Faculty of Engineering
Computer Engineering Department

CMPN 402/CMP402a
Fall 2019

Go Communication Protocol

This document describes the communication details between the server and the clients in the Go Contest in CMPN402 and CMP402A Fall 2019. The document is divided into 5 sections:

- Websockets.
- Formats and Enumerations.
- Scenarios.
- Server State Machine.
- Client State Machine.

Websockets

The communication will be done using Websockets protocol version 13 as defined by RFC6455 (https://tools.ietf.org/html/rfc6455). All of the data will be passed using string messages in a json format. The server will be listening over a port that will be specified at the beginning of the contest.

The server will implement a heartbeat system to check for client disconnections. Initially, every client is assumed to be alive and every ping interval, the server will do the following:

- Check if any client is not flagged as alive and terminate its connection.
- Send a ping message to every alive client then remove the alive flag.
- If the client replies with a pong message, it is flagged as alive again.

The ping interval is configurable.

Formats and Enumerations

Common Formats:

- **Whole** is an unsigned integer (whole number)**, Decimal** is a floating point number and **String** is a string**.**
- **Point** denotes a point on the board. It has the format: {"row": **Whole** , "column": **Whole** }.
- **Time** is used to denote time in milliseconds. It is represented as a **Whole** number.
- **Color** is used to denote the player's color. It is represented as a string with 2 possible values: "W" for white, "B" for black.
- **Board** is used to denote a board state.
    - It is represented as an array containing elements of type **Row**. The length is equal to the number of rows in the board.
    - **Row** is represented as an array containing element of type **Color** for stones or **"."** if empty. The length is equal to the number of columns in the board.
- **Move** is used to denote a player move. It is represented as one of the following formats:
    - A "pass" move with the format {"type": "pass"}.
    - A "resign" move with the format {"type": "resign"}.
    - A "place" move with the format {"type": "place", "point": **Point**} where "point" denote the added stone's location.
- **GameState** is used to denote the current board and player states.
    - It has the format: {"board": **Board**, "players": {"B": **PlayerState**, "W": **PlayerState**}, "turn": **Color**}.
    - "Board" contains the board configuration and "turn" contains the color of the player with the current turn.
    - PlayerState has the format {"remainingTime": **Time**, "prisoners": **Whole**} where "remainingTime" contains the remaining time for the player and "prisoners" contains the number of stones captured by the player from the opponent.
- **GameConfiguration** contains all the game details at the start of a new session.
    - It has the format: {"initialState": **GameState**, "moveLog": [**LogEntry**], "komi": **number**, "ko": **boolean**, "scoringMethod": **string**, "prisonerScore": **number,** "idleDeltaTime"**: Time**}.
    - "moveLog" is an array containing the history of the game starting from the "initialState" till the current state. Each element in the array is of type **LogEntry**. In order to reconstruct the current state, the client has to apply all the moves in the log to the "initialState".
    - **LogEntry** has the format: {"move": **Move**, "deltaTime": **Time**} where the property "deltaTime" contains the time span between the last move and the current move.
    - The property "komi" contains the komi score added to the white player. It will always have the value of 6.5 in the competition.
    - The property "ko" is true if the ko rule is applied and false otherwise. It will always have the value of true in the competition.

    - The property “scoringMethod” can have only one of two values: “area” for area scoring and “territory” for territory scoring. It will always have the value “area” in the competition.
    - The “prisonerScore” is a factor multiplied by the number of enemy stones captured by the player before being added to their score. It will always have a value of 1 in the competition
    - The “idleDeltaTime” is the number of milliseconds spent in idle time between the last move and the checkpoint state. This number should be subtracted from the time of the current player after reconstructing the game state.
- **GameConfigurationV2** is the same as **GameConfiguration** but contains an extra field “finalStates” which is an array of **GameState** that contains the latest 2 states in chronological order (or 1 state only if the game does not have any moves yet). This is sent in place of **GameConfiguration** if the client chooses Protocol **v2**.
- **RemainingTime** contains the remaining time for each player.
    - It has the format {“B”: **Time**, “W”: **Time**}

Server to Client Message Formats:

- **NAME** is a message to request the client name**.**
    - It has the format { “type”: “NAME” }.
- **START** is a message to start the game.
    - It has the format { “type”: “START”, “configuration”: **GameConfiguration**, “color”: **Color**}.
    - “configuration” is the game configuration and “color” is the color of the player.
    - **Note**: If the client chooses Protocol **v2**, the “configuration” field will have the type **GameConfigurationV2**.
- **MOVE** is a message to inform a player about their opponent’s move.
    - It has the format { “type”: “START”, “move”: **Move**, “remainingTime”: **RemainingTime**}.
- **VALID** is a message to inform the player that their move is accepted.
    - It has the format { “type”: “VALID”, “remainingTime”: **RemainingTime**}.
- **INVALID** is a message to inform the player that their move is rejected.
    - It has the format { “type”: “INVALID”, “message”: **String**, “remainingTime”: **RemainingTime**}.
    - “Message” is a human readable string that contains the reason for rejecting the move. It is only designed for human debugging.
- **END** is a message to inform both players about ending the game including the results.
    - It has the format { “type”: “END”, “reason”: **Reason**, “winner”: **Color**, “players”: {“B”: **ScoreAndTime**, “W”: **ScoreAndTime**}.
    - Reason is a string and can have the value “resign”, “pass”, “timeout”, “pause”, “error”.
    - “Winner” will contain the color of the winner or “.” if it is a draw. If the reason is “pause” or “timeout”, the “winner” will always be “.”.
    - **ScoreAndTime** has the format {“score”: **Decimal**, “remainingTime”: **Time**}.

Client to Server Message Formats:

- **NAME** is a message to request the client name**.**
  - It has the format { “type”: “NAME”, “name”: String }.
  - **Note**: Optionally, this message can also contain a field “protocol” which can have one of the following values:
    - “**v1**” which is the default protocol and the server.
    - “**v2**” which will prompt the server that client needs the “finalStates” in the game configuration. See the **START** message and the common formats for more info.
  - If the “protocol” field is not included in the message, the server will default to Protocol **v1** for the sake of backward compatibility.
- **MOVE** is a message to inform the server about the agent’s move.
  - It has the format { “type”: “START”, “move”: **Move**}.

Table 1: Enumerations

Enumerations		
Message Types	**Server States**	**Client States**
NAME	INIT	INIT
START	IDLE	READY
MOVE		IDLE
VALID		THINKING
INVALID		AWAITING_MOVE_RESPONSE
END		

Scenarios

Initialization: Any new client will receive any empty NAME message to which it should reply with a NAME message containing its team name, then it should wait in the READY state. After the server picks two clients to play, the server should be ready to start a game.

Starting a game: The server will send a START message containing the game configuration, players' colors to both players. After reconstructing the game state, the client whose color is the same as the current turn should move to THINKING state while the other should move to IDLE state. **Hint**: the client should go to THINKING if its color is equal to the initial state turn and the move log length is even or if its color is not equal to the initial state turn and the move log length is odd. Otherwise, it should go to IDLE.

Ending a game: At any moment, the client should expect an END message which means that the game has ended. The message will include the final scores and the reason for ending the game (resign, pass, timeout, connection error or pause button pressed). The client should not exit and should return to READY state to wait for a new START message.

Doing a move: if a player wants to do a move, they should send a MOVE message to the server and await for the server response in the AWAITING_MOVE_RESPONSE. The server will either reply with a VALID or INVALID message to inform the client whether the move was valid or not. If the message is of type VALID, the client should move to the IDLE state, otherwise it should return to the THINKING state. If the move was valid, the server will also send the MOVE message to the opponent. Messages from the server of type VALID, INVALID and MOVE will all include the remaining time of both players.

Disconnection: Any MOVE will cause the server to save a new checkpoint. If one or more clients disconnect from the server, the server will save the remaining time for both players into the last checkpoint, and send END to connected clients with reason="error" then it will go to the initial state to listen for connection requests. Clients that receive the END message should stay READY for the game to continue. The disconnected client should request a new connection then go through the initialization sequence. After both clients are connected, the server can resume the game and send a new START message containing the checkpoint state to both clients.

Server State Machine

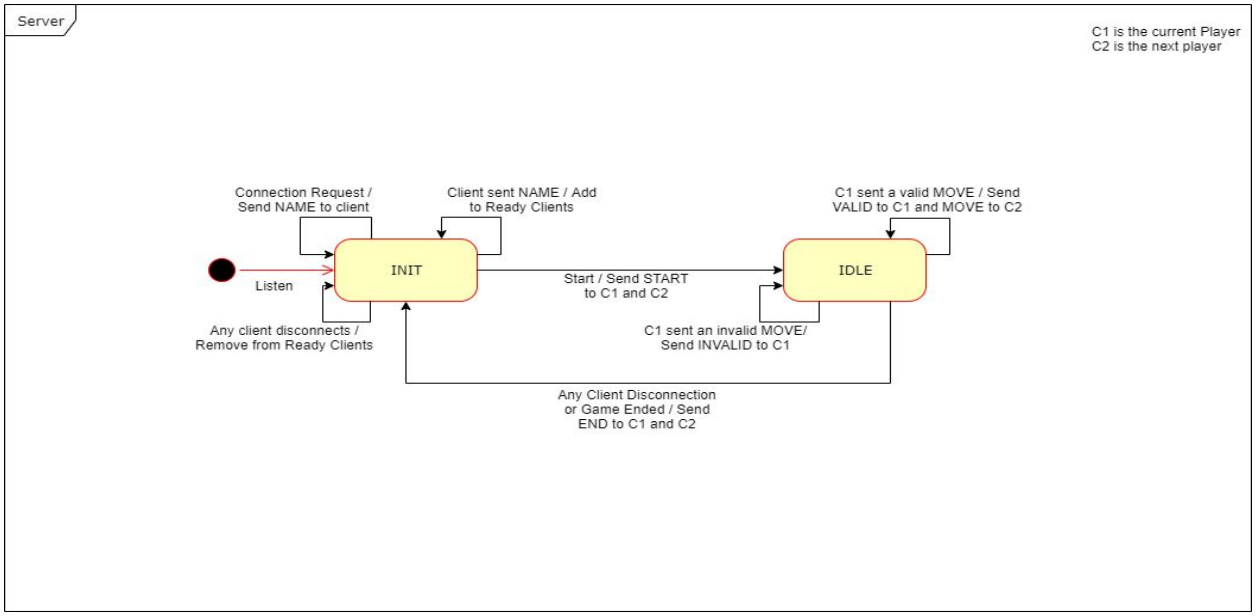

Figure 1: Server State Machine

State: INIT

- Input: Connection request from Client. Goto: **INIT**
  - Send NAME to client
- Input: Received NAME from Client. Goto: **INIT**
  - Add to ready clients.
- Input: Any client disconnects. Goto: **INIT**
  - Remove from ready clients.
- Input: Start Game Button Clicked or Recovery from Connection Error. Goto: **IDLE**
  - Send START to C1 and C2 which each having a different value for “color”

State: IDLE

- Input: Received MOVE from C1 and it is invalid. Goto: **IDLE**
  - Send INVALID to C1
- Input: Received MOVE from C1 and it is valid. Goto: **IDLE**
  - Send VALID to C1
  - Send MOVE to C2
  - Swap C1 and C2.
- Input: Game ending condition occurred. Goto: **READY**
  - Send END to C1 and C2.
- Input: Disconnection from any of the clients. Goto: **INIT**
  - Send END to any connected players with reason=”error”.

Client State Machine

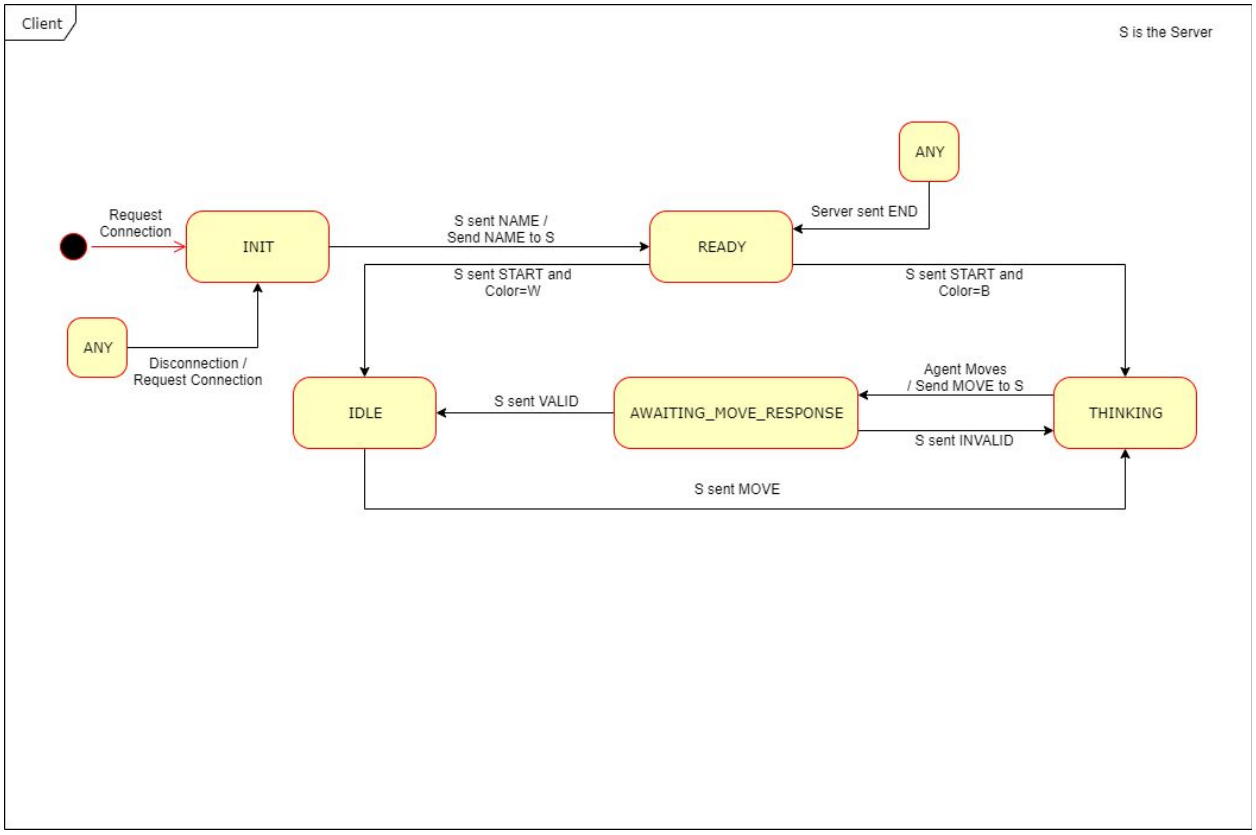

Figure 2: Client State Machine

State: INIT

- Input: Received NAME from S. Goto: **READY**
    - Send NAME to S.

State: READY

- Input: Received START from S and Color is equal to the current turn. Goto: **THINKING**
- Input: Received START from S and Color is not equal to the current turn. Goto: **IDLE**

State: THINKING

- Input: Agent moves. Goto: **AWAITING_MOVE_RESPONSE**
    - Send MOVE to S.

State: AWAITING_MOVE_RESPONSE

- Input: Received INVALID from S. Goto: **THINKING**
- Input: Received VALID from S. Goto: **IDLE**

State: IDLE

- Input: Received MOVE from S. Goto: **THINKING**

State: * ANY

- Input: Received END from S. Goto: **READY**
- Input: Disconnection from Server. Goto: **INIT**
    - Request new connection.