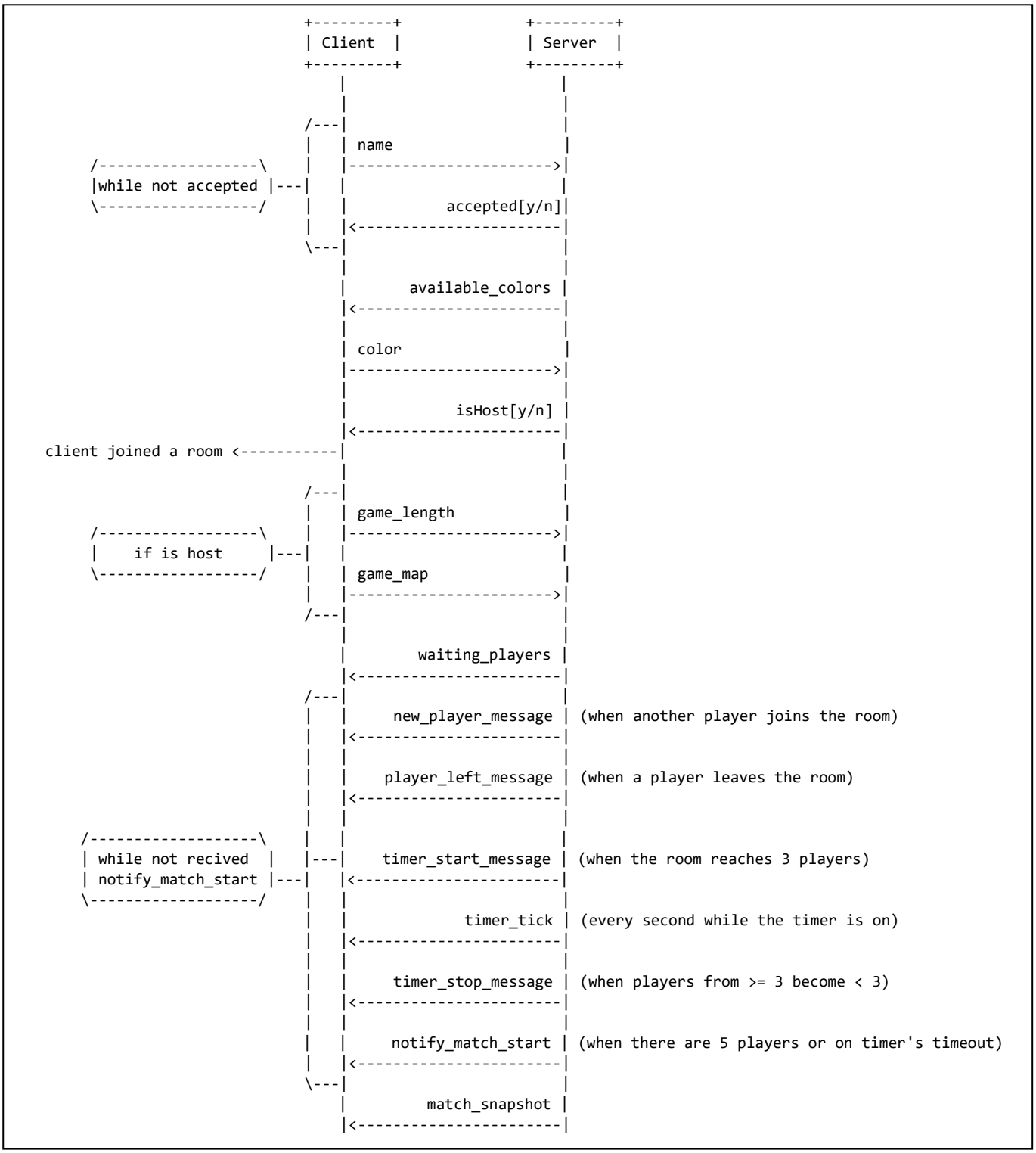
Communication protocol

The following is a binary-protocol.
Client and Server only exchange fundamental data types and serializable objects.
Except from the login phase, the Server always starts the communication.
The whole protocol can be logically divided into four phases:

Login and room join



name: String

accepted: boolean

available_colors = {$color_0$, …, $color_{n-1}$} : Strings

colour: String

isHost: boolean

game_length: int

game_map: int

waiting_players = {$player_0$, … , $player_{n-1}$}: Strings

new_player_message: String

player_left_message: String

timer_start_message: String

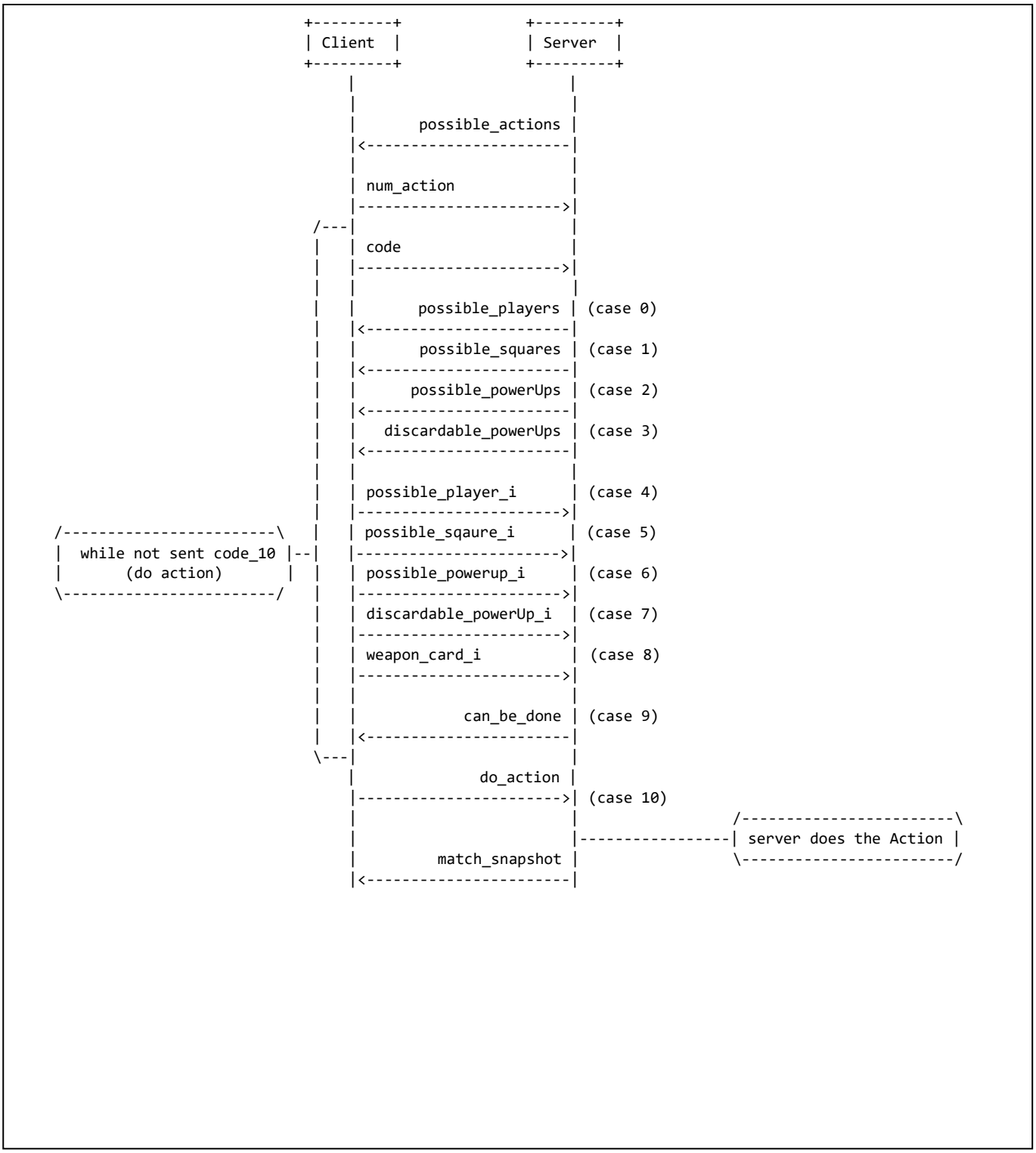
timer_tick: int

timer_stop_message: String

notify_match_start: boolean

match_snapshot: Serializable Object MatchSnapshot

Action


Client
Server
possible_actions
num_action
code
possible_players
possible_squares
possible_powerUps
discardable_powerUps
possible_player_i
possible_sqaure_i
possible_powerup_i
discardable_powerUp_i
weapon_card_i
can_be_done
do_action
match_snapshot
(case 0)
(case 1)
(case 2)
(case 3)
(case 4)
(case 5)
(case 6)
(case 7)
(case 8)
(case 9)
(case 10)
while not sent code_10
(do action)
server does the Action

possible_actions = {RemoteAction$_0$, …, RemoteAction$_{n-1}$} : Objects

num_action: int $0 \leq num\_action \leq n-1,\ n = \ len\ of\ possible\_actions$

code: int

possible_players = $\{PlayerName_1, \dots, PlayerName_{n-1}\}$: Strings

possible_squares = $\{SquareName_1, \dots, SquareName_{n-1}\}$: Strings

possible_powerups = $\{PoweupName_1, \dots, PowerupName_{n-1}\}$: Strings

discardable_powerups = $\{PowerupName_1, \dots, PowerupName_{n-1}\}$: Strings

possible_player_i: int $0 \leq possible\_player\_i \ \leq n-1,\ n = len\ of\ possible\_players$

possible_square_i: int $0 \leq possible\_player\_i \ \leq n-1,\ n = len\ of\ possible\_squares$

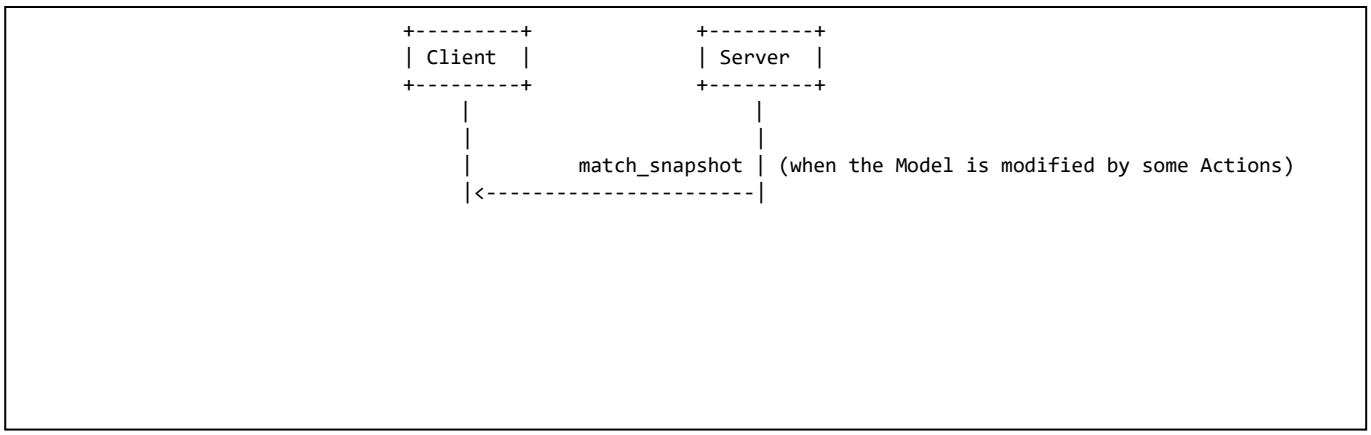
possible_powerup_i: int $0 \leq possible\_powerup\_i \ \leq n-1,\ n = len\ of\ possible\_powerups$

discardable_powerup_i: int $0 \leq discardable\_powerup\_i \ \leq n-1,\ n = len\ of\ discardable\_powerups$

weapon_card_i: int $0 \leq weapon\_card \ \leq n-1,\ n = count\ of\ player's\ weapon$

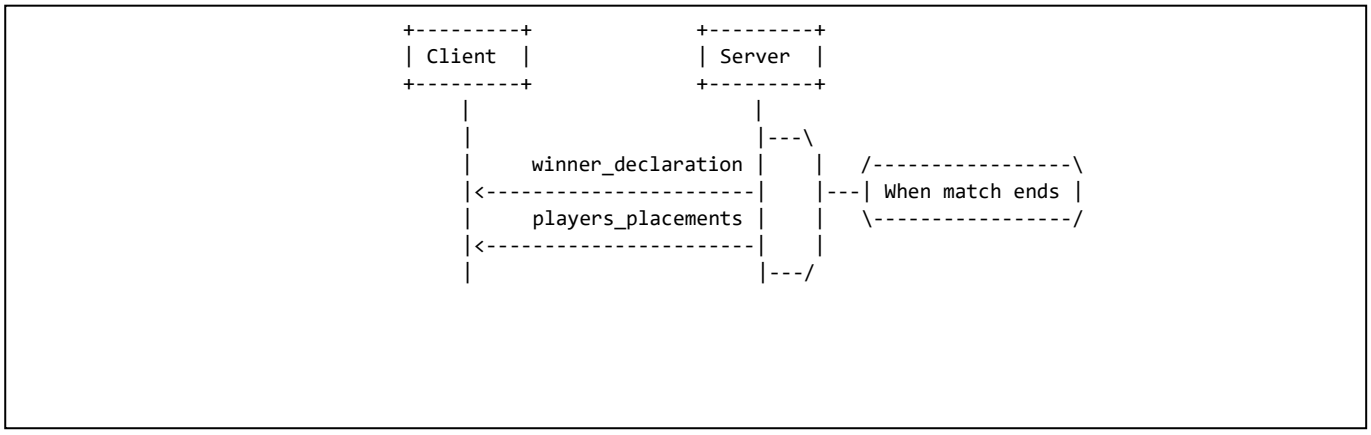
can_be_done: boolean

On match update



match_snapshot: Serializable Object MatchSnapshot

On match end



winner_declaration: String

player_placements = {$player_0$, … , $player_{n-1}$}: Strings

Objects description

Action: an action is the way the model can be modified after that a match begins. Every action represents an "atomic" step; for example, with a MoveAction the player can move himself into the map and with a ShootAction he can shoot other players.

With a **RemoteAction** a player interacts with an action (and then he can modify the model).

Server sends a list of possible (remote) actions and waits the client to reply.

With a RemoteAction a player can **get**

- Players-targets (*possible_players*)
- Squares-targets (*possible_squares*)
- Possible-powerups (possible_powerups)
- Discardable-powerups (*discardable_powerups)*

and can **notify** server about the choice of:

- Player_target (*player_target_i*)
- Square target (*square_target_i*)
- Possible powerup (*possible_powerup_i*)
- Discardable powerup (*discardable-powerup_i*)
- Weapon card (weapon_card_i)

Finally, when is possible (can_be_done), the player can do the action with the selected targets and using the selected powerups (code 10).

The exact modalities are descripted upside.

PlayerSnapshot: Contains info like: like name, ammo, color, cards.

SquareSnapshot: Contains info like: types of borders, players in the square.

GameboardSnapshot: Contains info like: list of SquareSnapshots, skulls, mapType.

MatchSnapshot: Contains the GameBoardSnapshot and the list of PlayerSnapshots (playing the match).