Software Engineering 2

Project Game System

version 2.0.0

Michał Okulewicz
Modified by Bartłomiej Szymański

March 8, 2019

Contents

Glossary

The Project Game - a real-time board game played by 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 Field - a type of field modelling a single project objective

Goal Area - a part of the board where the *Players* of a *Team* have exclusive access and *Goal Fields* are located (not all fields in the *Goal Area* are *Goal Fields*)

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

Game Goal - discovering the all the *Goal Fields* by placing pieces in the fields of 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 location of the *Goal Fields*, holds the whole state of the game and generates new pieces in the *Tasks Area*,

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

Change set

1.1.0 2018-04-09 Destroy piece action added

1.2.0 2018-04-09 Knowledge exchange communication protocol changed

1.2.1 2018-04-09 A more explicit statement about message queue on Game Master

1.2.2 2018-04-15 System extended with an error message

1.2.3 2018-04-15 Fixed leader—member issues

1.3.0 2018-04-16 Added ability to suggest actions

2.0.0 2019-03-08 Updated the documentation to work with the 2019 Software Engineering 2 project

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.

[1] the size of the board may vary

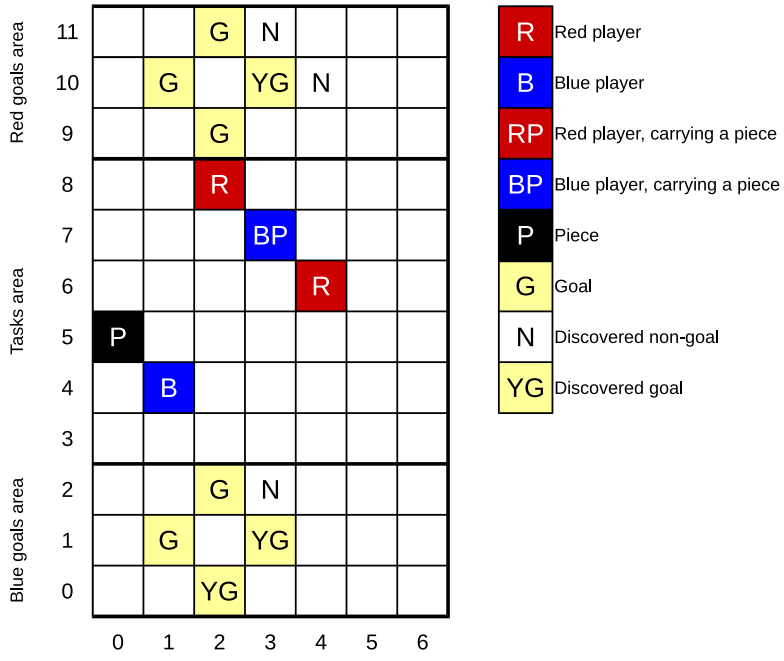

Figure 1.1: Global state of the game.

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.

9. Player may carry only a single piece and a field in the tasks area may contain only a single piece[2].

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.

[2]Player cannot place a piece on a field containing a piece, but Game Master can do it and destroy the piece which previously has been in the field

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 (up, down, right and left),
- discovering the contents of 8 neighbouring fields (and the currently occupied field),
- picking up a piece from a board,
- testing the picked piece for being a sham,
- destroying a piece held be a player (usually because it is a sham),
- placing a piece in the field in the goal area, in hope of completing one of the project objectives (or leaving a sham piece on the board),
- request exchange of information with another player,
- suggest actions to other players.

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 project's goals has been completed (field has been a goal field).
- 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 (field has not been a goal field).
- Placing a piece which is a sham results in getting no information (status of the field remains unknown to the player).

Player actions have following ramifications and constrains:

- Player cannot move into a field occupied by another player.
- Player cannot move into the goals area of the opposing team.
- The piece may be picked only by a Player which is located in the same field as that piece.
- Player can handle only one piece (he cannot pick up another piece).
- Player can place piece only on an empty field (with no other piece on it).
- 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.

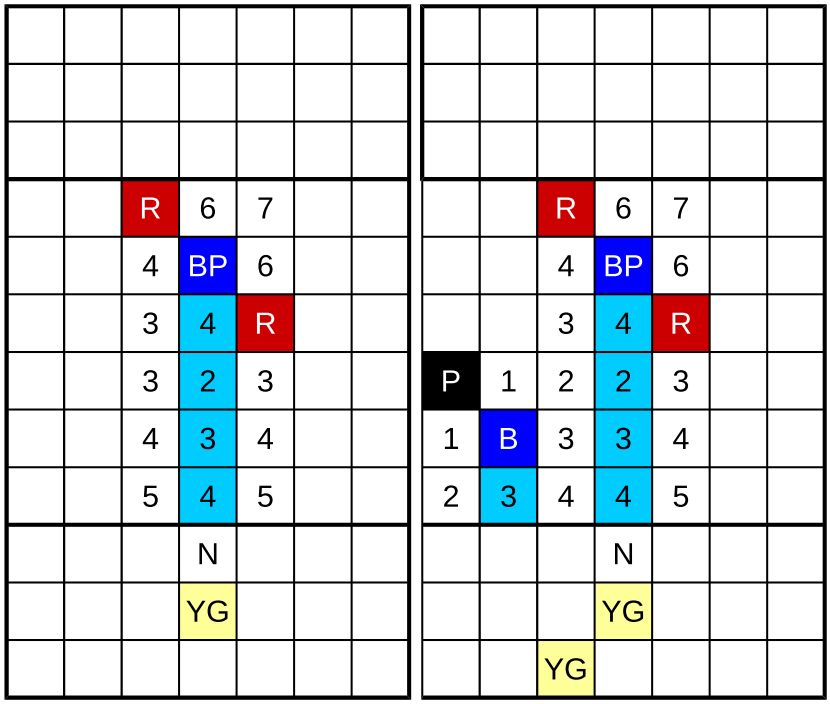

Figure 1.2: Personal game state perceived by one of the blue players. The left one is before he exchanged knowledge with the other blue player, while the right one after the exchange. Light blue color marks the traces of each of the clients. Visible piece has been added by Game Master during player movement.

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 requests metadata and a victory or defeat event 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 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 Supportability

Game Master should log the following data after receiving any type of request from a player:

- Request type (name of a root element in the JSON message)
- Timestamp (date and time of receiving the request)
- Game id
- Player id

- Player GUID
- Player colour
- Player role

Game Master should also place an event winning/losing game event in the same log (marked as Victory or Defeat for each of the players). The log should be stored in an UTF-8 encoded CSV file located in the same directory as the binary files of the Game Master.
A sample part of the event log should look as follows:

Type	Timestamp	Game ID	Player ID	Player GUID	Colour	Role
TestPiece	2019-03-21T23:12:01.456	1	2	c094...	blue	member
Move	2019-03-21T23:12:02.140	1	2	c094...	blue	member
Move	2019-03-21T23:12:03.241	1	2	c094...	blue	member
Move	2019-03-21T23:12:02.211	1	1	c179...	red	leader
PlacePiece	2019-03-21T23:12:03.456	1	2	c094...	blue	member
Victory	2019-03-21T23:12:03.456	1	2	c094...	blue	member
Defeat	2019-03-21T23:12:03.456	1	1	c179...	red	leader

2.3 Performance

Communication Server and Game Master are able to serve smoothly at least 16 Players. Theoretical number of Players is unlimited. Communication Server is allowed to host only one independent game.
The size and number of messages should be minimal.

2.4 Security

Player should be unable to get information available only to other players. Player should not try to influence other Players' decisions with false information.

2.5 Technical settings

All the technical settings are set through a JSON 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.

2.6 Game settings

All the game settings are set through an JSON 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]
- Destroy delay (100ms)
- Pick-up delay (100ms)
- Placing delay (100ms)
- Knowledge exchange (1200ms)[2]

Delays should be introduced in such a way, that the Player's actions affect the state of the game only after all preceding delay times have already passed. The players action might be activated at any moment during its delay, but a given player must not receive results before the delay time has passed.

2.7 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)
- XX - group identifier

[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.

Communication Server runtime parameters:
`YY-XX-cs --port [port number binded on all interfaces] --conf [configuration filename]`

Player and Game Master runtime parameters:
`YY-XX-[pl|gm] --address [server IPv4 address or IPv6 address or host name] --port [server port number] --conf [configuration filename]`

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, they would still be given a vacant role in a red team, as long as the specified name of the game has a matches a name registered by a Game Master.

The choice of the the role is remarked as secondary. Therefore, a player requesting to be a blue leader, when there are vacant roles of blue member and red leader, will become a blue member. Therefore, apart from the technical setting the Player has additional parameters:
`--game [name of the game] --team [red|blue] --role [leader|member]`

Chapter 3

Communication protocol

The communication is maintained through the TCP/IP protocol. All messages are passed through the Communication Server which acts as 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 JSON 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. Keep alive byte is sent from Player and Game Master to the Communication Server and responded with a keep alive byte. Sending a normal message also acts as keep alive.

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 player's knowledge each action message intended for the Game Master has a unique player GUID known only to that Player and Game Master.

If an error occurs on one of the components, a general purpose `ErrorMessage` might be used as a wrapper over the Exceptions mechanism. The message is intended mostly for debugging, so the component sending and `ErrorMessage` should not expect any particular behavior. Component receiving such a message should treat it as if an exception has occured.

3.1 Starting a game

In order to start a new game Game Master must register the game on the Communication Server by `RegisterGame` message.

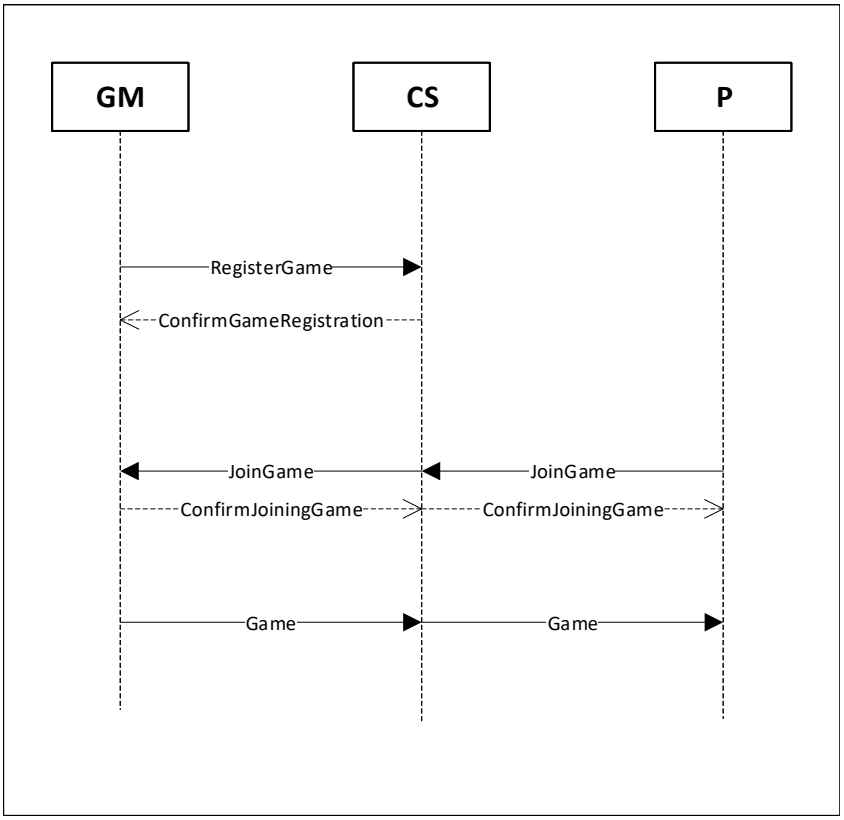

Figure 3.1: The messages passed between a sample (P)layer, (G)ame (M)aster and (C)ommunication (S)erver when a new game is organized in the system.

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.

```
{
    "action": "start"
}
```

Figure 3.2: An example of `RegisterGame` message.

```
{
    "action": "start",
    "result": "OK"
}
```

Figure 3.3: An example of `ConfirmGameRegistration` message starting the game.

```
{
    "action": "connect",
    "preferredteam": "blue",
    "type": "player"
}
```

Figure 3.4: A `JoinGame` message with player trying to join, as a player of a blue team.

```
{
    "action": "connect",
    "response": "OK",
    "type": "player"
}
```

Figure 3.5: A `ConfirmJoiningGame` message.

```
{
    "action": "connect",
    "response": "denied",
    "type": "player"
}
```

Figure 3.6: A `RejectJoiningGame` message informing that Game Master rejects player.

```
{
    "action": "start",
    "team": "blue",
    "role": "leader",
    "id": 2,
    "blueTeamSize": 4,
    "redTeamSize": 4,
    "location": {
        "X": 2,
        "Y": 5
    },
    "board": {
        "width": 5,
        "tasksHeight": 5,
        "goalsHeight": 3
    }
}
```

Figure 3.7: A `GameMessage` sent to Player 2 of the Blue Team.

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 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.
Please note, that the Manhattan distance to the nearest piece is calculated to a nearest piece currently placed on a field in the Tasks Area, regardless of its status (sham or not-sham). If no such piece is currently in the game, the distance value is equal to -1. The distance is calculated only for the fields in the Tasks Area, but can be observed from a Goals Area if the discover action is applied while standing on a field in the Goals Area neighbouring fields in the Tasks Area.

Figure 3.8 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.

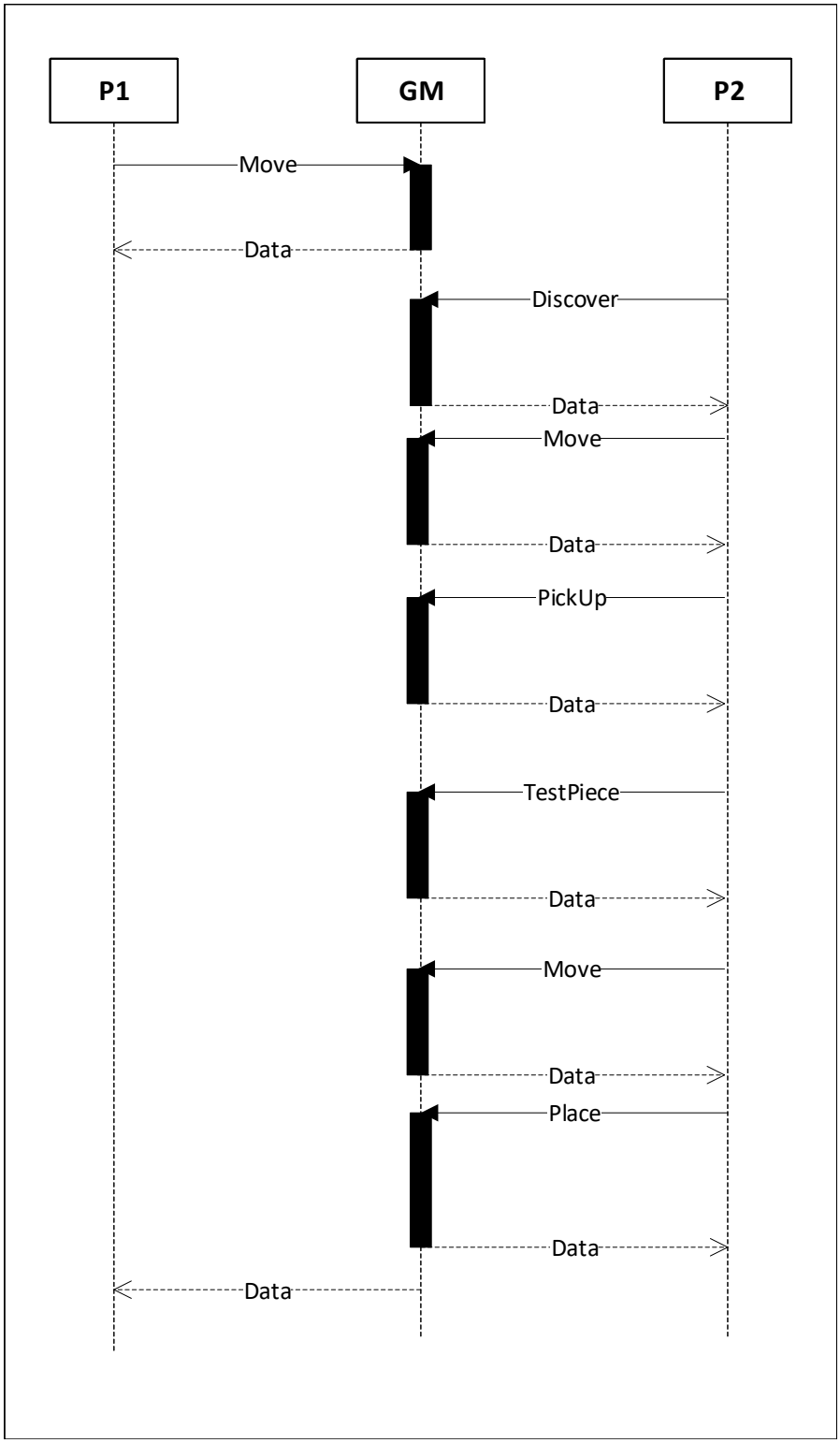

Figure 3.8: The messages passed between sample Players P1,P2 and a (G)ame (M)aster during normal gameplay. The Communication Server is not depicted but acts as a proxy for communication.

```
{
    "action": "state",
    "scope": {
        "X": 2,
        "Y": 5
    }
}
```

Figure 3.9: A `Discover` message from Player.

```
{
    "action": "state",
    "result": "OK",
    "scope": {
        "X": 2,
        "Y": 5
    },
    "fields": [
        {
            "X": 2,
            "Y": 5,
            "type": "sham",
            "id": -1
        },
        {
            "X": 2,
            "Y": 6,
            "type": "player",
            "id": 2
        },
        {
            "X": 3,
            "Y": 5,
            "type": "empty",
            "id": -1
        }
    ]
}
```

Figure 3.10: A `Data` message response for the discover action.

```
{
    "action": "move",
    "direction": "N"
}
```

Figure 3.11: A `Move` message from Player.

```
{
    "action": "move",
    "result": "OK",
    "underneath": "2"
}
```

Figure 3.12: A `Data` message response for the proper move action.

```
{
    "action": "move",
    "result": "denied"
}
```

Figure 3.13: A `Data` message response for the move action, when trying to enter an occupied field or field away from the board.

```
{
    "action": "pickup"
}
```

Figure 3.14: A `PickUp` message from a Player.

```
{
    "action": "pickup",
    "result": "OK"
}
```

Figure 3.15: A `Data` message response for the piece pick up action.

```
{
    "action": "test"
}
```

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

```
{
    "action": "test",
    "result": "OK",
    "test": true
}
```

Figure 3.17: A `Data` message response for the testing of a piece action.

```
{
    "action": "place"
}
```

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

```
{
    "action": "place",
    "result": "OK",
    "consequence": "meaningless"
}
```

Figure 3.19: A `Data` message response for the placing of a piece in a goals area 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 sent from Player 1 to the Game Mastered and relayed to the intended Player 2 as `KnowledgeExchangeRequest` message
2. Player 1 sends also `Data` message **immediately** after `AuthorizeKnowledgeExchange` message
3. Player 2 might reject the offer (without further delay) by sending a `RejectKnowledgeExchange` to Player 1 (through Game Master, without further delay)
4. Player 2 might accept the offer by sending a `Data` message containing his whole state to Player 1 (through Game Master)
5. Game Master relays the `Data` message from Player 2 to Player 1, and after a delay `Data` message from Player 1 to Player 2.

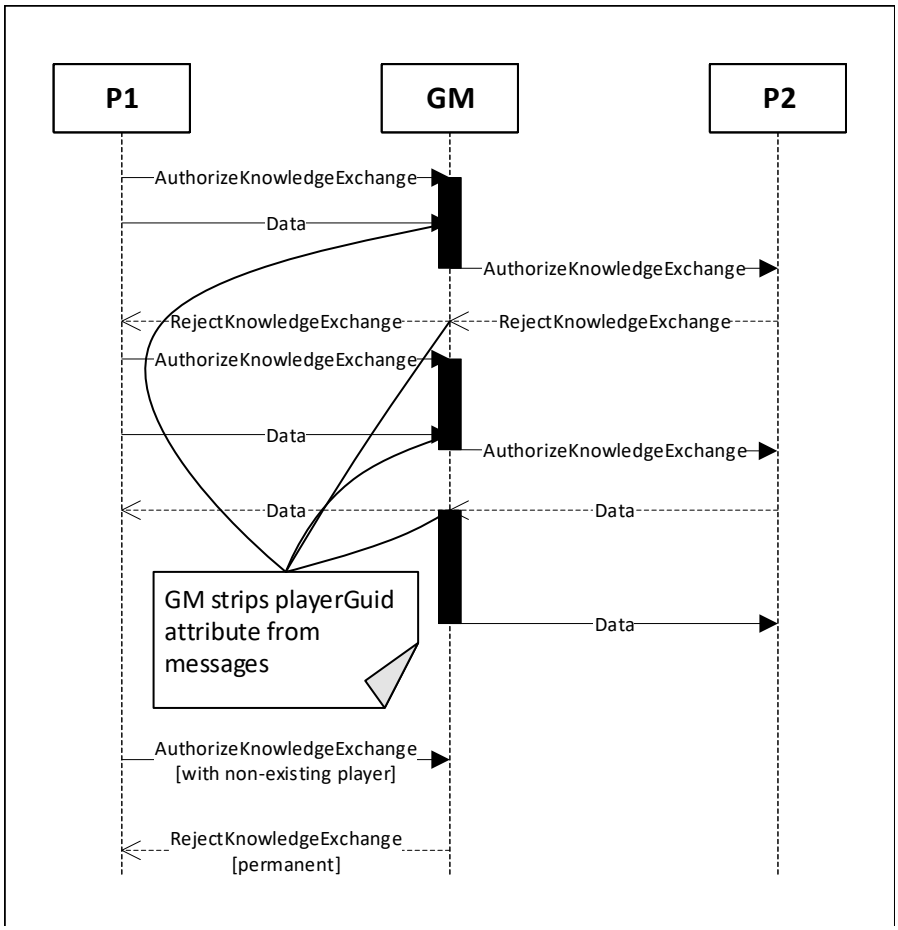

Figure 3.20: The messages passed between sample Players P1,P2 and a (G)ame (M)aster with possible scenarios for data exchange. The Communication Server is not depicted but acts as a proxy for communication.

```
1 {
2     "action": "exchange",
3     "playerID": 3
4 }
```

Figure 3.21: An `AuthorizeKnowledgeExchange` message.

```
{
    "action": "exchange",
    "result": "denied"
}
```

Figure 3.22: A `RejectKnowledgeExchange` message.

```
{
    "action": "exchange",
    "result": "OK"
}
```

Figure 3.23: An `AcceptExchangeRequest` message.

```
{
    "action": "send",
    "to": 2,
    "fields": [
        {
            "X": 2,
            "Y": 5,
            "value": "2"
        },
        {
            "X": 2,
            "Y": 6,
            "value": "T"
        },
        {
            "X": 3,
            "Y": 5,
            "value": "5"
        }
    ]
}
```

Figure 3.24: A `Data` message with a knowledge exchange/accept exchange response data.

3.2.2 Suggest actions

In order to provide an ability to coordinate players' actions a `SuggestAction` and `SugestActionResponse` messages have been added to the communication protocol. The idea is that any player might suggest scouting certain areas of the board or trying to fulfill a given goal.

The player, to which the suggestion has been send, responds with information what it is planning to do. Those might be the same or different Task and Goal fields.