# Communications Software, Protocols and Architectures

## Project work

Specifications Document

Group ID 01
Members: Juan Antonio Aldea Armenteros 0404450
Topic A: Reversi

## Table of Contents

## 1 Game overview

Reversi is a two players turn-based board game, played on an 8x8 squares board. There are 64 pieces and all are identical, each has a black side and a white side. Every turn one player places one piece on the board with his color facing up.
Each piece must be place so that an opponent's piece, or a row of opponent's pieces, is flanked by your pieces. All of the opponent's pieces between the player’s pieces are then turned over to become the player’s color.

<u>Start of the game:</u> The game is started in the position shown below on a Reversi board consisting of 64 squares in an 8×8 grid.

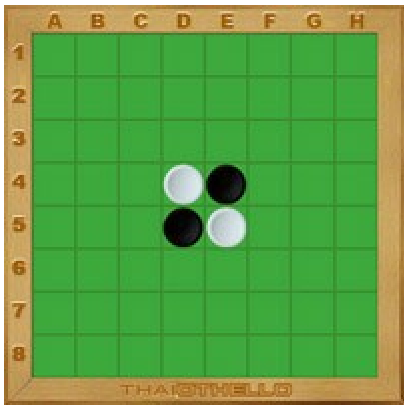


*Illustration 1: Starting Board*

<u>Aim of the game:</u> The objective of the game is to own more pieces than your opponent when the game end. The game is over when neither player has a move. This occurs when the grid has filled up or when neither player can legally place a piece in any of the remaining squares. Players take alternate turns. If a player cannot make a valid move, turn is given to the other player. When neither player can move  game ends. [WOF]

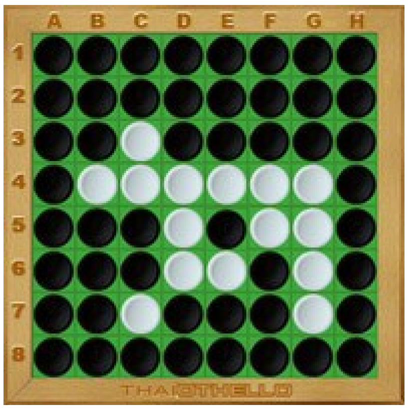


*Illustration 2: Game End*

## 2 System overview

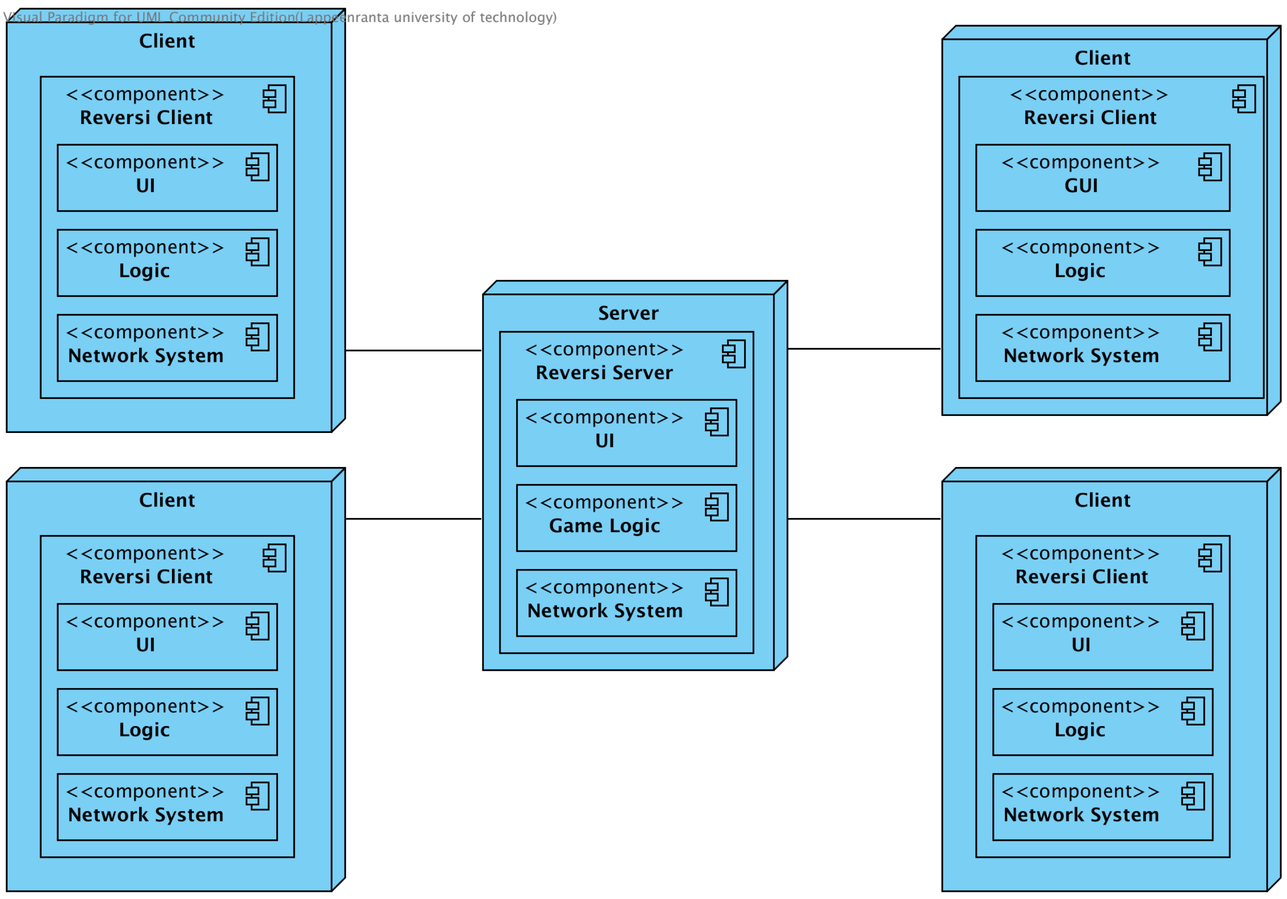
The game system follows the client-server model. There are two basics entities with this model, one of them acting as the client and the other acting as the server.

The server receives the actions from the users, handles the results of these actions on the current game and then sends the game updates to the clients. The server is able to handle multiple games sessions, having multiple pairs of players connected and playing simultaneously.

In the client side, the GUI displays the current state of the game and also receives the user input and notifies about them to the Logic, then the Logic sends those actions to the server and receives the game updates from the server.

All communications are handled by the TCP/IP [TCP] protocol.

If a server is running and listening for incoming client connections and two clients connect, a new game session is initiated as a child process and this child will be in charge of handling the game, and keeping the rules of the game followed.



Visual Paradigm for UML Community Edition(Lappeenranta university of technology)
Client
<<component>> Reversi Client
<<component>> UI
<<component>> Logic
<<component>> Network System
Client
<<component>> Reversi Client
<<component>> GUI
<<component>> Logic
<<component>> Network System
Server
<<component>> Reversi Server
<<component>> UI
<<component>> Game Logic
<<component>> Network System
Client
<<component>> Reversi Client
<<component>> UI
<<component>> Logic
<<component>> Network System
Client
<<component>> Reversi Client
<<component>> UI
<<component>> Logic
<<component>> Network System

*Illustration 3: System Overview*

# 3 Component description

## *3.1 Reversi Server*

### 3.1.1 UI

A real UI is not implemented, just a few debug messages printed to STDOUT

### 3.1.2 Game Logic

This is where the game server is implemented, it contains the game logic, data structures and protocol realization necessary to manage  games, It is the upper most layer of the server

This game layer uses the service provided by the below transport layer to establish communication between clients and servers.

### 3.1.3 Network System

The network system consists on the communication parts that are tied to the underlying TCP/IP protocol and the TCP/IP primitives.

## *3.2 Reversi Client*

## *3.3 UI*

Client GUI displays the current state of the game it also acts as the interface between the user and the program logic.

## *3.4 Logic*

Client side logic keeps the state of the connection, allowing user to make actions according to that state, for example clients can't disconnect if they are not connected.

## *3.5 Network System*

Same as server-side network system

## *3.6 Game Protocol*

The game protocol is implemented on both sides, Its responsibility is to package and handle all the messages between clients and the server

# 4 Layer model

## 4.1 Layer diagram

This communication Architecture Diagram specifies the entities communicating within the game layer and the transport layer. It shows the entities of each layer and the communication channels between them.

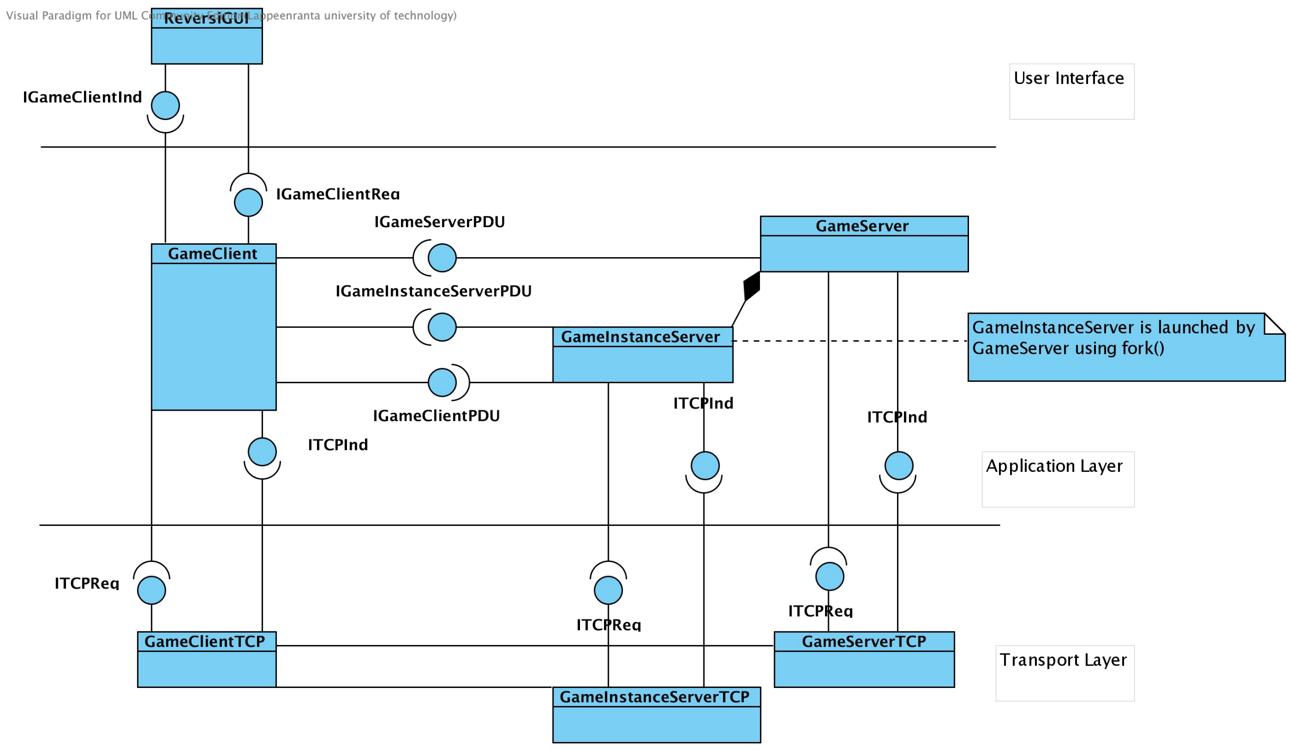


*Illustration 4: Layer Model*

## 4.2 Layers description

### 4.2.1 Game Layer

This layer is in charge of keeping the games running, ensuring games are played following game's rules and maintaining the client state in consistent state, it implements all the behavior of both client and server sides

### 4.2.2 Network Layer

Network layer is composed by all the components that are closely tied to the underlying TCP/IP protocol.

## 4.3 Layer interfaces

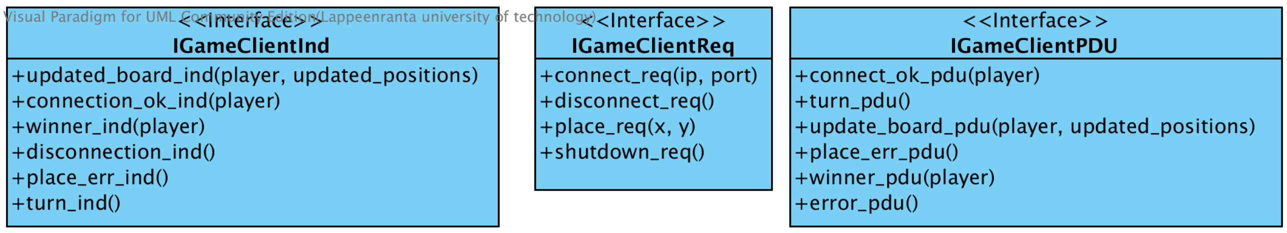


*Illustration 5: Client interfaces*

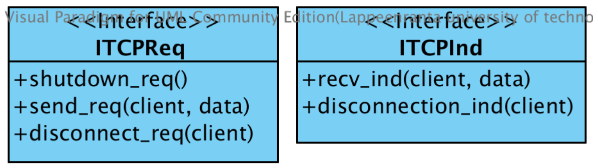


*Illustration 6: TCP interfaces*

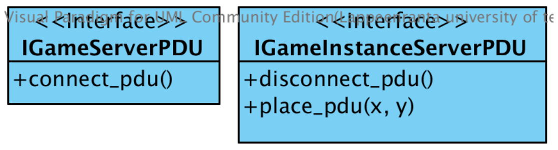


*Illustration 7: Server interfaces*

# 5 Protocol data units

## 5.1 Abstract Message Definitions

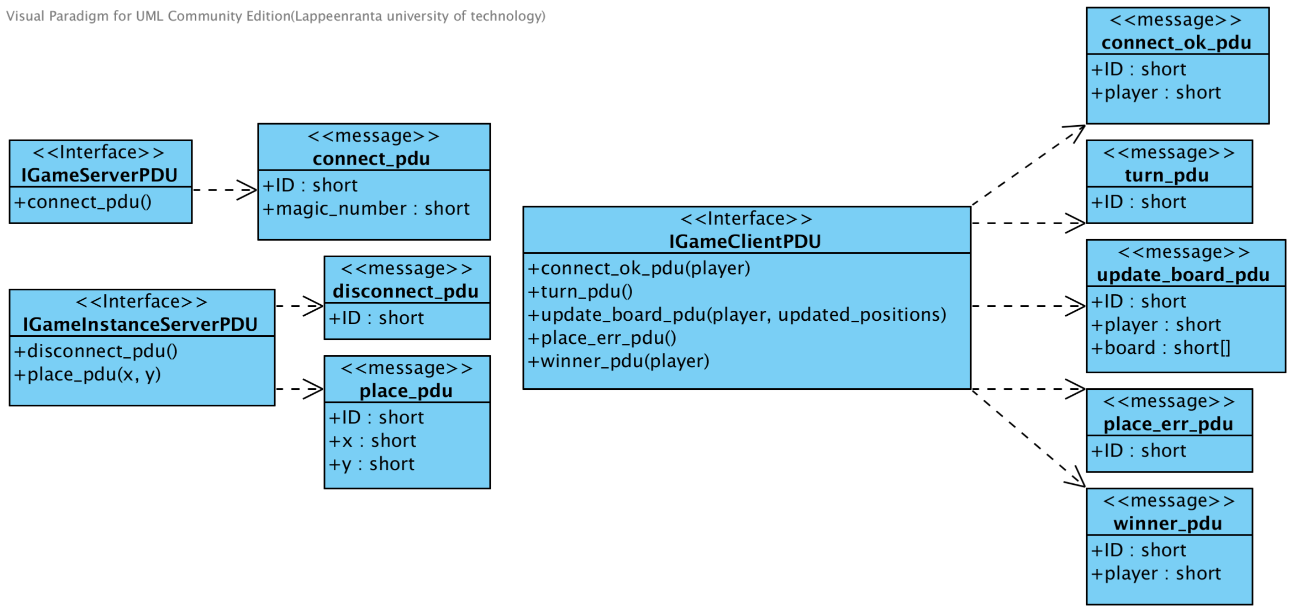


*Illustration 8: Abstract PDUs*

## 5.2 Concrete Message Definitions

NOTE: type of msg_size and player are type short

| **error_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | ID | ID |

Reversi Specification CSPA - 2013

| **connect_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | ID | ID |

| **connect_ok_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | ID | ID |

| **disconnect_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | ID | ID |

| **place_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | ID | ID |
| **1** | x | x | y | y |

| **turn_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | ID | ID |

| **update_board_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | player | player |
| **1** | x | x | y | y |
| **2** | x | x | y | y |
| **3** | … | … | … | … |
| **4** | … | … | … | … |
| **5** | … | … | … | … |
| **6** | … | … | … | … |
| **7** | … | … | … | … |
| **8** | … | … | … | … |
| **9** | … | … | … | … |
| **10** | … | … | … | … |
| **11** | … | … | … | … |
| **12** | … | … | … | … |
| **13** | x | x | y | y |

| **place_error_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | ID | ID |

| **winner_pdu** | | | | |
|---|---|---|---|---|
| | **0..7** | **8..15** | **16..23** | **24..31** |
| **0** | msg_size | msg_size | ID | ID |
| **1** | player | player | | |

### 5.3 Message IDs enum

| Messages IDs | Value |
|---|---|
| ERROR | -1 |
| CLIENT_CONNECT | 0 |
| CLIENT_CONNECT_OK | 1 |
| CLIENT_QUIT | 2 |
| TURN | 3 |
| PLACE | 4 |
| PLACE_ERROR | 5 |
| UPDATE_BOARD | 6 |
| WINNER | 7 |

### 5.4 Player IDs enum

| Player | Value |
|---|---|
| EMPTY | 0 |
| WHITE | 1 |
| BLACK | 2 |

### 5.5 Type catalog

| Name | Type | Bit length |
|---|---|---|
| short | integer | 16 |

## 6 State machine diagrams

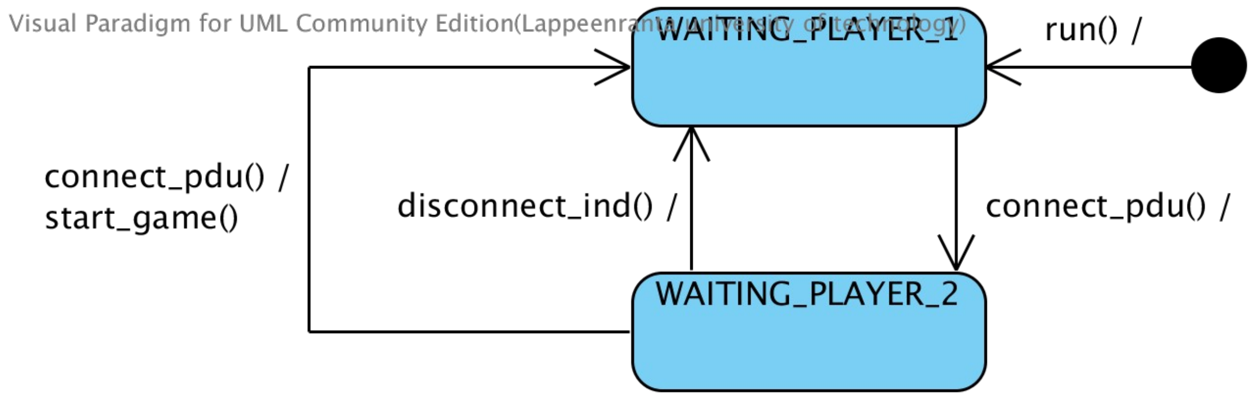

Visual Paradigm for UML Community Edition(Lappeenranta university of technology)
WAITING_PLAYER_1
run() /
connect_pdu() / start_game()
disconnect_ind() /
connect_pdu() /
WAITING_PLAYER_2

*Illustration 9: GameServer state machine diagram*

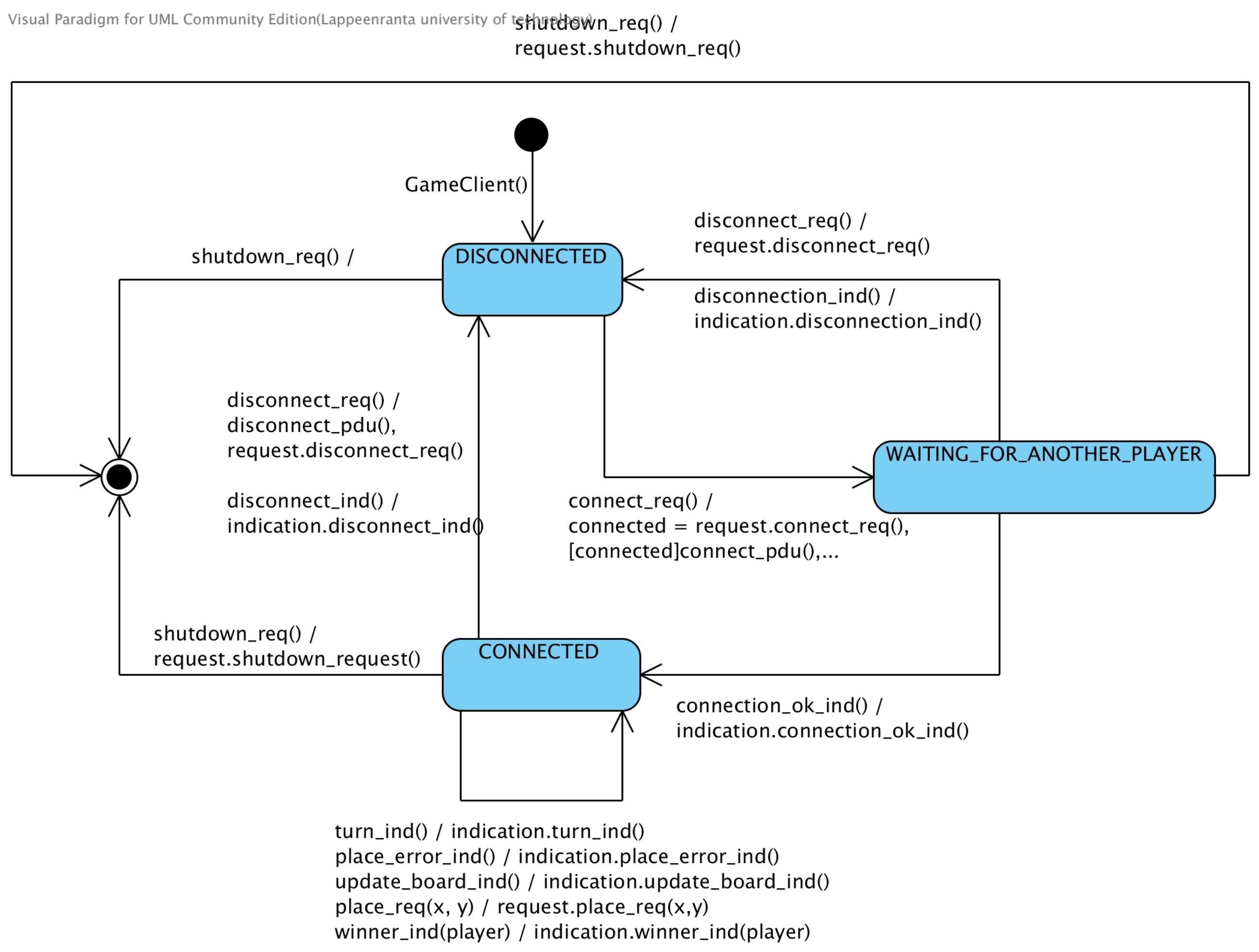

Visual Paradigm for UML Community Edition(Lappeenranta university of technology)
shutdown_req() /
request.shutdown_req()
GameClient()
DISCONNECTED
disconnect_req() /
request.disconnect_req()
shutdown_req() /
disconnection_ind() /
indication.disconnection_ind()
disconnect_req() /
disconnect_pdu(),
request.disconnect_req()
WAITING_FOR_ANOTHER_PLAYER
disconnect_ind() /
indication.disconnect_ind()
connect_req() /
connected = request.connect_req(),
[connected]connect_pdu(),...
shutdown_req() /
request.shutdown_request()
CONNECTED
connection_ok_ind() /
indication.connection_ok_ind()
turn_ind() / indication.turn_ind()
place_error_ind() / indication.place_error_ind()
update_board_ind() / indication.update_board_ind()
place_req(x, y) / request.place_req(x,y)
winner_ind(player) / indication.winner_ind(player)

*Illustration 10: GameClient state machine diagram*

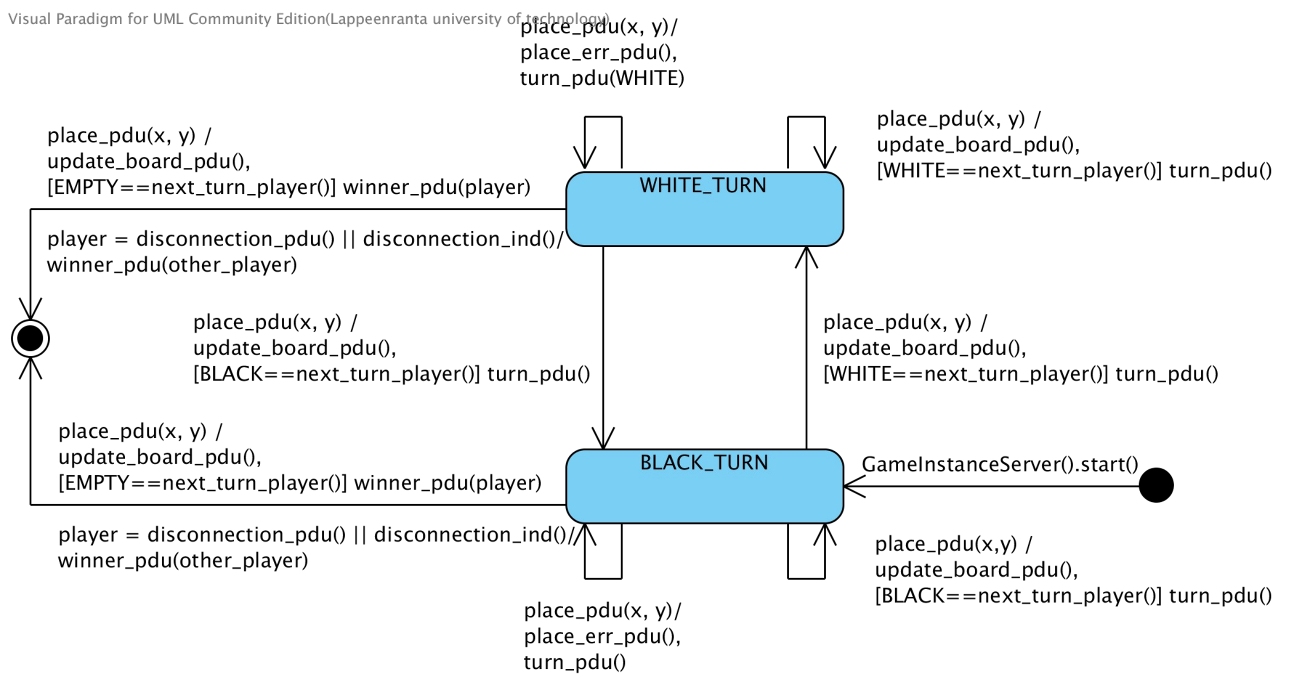

Visual Paradigm for UML Community Edition(Lappeenranta university of technology)
place_pdu(x, y)/
place_err_pdu(),
turn_pdu(WHITE)
place_pdu(x, y) /
update_board_pdu(),
[WHITE==next_turn_player()] turn_pdu()
place_pdu(x, y) /
update_board_pdu(),
[EMPTY==next_turn_player()] winner_pdu(player)
WHITE_TURN
player = disconnection_pdu() || disconnection_ind()/
winner_pdu(other_player)
place_pdu(x, y) /
update_board_pdu(),
[BLACK==next_turn_player()] turn_pdu()
place_pdu(x, y) /
update_board_pdu(),
[WHITE==next_turn_player()] turn_pdu()
place_pdu(x, y) /
update_board_pdu(),
[EMPTY==next_turn_player()] winner_pdu(player)
BLACK_TURN
GameInstanceServer().start()
player = disconnection_pdu() || disconnection_ind()/
winner_pdu(other_player)
place_pdu(x,y) /
update_board_pdu(),
[BLACK==next_turn_player()] turn_pdu()
place_pdu(x, y)/
place_err_pdu(),
turn_pdu()

*Illustration 11: GameServerInstance state machine diagram*

# 7 Use Case model

## 7.1 Requirements

| ID | Requirements |
|---|---|
| RC1 | Use TCP to transfer game messages |
| RC2 | Connect to server |
| RC3 | Make move |
| RC4 | End game |
| RC5 | Quit game |

*Table 1: Game Protocol Requirements*

## 7.2 Use Case diagram

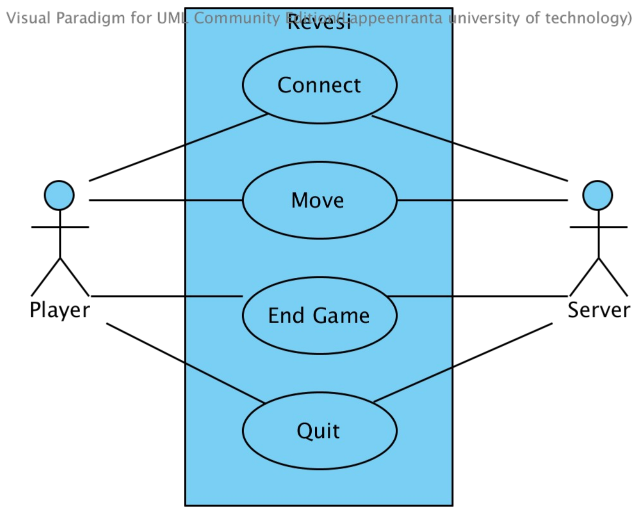

Visual Paradigm for UML Community Edition [not for commercial use] (Lappeenranta university of technology)
Revesi
Connect
Move
End Game
Quit
Player
Server

*Illustration 12: Use case diagram*

## 7.3 Use case sequence diagrams

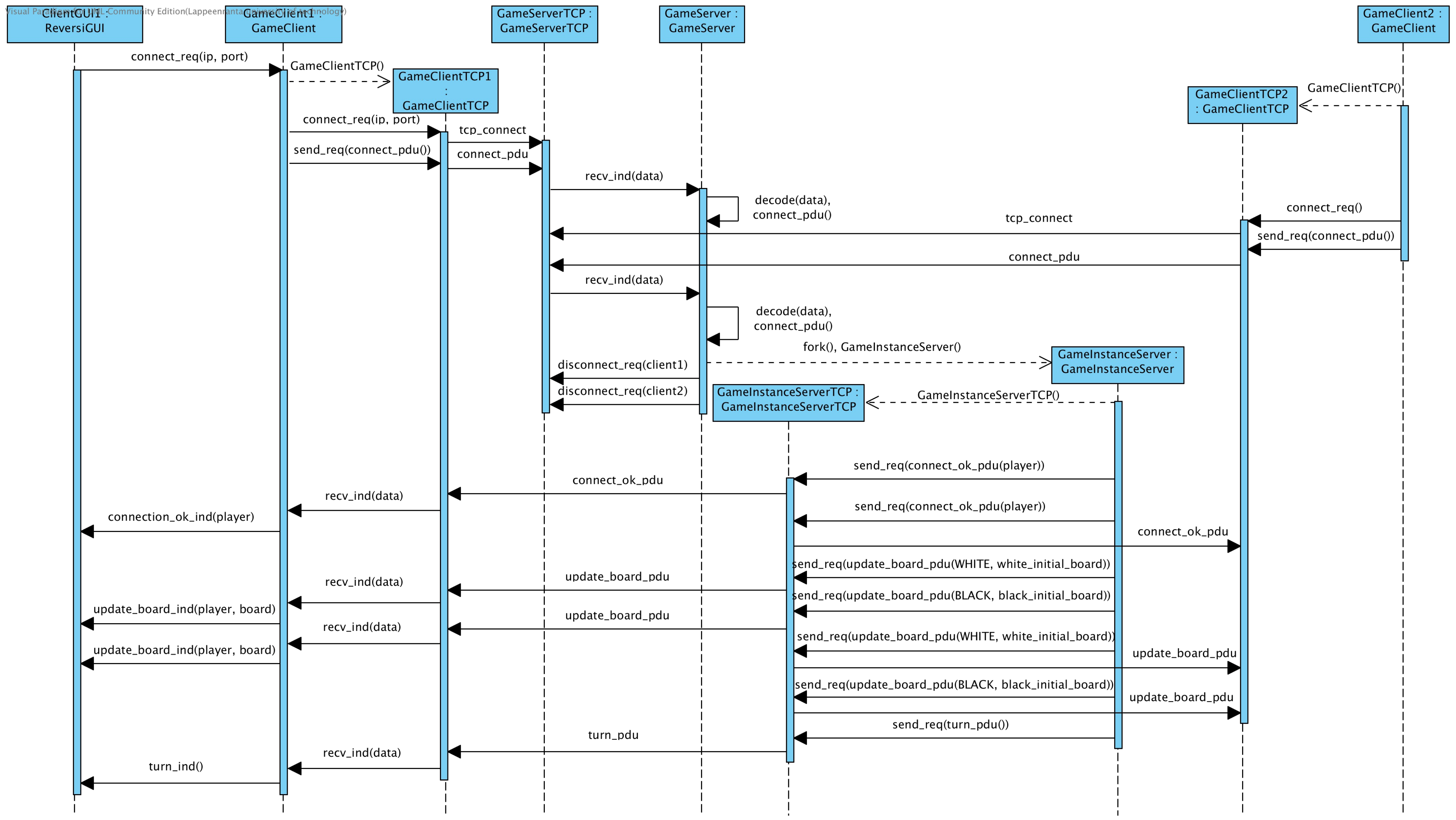
Use case diagrams are located in Appendix 1

## References:

[TCP] RFC 793 http://www.ietf.org/rfc/rfc793.txt (as retrieved on 2013)

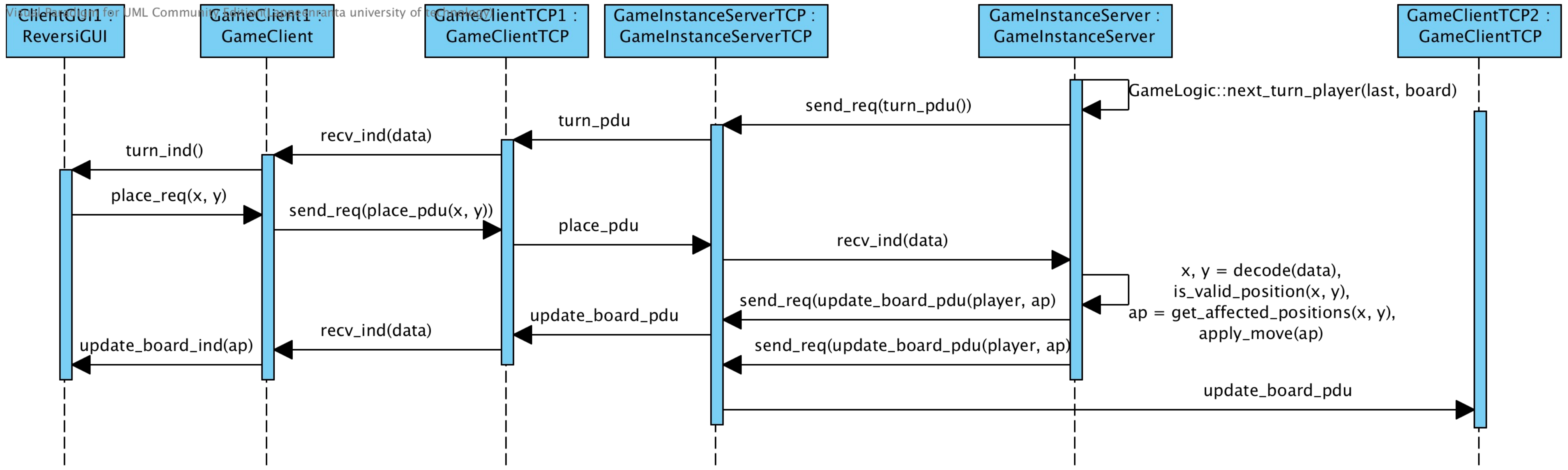
[WOF] World Othello Federation http://www.worldothellofederation.com/rules_english.asp (as retrevied on 2013)

## Appenix 1: Use case sequence diagrams



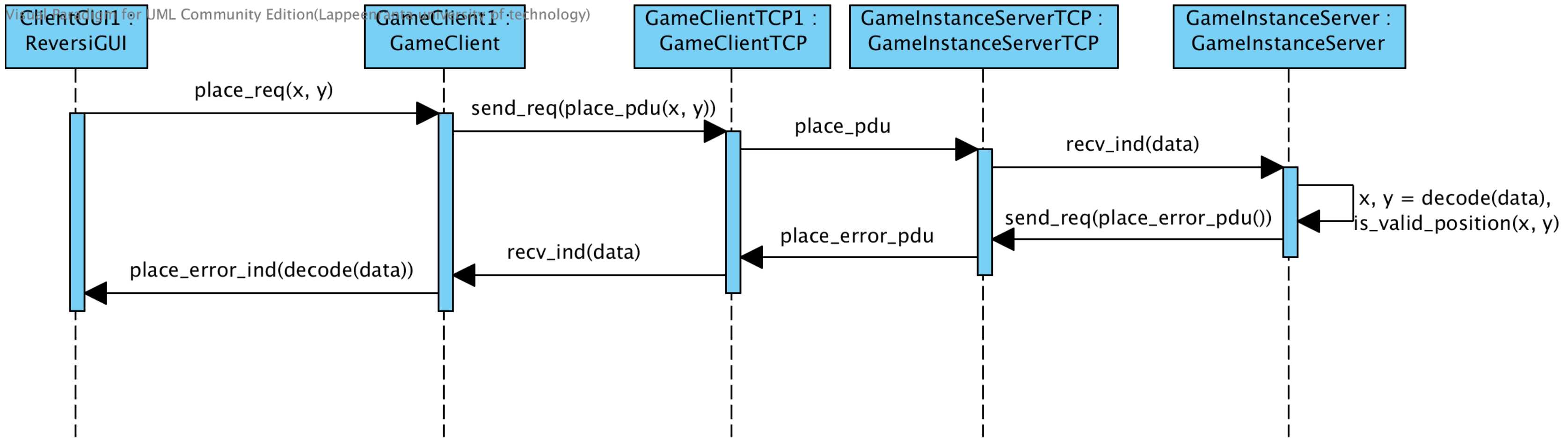
Visual Paradigm for UML Community Edition(Lappeenranta University of Technology)
ClientGUI1 : ReversiGUI
GameClient1 : GameClient
GameServerTCP : GameServerTCP
GameServer : GameServer
GameClient2 : GameClient
connect_req(ip, port)
GameClientTCP()
GameClientTCP1 : GameClientTCP
GameClientTCP()
GameClientTCP2 : GameClientTCP
connect_req(ip, port)
tcp_connect
send_req(connect_pdu())
connect_pdu
recv_ind(data)
decode(data), connect_pdu()
connect_req()
tcp_connect
send_req(connect_pdu())
connect_pdu
recv_ind(data)
decode(data), connect_pdu()
fork(), GameInstanceServer()
GameInstanceServer : GameInstanceServer
disconnect_req(client1)
disconnect_req(client2)
GameInstanceServerTCP : GameInstanceServerTCP
GameInstanceServerTCP()
send_req(connect_ok_pdu(player))
connect_ok_pdu
recv_ind(data)
connection_ok_ind(player)
send_req(connect_ok_pdu(player))
connect_ok_pdu
send_req(update_board_pdu(WHITE, white_initial_board))
update_board_pdu
recv_ind(data)
update_board_ind(player, board)
send_req(update_board_pdu(BLACK, black_initial_board))
update_board_pdu
recv_ind(data)
update_board_ind(player, board)
send_req(update_board_pdu(WHITE, white_initial_board))
update_board_pdu
send_req(update_board_pdu(BLACK, black_initial_board))
update_board_pdu
send_req(turn_pdu())
turn_pdu
recv_ind(data)
turn_ind()

*Illustration 13: Connect use case*



ClientGUI1 : ReversiGUI
GameClient1 : GameClient
GameClientTCP1 : GameClientTCP
GameInstanceServerTCP : GameInstanceServerTCP
GameInstanceServer : GameInstanceServer
GameClientTCP2 : GameClientTCP
GameLogic::next_turn_player(last, board)
send_req(turn_pdu())
turn_pdu
recv_ind(data)
turn_ind()
place_req(x, y)
send_req(place_pdu(x, y))
place_pdu
recv_ind(data)
x, y = decode(data),
is_valid_position(x, y),
ap = get_affected_positions(x, y),
apply_move(ap)
send_req(update_board_pdu(player, ap)
update_board_pdu
recv_ind(data)
update_board_ind(ap)
send_req(update_board_pdu(player, ap)
update_board_pdu

*Illustration 14: Move OK use case*



ClientGUI1 : ReversiGUI
GameClient1 : GameClient
GameClientTCP1 : GameClientTCP
GameInstanceServerTCP : GameInstanceServerTCP
GameInstanceServer : GameInstanceServer
place_req(x, y)
send_req(place_pdu(x, y))
place_pdu
recv_ind(data)
x, y = decode(data),
is_valid_position(x, y)
send_req(place_error_pdu())
place_error_pdu
recv_ind(data)
place_error_ind(decode(data))

*Illustration 15: Move error use case*

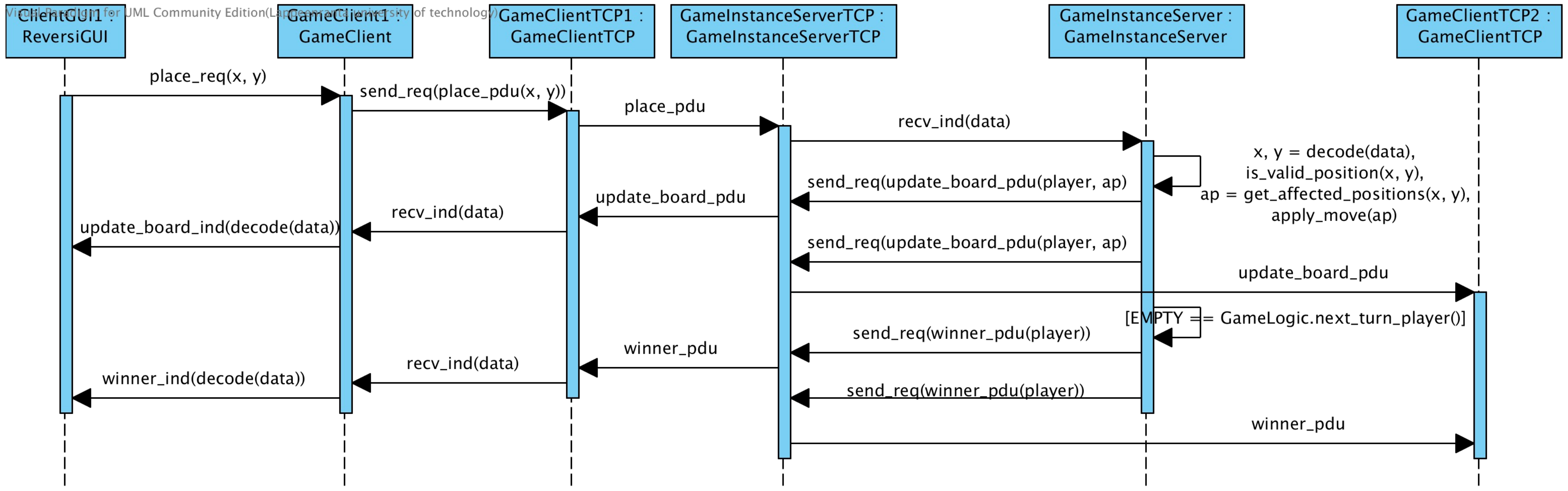

Visual Paradigm for UML Community Edition(Lappeenranta university of technology)
ClientGUI1 : ReversiGUI
GameClient1 : GameClient
GameClientTCP1 : GameClientTCP
GameInstanceServerTCP : GameInstanceServerTCP
GameInstanceServer : GameInstanceServer
GameClientTCP2 : GameClientTCP
place_req(x, y)
send_req(place_pdu(x, y))
place_pdu
recv_ind(data)
x, y = decode(data),
is_valid_position(x, y),
ap = get_affected_positions(x, y),
apply_move(ap)
send_req(update_board_pdu(player, ap)
update_board_pdu
recv_ind(data)
update_board_ind(decode(data))
send_req(update_board_pdu(player, ap)
update_board_pdu
[EMPTY == GameLogic.next_turn_player()]
send_req(winner_pdu(player))
winner_pdu
recv_ind(data)
winner_ind(decode(data))
send_req(winner_pdu(player))
winner_pdu

*Illustration 16: End game use case*

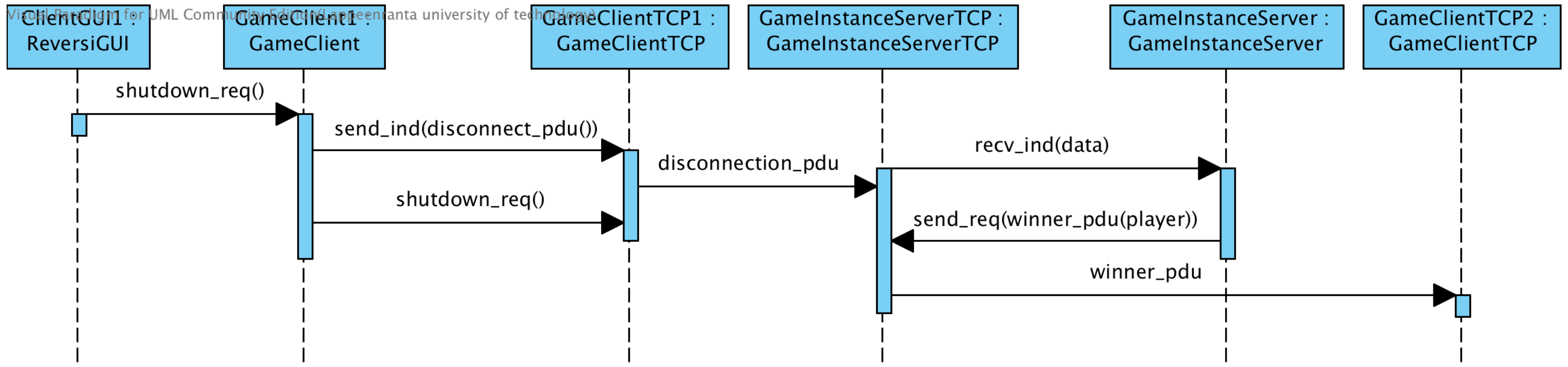

Visual Paradigm for UML Community Edition(Lappeenranta university of technology)
ClientGUI1 : ReversiGUI
GameClient1 : GameClient
GameClientTCP1 : GameClientTCP
GameInstanceServerTCP : GameInstanceServerTCP
GameInstanceServer : GameInstanceServer
GameClientTCP2 : GameClientTCP
shutdown_req()
send_ind(disconnect_pdu())
disconnection_pdu
recv_ind(data)
shutdown_req()
send_req(winner_pdu(player))
winner_pdu

*Illustration 17: Quit use case*