

User Guide for the Discord Computer Olympiad Interface

Quentin Cohen-Solal

quentin.cohen-solal@dauphine.psl.eu

November 16, 2025

This document explains how to use the three main features of the Discord Computer Olympiad Interface (DCOI), namely:

- running a match in manual mode: a human manually enters the actions of their program as text messages in the Discord conversation (Section 1).
- running a match in automatic mode: using a player Discord bot that writes the game actions provided by your program in the Discord conversation on your behalf (Section 2).
- creating your own game server: to run a local beta test for the Computer Olympiad or to organize your own competition (Section 3).

If you wish to use only the automatic mode, it is nevertheless necessary to read the following section. In addition to briefly describing how to play in manual mode, it explains the DCOI in order to make the following sections easier to understand — including how to select a game and start a match. This first section also covers the Free_game mode, which allows you to play (either in manual or automatic mode) any perfect-information game, by delegating to the players the verification of valid moves as well as the determination of the end of the game.

Remark 1. If you encounter any frustrations or bugs while reading this document or using the DCOI, please don't hesitate to let me know.

This document is a translation of “Guide d’utilisation du Discord Computer Olympiad Interface” (guide_utilisateur_DCOI.pdf). If you have any problems with the translation, please consult this document or the author of this document.

Contents

1 Description of the DCOI and use in manual mode	2
1.1 Overall Functioning	2
1.1.1 DCOI and Referee Bot	2
1.1.2 Focus on Player Bots	3
1.2 The Discord Application and Match Channels	3
1.2.1 Discord: Servers and Channels	3
1.2.2 Matches and Discussions	3
1.2.3 Warning About Editing or Deleting Messages	3
1.3 Starting a Match	6
1.3.1 Choosing the Game and Other Parameters	6
1.3.2 Special Case: Free_game Mode	6
1.3.3 Starting the Match	7
1.3.4 Progression and End of the Match	9
1.3.5 Non-Standard Match Endings: Timeout / Forfeit	9

1.3.6	Action Syntax (Text Format)	10
1.4	Match Logging	10
1.5	Warning About Matches That Are Too Short	11
2	Automated Play via Discord Bot	11
2.1	Installation of Required DCOI Packages	11
2.2	Creating Your Player Discord Bot Account	12
2.2.1	Creating the Bot Account and Inviting It to the Server	12
2.2.2	Retrieving Your Discord ID	14
2.2.3	Specifying the Configuration File	14
2.3	Programming a Discord Bot Using Python	16
2.4	Using the Generic Player Discord Bot via the GTP	17
2.4.1	Starting the GTP Bot	17
2.4.2	Specifying the GTP Commands	18
2.4.3	Displaying Information in the Terminal	19
2.4.4	Error Handling with GTP	19
2.5	Advanced Features	20
2.5.1	Crash Management	20
2.5.2	Pause mode	20
2.5.3	Bot Profiling	20
2.5.4	Graphical match viewing	20
3	Creating a Personal Discord Game Server	20
3.1	Creating a Discord Server	21
3.2	Creating the Referee “Bot Account”	22
3.3	Inviting Players to the Server	22
3.4	Inviting Bots to the Server	22
3.5	Starting the Referee Bot	22

1 Description of the DCOI and use in manual mode

To use the DCOI, you must be able to use Discord. Therefore, you need to install the Discord application or use the web version, and create an account if you haven’t already done so. (see <https://discord.com>). This section begins by presenting the overall functioning of the system (Section 1.1), then explains how to access the match channels (Section 1.2), followed by instructions on how to play a match (Section 1.3), including an explanation of the action syntax (Section 1.3.6). A few warnings are also described in Sections 1.2.2 , 1.2.3 and 1.5. Finally, we briefly discuss the match logging system (Section 1.4).

1.1 Overall Functioning

1.1.1 DCOI and Referee Bot

The DCOI is both a Python library and a collection of ready-to-use or extensible Python bots designed to create player bots and referee bots, enabling gameplay and match management within a Discord conversation.

Players and program operators join the competition’s Discord server. When match launch commands are entered in a Discord channel, the game begins. It is then managed by a referee bot (set up by the organizers). Players and player bots must post their game actions in the conversation when prompted by the referee bot. The referee bot checks the legality of moves, handles errors, tracks player time, and announces the end of the game (either normal or due to a time limit being exceeded) along with the winner.

1.1.2 Focus on Player Bots

All aspects related to Discord are automated through the Discord bots. Users only need to connect their program to the player Discord bot, which handles Discord events and posts messages in the conversation, among other things. To interface the two programs, you can either subclass a player bot and implement the plays method using DCOI's gameplay methods, or implement the Go Text Protocol (GTP) communication protocol. In both cases, you may need to convert the DCOI game action syntax into your program's own syntax. Finally, you must create a bot account and specify a configuration file to link the bot account with your bot program.

1.2 The Discord Application and Match Channels

1.2.1 Discord: Servers and Channels

We will now briefly review the different elements of the Discord application that are needed to run a match.

A screenshot of the application is shown in Figure 1. The first step is to join the competition server (servers on Discord are called guilds). On your first access, click the “+” icon at the bottom of the server list (see Figure 2). During the competition or the beta test, click “Join a Server”, then enter the invitation link provided by the organizers. For the beta test, you must join the beta test server: <https://discord.gg/npCBBJGe>. Once the server has been added to your account, you can return to it at any time by clicking its icon, now visible in your server list (see Figure 2). After joining the server, you need to enter a match channel by clicking on its name. The available channels are displayed in the designated area (see Figure 3). You may also join the general or discussion channel to talk with other participants, or any other channel — for example, to ask for help. As in any instant messaging platform, to write in the selected channel, simply type in the text input area provided for this purpose (see Figure 4). Once you press Enter, your message will be sent and will appear in the area showing the conversation history (the middle of the window — see Figure 5, if needed).

1.2.2 Matches and Discussions

Note that you can also talk in the match channels, as long as your message does not consist solely of a text-formatted action. Indeed, if the match has started and you are one of the players, your message will be interpreted as a game move.

For example, never write just “A1” or “A2-B3” during a conversation. Instead, you may write something like “Why did you play A1?”, or even just “A1.” or “A1?”.

In general, try to avoid chatting in the match channel as much as possible, at least during ongoing matches, to prevent cluttering the conversation.

1.2.3 Warning About Editing or Deleting Messages

Discord allows you to edit and delete your messages. During a match—whether you or your bot are playing—never use this feature under any circumstances. There is a small but non-negligible chance that doing so could cause system instability, preventing the match from being completed.

If such an event occurs, you will be held responsible and will automatically lose the match.

If you accidentally edit your move, immediately edit it again to restore the original action — this should resolve the issue without consequence.

If you accidentally delete your move, unfortunately, this action is irreversible, and you must hope it does not affect the rest of the game.

Note that the system's default behavior is to ignore message deletions and edits, so there is absolutely no benefit in attempting such actions.

In case of any problem, contact an organizer.

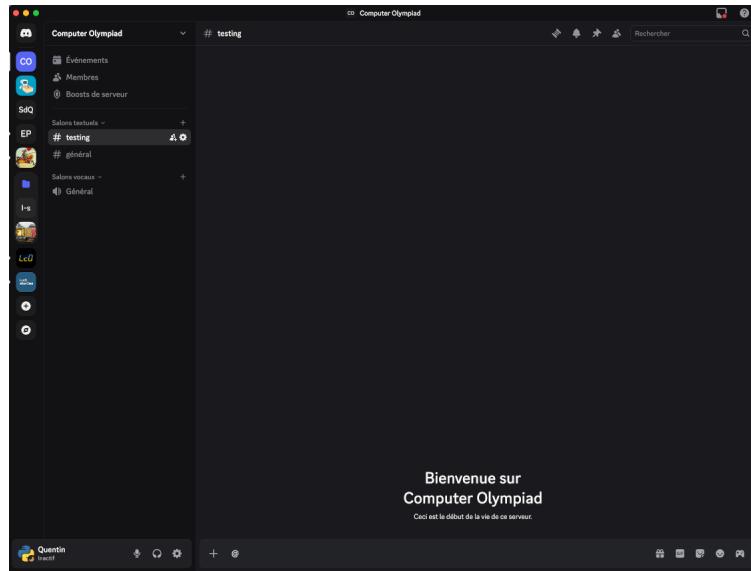


Figure 1: Discord App

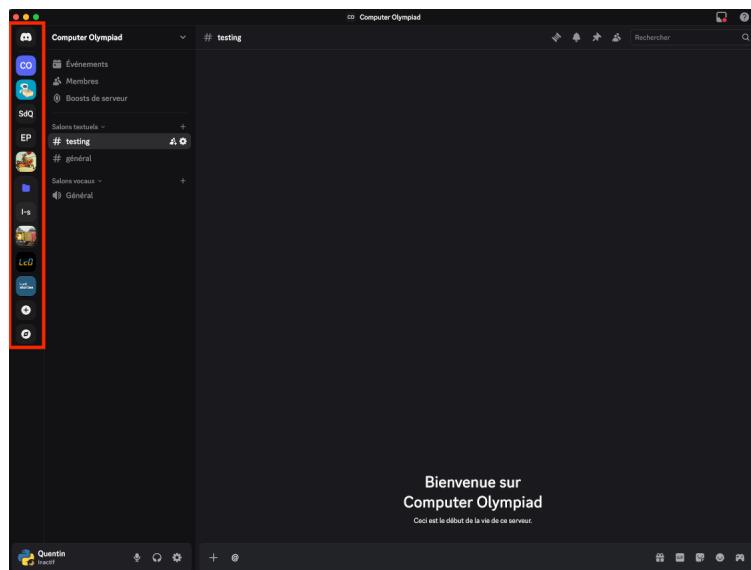


Figure 2: Discord servers list (red box)

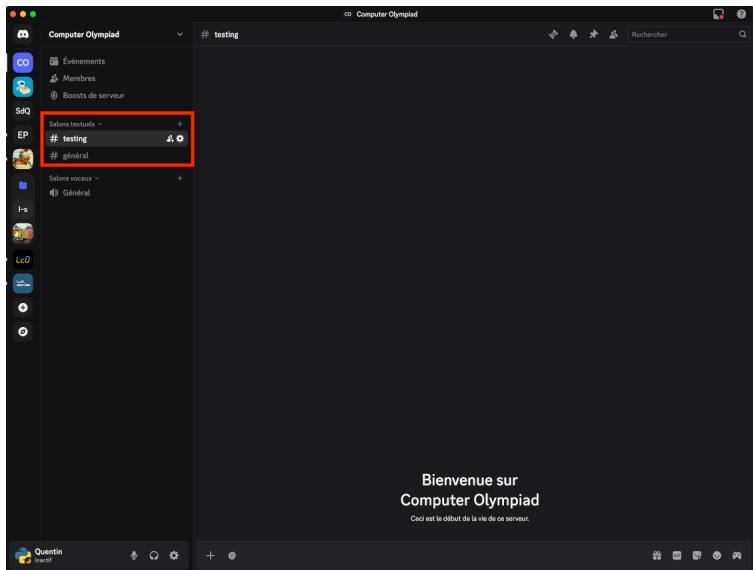


Figure 3: Discord channel list (red box)

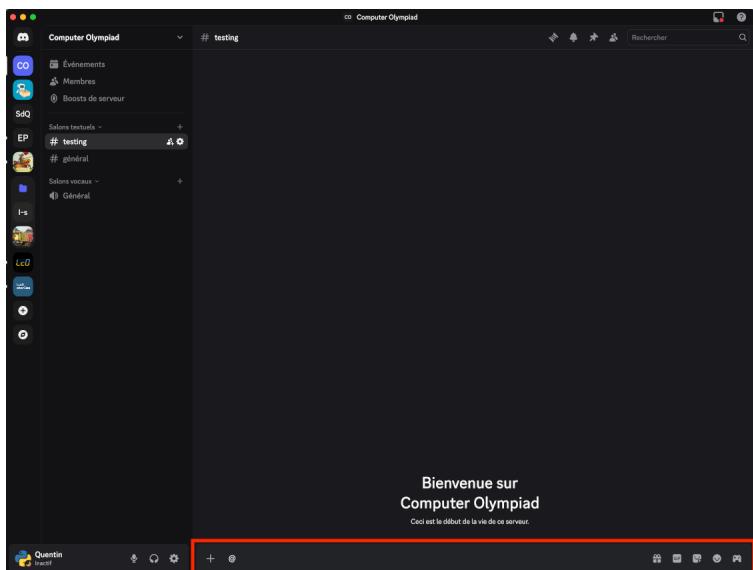


Figure 4: Discord input area for conversation and match playing (red box)

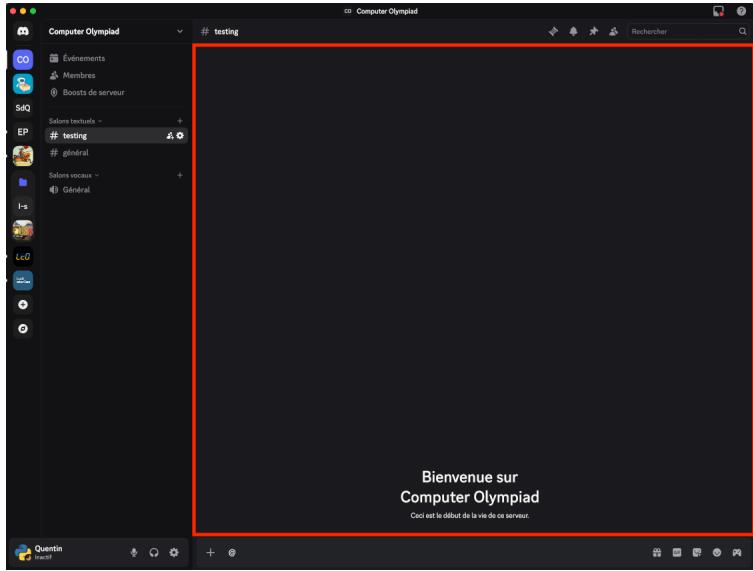


Figure 5: Discord conversation / match history (red box)

1.3 Starting a Match

We now move on to the core of the topic and explain what you need to know — and how to proceed — to start a match.

1.3.1 Choosing the Game and Other Parameters

To choose the game for the match, type the following command in the conversation:

`“!set game <GAME_NAME>”.`

Example 2. “!set game Clobber”.

Remark 3. If the command worked, the referee bot will respond in the conversation: “Game set to `<GAME_NAME>`”.

To see the list of available games, you can type the following command in the conversation:

`“!available_games”.`

If you need to adjust the total time allocated to each player per match (default is 30 minutes per player per match), you can use the command:

`“!time <T>”`

where `<T>` is an integer indicating the time in seconds, or an integer followed by one of the units “s”, “min”, or “h” to specify the unit (for example, “!time 20min” ou “!time 1h”).

Remark 4. If you plan to play a series of matches on the same game, it is not necessary to set the game before each launch.

1.3.2 Special Case: Free_game Mode

The DCOI provides the ability to play any perfect-information, deterministic game using the “pseudo-game” called `Free_game`. Note, however, that — while this is not a practical limitation — players

must take turns alternately (i.e., never play twice in a row). This mode allows matches to be played for such games but with the following trade-off: no legality checking of moves and no automatic winner determination. Players must therefore take turns entering the action “end” to signal the end of the match. This feature should only be used if participants are certain they are following the same rules and each have their own game engine. To use Free_game, type:

```
“!set game Free_game”.
```

The valid actions for this mode are all actions that follow the DCOI action syntax (for example, “A1”, “B2-C3”, “D4-E17-H23”; see Section 1.3.6) plus the action “end”. Yet, you can also add custom special actions to Free_game with the command:

```
“!add_freegame_moves”.
```

You can list all additional special actions of Free_game with the command:

```
“!show_extra_freegame_moves”.
```

And you can remove all special actions from Free_game with the command:

```
“!clear_freegame_moves”.
```

These commands must, of course, be used before the match begins.

1.3.3 Starting the Match

We now turn to the practical process of running a match. The first step is to ask the referee bot to propose a game. To do this, use the command:

```
“!start @Player1 @Player2”.
```

Remark 5. @Player1 can be “@Name” (for example “@Alice” or “@Bot1” or more succinctly “Alice” or “Bot1”). @Player1 can also be “@<member_ID>”, for example “@<165226745874577>”. When you type @, Discord displays a list of available user IDs. Use the up/down arrow keys to select the ID of the player in the match and press Enter. If it is not in the list, you can start typing its name until it appears.

Remark 6. You can save time by using the alias “!s” instead of “!start”.

Remark 7. Run “!start” without specifying the players to start a match with the previous players, but in reversed playing order.

If one of the players is using a bot, you must specify the bot’s ID, not the operator’s. Moreover, if one of the players is a bot, that bot must already be running before the !start command is sent.

Once the command is executed, the referee displays the details of the proposed match (see Figure 6). Participants must then click the thumbs-up icon under the message to confirm the match. If you are using a bot, this step is handled automatically by the library — you don’t need to do anything. In the current configuration, players have 15 minutes to accept the match proposal. When all players have clicked the thumbs-up, the match begins. If you wish to decline the proposed match, click the thumbs-down icon instead.

Remark 8. It is normal to see the number 1 next to the thumbs-up and thumbs-down icons. These reactions were added by the referee bot to make them visible and to make it easier for players to react.

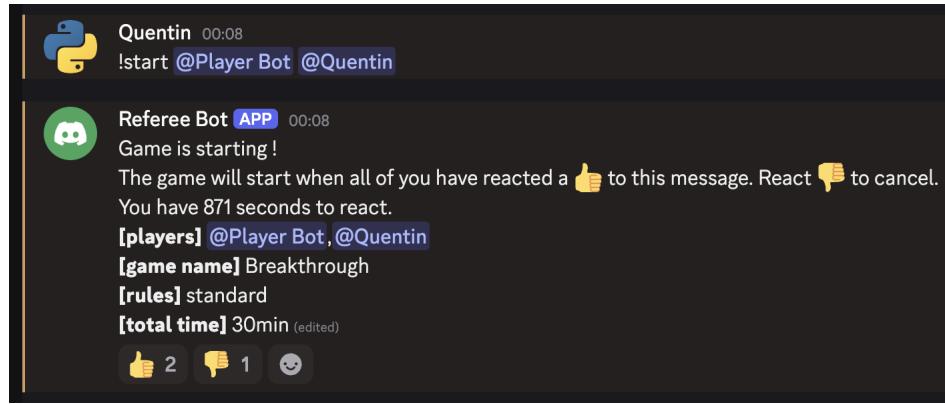


Figure 6: Example of the beginning of a match (example of waiting for player confirmation to start the game)

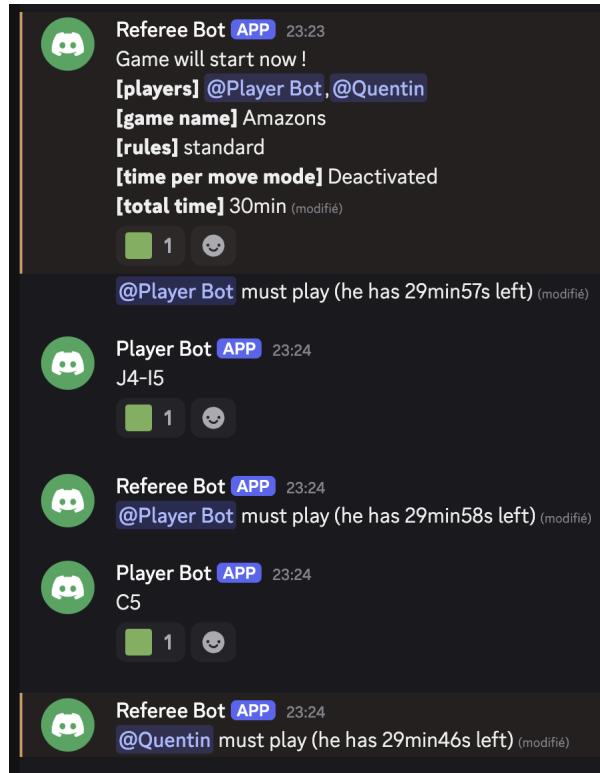


Figure 7: Example of the beginning of a match

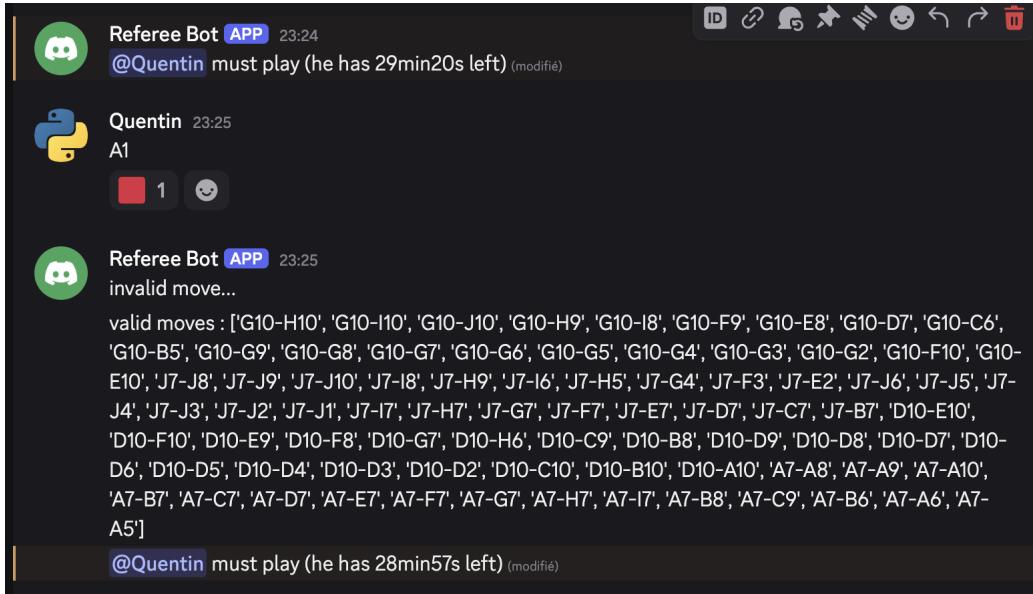


Figure 8: Example of invalid move

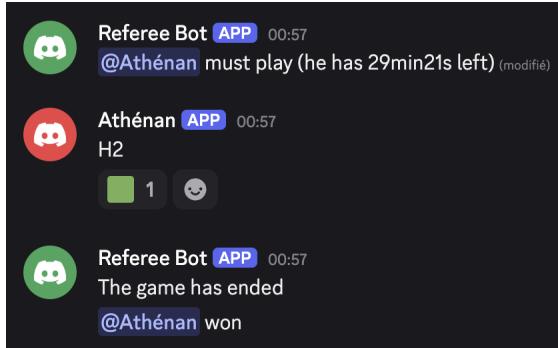


Figure 9: Example of endgame

1.3.4 Progression and End of the Match

During a match, the referee indicates when it is a player's turn to move. It writes a message in the conversation such as: “@<PLAYER> must play (he has <TIME> left)”. At that moment, the mentioned player may enter in the conversation the action chosen by their program. A player must not post any move before being explicitly told to play. Figure 7 shows the beginning of a match corresponding to the following sequence of actions: “J4-I5” followed by “C5” (the first player plays twice in a row).

If the action entered is incorrect, it is marked with a red square, and the referee indicates the error and provides the list of legal actions (see Figure 8).

When the match is over, the referee announces it and indicates the winner (see Figure 9).

Once the match is finished, you can start a new game using the “!start” command.

1.3.5 Non-Standard Match Endings: Timeout / Forfeit

We now turn to special match endings, triggered either by a player exceeding their thinking time (default: 30 minutes per player per game) or by a player's decision.

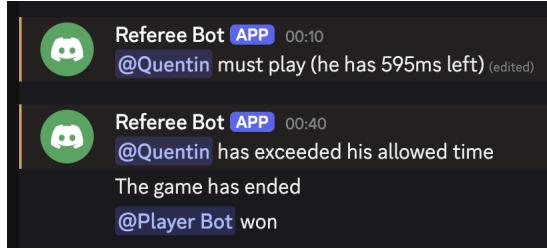


Figure 10: Example of timeout

If a player exceeds their allotted thinking time, the match ends and that player loses the game (see Figure 10).

If you need to stop an ongoing match, you can use the “!stop” command. This will trigger a vote. Note that this command does not currently work if you are using a Discord bot.

If you wish to forfeit a match, when it is your turn to play, you can enter the action resign (type “resign”). This will immediately end the match and award the victory to your opponent (without triggering a vote).

If you are using a bot, use the “!resign” command instead. When it is your bot’s turn, it will execute the resign action, which will similarly end the match.

1.3.6 Action Syntax (Text Format)

With the DCOI, it is important to distinguish between action syntax and legal actions. In order for the DCOI to differentiate a normal conversation from a game move, it requires a specific syntax. This syntax includes moves like “A1”, “B2-C12”, and also longer sequences such as “D5-E3-G15”, possibly combined with specific keywords, e.g., “queen-A8”. Note that “resign” is an example of a keyword common to all games. To see the list of keywords for the current game, use the command:

```
!show_move_keywords.
```

Note that the general syntax for game actions is defined by the following regular expression:

$$([a-zA-Z][0-9]\{1,2\} \mid [0-9]\{1,2\} \mid \text{keyword1} \mid \dots \mid \text{keyword_n})(-[a-zA-Z][0-9]\{1,2\} \mid [0-9]\{1,2\} \mid \text{keyword1} \mid \dots \mid \text{keyword_n})^*$$

In other words, a text-formatted action is a sequence of blocks of length one or more, separated by hyphens, where each block is either a coordinate (e.g., A12, B5, C18), a number or a keyword.

When a message is posted in the conversation, the referee bot checks whether the text follows the expected syntax. If it does, and if it is that player’s turn, the bot will attempt to play the action. If the action is valid, it is executed; otherwise, the bot reports that the action is illegal, as described previously. If the syntax is not respected, the referee simply ignores the message.

To find the coordinates associated with the different games, consult the document “game_move_coordinates.pdf”.

1.4 Match Logging

Matches are saved in .json files (in the directory discord_interface/log/bot_play_log/), using the following format:

$$\text{“play_}<\text{PLAYER_ID}>\text{_Computer-}\\ \text{Olympiad_}<\text{CHANNEL_NAME}>\text{_}<\text{DATE}>\text{_}<\text{TIME}>\text{.json”}.$$

This file contains a human-readable Python dictionary with many attributes, in particular:

- game_name – the name of the game,

- moves – the history of moves in the match,
- players – a list of PLAYER_IDs,
- winner – True if you or your bot won the match.

1.5 Warning About Matches That Are Too Short

Note that there is a delay in action communication due to internet latency (which depends on the quality of your connection), as well as Discord’s latency, and to a lesser extent, the DCOI’s own processing time. This delay is measured in seconds, typically around one second on average. The DCOI is therefore not suitable for matches with an average time of less than 1 second per action. Moreover, sending more than one message per second by a bot or user violates Discord’s terms of service, which interprets such behavior as flooding. This can lead to various penalties, including bans, shadow bans, or slowed Discord functions. However, in practice, sending actions at intervals of about one second has never caused issues during our extensive testing.

2 Automated Play via Discord Bot

In this section, we explain the different ways to conduct a match with the DCOI in automatic mode.

In all cases, automated play is carried out using a Discord player bot. This requires creating a Discord bot account on the Discord Developer Portal (see Section 2.2). Next, the native way to run matches in automatic mode is to extend one of the DCOI Python Player classes (Section 2.3). The other method, described in Section 2.4, is to use a fully implemented bot that communicates with your program via input/output (inter-process communication) using the Go Text Protocol (GTP). This second approach has the advantage of not requiring any programming with the DCOI library — nor any knowledge of the Python language. Finally, in Section 2.5, we introduce advanced features related to bots.

We begin by detailing the installation of the Python packages required by the DCOI.

2.1 Installation of Required DCOI Packages

We will now detail the installation of DCOI. Note that Python 3.10+ is required to install and use DCOI.

Remark 9. If you are using Python for your projects, to avoid breaking dependencies, it is recommended to use a Python virtual environment.

To install the DCOI packages, you have two possible methods:

Quick method: Execute the following commands in your terminal:

- “cd discord_interface”
- “chmod +x install.sh”
- “bash install.sh”
- (optional) Test the installation: run in the terminal: “python3 test_installation.py”.

Remark 10. If you move the discord_interface folder to another location, you will need to run “bash install.sh” again.

Step-by-step method: The DCOI is based on the official Python library for Discord. To use the DCOI library, you need to install it by running the following command in your terminal: “python3 -m pip install -U discord.py”. You also need the numpy library to run the games: “python3 -m pip install -U numpy”. If you wish to use GTP, you will additionally need the pexpect library to enable inter-process communication. Install it with: “pip install pexpect”. You can then test your installation by running in the terminal: “python3 test_installation.py”.

Finally, you need to modify the Python path to include the parent folder of “discord_interface”:

```
UPPER_DIR_DCOI='<path_to_discord_interface>'  
echo 'export PYTHONPATH=$PYTHONPATH:$UPPER_DIR_DCOI' >> ~/.bashrc  
source ~/.bashrc
```

Remark 11. If you move the discord_interface folder, you will need to update the PYTHON PATH with the new path.

2.2 Creating Your Player Discord Bot Account

We now turn to creating your bot account, which allows the bot program to connect to the Discord application. More specifically, this series of steps lets you retrieve various pieces of information. Some of this information must be entered into the parameters.conf configuration file located at the root of the discord_interface folder. Creating the account also provides a link that you must give to the organizer so they add your bot to the competition’s Discord server.

2.2.1 Creating the Bot Account and Inviting It to the Server

1. Go to the Discord Developer Portal website: <https://discord.com/developers/applications>. Log in with your Discord account. Then click “New Application” at the top right of the screen (see Figure 11 if needed) and give your bot a name. You will then arrive at the bot’s page.
 - (a) Use the name of your program to name the bot.
 - (b) If you encounter difficulties, try using a different web browser.
2. Next, enable the “Message Content Intent” option in the Bot tab (see Figure 12, if needed).
3. Then, retrieve the TOKEN of the bot — this is essentially its password, so it must be kept secret. Note it down, as we will need it to configure the bot in the next section.
 - (a) Go to the Bot tab (still on the bot page). In the TOKEN section, click the “Reset Token” button (see Figure 13 if needed).
4. Go to the OAuth2 tab, then “OAuth2 URL Generator”. Under the SCOPES section, check the “bot” option.
5. In the bot permissions subsection, check the following options:
 - (a) “View Channels”,
 - (b) “Send Messages”,
 - (c) “Read Message History”,
 - (d) “Add Reactions”.
6. Finally, at the bottom of the page, in the “Generated URL” field, copy the URL. Send this URL to the organizer so they can add your bot to the competition’s Discord server.

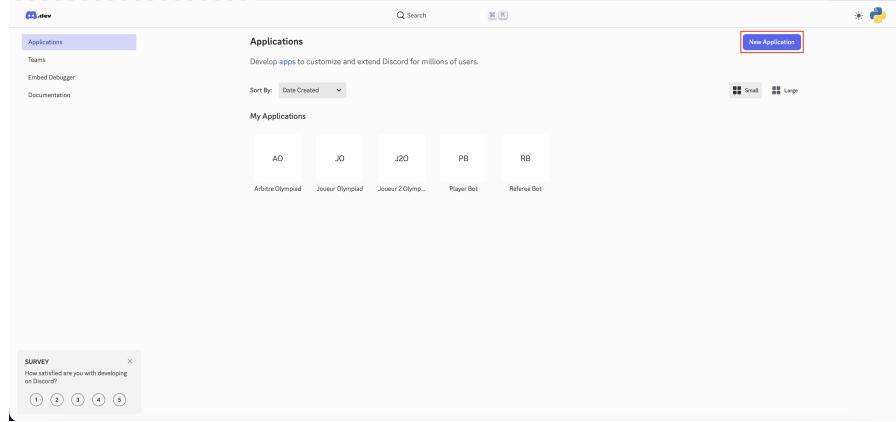


Figure 11: New application button (red box)

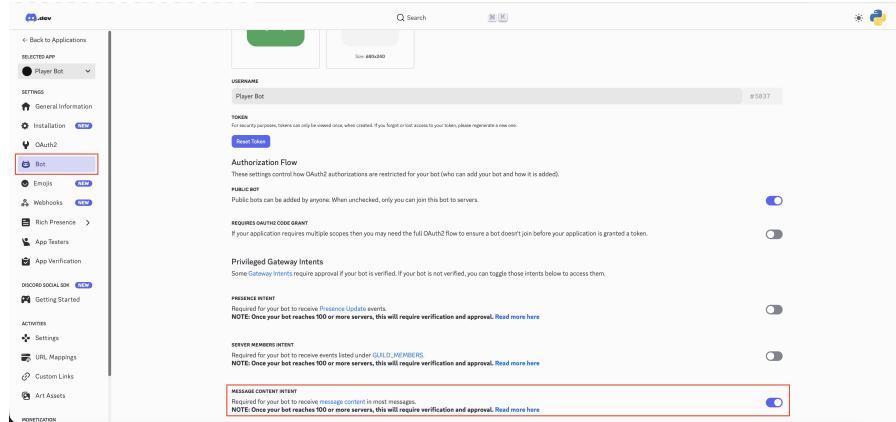


Figure 12: Bot window and message content intent option to be activated (red boxes)

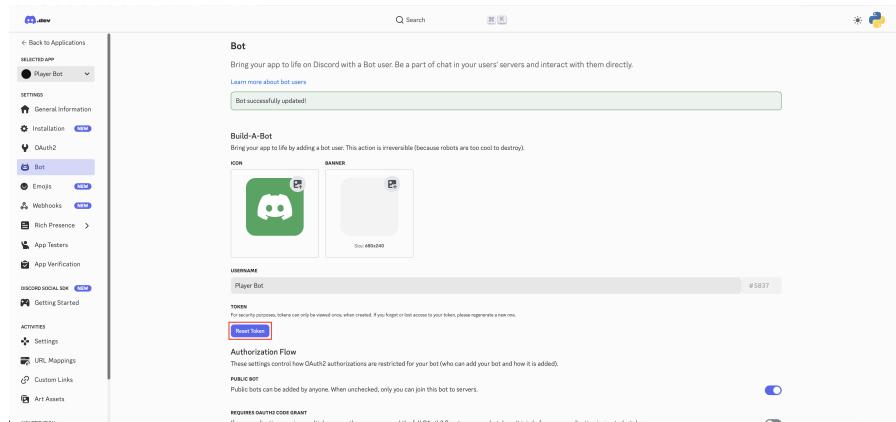


Figure 13: Reset Token button (red box)

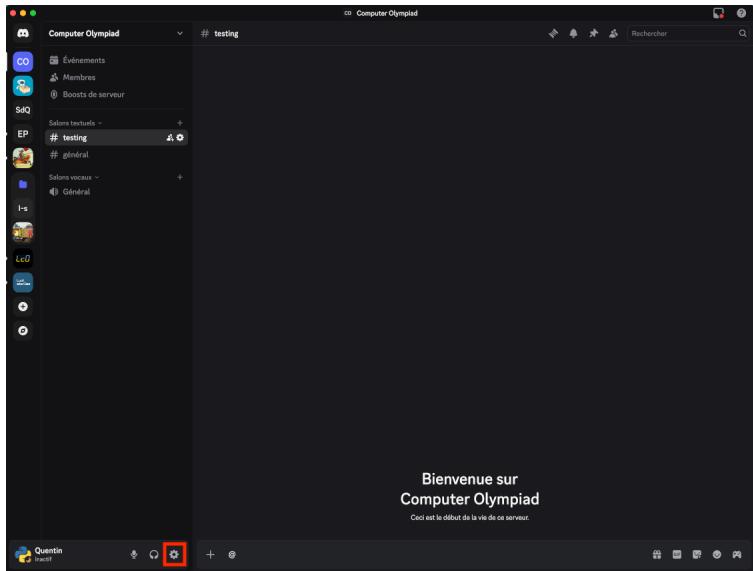


Figure 14: Discord parameters button (red box)

2.2.2 Retrieving Your Discord ID

Next, you must retrieve your Discord user ID to specify it in the configuration file. It allows you to use commands that control your bot to access certain features. For example, it is required to resume a match after a crash (see Section 2.5). In this section, we explain how to obtain your Discord ID.

The first step is to enable Developer Mode on your Discord account. To do this:

1. Open the Discord application.
2. Go to Settings (click the “gear” icon; see Figure 14, if needed).
3. Click on the “Advanced” tab.
4. Enable the “Developer Mode” option.
5. Click on your username at the bottom right.
6. In the small pop-up window, click “Copy User ID” (see Figure 16, if needed).
7. Return to Settings and disable the Developer Mode option.

2.2.3 Specifying the Configuration File

We now explain how to configure the parameters.conf file, located at the root of the discord_interface folder, which allows the bot program to access Discord.

1. Set the value of PLAYER_BOT_DISCORD_TOKEN to the TOKEN of your bot account (retrieved in Section 2.2.1).
2. Optional but recommended: set the value of OWNER_ID to your Discord user ID (retrieved in Section 2.2.2).

To verify that the configuration file is correctly set, launch your bot and check:

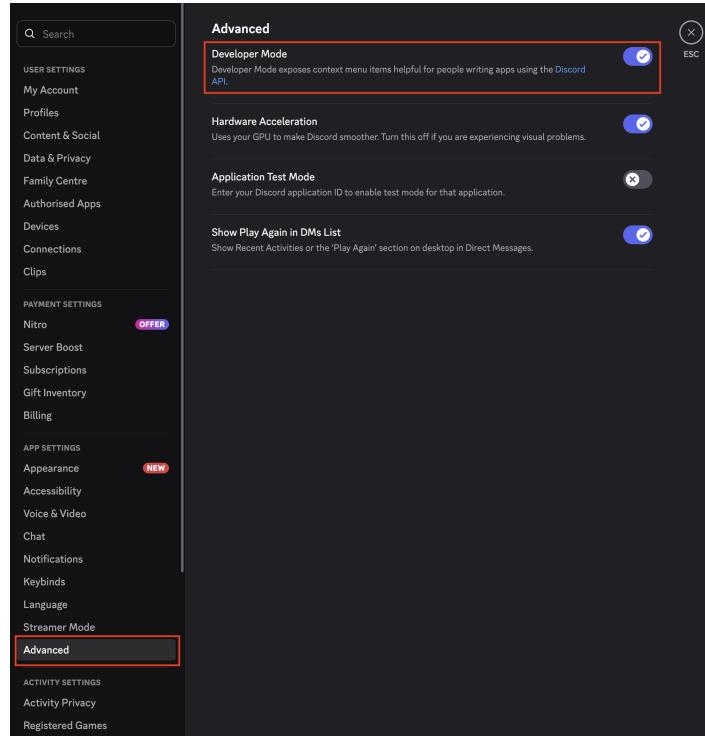


Figure 15: Discord advanced window and developer mode option (red boxes)

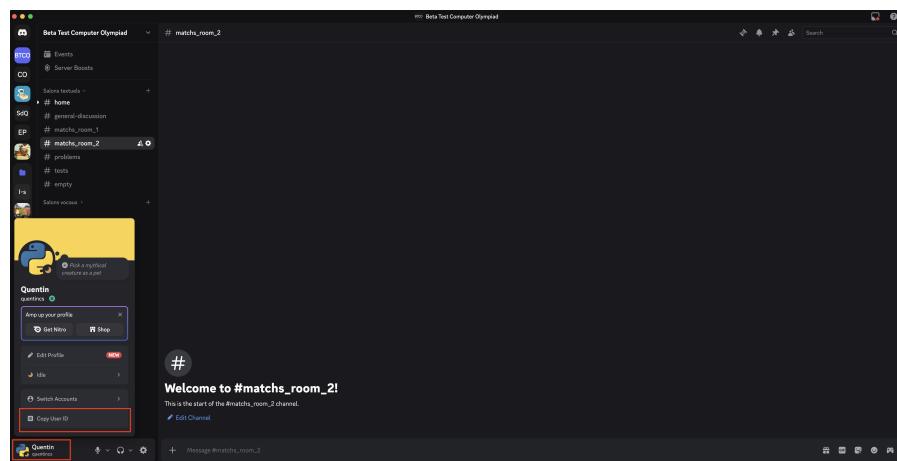


Figure 16: Open owner window and copy Owner ID (red boxes)

game methods of any player	
self.game.get_numpy_board()	return a numpy array representing the game board
self.game.get_current_player()	returns i if the current player is the player i ($i \in \mathbb{N}$)
self.game.textual_legal_moves()	list of valid actions in text format (e.g. “A1” or “A2-B3”)
self.game.ended()	return True if the game is ended
self.game.winner	return the index i of the winner player ($i \in \mathbb{N}$)
self.game.textual_plays(move)	apply the move in the current game state (i.e. self.game)
self.game.undo()	undo the last action of the current game state (i.e. self.game)

Table 1: Elementary methods of any Player Python object for programming an AI player.

```
cd <discord_interface_path>/discord_interface
python3 main_random_ai.py
```

If the terminal shows the error “Bad OWNER_ID”, you need to correct the OWNER_ID variable in “parameters.conf”.

For a more advanced test, launch your bot and type in a channel on the competition’s Discord server the following command:

“!conf_test”.

- If nothing happens, contact me.
- If everything works correctly, “All is OK!” will appear in the terminal.
- If an error occurs in the terminal (discord.errors.LoginFailure: Improper token has been passed), the PLAYER_BOT_DISCORD_TOKEN has been incorrectly specified.

2.3 Programming a Discord Bot Using Python

We now explain how to program a Discord bot in Python. You need to inherit one of the Python Player classes provided by the DCOI library. Here, we describe the simplest way to do this.

Bot programming To program your bot, you need to inherit the BasicPlayer class from the file: “discord_interface/player/model/basic_player.py”. The only thing you need to do is define the method: my_plays(self, game_history, time_left=inf, opponent_time_left=inf). This method should return the action that your program wants to play in the current game state. The current state can be reconstructed from the game_history parameter, which contains the list of moves in text format played since the start of the match. You can also use the native game methods provided by the DCOI, which allows you to avoid implementing the game logic yourself. In this second case, you will need the methods described in Table 1.

Pay attention to the time_left parameter: it indicates the remaining time your program has for the rest of the match. If this value reaches 0, you will lose the game.

An example of inheritance is provided in the TextualRandomAI class in the file:

“discord_interface/player/instances/textual_random_ai.py”.

This bot plays randomly, except when there is a winning move among the available actions. In that case, it plays the winning move.

Launching the bot Once your bot account is created, the configuration file is set up, and your Player class is inherited, all that remains is to launch your bot in a main script. To do this, use the method: `bot_starting(AI_Class)` from the file: `discord_interface/player/model/bot_launcher.py` with `AI_Class` set to your AI Python Player class. An example of a bot launcher program is provided in the file:

```
“discord_interface/main_random_ai.py”.
```

All that remains for you to do is: “`python3 <your_main>.py`” in the folder “`discord_interface`”. For example:

```
cd <discord_interface_path>/discord_interface
python3 main_textual_random_ai.py
```

The bot now waits for the command: `!s @<Bot_name_1> @<Bot_name_2>` to be executed and for the referee bot to display the match information. At that point, the bot performs its initialization and adds its “thumbs up.” The match then begins, and the bot plays automatically whenever it is its turn, until the end of the game.

Remark 12. If you used a Python virtual environment during installation, remember to activate it before launching the bot.

Remark 13. If your bot plays twice on its turn instead of once, it is very likely because two processes of the same bot program are running simultaneously. Stop one of them. If you cannot locate the other process, restart your computer.

Advanced Features Finally, optionally, if you need advanced features, such as code to execute at the initialization or end of the match, you can inherit from the class `AdvancedBasicPlayer` (file: `discord_interface/player/model/advanced_basic_player.py`) and define the methods: `my_initialisation(game_name)` and `my_end()`.

2.4 Using the Generic Player Discord Bot via the GTP

We now explain how to run automatic matches using GTP. Information about GTP is available, for example, on this website: <https://www.lysator.liu.se/~gunnar/gtp/gtp2-spec-draft2/gtp2-spec.html>. In Section 2.4.2, we will show how to specify a GTP program. Here, we begin by describing how to launch the GTP bot, assuming you already have a program specified in the GTP format. Finally, we will cover an advanced feature that allows you to display information from your program in the terminal, (Section 2.4.3) as well as how to handle errors in your program when using GTP (Section 2.4.4).

2.4.1 Starting the GTP Bot

We begin by explaining the prerequisites for launching your GTP bot. First, you need to connect your program to the GTP bot. To do this, you must specify three parameters:

- “`program_name`”: the name of your player program,
- “`program_arguments`”: the list of arguments for your program (separated by spaces),
- “`program_directory`”: the absolute path to your program (or the path relative to the `discord_interface` folder; for example, you can use an empty string if the program is located at the root of the `discord_interface` folder).

These variables must be specified in the `parameters.conf` file at the root of the `discord_interface` folder.

To run your program, all you need to do is:

GTP command	return	effect
clear_board	=	restart the game
undo	=	cancel the last move
play <player> <move>	=	plays <move> for <player>
time_left <player> <time>	=	inform the time left in seconds
genmove <player>	=<move>	return the <move> to play and play it
quit	=	shutdown the program
game <game>	=	initialization for <game>

Table 2: List of GTP commands to implement, the syntax of their return and their expected effect (<move> is an action in text format (see the syntax in Section 1.3.6) ; <player> is an integer $n \in \mathbb{N}$; <game> is the game name of the match). Remark: commands “game” is not part of standard GTP ; it is optional (make them only return “=”): “game” is required only for multi-games participation. Remark: each return starts with “=”.

“python3 main_gtp_ai_from_conf.py”

in the terminal from the discord_interface folder.

Remark 14. If you are comfortable with Python, you can also modify and run the main_gtp_ai.py file to avoid editing the configuration file.

2.4.2 Specifying the GTP Commands

In order to use the GTP-based bot, your program must read GTP commands from standard input (which will be sent by the GTP bot) and write the results of these commands to standard output (which will then be read and processed by the GTP bot).

Remark 15. For this reason, your program must **not** use standard output (stdout/stderr) except to respond to a GTP command. If you need to write something that is not a GTP response, write it to a file (for example) instead.

The required GTP commands to run the GTP bot, which need to be implemented in your program, are described in Table 2. Note that each command must return ‘=’, optionally followed by a string. To verify that your GTP command implementation is correct, you can run the following test program in the terminal: “python3 test_gtp_ai_from_conf.py”. In case of any issue, open the file “<program_name>_gtp.log” to view the history of the GTP communication.

If needed, an example of a Python implementation of GTP commands is available in the file: “discord_interface/player/instances/autres/gtp_random_ai.py”.

Remark 16. Example of GTP communication between the bot and the player program:

```
genmove 0
=A2-A3
play 1 A7-A6
=
genmove 0
=A3-A4
...
```

Remark 17. If you want your bot to be able to resign automatically, it simply needs to return “=resign” in response to the genmove command at the appropriate time.

Remark 18. If you want to resign your bot manually, use the already functional command !resign.

Remark 19. If your bot plays twice on its turn instead of once, it is very likely because two processes of the same bot program are running simultaneously. Stop one of them. If you cannot locate the other process, restart your computer.

Remark 20. If you wish to use the Go-style nomenclature for GTP — i.e., “black” or “b” instead of 0 for the first player, “white” or “w” instead of 1 for the second player, and actions written in lowercase without hyphens (for example, a2b3 instead of A2-B3) — you can use:

- “python3 main_go_gtp_ai_from_conf.py” instead of “python3 main_gtp_ai_from_conf.py”,
- “python3 test_go_gtp_ai_from_conf.py” instead of “python3 test_gtp_ai_from_conf.py”.

2.4.3 Displaying Information in the Terminal

Finally, we briefly discuss an option that allows you to display information from your program in the terminal when using the GTP bot.

Setting up this feature is very simple. For each GTP command return, simply concatenate the return with the # symbol followed by the line you want to display in the terminal.

Example 21. If you want to display in the terminal: “The move was correctly cancelled” when the undo command is executed, the return of undo should be: “=#The move was correctly cancelled”.

Note that the line to display in the terminal must not contain line breaks. However, you can create line breaks by adding multiple # symbols. Each additional # will be interpreted as a line break.

Example 22. If the return of the genmove command is: “=<move>#Calculating the next move completed#Move played: <move>#Calculation time 15 seconds”, this will display the following in the terminal:

```
Calculating the next move completed
Move played: <move>
Calculation time 15 seconds
```

2.4.4 Error Handling with GTP

We now present the handling of errors related to GTP.

Symbol ? If you want error handling with GTP, meaning that your program notifies the Discord bot when an error occurs, you must use the standard GTP syntax designed for this purpose. Specifically, when an error occurs and needs to be communicated, write to standard output but modify the output format by replacing “=<output_line>” with “?<output_line>”. The presence of the “?” allows the Discord bot to trigger an error, which will be displayed in the terminal. Additional information will also appear in the terminal, including:

- The name of the command that caused the “?”,
- The output of your program,
- Any comments displayed using the feature described in the previous section.

The “!gtp_replay” command We will now turn our attention to the following incident: your bot program has stopped playing but is still running.

If the issue comes from your program (the one controlled using GTP commands), the following protocol should allow you to solve the problem. Use the “!gtp_replay” command in the Discord channel. This restarts your program and replays all GTP commands from the beginning, then continues the current Discord match.

If that doesn't solve the problem, your program probably needs to be modified. To understand what's happening, look at the GTP communication history; it can be found in the following file: "<your_program_name>_gtp.log". Once your program is corrected, run the "!gtp_replay" command again.

2.5 Advanced Features

We now describe several advanced features of player bots.

2.5.1 Crash Management

If for any reason your bot crashes and needs to be restarted, to resume the ongoing match, after relaunching your bot in the terminal, you must use in the Discord match channel the command:

"!continue".

Its alias is "!c".

Before the competition, I suggest forcing a bot kill during a match, then relaunching it and verifying that the !continue command works correctly. This ensures you can resume a match in case of a problem.

If "!continue" doesn't work, you can enter your program's moves manually and try "!continue" again later during the match, or finish the match manually.

If the referee bot does not respond, contact a competition official to restart the referee and also execute the !continue command.

2.5.2 Pause mode

If you want your bot to temporarily stop playing (stop writing actions in the conversation), use "!freeze".

To make it resume the match, use "!unfreeze".

2.5.3 Bot Profiling

Profiling can be performed using the command: "!p".

Remark 23. If you are using a GTP bot, some of the profiling values provided may be incorrect if the GTP "player" command has not been implemented or has been implemented incorrectly.

2.5.4 Graphical match viewing

To view the current match with a basic graphical interface, run in the terminal: "python games/game_viewing/player_game" (this does not work for all games).

3 Creating a Personal Discord Game Server

In this section, we explain how to create your own personal game server, either for a local beta test of the Computer Olympiad or to organize your own competition. Several steps must be completed:

1. Creating a Discord server (Section 3.1),
2. Creating the "bot account" for the referee bot (Section 3.2),
3. Inviting players to the server (Section 3.3),
4. Inviting bots to the server (Section 3.4),
5. Launching the referee bot (Section 3.5).

Note that there can be only one referee bot per channel, and you must specify the corresponding channel for each referee bot. This will be addressed in the following section. Therefore, you need multiple referee bots if you want to run multiple matches in parallel.

3.1 Creating a Discord Server

We begin by detailing how to create a Discord server:

1. In the vertical server bar on the left of the Discord application:
 - (a) Click the “+”, then “Create My Own” (choose your options and server name).
2. Enable “Developer Mode” (steps 1 to 4 in Section 2.2.2).
3. Left-click the icon of the server you just created in the server bar.
4. Click “Copy Server ID”.
5. For a Computer Olympiad beta test:
 - (a) In the file parameters.conf:
 - i. copy the line starting with:
 - A. “BETA_TEST_COMPUTER_OLYMPIAD_GUILD_ID=”.
 - ii. Comment out the original line with #.
 - iii. Replace the value of BETA_TEST_COMPUTER_OLYMPIAD_GUILD_ID with the ID of the created server.
 - iv. Ensure that in parameters.conf, BETA_TEST_MODE=True.
 - (b) For each referee bot (index i):
 - i. In the server, create a channel named “match_ i ”.
 - ii. Right-click the channel and select Copy ID.
 - iii. In parameters.conf
 - A. paste it as the value of REFEREE_BOT_ i _DISCORD_CHANNEL.
6. For a server used for other purposes:
 - (a) In parameters.conf:
 - i. Duplicate the line starting with:
 - A. “COMPUTER_OLYMPIAD_GUILD_ID=”.
 - ii. Comment out the original line with #.
 - iii. Replace the value of “COMPUTER_OLYMPIAD_GUILD_ID” with the ID of the created server.
 - iv. Ensure that in parameters.conf, BETA_TEST_MODE=False.
 - (b) For each referee bot (index i):
 - i. In the server, create a channel named “match_ i ”.
 - ii. Right-click the channel and select Copy ID.
 - iii. In parameters.conf
 - A. paste it as the value of BETA_TEST_REFEREE_BOT_ i _DISCORD_CHANNEL
7. Deactivate “Developer Mode”

Remark 24. If you later participate in the Computer Olympiad beta test, remember to remove the added line and uncomment the original.

Remark 25. If you later participate in the Computer Olympiad, remember to set BETA_TEST_MODE=False.

3.2 Creating the Referee “Bot Account”

We now explain how to create a “bot account” for a referee bot. Note that, you need to create a bot account for each referee bot.

The procedure is identical to the one in Section 2.2.1 except that:

- for step 2: enable “Server Members Intent” (in addition to “Message Content Intent”),
- for step 5: also enable the “Manage Messages” permission and,
- for each referee bot, the corresponding token must be set for REFEREE_BOT_ i _DISCORD_TOKEN in parameter.conf.

3.3 Inviting Players to the Server

We now explain how to invite human players and bot operators to your server:

1. Right-click the icon of your server in the vertical server bar on the left.
2. Click “Invite People”.
3. Click the “Copy” button.
4. Send the copied link to the people you want to invite.

The invited participants must then follow the protocol described in Section 1.2.1 using this link instead of the Computer Olympiad server link.

3.4 Inviting Bots to the Server

We now explain how to invite player bots and referee bots to your server. We assume that you have followed Section 3.1 and obtained the referee bot invitation URL, and that participants have also provided their invitation URLs after following Section 2.2. For each URL:

1. Paste the URL into your browser.
2. Log in to your Discord account (if required).
3. Choose your server from the list of available servers.
4. Click the “Continue” button.
5. If it is someone else’s bot:
 - (a) Verify that the requested permissions match those specified in the corresponding bot account creation section (player: Section 2.2 / referee: Section 3.1).
6. Click the “Authorize” button.

3.5 Starting the Referee Bot

We explain in this section how to launch the referee bot. To do so, run the following command in the terminal from the root of the discord_interface folder:

“python3 main_referee_1.py”

Manual-mode players should then follow Section 1 and automatic-mode players should follow Section 2.

Remark 26. To start a second referee bot, run: “python3 main_referee_2.py”.

Remark 27. If the referee bot program crashes during a match, restart it and then execute the !continue command in the match channel.

Remark 28. If one of the player bots crashes during a match and restarting it and using !continue does not fix the issue, you can try killing the referee bot, restarting it, and using the !continue command (if you suspect the problem originated from or propagated to the referee bot).

Remark 29. To modify the remaining time of a player, run the Discord command: “!increase_time @<player> <time_in_seconds>”. For example, “!increase_time @Alice 60” increases the time of Alice by 60 seconds.