Among AIs - Protocols cheat sheet

November 21st, 2020 - final Proto, initial Alpha release

Chat protocol

Connection

TCP/IP, server listening on margot.di.unipi.it port 8422. A WebSocket bridge is provided at ws://margot.di.unipi.it:8522.

Format

Line-oriented; UCS-2 symbols, UTF-8 encoding. Comands are ASCII; argument can contain non-ASCII symbols, but cannot contain \n. Space (ASCII 32) used as separator.

Timing

Completely asynchronous; both messages from clients to server and messages from server to clients do not produce nor expect a response. Errors are not reported, invalid commands are silently ignored. Connections are maintaned for up to 15 minutes without any activity, after which they are silently closed.

Messages

Client-to-server

NAME *username*	Declares this client's *username*; the name is public and will identify this user on all future messages. Once set, the name cannot be changed (for the duration of the connection). Clients **must** use NAME as their first command (all other commands will be ignored until a name is set).
JOIN *channel*	Joins the named *channel*; after this command, the client will start receiving messages posted to that channel. If the client has already joined the channel, this is a no-op.
LEAVE *channel*	Leaves the named *channel*; after this command, the client will no longer receive messages posted to that channel. If the client is not registered to the channel, this is a no-op.
POST *channel message*	Posts the given *message* (a single line of text, whitespace is preserved) to the named *channel*. The client needs not be registered on a channel in order to post to that channel.

Server-to-client

channel username msg	Signals that *username* POSTed the given *msg* on the named *channel*.

Conventions

1. *username*s need not be unique, not even in the same channel, but it is a useful convention to try to use unique names. All references to a given *channel* refer the same channel, so there cannot be multiple channels with the same name.
2. Usernames starting with @ and channel names starting with # are reserved for system use.
3. Both usernames and channel names must consist of a single word (i.e., they cannot contain whitespace); for readibility — and to help humans that might have to type them — it is advides to use simple, short, descriptive names.
4. For the Alpha release cycle, AI clients in public matches should use usernames starting with their group moniker (at least in “public” channels), to simplify identification and debugging.

Game protocol

Connection

TCP/IP, server listening on margot.di.unipi.it port 8421. A WebSocket bridge is provided at ws://margot.di.unipi.it:8521.

Format

Line-oriented; UCS-2 symbols, UTF-8 encoding. Comands are ASCII; argument can contain non-ASCII symbols, but cannot contain \n. Space (ASCII 32) used as separator.

Timing

Strictly Request/Response protocol; all actions initiated by clients, and processed in-order by the server; all requests will be paired with a response containing error status and possibly information about the outcome of the command.
Connections are maintained for up to 50 seconds without any activity, after which they are silently closed. Commands cannot be sent more frequently than once every 0.5 seconds, except when stated otherwise. In other words, commands can be issued at an interval t between them, with $0.5 < t < 50$ seconds.

Chat & Logs

Many commands cause the Game Server to emit lines in either Global chat (channel #GLOBAL) or in Game chat (channel named after the game name). Specific messages are documented below.
The same goes for game logs; all requests and responses from players are logged, as well as other messages. All log messages are prefixed by a timestamp, and a log tag (documented below).
Log entries for requests are tagged with REQUEST-FROM <player> followed by a copy of the specific command, whereas responses are tagged with RESPONSE-TO <player> followed by the response line. The contents of LONG-form responses (i.e., LOOK and STATUS) are only logged periodically.

Commands

Service

Messages that represent generic utilities, not particularly tied to any in-game activity.

Req	**<game> NOP** Does nothing (but resets the command timer).
Res	**OK** or **ERROR <code> <text>**. It is considered an error to issue a NOP before 10s have passed since the previous command (the intended use of NOPs being to hold a connection alive while there are no other commands to issue, and considering that the connection is maintained for 50s). Since this case is meant to protect the server from flooding, errors here are fatal (i.e., the client is disconnected). Specific Responses Ok No operation ERROR 510 Don't abuse NOPs!

Bootstrap

Messages to create games and join games that are in "Lobby mode".

Req	**NEW <game> <kind>** Creates a new game, with identifier <game>. All other commands will be preceded by this identifier. The creator can specify a specific <kind> of game to create; based on the kind, default parameters of the game may vary. It is envisioned that we might have kinds for training purposes (with reduced time limits) and for official league purposes (with specific team-assignment strategies); we might also have kinds to request for special maps. In the Proto release cycle, <kind> is ignored.
Res	**OK Created** or **ERROR <code> <text>** Only on a response of OK the game will be created, in "Lobby mode". Specific Responses OK Created ERROR 401 Too fast

	ERROR 502 Attempted to open lobby in invalid state
Chat	<game> Lobby opened.

Req	**<game> JOIN <player-name> <nature> <role> <user-info>** Joins the <game>, assuming name <player-name> for display purposes; declares <nature> (H=human, AI=AI). The <role> is ignored for Proto, but may impact gameplay for Alpha. The <user-info> is any arbitrary text, to be used for debugging purposes (e.g., internal codename and version of an IA, or which user is running the AI, etc.)
Res	**OK <team> <loyalty>** or **ERROR <code> <text>** On an OK response, the player is accepted, belongs to the <team> (assigned by the server), and has the given secret <loyalty> (also assigned by the server). Both <team> and <loyalty> are encoded as 0 or 1. The player will be awarded success points if the team indicated by <loyalty> wins the match. Specific Responses OK team=<team> loyalty=<loyalty> ERROR 401 Too fast ERROR 404 Game not found ERROR 410 Player name already taken in this game ERROR 502 Player cannot join ERROR 502 Game must be in open lobby ERROR 502 Both teams full
Chat	<game> <player-name> joined the game.

Req	**<game> START** Requests that the <game> be started. The <game> must curently be in Lobby mode, and there must be enough players to play a match (i.e., at least 1 per team). Only the client that created the game with NEW can start it. Common practice would dictate that the player issuing CREATE should not issue START until a sufficient amount of time has passed (to allow time to other players to join the lobby).
Res	**OK** or **ERROR <code> <text>** On an OK response, the player can start issuing commands from the Match protocol. On ERROR, the match is aborted and the game transitions to state 5. Specific Responses OK Game started ERROR 502 Can only start while in lobby ERROR 502 Only creator can start a game ERROR 502 Need two non-empty teams to start ERROR 502 Invalid player in lobby state while starting game
Chat	<game> Now starting!

Battleground

Commands to play the battle on the ground (moving, shooting, looking, and status info).

Req	**<game> LOOK** Requests a map of the terrain. The map may be full, or include “unknown” markers for fog-of-war, etc. We expect that in future versions of the protocol, additional arguments could be added (e.g., look in a particular direction)
Res	**OK LONG** / **<map>** / **«ENDOFMAP»** or **ERROR <code> <text>** On an OK response, the (possibly obfuscated) current state of the map is returned. The map will be provided on multiple lines following the “OK” line, with the given termination marker, to simplify debugging of the protocol. It is possible that in future versions of the protocol, the <map> will be a low-resolution screenshot of the view from the current players’s position. Specific Responses OK LONG … «...» - long answer, with terminator. ERROR 502 <exception> (no specific errors exist right now)

Req	**<game> MOVE <direction>** Move the player 1 step from its current position in the given <direction> (encoded as: N, E, S, W with obvious meaning).
Res	**OK** or **ERROR <code> <text>** On OK, the player will have attempted moving in the given <direction>. On ERROR, the player is still in its previous position. Moving can have side-effects (both on the player and on the map), depending on where the player moved to. Specific Responses OK blocked — the command was valid, but the player could not move OK moved — the command was valid, and the player has moved in <direction> ERROR 502 <exception> — various exceptions, e.g. <direction> invalid Side Effects If the destination tile is: trap (!) → the player looses energy recharge ($) → the player gains energy; the recharge is removed from the map obstacle (&) → if the next tile in <direction> is open, the obstacle is pushed to that position, and the player occupies the previous position of the obstacle. It is expected that “bodies” of dead player will exhibit the same behaviour. flag (x or X) → if the flag is that of the opposing team, then the flag is captured, the player won, and the game terminates.
Chat	<game> Game finished! <name> from team <team> won the game. <game> <multiple lines with final scores of all players in the game>
Log	On victory, the following log tags are produced: VICTORY <team> <winner-simbol> <winner-name> SCORES symbol=<s> name=<n> team=<t> status=<k> score=<d>

Req	**<game> SHOOT <direction>** Fire a weapon (e.g., a laser) in the given <direction>. Deducts the distance travelled from the player’s energy budget. May have various effects, depending on where the shoot lands. Effect is computed based on the *current* state of the game, which may differ from the one obtained from a previous LOOK.

Res	**OK <cell>** or **ERROR <code> <text>** On an OK response, the map-code of the cell where the shoot landed is returned.This could be an empty cell, or a wall, or a player from the team indicated by <cell>, etc. Shooting can have side effects on the player, on other players, and on the map. Specific Responses OK <cell> ERROR 406 Cannot shoot — the cell right <direction> of the player is blocked. ERROR 502 <exception> — various exceptions, e.g. invalid <direction> ERROR 505 Need a direction in MOVE — command missing <direction>
Chat	<game> <player> shot <direction> <game> <player> hit <victim>

Req	**<game> STATUS** Returns the current status of the player (which will include multiple variables).
Res	**OK LONG** / **<status>** / **«ENDOFSTATUS»** or **ERROR <code> <text>** On an OK response, the current status of the game and of the players is returned. The status is provided on multiple lines following the “OK” line, with a suitable termination marker, to simplify debugging of the protocol. Each line is tagged with a 2-letters code specifyng both the format, and what the line refers to. The generic status line has the format: <tag>: <key>=<value> … <key>=<value> The following tags are currently defined: GA: game-related info; keys: name, state, size ME: info related the the player issuing STATUS; keys: symbol, name, team, loyalty, energy, score. PL: info related to all players in the game, keys: symbol, name, team, x, y, state

Social deduction

Two social deduction games are embedded in AmongAIs. The first one is about discovering impostors. The second one is about identifying humans and AIs.

Req	**<game> ACCUSE <player>** Accuse the <player> (from the same team) of being an impostor. The accused <player> is indicated by name (which is known to human players), not by map symbol (which may not be shown in a human-oriented map). The first accusation will start an emergency meeting, which lasts for 15 seconds. During the meeting, other players can also send their ACCUSE commands, which are tallied for the meeting. At the end of the period, all votes are tallied and if a simple majority is reached, on any player (not necessarily <player>), that player is expelled from the game. There might be at most one emergency meeting (per team) at any given time. Accusations may be changed while the meeting lasts (by re-issuing ACCUSE), but cannot be withdrawn.

Res	**OK** or **ERROR <code> <text>** On an OK response, the system has noted the accusation, and a team meeting is currently ongoing. <u>Specific Responses</u> OK Noted. ERROR 410 <player> not in game. ERROR 411 Accuse failed. — various protocol violations, e.g. <player> not from the same team, or client not currently able to issue accusations (e.g., dead in-game) etc. <u>Side effects</u> At the end of the emergency meeting, if a player has obtained a majority of votes, that player is killed. The scores of both the killed player and other team mates may be updated depending on how they voted and what the victim true loyalty was.
Chat	*On starting a meeting:* <game> EMERGENCY MEETING! Called by <name> *On each accusation (including the first):* <game> <accuser> accuses <player> *On ending a meeting:* <game> EMERGENCY MEETING ended. *If no player got a majority of votes:* <game> EMERGENCY MEETING miscarriage. No majority on impostor. *If a player got a majority of votes:* <game> EMERGENCY MEETING condamned <name>.
Log	*On starting a meeting:* IMPOSTOR Team <d>: call meeting *On ending a meeting where no player got a majority of votes:* IMPOSTOR Team <d>: no majority *On ending a meeting where a player got a majority of votes:* IMPOSTOR Team <d>: expelling <name>

Req	**<game> JUDGE <player> <nature>** Opines that the <player> (from any team) has the given <nature> (H for human, AI for AI). The <player> is indicated by name (which is known to human players), not by map symbol (which may not be shown in a human-oriented map). Judgements can be issued at any time, and will have no effect till the end of the game. At that point, the player is awarded points for correctly classifying other players. Judgements may be changed at any time during the game (by re-issuing JUDGE), but cannot be withdrawn.
Res	**OK** or **ERROR <code> <text>** On an OK response, the system has noted the judgement. <u>Specific Responses</u> OK Noted. ERROR 410 <player> not in game.

Finalization

A player can voluntarily leave a game. The player is removed from the team, and cannot act any longer, but can still look at the game.

Req	**<game> LEAVE <reason>** The player leaves the game. The <reason> may be reported in the game log. Further observation commands (e.g. LOOK, STATUS) will be honoured; action commands (e.g. MOVE; SHOOT, ACCUSE) will be rejected.
Res	**OK** or **ERROR <code> <text>** On an OK response, the player no longer is playing the game. Specific Responses OK Left game ERROR 502 <exception>
Chat	<game> <name> left the game.

The world and the map

The world is tiled, with each cell having a *terrain type* and possibly containing a *player* or an *object*. These are listed below; each terrain type, player or object is denoted by a single ASCII character for ease of reference.

The shape of the world

Our world is non-thoroidal, and entirely contained in a rectangle (think: an island). Games will happen on randomly-generated maps, but we may pre-define certain <game> names to correspond to fixed maps, for training and debugging purposes (i.e., ensure repeatability).
Currently the world is a square, with a side that is a power of two (e.g., from 64x64 to 256x256 cells), in order to facilitate using a CNN to “look” at the map. Future release cycles might use rectangular shapes, e.g. 256x128, with two teams initially “owning” one of the two 128x128 areas. All positions outside the “world” are void.

Elements on the map

Terrain types

.	**grass**, freely walkable, allow shooting.
#	**wall**, not walkable, stops shoots.
~	**river**, walkable, cannot shoot while on it, allow shooting through it.
@	**ocean**, not walkable, allow shooting through it.
!	**trap**, will subtract energy from player if walked on, allow shooting through it.
?	**void,** not shown on map, but implicitly frames the “world” (can be returned by the SHOOT command if a shoot ends on the border of the map). Not walkable, stops

	shoots.

Players

Players will always be on a walkable terrain type.

a-t A-T	**player** from team “lowercase” or team “uppercase”. Different players use different symbols in the map; the STATUS command provides a key.

Objects

Objects will always be on a walkable terrain type. A cell with an object will thus count as walkable in terms of players’ movements (but the underlying terrain type, which may be hidden in the map, still applies), except when specified otherwise. Objects do not stop shootings, except when specified otherwise.

x X	**flag** to capture for the corresponding team. The flag of a team is non-walkable for players of that team.
$	bonus energy **recharge**; if a player steps on a $, its energy will be recharged by a certain amount. The bonus energy recharge will disappear once used. Optionally, we may have $ respawn (possibly at random).
&	**barrier**: if a player pushes the barrier in a direction, and there is grass (with no object nor player) on the opposite side, the barrier will be moved into that cell. If the barrier is not moveable because the destination cell is not grass or occupied, then the barrier is not walkable either. Barriers stop shootings.