Software Engineering 2

Project Game System

ver. 0.9 RC1

Michał Okulewicz

February 28, 2017

Contents

Glossary

The Project Game - a real-time board game played by a two competing teams of cooperating players

Player - an agent playing the game, holds its own view of the state of the game

Team - group of *Players* who cooperate in order to achieve the goal of the game

Piece - a token representing a project resource which is initially located in the *Tasks Area* of the board by the *Game Master*,

Goal Area - a part of the board where the *Players* place the *pieces*

Tasks Area - a part of the board from which the *Players* may collect the *pieces*

Game Goal - discovering the shape of the project goal by placing a set of pieces in the *Goal Area*

Project Game System - a distributed IT system for managing multiple *Project Games* and the agents participating in them

Game Master - an agent responsible for generating the board and the shape of the project goals, holds the whole state of the game and generates new pieces on the tasks area,

Communications Server - a module responsible for passing messages between *Game Master* and *Players*

Introduction

The purpose of this document is to present the rules and the goal of The Project Game and specify the requirements for multi-agent system for playing the game.

The purpose of The Project Game itself is to create an environment in which computer and human agents might interact and whose behaviour can be observed, quantified and simulated. The scope of this particular system is focused on the message passing system and computer agents only.

The system can be seen from the following perspectives:

- as a network system,
- as a distributed database system,
- as a multi-agent system.

Network system perspective

The Players are separated from each other and from Game Master by a network connection. In the network system perspective, the goal of the project is to implement transparent communication between Players and Game Master agents. The technical part of the communication should be properly separated from the logic through the message objects management and communication management layers.

Distributed database perspective

Game Master holds the whole and most accurate state of the game (master database) and acts as a module for timestamping the data. Players keep partial local copies of the state of the game (partial mirror databases). Access to master database is limited to 9 fields per request, while access to the data in mirror databases is limited by the accessibility of those nodes and the state of their data. In the distributed system perspective, the goal of the project is to implement a best strategy for obtaining the most accurate and crucial data with as little cost as possible.

Multi-agent perspective

Each Player forms beliefs from the knowledge obtained from the Game Master and other Players. Each Player assumes certain intentions of the Players in his Team and the opposite Team. In the multi-agent perspective, the goal of the project is to implement the best strategy for obtaining, testing, placing the pieces and exchanging information resulting in the fastest possible winning the game.

Document contents

Part I of the document describes the game organization, the objects appearing in the game, the possible moves of the players and their effects. Part II specifies the system technical requirements and defines the communication protocol.

Part I

The Project Game

Chapter 1

Game description

1.1 Game concept

The idea of The Project Game is to simulate competitive project development by two teams of players. The game simulates the following properties of projects development: (1) tasks are connected with risks (usually negative), (2) goals of the project are unclear (and need to be discovered), and (3) communication between members helps to speed up the process of the development.

1.2 Game rules

The rules of the game could be summarized in the following way:

1. The game is played by two teams of players: red and blue.
2. The game is played on a rectangular board[1] (depicted in Figure 1.1).
3. The board is divided into 3 parts: a common tasks area, a red team goals area and a blue team goals area.
4. The game is controlled by a game master, who places (in secret and randomly) pieces on the tasks area.
5. The player has ability to move around the board, discover the state of surrounding fields, handle the pieces and exchange information with other players.
6. The game is played in real time with a time cost assigned to each of the possible player's actions.
7. The objective of the team is to complete the project as fast as possible by discovering all the goal fields in the goals area.
8. The goal fields are known only to the game master, a player who discovered them and the players with whom that information has been shared.

[1] the size of the board may vary

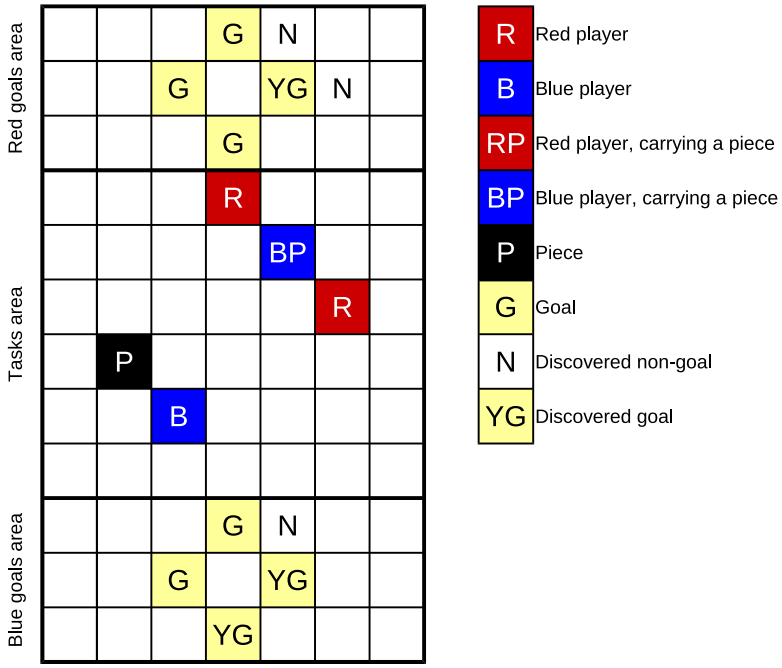

Figure 1.1: Global state of the game.

1.3 Game state

The state of The Project Game consists of the following data:

- Players' locations
- Pieces' locations
- Pieces' state (sham or non-sham)
- Project's goals locations
- Project's goals discovery state

The true state of the game is known only to the Game Master. Figure 1.1 presents the state of the game as seen by the Game Master.

1.4 Game actions

The Players discover the state of the game by interacting with the board, the pieces and each other. State of the game perceived by one of the blue Players before and after exchanging information with another blue Player is depicted in Figure 1.2.

Possible moves of the Player consist of:

- moving in one of 4 directions,
- discovering the contents of 8 neighbouring fields (and the currently occupied field),
- testing the picked piece for being a sham,
- placing a piece in the goals, in hope of completing one of the project objectives (or leaving a sham piece on the board),
- request exchange of information with another player.

The Player may discover the goals 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.

Player actions have following ramifications and constrains:

- Player cannot move into a field occupied by another player.
- Player cannot move into a goal field of the opposing team.
- The piece may be picked only by a Player which is located in the same field as that piece.
- Observing a field (either by discovering or entering it) results in receiving information about the Manhattan distance to the nearest piece.
- Team leader must exchange information with another Player from his team.
- Member of a team must exchange information with his team leader.

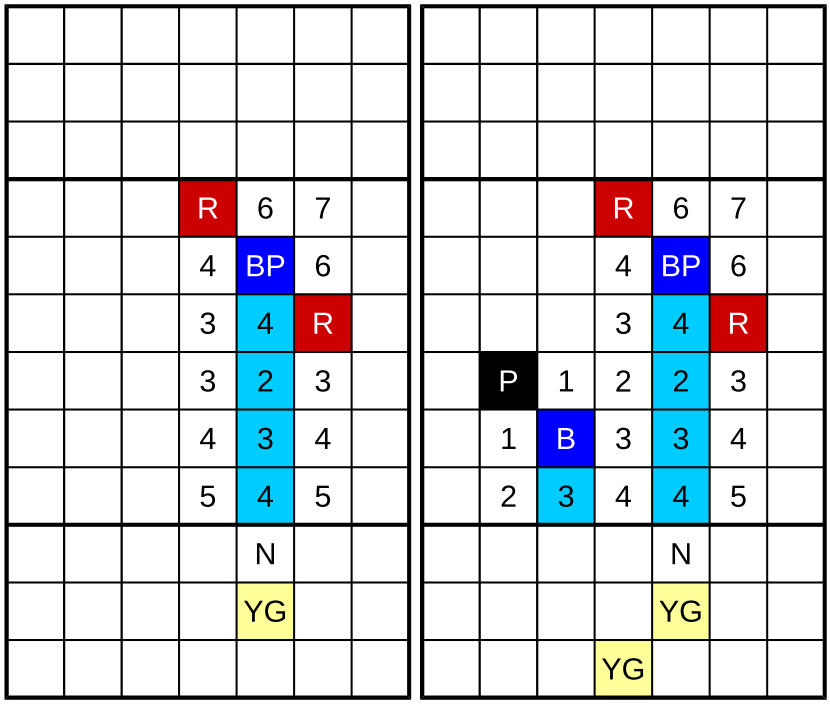

Figure 1.2: Personal game state perceived by one of the blue players. The left one is before he exchanged knowledge with the other blue player, while the right one after the exchange. Light blue color marks the traces of each of the clients. Visible piece has been added by Game Master during player movement.

Part II

The Project Game System

Chapter 2

Requirements

The system organizes matches between teams of cooperating agents. The game is played in real time on a rectangular board, consisting of the tasks area and the goals area. Each team of agents needs to complete an identical project, consisting of placing a set of pieces, picked from the tasks area, in the project goals area. The pieces appear at random within the tasks area. The agent's view of the tasks area is limited and the shape of the goals needs to be discovered. The game is won by the team of agents which is first to complete the project.

2.1 Features

All communication passes through Communication Server and uses TCP sockets. Game Master serves one game at a time. After completing one game Game Master and Players immediately register for another game (with the same settings). Game Master logs in response times and number of victories for each of the Players. Displays their ranking and average response times after each game on a text console.

Each Player displays the whole state of the board on a text console after completing the game.

All the components should be able to run a in a text-mode only. Graphical User Interface is unnecessary, but may be additionally developed as a means of presenting the game state for development or testing purpose.

2.2 Performance

Communication Server and Game Master are able to serve smoothly at least 64 Players. Theoretical number of Players is unlimited. Communication Server is able to serve at least 8 independent games. Theoretical number of games is unlimited.

The size and number of messages should be minimal.

2.3 Security

Player should be unable to get information available only to other players. Player should be unable to influence other Players decisions with false information.

2.4 Technical settings

All the technical settings are set through an XML configuration file.

The following parameters are set in all the components:

- Interval between keep alive bytes (or expected keep alive)

The following parameters are set in the Player agent:

- Interval between retries for joining the game with a given name.

2.5 Game settings

All the game settings are set through an XML configuration file.

In order to create various types of games the following elements of the game are configurable as the parameters of the Game Master agent:

- Probability of piece being a sham
- Frequency of placing new pieces on board
- Initial number of pieces on board
- Width of the board, length of the tasks area, length of a single goals area
- Number of players in each of the teams
- Goal definition
- Name of the game

In order to balance different actions of the players, response delay for each of the actions is configured as the parameters of the Game Master agent (with default values given in parenthesis):

- Move delay (100ms)
- Discovery delay (450ms)
- Test delay (500ms)[1]
- Pick-up delay (100ms)
- Placing delay (100ms)
- Knowledge exchange (1200ms)[2]

[1]The idea: should take as long as crossing half of the board.

[2]The idea: should take as long as discovering number of the board task fields divided by number of players in team.

2.6 Running the application

Apart from the configuration files the following parameters are passed while starting the application and a proper shell script / batch file accepting them, and running certain modules of the system, must be dispatched with the project.

- YY - the current year (17)
- LANG - language (PL or EN)
- XX - group identifier

Communication Server runtime parameters:
`YY-LANG-XX-cs --port [port number binded on all interfaces] --conf [configuration filename]`
Player and Game Master runtime parameters:
`YY-LANG-XX-[pl|gm] --address [server IPv4 address or IPv6 address or host name] --port [server port number]`

In order to join a particular type of a game in a particular role the following elements are configurable as the parameters of the Player agent:

- Name of the game to join
- Preferred colour
- Preferred role

The Player will play only the game named exactly as requested, but the colour and the role might be assigned by Game Master as needed. For example, if the game has already all the necessary blue players, and there are new players coming, who prefer to be blue,

Therefore, apart from the technical setting the Player has additional parameters:
`--game [name of the game] --team [red|blue] --role [leader|player]`

Chapter 3

Communication protocol

The communication is maintained through the TCP/IP protocol. All messages are passed through the Communication Server which acts the service for registering and deregistering the games and as a proxy for passing all other messages.

The specification presents the exchange of messages between the components for the following typical scenarios: registering and starting a game, normal gameplay, information exchange between the players.

All messages are text XML messages. The messages are separated by a bytecode 23 (ETB - End transmission blocks). The code is present after each message and is also used as a connection keep alive data. The TCP/IP connections are maintained during the whole time of system operations, until the process, of at least one of the components in that particular connection, is finished.

In order for the Communication Server to be able to host multiple games at the same time, all messages send from Players and Game Master intended for a given Player or a Game Master are marked with the `playerId` or `gameId`. In order to provide security to the players knowledge each action message intended for the Game Master has a unique player GUID known only to that Player and Game Master.

3.1 Starting a game

In order to start a new game Game Master must register the game on the Comunication Server by `RegisterGame` message. When Players discover the existence of game by sending `GetGames` message, they subsequently choose the game they wish to join by sending `JoinGame` message. After completing all the Players, Game Master broadcasts the `Game` message to all Players, containing information about the size of the board, their teams and their initial location. The exchange of those messages and their responses is presented in Fig. 3.1.

3.2 Standard gameplay

During a standard gameplay Players send `Discover`, `Move`, `PickUp`, `TestPiece` and **Place** messages to the GameMaster. The messages specify only the information necessary for the

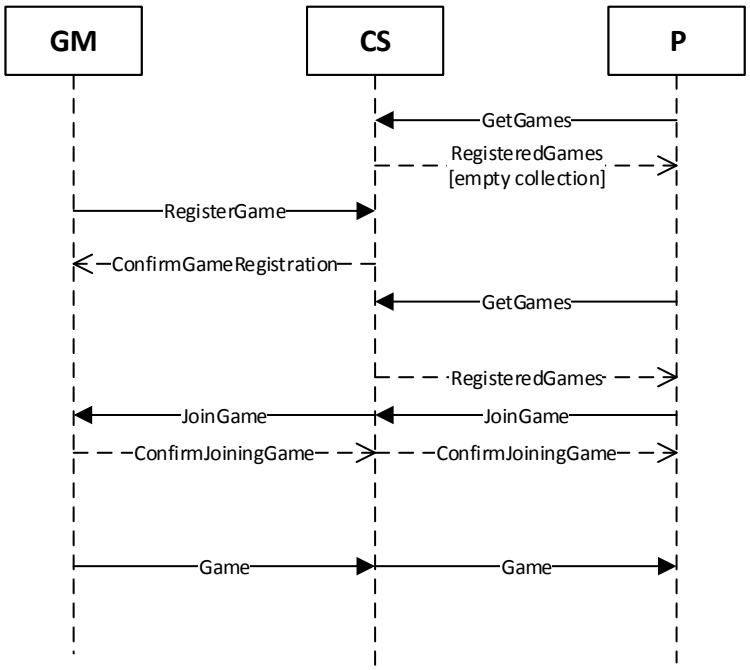

Figure 3.1: The messages passed between a sample (P)layer, (G)ame (M)aster and (C)ommunication (S)erver when a new game is organized in the system.

```
<?xml version="1.0" encoding="utf-8" ?>
<GetGames xmlns="http://theprojectgame.mini.pw.edu.pl/" />
```

Figure 3.2: An example of `GetGames` message

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterGame xmlns="http://theprojectgame.mini.pw.edu.pl/">
  <NewGameInfo
        name="easyGame"
        blueTeamPlayers="2"
        redTeamPlayers="2" />
</RegisterGame>
```

Figure 3.3: An example of `RegisterGame` message with a custom name and a two players teams setup.

```
<?xml version="1.0" encoding="utf-8" ?>
<ConfirmGameRegistration
       xmlns="http://theprojectgame.mini.pw.edu.pl/"
       gameId="1" />
```

Figure 3.4: An example of `ConfirmGameRegistration` message assigning id 1 to the game.

```
<?xml version="1.0" encoding="utf-8" ?>
<RegisteredGames xmlns="http://theprojectgame.mini.pw.edu.pl/">
  <!-- Numbers of players indicate how many slots are left for each team -->
  <GameInfo name="easyGame" blueTeamPlayers="2" redTeamPlayers="2"/>
  <GameInfo name="hardForBlueGame" blueTeamPlayers="5" redTeamPlayers="10"/>
</RegisteredGames>
```

Figure 3.5: An example of `RegisteredGames` message with two games listed.

```
<?xml version="1.0" encoding="utf-8" ?>
<JoinGame xmlns="http://theprojectgame.mini.pw.edu.pl/"
  gameName="easyGame"
  preferedRole="leader"
  preferedTeam="blue" />
```

Figure 3.6: A `JoinGame` message with player trying to join, as the leader of a blue team, the game denoted as *easyGame*.

```
<?xml version="1.0" encoding="utf-8" ?>
<ConfirmJoiningGame xmlns="http://theprojectgame.mini.pw.edu.pl/"
                    gameId="1"
                    playerId="2"
                    privateGuid="c094cab7-da7b-457f-89e5-a5c51756035f">
  <PlayerDefinition id="2" team="blue" type="player"/>
</ConfirmJoiningGame>
```

Figure 3.7: A `ConfirmJoiningGame` message setting the players unique Id and private GUID and informing about the Player's role in the game.

```
<?xml version="1.0" encoding="utf-8"?>
<Game xmlns="http://theprojectgame.mini.pw.edu.pl/"
     playerId="2">
  <Players>
    <Player team="red" type="leader" id="5" />
    <Player team="red" type="player" id="6" />
    <Player team="red" type="player" id="7" />
    <Player team="red" type="player" id="8" />
    <Player team="blue" type="leader" id="1" />
    <Player team="blue" type="player" id="2" />
    <Player team="blue" type="player" id="3" />
    <Player team="blue" type="player" id="4" />
  </Players>
  <Board width="5" tasksHeight="5" goalsHeight="3" />
  <PlayerLocation x="0" y="3" />
</Game>
```

Figure 3.8: A `GameMessage` for Player 2.

proxy server to be dispatched to a proper Game Master (`gameId` field) and the data allowing to safely identify the Player (`playerGuid` field) and decide about the Player's action (direction in the case of movement). All the other details (like the ID of the piece being picked up) are maintained by the Game Master only. To each request Game Master responses with a `Data` message containing the information obtained by the Player, concerning part of the game state. After the move finishing the game GameMaster broadcasts `Data` message containing the full state of the game, with the info that the game has finished to all the Players. The `Data` messages include the information allowing the server to dispatch the data to a proper player (`playerId` field) and the data about the game state aggregated in the `TaskFields`, `GoalFields`,`Pieces` and `PlayerLocation` elements.

Figure 3.9 presents the messages exchange between Players and Game Master. Please note, that the Communication Server **always acts as a proxy** in this communication, but is not depicted. Knowledge exchange between the Players, which is also a part of a gameplay, is explained in the next section.

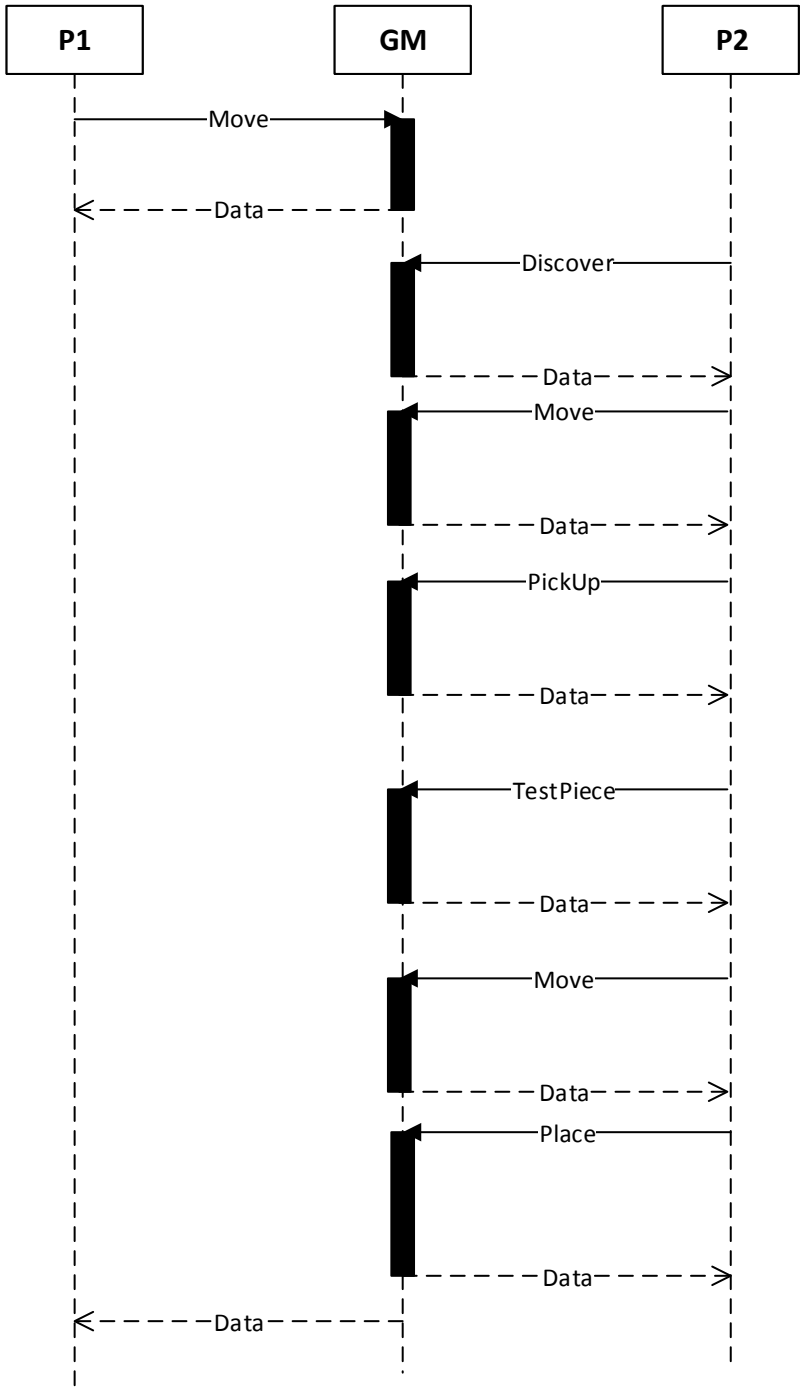

Figure 3.9: The messages passed between a sample Players P1,P2 and a (G)ame (M)aster during normal gameplay. The Communication Server is not depicted but acts as a proxy for communication.

```
<?xml version="1.0" encoding="utf-8"?>
<Discover xmlns="http://theprojectgame.mini.pw.edu.pl/"
    gameId="1"
    playerGuid="c094cab7-da7b-457f-89e5-a5c51756035f"
    />
```

Figure 3.10: A `Discover` message from Player.

```
<Data xmlns="http://theprojectgame.mini.pw.edu.pl/"
     playerId="1"
     gameFinished="false" >
  <TaskFields>
    <TaskField x="1" y="4" timestamp="2017-02-23T17:20:11"
      distanceToPiece="1" />
    <TaskField x="1" y="5" timestamp="2017-02-23T17:20:11"
      distanceToPiece="0" playerId="2" pieceId="2" />
    <TaskField x="1" y="6" timestamp="2017-02-23T17:20:11"
      distanceToPiece="1" />
    <TaskField x="0" y="4" timestamp="2017-02-23T17:20:11"
      distanceToPiece="2" />
    <TaskField x="0" y="5" timestamp="2017-02-23T17:20:11"
      distanceToPiece="1" />
    <TaskField x="0" y="6" timestamp="2017-02-23T17:20:11"
      distanceToPiece="2" />
    <TaskField x="2" y="4" timestamp="2017-02-23T17:20:11"
      distanceToPiece="2" />
    <TaskField x="2" y="5" timestamp="2017-02-23T17:20:11"
      distanceToPiece="1" />
    <TaskField x="2" y="6" timestamp="2017-02-23T17:20:11"
      distanceToPiece="2" />
  </TaskFields>
</Data>
```

Figure 3.11: A `Data` message response for the discover action.

```
<?xml version="1.0" encoding="utf-8"?>
<Move xmlns="http://theprojectgame.mini.pw.edu.pl/"
     gameId="1"
     playerGuid="c094cab7-da7b-457f-89e5-a5c51756035f"
     direction="up"/>
```

Figure 3.12: A `Move` message from Player.

```
<?xml version="1.0" encoding="utf-8"?>
<Data xmlns="http://theprojectgame.mini.pw.edu.pl/"
      playerId="1"
      gameFinished="false">
  <TaskFields>
    <TaskField x="1" y="5" timestamp="2017-02-23T17:20:11"
               distanceToPiece="5" />
  </TaskFields>
  <PlayerLocation x="1" y="5" />
</Data>
```

Figure 3.13: A `Data` message response for the proper move action.

```
<?xml version="1.0" encoding="utf-8"?>
<Data xmlns="http://theprojectgame.mini.pw.edu.pl/"
      playerId="1"
      gameFinished="false">
  <TaskFields>
    <TaskField x="1" y="5" timestamp="2017-02-23T17:20:11"
               distanceToPiece="0" playerId="2" pieceId="2" />
  </TaskFields>
  <Pieces>
    <Piece id="2" timestamp="2017-02-23T17:20:11" playerId="2"
           type="unknown" />
  </Pieces>
  <PlayerLocation x="1" y="4" />
</Data>
```

Figure 3.14: A `Data` message response for the move action, when trying to enter an occupied field.

```
<?xml version="1.0" encoding="utf-8"?>
<Data xmlns="http://theprojectgame.mini.pw.edu.pl/"
      playerId="1"
      gameFinished="false">
  <TaskFields>
  </TaskFields>
  <PlayerLocation x="1" y="7" />
</Data>
```

Figure 3.15: A `Data` message response for the move action, while trying to step out of the board.

```
<?xml version="1.0" encoding="utf-8"?>
<PickUpPiece
    xmlns="http://theprojectgame.mini.pw.edu.pl/"
    gameId="1"
    playerGuid="c094cab7-da7b-457f-89e5-a5c51756035f" />
```

Figure 3.16: A `PickUp` message from a Player.

```
<?xml version="1.0" encoding="utf-8"?>
<Data xmlns="http://theprojectgame.mini.pw.edu.pl/"
      playerId="1"
      gameFinished="false">
  <Pieces>
    <Piece id="2" timestamp="2017-02-27T12:00:34" playerId="1"
           type="unknown" />
  </Pieces>
</Data>
```

Figure 3.17: A `Data` message response for the piece pick up action.

```
<?xml version="1.0" encoding="utf-8"?>
<TestPiece xmlns="http://theprojectgame.mini.pw.edu.pl/"
      gameId="1"
      playerGuid="c094cab7-da7b-457f-89e5-a5c51756035f" />
```

Figure 3.18: A `TestPiece` message from a Player.

```
<?xml version="1.0" encoding="utf-8"?>
<Data xmlns="http://theprojectgame.mini.pw.edu.pl/"
      playerId="1"
      gameFinished="false">
  <Pieces>
    <Piece id="2" type="sham" playerId="1" timestamp="2017-02-28T13:45:56" />
  </Pieces>
</Data>
```

Figure 3.19: A `Data` message response for the placing of a piece action.

3.2.1 Knowledge exchange

Knowledge exchange is a special type of action during the gameplay as it involves not only a single Player and a Game Master, but also another Player. Therefore, it needs special handling as the cost of all actions need to be controlled and imposed by the Game Master.

The knowledge exchange is depicted in Fig. 3.20, with omission of the Communication Server as a proxy, and performed in a following way:

1. `AuthorizeKnowledgeExchange` message is send from Player 1 to the Game Mastered and relayed to the intended Player 2 as `KnowledgeExchangeRequest` message
2. Player 1 might reject the offer (without further delay) by sending a `RejectKnowledgeExchange` to Player 2
3. Player 2 might accept the offer by sending a `Data` message containing his whole state to Player 1 and an `AuthorizeKnowledgeExchange` message to Game Master
4. Game Master relays the message as an `AcceptExchangeRequest` message, which cannot be refused, and results in a response `Data` message from Player 1 to Player 2.

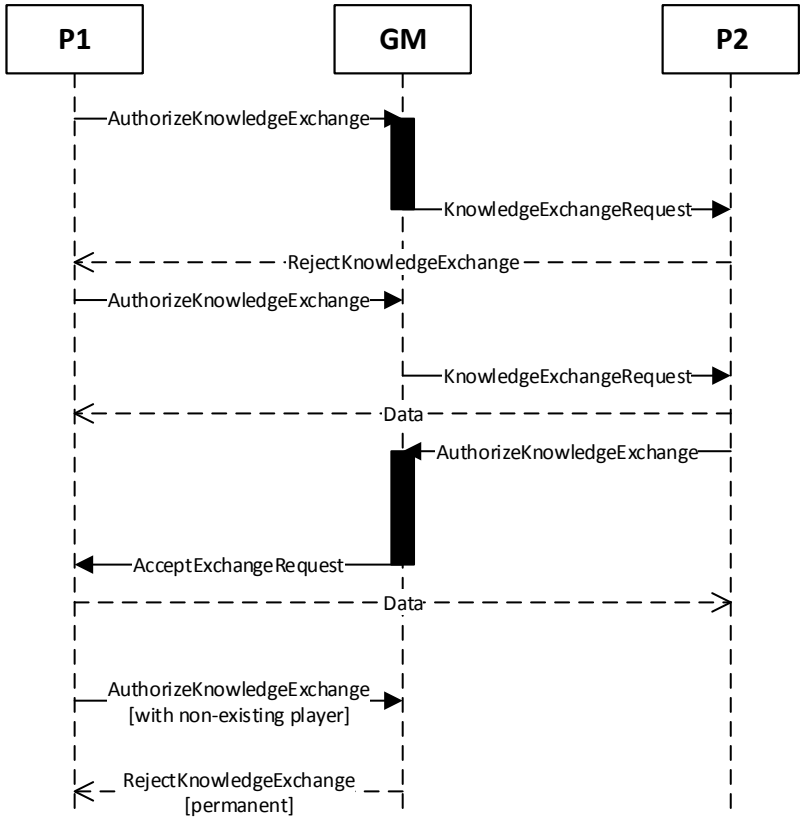

Figure 3.20: The messages passed between a sample Players P1,P2 and a (G)ame (M)aster with possible scenarios for data exchange. The Communication Server is not depicted but acts as a proxy for communication.

```
<?xml version="1.0" encoding="utf-8"?>
<AuthorizeKnowledgeExchange xmlns="http://theprojectgame.mini.pw.edu.pl/"
  withPlayerId="2"
  gameId="1"
  playerGuid="c094cab7-da7b-457f-89e5-a5c51756035f"
  />
```

Figure 3.21: An `AuthorizeKnowledgeExchange` message.

```
<?xml version="1.0" encoding="utf-8"?>
<KnowledgeExchangeRequest xmlns="http://theprojectgame.mini.pw.edu.pl/"
  playerId="2"
  senderPlayerId="1" />
```

Figure 3.22: A `KnowledgeExchangeRequest` message.

```
<?xml version="1.0" encoding="utf-8"?>
<RejectKnowledgeExchange xmlns="http://theprojectgame.mini.pw.edu.pl/"
  permanent="false"
  playerId="1"
  senderPlayerId="2" />
```

Figure 3.23: A `RejectKnowledgeExchange` message.

```
<?xml version="1.0" encoding="utf-8"?>
<AcceptExchangeRequest xmlns="http://theprojectgame.mini.pw.edu.pl/"
  playerId="2"
  senderPlayerId="2"
  />
```

Figure 3.24: An `AcceptExchangeRequest` message.

```
<?xml version="1.0" encoding="utf-8"?>
<Data xmlns="http://theprojectgame.mini.pw.edu.pl/"
      playerId="1"
      gameFinished="false">
  <TaskFields>
    <TaskField x="1" y="5" timestamp="2017-02-23T17:20:11"
               distanceToPiece="5" />
    <TaskField x="1" y="4" timestamp="2017-02-23T17:20:13"
               distanceToPiece="4" />
  </TaskFields>
  <GoalFields>
    <GoalField x="0" y="9" timestamp="2017-02-23T17:20:17"
               team="blue" type="non-goal"/>
    <GoalField x="1" y="9" timestamp="2017-02-23T17:20:19"
               team="blue" type="goal" playerId="2"/>
  </GoalFields>
  <Pieces>
    <Piece id="1" timestamp="2017-02-23T17:20:09" type="sham" />
    <Piece id="2" timestamp="2017-02-23T17:19:09" type="unknown" />
  </Pieces>
</Data>
```

Figure 3.25: A `Data` message with a knowledge exchange/accept exchange response data.