- **PlayerPort**: załadowany z pliku konfiguracyjnego port, na którym *CS* ma nasłuchiwać połączenia od graczy.

Działanie *COMMUNICATION SERVER* rozpoczyna się od wywołania metody **StartServer()**. W ramach tej metody wywoływane jest **LoadConfig()**, które ładuje konfigurację oraz ustawia odpowiednie porty i inicjuje listenery. Następnie serwer wywołuje metodę **ListenForConnections()**, która nasłuchuje połączeń od graczy i serwera, następnie dodaje odpowiednie sockety.

Po dodaniu wszystkich połączeń i rozpoczęciu gry, serwer w nieskończonej pętli wywołuje **HandleRequest()**; metoda ta ma na celu obsługę żądań (tj. przesyłanie wiadomości na linii gracz $\longleftrightarrow$ serwer).

Wszelka komunikacja w grze odbywa się przy użyciu protokołu TCP oraz komunikatów w formacie JSON.


Communication Server
- gameMasterSocket: TcpSocket
- playerSocket: List<TcpSocket>
- gameMasterListener: TcpListener
- playerListener: TcpListener
- gameMasterPort: int
- playerPort: int
- ListenForConnections(): void
- HandleRequest(): void
- LoadConfig(): void
- StartServer(): void

5.3 Game Master

Moduł *GAME MASTER* opisuje mistrza gry, obiekt, które decyduje o warunkach zwycięstwa i posiada pełną informację o stanie gry. *GAME MASTER*, podobnie jak *PLAYER*, agreguje obiekt *GRID* o analogicznych właściwościach co w obiekcie *PLAYER*. Inną strukturę natomiast posiada obiekt *TILE*, który jest agregowany przez *GRID*. Pojedynczy *TILE* posiada poniższe pola:

- **IsGoal:** przyjmuje wartość true, jeśli dane pole jest w *GOAL* AREA i jest celem do zrealizowania. To, jakie pola będą celami jest ustawiane losowo na początku gry, a ilość celów jest ustawiana w konfiguracji. Jeśli cel zostanie zrealizowany, to IsGoal zmieni wartość na false.
- **Piece:** referencja na *PIECE*, który znajduje się na danym polu w *TASK AREA*. Jeżeli jakiś gracz podniesie dany *PIECE*, to ta referencja przenosi się do obiektu *PLAYER MODEL*.
- **IsOccupied:** przyjmuje wartość true, jeśli na danym polu znajduje się gracz. W przeciwnym wypadku jest to false.

PIECE jest w tym module obiektem, który może być agregowany albo przez *TILE*, albo przez *PLAYER MODEL*. Jako taki zawiera wyłącznie pole **IsShame** - informację o tym, czy jest shamem.

GAME MASTER agreguje także dwa obiekty typu *TEAM*: są to drużyny biorące udział w rozgrywce. Każda z dwóch drużyn posiada informację o graczach, którzy do niej przynależą (czyli drużyna agreguje kolekcję obiektów *PLAYER MODEL*), a także pozostałe cele, które drużyna musi zrealizować, w postaci kolekcji referencji na obiekty *TILE*. Metodą *GAME MASTERA* na sprawdzanie warunku zakończenia gry jest sprawdzenie, czy kolekcja pozostałych celów jest pusta.

Pojedynczy *PLAYER MODEL* zawiera wszystkie niezbędne informacje o graczu: jego pozycję (**Location**), drużynę (**TeamId**), posiadany *PIECE* (**Piece**), długość timeoutu **(SleepTimeout**) oraz informację o tym, czy jest liderem (**IsLeader**). Każda akcja gracza powoduje, że zostanie nałożony na niego timeout na określoną ilość milisekund. Jeśli timeout zejdzie do zera to *GAME MASTER* wysyła wiadomość budzącą do danego gracza. Jak było wspominane przy opisie *COMMUNICATION SERVER*, każda wymiana informacji między graczami musi przechodzić przez *GAME MASTERA*. *GAME MASTER* wykorzystuje informację IsLeader o danym graczu by określić, czy wysyłana prośba ma być jedynie prośbą, czy też żądaniem (i taka dodatkowa informacja trafia do odbiorcy).

Działanie *GAME MASTERA* rozpoczyna się od wywołania metody **StartGameMaster()**. W środku tej metody jest wywołanie metody **LoadConfig()**, która inicjuje z pliku wartości pól:

- **nPlayersInTeam**: liczba graczy w każdym zespole.
- **nPiecesOnGrid**: ile ma być niepodniesionych *PIECE'ÓW* na planszy.
- **timeoutValues**: określa długości timeoutów dla poszczególnych akcji.
- **shamChance**: dla każdego wygenerowanego *PIECE'A*: jakie jest prawdopodobieństwo, że będzie on shamem.

oraz wartości pól *GRIDA*:

- **Width**: szerokość planszy.

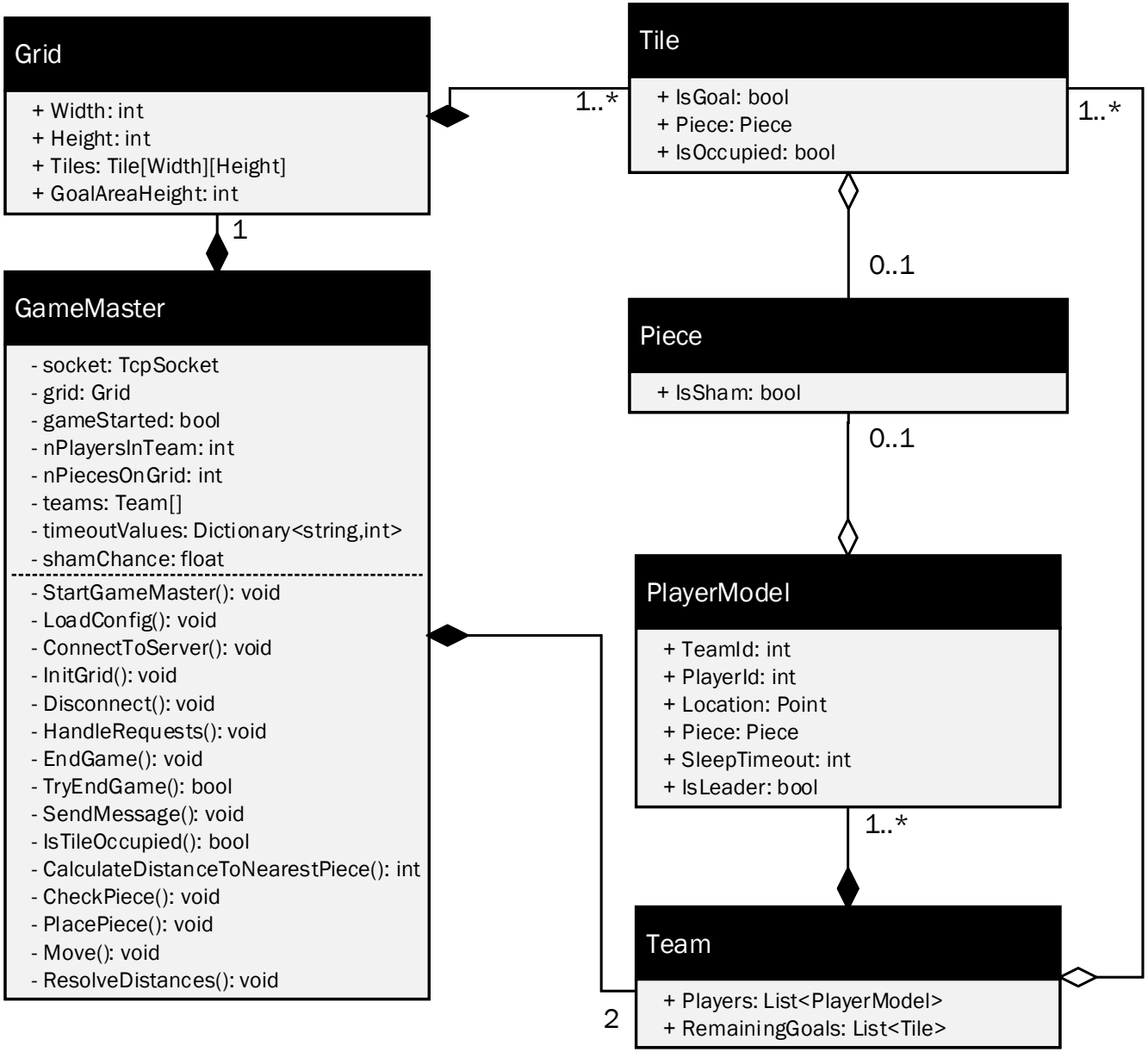
- **Height**: wysokość planszy.
- **GoalAreaHeight**: wysokość Goal Area.

Konfiguracja zawiera też informacje o połączeniu z *COMMUNICATION SERVER*. W ramach tej metody sprawdzana jest poprawność konfiguracji. Kolejnym krokiem jest wywołanie metody **ConnectToServer()**, która łączy *GAME MASTERA* z *COMMUNICATION SERVER*. *GAME MASTER* teraz czeka dopóki obie drużyny nie będą wypełnione, po czym wywołuje **InitGrid()**, która to metoda ma na celu zainicjowanie siatki gry (pozycje celów, pozycje graczy, pozycje *PIECE'ÓW*). Potrzebne informacje, wraz z informacją o rozpoczęciu gry, wysyłane są do graczy.

Po wstępnej konfiguracji i rozpoczęciu gry, *GAME MASTER* przechodzi w tryb obsługi gry. Wywoływana jest metoda HandleRequests(), która w kółko czeka na prośby graczy i je przetwarza. Metoda ta posiada kolejkę komunikatów. Co każdy tick gry przetwarzane są wszystkie komunikaty i aktualizowane są dane (np. długości timeoutów graczy). Potem *GAME MASTER* wyłącznie dodaje ewentualne prośby do kolejki i czeka na kolejny tick.

GAME MASTER, w ramach metody HandleRequests(), będzie brał kolejne komunikaty i je przetwarzał. W zależności od typu komunikatu wywoływana będzie odpowiednia metoda do ich obsługi. Na końcu każdego zapytania *GAME MASTER* wysyła do adresata odpowiednią wiadomość, wywołując metodę **SendMessage()**. Przed obsługą requestu sprawdzane jest, czy dany gracz nie jest zablokowany. Jeśli jest, to otrzymuje informację że jest zablokowany. Poniżej omówione są metody odpowiadające za obsługę poszczególnych zapytań:

- **CheckPiece():** jeśli gracz trzyma kawałek, to dostanie informację czy trzymany kawałek jest shamem. Jeśli jest, to automatycznie ten kawałek znika. Jeśli gracz nie trzyma kawałka, to dostanie jedynie informację o nielegalnej akcji.
- **PlacePiece():** wywoływane jeśli gracz chce postawić kawałek. Obsługa jest zgodna z regułami gry. Każde wywołanie tej metody powoduje wywołanie metody TryEndGame(), która sprawdza spełnienie warunków zakończenia gry. Jeśli metoda zwróci true, czyli drużyna postawiła ostatni *PIECE*, to gra się kończy: wywoływana jest metoda EndGame(), wyjście z metody HandleRequests(), wysyłane są komunikaty o zakończeniu do wszystkich graczy i na końcu rozłączenie Disconnect().
- **Move():** metoda ta wewnętrznie wykorzystuje IsTileOccupied() do sprawdzenia, czy pole do którego zmierza gracz nie jest zajęte. Jeśli jest, to wysyłana jest odpowiednia wiadomość, a jeśli nie jest, to wywoływana jest metoda CalculateDistanceToNearestPiece() i graczowi wysyłana jest wiadomość o sukcesie i o tym, jaka jest odległość do najbliższego *PIECE'A* z jego nowej pozycji.
- **ResolveDistances():** metoda ta służy do obsługi żądania o odkrycie odległości przylegających do danego gracza pól od najbliższego *PIECE'A*. Wewnętrznie metoda ta wywołuje osiem razy CalculateDistanceToNearestPiece() i wysyła odpowiedni komunikat.


Grid
+ Width: int
+ Height: int
+ Tiles: Tile[Width][Height]
+ GoalAreaHeight: int
1
1..*
Tile
+ IsGoal: bool
+ Piece: Piece
+ IsOccupied: bool
1..*
0..1
Piece
+ IsSham: bool
0..1
GameMaster
- socket: TcpSocket
- grid: Grid
- gameStarted: bool
- nPlayersInTeam: int
- nPiecesOnGrid: int
- teams: Team[]
- timeoutValues: Dictionary<string,int>
- shamChance: float
- StartGameMaster(): void
- LoadConfig(): void
- ConnectToServer(): void
- InitGrid(): void
- Disconnect(): void
- HandleRequests(): void
- EndGame(): void
- TryEndGame(): bool
- SendMessage(): void
- IsTileOccupied(): bool
- CalculateDistanceToNearestPiece(): int
- CheckPiece(): void
- PlacePiece(): void
- Move(): void
- ResolveDistances(): void
PlayerModel
+ TeamId: int
+ PlayerId: int
+ Location: Point
+ Piece: Piece
+ SleepTimeout: int
+ IsLeader: bool
1..*
2
Team
+ Players: List<PlayerModel>
+ RemainingGoals: List<Tile>

6 Diagramy stanów

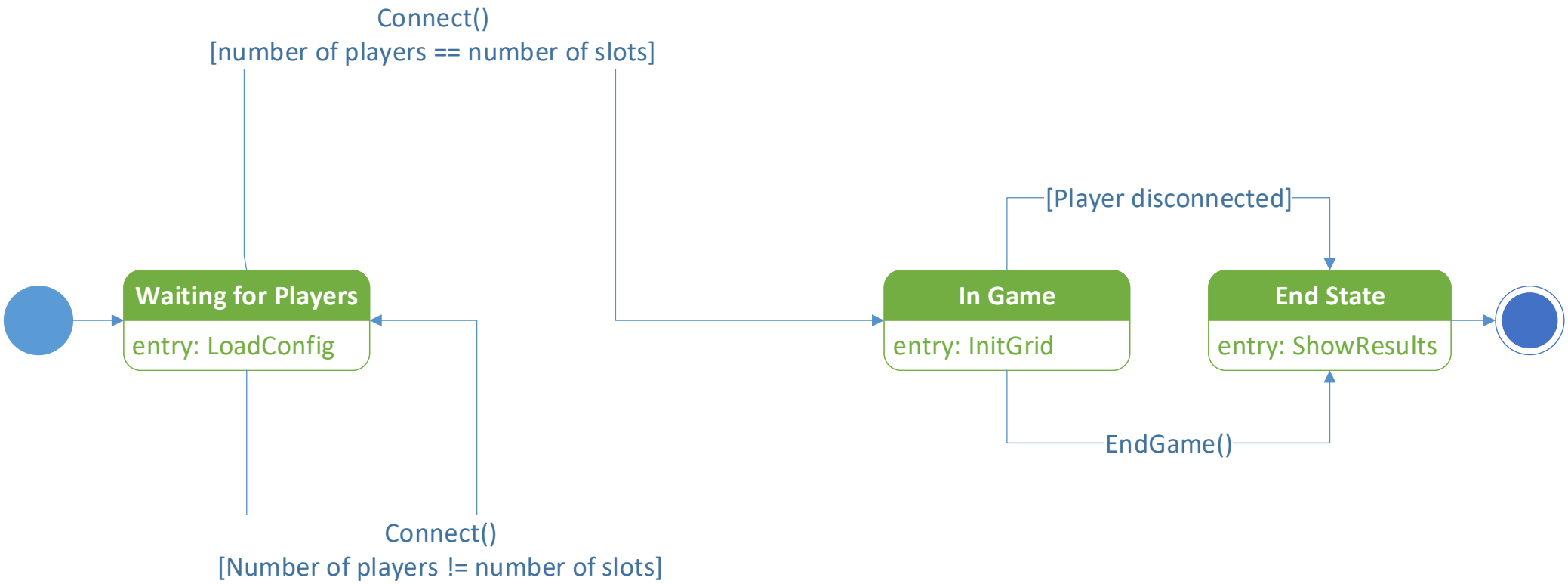
6.1 Game

Po uruchomieniu gra wczytuje plik konfiguracyjny i przechodzi w stan oczekiwania na graczy (*Waiting for Players*). Gra pozostaje w tym stanie, dopóki nie połączy się odpowiednia liczba graczy, określona wcześniej w pliku konfiguracyjnym.

Gdy wszyscy gracze nawiążą połączenie, gra przechodzi w stan *In Game*. Po wejściu do tego stanu zainicjowana zostaje plansza. W stanie *In Game* toczy się główna część gry.

Jeżeli Game Master utraci kontakt z jednym z graczy, lub osiągnięty zostanie warunek końca gry, gra przechodzi w stan *End State*. W tym stanie wyświetlone zostają wyniki gry, po czym gra się kończy.

Game


Connect()
[number of players == number of slots]
[Player disconnected]
Waiting for Players
entry: LoadConfig
In Game
entry: InitGrid
End State
entry: ShowResults
EndGame()
Connect()
[Number of players != number of slots]

6.2 Piece (GM)

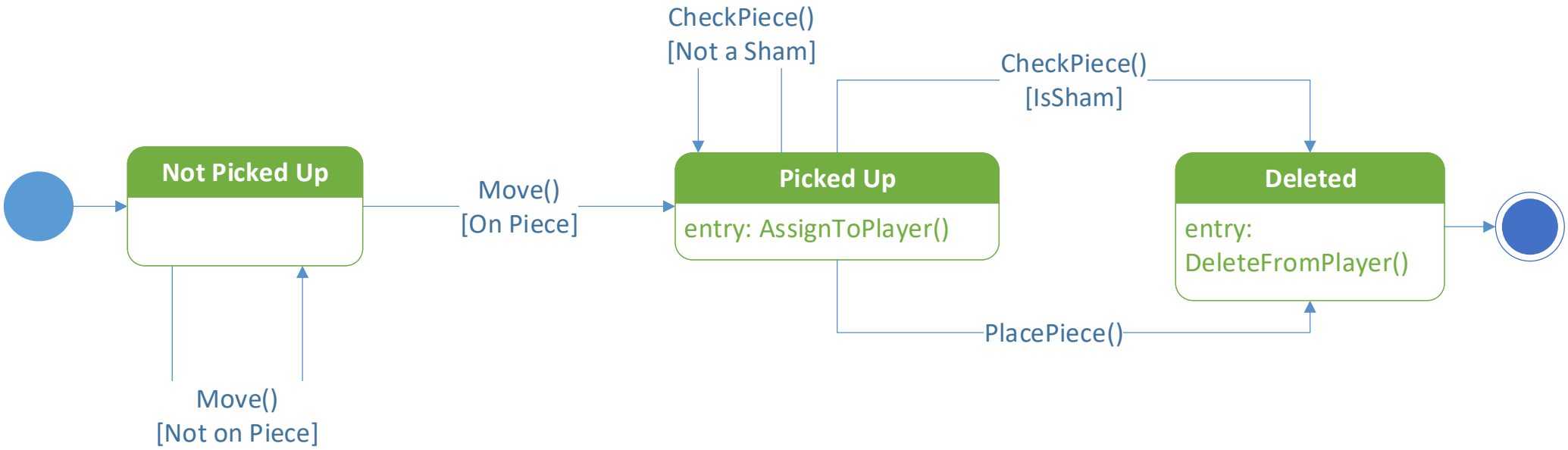
Piece z perspektywy Game Mastera zaczyna w stanie *Not Picked Up*. Pozostaje w tym stanie, dopóki żaden z graczy go nie podniesie.

Jeżeli któryś z graczy wykona ruch na pole, na którym znajduje się Piece, zostaje on automatycznie podniesiony i przechodzi w stan *Picked Up*.

W stanie *Picked Up* gracz może sprawdzić prawdziwość Piece. Jeżeli Piece nie jest Shamem, pozostaje w tym samym stanie. Jeżeli Piece jest Shamem, przechodzi w stan *Deleted*.

Jeżeli gracz położy swój Piece, przechodzi on w stan *Deleted*. Dzieje się tak niezależnie, czy Piece jest Shamem, i niezależnie, czy Tile, na którego zostało położone Piece, jest Goalem. Po wejściu do tego stanu Piece zostaje usunięty z ekwipunku gracza i z gry.

Piece (GM)


CheckPiece()
[Not a Sham]
CheckPiece()
[IsSham]
Not Picked Up
Move()
[On Piece]
Picked Up
entry: AssignToPlayer()
Deleted
entry:
DeleteFromPlayer()
PlacePiece()
Move()
[Not on Piece]

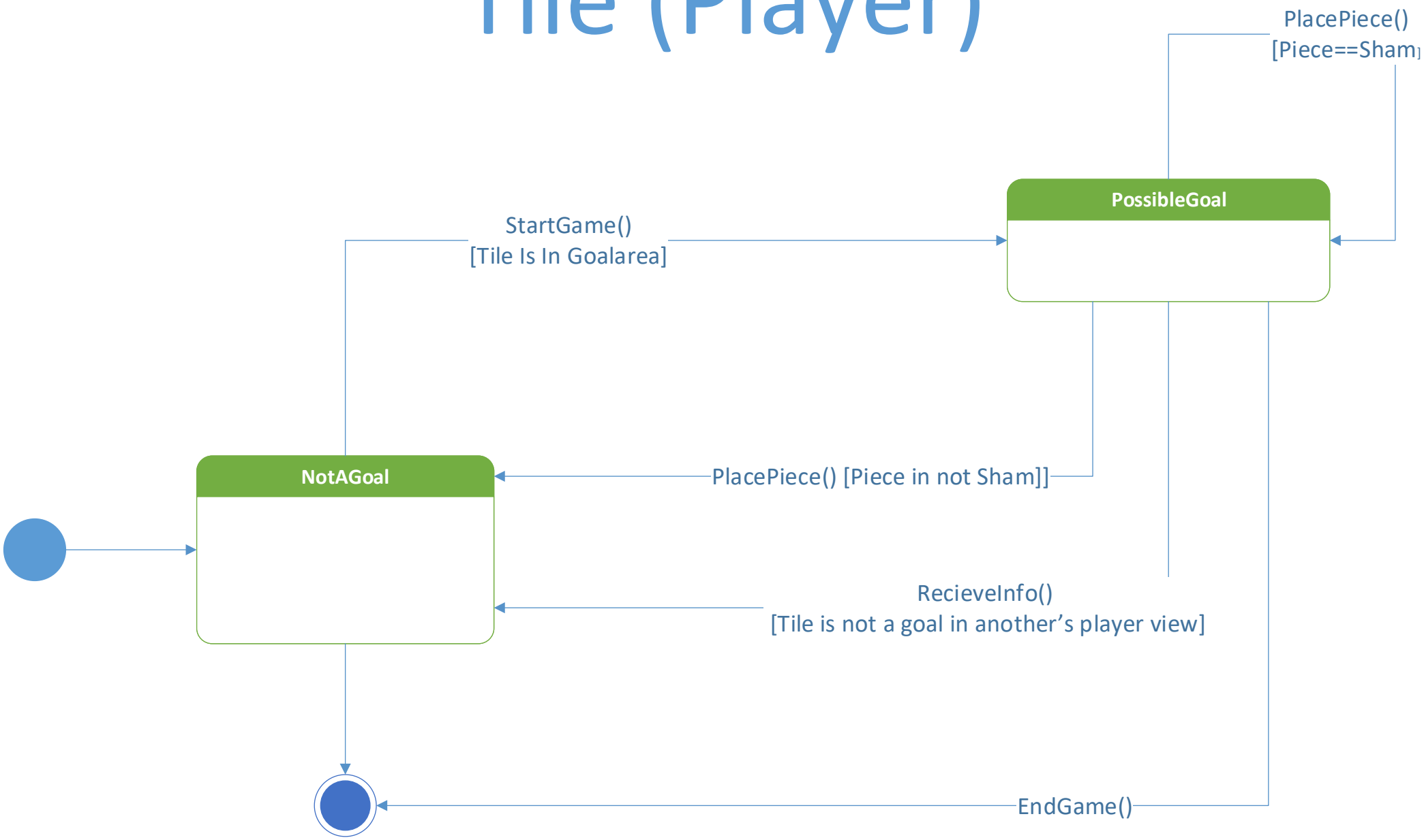
6.3 Tile (Player)

Każdy Tile z perspektywy gracza zaczyna w stanie *NotAGoal*. Po rozpoczęciu gry, gdy gracz otrzyma informacje o położeniu Goal Area, każdy Tile w obrębie Goal Area drużyny danego gracza przechodzi w stan *Possible Goal*.

Jeżeli gracz położy na Tile Piece, który nie był Shamem, Tile wraca do stanu *NotAGoal*. Dzieje się tak niezależnie, czy dany Tile faktycznie był Goalem. Jeśli gracz otrzymuje od innego gracza informację, że dany Tile jest u niego w stanie *NotAGoal*, Tile u gracza otrzymującego informację także przechodzi w ten stan.

Jeżeli Piece położony na Tile był Shamem, Tile nie zmienia swojego stanu.

Tile (Player)


PlacePiece()
[Piece==Sham]
PossibleGoal
StartGame()
[Tile Is In Goalarea]
NotAGoal
PlacePiece() [Piece in not Sham]]
RecieveInfo()
[Tile is not a goal in another's player view]
EndGame()

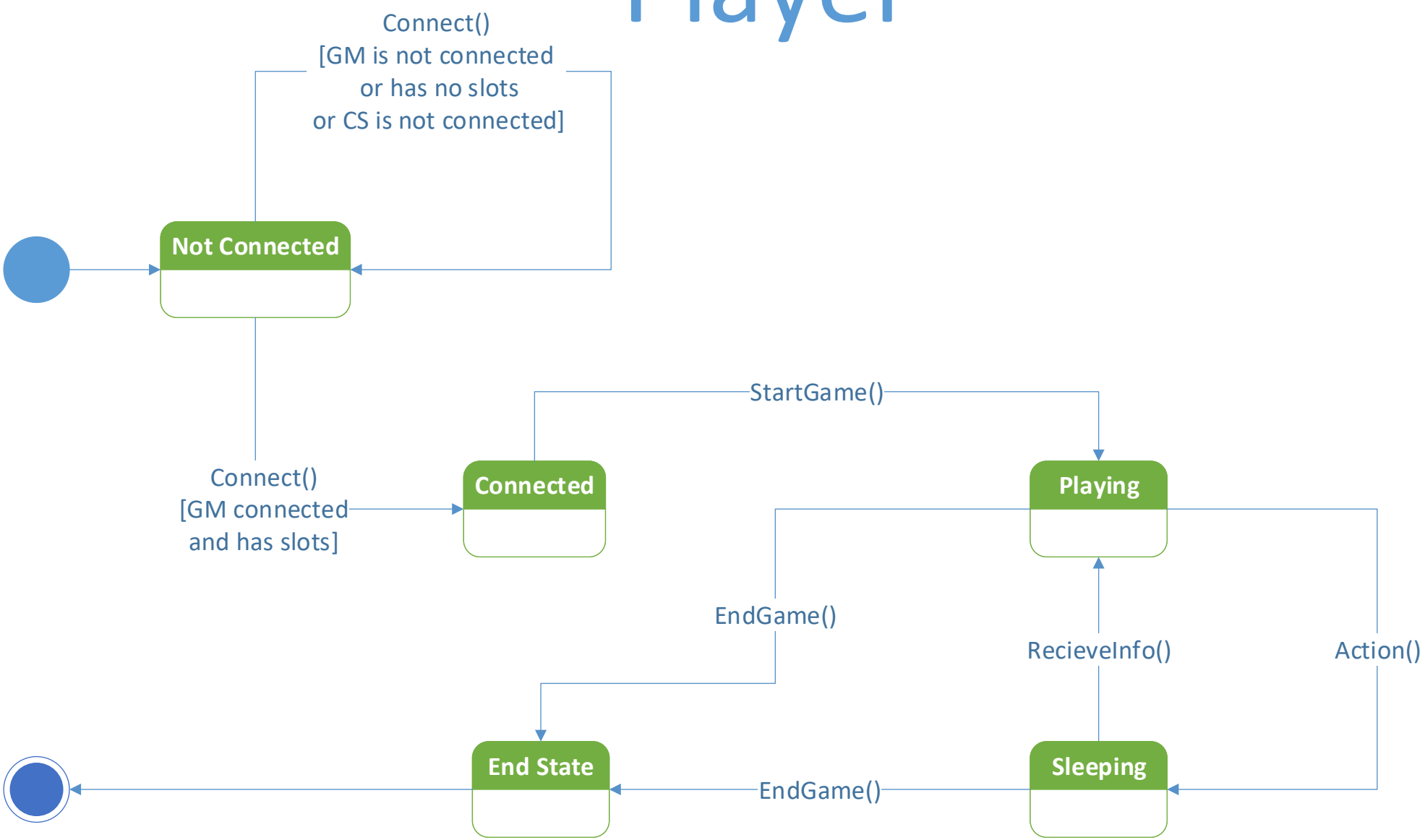
6.4 Player

Player zaczyna w stanie *Not Connected*. Jeżeli uda mu się połączyć do gry, przechodzi w stan *Connected*. Po rozpoczęciu gry Player przechodzi w stan *Playing*. Po podjęciu dowolnej akcji gracz przechodzi w stan *Sleeping*. Pozostaje w tym stanie dopóki nie zostanie wybudzony przez Game Mastera. Wówczas wraca do stanu *Playing* i może podjąć kolejną akcję.

Jeżeli gra zakończy się z dowolnego powodu, gracz przechodzi ze stanu *Playing* lub *Sleeping* do stanu *End State*.

- Player po podjęciu dowolnej akcji zawsze przechodzi w stan *Sleeping*, nawet, jeżeli request był niepoprawny.
- Game Master odpowiada za wybudzanie graczy. Jeżeli wysłany przez Playera request był niepoprawny, Player zostaje wybudzony natychmiast.

Player

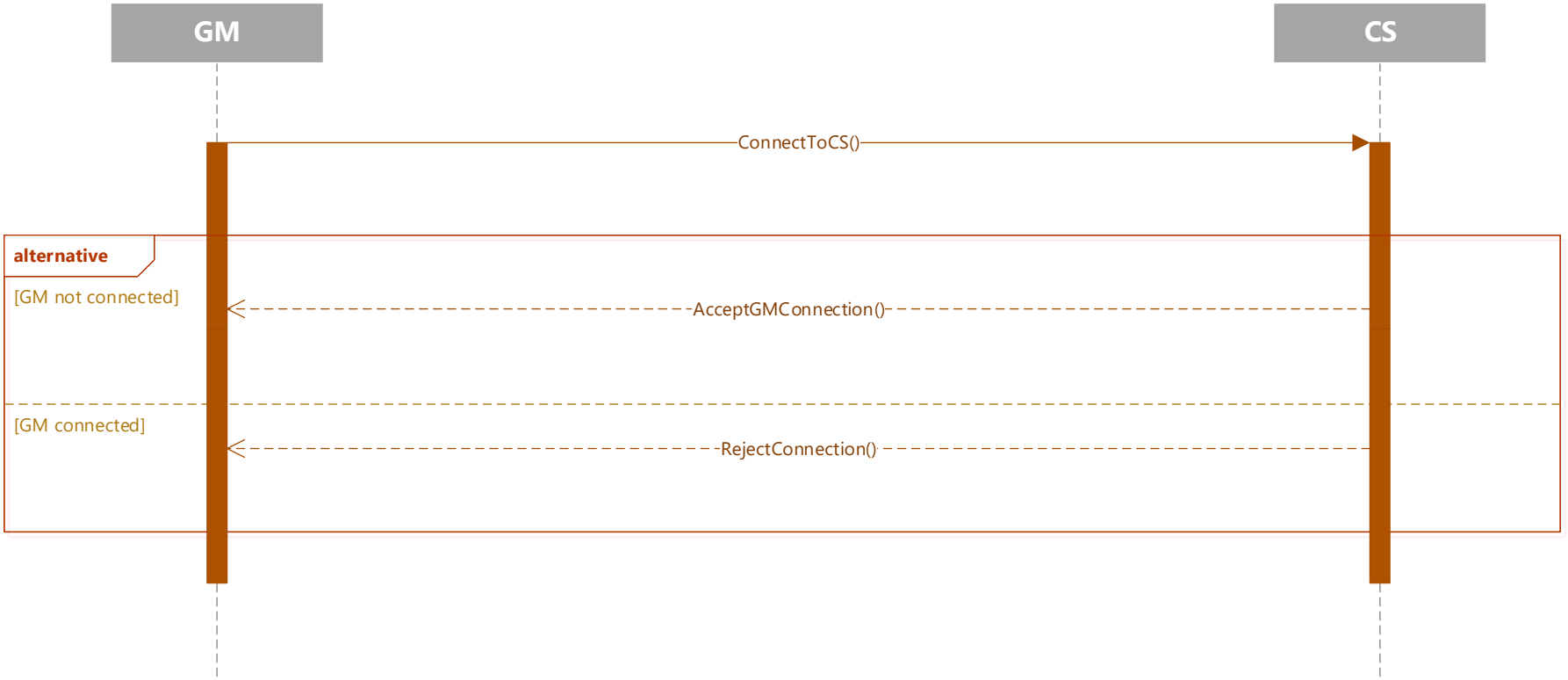

Connect()
[GM is not connected
or has no slots
or CS is not connected]
Not Connected
Connect()
[GM connected
and has slots]
Connected
StartGame()
Playing
EndGame()
RecieveInfo()
Action()
End State
Sleeping
EndGame()

7 Diagramy sekwencji

7.1 Connect GM-CS

GameMaster wysyła do CommunicationServera (CS) prośbę o połączenie (ConnectToCS). Jeżeli inny GameMaster jest już połączczony z CS, to połączenie zostaje odrzucone (RejectConnection). W innym wypadku połączenie jest akceptowane (AcceptGMConnection). Nie należy mylić z wiadomością AcceptConnection, ponieważ ta odnosi się do połączenia między graczem a GameMasterem i posiada pole ID.

Connect GM-CS

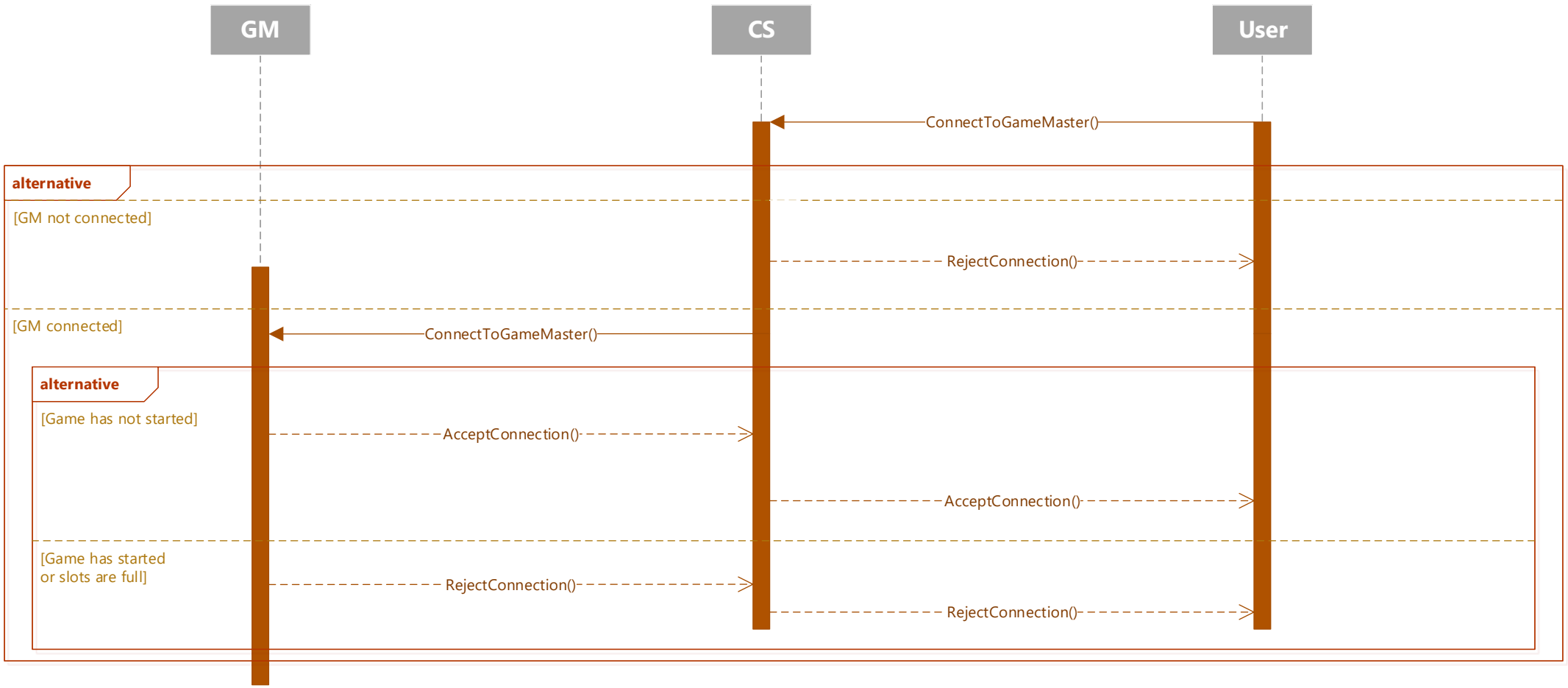


7.2 Connect GM-USER

Diagram ten modeluje połączenie nowego gracza do GameMastera. Gracz na początku deleguje żądanie ConnectToGameMaster do CommunicationServera, który sprawdza, czy jest już połączony GameMaster. Jeśli nie, to CS odrzuca żądanie poprzez RejectConnection.

Jeżeli GM jest podłączony do CSa, ten przesyła dalej żądanie o połączeniu. Prośba o podłączenie zostanie zaakceptowana tylko wtedy, gdy gra się jeszcze nie rozpoczęła oraz są wolne miejsca w drużynie, do której gracz chce się połączyć. W tym przypadku GM wysyła komunikat AcceptConnection. W tej wiadomości wysyłane jest graczowi jego ID. Jeżeli gracz nie może się połączyć, wysyłany jest komunikat RejectConnection.

Connect GM-User



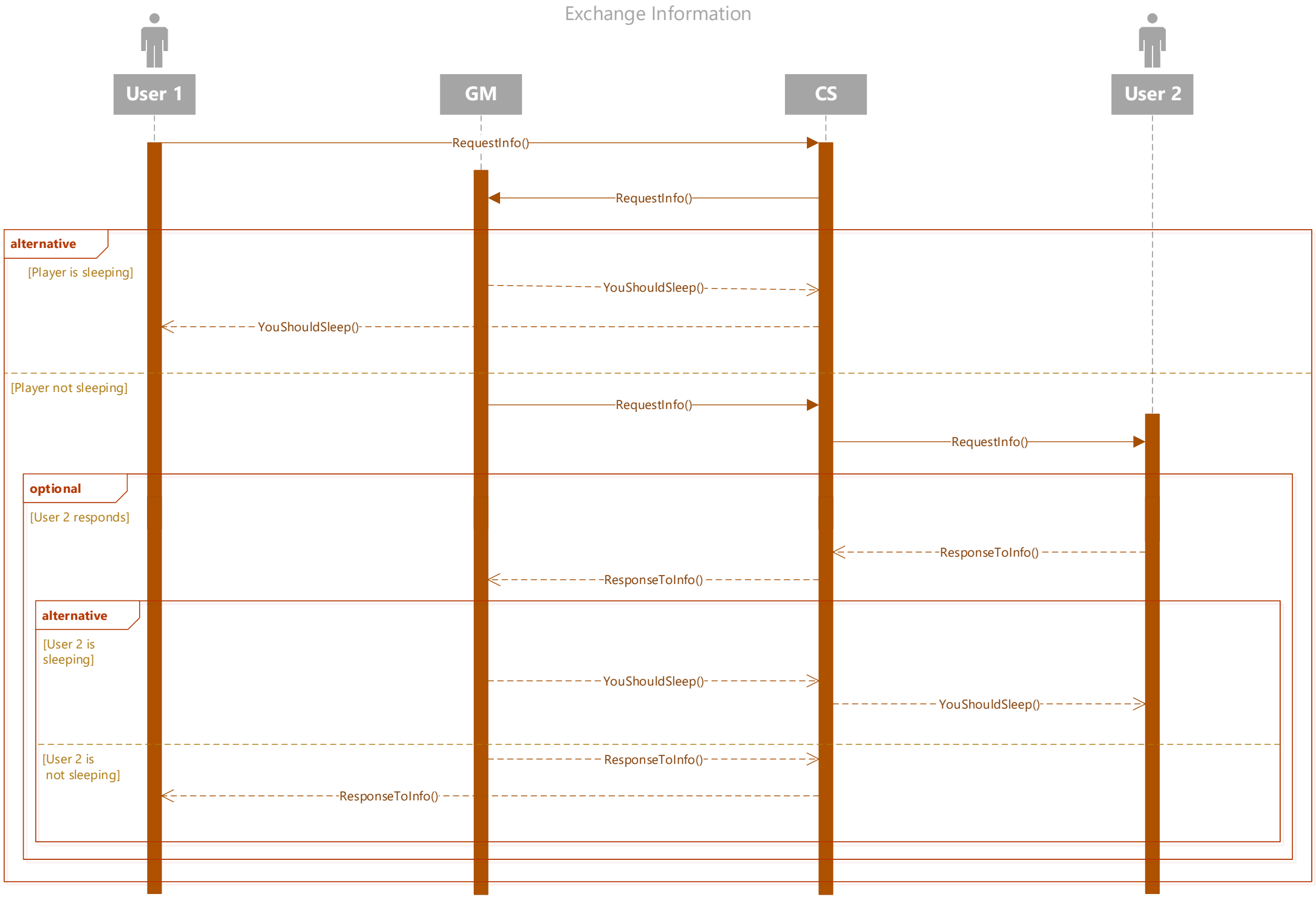
7.3 Exchange Information

Diagram ten modeluje wymianę informacji o stanie planszy między dwoma graczami (Userami). User1 rozpoczyna wymianę informacji poprzez wysłanie zapytania RequestInfo. Aby uprościć diagram, nie jest uwzględnione sprawdzanie przez serwer komunikacyjny czy jest podłączony GameMaster. Implicite takie sprawdzanie zachodzi i jeśli nie ma żadnego GameMastera, zostaje wysłana odpowiedź RejectConnection.

Gdy zapytanie dojdzie już do GameMastera, ten sprawdza, czy pytający gracz nie powinien spać. Jeśli powinien, GameMaster odrzuca jego zapytanie. W innym wypadku GameMaster modyfikuje treść zapytania: dodaje wartość na pole IdFrom (Id gracza pytającego), oraz na pole ObligatoryResponse (GM sprawdza, czy pytający jest liderem).

Gdy zapytanie dojdzie do adresata, ten decyduje, czy chce odpowiedzieć (teraz lub później; jeśli zapytanie było Obligatory to musi). Jeśli odpowiada, wysyła wiadomość rodzaju ResponseToInfo (ze stanem swojej planszy) przez CS do GM.

Ostatnim krokiem jest sprawdzanie przez GameMastera czy gracz wysyłający odpowiedź śpi. Jeśli śpi, GM wysyła do niego wiadomość YouShouldSleep. W innym wypadku odpowiedź przesyłana jest z powrotem przez CS do gracza pytającego (User1).

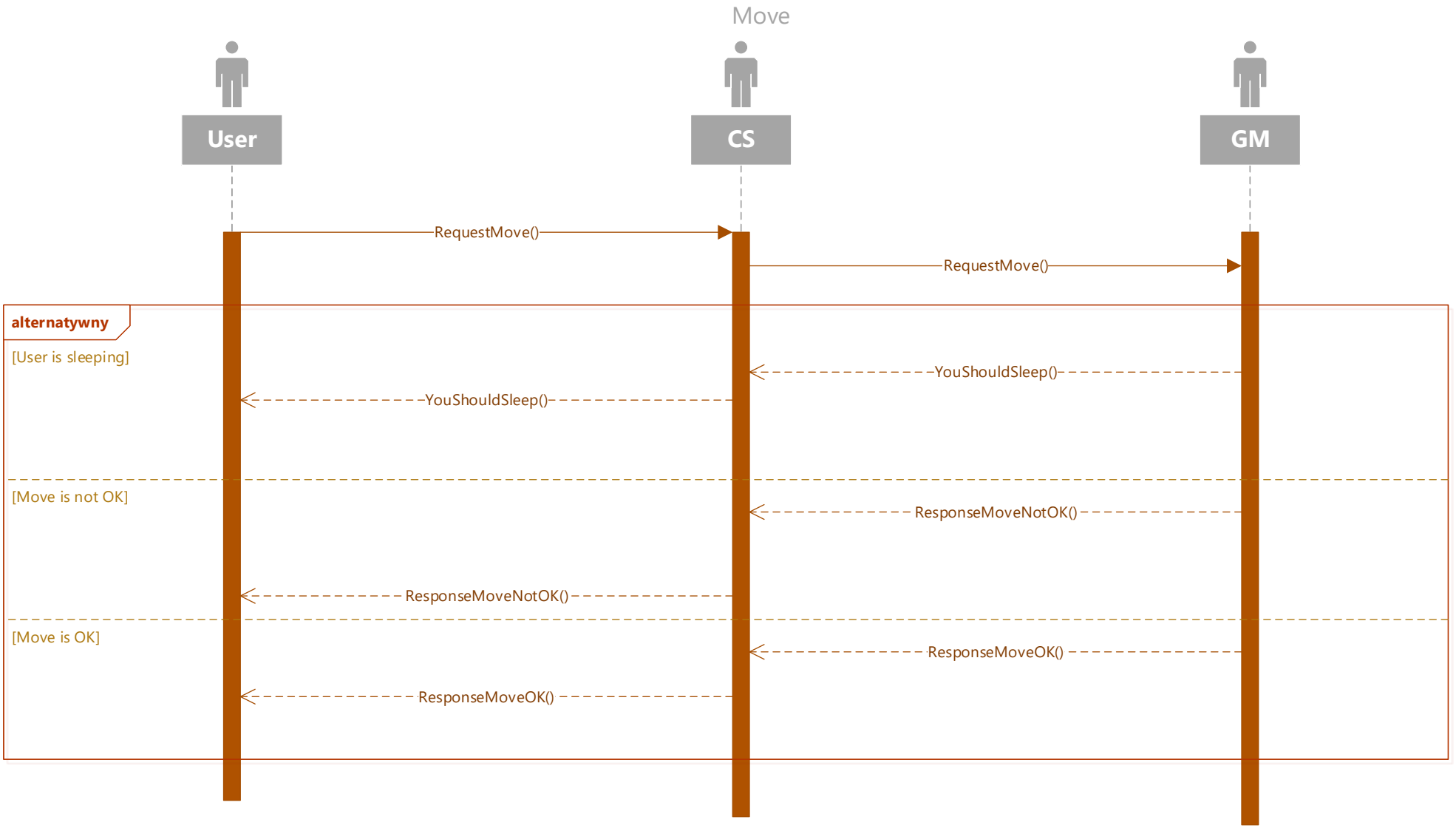

Exchange Information
User 1
GM
CS
User 2
RequestInfo()
RequestInfo()
alternative
[Player is sleeping]
YouShouldSleep()
YouShouldSleep()
[Player not sleeping]
RequestInfo()
RequestInfo()
optional
[User 2 responds]
ResponseToInfo()
ResponseToInfo()
alternative
[User 2 is sleeping]
YouShouldSleep()
YouShouldSleep()
[User 2 is not sleeping]
ResponseToInfo()
ResponseToInfo()

7.4 Move

Diagram ten modeluje żądanie ruchu przez gracza. Na początku gracz (User) wysyła żądanie ruchu RequestMove do GM przez CS. Podobnie jak w diagramie ExchangeInformation, sprawdzanie, czy GM jest podłączony do CS, zachodzi implicite i nie jest uwzględnione w diagramie.

Gdy żądanie dotrze do GameMastera, to ten będzie sprawdzał kilka warunków:

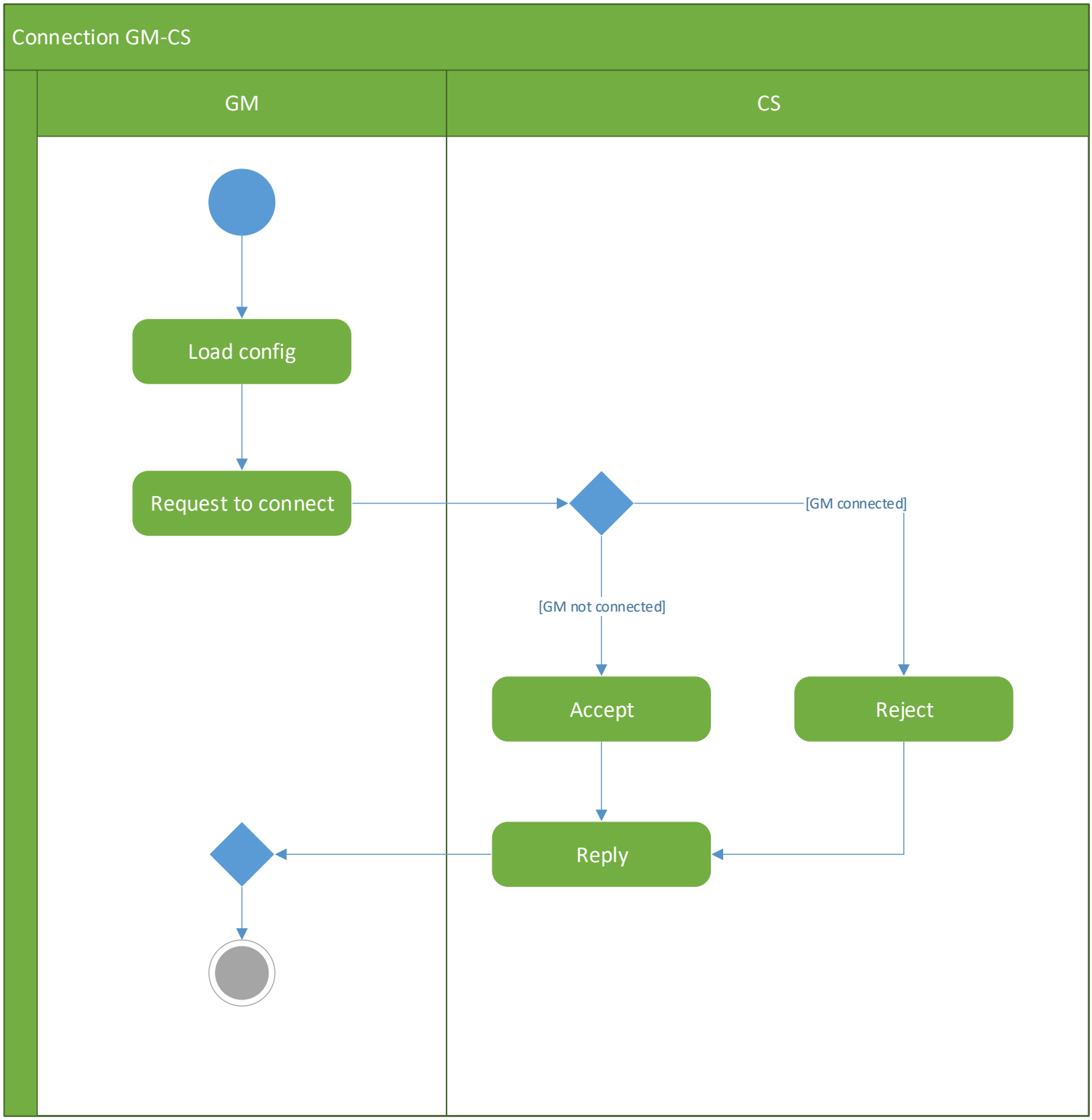
- Jeżeli gracz powinien spać, GM wysyła YouShouldSleep do gracza proszącego gracza
- Jeżeli ruch nie jest dozwolony (cel poza zasięgiem, zajęty, poza planszą), GM wysyła RequestMoveNotOk
- Jeżeli ruch jest dozwolony, GM wysyła do gracza wiadomość ResponseMoveOk, dołączając informację o tym, czy gracz podniósł kawałek i jaka jest odległość w metryce Manhattan kawałka do najbliższego leżącego kawałka


Move
User
CS
GM
RequestMove()
RequestMove()
alternatywny
[User is sleeping]
YouShouldSleep()
YouShouldSleep()
[Move is not OK]
ResponseMoveNotOK()
ResponseMoveNotOK()
[Move is OK]
ResponseMoveOK()
ResponseMoveOK()

8 Diagramy aktywności

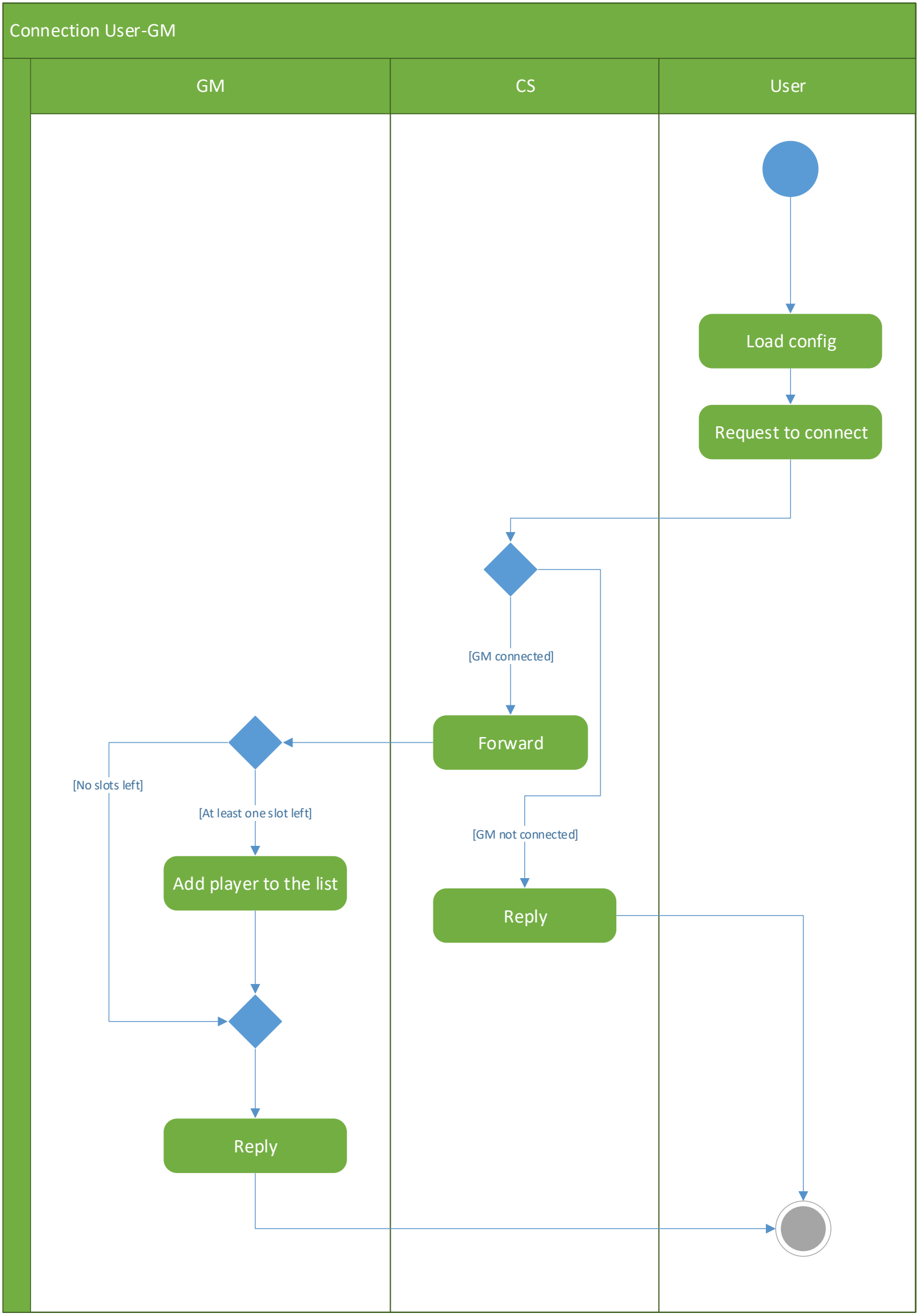
8.1 Connection GM-CS

Pierwszym krokiem jest wczytanie konfiguracji przez Game Mastera - m.in. adres IP i port Connection Servera. Następnie Game Master próbuje połączyć się do serwera. Jeśli do serwera jest już podłączony innym Game Master połączenie jest odrzucane. W przeciwnym wypadku połączenie jest odrzucane i Game Master jest podłączony do Connection Servera.


Connection GM-CS
GM
CS
Load config
Request to connect
[GM connected]
[GM not connected]
Accept
Reject
Reply

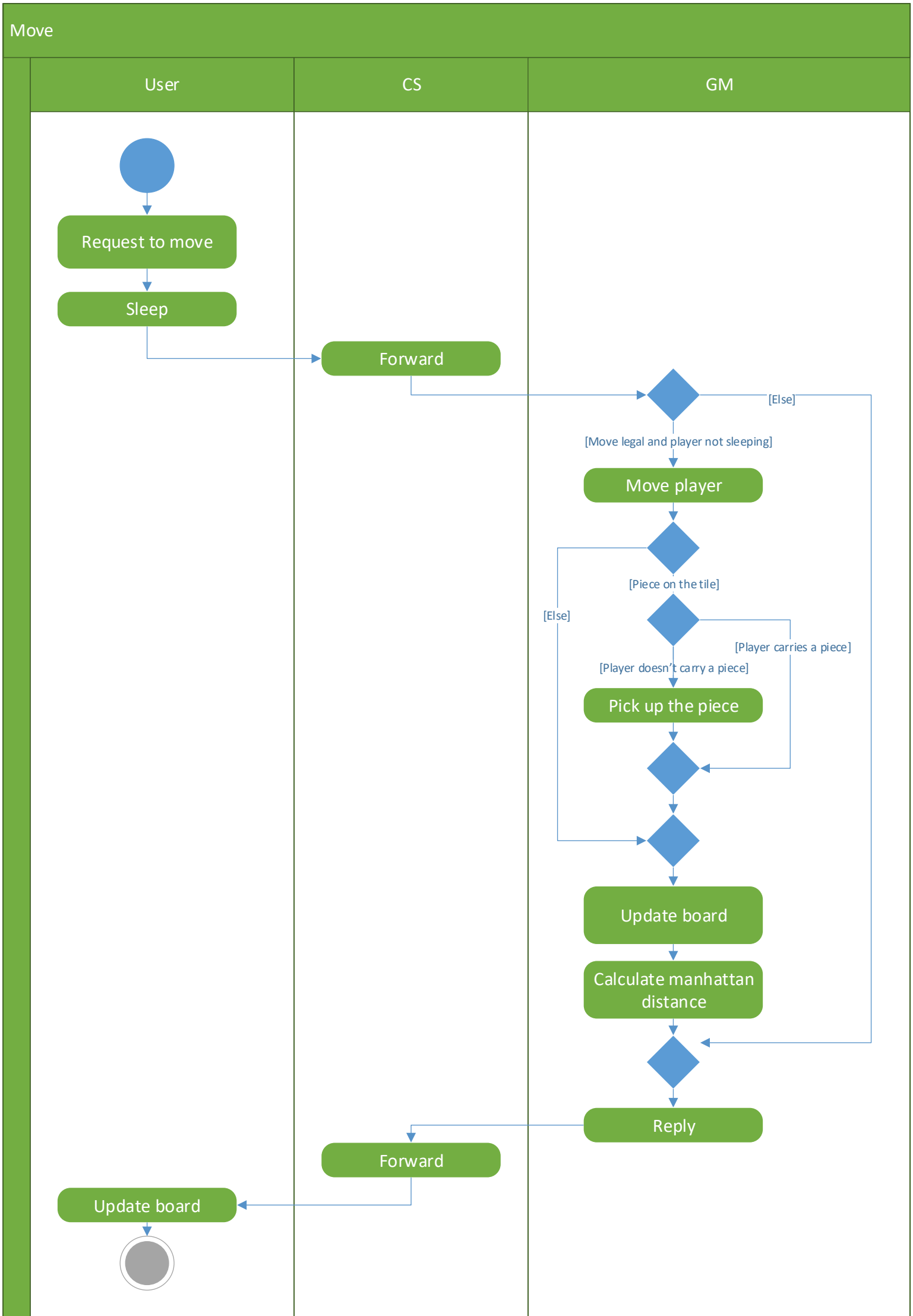
8.2 Connection User GM

Pierwszym krokiem jest wczytanie konfiguracji przez agenta - m.in. adres IP i port Connection Servera. Następnie User wysyła komunikat do Connection Servera z prośbą o podłączenie. Jeśli do serwera nie jest jeszcze podłączony Game Master połączenie jest odrzucane. W przeciwnym wypadku prośba przekazywana jest do Game Mastera, który dodaje gracza do listy, jeśli są jeszcze wolne miejsca. Na końcu Game Master wysyła wiadomość do Usera z odpowiedzią, czy ten został przyjęty.


Connection User-GM
GM
CS
User
Load config
Request to connect
[GM connected]
Forward
[No slots left]
[At least one slot left]
[GM not connected]
Add player to the list
Reply
Reply

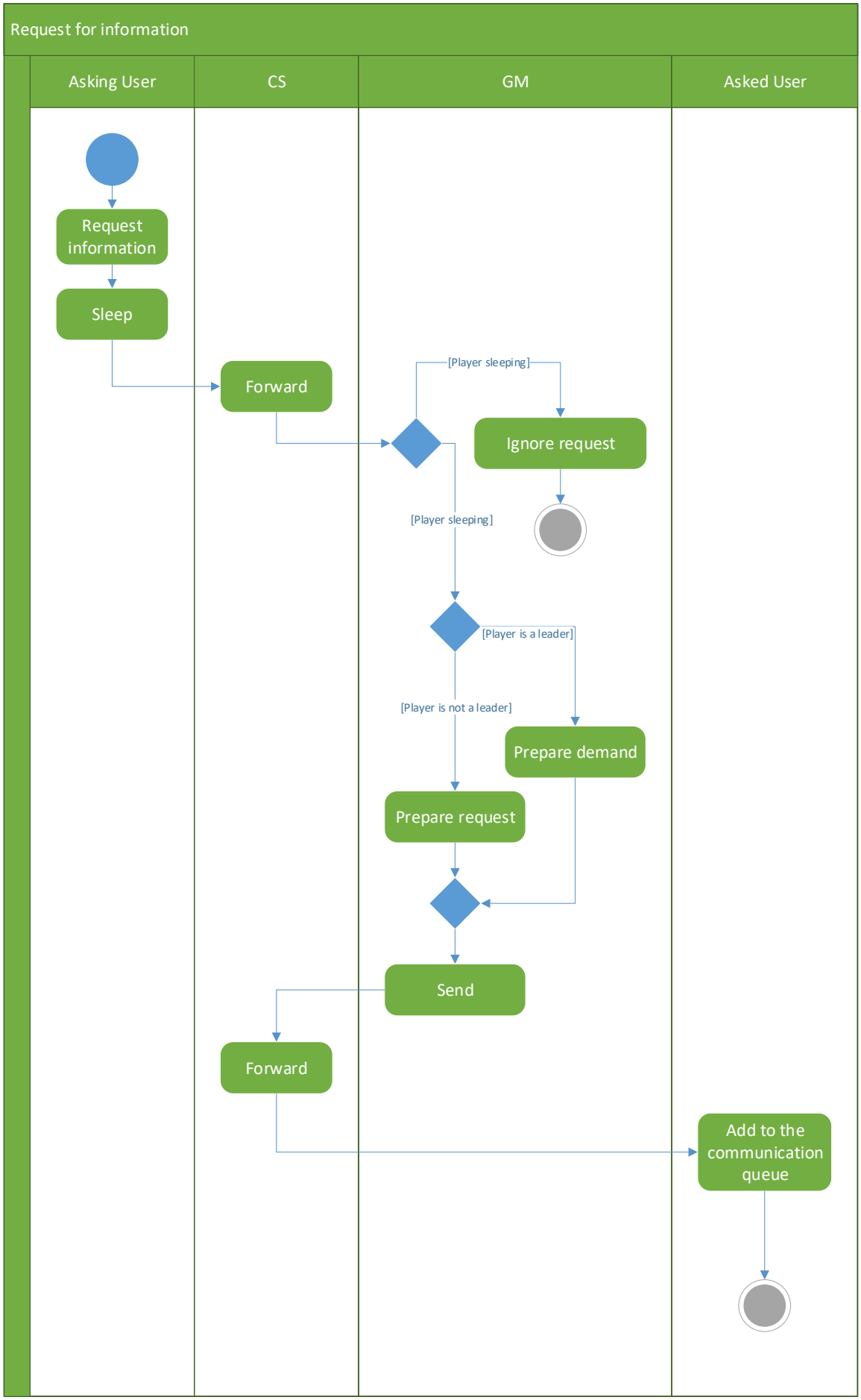
8.3 Move

W czasie trwania gry agent może poprosić Game Mastera o wykonanie ruchu, po czym przechodzi do stanu snu. Serwer przekazuje prośbę do Game Mastera. Game Master sprawdza legalność akcji - czy gracz przespał już odpowiednią ilość czasu od poprzedniego ruchu i czy danych ruch może być wykonany (czy gracz nie wyjdzie poza planszę, czy dane pole nie jest już zajęte). Jeśli ruch jest legalny Game Master przemieszcza gracza. W przeciwnym wypadku wysyłana jest odpowiedź o odrzuceniu ruchu. Po przesunięciu gracza Game Master sprawdza obecność kawałka na danym polu. Jeśli taki jest i gracz nie miał w danej chwili innego kawałka to Gamer Master przypisuje kawałek do gracza. Następnie plansza jest aktualizowana, obliczana jest odległość do najbliższego kawałka w metryce miejskiej i przez Communication Server przekazywana jest odpowiednia odpowiedź do Usera, który aktualizuje swój widok planszy.


Move
User
CS
GM
Request to move
Sleep
Forward
[Else]
[Move legal and player not sleeping]
Move player
[Piece on the tile]
[Else]
[Player carries a piece]
[Player doesn't carry a piece]
Pick up the piece
Update board
Calculate manhattan distance
Reply
Forward
Update board

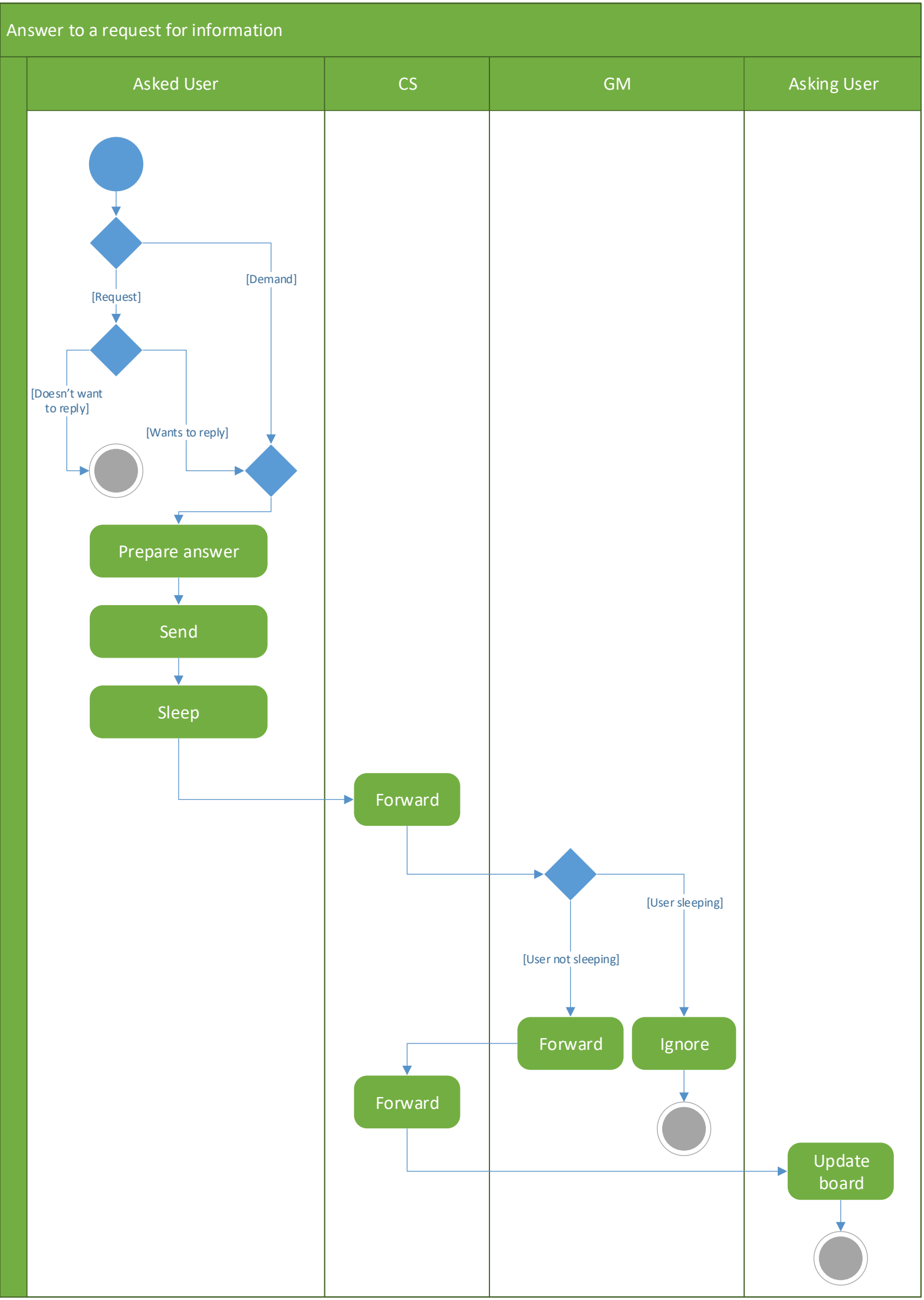
8.4 Request for information

W czasie trwania gry User może poprosić innego gracza z tej samej drużyny o informacje o planszy z jego punktu widzenia. Po wysłaniu zapytania User idzie spać. Communication Server przekazuje zapytanie do GameMastera, który sprawdza, czy dany gracz mógł zadać to pytanie. Jeśli nie odczekał odpowiedniej ilości czasu od poprzedniej aktywności prośba jest ignorowania. W przeciwnym wypadku przygotowuje prośbę lub żądanie w zależności od tego czy User pytający jest odpowiednio zwykłym graczem czy liderem. Wiadomości jest wysyłana do żądanego gracza, który dodaje pytanie do kolejki komunikatów.


Request for information
Asking User
CS
GM
Asked User
Request information
Sleep
Forward
[Player sleeping]
Ignore request
[Player sleeping]
[Player is a leader]
[Player is not a leader]
Prepare demand
Prepare request
Send
Forward
Add to the communication queue

8.5 Answer to a request for information

Po tym jak User dostał pytanie o informacje sprawdza czy User pytający jest liderem - w tym przypadku musi odpowiedzieć. Jeśli nie jest, User decyduje, czy chce odpowiedzieć na pytanie. W przypadku odpowiedzi User przygotowuje wiadomość, po czym ją wysyła i idzie spać. Communication Server przekazuje komunikat do Game Mastera, który sprawdza czy gracz odczekał odpowiednią ilość czasu od poprzedniej aktywności. Jeśli nie, wiadomość jest ignorowana. W przeciwnym wypadku Game Master przekazuje wiadomość do Usera poprzez Communication Server. User pytający aktualizuje swój widok planszy.


Answer to a request for information
Asked User
CS
GM
Asking User
[Request]
[Demand]
[Doesn't want to reply]
[Wants to reply]
Prepare answer
Send
Sleep
Forward
[User sleeping]
[User not sleeping]
Forward
Ignore
Forward
Update board

9 Opis formatu wiadomości JSON

CONNECT (GM-USER):
"MessageType" : "ConnectToGameMaster",
"Team" : int
// Team przyjmuje wartość 0 lub 1. Oznacza drużynę do której gracz chce się połączyć

CONNECT (GM-CS):
"MessageType" : "ConnectToCS"

ACCEPT (GM-CS):
"MessageType": "AcceptGMConnection"

REJECT:
"MessageType": "RejectConnection"

ACCEPT:
"MessageType": "AcceptConnection"
"Id": int
// Id jest identyfikatorem gracza. GM przesyła taką informację do łączącego się gracza.

REQUESTINFO:
"MessageType": "RequestInfo",
"ObligatoryResponse": boolean,
"IdFrom": int,
"IdTo": int
// ObligatoryResponse będzie identyfikowało request od lidera drużyny, co oznacza, że na ten request musi być wysłana odpowiedź. GM będzie sprawdzał czy pytający jest liderem.

RESPONSETOINFO:
"MessageType": "ResponseToInfo",
"IdFrom": int,
"IdTo": int,
"Grid":[["IsPossibleGoal": boolean, "DistToPiece": int, "Timestamp": long]],
"DestinationX": int,
"DestinationY": int,
"PositionX": int,
"PositionY": int
// Odpowiedź na prośbę o informacje zawiera całą planszę z perspektywy gracza odpowiadającego. Plansza reprezentowana jest jako jednowymiarowa tablica informacji o tile'ach. Do ustalenia współrzędnych (x,y) danego elementu w tablicy gracz będzie korzystał ze swojej wiedzy o szerokości i wysokości planszy.
// DestinationX oraz DestinationY to obecne cele danego gracza. Jeśli przynajmniej jedna z tych

wartości jest mniejsza od zera, to znaczy to, iż gracz obecnie nie ma żadnego celu.

YOUSHOULDSLEEP:
"MessageType": "YouShouldSleep",
"TimeoutTime": long
// TimeoutTime nie daje informacji o tym, ile gracz powinien spać: o tym decyduje wyłącznie GameMaster wysyłając wiadomość WakeUp. Natomiast TimeoutTime powinien dawać informacje o tym, na ile snu gracz powinien się przygotować.

WAKEUP:
"MessageType": "WakeUp"
// Wiadomość ta, wysłana do gracza, wybudza go ze snu. Jeśli gracz nie śpi, to nic nie robi.

REQUESTMOVE:
"MessageType": "RequestMove",
"Id": int,
"DestinationX": int,
"DestinationY": int
// DestinationX i DestinationY to współrzędne (x,y) pola, na które gracz chce się ruszyć

RESPONSEMOVEOK:
"MessageType": "ResponseMoveOk",
"DestinationX": int,
"DestinationY": int,
"DidPickupPiece": boolean,
"ManhattanDistanceToNearestPiece": int
// DestinationX i DestinationY to współrzędne (x,y) pola, na które gracz się ruszył. DidPickupPiece informuje gracza, czy wszedł na pole z Piece'm i go podniósł. ManhattanDistanceToNearestPiece to dystans z nowego pola do innego najbliższego kawałka.

RESPONSEMOVENOTOK:
"MessageType": "ResponseMoveNotOk",
"ErrorString": int,
// ErrorString to enumeracja, która przyjmuje jedną z trzech wartości: 0 (Out of Reach), 1 (IsOccupied), 2 (Out of Bounds). Out of Reach oznacza, że pole jest zbyt daleko i gracz nie może do niego przejść w jednym ruchu. IsOccupied oznacza, że na polu już jest gracz. Out of Bounds oznacza, że pole jest poza planszą.

REQUESTCHECKPIECE:
"MessageType": "RequestCheckPiece",
"Id": int

RESPONSECHECKPIECEOK

"MessageType": "ResponseCheckPieceOk",
"IsSham": boolean

RESPONSECHECKPIECENOPIECE
"MessageType": "ResponseCheckPieceNoPiece"
// Taką odpowiedź otrzyma gracz, jeśli wysłał prośbę o sprawdzenie kawałka, podczas gdy tego kawałka nie posiada.

REQUESTPLACEPIECE:
"MessageType": "PlacePiece",
"Id": int

RESPONSEPLACEPIECEOK:
"MessageType": "ResponsePlacePieceOk",
"GoalAchieved": boolean

RESPONSEPLACEPIECENOPIECE:
"MessageType": "ResponsePlacePieceNoPiece"
// Taką odpowiedź otrzyma gracz, jeśli wysłał prośbę o położenie kawałka, podczas gdy tego kawałka nie posiada.

REQUESTRESOLVEDISTANCES:
"MessageType": "RequestResolveDistances",
"Id": int
// Prośba gracza o określenie dystansów ośmiu otaczających pól od najbliższych kawałków
RESPONSERESOLVEDISTANCES
"MessageType": "ResponseResolveDistances",
"Distances": [["PositionX": int, "PositionY": int, "Distance": int], ...]
// Odpowiedź składa się z 9 elementów. Każdy z nich ma współrzędne punktu względem którego liczymy dystans Manhattan, oraz ten dystans

STARTGAME
"MessageType": "StartGame",
"Width": int,
"Height": int,
"GoalAreaHeight: int,
"PlayerCount": int,
"StartingPositionX": int,
"StartingPositionY": int
// PlayerCount to liczba graczy w jednej drużynie. Każdy gracz otrzymuje indywidualną wiadomość StartGame, a StartingPosition to startowa pozycja gracza. Width i Height to szerokość i wysokość planszy, GoalAreaHeight to wysokość Goal Area. GoalAreaHeight jest taka sama dla obu drużyn.

ENDGAME

"MessageType": "EndGame",

"Score0": int,

"Score1": int

10 Przykładowa konfiguracja

10.1 Konfiguracja GameMaster

- n_players: 5 (liczba graczy w jednej drużynie)
- grid_width: 10
- grid_height: 10
- goal_area_height: 3
- n_goals: 3
- n_pieces: 4
- sham_chance: 0.2
- request_info_timeout: 2000
- send_info_timeout: 5000
- discover_neighbor_fields_timeout: 1250
- place_piece_timeout: 1000
- test_piece_timeout: 1500
- move_timeout: 250
- cs_address: 192.168.0.1
- cs_port: 7573

10.2 Konfiguracja CommunicationServer

- game_master_lister_port: 7573
- player_listener_port: 1237

10.3 Konfiguracja Player

- cs_address: 192.168.0.1
- cs_port: 1237
- team: 0