The Project Game

Software Engineering 1

Paweł Rzepiński, Sebastian Sowik, Mateusz Stelmaszek,
Ryszard Szymański, Wojciech Trześniowski

January 28, 2018

Contents

List of diagrams

List of code listings

1 Specification

1.1 Business Description

The objective of the project is to create a game simulating project development in a cooperation and competitive multi-agent environment. The game takes place on a rectangular board divided into three areas: two goal areas(one per team) and a shared task area. The board is divided in a 1:2:1 ratio.

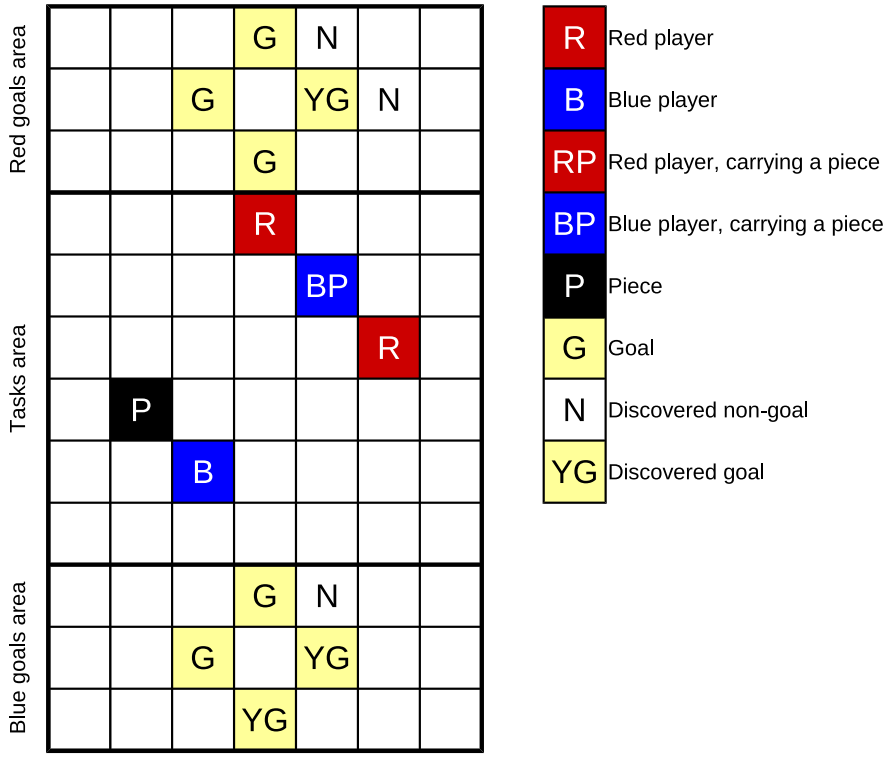

Diagram 1: The overview of game board

Players are divided into two teams: Red and Blue. Each team has a distinguished member called a Team Leader. The objective of the game is to complete a given project before the opposing team. Both squads are given an identical project to complete. In order to fulfill the objective agents have to place pieces in their team's goal area. Pieces are generated at the beginning and located within the tasks area on random positions. The communication between the Game Master and the players is handled by the Communication Server.

1.2 Functional requirements

Three actors are distinguished in the project: Player, Communication Server and Game Master. All three actors have access to specific actions which are presented as an UML use case diagram and described in tables below. A more in depth description of the actors can be found in section 2.1.

1.2.1 Game Master

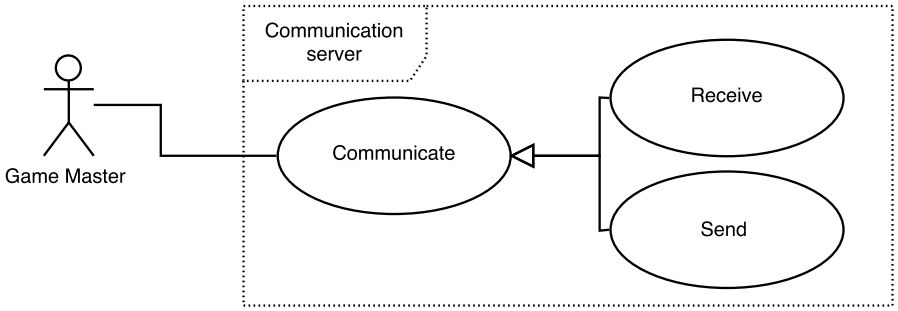

Diagram 2: Game Master Use Case diagram

Action Name	**Description**
Receive	Receives a message from the Communication Server
Send	Sends a message to the Communication Server.

1.2.2 Player

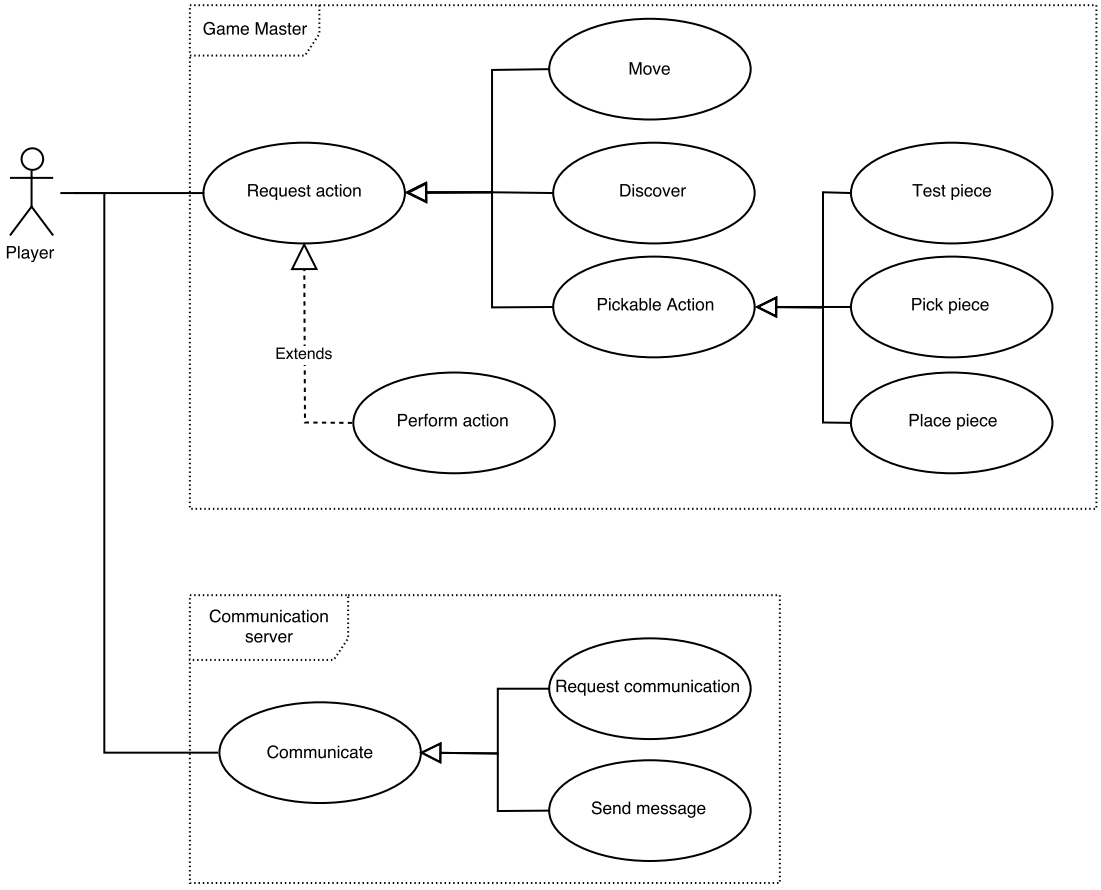

Diagram 3: Player Use Case diagram

Action Name	**Description**
Request Action	Sends an action request to the Game Master.
Move	Moves to one of the neighboring tiles.
Discover	Discovers the content of the neighboring fields.
Pickable Action	Requests an action that involves a piece.
Test Piece	Checks whether the piece is a sham.
Pick piece	Picks up a piece from a tile.
Place piece	Places a piece in the goal area.
Receive	Receives a message from the Communication Server
Communicate	Communicate with other components.
Request communication	Sends a communication request to a player.
Send message	Sends addressed message to the Communication Server.

The application requires Internet connection in order to work properly. The Communication Server is designed in a particular way which allows for an easy to implement batch extension. The specific description of the data protocol is described below in section 2.6.

1.3 Non-functional requirements

Non-functional requirements according to FURPS classification:

Type of requirement	No.	Description
Usability	1	All of the game functionality available on desktops with a 1920x1080 screen resolution.
	2	Visualization of the current game state from the Game Master's perspective.
	3	Basic English knowledge is required from the user.
Reliability	4	Connection errors handling by a mechanism of timeouts and reconnection attempts.
	5	Resistance to communication errors by message schema checking.
	6	In case of any component failure timeouts are used to guarantee safe system shutdown.
Performance	7	Maximum message latency of 3 seconds on a $1\frac{mb}{s}$ connection is guaranteed for no more than 16 active players.
Supportability	8	Communication and error logs stored in a .txt file locally for each system component.
	9	Game master configuration with game parameters stored in a .txt file.
	10	User manual for each component delivered along with the system.

1.4 Components' runtime parameters and input data format specification

Components runtime parameters are stored locally in XML with UTF-8 encoding.

1.4.1 Game Master runtime parameters

Game Master's configuration file fields description:

- `BoardWidth` - goal side length
- `BoardHeight` - goal-task-goal side length

- `ServerAddress` - IP address of the Communication Server
- `ReconnectionAttemptsCount` - number of reconnection attempts before proceeding to shutdown
- `ReconnectionAttemptInterval` - time interval between reconnection attempts (in seconds)
- `PlayersPerTeam` - number of players in one team
- `Pieces` - number of pieces
- `ShamsPercentage` - percentage of shams in pieces (in percentages)
- `GoalsPositions` - goals positions in bottom team; arrangement is mirrored accordingly for the top(Red) team

Listing 1: Game configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<GameConfiguration>
  <BoardWidth>20</BoardWidth>
  <BoardHeight>40</BoardHeight>
  <ServerAddress>192.168.1.1</ServerAddress>
  <ReconnectionAttemptsCount>5</ReconnectionAttemptsCount>
  <ReconnectionAttemptInterval>20</ReconnectionAttemptInterval>
  <PlayersPerTeam>10</PlayersPerTeam>
  <Pieces>20</Pieces>
  <ShamsPercentage>25</ShamsPercentage>
  <GoalsPositions>
    <Position>
      <X>0</X>
      <Y>0</Y>
    </Position>
    <Position>
      <X>1</X>
      <Y>0</Y>
    </Position>
    <Position>
      <X>2</X>
      <Y>0</Y>
    </Position>
    <Position>
      <X>3</X>
      <Y>0</Y>
    </Position>
  </GoalsPositions>
</GameConfiguration>
```

1.4.2 Player parameters

Player's configuration file contains only team preference: Blue or Red.

Listing 2: Player configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<PlayerConfiguration>
  <TeamPreference>Blue</TeamPreference>
</PlayerConfiguration>
```

2 Documentation

2.1 System architecture

The Project Game system consists of three main components: Game Master, Communication Server and Players. At the beginning of the game the players and the Game Master establish a connection with the Communication Server in order to communicate using a common message protocol.

Each of the components is following event-driven architecture pattern. Game components are descendants of MessageReceivedEventListener class in order to allow them to implement their own version of the Message Received event handler. Game Master and Players implement ICommunication interface containing Send method, which defines their way of addressing messages and the process of sending them to Communication Server.

Communication Server handles the communication by passing messages between the Players and the Game Master without processing content of the messages.

Game Master is responsible for managing the game state by responding to the Players action requests and changing the global state of the game and the board accordingly to action definition.

Players interact with the Game Master and each other in order for their team to complete the game objectives as fast as possible.

2.2 System components description

2.2.1 Game Master

The first task of the Game Master, after all participating players has joined the game session, is creating the game board and placing game related objects accordingly on the board. Game related objects consist of Pieces, Goals and Pawns(that represents each Player).

After setting up all of the board objects Game Master informs all Players about the board dimensions and team division. Additionally every Player receives information about their initial position on the board.

Game Master receives action requests from Players during the game and responses in accordance to the current board configuration and updates the board state afterwards. Before responding Game Master checks if the action is in compliance with the game rules.

2.2.2 Communication Server

The role of Communication Server is to manage communication between the Players and the Game Master. It mainly consists of passing the messages to the appropriate addressee without processing their content.

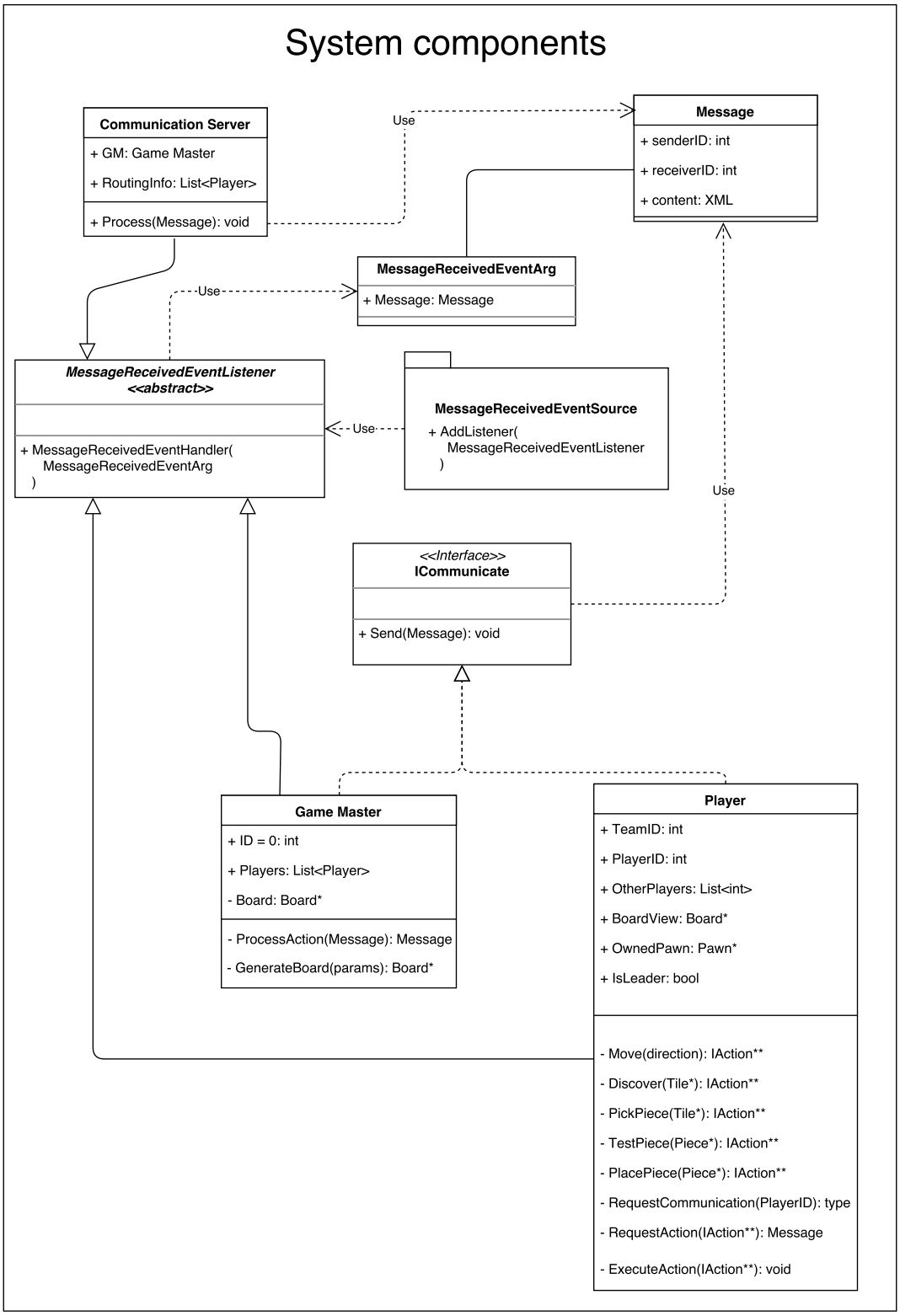

Diagram 4: Class diagram of system main components

*objects described in game related objects diagram
**objects described in players actions diagram

2.2.3 Players

Player is an agent participating in a game. The Project Game consists of many Players divided in two teams. Every Player holds its own view of the state of the game and their goal is to cooperate with other team members in order to assure their team victory by completing all the goals before theirs opponents.

Players can send requests the Game Master to take an action allowed by the game rules. Completion of the requested action depends on the decision of the Game Master and cannot take place without his approval.

Actions allowed for Player are described by game rules in section 2.4. Player's actions are modeled as objects, following abstract method design pattern. RequestAction method creates message object in appropriate format(described in protocol in section 2.6) that can be later sent to Game Master using Send function(from ICommunicate interface). Message creation utilizes CreateMessage() method from IAction object as in Inversion of Control design pattern. ExecuteAction function updates player's board view after processing Game Master confirmation response.

2.3 Game related objects

Players are divided into teams that include a distinguished member called Team Leader. Team Leader must agree to request for information exchange originating from his own team's players.

The game is played on a board that is divided in three zones: red team goal area, blue team goal area and task area. Fields in goal area may contain Goals, which Players have to discover and complete them (by placing a task in it) in order to win. The rules of completing a task are described in game rules section 2.4. Every team has its goal area which contains goals for them to complete.

Pieces, goals and pawns are situated on specified positions on board by the Game Master. This objects features are reflected by the inheritance hierarchy and object relations shown in the diagram below. The piece could turn out to be a sham or used to complete goal (more details are described by game rules in section 2.4).

Every field of the board is represented by an abstract class named BoardObject. Collection of those objects is stored by Board in the form of a matrix. The BoardObject contains fields X and Y representing Cartesian position on the board.

BoardObject's descendant Tile extends its parent with fields ContainedObject, PlayerPawn, TimeStamp and DistanceToNearestPiece. Tile may also contain one of TileObject's descendants: Piece or a Sham. Field TimeStamp allows to determine which TileObject contains the most recent information about game state. There are two descendants of Tile class which correspond to different parts of board GoalAreaTile and TaskAreaTile.

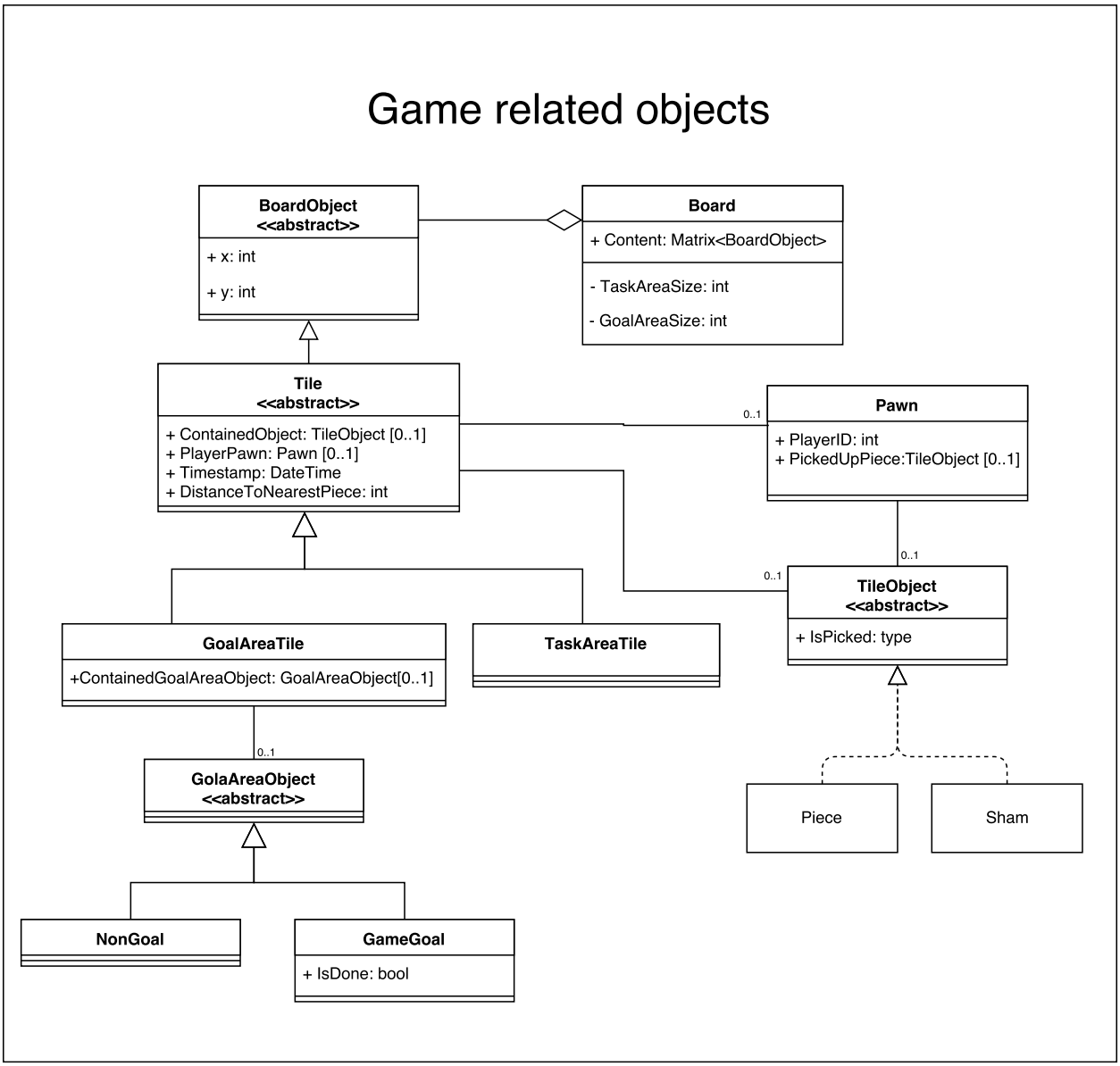

Diagram 5: Class diagram of game related objects

GoalAreaTile represents Tiles in the goal area which may contain GameGoal or NonGoal. Boolean field IsDone in the GameGole class has meaning only for the Game Master. NonGoal object may appear on the Players board when he receives information from Game Master, that his attempt to place a piece was meaningless. Tiles in the task area are modeled by TaskAreaTile class.

PlayersPawn represents Player's position on the board. PlayerID field enables to recognize which Pawn belongs to which Player.

Player actions are represented using combination of Abstract Method and Inversion of Control design pattern. This object modeling is shown in Player's actions model diagram. CreateMessage method is abstracted to allow implementing custom conversion to Message object and action's execution logic(and they way it changes board state).

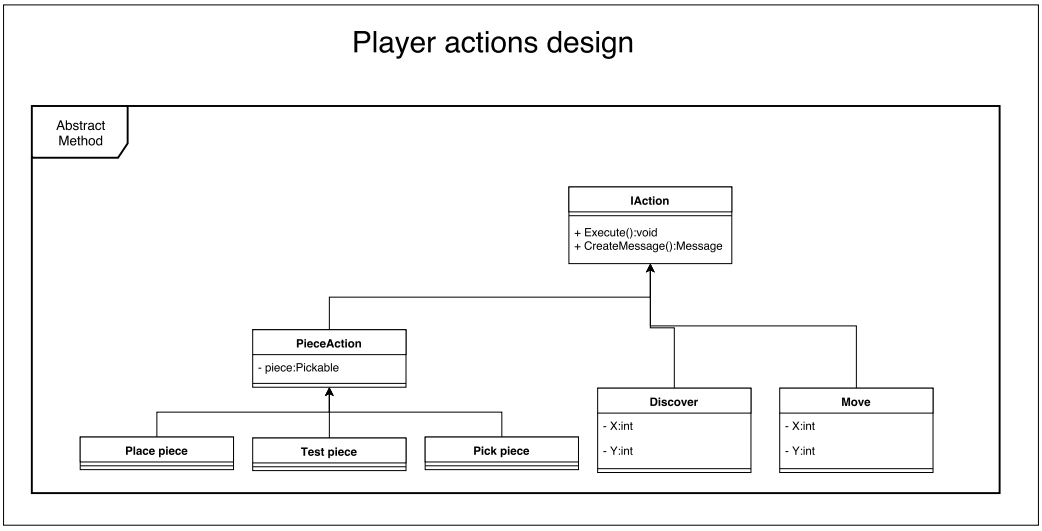

Diagram 6: Class diagram showing how Player actions are modeled

Different states of the piece are required because it models a Task concept. In real life pursing a task is often associated with failure of completing it. Therefore, a piece can be in fact a sham. Testing action can be used to determine whether a piece is valid.

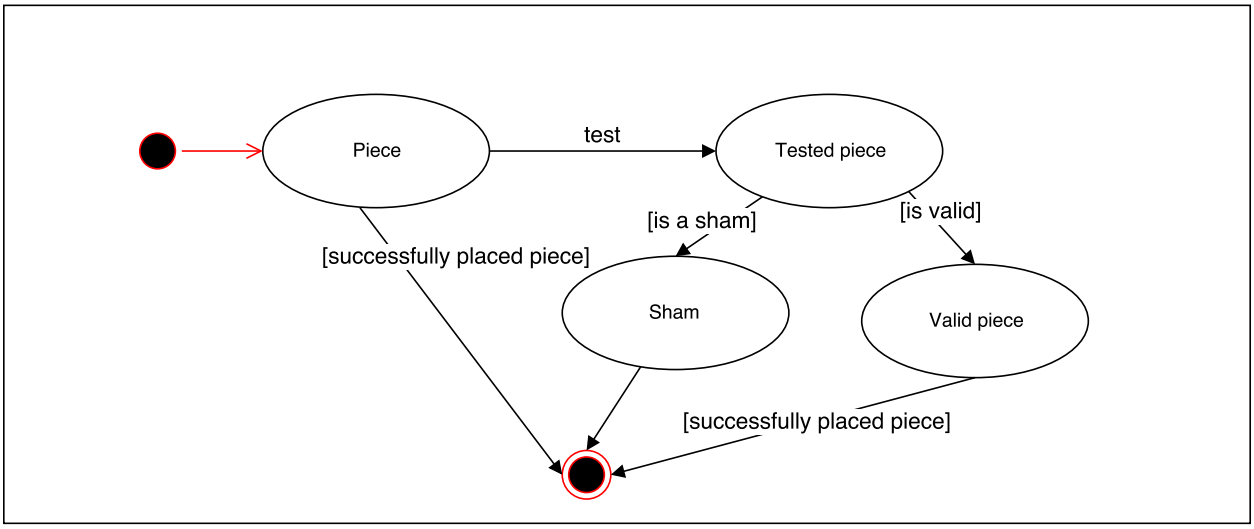

Diagram 7: State diagram showing possible states of the piece

During the game Player discovers new Tiles and gains new information about the board state. Tile may store a Piece, Player's pawn or may be empty. TaskAreaTile states diagram describes those features.

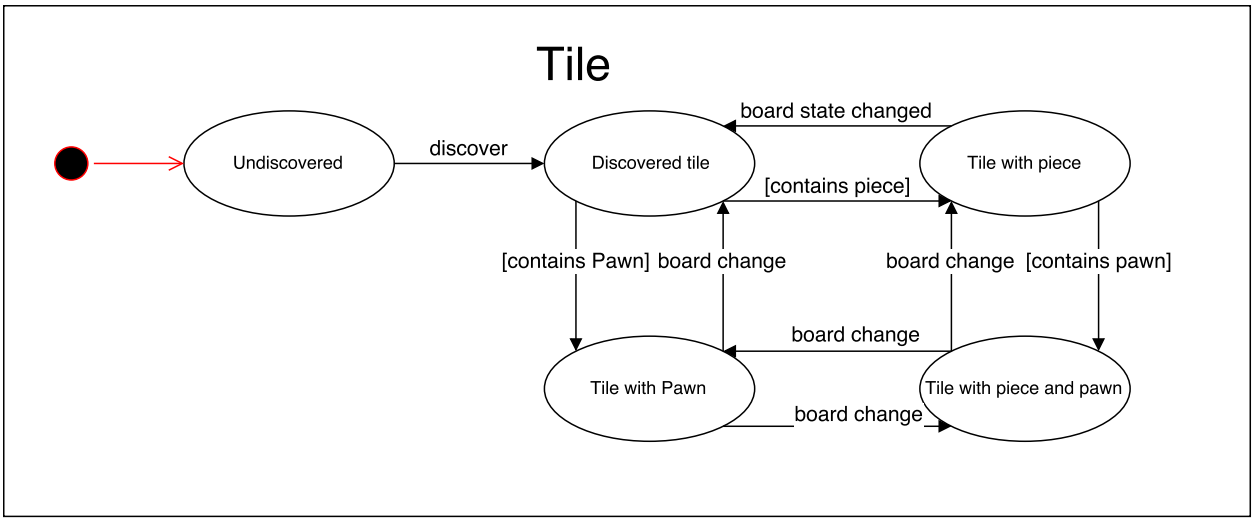

Diagram 8: State diagram showing possible states of the TaskAreaTile

GoalAreaTiles are used by the Players in attempt to complete the goal by placing a valid piece. Success of this action depends on type of this Tile - Goal or NonGoal.

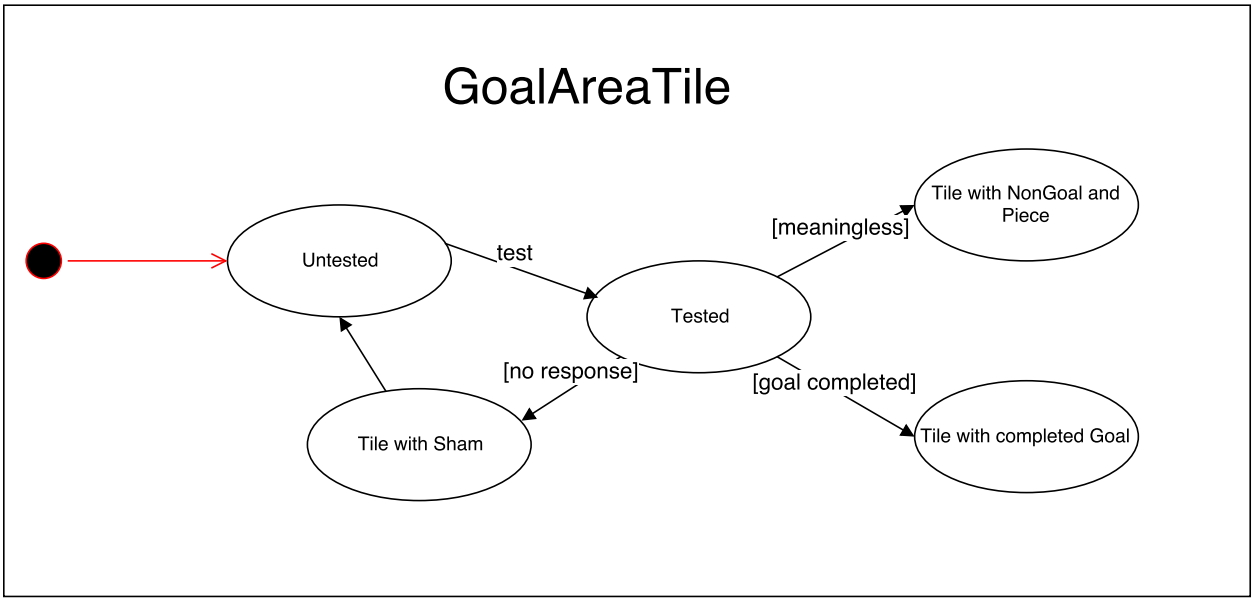

Diagram 9: State diagram showing possible states of the GoalAreaTile

2.4 Game rules

The game is played on a board that is divided into three zones: Red team goal zone, Blue team goal zone and task area. The true state of the fields in the goal areas is known only to the Game Master. The Player can learn about them only by placing (using) a piece in a given field of the goal area:

- A correctly placed piece results in an information to the Player that one of the goals of the project have been completed.
- Incorrectly placed piece results in an information that the completed action (getting the piece from the tasks board and placing it in the goals area) has been meaningless, in the sense of project completion.
- Placing a piece which is a sham results in getting no information.

Possible moves of the Player consist of:

- moving in one of 4 directions,
- discovering the contents of 8 neighbouring fields *,
- testing the picked piece for being a sham,
- placing a piece in the goals, in hope of completing one of the project objectives,
- request exchange of information with another player.

Player actions have following ramifications and constrains:

- One player can carry only one piece. **
- Piece pick up is a separate action, it does not occur automatically when a player moves into a tile that contains piece. ***
- Player cannot move into a field occupied by another player.
- Observing a field (either by discovering or entering it) results in receiving information about the Manhattan distance to the nearest piece.

2.5 Game interactions

Game interactions between system components can be divided into 3 phases: game initialization phase, game phase and game finalization phase.

Game initialization phase

The Game Master establishes a connection with the Communication Server. Before that connection is made, all player requests are rejected. The server registers the Game Master and starts to allow players to join in. When agents are connected, the Game Master can start the game. The

*In case of GoalAreaTile this action does not reveal positions of Goals

**Additional ramification added for simplicity

***Additional, replaces "The piece is picked up by a Player which moves into the piece's field first."

Game Master sends the initial state of the Game Board and information about the teams to all participants.

Game phase

When the game is running, players' actions depends on their knowledge about the board and location of potential goals.

Game finalization phase

The Game Master notifies players about the results - game statistics or failure description(in unexpected error situation). All system components disconnect from each other.

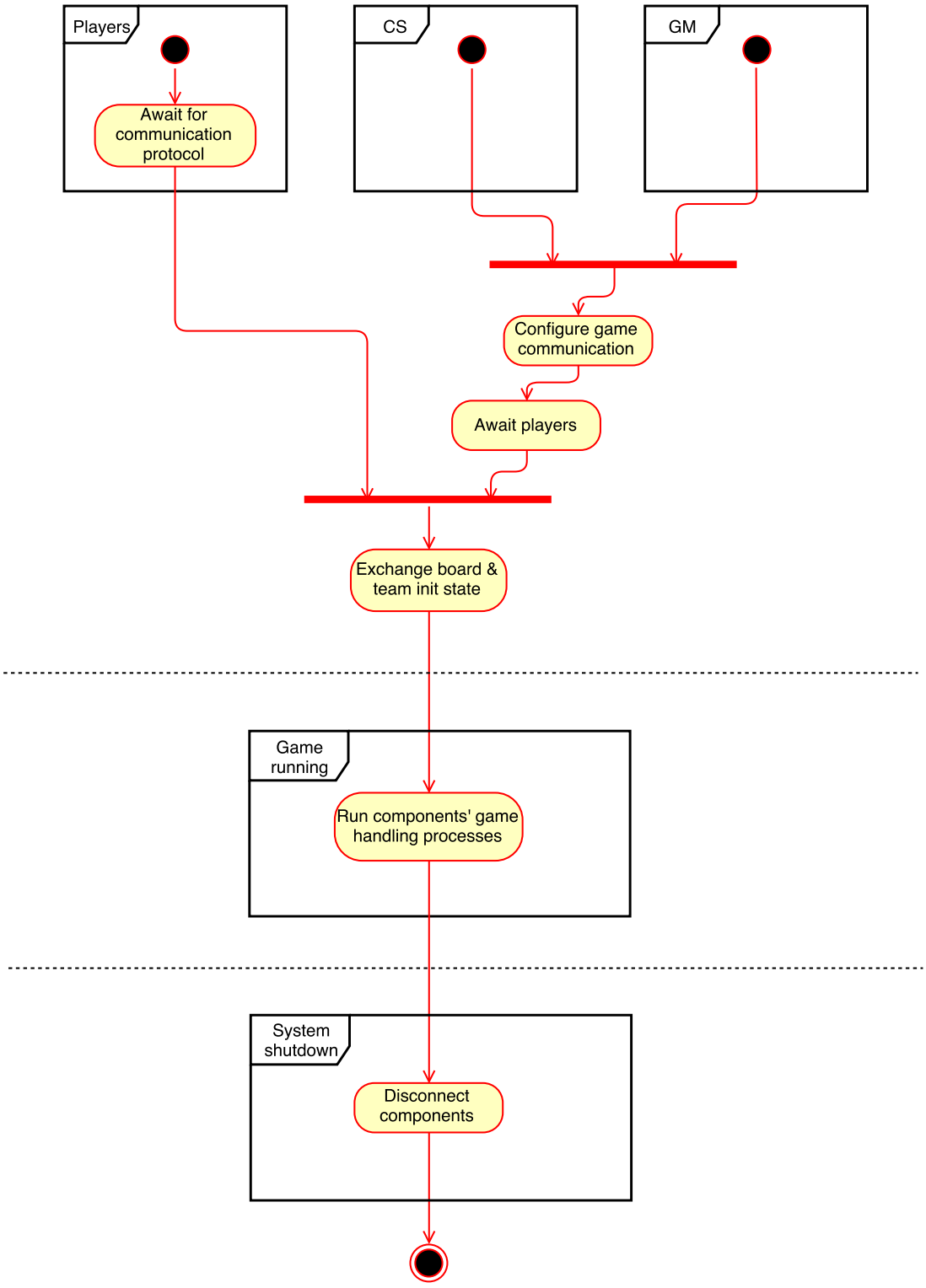

Diagram 10: Activity diagram presenting game flow

Game master interactions

During the course of the game the Game Master awaits new messages to process. Whenever a message arrives, the system waits to receive the whole request and then rebuilds it to the understandable format and processes it. If the structure of the message is incorrect or the player requests an action that is unavailable to him the Game Master builds a message in which it denies the desired action. Otherwise, the request is accepted, and in case it is necessary, the game state is updated and an accept message is built. Next the messages are sent through the Communication Server to the players and the system waits for new requests. When the Game Master detects that the game is finished, it sends a message about the end of the game to all participants and proceeds to shut down the game.

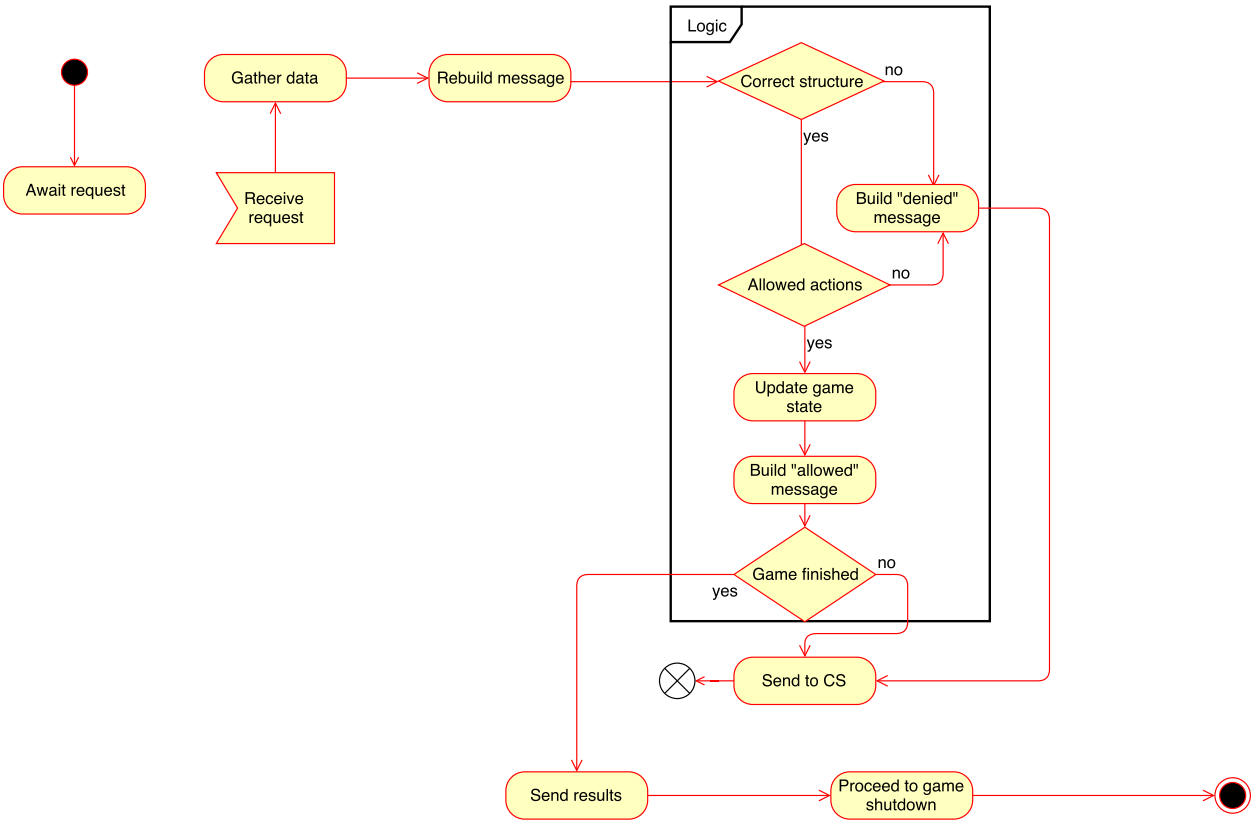

Diagram 11: Activity diagram for Game Master

Player actions and strategy

During the game, the main goal of the player is to find pieces as fast as possible and to exchange as many information as possible with other team members. Firstly, he checks whether there is an available task. If there is not task available the player carries on exploring the game board. The exploration consists of three actions: moving randomly, discovering the tiles around him and exchanging information with the team. If there is an available task, the player moves towards it, then picks it up. If the pick is unsuccessful or after checking the piece it turns out that the piece is a sham, the player starts the whole process again from checking if a task is available. Now the player has a piece and exchanges that information with the team. If there is an available goal, he moves towards it, otherwise, he has to explore the base in order to find it. When the player reaches the tile where the goal is located, he places the task in it. The whole process starts again from checking whether a task is available.

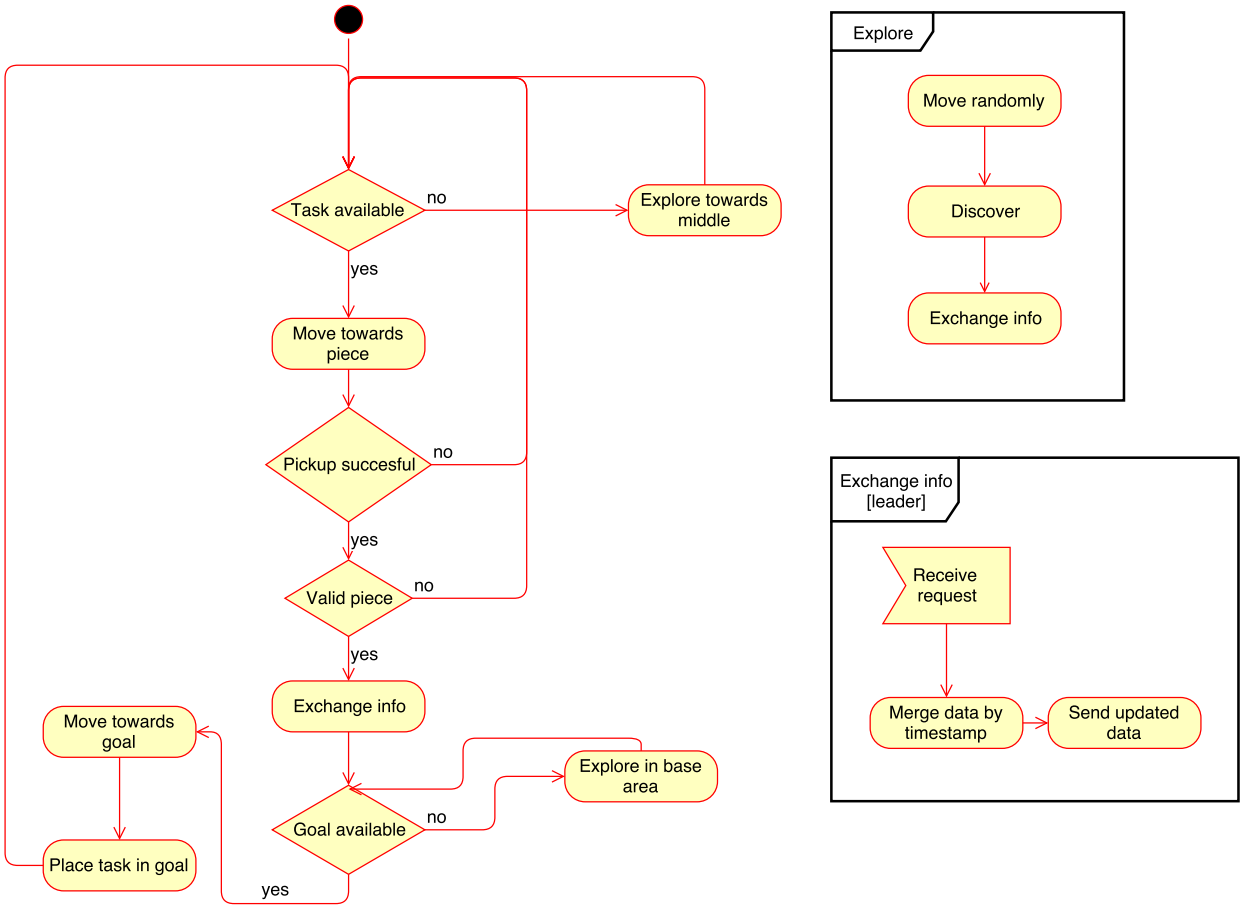

Diagram 12: Activity diagram for the player

2.6 Communication Protocol

The protocol is designed for asynchronous communication in a queued request-response model. Communication between system components is performed using XML-formatted messages encoded in UTF-8. Their design is specified and validated by XML Schemas. Usage of UUID* allows message differentiation and identification. Moreover, support of batched messages is included for extensibility and network performance.

Exact communication flow is depicted in flow diagram. The game consists of four phases:

1. Connection establishment phase

   The Game Master connects to the Communication Server and allows the Players to join the game.

2. Game Start phase

   Players send their team preferences to the Game Master and await teams assembling completion. Then, the Game Master sends the initial board state and starts the game.

3. Game process phase

   Players send requests to the Game Master in order to perform actions and communicate with each other to complete the game objective.

4. End game phase

   The Game Master finishes the game and notifies players about the results - game statistics or failure description(in unexpected error situation).

All participants are provided with the IP address of the Communication Server. Communication initiates with the connection establishment phase in which all participants pass their hash(UUID) for identification purposes(during the connection handshake). It also allows for basic authentication - Communication Server and Game Master could be provided with a predefined set of hashes, that would allow for identifying incoming connection as originated from the Game Master.

Firstly the Game Master connects to the Communication Server. Then the Players can start contacting the Communication Server in order to report their IP address and to receive identification numbers from the Game Master(ID 0 is for Game Master, 0+ for Players) that will be used during the game.

The game state is propagated using `GameState` messages, which can be divided into three categories:

1. `Setup` - regarding team preference and team leaders assignment
2. `Start` - used to start the game by passing the board and teams configuration
3. `End` - notifying about the game finish and results announcement

During the game Players can use one of the following message types:

1. `Action` - to request any possible action described in player's functions diagram
2. `ExchangeKnowledge` - to perform information exchange between them and any other players(flow according to game rules in section 2.4)

*UUID format description: http://www.ietf.org/rfc/rfc4122.txt

General remarks about protocol implementation:

- During communication errors in processing messages result in omitting the message. Thus, timeouts should be set for every participants communication functions.
- Players can report their team preference in setup phase, but the team assignment is not guaranteed to be the same as requested.

2.6.1 Extensibility

Protocol contains several additional fields to support some basic extensions:

1. `protocolVersion` - Allows protocol versioning
2. `senderHash` - Participants could use this field when sending messages to sign them and allow Communication Server to confirm their identity. In such case, the Communication Server would also be responsible for stripping out that hash before forwarding the message.
3. `contextId` - The Players and the Game Master could utilize it to simplify matching request and response message pairs. Every response message should contain id of request message, thus allowing player to recognize to which action request is response referring to.

2.6.2 Communication sequence diagrams

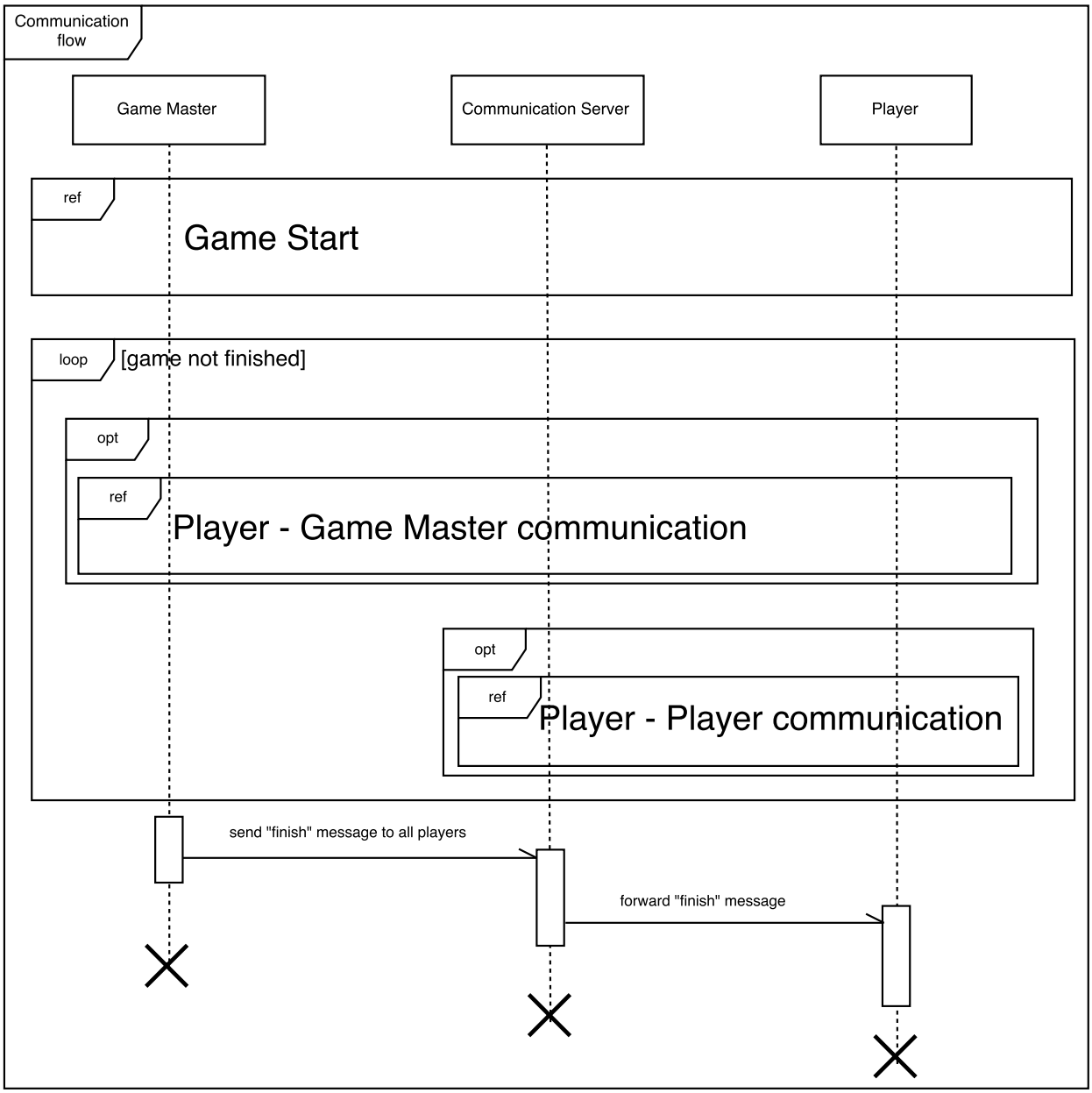

Diagram 13: Communication flow during game

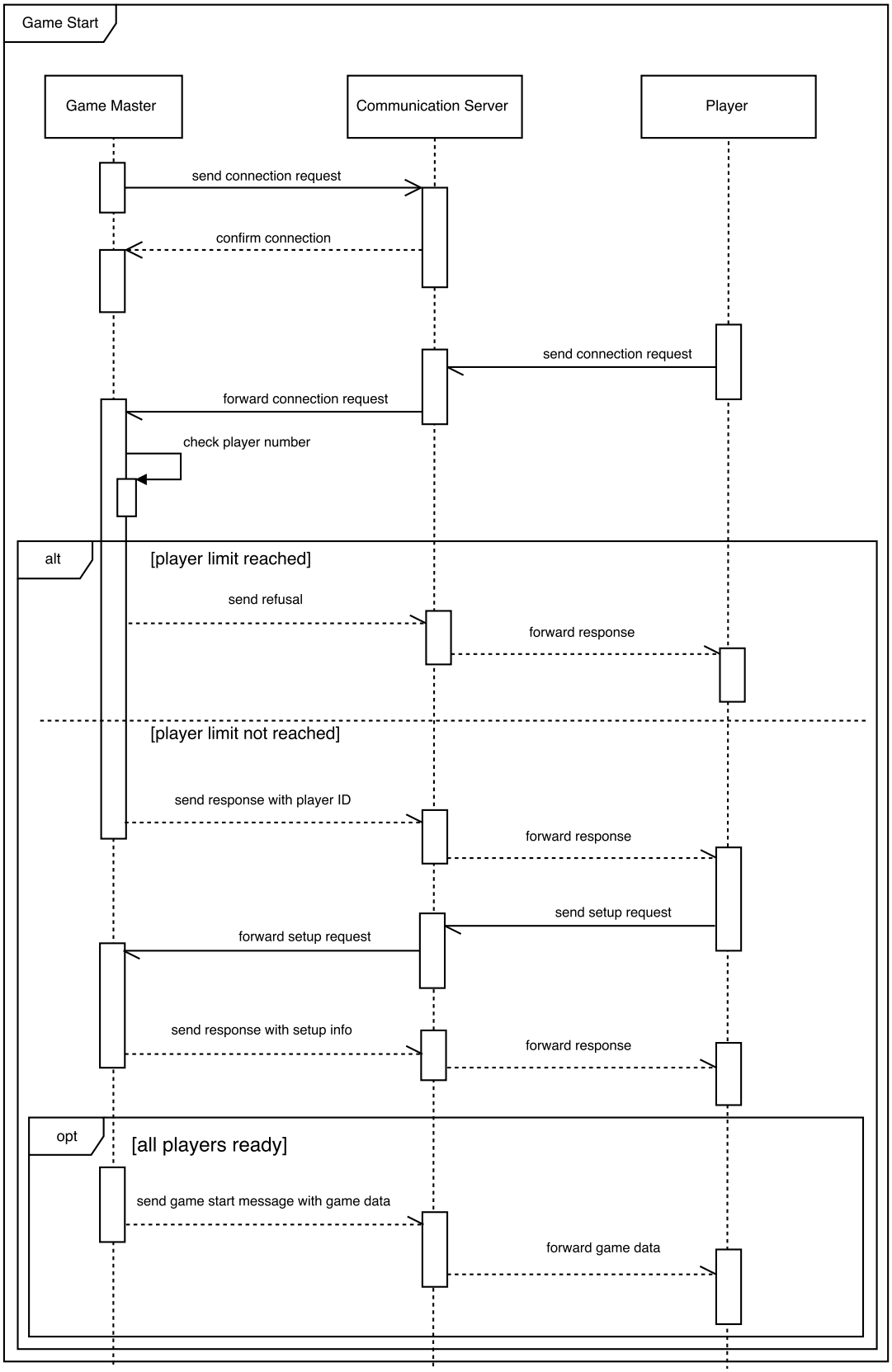

Diagram 14: Establishing connection and exchanging setup messages

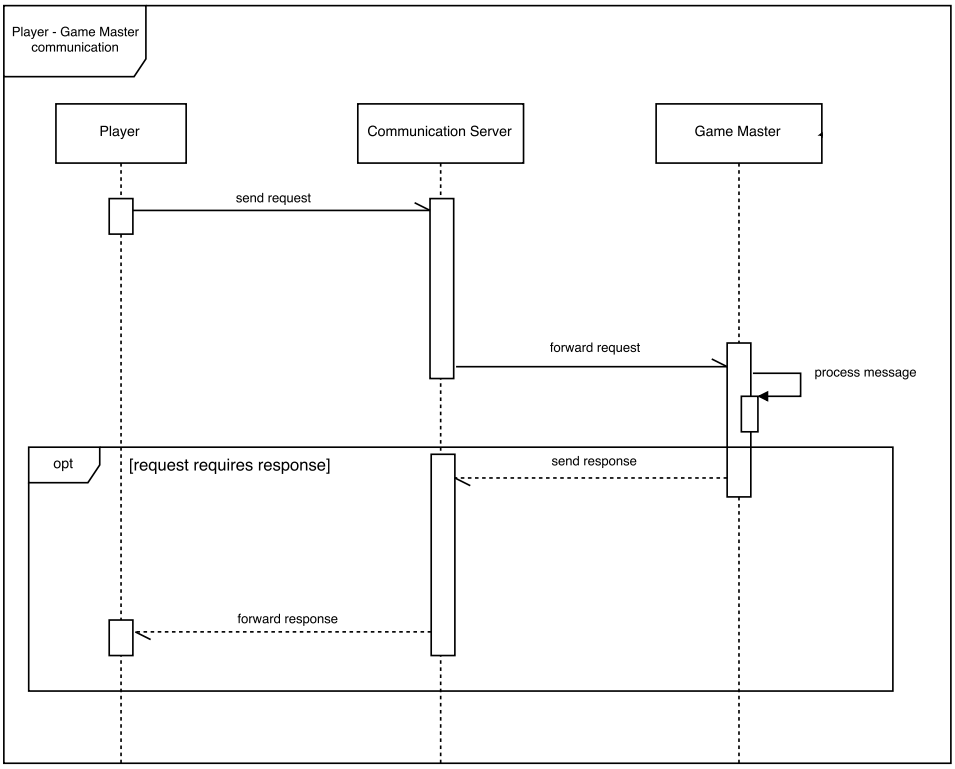

Diagram 15: Communication required to perform action by particular Player

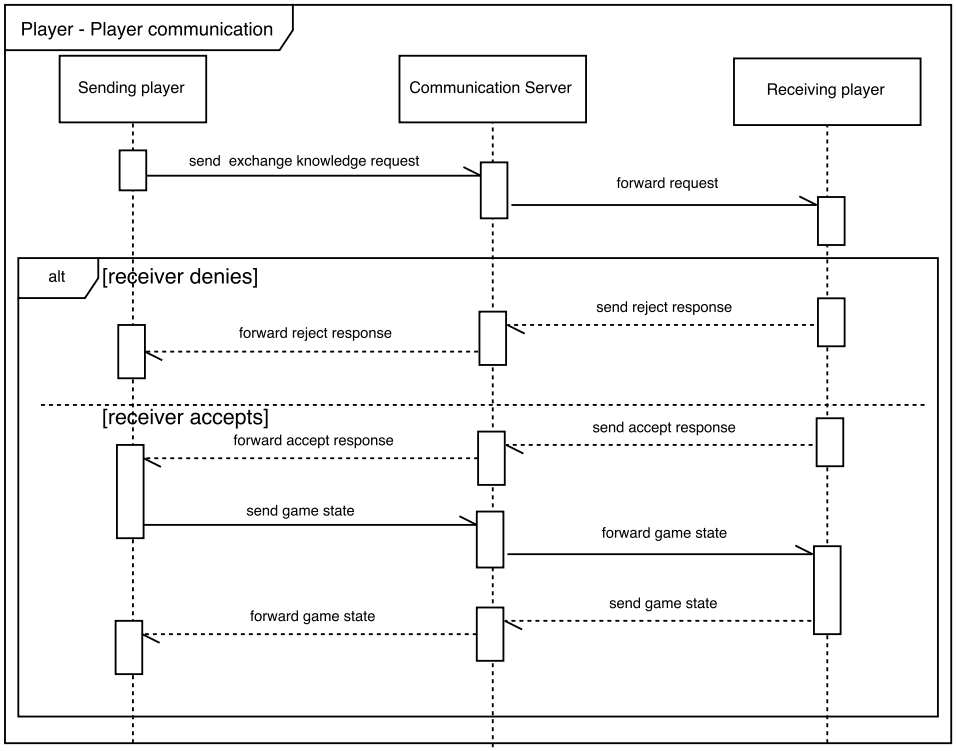

Diagram 16: Exchange of information between two players

2.6.3 Messages examples

Listing 3: Example of connection messages

```
<?xml version="1.0" encoding="UTF-8"?>
<Connection protocolVersion="1">
  <Request>
    <Hash>2707d2ff-e818-4ea9-be4b-fbce3486431e</Hash>
    <SenderAddress>192.168.0.0:2000</SenderAddress>
    <Role>Player</Role>
  </Request>
</Connection>

<?xml version="1.0" encoding="utf-8"?>
<Connection protocolVersion="1">
  <Confirmation>
    <Hash>d7a070d8-3da0-484c-8036-1dd2ffc2e762</Hash>
    <Id>1</Id>
  </Confirmation>
</Connection>
```

Listing 4: Example of game state messages

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages protocolVersion="1"
          senderHash="2707d2ff-e818-4ea9-be4b-fbce3486431e">
  <Message id="2707d2ff-e818-4ea9-be4b-fbce3486431e">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <GameState>
        <Setup>
          <Request>
            <Team>Blue</Team>
          </Request>
        </Setup>
      </GameState>
    </Content>
  </Message>
  <Message id="2707d2ff-e818-4ea9-be4b-fbce3486431e">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <GameState>
        <Setup>
          <Confirmation>
            <Role>TeamLeader</Role>
            <Team>Red</Team>
          </Confirmation>
        </Setup>
      </GameState>
    </Content>
  </Message>
  <Message id="2707d2ff-e818-4ea9-be4b-fbce3486431e">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <GameState>
        <Start>
          <Players>
            <Player id="3">
              <Team>Blue</Team>
            </Player>
            <Player id="4">
              <Team>Red</Team>
            </Player>
          </Players>
```

```
        <Board>
          <Width>12</Width>
          <Height>14</Height>
        </Board>
        <StartPosition>
          <X>7</X>
          <Y>12</Y>
        </StartPosition>
      </Start>
    </GameState>
  </Content>
 </Message>
 <Message id="2707d2ff-e818-4ea9-be4b-fbce3486431e">
  <Sender id="1"></Sender>
  <Receiver id="2"></Receiver>
  <Content>
    <GameState>
      <End>
        <Reason>Failure</Reason>
        <Result>
          <WinningTeam>Blue</WinningTeam>
          <Team name="Blue">
            <GoalsCompleted>3</GoalsCompleted>
          </Team>
          <Team name="Red">
            <GoalsCompleted>20</GoalsCompleted>
          </Team>
        </Result>
      </End>
    </GameState>
  </Content>
 </Message>
</Messages>
```

Listing 5: Example of action message

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages protocolVersion="1"
         senderHash="2707d2ff-e818-4ea9-be4b-fbce3486431e">
  <Message id="be62283e-5ae4-49cc-85ce-c967c03a53e5">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <Action>
        <Move>
          <Request>
            <Direction>Up</Direction>
          </Request>
        </Move>
      </Action>
    </Content>
  </Message>
  <Message id="2707d2ff-e818-4ea9-be4b-fbce3486431e" contextId="be62283e-5ae4-49cc-85ce-c967c03a53e5">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <Action>
        <Move>
          <Response>
            <Result>Success</Result>
            <Position>
              <X>2</X>
              <Y>5</Y>
            </Position>
            <Distance>21</Distance>
          </Response>
        </Move>
      </Action>
    </Content>
  </Message>
  <Message id="2707d2ff-e818-4ea9-be4b-fbce3486431e" contextId="be62283e-5ae4-49cc-85ce-c967c03a53e5">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <Action>
        <Move>
          <Response>
            <Result>Failure</Result>
          </Response>
        </Move>
      </Action>
    </Content>
  </Message>
</Messages>
```

Listing 6: Example of exchange knowledge messages

```
<?xml version="1.0" encoding="UTF-8"?>
<Messages protocolVersion="1"
         senderHash="2707d2ff-e818-4ea9-be4b-fbce3486431e">
  <Message id="be62283e-5ae4-49cc-85ce-c967c03a53e5">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <ExchangeKnowledge>
        <Request>
        </Request>
      </ExchangeKnowledge>
    </Content>
  </Message>
  <Message id="2707d2ff-e818-4ea9-be4b-fbce3486431e" contextId="be62283e-5ae4-49cc-85ce-c967c03a53e5">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <ExchangeKnowledge>
        <Response>
          <Result>Allow</Result>
        </Response>
      </ExchangeKnowledge>
    </Content>
  </Message>
  <Message id="2707d2ff-e818-4ea9-be4b-fbce3486431e" contextId="be62283e-5ae4-49cc-85ce-c967c03a53e5">
    <Sender id="1"></Sender>
    <Receiver id="2"></Receiver>
    <Content>
      <ExchangeKnowledge>
        <Data>
          <Tiles>
            <Tile timestamp="2002-05-30T09:00:00">
              <Position>
                <X>2</X>
                <Y>5</Y>
              </Position>
              <Distance>25</Distance>
              <Player id="1">
                <Team>Blue</Team>
              </Player>
              <TileObject>Piece</TileObject>
              <IsPursued>0</IsPursued>
            </Tile>
            <Tile timestamp="2002-05-30T10:00:00">
              <Position>
                <X>10</X>
                <Y>1</Y>
              </Position>
              <Distance>5</Distance>
              <TileObject>CompletedGoal</TileObject>
              <IsPursued>1</IsPursued>
            </Tile>
            <Tile timestamp="2002-05-30T10:00:00">
              <Position>
                <X>10</X>
                <Y>1</Y>
              </Position>
              <Distance>1</Distance>
              <TileObject>NonGoal</TileObject>
            </Tile>
          </Tiles>
        </Data>
      </ExchangeKnowledge>
    </Content>
  </Message>
</Messages>
```

2.6.4 Messages schemas

Listing 7: Used types schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
          elementFormDefault="qualified"
          targetNamespace="http://messageSchema.something.com"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns="http://messageSchema.something.com">

  <xs:simpleType name="Guid">
    <xs:restriction base="xs:string">
      <xs:length value="36" fixed="true"/>
      <xs:pattern value="[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-4[0-9a-fA-F]{3}-[8-9a-bA-B][0-9a-fA-F]{3}-[0-9a-fA-F
          ]{12}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ProtocolVersion">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="Role">
    <xs:restriction base="xs:string">
      <xs:enumeration value="GameMaster"/>
      <xs:enumeration value="Player"/>
      <xs:enumeration value="TeamLeader"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ActionBoolResult">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Success"/>
      <xs:enumeration value="Failure"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="Position">
    <xs:sequence>
      <xs:element name="X" type="xs:int"/>
      <xs:element name="Y" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="TaskAreaTileObject">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Nothing"/>
      <xs:enumeration value="Piece"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="TileObject">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Nothing"/>
      <xs:enumeration value="Piece"/>
      <xs:enumeration value="CompletedGoal"/>
      <xs:enumeration value="NonGoal"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="Tile">
    <xs:sequence>
      <xs:element name="Position" type="Position"/>
      <xs:element name="Distance" type="xs:int"/>
      <xs:element name="Player" type="Player" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Player">
    <xs:sequence>
      <xs:element name="Team" type="Team"/>
    </xs:sequence>
```

```
    <xs:attribute name="id" type="xs:int" use="required"/>
  </xs:complexType>
  <xs:simpleType name="Team">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Blue"/>
      <xs:enumeration value="Red"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Listing 8: Connection messages schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
           elementFormDefault="qualified"
           targetNamespace="http://messageSchema.something.com"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://messageSchema.something.com">
  <xs:include schemaLocation="./CommonTypes.xsd"/>

  <xs:simpleType name="IpAddress">
    <xs:restriction base="xs:string">
      <xs:pattern value="^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.)
           {3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]):[0-9]+$"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="ConnectionRequest">
    <xs:sequence>
      <xs:element name="Hash" type="Guid"/>
      <xs:element name="SenderAddress" type="IpAddress"/>
      <xs:element name="Role" type="Role"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ConnectionConfirmation">
    <xs:sequence>
      <xs:element name="Hash" type="xs:string"/>
      <xs:element name="Id" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ConnectionRefusal">
    <xs:sequence>
      <xs:element name="Hash" type="xs:string"/>
      <xs:element name="Description" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Connection">
    <xs:complexType>
      <xs:choice>
        <xs:element name="Request" type="ConnectionRequest"/>
        <xs:element name="Confirmation" type="ConnectionConfirmation"/>
        <xs:element name="Refusal" type="ConnectionRefusal"/>
      </xs:choice>
      <xs:attribute name="protocolVersion" type="ProtocolVersion" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 9: Messages schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
          elementFormDefault="qualified"
          targetNamespace="http://messageSchema.something.com"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns="http://messageSchema.something.com">
  <xs:include schemaLocation="./CommonTypes.xsd"/>

  <xs:element name="Messages">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Message" minOccurs="1" maxOccurs="100">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Sender">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:int" use="required"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="Receiver">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:int" use="required"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="Content" type="Content"/>
            </xs:sequence>
            <xs:attribute name="id" type="Guid" use="required"/>
            <xs:attribute name="contextId" type="Guid"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="protocolVersion" type="ProtocolVersion" use="required"/>
      <xs:attribute name="senderHash" type="Guid"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="Content">
    <xs:choice>
      <xs:element name="ExchangeKnowledge" type="ExchangeKnowledge"/>
      <xs:element name="Action" type="Action"/>
      <xs:element name="GameState" type="GameState"/>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="ExchangeKnowledge">
    <xs:choice>
      <xs:element name="Request" />
      <xs:element name="Response" type="ExchangeKnowledgeResponse"/>
      <xs:element name="Data" type="ExchangeKnowledgeData"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="ExchangeKnowledgeData">
    <xs:sequence>
      <xs:element name="Tiles">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Tile" minOccurs="1" maxOccurs="unbounded">
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="Tile">
                    <xs:sequence>
                      <xs:element name="TileObject" type="TileObject"/>
                      <xs:element name="IsPursued" type="xs:boolean" minOccurs="0"/>
                    </xs:sequence>
                    <xs:attribute name="timestamp" type="xs:dateTime"/>
                  </xs:extension>
```

```
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="ExchangeKnowledgeResponse">
 <xs:sequence>
  <xs:element name="Result">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="Allow"/>
     <xs:enumeration value="Decline"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:element>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="GameState">
 <xs:choice>
  <xs:element name="Setup" type="Setup"/>
  <xs:element name="Start" type="Start"/>
  <xs:element name="End" type="End"/>
 </xs:choice>
</xs:complexType>
<xs:complexType name="Setup">
 <xs:choice>
  <xs:element name="Request" type="SetupRequest"/>
  <xs:element name="Confirmation" type="SetupConfirmation"/>
 </xs:choice>
</xs:complexType>
<xs:complexType name="SetupRequest">
 <xs:sequence>
  <xs:element name="Team" type="Team"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="SetupConfirmation">
 <xs:sequence>
  <xs:element name="Role" type="Role"/>
  <xs:element name="Team" type="Team"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="Start">
 <xs:sequence>
  <xs:element name="Players">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="Player" type="Player" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="Board">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="Width" type="xs:int"/>
     <xs:element name="Height" type="xs:int"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="StartPosition" type="Position"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="End">
 <xs:sequence>
  <xs:element name="Reason">
```

```
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Success"/>
          <xs:enumeration value="Failure"/>
          <xs:enumeration value="Timeout"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="Result">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="WinningTeam" type="Team"/>
          <xs:sequence>
            <xs:element name="Team" minOccurs="2" maxOccurs="2">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="GoalsCompleted" type="xs:int"/>
                </xs:sequence>
                <xs:attribute name="name" type="Team" use="required"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Action">
  <xs:choice>
    <xs:element name="Move" type="MoveAction"/>
    <xs:element name="Discover" type="DiscoverAction"/>
    <xs:element name="Pickup" type="PickupAction"/>
    <xs:element name="Test" type="TestAction"/>
    <xs:element name="Place" type="PlaceAction"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="MoveAction">
  <xs:choice>
    <xs:element name="Request" type="MoveRequest"/>
    <xs:element name="Response" type="MoveResponse"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="MoveRequest">
  <xs:sequence>
    <xs:element name="Direction">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Left"/>
          <xs:enumeration value="Up"/>
          <xs:enumeration value="Right"/>
          <xs:enumeration value="Down"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="MoveResponse">
  <xs:sequence>
    <xs:element name="Result" type="ActionBoolResult"/>
    <xs:element name="Position" type="Position" minOccurs="0"/>
    <xs:element name="Distance" type="xs:int" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="DiscoverAction">
  <xs:choice>
```

```
    <xs:element name="Request"/>
    <xs:element name="Response" type="DiscoverResponse"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="DiscoverResponse">
  <xs:sequence>
    <xs:element name="Tiles">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Tile" minOccurs="1" maxOccurs="8">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension base="Tile">
                  <xs:sequence>
                    <xs:element name="TileObject" type="TaskAreaTileObject"/>
                  </xs:sequence>
                </xs:extension>
              </xs:complexContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="PickupAction">
  <xs:choice>
    <xs:element name="Request"/>
    <xs:element name="Response" type="PickupResponse"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="PickupResponse">
  <xs:sequence>
    <xs:element name="Result" type="ActionBoolResult"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="TestAction">
  <xs:choice>
    <xs:element name="Request"/>
    <xs:element name="Response" type="TestResponse"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="TestResponse">
  <xs:sequence>
    <xs:element name="Result">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Piece"/>
          <xs:enumeration value="Sham"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="PlaceAction">
  <xs:choice>
    <xs:element name="Request"/>
    <xs:element name="Response" type="PlaceResponse"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="PlaceResponse">
  <xs:sequence>
    <xs:element name="Result">
      <xs:simpleType>
        <xs:restriction base="xs:string">
```

```
          <xs:enumeration value="Completed"/>
          <xs:enumeration value="Meaningless"/>
          <xs:enumeration value="Disallowed"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

2.7 System special states

The game flow can be divided into three phases: game initialization phase, game phase and game finalization phase. During each of those phases components can enter one of several system's special states.

All of the components share the same set of special states which are described below:

- `Initialization state` - during this state a particular component allocates its resources and connects to other components.
- `Communication error state` - a component enters into a communication error state when communication exceptions occur. The following communication exceptions are distinguished:
  - incompatibility with the communication protocol
  - parsing error
- `Connection error state` - a component enters into a connection error state when connection exceptions occur. The following connection exceptions are distinguished:
  - Internet connection malfunctions
  - invalid recipient address - applies only to the Communication Server
- `Finalization state` - during this state all components deallocate their resources and disconnect from other components. Additionally the Game Master sends messages containing information about the result to all players.

In case of error states the system behaves according to the following UML state diagrams:

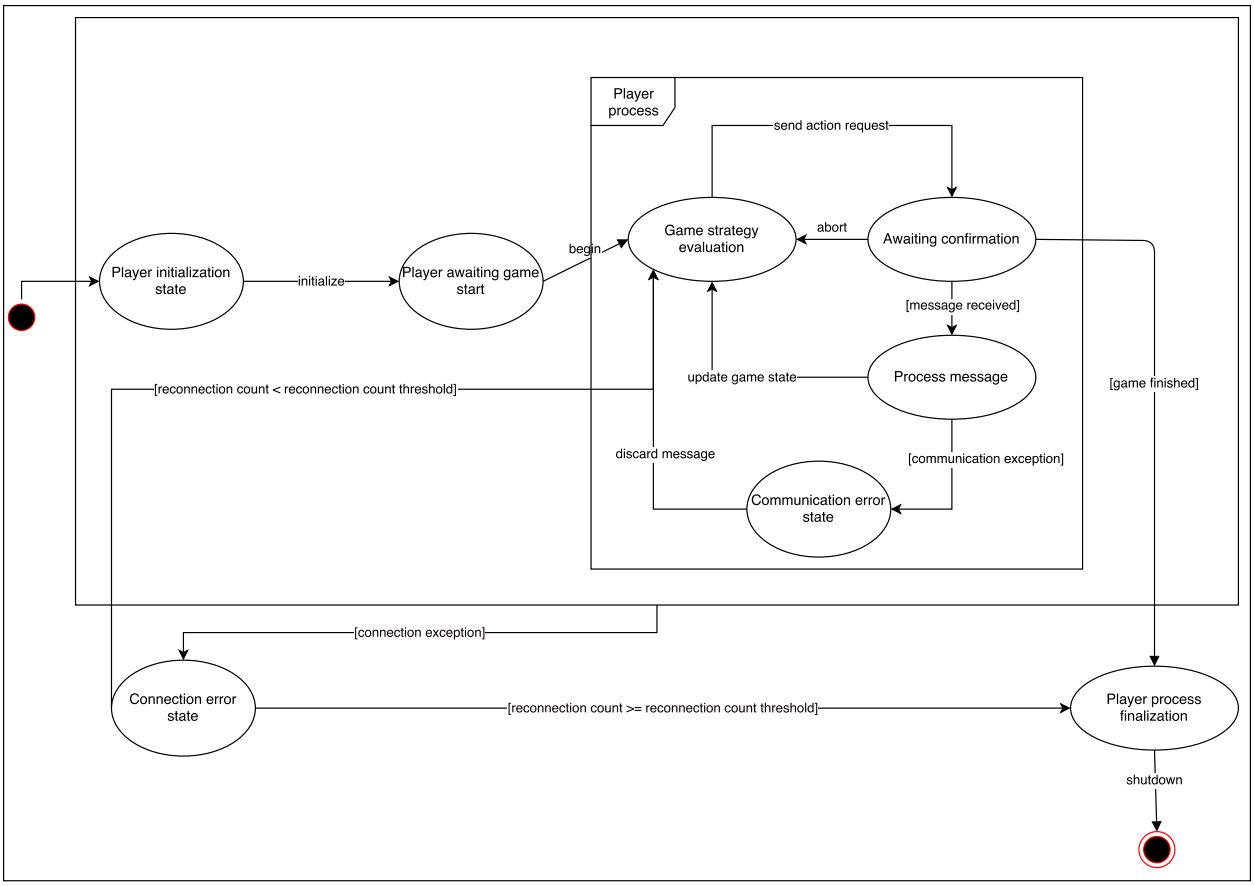

Diagram 17: Player state diagram

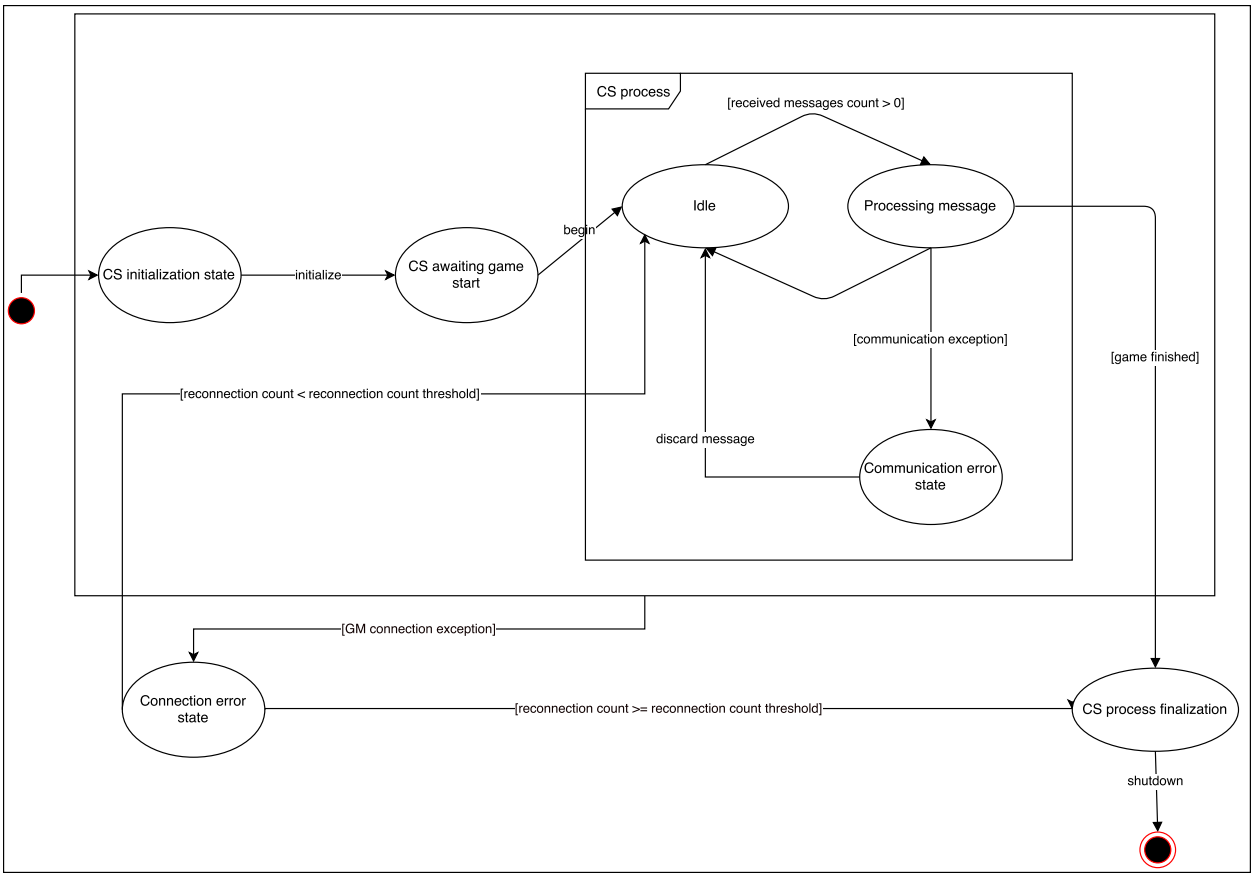

Diagram 18: Communication Server state diagram

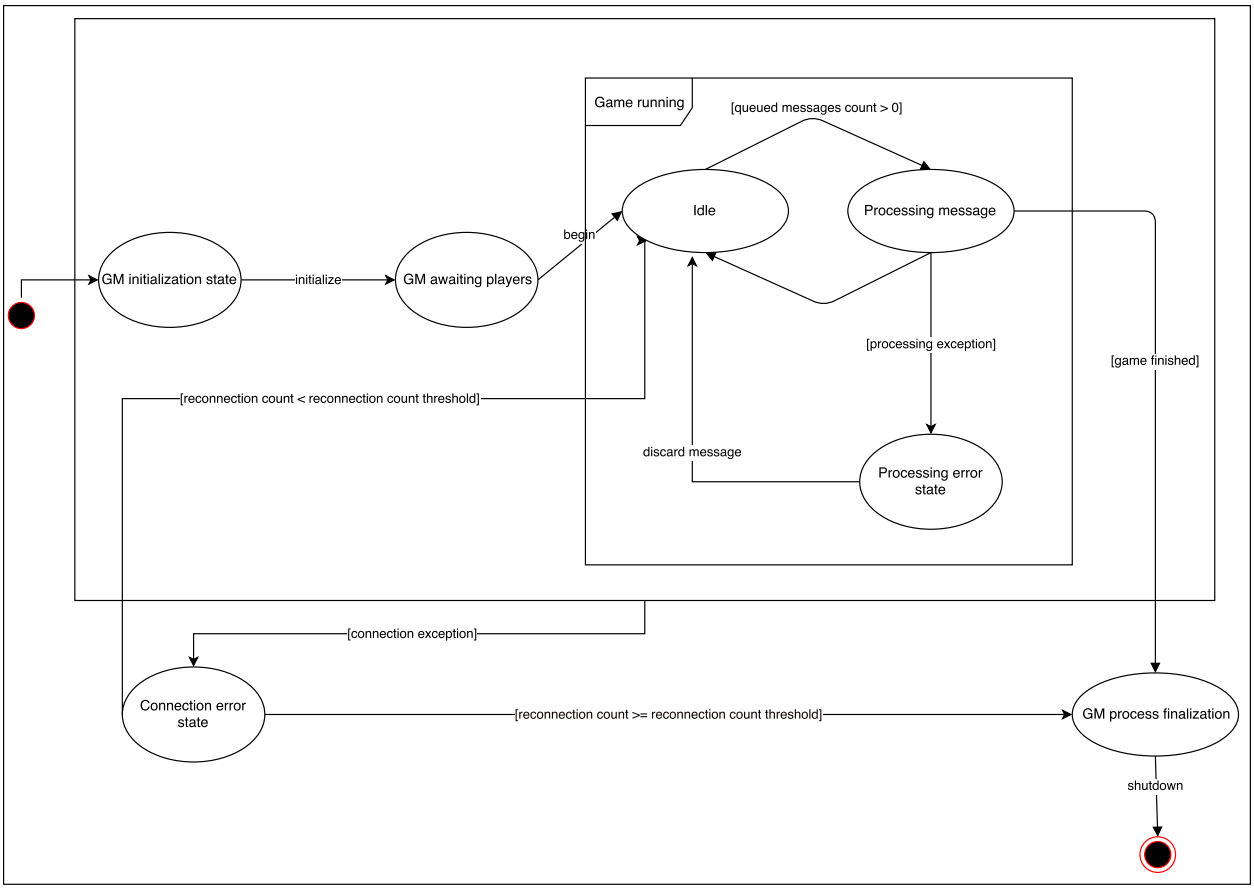

Diagram 19: Game Master state diagram

When a communication error state occurs the current message being processed is discarded. The system introduces a mechanism of timeouts in order to handle connection error states. The component tries to reconnect to the game a given amount of times. The limit of re-connections is set in the configuration file described in section 1.4. It is necessary to distinguish a case when a message cannot be delivered because of a player malfunction(including disconnection). In that case the message gets discarded.