IO - The Project Game

Bartosz Chrostowski Emil Dragańczuk Jakub Drak Sbahi
Mikołaj Molenda Alicja Moskal

Spis treści

Język

- **GM** *Game master* - moduł odpowiedzialny za prowadzenie rozgrywki
- **Agent** *Player* - moduł sterujący graczem, utożsamiany z **graczem**
- **Serwer komunikacyjny** *Communications Server* - serwer odpowiedzialny za komunikację między Agentami a GM
- **plansza** - macierz na której rozgrywana jest gra
- **pole** *field* - komórka macierzy planszy
- **fragment** *piece* - obiekt pojawiający się na polu zadań, który należy położyć na cel
- **fragment fikcyjny** *sham piece* - fragment który po położeniu w polu bramkowym nie powoduje informacji zwrotnej czy pole było celem
- **pole zadań** *Tasks area* - zbiór pól niebędących polami bramkowymi na których mogą pojawić się fragmenty
- **cel** *goal* - pole znajdujące się w polu bramkowym na które należy położyć fragment
- **pole bramkowe** *Goals area* - zbiór pól na których mogą być cele
- **akcja** - czynność wykonywana przez agenta związana z rozgrywką
- **akcja odkrycia** *discovery* - akcja agenta polegająca na odpytaniu o odległości przylegajacyh do agenta pól
- **punkty** - ilość celów zakrytych fragmentami
- **blokada** - czas podczas którego Agent nie może wykonywać akcji

1 Reguły

Gracze - Agenci są podzieleni na dwie drużyny: czerwoną i niebieską. Agenci w obrębie drużyny muszą ze sobą współpracować. Możliwe akcje dla Agentów:

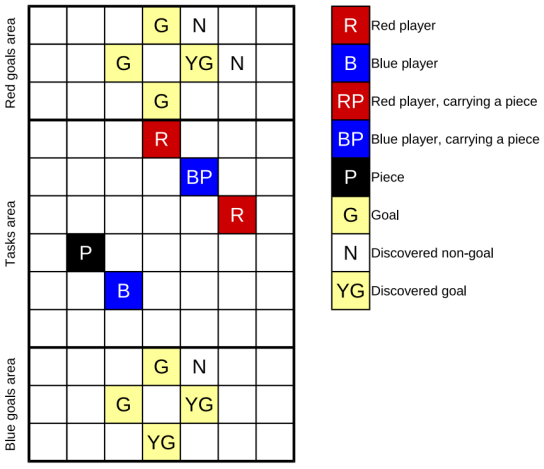
- ruch po planszy
- odkrycie 8 przyległych pól
- sprawdzenie fikcyjności fragmentu
- zniszczenie fragmentu
- odłożenie fragmentu
- wymiana informacji z innym agentem

Całość rozgrywki odbywa się na wygenerowanej planszy na której znajdują się Agenci, projekty (zbiory pól - celi dla danej drużyny) i fragmenty. Agenci poruszają się po planszy i zbierają fragmenty, aby przenieść je i ułożyć na swoich celach.

Plansza podzielona jest na 3 rozłączne obszary, kolejno: pole bramkowe drużyny czerwonej, pole zadań i pole bramkowe drużyny niebieskiej. Wymiary planszy są ustalane przed rozpoczęciem rozgrywki. W polach bramkowych znajdują się ukryte cele dla odpowiadających im drużyn. W polu zadań podczas rozgrywki pojawiają się fragmenty i fragmenty fikcyjne.

Celem każdej drużyny jest skompletowanie projektu, czyli zakrycie odpowiednich pól zebranymi fragmentami. Wygrywa drużyna, która jako pierwsza zakryje wszystkie swoje pola celów.

Projekty generowane są na początku rozgrywki, identyczne dla każdej drużyny. Pola projektu początkowo są ukryte. Agenci w trakcie rozgrywki mogą je odkryć poprzez akcję sprawdzenia wybranego pola lub położenie na niego fragmentu.



Rysunek 1: Mapa planszy

1.1 Założenia ogólne

- Są dwie drużyny
- Powierzchnia pól bramkowych jest równa dla obu drużyn
- Pole bramkowe zawsze ma szerokość taką jak szerokość pola planszy
- Pola bramkowe stanowi n pierwszych i n ostatnich wierszy planszy, $n > 0$
- Pola bramkowe są spójne
- Powierzchnia pola zadań oraz powierzchnia obu pól bramkowych są niezerowe
- Pola bramkowe (ułożenie celów) obu drużyn są symetryczne względem środka planszy
- Odległości liczone są w metryce Manhattan
- Każda akcja powoduje założenie na agenta blokady, na czas zależny od akcji i określony przed rozgrywką, podczas której nie może wykonywać akcji
- W każdej drużynie jest jeden lider
- Projekty do ułożenia są dla każdej drużyny identyczne
- Warunkiem wygranej jest zakrycie wszystkich celów, czyli zdobycie tylu punktów ile jest celów
- Akcje agentów wykonywane są asynchronicznie

1.2 Kary czasowe

- Każda akcja (oraz zapytanie o wymianę informacji i odpowiedź na wymianę informacji) wiąże się z koniecznością odczekania czasu przed kolejną akcją
- W przypadku wysłania zapytania przed upłynięciem kary czasowej, jest ono ignorowane przez GM i wysyłany jest stosowny komunikat błędu
- Kary czasowe ustalane są przed rozgrywką

1.3 Poruszanie

- Agent może poruszać się w 4 kierunkach: zachód, północ, wschód, południe, to jest; może przemieścić się na pole które jest przyległe do pola na którym się znajduje
- Agent może przebywać na dowolnym polu planszy niezajętym przez innego Agenta i nie należącym do pola bramkowego drużyny przeciwnej
- Próba ruchu poza planszę skutkuje brakiem ruchu
- Próba poruszenia się na pole zajęte przez innego agenta skutkuje brakiem ruchu
- Próba poruszenia się dwóch różnych agentów na te samo pole skutkuje poruszeniem się jednego z nich, wybór agenta do poruszenia jest dowolny
- Agent wchodzący do pola otrzymuje odległość tego pola od najbliższego kawałka który może podnieść

1.4 Fragmenty

- Na każdym polu planszy, które nie jest polem bramkowym, może znajdować się dowolna ilość fragmentów
- Agent może podnieść fragment jedynie z pola na którym on sam się znajduje
- Agent może trzymać maksymalnie jeden fragment
- W każdym momencie w grze może znajdować się maksymalnie liczba fragmentów, która ustalana jest przed rozpoczęciem rozgrywki w konfiguracji
- Fragmenty generowane są przez Game Mastera co określony interwał czasowy i odkładane na losowe pole zadań
- Fragmenty mogą być fikcyjne
- Odłożenie wygenerowanego przez Game Mastera fragmentu na pole na którym znajduje się Agent, skutkuje natychmiastowym podniesieniem fragmentu przez tego Agenta, jeśli nie trzyma on już innego fragmentu. W przeciwnym przypadku fragment odkładany jest na pole
- Odłożenie fragmentu niefikcyjnego na polu bramkowym które jest celem skutkuje przyznaniem punktu drużynie do której należy dane pole bramkowe. Może nastąpić sytuacja, gdy Agent zdobywa punkt dla przeciwnej drużyny
- Odłożenie fragmentu niefikcyjnego na polu bramkowym które nie jest celem zwraca informację agentowi, że cel nie został osiągnięty
- Odłożenie fragmentu fikcyjnego na polu bramkowym zwraca informację agentowi, że dany fragment był fikcyjny oraz nie przyznaje punktów, nawet jeśli dane pole było celem
- Cel na który został odłożony fragment niefikcyjny przestaje być celem, czyli kolejne odłożenie na nim fragmentu nie przyznaje punktów
- Odłożenie fragmentu na polu bramkowym skutkuje wyłączeniem tego fragmentu z gry, nie można go już podnieść
- Fragment trzymany przez gracza w każdym momencie może być przetestowany na bycie fikcyjnym
- Przetestowany na fikcyjność fragment którego wynik jest pozytywny, jest natychmiast niszczony
- Fragment trzymany przez gracza w każdym momencie może być zniszczony

1.5 Akcja odkrycia

- Udziela informacji o odległościach w metryce Manhattan pól przyległych oraz ze wspólnymi wierzchołkami do pola na którym znajduje się Agent
- Jeśli Agent znajduje się na krawędzi planszy, nie otrzymuje informacji o polach znajdujących się poza planszą

1.6 Wymiana informacji

- W każdym momencie gry agent może zarządzać wymiany informacji z innym agentem
- Odpowiedź na zapytanie agenta niebędącego liderem drużyny pytanego agenta nie jest obligatoryjna
- Odpowiedź na zapytanie agenta będącego liderem drużyny pytanego agenta jest obligatoryjna i skutkuje zablokowaniem akcji pytanego agenta do udzielenia przez niego odpowiedzi
- Odpowiedź na zapytanie może zostać udzielona w dowolnym czasie od zapytania o wymianę informacji, ale nie może zostać wysłana jeśli takie zapytanie nie miało wcześniej miejsca
- Udzielane informacje nie muszą być prawdziwe

2 FURPS - wymagania niefunkcjonalne

2.1 Usability

- system powinien działać po podaniu właściwych opcji konfiguracyjnych,
- moduł GM powinien wyświetlać estetyczną planszę z przebiegiem rozgrywki,
- system powinien działać na domyślnej konfiguracji modułów.

2.2 Reliability

- system powinien być niewrażliwy na ewentualne odłączenie się innych modułów, wykryć takie zachowanie i odpowiednio zareagować,
- system powinien być odporny na przesyłanie do poszczególnych modułów niepoprawnych wiadomości,
- serwer komunikacyjny powinien zapisywać logi z niepoprawnymi wiadomościami.

2.3 Performance

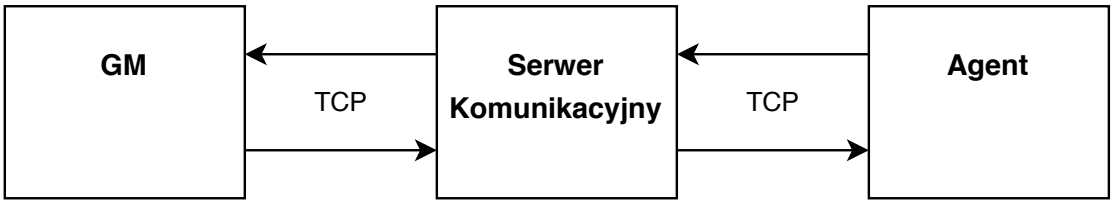
- Instalując moduł Game Mastera i serwera komunikacyjnego na przeciętnym laptopie, system powinien obsłużyć około 40 agentów, a wielkość planszy powinna mieć pomijalny wpływ na wydajność gry,
- skalowalność systemu powinna zależeć od ilości wątków procesora ze względu na asynchroniczny charakter pracy modułów Game Mastera i serwera komunikacyjnego.

2.4 Supportability

- system powinien mieć domyślne ustawienia, które zostają każdorazowo nadpisane przez odpowiednie pliki konfiguracyjne,
- system powinien działać w sieciach wspierających protokół TCP/IP,
- system powinien być aplikacją typu portable, czyli taką którą można uruchomić bez instalacji,
- system powinien być napisany możliwe przenośnie, to znaczy maksymalnie ułatwiając jego przeniesienie na inny system operacyjny.

3 Diagram systemu

System złożony jest z 3 modułów komunikujących się ze sobą przez TCP. W celu uniknięcia skomplikowanej logiki buforowania wiadomości, każdy moduł kończy pracę na niepowodzenie komunikacji, System należy odpalać w kolejności CS -> GM -> Agenty, inna kolejność uruchamiania może powodować zamknięcie całego systemu kaskadowo.

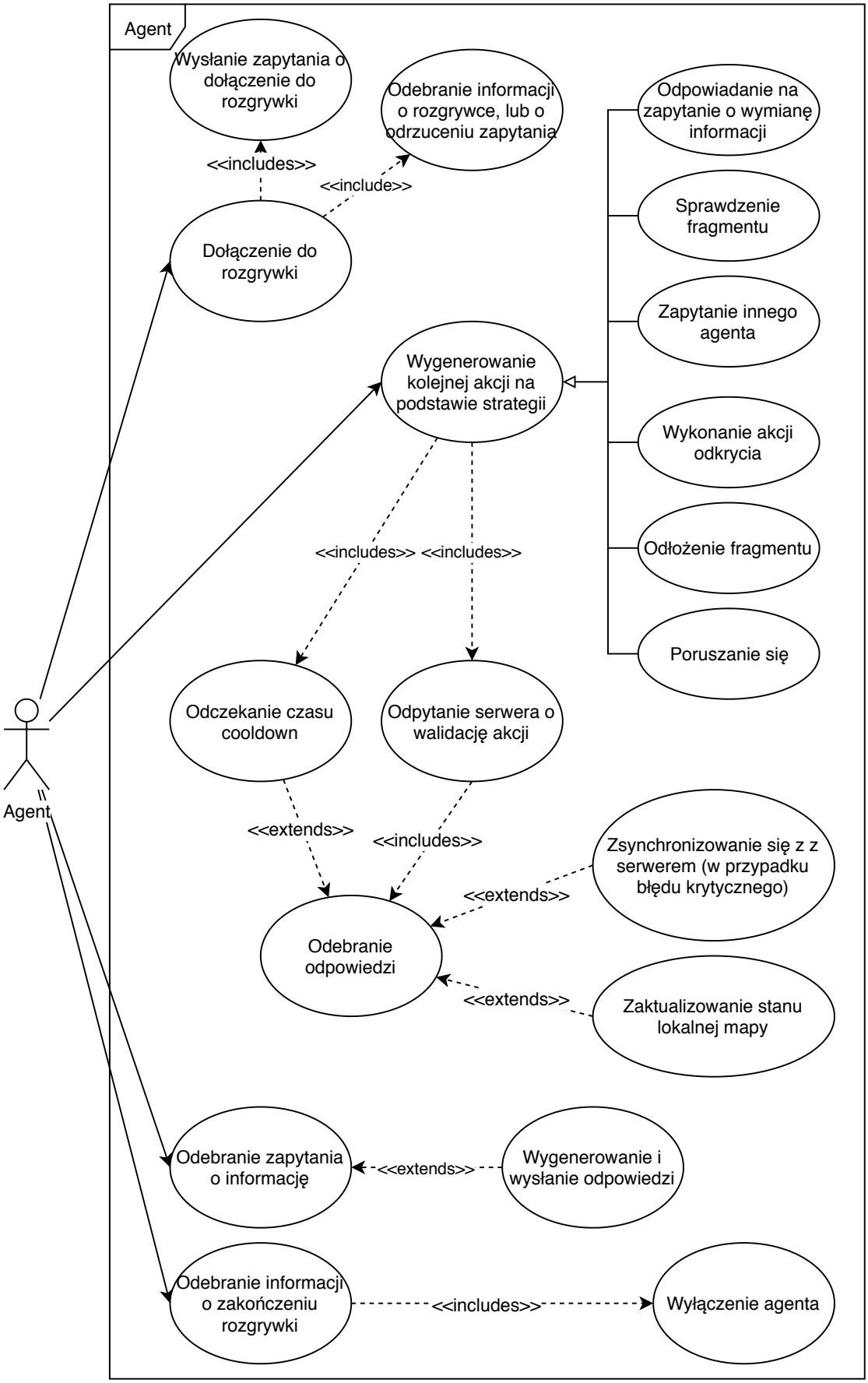


Rysunek 2: Diagram systemu

4 Przypadki Użycia

4.1 Agent PU

Agent jest graczem w **The Project Game**. Należy do jednej z dwóch drużyn. Integruje się z modułem **Game Master** poprzez moduł **Serwera Komunikacyjnego**. Przed wykonaniem jakichkolwiek akcji musi dołączyć do rozgrywki wysyłając odpowiednie zapytanie do GM. Posiada ograniczoną wiedzę na temat aktualnego stanu planszy i może ją zdobywać poprzez wykonywanie akcji oraz przez wymianę informacji z innymi graczami, zatem jego wiedza może być nieprawdziwa. Z tego powodu, aby wykonać jakąkolwiek akcję, agent musi uprzednio odpytać o nią GM, który posiada pełną wiedzę na temat aktualnego stanu planszy i może sprawdzić poprawność i wynik akcji. Celem Agenta jest wygrana drużyny do której należy, poprzez realizację obranej strategii.



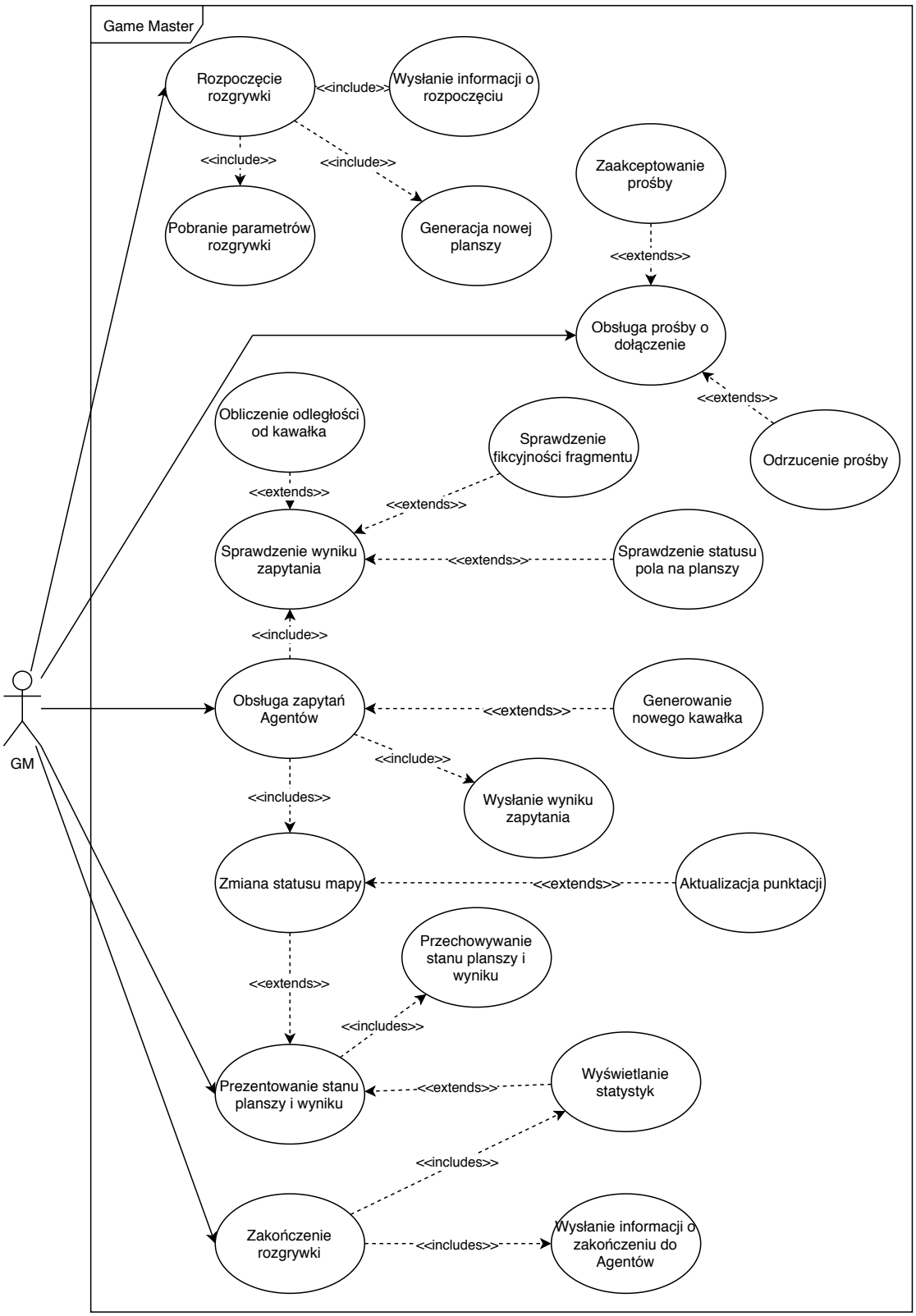
Rysunek 3: Diagram przypadków użycia modułu Agent i aktora Agent

- Dołączenie do rozgrywki
  - Odbywa się poprzez połączenie się z Serwerem Komunikacyjnym na podstawie danych połączenia wprowadzonych przez użytkownika i wykonanie odpowiedniego zapytania do GM. Jeśli połączenie nie powiodło się, Agent kończy działanie.
  - Odpowiedź zwrotna może być negatywna, w tym przypadku Agent kończy działanie.
  - W przypadku odpowiedzi pozytywnej, Agent aktualizuje swoje informacje o parametrach rozgrywki na podstawie informacji otrzymanych z GM i przystępuje do oczekiwania na informację o rozpoczęciu rozgrywki, podczas którego nie wykonuje żadnych czynności.
- Trwająca rozgrywka
  - Agent może wygenerować akcję na podstawie swojej strategii, przed wysłaniem zapytania do GM, powinien uprzednio odczekać czas kary nałożony przez GM w ramach wykonania poprzedniej akcji, w przeciwnym przypadku otrzyma odpowiedź o błędzie od GM.
  - W zależności od typu akcji, agent generuje różne rodzaje zapytań do GM. Akcje niewymagające przesyłania dodatkowych informacji do GM to: odłożenie fragmentu, sprawdzenie fragmentu, oraz wykonanie akcji odkrycia. Zapytanie o ruch wymaga wyspecyfikowania kierunku, zapytanie o wymianę informacji wymaga wyspecyfikowania id pytanego gracza, a odpowiedź na wymianę informacji wymaga wyspecyfikowania wiedzy agenta.
  - Na każde zapytanie agent otrzymuje odpowiedź, którą powinien odebrać i zinterpretować, na podstawie nich Agent może zaktualizować stan swojej wiedzy o planszy. Jeśli była to odpowiedź o błędzie, Agent może na podstawie jej zawartości zsynchronizować się z GM.
  - W trakcie trwającej rozgrywki Agent może otrzymać zapytanie o udzielenie informacji. W takim przypadku powinien, albo odpowiedzieć na takie zapytanie natychmiastowo, lub odłożyć taką odpowiedź na później i wysłać ją w ramach wygenerowania akcji na podstawie strategii. Jeśli zapytanie zostało wysłane przez lidera, odpowiedź jest obligatoryjna i agent powinien odpowiedzieć na takie zapytanie natychmiastowo.
- Zakończenie rozgrywki
  - W trakcie rozgrywki Agent może otrzymać informację o jej zakończeniu. W przypadku takiej informacji, niezależnie od powodu zakończenia rozgrywki, Agent powinien zakończyć działanie.
  - Informacją o zakończeniu rozgrywki jest zarówno specjalna wiadomość wygenerowana przez GM, jak i każda informacja o błędzie połączenia. Nie są prowadzone ponowne próby połączenia.

4.2 Game Master

Moduł Game Master przeprowadza całą rozgrywkę w The Project Game. Przechowuje aktualny stan gry, a każda akcja Agentów musi najpierw zostać odnotowana i zwalidowana

przez niego. Cała komunikacja między Agentami i Game Masterem odbywa się poprzez moduł Serwera Komunikacyjnego. Game Master odpowiada także za dostarczenie interfejsu graficznego do prezentacji stanu gry i wyników.



Rysunek 4: Diagram przypadków użycia modułu GM i aktora GM

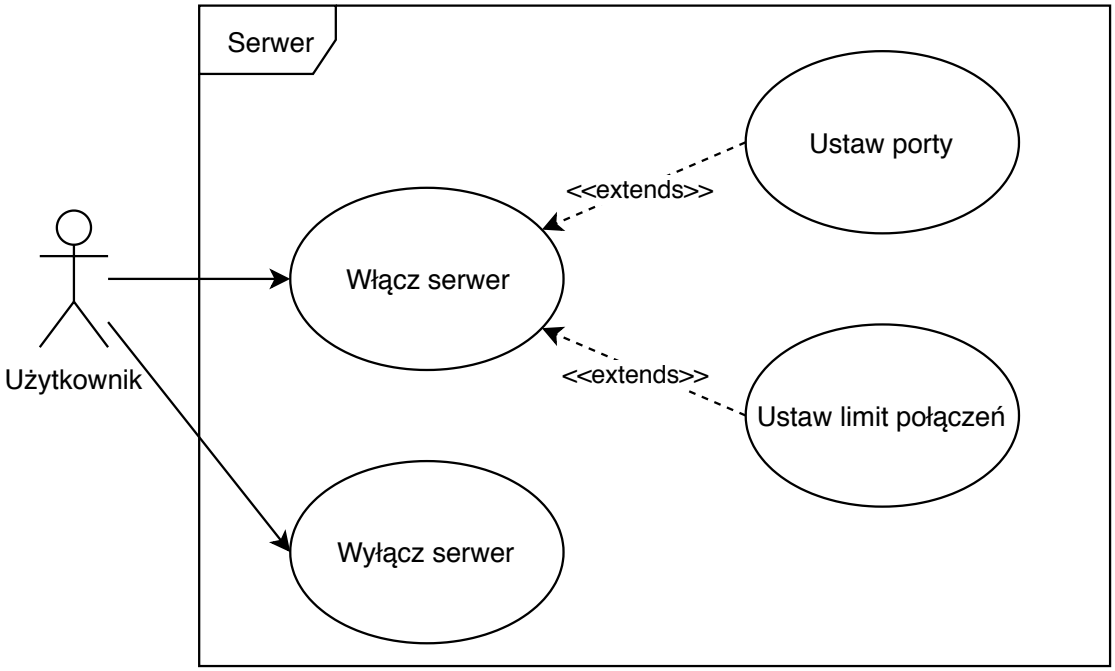
- Inicjacja
  - GM pobiera parametry rozgrywki z domyślnej konfiguracji, bądź dostarczonej przez użytkownika.
  - Jeśli konfiguracja została podana przez użytkownika, Game Master sprawdza czy ilość pól dostępnych na planszy jest większa niż maksymalna ilość graczy. Jeśli nie odpytuje użytkownika o ponowne wprowadzenie konfiguracji.
  - Nawiązuje połączenie z Serwerem Komunikacyjnym na podstawie danych połączenia zawartych w konfiguracji, w przypadku nieudanej próby połączenia kończy działanie.
  - Generuje nową planszę na podstawie konfiguracji oraz przejście do oczekiwania na dołączenie agentów.
- Obsługa prośby o dołączenie
  - Game Master może odrzucić lub zaakceptować prośbę, w zależności od limitu Agentów w drużynie określonej w konfiguracji.
- Rozpoczęcie rozgrywki
  - Na komendę użytkownika GM rozpoczyna rozgrywkę.
  - Ustala początkowe pozycje każdego z agentów, liderów drużyn i pozycje fragmentów.
  - Wysyła informację o rozpoczęciu do wszystkich agentów, wraz z początkowymi danymi na temat planszy (wielkość planszy, wielkosć stref punktowych, pozycja startowa Agenta, oraz identyfikatory pozostałych Agentów i ich liderów).
  - Przechodzi w stan obsługi zapytań agentów.
- Obsługa zapytań Agentów
  - Game Master na podstawie globalnego stanu planszy waliduje zapytanie Agenta i jeśli zapytanie jest nieprawidłowe odsyła informację o błędzie i jego przyczynie.
  - Jesli zapytanie było prawidłowe, GM na podstawie aktualnego stanu mapy generuje odpowiedź dla Agenta i wysyła ją.
  - Akcja agenta może spowodować zmianę stanu planszy, w tym wygenerowanie nowego kawałka jeśli akcją było odłożenie fragmentu w polu bramkowym lub zniszczenie go.
  - Akcja agenta może również spowodować zmianę stanu punktacji drużyn, jeśli tak się stało GM powinien sprawdzić czy rozgrywka nie powinna zostać zakończona.
- Prezentowanie stanu planszy wraz z wynikiem
  - Game Master w graficznym interfejsie użytkownika prezentuje aktualny stan planszy, wyniki drużyn oraz meczowe statystyki.
  - Każda akcja zmieniająca stan planszy powoduje aktualizację wyświetlanego widoku.
- Zakończenie rozgrywki

- Zakończenie rozgrywki może być rezultatem trzech zdarzeń, wygraną którejś z drużyn, zarządzeniem zakończenia przez użytkownika oraz utratą połączenia z Serwerem Komunikacyjnym.
- Po zakończeniu Game Master, o ile to możliwe, wysyła informacje o końcu gry do Agentów i prezentuje statystyki w interfejsie graficznym.

4.3 Użytkownik

Użytkownik jest jest osobą obsługującą The Project Game. Inicjuje wszystkie moduły. Posiada wymienione poniżej możliwości ingerencji w działanie poszczególnych modułów.

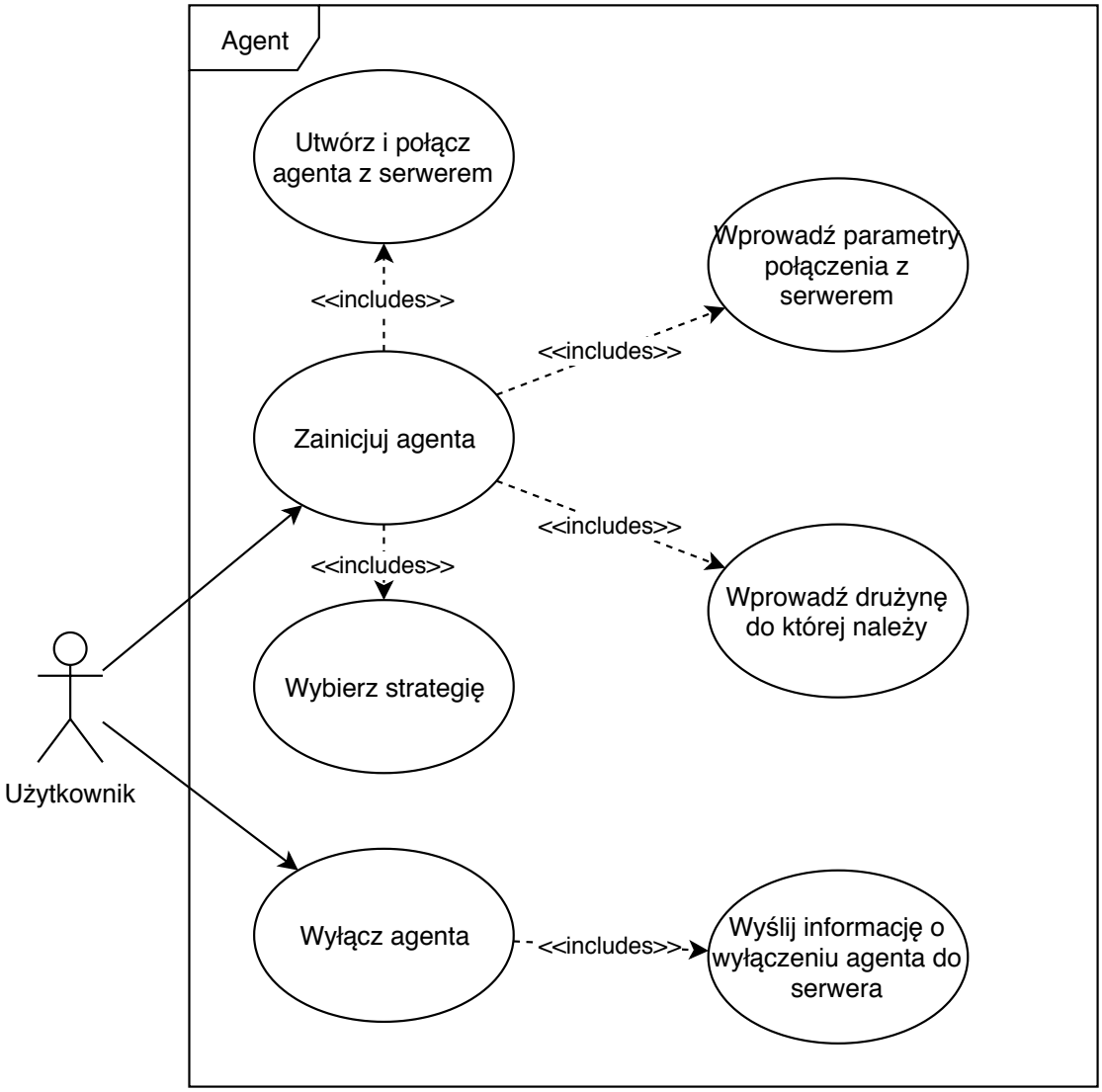
4.3.1 Moduł Serwera Komunikacyjnego



Rysunek 5: Diagram przypadków użycia modułu Serwera Komunikacyjnego i aktora Użytkownik

- Włączanie serwera
  - Ustawianie portów do połączeń z GM oraz agentami.
  - Ustawianie limitu ilości połączeń.
- Wyłączanie serwera w dowolnym momencie

4.3.2 Moduł agenta



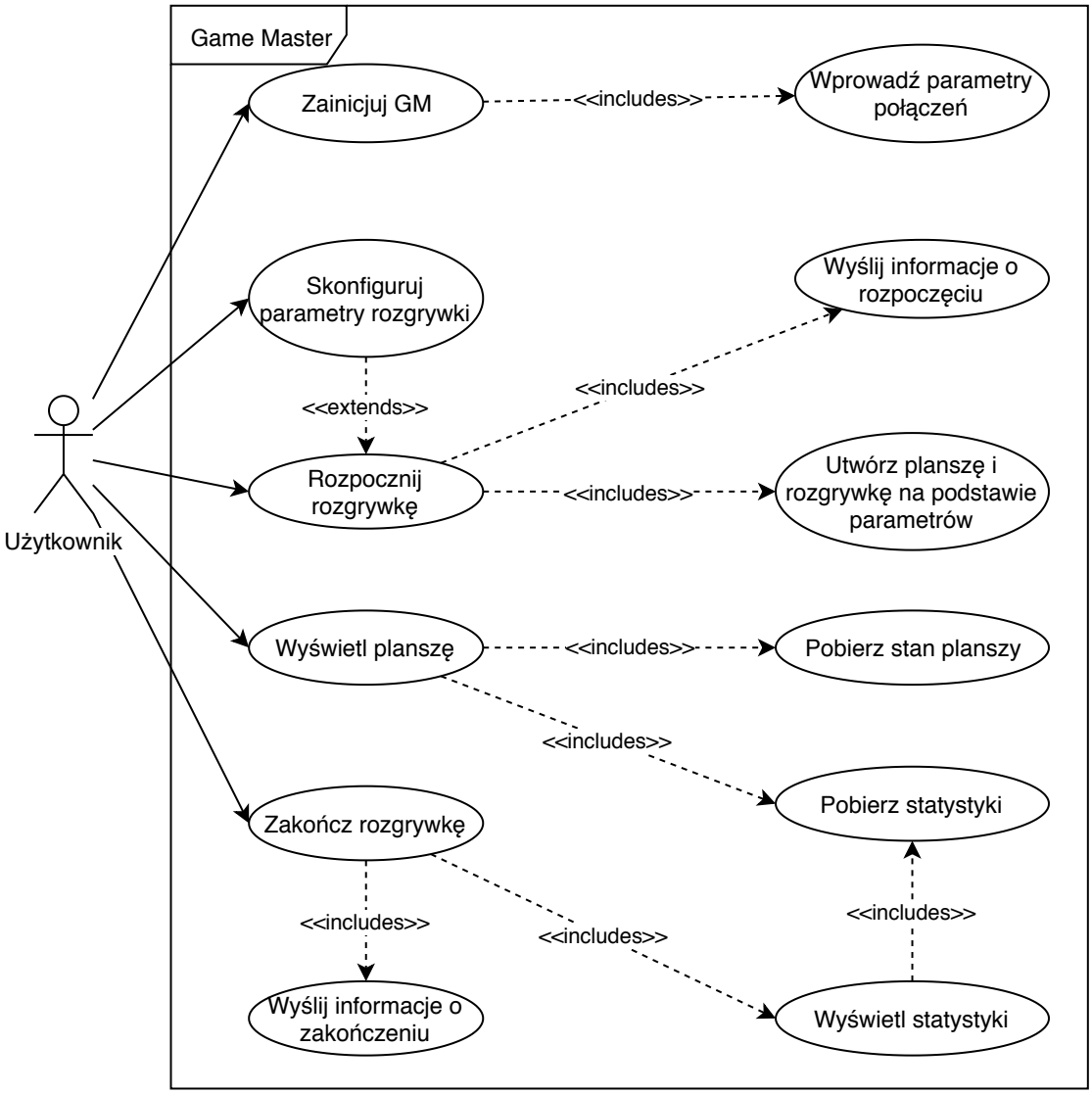
Rysunek 6: Diagram przypadków użycia modułu Agent i aktora Użytkownik

- Inicjowanie agenta
  - Zainicjowanie agenta wymaga parametrów połączenia z Serwerem Komunikacyjnym.
  - Wyspecyfikowanie do której drużyny należy agent (Red, Blue).
  - Wybranie strategii używanej podczas gry przez agenta spośród zaimplementowanych w module agenta. Od wybranej strategii zależy sposób podejmowania

przez agenta takich decyzji jak: przesunięcie się na planszy, opuszczenie fragmentu, czy odpowiedź innemu agentowi na zapytanie. Wybór strategi sprowadza się do wybrania identyfikatora strategii spośród specyfikowanych przez konkretnego agenta.

- Wyłączanie agenta
  - Agenta można wyłączyć w każdym momencie, nie ma to wpływu na przebieg rozgrywki w której uczestniczył.

4.3.3 Moduł GM



Rysunek 7: Diagram przypadków użycia modułu GM i aktora Użytkownik

- Inicjowanie GM
  - Zainicjowanie GM wymaga wprowadzanie parametrów połączeń.
- Użytkownik ma możliwość skonfigurowania parametrów rozgrywki, przed jej rozpoczęciem
  - Ilość celów w polu bramkowym

  - Wymiary pola bramkowego
  - Opóźnienie w wykonywaniu ruchów przez agenta
  - Wymiary planszy
  - Maksymalna ilość agentów
  - Prawdopodobieństwo, że pojawiający się fragment jest fragmentem fikcyjnym
  - Częstotliwość generowania nowego fragmentu na planszy
- Rozpoczynanie rozgrywki
  - Użytkownik może zażądać rozpoczęcia rozgrywki, od tego momentu GM przechodzi w stan wyświetlania planszy i nie możliwa jest zmiana kofiguracji rozgrywki.
- Wyświetlanie planszy
  - Wyświetlenie graficznej interpretacji aktualnego stany gry. Na widoku znajduje się mapa z rozmieszeniem pól, Agentów, fragmetów i podstawowe statystyki rozgrywki.
- Zakończenie rozgrywki
  - W dowolnym momencie trwającej rozgrywki, użytkownik może zażądać zakończenia jej. Żądanie to przekierowuje na ekran z statystykami rozgrywki po czym kończy działanie GM.

5 Mierzenie czasów

Czasy liczone są indywidualnie przez każdy moduł, synchronizacja czasów nie jest prowadzona. Czasy kar podane w konfiguracji są przez GM zaokrąglane do 10ms i taka powinna być dokładność zegarów wszystkich modułów. Agenty powinny same być w stanie obliczyć, kiedy mogą wykonać kolejną akcję na podstawie długości kar i czasów wysłania/odebrania wiadomości. GM na podstawie własnego zegara decyduje czy agent odczekał karę.

6 Komunikacja

Komunikacja odbywa się za pośrednictwem protokołu TCP. Pierwsze 2 bajty zapisane w kolejności `little endian` Oznaczają liczbę bajtów wiadomości do odczytania (wyłączając pierwsze dwa bajty). Wiadomości nie powinny przekraczać długości 8 KiB. Wiadomości kodowane są w UTF-8. Utrata połączenia z którymkolwiek modułem skutkuje w każdym z modułów poinformowaniem użytkownika i zakończeniem działania. Wiadomości wysyłane są w formacie JSON. Wiadomości wysyłane przez GM i Agentów opakowywane są w adapter opisujący kontekst wysyłanej wiadomości. Pole *agentID* uzupełniane jest przez serwer na podstawie jego własnego mapowania klientów na identyfikatory. Po nawiązaniu połączenia przez Agenta z serwerem, serwer nadaje mu unikane *agentID* (liczbę całkowitą), które będzie służyło do korelacji oraz będzie wykorzystywane przez GM w logice gry. *agentID* może być unikalną losowa wartością lub kolejną wartością predefiniowanego ciągu unikalnych wartości. To samo *agentID* wysyłane jest do Agenta w odpowiedzi na prośbę o dołączenie

oraz w wiadomości o rozpoczęciu rozgrywki. Ponowne połączenia po utracie komunikacji nie są możliwe.

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["messageID", "timeSent"],
    "properties": {
        "messageID": {
            "type":"integer"
        },
        "agentID": {
            "type":"integer"
        },
        "payload": {
            "type":"object"
        }
    }
}
```

6.1 Wiadomości Agenta

6.1.1 Zapytanie czy trzymany fragment jest fikcyjny

Payload pusty

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object"
}
```

6.1.2 Zapytanie o zniszczenie fragmentu

Payload pusty

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object"
}
```

6.1.3 Zapytanie o akcję odkrycia

Payload pusty

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object"
}
```

6.1.4 Odpowiedź na wymianę informacji

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["respondToID","distances","redTeamGoalAreaInformations","
       blueTeamGoalAreaInformations"],
    "properties": {
        "respondToID": {
            "type":"integer"
        },
        "distances": {
            "type":"array",
            "items": {
                "type": "number"
            }
        },
        "redTeamGoalAreaInformations": {
            "type":"array",
            "items": {
                "type": "string",
                "enum": ["IDK", "N", "G"]
            }
        },
        "blueTeamGoalAreaInformations": {
            "type":"array",
            "items": {
                "type": "string",
                "enum": ["IDK", "N", "G"]
            }
        }
    }
}
```

6.1.5 Zapytanie o wymianę informacji

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["askedAgentID"],
    "properties": {
        "askedAgentID": {
            "type":"integer"
        }
    }
}
```

6.1.6 Zapytanie o dołączenie do rozgrywki

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["teamId"],
    "properties": {
        "teamId": {
            "type":"string",
            "enum": ["red","blue"]
        }
    }
}
```

6.1.7 Zapytanie o ruch

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["direction"],
    "properties": {
        "direction": {
            "type":"string",
            "enum": ["N", "E", "S", "W"]
        }
    }
}
```

6.1.8 Zapytanie o podniesienie fragmentu

Payload pusty

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object"
}
```

6.1.9 Zapytanie o położenie fragmentu

Payload pusty

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object"
}
```

6.2 Wiadomości GM

6.2.1 Odpowiedź na sprawdzenie fikcyjności

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["sham"],
    "properties": {
        "sham": {
            "type": "boolean"
        }
    }
}
```

6.2.2 Odpowiedź na prośbę o zniszczenie fragmentu

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object"
}
```

6.2.3 Odpowiedź na akcję discovery

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["distanceFromCurrent", "distanceN", "distanceNE", "distanceE"
        , "distanceSE", "distanceS", "distanceSW", "distanceW", "distanceNW"]
        ,
    "properties": {
        "distanceFromCurrent": { "type":"integer" },
        "distanceN": { "type":"integer" },
        "distanceNE": { "type":"integer" },
        "distanceE": { "type":"integer" },
        "distanceSE": { "type":"integer" },
        "distanceS": { "type":"integer" },
        "distanceSW": { "type":"integer" },
        "distanceW": { "type":"integer" },
        "distanceNW": { "type":"integer" }
    }
}
```

6.2.4 Wiadomość o zakończeniu gry

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["winner"],
    "properties": {
        "winner": {
            "type":"string",
            "enum": ["red","blue"]
```

```
        }
    }
}
```

6.2.5 Wiadomość o rozpoczęciu gry

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["agentID","alliesIDs","enemiesIDs","teamId","boardSize","
       goalAreaSize","numberOfPlayers","numberOfPieces","numberOfGoals","
       penalties","shamPieceProbability","position", "leaderID"],
    "properties": {
        "agentID": {
            "type":"integer"
        },
        "alliesIDs": {
            "type":"array",
            "items": {
                "type": "integer"
            }
        },
        "leaderID": {
            "type":"integer"
        },
        "enemiesIDs": {
            "type":"array",
            "items": {
                "type": "integer"
            }
        },
        "teamId": {
            "type":"string",
            "enum": ["red","blue"]
        },
        "boardSize": {
            "type":"object",
            "required":["x", "y"],
            "properties": {
                "x": {"type": "integer"},
                "y": {"type": "integer"}
            }
        },
        "goalAreaSize": {
            "type":"integer"
        },
        "numberOfPlayers": {
```

```
            "type":"object",
            "required":["allies", "enemies"],
            "properties": {
                "allies": {"type":"integer"},
                "enemies": {"type":"integer"}
            }
        },
        "numberOfPieces": {
            "type":"integer"
        },
        "numberOfGoals": {
            "type":"integer"
        },
        "penalties": {
            "type":"object",
            "required":["move","checkForSham","discovery","destroyPiece","
               putPiece","informationExchange"],
            "properties": {
                "move": {"type":"string"},
                "checkForScham": {"type":"string"},
                "discovery": {"type":"string"},
                "destroyPiece": {"type":"string"},
                "putPiece": {"type":"string"},
                "informationExchange": {"type":"string"}
            }
        },
        "shamPieceProbability": {
            "type":"number"
        },
        "position": {
            "type":"object",
            "required":["x", "y"],
            "properties": {
                "x": {"type": "integer"},
                "y": {"type": "integer"}
            }
        }
    }
}
```

6.2.6 Wiadomość przekazująca zapytanie o wymianę informacji do adresata

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["askingID","leader","teamId"],
    "properties": {
```

```
        "askingID": {
            "type": "integer"
        },
        "leader": {
            "type": "boolean"
        },
        "teamId": {
            "type":"string",
            "enum": ["red","blue"]
        }
    }
}
```

6.2.7 Odpowiedź na zapytanie o dołączenie

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["accepted"],
    "properties": {
        "accepted": {
            "type":"boolean"
        },
        "agentID": {
            "type":"integer"
        }
    }
}
```

6.2.8 Odpowiedź na zapytanie o ruch

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["madeMove","currentPosition","closestPiece"],
    "properties": {
        "madeMove": {
            "type": "boolean"
        },
        "currentPosition": {
            "type":"object",
            "required":["x", "y"],
            "properties": {
                "x": {"type": "integer"},
                "y": {"type": "integer"}
            }
        },
        "closestPiece": {
```

```
            "type": "integer",
            "minimum": 0
        }
    }
}
```

6.2.9 Odpowiedź na podniesienie kawałka

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object"
}
```

6.2.10 Odpowiedź na położenie kawałka

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object"
}
```

6.3 Wiadomości błędów

6.3.1 Błędny ruch

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["position"],
    "properties": {
        "position": {
            "type":"object",
            "required":["x", "y"],
            "properties": {
                "x": {"type": "integer"},
                "y": {"type": "integer"}
            }
        }
    }
}
```

6.3.2 Błędne odłożenie kawałka

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["errorSubtype"],
    "properties": {
        "errorSubtype": {
            "type":"string",
```

```
            "enum": ["NothingThere", "Other"]
        }
    }
}
```

6.3.3 Błędne położenie kawałka

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["errorSubtype"],
    "properties": {
        "errorSubtype": {
            "type":"string",
            "enum": ["AgentNotHolding", "CannotPutThere", "Other"]
        }
    }
}
```

6.3.4 Nie odczekanie kary

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["waitUntil"],
    "properties": {
        "waitUntil": {
            "type":"string",
            "format":"date-time"
        }
    }
}
```

6.3.5 Niezdefiniowany błąd

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "position": {
            "type":"object",
            "required":["x", "y"],
            "properties": {
                "x": {"type": "integer"},
                "y": {"type": "integer"}
            }
        },
        "holdingPiece": {
            "type":"boolean"
```

```
        }
    }
}
```

6.4 Id wiadomości

6.4.1 Wiadomości Agenta

Id	Nazwa wiadomości
001	Zapytanie czy trzymany fragment jest fikcyjny
002	Zapytanie o zniszczenie fragmentu
003	Zapytanie o akcję odkrycia
004	Odpowiedź na wymianę informacji
005	Zapytanie o wymianę informacji
006	Zapytanie o dołączenie do rozgrywki
007	Zapytanie o ruch
008	Zapytanie o podniesienie fragmentu
009	Zapytanie o położenie fragmentu

6.4.2 Wiadomości GM

Id	Nazwa wiadomości
101	Odpowiedź na sprawdzenie fikcyjności
102	Odpowiedź na prośbę o zniszczenie fragmentu
103	Odpowiedź na akcję discovery
104	Wiadomość o zakończeniu gry
105	Wiadomość o rozpoczęciu gry
106	Wiadomość przekazująca zapytanie o wymianę informacji do adresata
107	Odpowiedź na zapytanie o dołączenie
108	Odpowiedź na zapytanie o ruch
109	Odpowiedź na podniesienie kawałka
110	Odpowiedź na położenie kawałka

6.4.3 Wiadomości błędów

Id	Nazwa wiadomości
901	Błędny ruch
902	Błędne odłożenie kawałka
903	Błędne położenie kawałka
904	Nie odczekanie kary
905	Niezdefiniowany błąd

7 Konfiguracje

7.1 Serwer Komunikacyjny

Konfiguracja Serwera Komunikacyjnego pobierana jest z pliku konfiguracyjnego w postaci JSON, bądź jako parametry wywołania programu.

```
{
        "$schema": "http://json-schema.org/draft-04/schema",
        "type": "object",
        "required":["portAgentow", "portGM"],
        "properties": {
            "portAgentow": { "type":"integer" },
            "portGM": { "type":"integer" }
        }
}
```

7.2 Agent

Parametry pobierane są z pliku konfiguracyjnego w postaci JSON, w przypadku gdy taki nie istnieje CLI odpytuje użytkownika o wprowadzenie potrzebnych parametrów.

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["CsIP", "CsPort", "teamID"],
    "properties": {
        "CsIP": { "type": "string" },
        "CsPort": { "type": "integer" },
        "teamID": { "type": "string", "enum": ["red", "blue"] },
        "strategy": { "type": "integer" }
    }
}
```

7.3 GM

GM zgodnie z założeniami posiada GUI, parametry konfiguracyjne są wybierane przez użytkownika w oknie ustawień przed rozpoczęciem rozgrywki, domyślne wartości uzupełniane są na podstawie pliku konfiguracyjnego w postaci JSON.

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "required":["CsIP", "CsPort", "movePenalty", "askPenalty",
         "discoveryPenalty", "putPenalty", "checkForShamPenalty",
         "responsePenalty", "boardX", "boardY", "goalAreaHight",
         "numberOfGoals", "numberOfPieces", "shamPieceProbability" ],
    "properties": {
        "CsIP": { "type": "string" },
        "CsPort": { "type": "integer" },
        "movePenalty": { "type": "integer" },
        "askPenalty": { "type": "integer" },
        "discoveryPenalty": { "type": "integer" },
        "putPenalty": { "type": "integer" },
```

```
        "checkForShamPenalty": { "type": "integer" },
        "responsePenalty": { "type": "integer" },
        "boardX": { "type": "integer" },
        "boardY": { "type": "integer" },
        "goalAreaHight": { "type": "integer" },
        "numberOfGoals": { "type": "integer" },
        "numberOfPieces": { "type": "integer" },
        "shamPieceProbability": { "type":"number"}
    }
}
```

7.4 Przykładowe pliki konfiguracyjne

7.4.1 Serwer Komunikacyjny

```
{
    "portAgentow": 3727,
    "portGM": 3728
}
```

7.4.2 Agent

```
{
    "CsIP": "192.168.0.0",
    "CsPort": 3729,
    "teamID": "blue",
    "strategy": 3
}
```

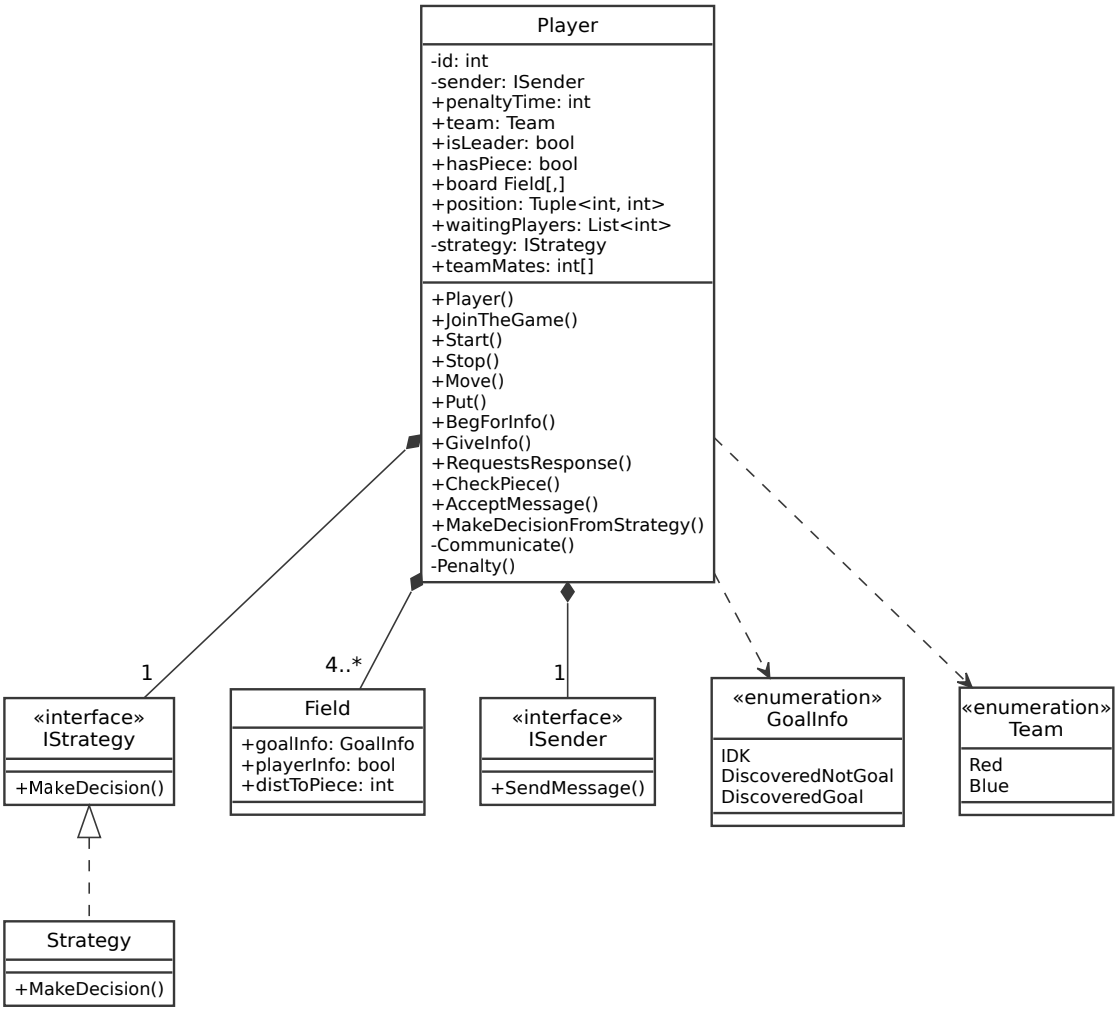
7.4.3 GM

```
{
    "CsIP": "192.168.0.0",
    "CsPort": 12313,
    "movePenalty": 1500,
    "askPenalty": 1000,
    "discoveryPenalty": 700,
    "putPenalty": 500,
    "checkForShamPenalty": 700,
    "responsePenalty": 1000,
    "boardX": 40,
    "boardY": 40,
    "goalAreaHight": 5,
    "numberOfGoals": 5,
    "numberOfPieces": 10,
    "shamPieceProbability": 0.50
}
```

8 Diagramy Klas

W diagramach klas zostały pominięte klasy odpowiedzialne za obsługę połączeń TCP. Jest to zabieg celowy. Nie chcemy narzucać implementacji obsługi połączeń, aby nie ograniczać wyboru API. Klasy posiadają specjalne metody puliczne **AcceptMessage**, do których należy przekazać odebraną na porcie wiadomość i które implementują logikę biznesową. Z anaglogicznego powodu nie narzucamy implementacji interfejsów odpowiedzialnych za wysyłanie wiadomości.

8.1 Moduł Agenta



Rysunek 8: Diagram klas modułu Agenta

8.1.1 Agent

- private int **id**
  Identyfikator gracza.
- public int **penaltyTime**
  Czas kary.
- public Team **team**
  Enum drużyny, w której jest gracz.
- public bool **isLeader**
  Zmienna informująca czy gracz jest liderem zespołu.
- public bool **hasPiece**
  Zmienna informująca czy gracz jest w posiadaniu piece'a.
- public Field[,] **board**
  Tablica dwuwymiarowa przechowująca planszę z perspektywy gracza.
- public Tuple<int,int> **position**
  Współrzędne pola na planszy, na którym stoi gracz.
- public List<int> **waitingPlayers**
  Lista id graczy oczekujących na odpowiedź.
- public int[] **teamMates**
  Lista identyfikatorów graczy z naszej drużyny.
- private IStrategy **strategy**
  Obiekt odpowiedzialny za używaną strategię.

Metody

- internal void **Player()**
  Konstruktor klasy.
- internal void **JoinTheGame()**
  Metoda dołączająca gracza do gry. Wysyła zapytanie do GM z prośbą o dołączenie.
- internal void **Start()**
  Metoda rozpoczynająca grę. Rozpoczyna realizowanie strategii przez agenta.
- internal void **Stop()**
  Metoda zatrzymująca grę. Zatrzymuje pracę agenta i wyłącza go.
- internal void **Move()**
  Metoda ruszająca gracza na wskazane pole. Wysyła zapytanie do GM i w zależności od odpowiedzi aktualizuje stan wiedzy agneta.
- internal void **Put()**
  Metoda odkładająca fragment na planszę. Wysyła zapytanie do GM i w zależności od odpowiedzi aktualizuje stan wiedzy agenta.

- internal void **BegForInfo()**
  Metoda wysyłająca do GM prośbę o wymianę informacji przez innego wybranego agenta
- internal void **GiveInfo()**
  Metoda udzielająca informacji o rozgrywce wskazanemu graczowi.
- internal void **RequestResponse()**
  Metoda wywoływana na przyjście wiadomości z prośbą o wymianę informacji. Zapamiętuje parametry gracza który poprosił informację do listy *waitingPlayers*. W przypadku gdy ten gracz był liderem wywołuje metodę *GiveInfo()*
- internal void **CheckPiece()**
  Metoda sprawdzająca czy fragment to fragment fikcyjny. Wysyła zapytanie do GM i w zależności od odpowiedzi aktualizuje stan wiedzy agenta.
- public void **AcceptMessage()**
  Metoda pobierająca informacje od serwera komunikacyjnego.
- internal void **MakeDecisionFromStrategy()**
  Metoda, która wywołuje metodę z obiektu strategy. Wywoływana jest okresowo i to ona decyduje o akcjach wykonywanych przez agenta. Używa IStrategy.
- private void **Communicate()**
  Metoda wysyłająca wiadomości do serwera komunikacyjnego. Używana przez metody wykonujące akcje do komunikacj z GM poprzez wysłanie wiadomości do Serwera Komunikacyjnego
- private void **Penalty()**
  Metoda czekająca przez okres kary. Po odczekaniu kary wywołuje generowanie kolejnej akcji.

8.1.2 Interface IStrategy

Metody

- internal void **MakeDecision(Agent)**
  Metoda decyduje na podstawie stanu Agenta jaką akcję wykonać i wykonuje ją na obiekcie Agenta.

8.1.3 Klasa Field

- public GoalInfo **goalInfo**
  Stan wiedzy gracza na temat tego pola.
- public bool **playerInfo**
  Informacja czy inny gracz stoi na tym polu.
- public int **distToPiece**
  Odległość pola od najbliższego fragmentu.

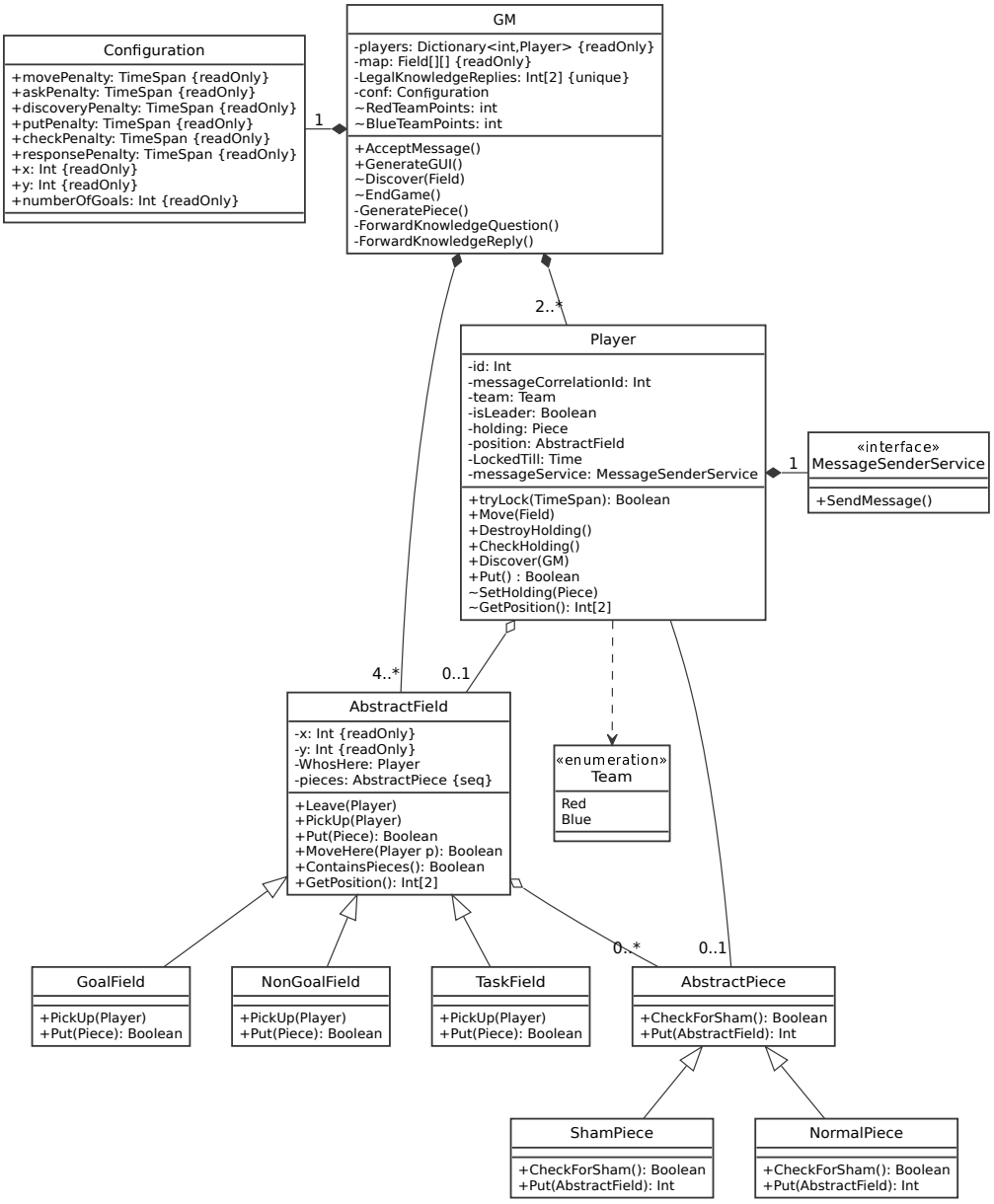
8.1.4 Enum Team

- **Red** - informacja, że gracz jest w drużynie czerwonej,
- **Blue** - informacja, że gracz jest w drużynie niebieskiej.

8.1.5 Enum GoalInfo

- **IDK** - gracz nic nie wie o danym polu,
- **DiscoveredNotGoal** - pole nie jest celem,
- **DiscoveredGoal** - pole jest celem i jest na nim położony fragment.

8.2 Moduł GM



Rysunek 9: Diagram klas modułu GM

8.2.1 Player

Zmienne

- private Int **id**
  Identyfikator gracza

- private Int **messageCorrelationId**
  Id nadane graczowi przez serwer komunikacyjny, służy do odsyłania wiadomości
- private Team **team**
  Drużyna,w której jest gracz
- private Boolean **isLeader**
  Zmienna informująca czy gracz jest liderem zespołu
- private Piece **holding**
  Zmienna informująca czy niesie fragment
- private AbstractField **position**
  Pozycja gracza na planszy
- private Time **LockedTill**
  Czas do kiedy gracz jest zablokowany
- private MessageSenderService **messageService**
  Obiekt interface'u odpowiedzialnego za komunikację

Metody

- public bool **tryLock(TimeSpan)**
  Metoda sprawdza czy gracz jest zablokowany. Jeśli jest zwraca false, jeśli nie jest zakłada nową blokadę czasową na czas podany w argumencie. Wykorzystywana do sprawdzania czy Agent jest w trakcie odczekiwania kary czasowej.
- public void **Move(Field)**
  Metoda poruszająca gracza na wskazane pole. Korzysta z metody obiektu postion. Wysyła agentowi informację przez MessageSenderService.
- public void **DestroyHolding()**
  Metoda niszcząca trzymany fragment.
- public void **CheckHolding()**
  Metoda sprawdzająca czy trzymany fragment jest fragmentem fikcyjnym. Korzysta z metody obiektu holding. Wysyła agentowi informację przez MessageSenderService.
- public void **Discover(GM)**
  Metoda wykonująca akcję odkrycia. Wykorzystuje funkcję klasy GM. Wysyła agentowi informację przez MessageSenderService.
- public bool **Put()**
  Metoda odkładająca fragment w polu, w którym się znajdujemy. Korzysta z metody obiektu postion. Wysyła agentowi informację przez MessageSenderService. Zwraca true jeśli drużynie należy przyznać punkt.
- internal void **SetHolding()**
  Metoda przypisuje graczowi fragment.
- internal int[2] **GetPosition()**
  Metoda pobiera współrzędne gracza.

8.2.2 Configuration

Zmienne

- public readonly TimeSpan **movePenalty**
  Czas do odczekania po wykonanym ruchu.
- public readonly TimeSpan **askPenalty**
  Czas do odczekania po odpytaniu innego gracza.
- public readonly TimeSpan **discoveryPenalty**
  Czas do odczekania po wykonanym akcji odkrycia.
- public readonly TimeSpan **putPenalty**
  Czas do odczekania po odłożeniu fragmentu na planszę.
- public readonly TimeSpan **checkPenalty**
  Czas do odczekania po sprawdzeniu czy fragment jest fikcyjny.
- public readonly TimeSpan **responsePenalty**
  Czas do odczekania po odpowiedzeniu innemu graczowi.
- public readonly Int **x**
  Szerokość planszy.
- public readonly Int **y**
  Wysokość planszy.
- public readonly Int **numberOfGoals**
  Liczba celow w grze

8.2.3 AbstractField

Zmienne

- private readonly Int **x**
  Współrzędna x pola na planszy.
- private readonly Int **y**
  Współrzędna y pola na planszy.
- private Player **WhosHere**
  Referencja na obiekt gracza, który znajduje się na polu.
- private AbstractPiece **pieces**
  Lista fragmentów położonych na tym polu.

Metody

- public void **Leave(Player)**
  Metoda wywoływana gdy gracz opuszcza pole. Usuwa referencję na gracza w obiekcie.

- public void **PickUp(Player)**
  Metoda abstrakcyjna, która podnosi fragment z planszy i przypisuje go do gracza. Wywoływana jest na wejście gracza (przemieszczenie się gracza do tego pola) do danego pola
- public bool **Put(Piece)**
  Metoda abstrakcyjna. Wykorzystywana przez Put(AbstractField) z NormalPiece na zasadzie **wzorca projektowego odwiedzający**.
- public bool **PutSham(Piece)**
  Metoda abstrakcyjna. Wykorzystywana przez Put(AbstractField) z ShamPiece na zasadzie **wzorca projektowego odwiedzający**. Zwraca true jeśli dane pole było bramkowe
- public bool **MoveHere(Player)**
  Metoda przesuwa gracza na pole.
- public bool **ContainsPieces()**
  Metoda zwraca informację czy pole zawiera fragment.
- public int[2] **GetPosition()**
  Metoda zwraca współrzędne pola.

NonGoalField : AbstractField Klasa reprezentująca pole bramkowe nie będące celem.

- public bool **PickUp(Player)**
  Nie można podnosić fragmentów z pola bramkowego
- public bool **Put(Piece)**
  Odłożenie nie przyznaje punktów.
- public bool **PutSham(Piece)**
  Odłożenie fragmentu fikcyjneg na dane pole, zwraca false.

GoalField : AbstractField Klasa reprezentująca pole bramkowe będące celem.

- public bool **PickUp(Player)**
  Nie można podnosić fragmentów z pola bramkowego
- public bool **Put(Piece)**
  Odłożenie przyznaje punkty, jeśli wcześniej na te pole nie był odłożony inny fragment
- public bool **PutSham(Piece)**
  Odłożenie fragmentu fikcyjneg na dane pole, zwraca true.

TaskField : AbstractField Klasa reprezentująca pole zadań.

- public bool **PickUp(Player)**
  Jeśli na danym polu jest fragment, Player podnosi go.
- public bool **Put(Piece)**
  Odłożenie fragmentu na dane pole.

- public bool **PutSham(Piece)**
  Odłożenie fragmentu fikcyjnego na dane pole, zwraca false.

8.2.4 AbstractPiece

Metody

- public bool **CheckForSham()**
  Metoda abstrakcyjna sprawdzająca czy fragment jest fragmentem fikcyjnym.
- public int **Put(AbstractField)**
  Metoda abstrakcyjna odkładająca fragment na pole. Wykorzystuje Put(Piece) z AbstractField na zasadzie **wzorca projektowego odwiedzający**.

8.2.5 ShamPiece : AbstractPiece

Metody

- public bool **CheckForSham()**
  Metoda zwraca true.
- public int **Put(AbstractField)**
  Implementacja odwiedzającego, wywołuje PutSham() z AbstractField, jeśli dane pole było bramkowe przekazuje tę informację do gracza.

8.2.6 NormalPiece : AbstractPiece

Metody

- public bool **CheckForSham()**
  Metoda zwraca false.
- public int **Put(AbstractField)**
  Implementacja odwiedzającego, wywołuje Put() z AbstractField, przekazuje informację czy przyznać punkty graczowi.

8.2.7 Interface MessageSenderService

Metody

- public void **SendMessage()**
  Wysłanie wiadomości do serwera komunikacyjnego. Używana przez Playera do wysyłania wiadomości.

8.2.8 GM

Zmienne

- private readonly Dictionary<int,Player> **players**
  Słownik mapujący id gracza z wiadomości na obiekt.

- private readonly Field[][] **map**
  Faktyczny stan planszy.
- private Set<(int,int)> **LegalKnowledgeReplies[2]**
  Zbiór par id które oznaczają, że dana odpowiedź z wymianą informacji jest legalna, bo była poprzedzona zapytaniem.
- private Configuration **conf**
  Obiekt parametrów rozgrywki.

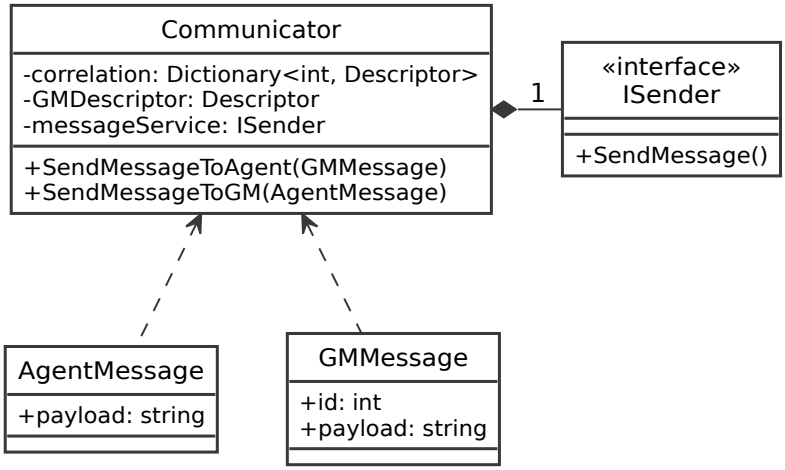
Metody

- public void **AcceptMessage()**
  Metoda służy do obsługi wiadomości, powinna być wywoływana przez API obsługujące komunikację
- private **Discover(Field)**
  Metoda oblicza wynik akcji discovery
- private void **GeneratePiece()**
  Metoda generuje i rozmieszcza fragmenty na planszy.
- private void **ForwardKnowledgeQuestion()**
  Metoda przekazuje prośbę o informację do właściwego gracza.
- private void **ForwardKnowledgeReply()**
  Metoda przekazuje odpowiedź na zapytanie do właściwego gracza, jeśli odpowiedź jest legalna, czyli jeśli dana odpowiedź była poprzedzona zapytaniem.

8.2.9 Enum Team

- **Red** - informacja, że gracz jest w drużynie czerwonej,
- **Blue** - informacja, że gracz jest w drużynie niebieskiej.

8.3 Moduł Serwera Komunikacyjnego



Rysunek 10: Diagram klas modułu Serwera Komunikacyjnego

8.3.1 Klasa Communicator

Zmienne

- private Dictionary<int, Descriptor> **correlation**
  Słownik mapujący wewnętrzne id korelacji gry na deskryptory do wysłania wiadomości wykorzystywane przez API do komunikacji
- private Descriptor **GMDescriptor**
  Deskryptor używany do wysyłania wiadomości do GM
- private ISender **senderService**
  Obiekt wykorzystywany do wysyłania wiadomości

Metody

- internal void **SendMessageToAgent(GMMessage)**
  Metoda wysyłająca odpowiednią wiadomość do Agenta. Na podstawie id z wiadomości znajduje odpowiedni deskryptor i wysyła na niego payload za pomocą ISender. Wywoływana przez API do komunikacji na zdarzenie otrzymania wiadomości od Agenta
- internal void **SendMessageToGM(AgentMessage)**
  Metoda wysyłająca odpowiednią wiadomość do Game Mastera. Wywoływana przez API do komunikacji na zdarzenie otrzymania wiadomości od GM

8.3.2 Klasa GMMessage

- public int **id**
  Identyfikator agenta do którego przesyłana jest wiadomość.

- public string **payload**
  Treść wiadomości przesyłana do agenta w formacie JSON.

8.3.3 Klasa AgentMessage

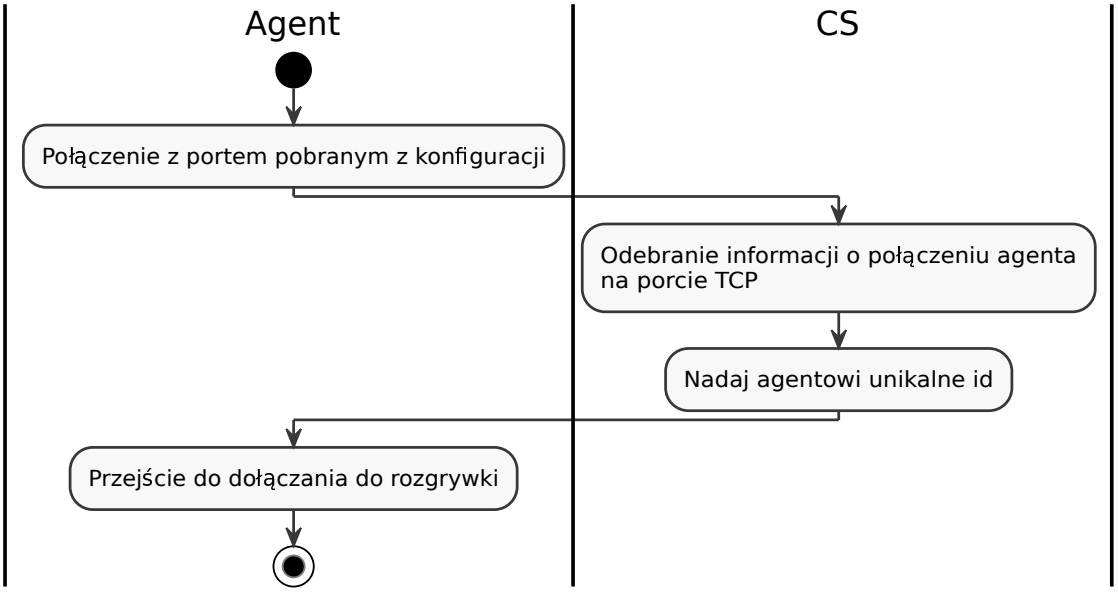
- public string **payload**
  Treść wiadomości przesyłana do Game Mastera w formacie JSON.

8.3.4 Interfejs ISender

- public void **SendMessage()**
  Metoda odpowiedzialna za wysłanie za pomocą API do komunikacji wiadomości do odpowiedniego Agenta lub GM, w zależności od paramterów wywołania

9 Diagramy Aktywności

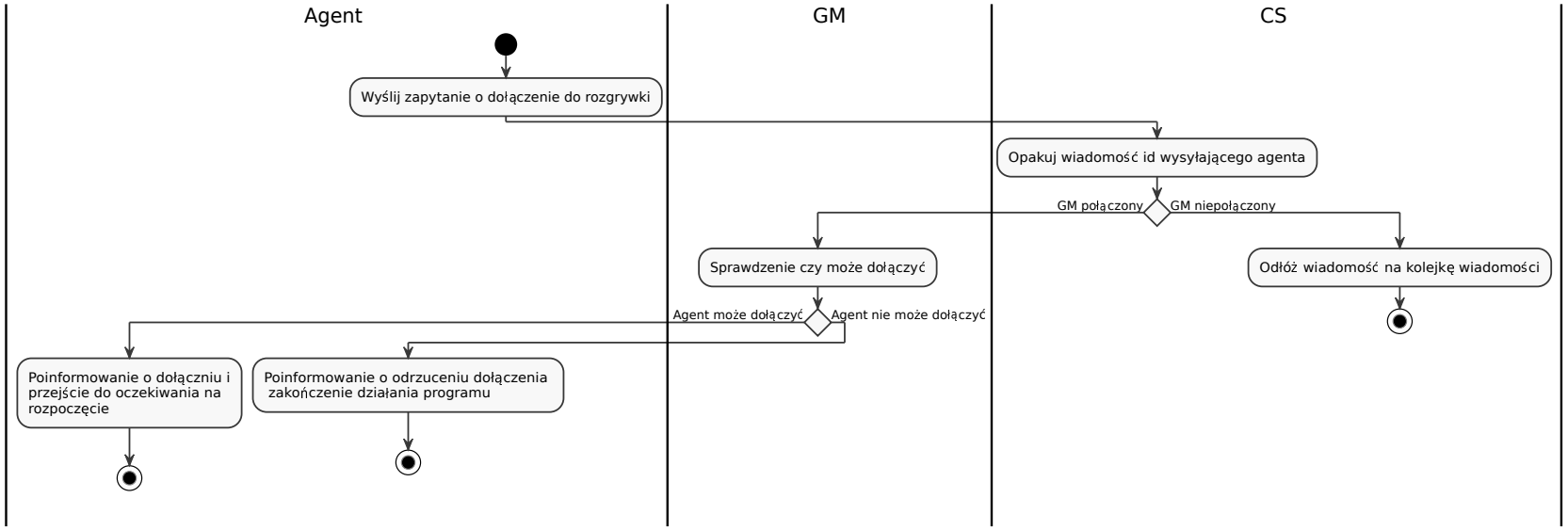
9.1 Połączenie z CS



Rysunek 11: Diagram aktywności połączenia Agenta z CS

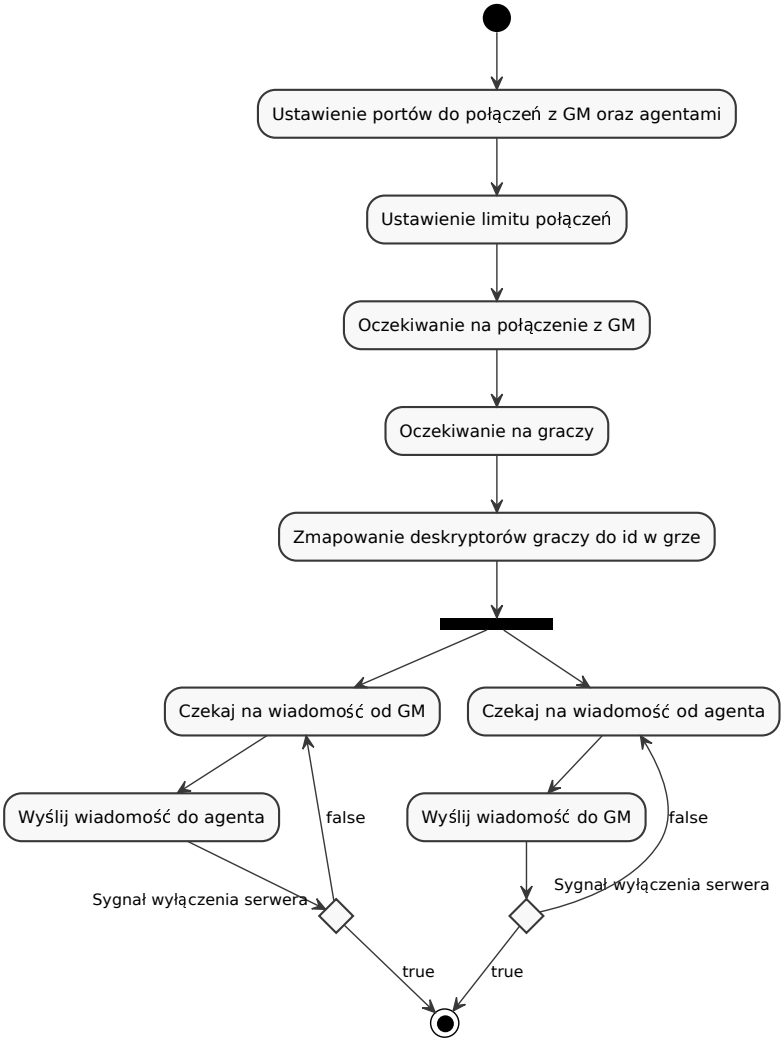
Połączenie Agenta skutkuje nadaniem przez CS unikalnego id (patrz Komunikacja), które później wykorzystywane jest w logice rozgrywki.

9.2 Połączenie z CS



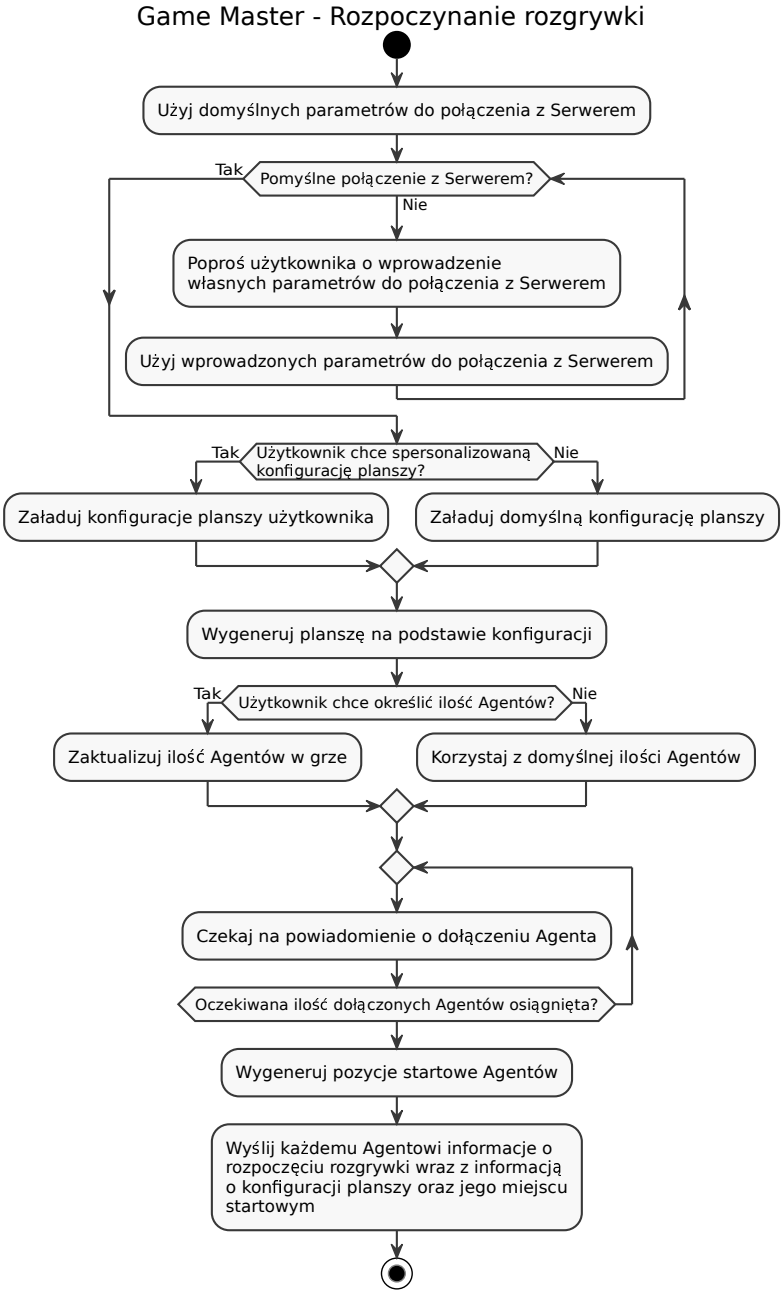
Rysunek 12: Diagram aktywności dołączenia do rozgrywki

9.3 Praca serwera komunikacyjnego



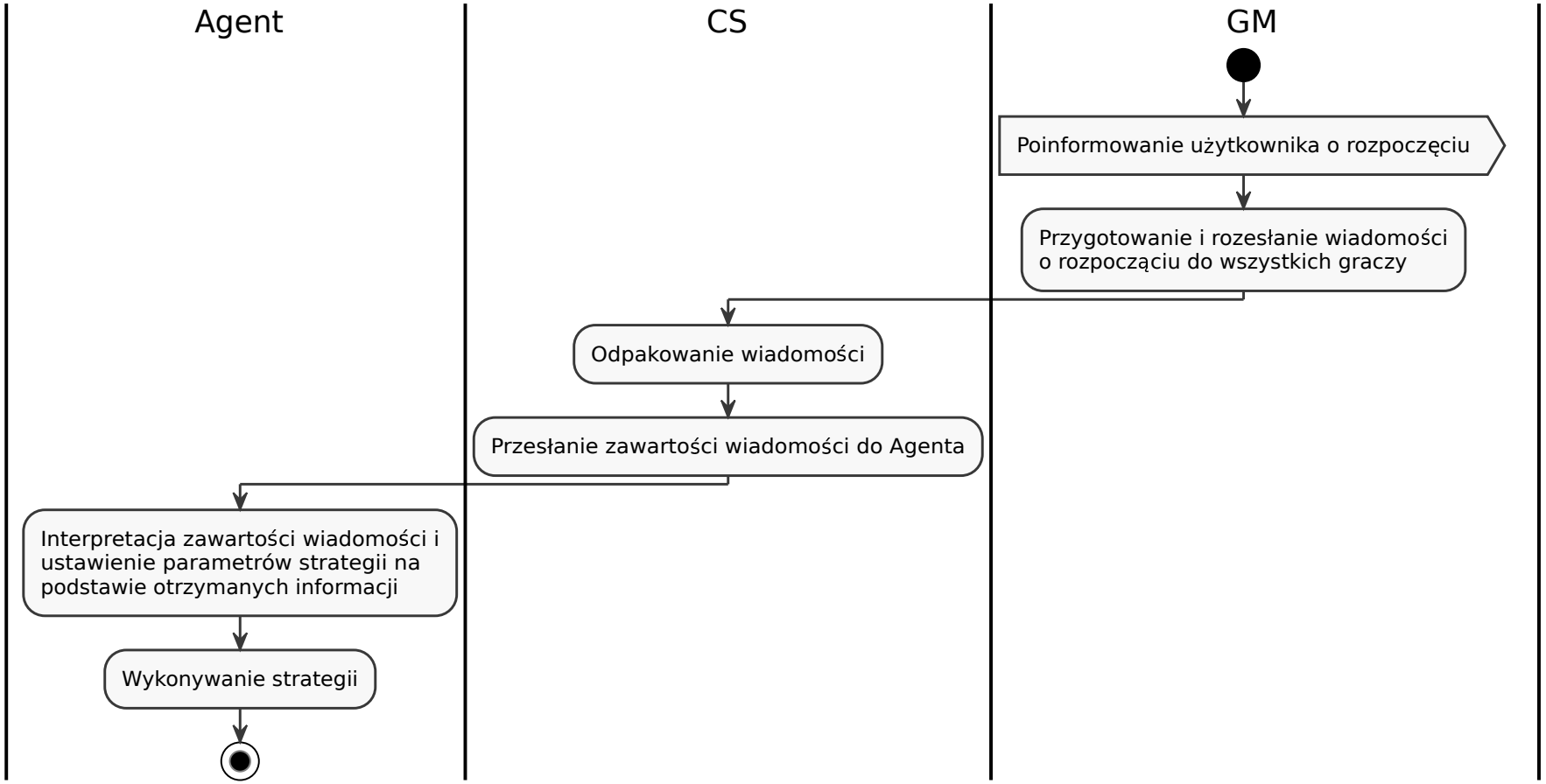
Rysunek 13: Diagram aktywności CS

9.4 Obsługa rozpoczęcia rozgrywki przez GM



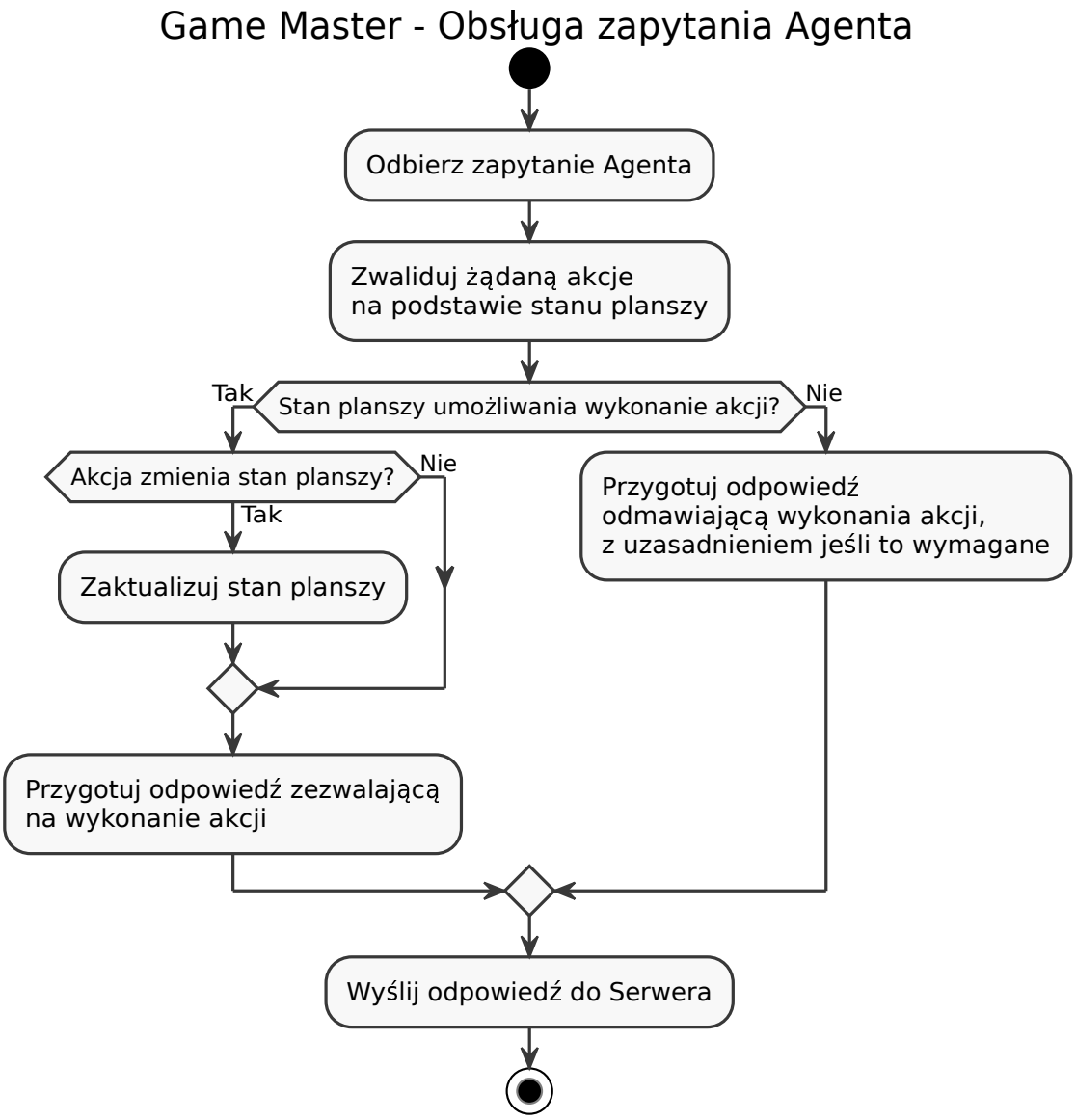
Rysunek 14: Diagram aktywności rozpoczęcia rozgrywki dla GM

9.5 Obsługa rozpoczęcia rozgrywki przez system



Rysunek 15: Diagram aktywności rozpoczęcia rozgrywki

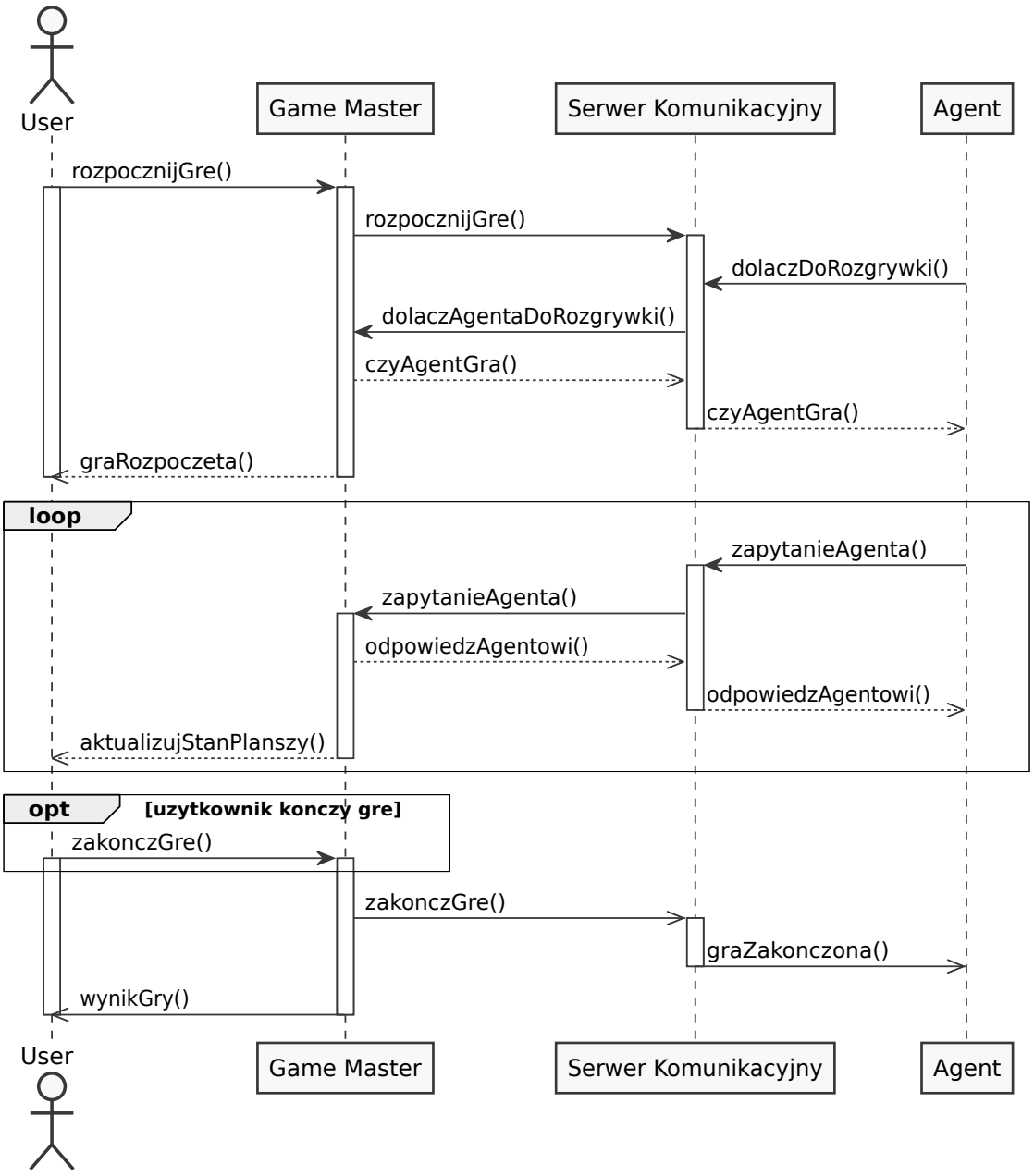
9.6 Obsługa zapytania przez GM



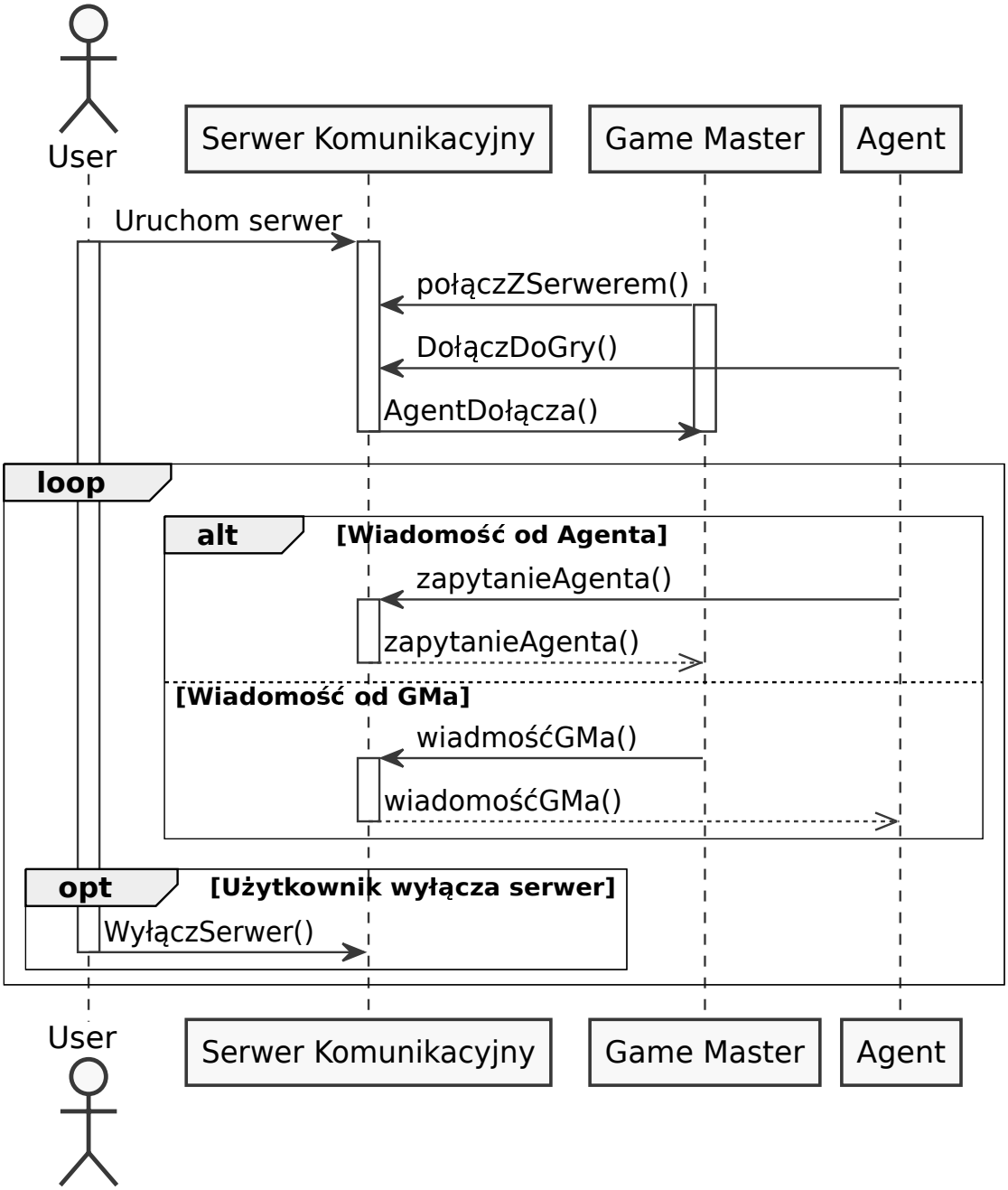
Rysunek 16: Diagram aktywności obsługi zapytań agenta

Każde żądanie Agenta jest walidowane przez GM. Sprawdza czy dana akcja nie łamie reguł, czy Agent odczekał karę czasową. Jeśli nie, przygotowuje odpowiednią odpowiedź błędu.

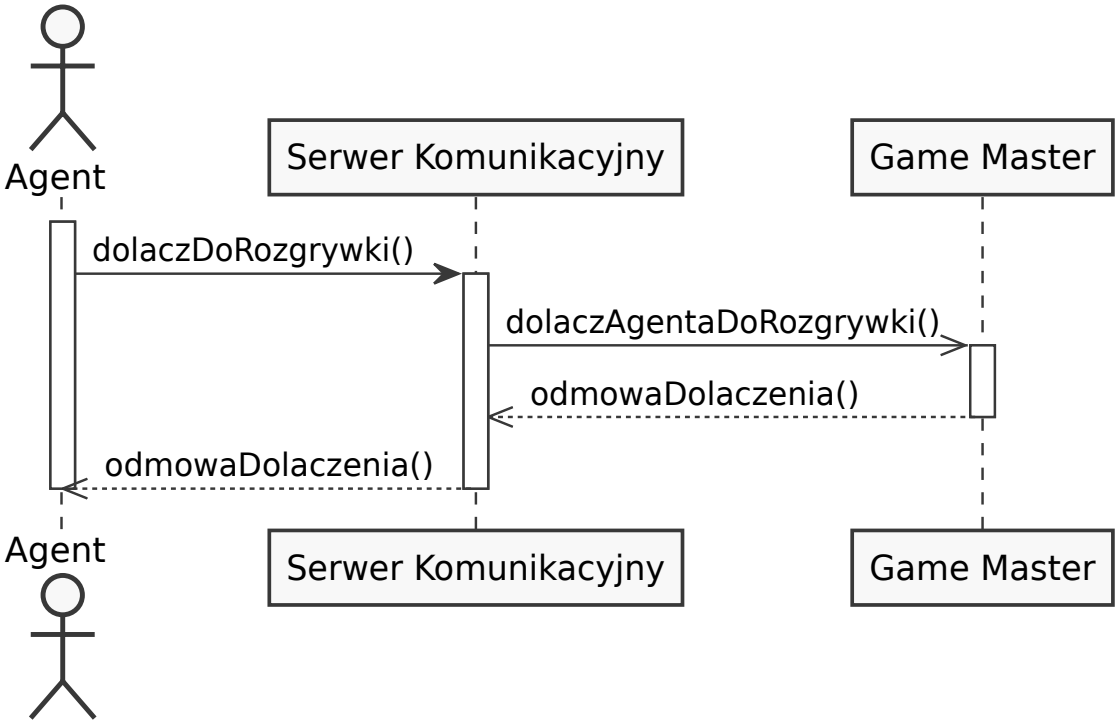
10 Diagramy Sekwencji



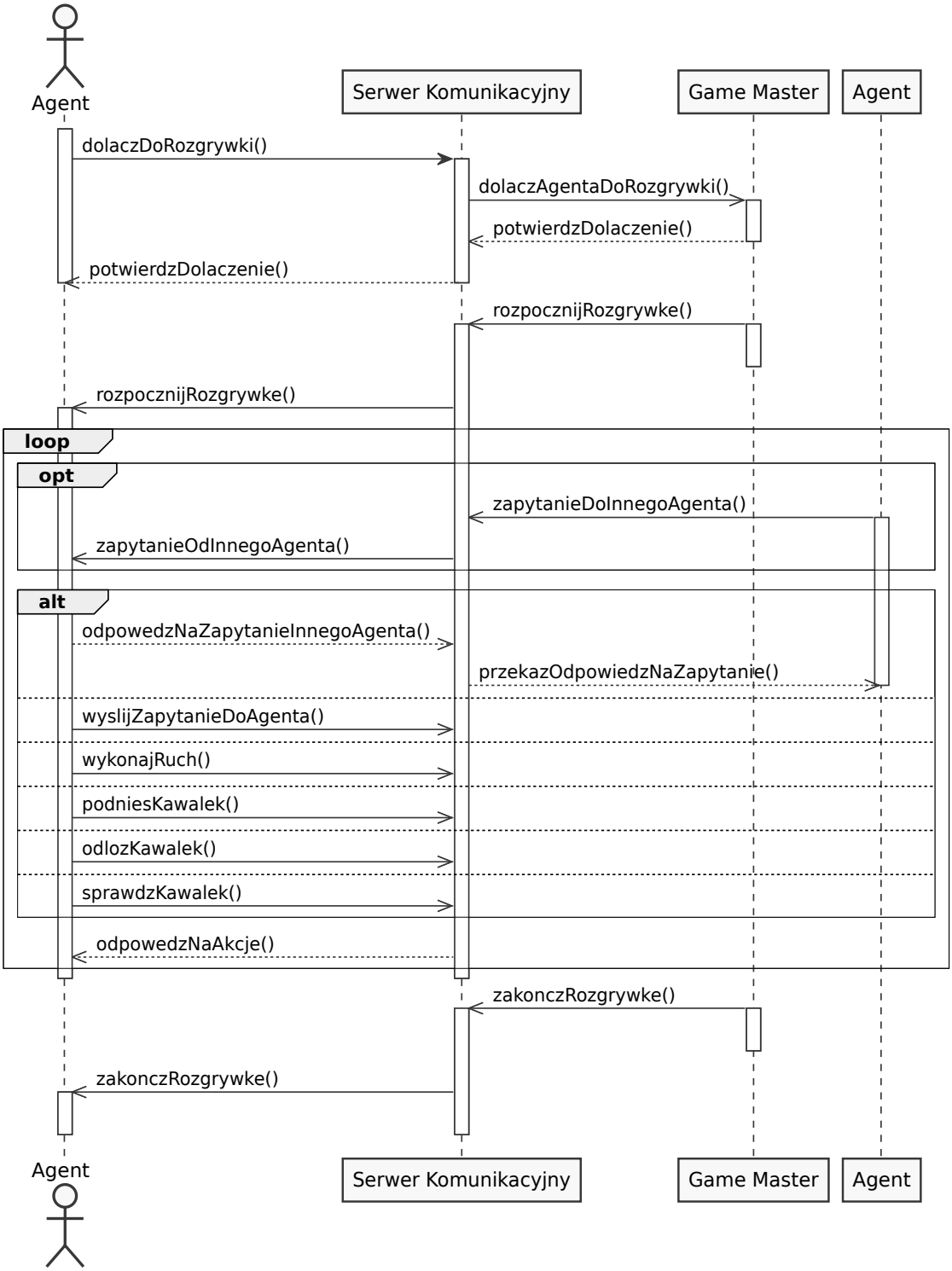
Rysunek 17: Diagram sekwencji GM



Rysunek 18: Diagram sekwencji serwera komunikacyjnego



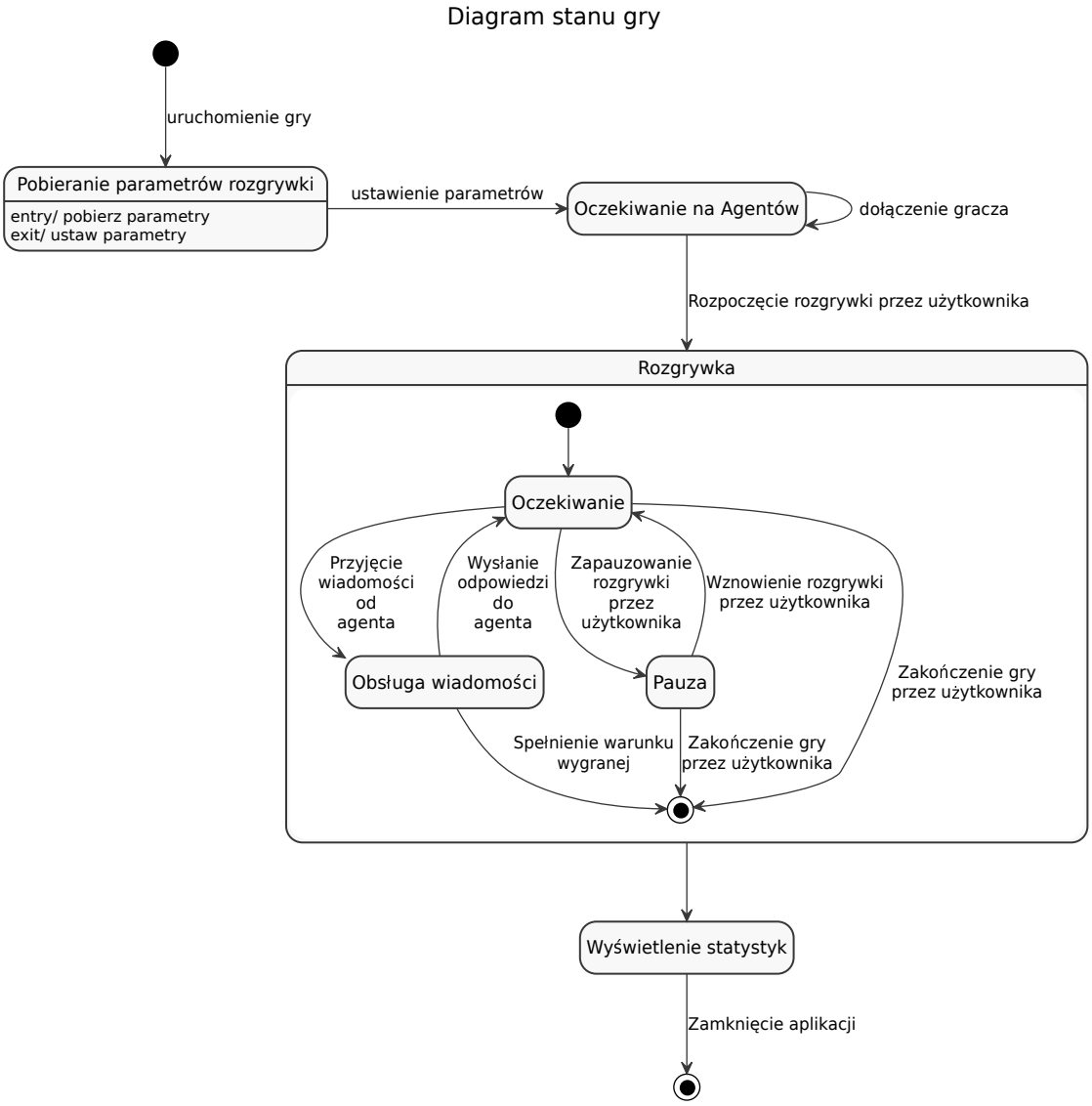
Rysunek 19: Diagram sekwencji agenta - odmowa



Rysunek 20: Diagram sekwencji agenta - akceptacja

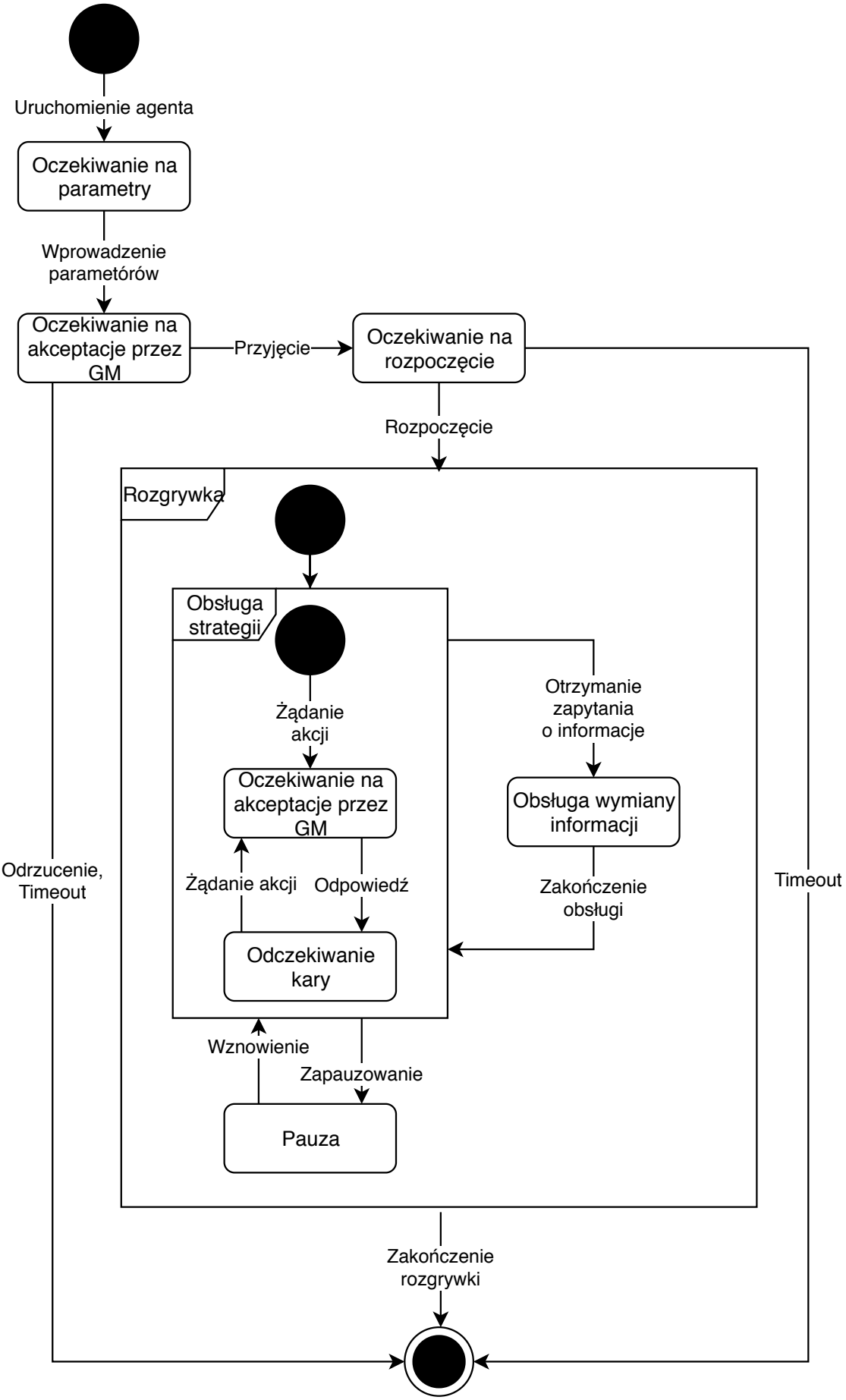
11 Diagramy Stanów

11.1 GM



Rysunek 21: Diagram stanów Game Mastera

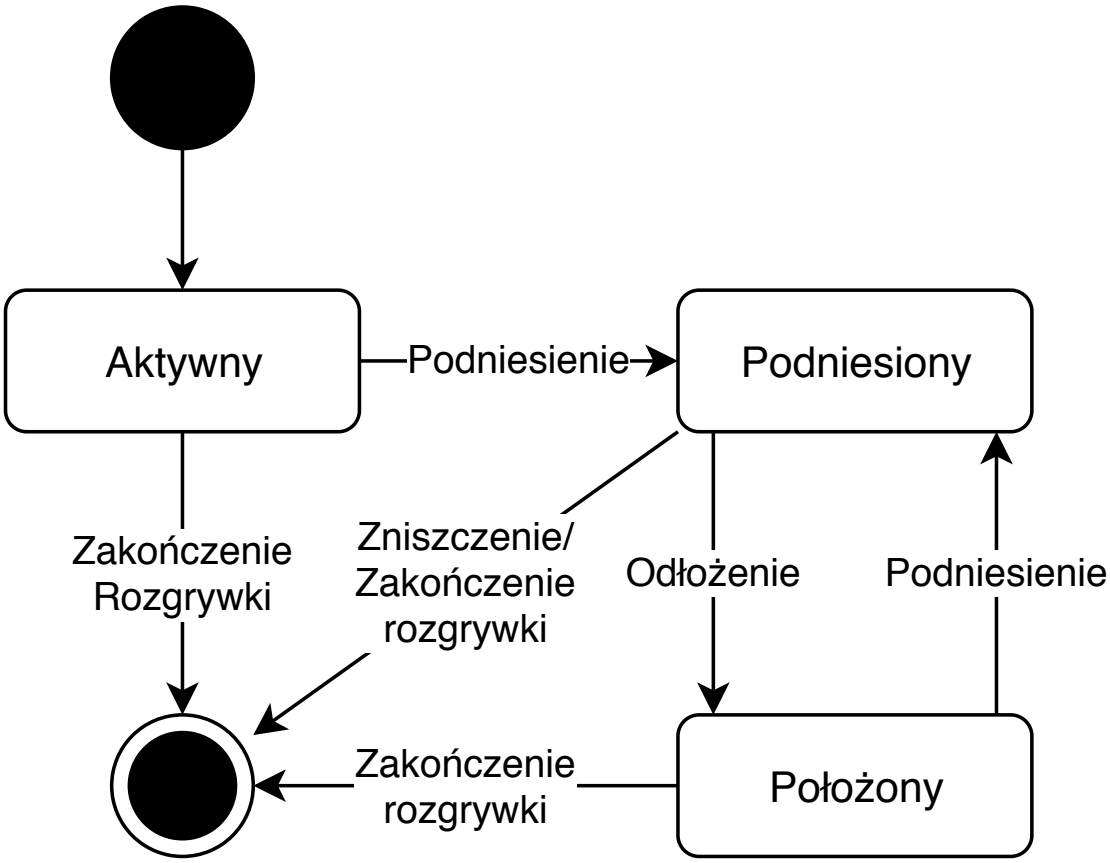
11.2 Agent



Rysunek 22: Diagram stanów Agenta

11.3 Kawałek

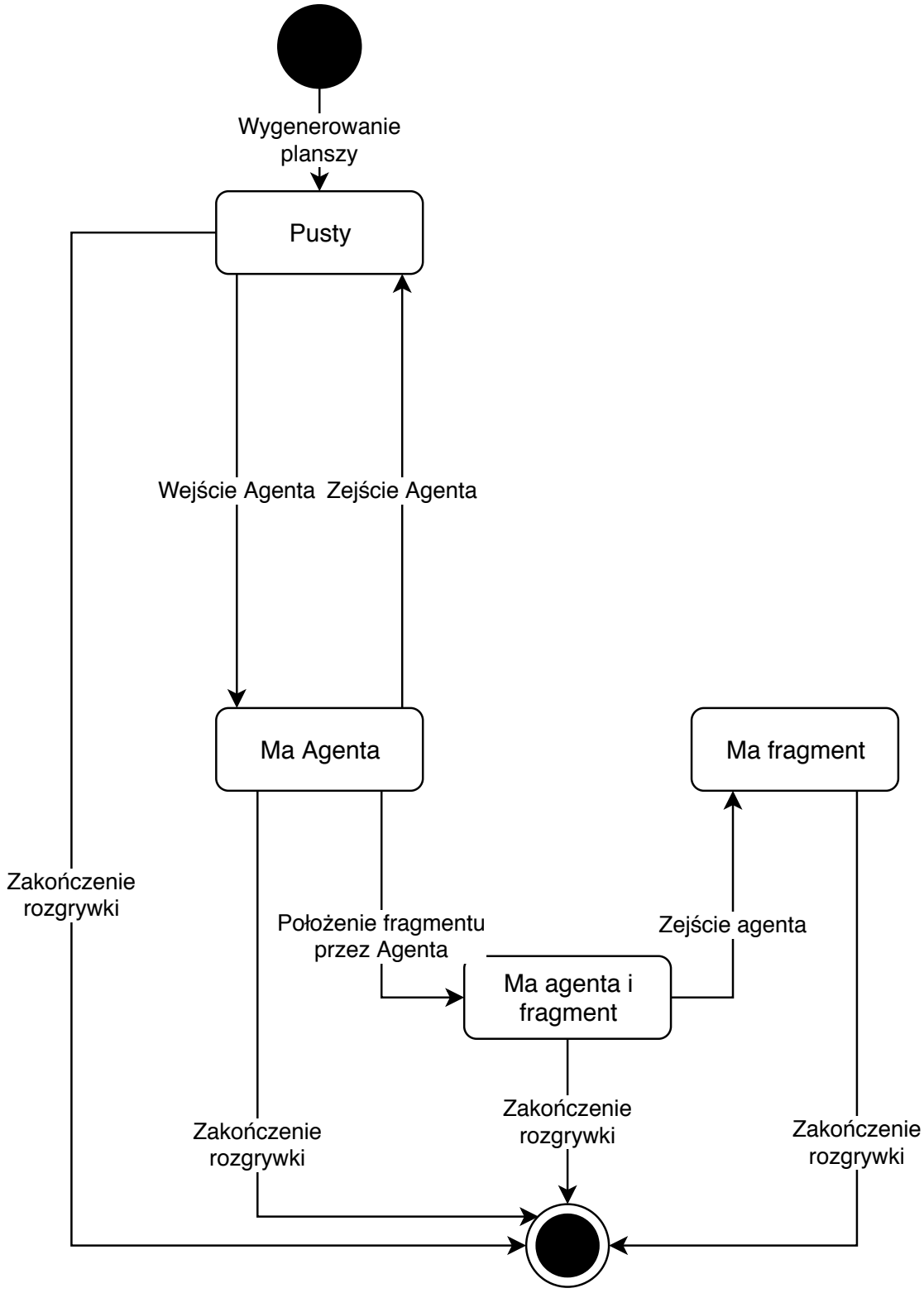
Stan kawałka może zmieniać jedynie Agent akcjami przez niego wykonywalnymi. Kawałki są usuwane (*jako obiekty*) tylko na zakończenie rozgrywki oraz na akcję zniszczenia. Odłożony kawałek nie zawsze może zostać podniesiony. O tym czy dany kawałek może zostać podniesiony decyduje typ pola na który został odłożony.



Rysunek 23: Diagram stanów kawałka

11.4 Pole bramkowe

Pole bramkowe rozpoczyna jako puste. Zgodnie z regułami raz odłożony fragment na pole bramkowe nie może zostać już podniesiony. Zatem po położeniu na pole bramkowe fragmentu, już nigdy nie przejdzie ono do stanu pustego. GM nie może wygenerować fragmentu na polu bramkowym.



Rysunek 24: Diagram stanów pola w polu bramkowym