User Manual

Dialogue Experimental Toolkit

Compiled 23rd August 2020

Introduction

This software is still being developed. We are still developing features. In previous versions of the software, we incorporated lots of features, attempting to second guess researchers' needs. But it turned out that often researchers didn't use these features, leading to feature-bloat. So, if you think something is missing or could be added – email us (g.j.mills@rug.nl) Or better still – get involved in the project!

This is the user manual for using the software. There is a separate manual for progamming the chattool [ ].

About

The toolkit is a text-based chat tool for carrying out experiments on dialogue. It can be used to collect naturalistic as well as task-oriented conversational data. It provides a set of customizable interfaces that capture in fine-grain participants' interaction with the interface and with each other. The chattool allows participants to be assigned to different linguistic communities.

In addition, the chat tool provides a toolkit for conducting finegrained experiments on the unfolding interaction. The server intercepts participants' typed turns and selectively interferes with their content. This allows very fine-grained experimental control over what participants perceive the other participants as having typed. This toolkit allows for the investigation of AI-mediated communication – the content, timing and sequencing of participants' turns can be manipulated

Some typical use cases:

- Running psycholinguistic experiments – e.g. collaborative referential tasks
- Running psycholinguistic experiments that directly manipulate participants' data.
- Running HCI/CMC/CSCW experiments that collect fine-grained data concerning participants' use of the various built-in chat interfaces
- Collecting longitudinal, naturalistic chat data from participants (using the Telegram interface)

Possible kinds of experimental intervention are:

Introducing artificial feedback into the dialogue

- Artificial clarification requests, such as "What?" "so you mean?"
- Acknowledgments, such as "ok" "ok right"
- Other discourse markers, e.g. "so?"

Blocking or transforming certain kinds of feedback:

- Substituting "ok" with "hmm" or "ok, tell me more"
- Blocking occurrences of "what?"
- Introducing fake interruptions

Introducing artificial hesitations and disfluencies

- introducing "umm" or "erm" into the dialogue

Transforming the identity of the other participant

- Making it appear as if the person is of different gender (e.g. use differently gendered name)
- Assign participants to different roles (e.g. overhearer vs. eavesdropper)

Substition of synonyms / hypernyms / hyponyms

- E.g. use WordNet or other resource to selectively substitute words in participants' turns.

Blocking or promoting alignment

The chat tool has a constantly expanding library of experiments that can be reconfigured. It also provides an extensive API that allows the programmer to design experimental interventions that are sensitive to the conversational context of the participants. The newest version of the chattool is integrated with Telegram Messenger. Participants can participate in dialogue experiments within the Telegram Messenger app.

What's new in v5?

The major changes since the last version of the chattool are:

- A *usermanual!*
- *Telegram:* Participants can connect to the chattool using the Telegram app
- *Remote admin:* Experimenters can monitor and control experiments remotely (using the Telegram app)
- *Customizable referential task:* The chattool contains a customizable referential task which allows experimenters to design joint reference tasks without needing to program anything. It can be used in experiments that use the Telegram app

How to use this manual

This manual consists of two separate parts. The first half describes how to use the chattool together with the Telegram platform. The second half describes how to use the chattool together with the built-in interfaces (Turn By Turn and What You See is What You Get).

Make sure you understand the advantages/disadvantages of using the different types of interface (see below)

Telegram vs. Built-in interfaces

Below are renderings of the three interfaces that can be used with the chattool

Figure 1 Telegram Interface: Participants download the Telegram app on their mobile (iphone/android) and use it to connect to the chattool via the Telegram bot service.

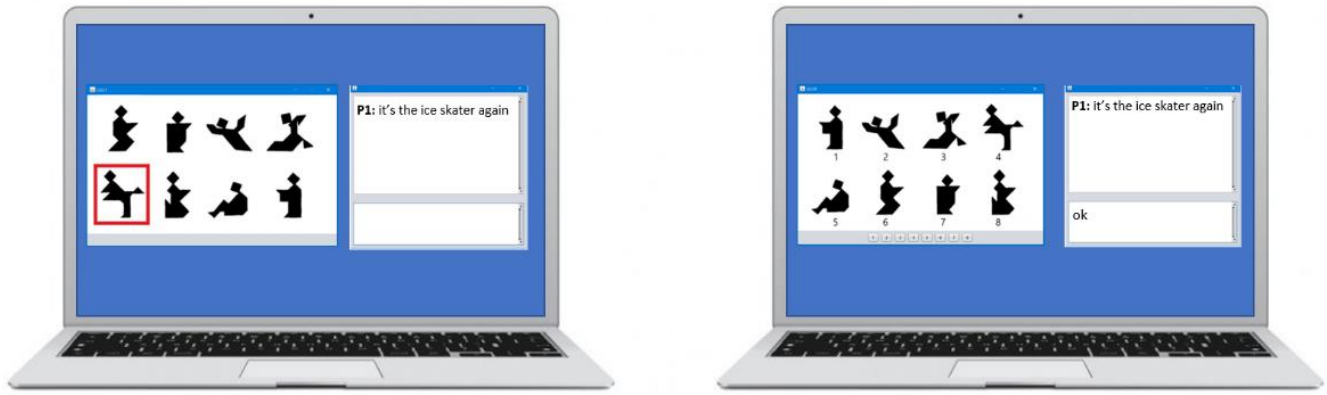


Figure 2 Built-in Turn By Turn (TBT) Interface. This is a custom, java programmed interface that captures the timestamp of each keypress and the timestamp of everything that is displayed on the interface

.

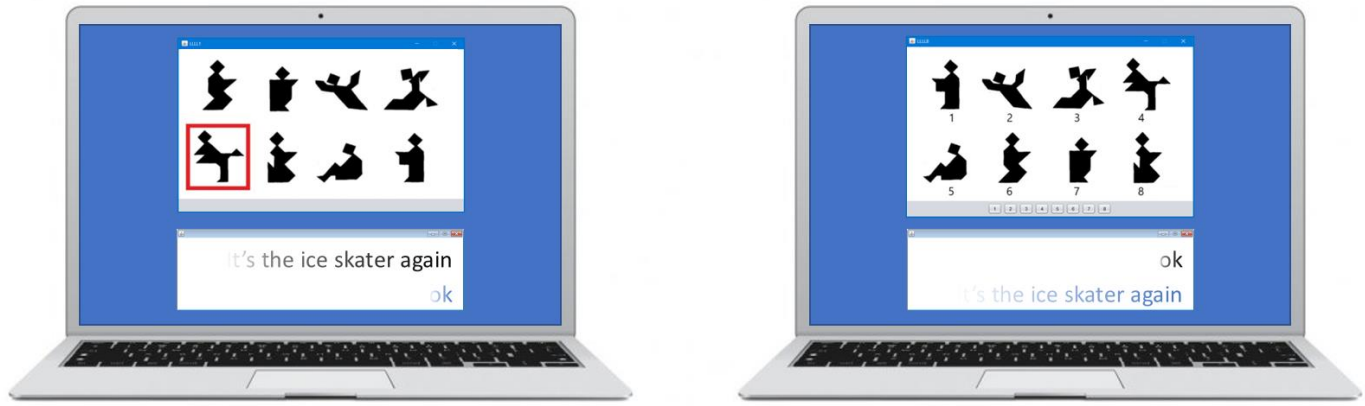


Figure 3 Built-in What You See is What You Get (WYSIWYG). This interface displays each character as it is typed. Each character disappears after being typed. This interface is suitable for investigating incrementality in language processing.

	Telegram	Built-in	
		Turn By Turn	WYSIWYG
Runs on participants' mobile phones	X		
Runs on particicipants' PC	X	X	X
Can require configuring network settings of firewall		X	X
Can run maze game		X	X
Can run referential tasks	X	X	X
Disappearing messages	X		X
Experimenter can block text entry by participant (i.e. prevent typing)		X	X
The timestamp of each keypress is recorded by the client		X	X
Manipulations can modify content of messages	X	X	X
Manipulations can modify the identity of the sender of messages	X	X	X
Manipulations can modify the timing of messages	X	X	X
Manipulations can assign participants to different (sub-) groups	X	X	X
Participants can send each other images	X		
Participants can send each other sound clips	X		

Part 1: Using Telegram

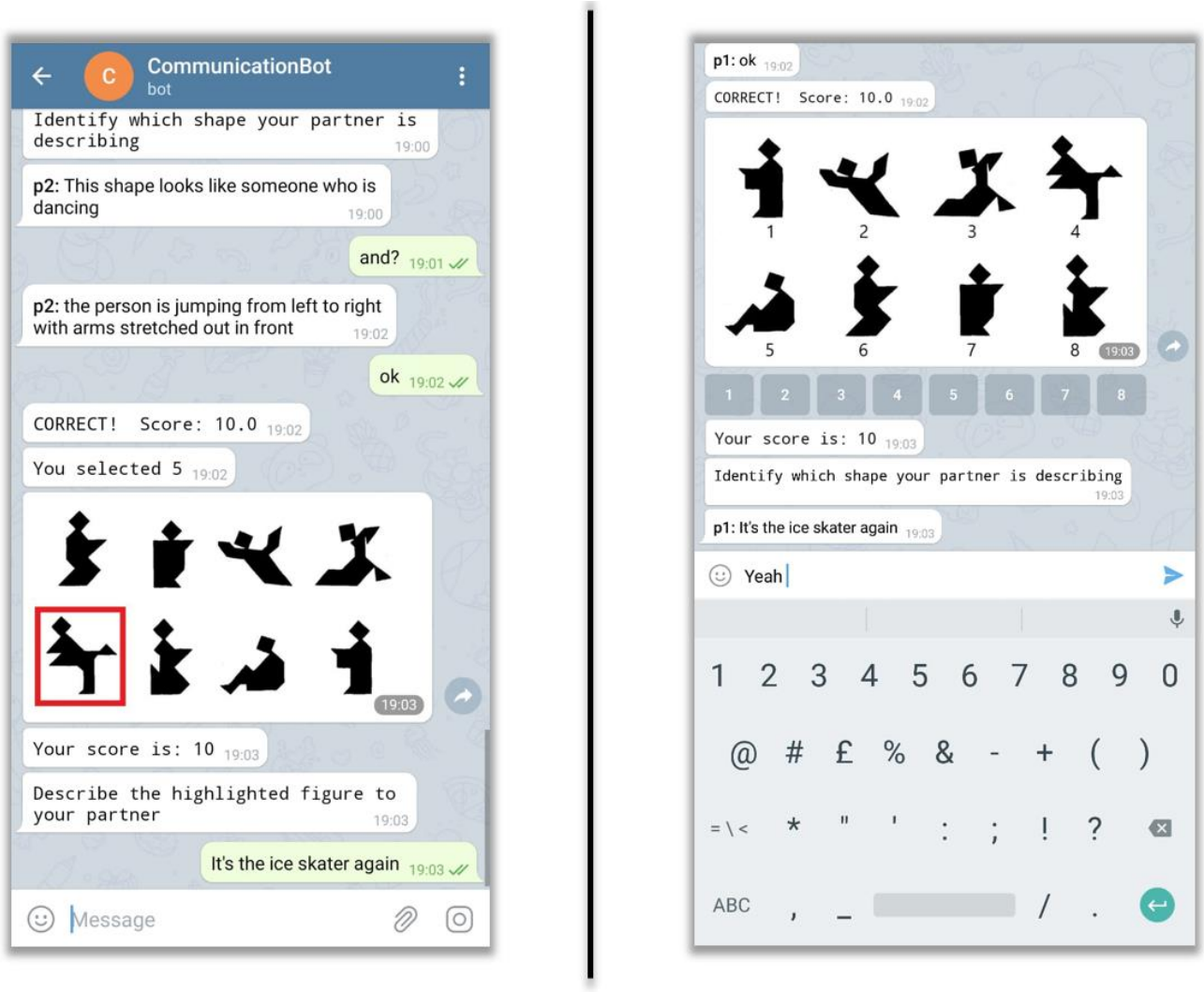


Figure 4 An example showing two participants playing the tangram task inside the Telegram app. In this setup, all participants' turns are sent to the DiET server. The DiET server also controls the presentation of the tangram stimuli

Why use Telegram?

- Participants can use their mobile phones to participate (Both android and iPhone)
- Telegram is much easier to setup for participants.
  It takes a few minutes to download the app from the google “play store” / apple “app store”.
- It is possible to run customizable referential tasks within the Telegram app
- Makes it easy to design experiments where participants take part over days/weeks.
- Telegram is a bona fide chat app – similar to Whatsapp, yielding more natural interactions.

When not to use Telegram

- If you want to run the maze game or confidence task, this requires using built-in DiET code
- If you want to record the exact timestamp of each keypress, you need to use a built-in interface
- If you want to use a character by character interface, you need to use the built-in WYSIWYG interface

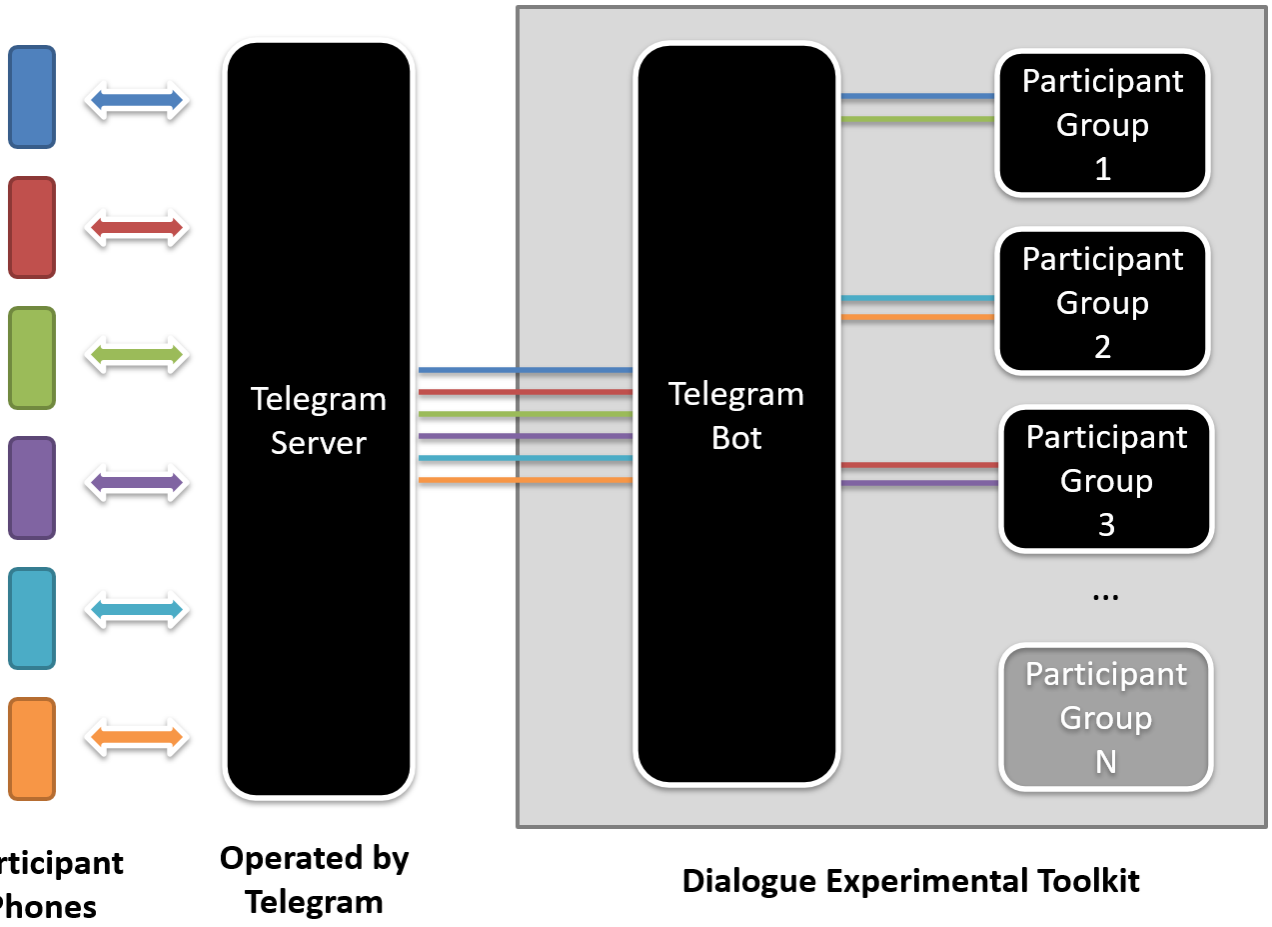


Figure 5 Schema showing how participants connect to the DiET server via the Telegram service. Notice how 6 participants are paired into three separate dyads (Group 1, Group 2, Group 3). Note that the Telegram Server runs "in the cloud", whereas the Dialogue Experimental Toolkit software runs on a local machine controlled by the experimenter.

How does it work?

Participants run the Telegram instant messaging app on their mobile phone. They interact with the DiET bot – which is connected to the Telegram service. This bot receives participants' messages and responds. However, unlike typical bots (such as a travel booking system or a bot that looks up items in a dictionary), this bot routes participants' messages *to other participants*, allowing control over who speaks to whom. The DiET server also allows experimental control over manipulating the content and timing of participants'turns. Note, the Dialogue Experimental Toolkit does not connect directly to the participants. Instead, the participants connect to the Telegram Service. Similarly the Dialogue Experimental Toolkit connects to the Telegram service, as a Bot. Subsequently, when participants,u sing the Telegram app send a message to the Bot, the Telegram service relays those messages to the Bot.

Equipment needed

To use the Telegram version – you need:

1. One PC (windows,apple,linux) that runs the DiET chat server software.
   The chat software needs an internet connection.
2. Participants use their mobile phones, running the Telegram app.

Quick start with Telegram

For this you will need a PC connected to the internet and one (ideally two) phones (iPhone or Android) that are also connected to the internet. This will take about 10 minutes. You only need to follow these steps once.

1. Download / install / run the chattool software (2 mins)
2. Register the chattool server with the Telegram service (3 mins)
3. Start the chattool server (1 mins)
4. Connect the participants' phones to the chattool via the Telegram service. (3 mins)

Step 1: Downloading the server software

First, download the latest zip file from github. The latest version (May 3rd 2020 is available at)
[https://github.com/dialoguetoolkit/chattool/releases/download/4.6.2/chattool_runme.zip]

Unzip the contents of the folder into a subfolder. For the purposes of these instructions, it is assumed that you unzipped the contents into a folder on the desktop: **/desktop/chattool/**
Verify you have the following files

/desktop/chattool/data
/desktop/chattool/experimentresources/
/desktop/chattool/jre-10.0.2/
/desktop/chattool/chattool.jar
/desktop/chattool/readme.txt
/desktop/chattool/start.bat

Step 2: Run the server software

Windows

Double-click on “start.bat”. This should start the program.
Press the “start server” button (See Figure 1 below)

Mac / Linux

1. Double-click on “chattool.jar” . If this works, this means you have java installed.
2. If double-clicking doesn’t work, open a Terminal window and navigate to the folder where “chattool.jar” is located, (i.e. **/desktop/chattool/**) then type j**ava -jar “chattool.jar**

If that doesn’t work, you need to install java. Follow the instructions on installing the latest version of java runtime for your computer (see e.g. https://support.apple.com/en-us/HT204036), and then retry steps 1 and 2 listed above.

When you see the screen in Figure [ ] (below), select the “start server” button

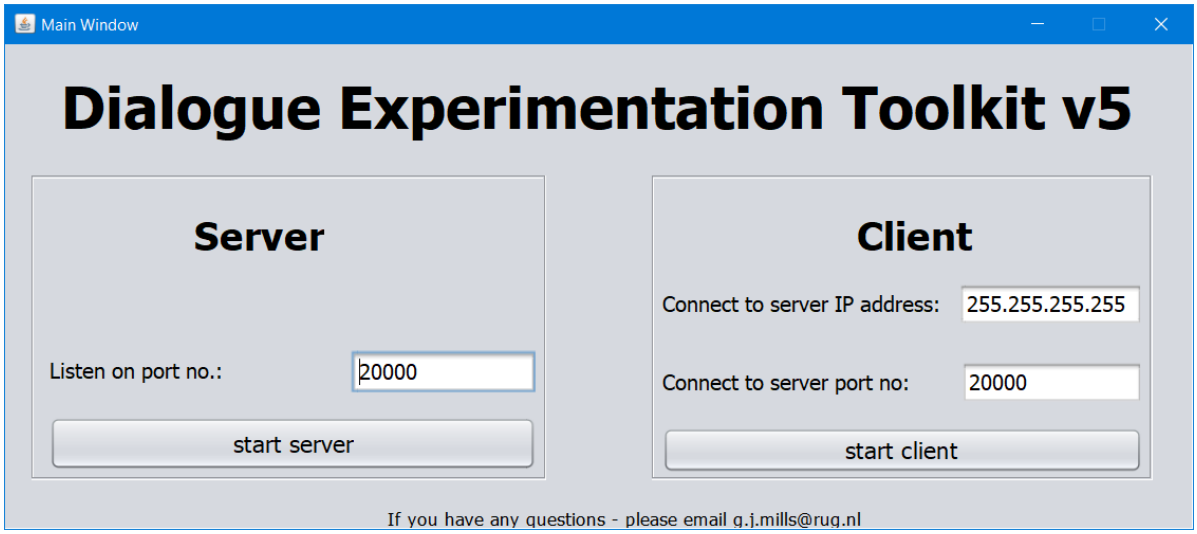
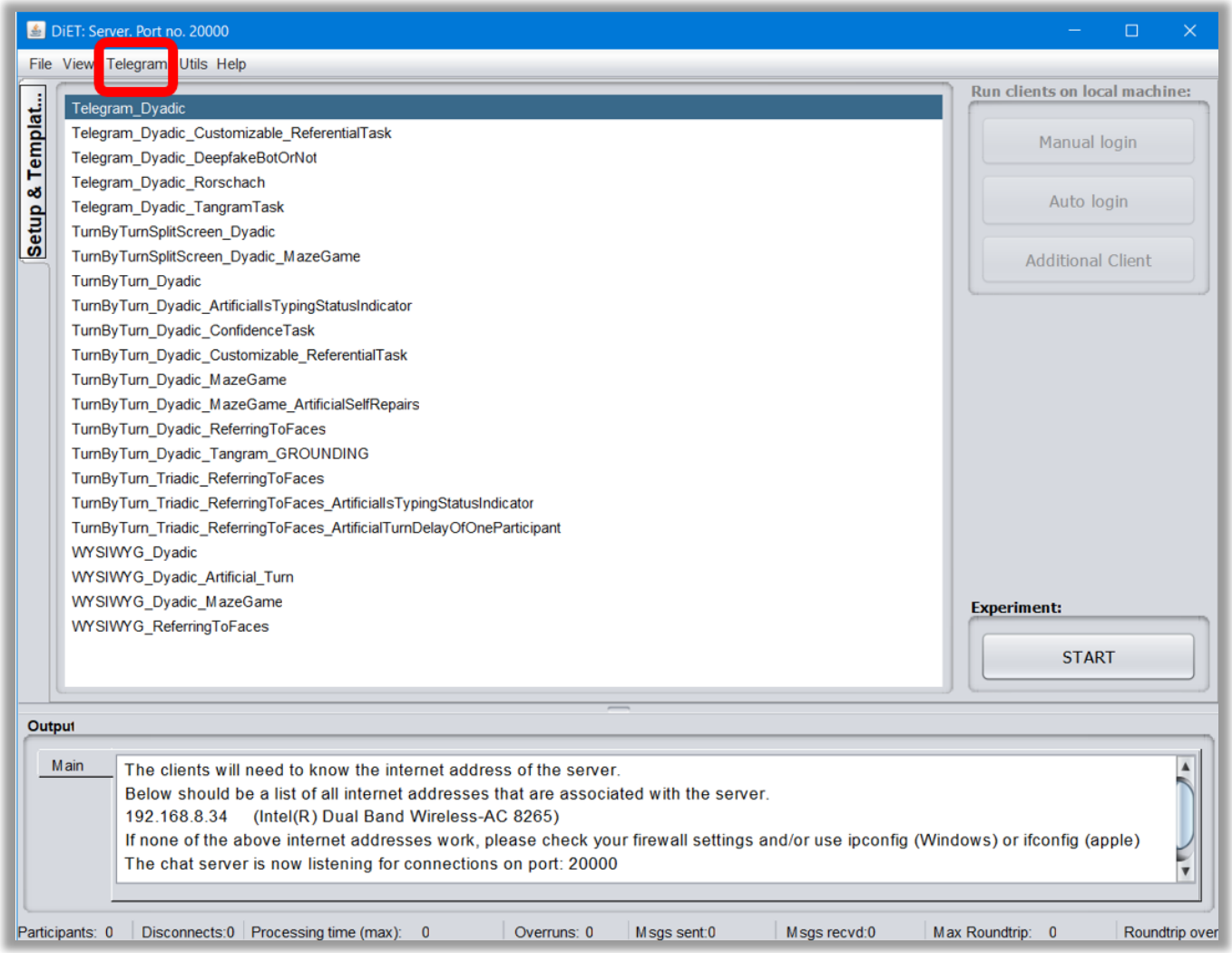


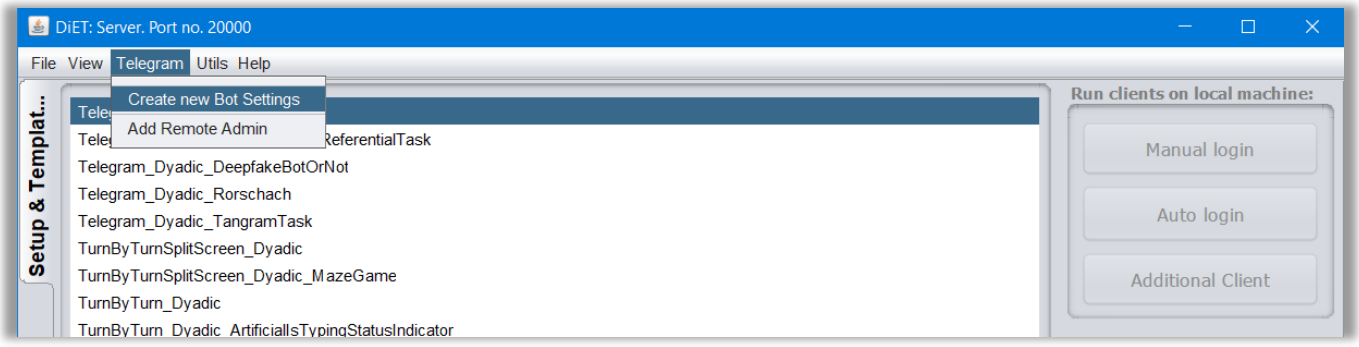
Figure 6 When you start the software without specifying any parameters, it lets you start the server or the client

After you select “start server”, the main window should open (see Figure [ ] below).

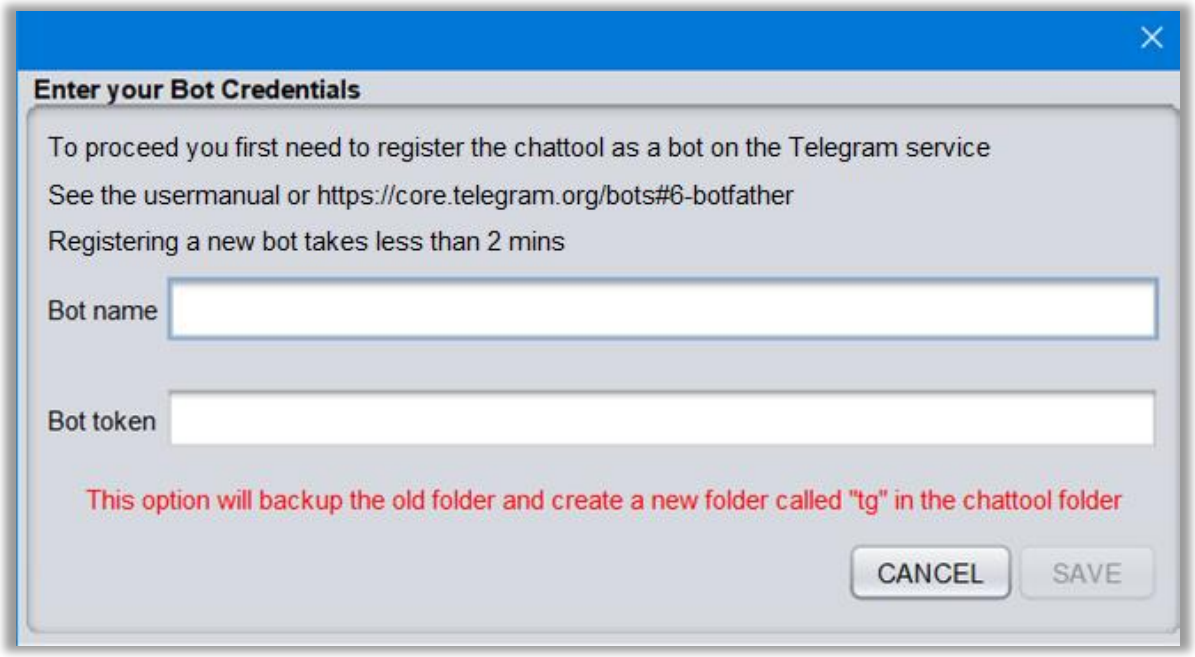
Step 3: Register the chattool server with the Telegram service



Select “Telegram” from the Menu and then select “Create new Bot Settings”



This will open a new menu.



In order for the chattool to function as a bot it needs to be registered on the Telegram service.

To register the chattool with the Telegram service, you first need to request a new set of credentials (bot name and bot token) from the Telegram service. This can only be done within the Telegram app (on a mobile phone). The next steps show how to this.

Step 4: Setting up Telegram – Creating Bot Credentials inside the Telegram App

To complete this step you need a mobile phone (android or iphone) with an internet connection.

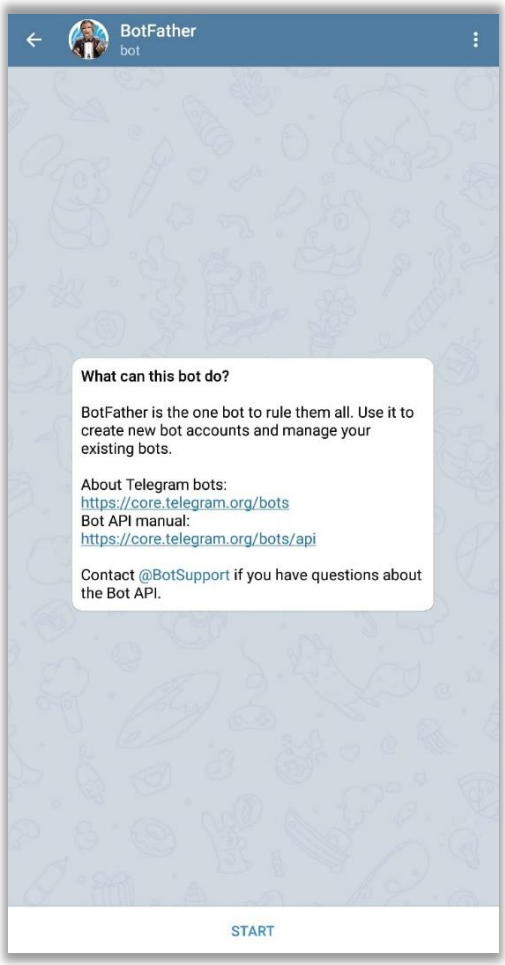
This information is from the official Telegram webpage https://core.telegram.org/bots#6-botfather

Step 4.1

Install the Telegram app. Then, in the Telegram app, select the search option in the top right hand corner of the screen
(highlighted here in red)

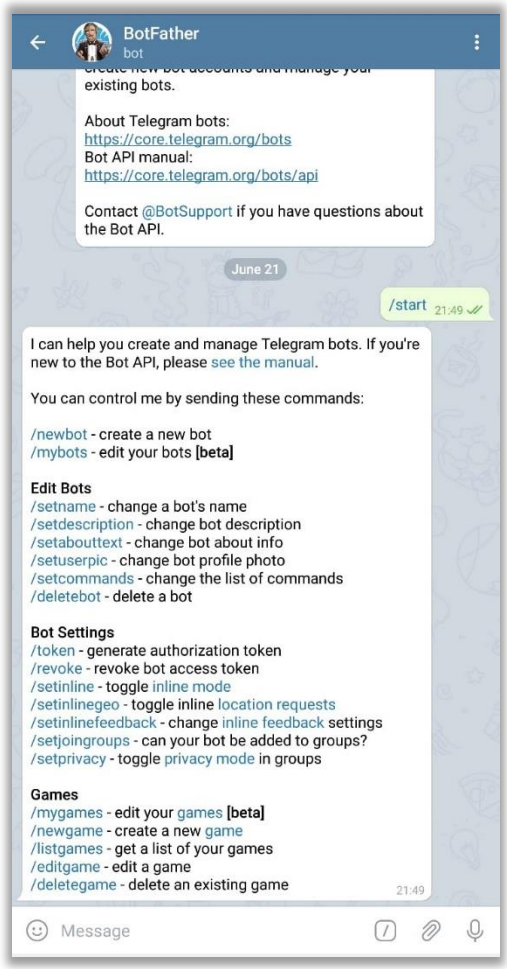
Step 4.2

In the text-field, enter “BotFather”.
This is the official Telegram service that allows you to create your own bots.



Step 4.3

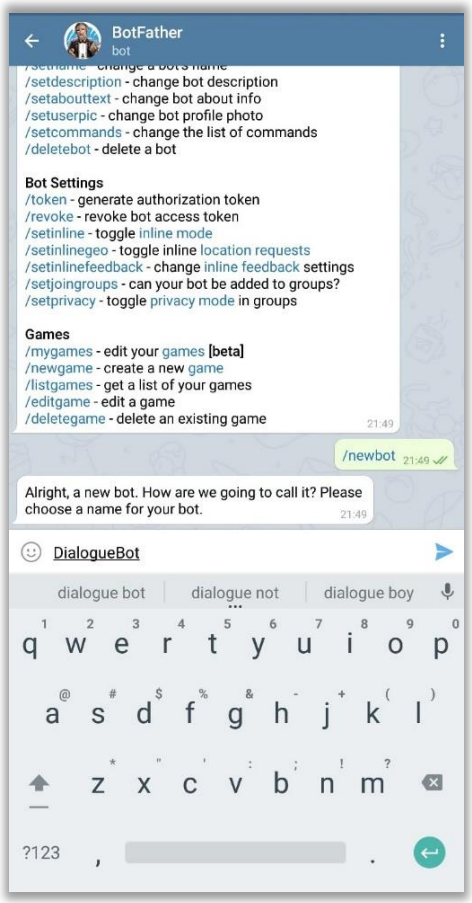
It should show you a summary of the "BotFather" app. At the bottom of the screen is the "start" button



Step 4.4

Select **/start** to start the "BotFather" app

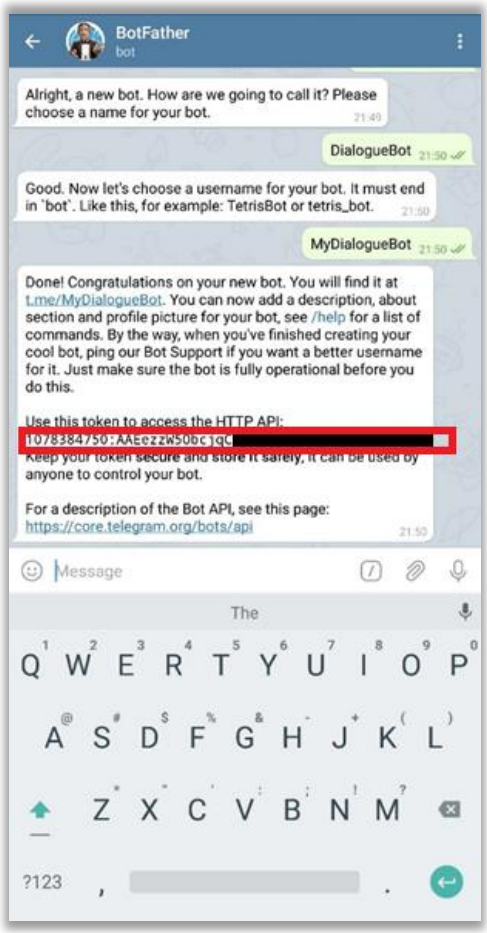
It should give you a list of options to select.



Step 4.5

Select /newbot to create a new Bot on the Telegram service.

(Choose a username for your bot. In this example the username is DialogueBot. You should choose a different name for your bot)



Step 4.6

Choose a name for your bot
(in this example the name is ***MyDialogueBot*)**

Make a note of your Telegram API token.
(In this example the token starts with
***1078384750:AAEezzW5Obcjq……..*)**

(N.B. Part of the token is obscured for security)

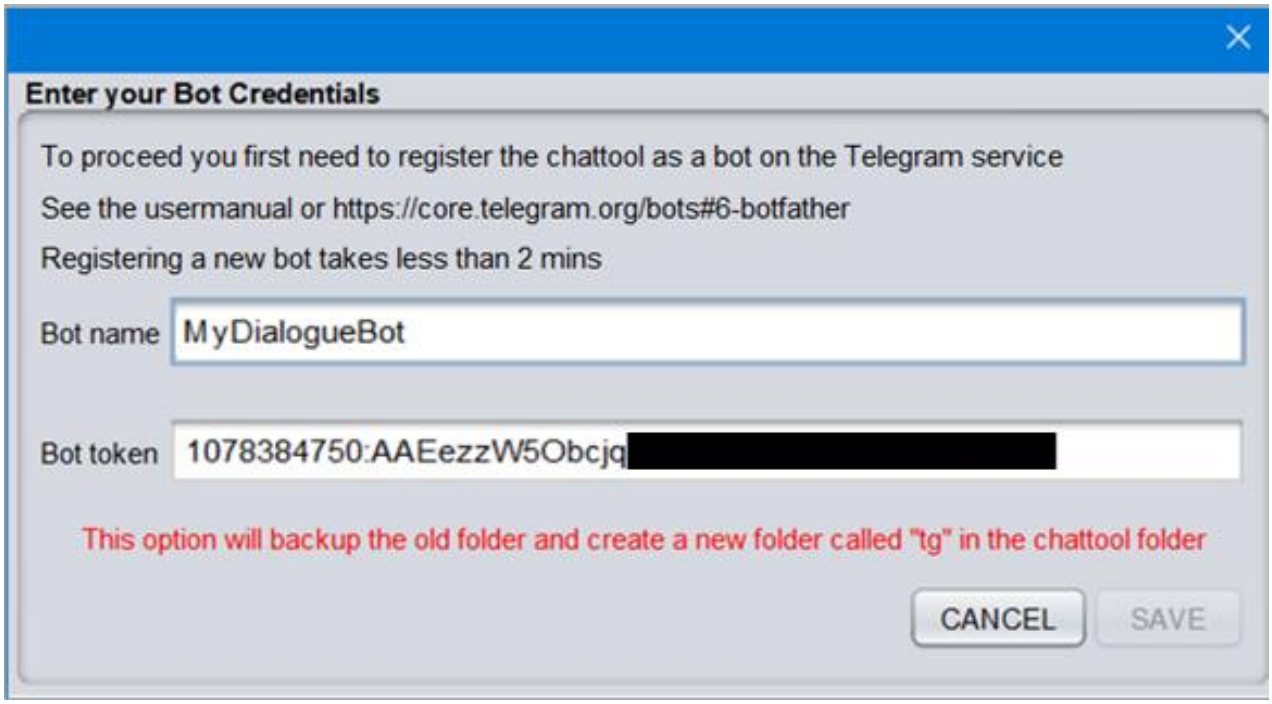
Now you should have two credentials:

Your Bot Name: *e.g. MyDialogueBot*
Your Bot Token: *e.g.* ***1078384750:AAEezzW5Obcjq***

You will need to use both credentials in the next steps.

Step 5: Entering the newly created Telegram credentials in the chattool

You need to enter the name of your bot and add your bot token from the step [ ] above:



The steps above have now created a new directory called “tg” in the chattool directory

In that folder is a text file called “botname.txt”

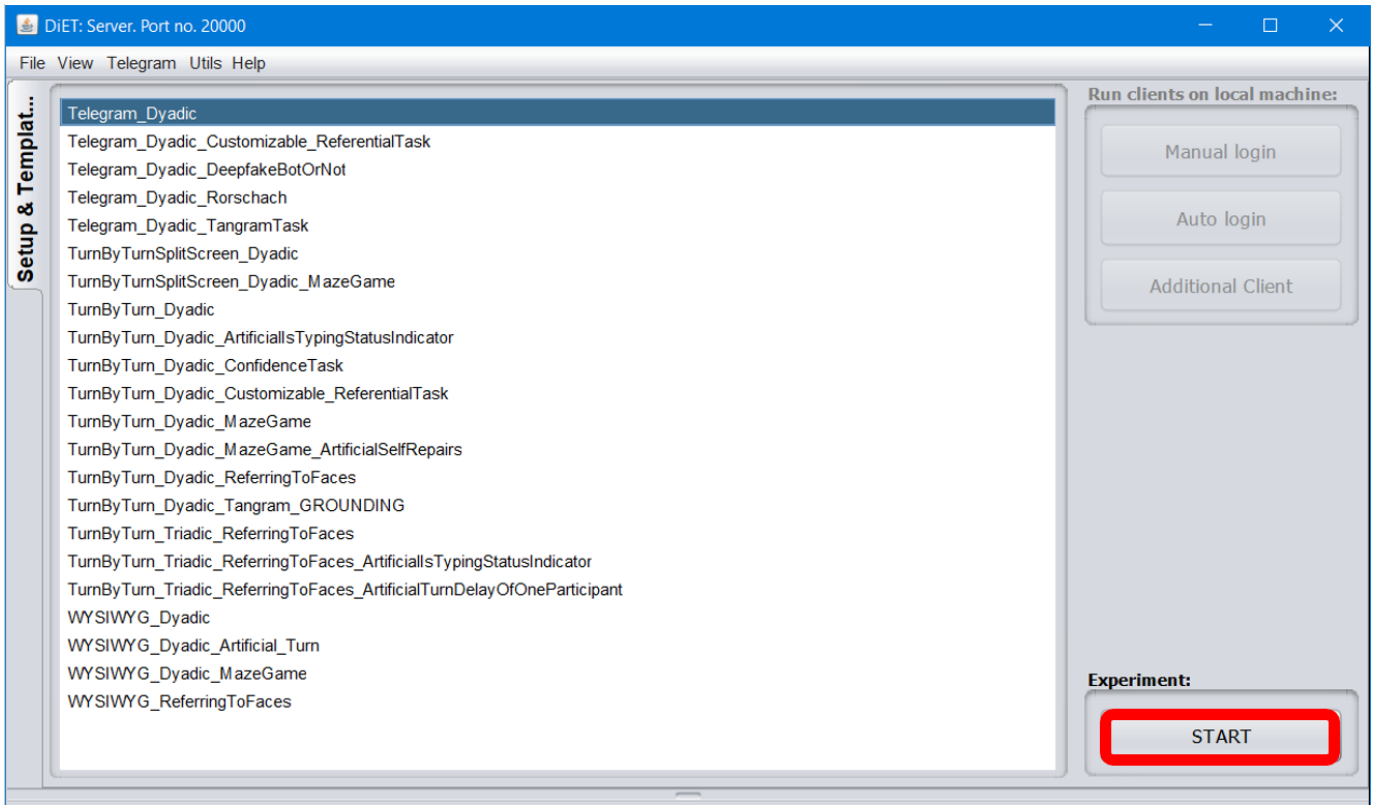
- The first row of “botname.txt” contains the name of the Bot (e.g. MyDialogueBot)
- The second row of “botname.txt” contains the token (e.g. 078384750:AAEezzW5Obcjq)

It’s best to close down the server and then restart the server: Select “File” and then “Exit”

Step 6: Starting the Telegram Template

Start the server (As in the steps above) [List them again]

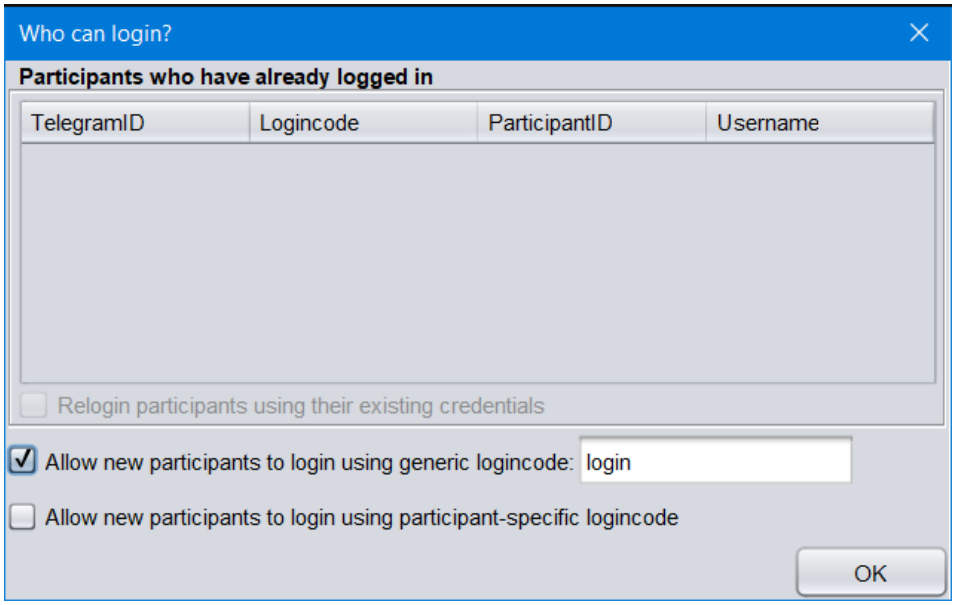
Select one of the Templates that is prefixed with “Telegram”, e.g. the template “Tangram_Dyadic”.
This is a simple template that connects pairs of Telegram users with each other.



Select “start” in the bottom right. This will start the template, and connect to the Telegram service.

N.B. The options in the top right of the GUI “Manual login, auto login and Additional client” are greyed out.
This is because these options are only relevant for interventions that use the Java interface (i.e. non-Telegram experiments)

You will now see an option asking you “Who can login?”



This is a screen that loads whenever you start a Telegram experiment. It asks you:

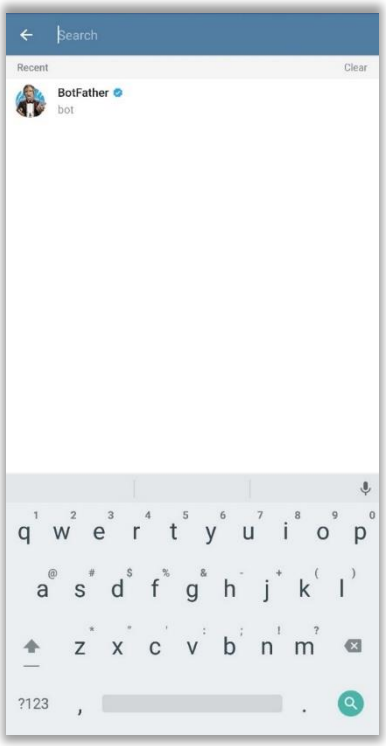
- What to do with participants who have logged in previously – should the chattool log them back in? (This is useful if you want to run longitudinal experiments and switch off the experiment, e.g. overnight/ for debugging). Since this is the first time running a Telegram experiment, participants haven’t logged in yet, which is why it is greyed out.
- Whether to let new users log in, and if so, how should they authenticate themselves. When participants connect to the server, the server asks them for a login code. There are two options:
    a. *Allow new participants to login using generic logincode.*
    The same logincode is given to all participants.
    When participants login and enter the code, the chattool automatically assigns them a participant ID and username.
    In this case the logincode is “login” (which can be changed by editing the textfield)
    b. Allow new participants to login using participant-specific logincode:
    Each participant uses a different login code.
    The chattool contains a list of unique login codes in
    “../experimentresources/permittedparticipantids/permittedparticipantids.txt
    This is useful if you have experimental interventions that depend on the specific identity of the participant.

Just select “OK” here

Step 7: Connecting a participant's phone to the telegram service

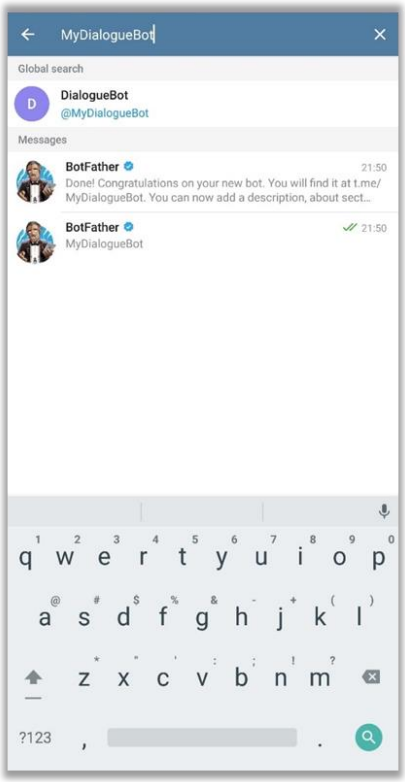
Take a phone, install Telegram on the phone (it can be the phone you used in the previous steps 4.1-4.6), and then follow the steps below:

Here, instead of searching for "BotFather" [as in steps 4.1 – 4.6 above] , we search for the Bot we created.



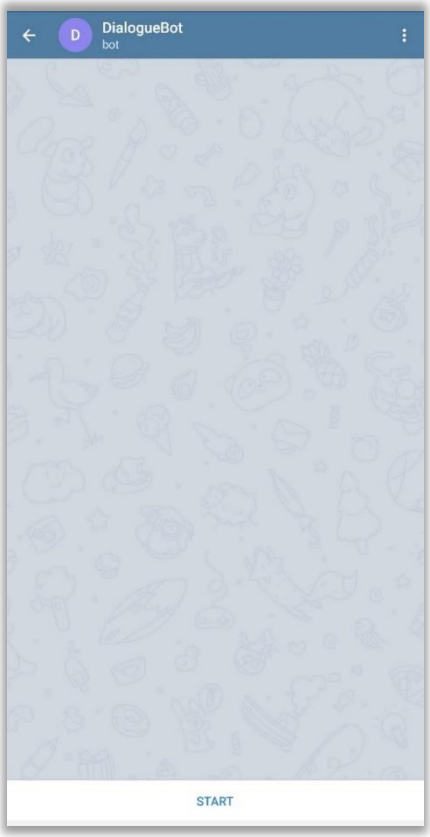
Step 7.1

Press the telegram search option at the top of the screen



Step 7.2

Enter the name of your bot you created in the previous step (in this example the bot is MyDialogueBot)



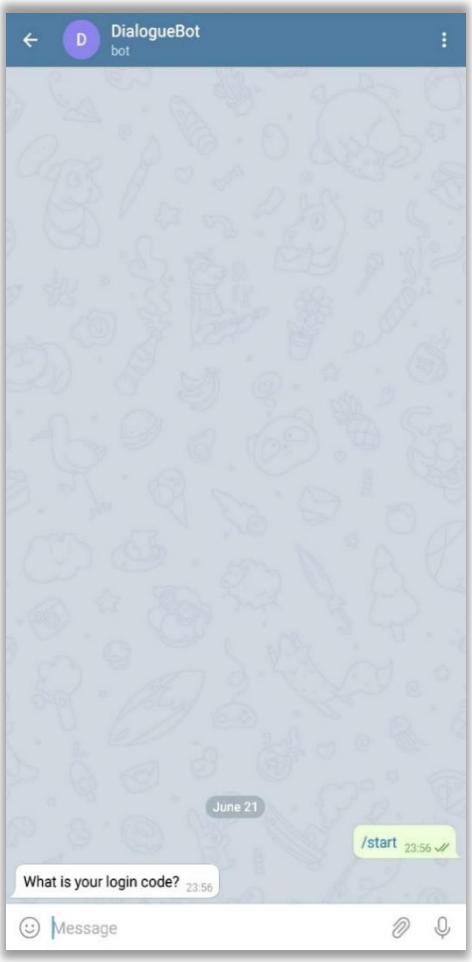
Step 7.3

The telegram app will open a new screen for your bot.

Notice the name of the bot (DialogueBot) at the top of the screen.

Press START

Once you have pressed start, all messages you type will be received by the Telegram service and then sent on to the chattool.



Step 7.4

You should see a login prompt “what is your login code?”

This is a login prompt that is generated by the DiET server.

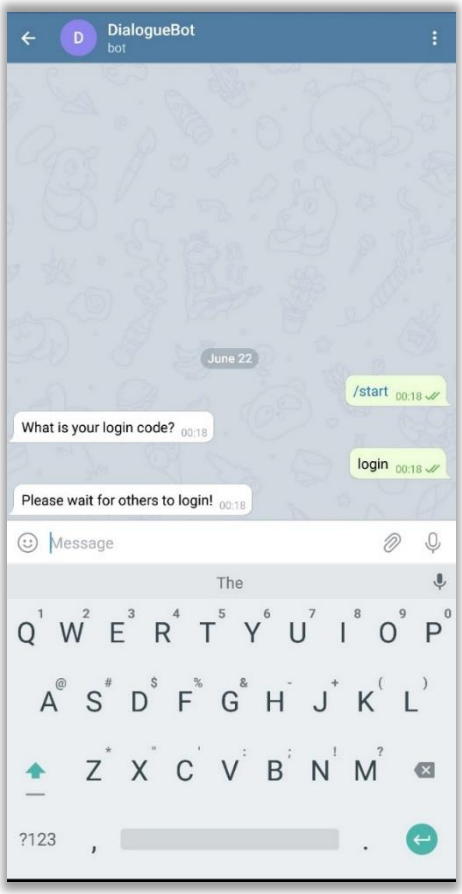
When you see this message, this means that the telegram client has connected to the DiET server.

Step 7.5

Enter “login” and press send.

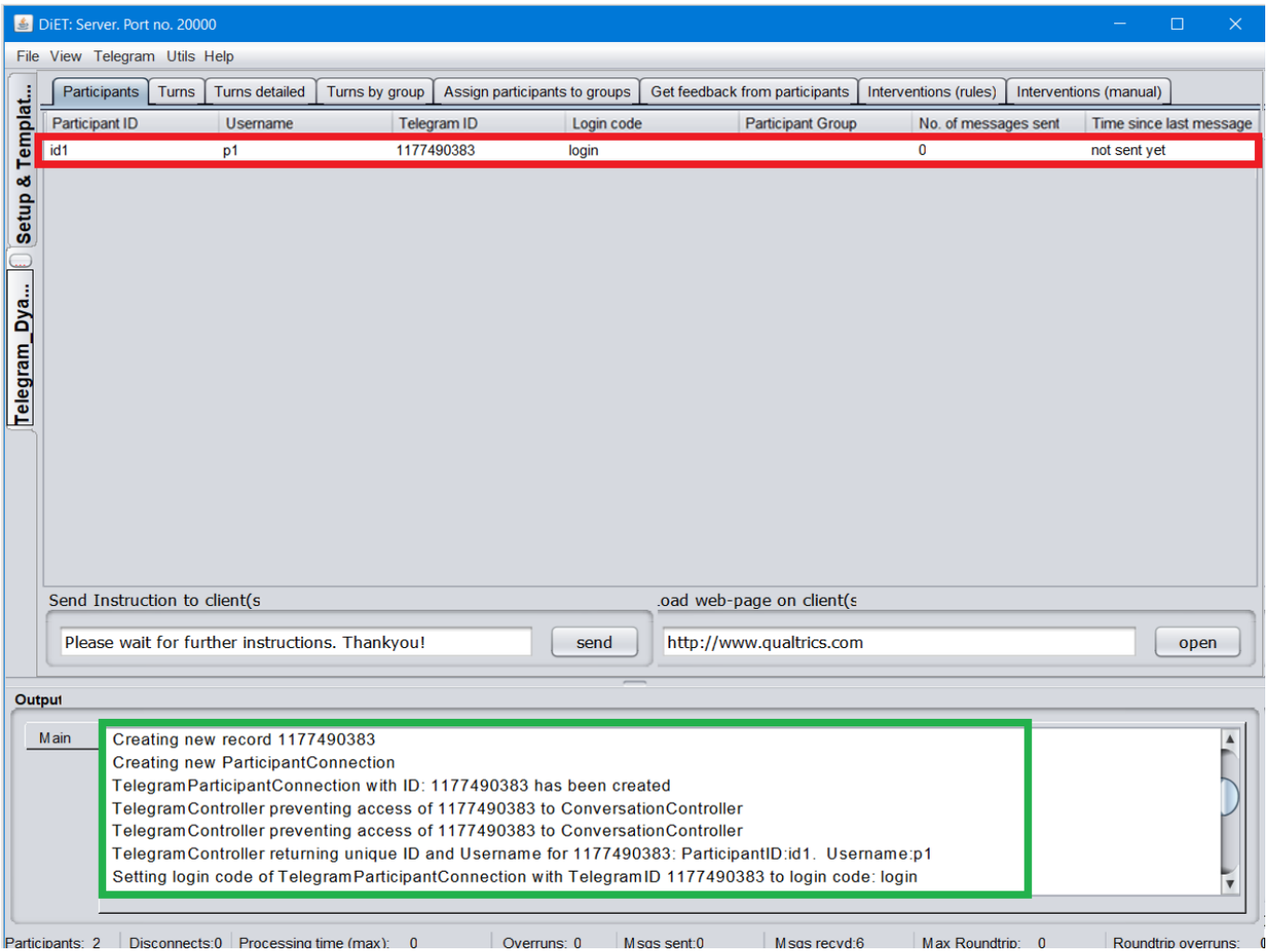
(This login key is necessary to prevent other telegram users from connecting to the server

(You can of course change this login key in the settings)



Step 7.6

You should see a message telling you to wait for the other participants to login.



In the top part of the DiET interface you should see confirmation that the participant has logged in. See the red highlighted rectangle. This shows that the participant is logged in, has a participant ID of id1, a username of "p1", a telegram ID of 1177490383, and that the participant logged in with a login code "login".

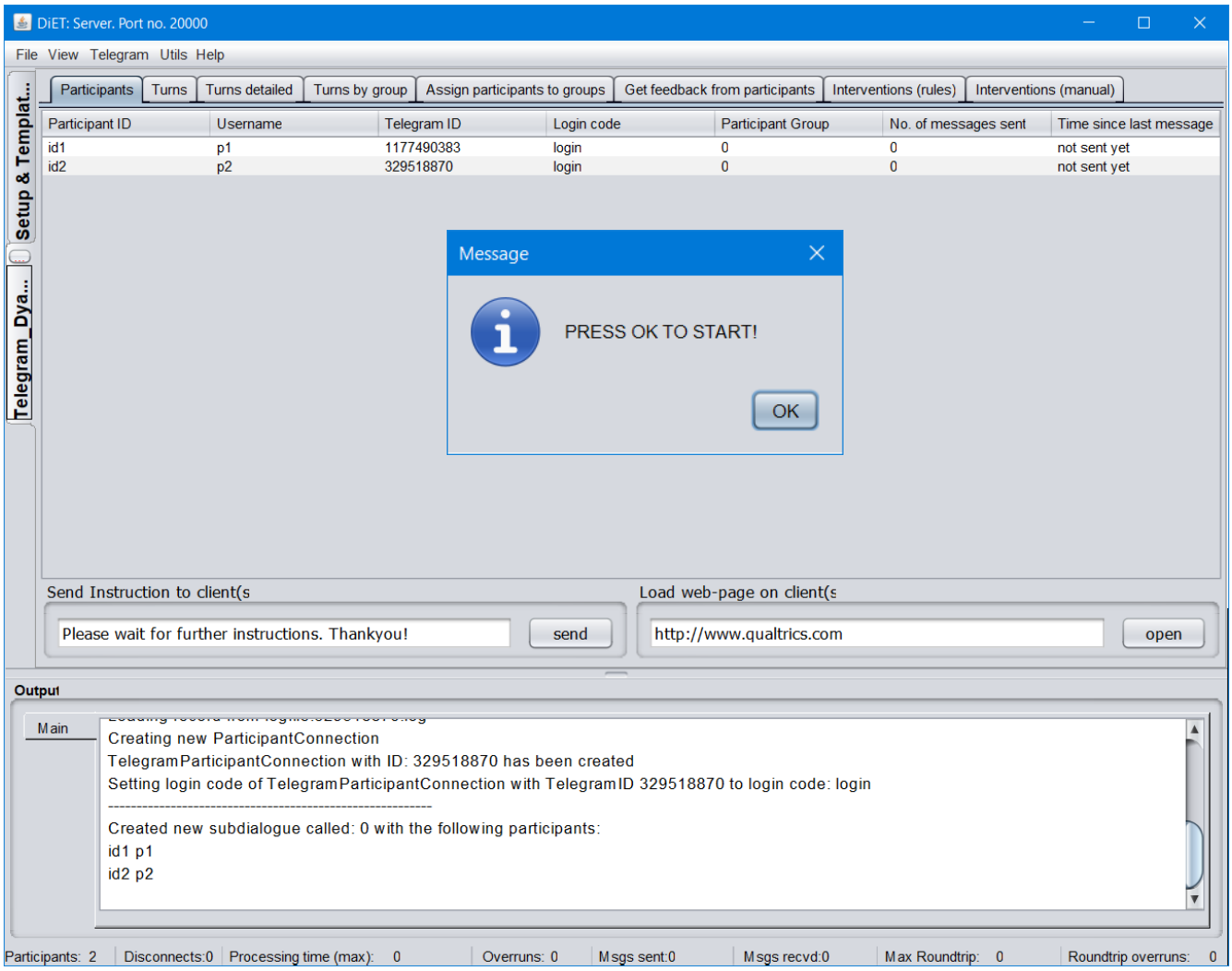
In the lower window of the DiET interface you should also see more detailed information about the login process (see the green highlighted rectangle).

When you have completed steps 1-6 above, take a second phone, install Telegram and repeat the previous 6 steps above.

Step 8: Starting the experiment

When both participants have logged in, you should see both participants listed on the server.

You should also see a confirmation message saying “PRESS OK TO START”. PRESS OK.

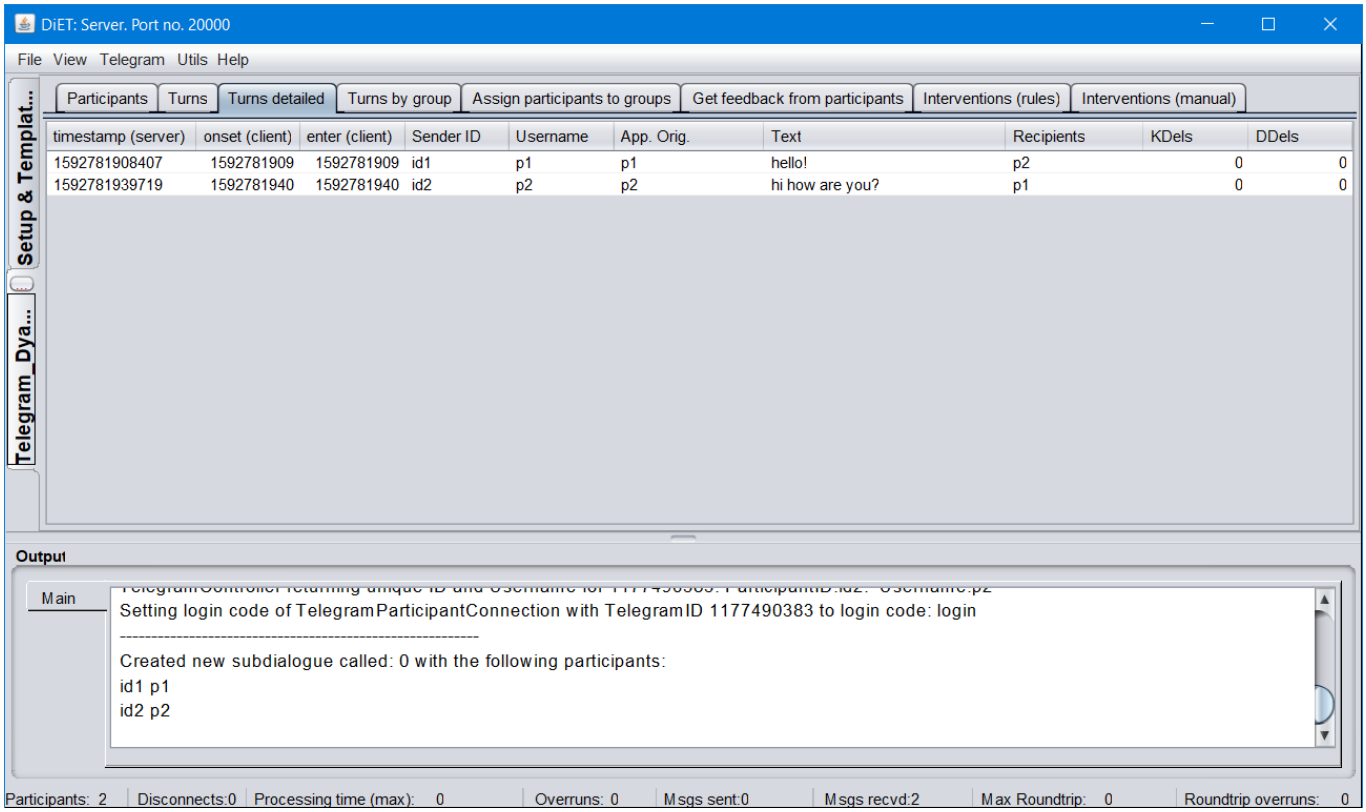


After pressing OK, both participants can now send each other messages

What happens now is that the messages are mediated via the DiET server:

when a participant types and sends a message, this message is relayed by the Telegram service to the chattool. The chattool takes this message and sends it on to the other participant(s).

To get a closer look at the messages, press the “turns detailed” tab. You can see the turns that have been sent.



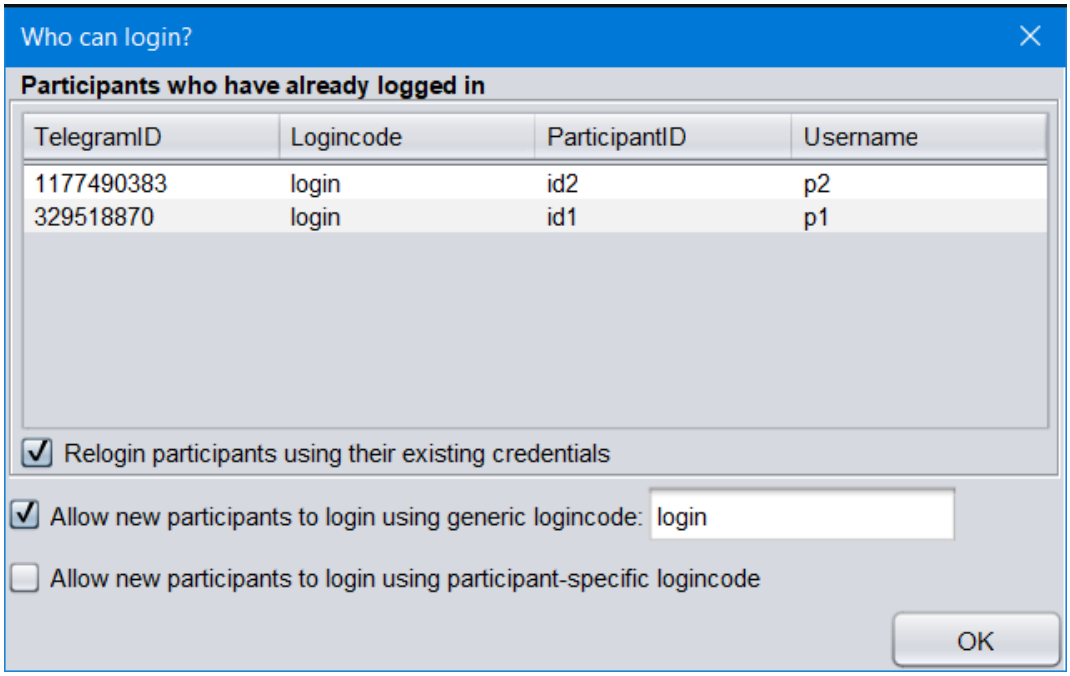
Step 9. Understanding how Telegram buffers messages

Close the chattool down (press “File”, then “Exit”)

Send some messages between the participants. Notice how inside the Telegram app it shows two ticks, but the messages aren’t received by the other participant. This is because the two ticks indicate that the message was received by the Telegram service – they are being buffered by the Telegram service and will be sent to the chattool as soon as the chattool restarts and connects again to the Telegram service (see next step).

Step 10. Restarting the experiment

Start the chattool and start the same Telegram template. You will see the following options:



This is the same popup as In [section …] above
[For now select "OK" – there is an explanation of what this interface does in section…]

There are three options:

1. *"Relogin participants using their existing credentials"*

   Notice in the table above the checkbox, it identifies the two participants you connected in the preceding steps. The first column contains the TelegramID (a unique ID assigned by the Telegram service to each participant's phone), Logincode, the participant ID and Username.
   The server asks if you want to log both participants back into the experiment.
   If you select this, these participants will be re-connected and the experiment can continue.
2. *Allow new participants to login using generic logincode*
   The server asks if you want to let more participants log into the experiment using "login" as the logincode
3. *Allow new participants to login using participant-specific logincode:*
   The server asks if you want to let more participants log into the experiment using one of the unique login codes , loaded from
   "./experimentresources/permittedparticipantids/permittedparticipantids.txt"

Select "OK".

Notice how the chattool logs the same participants back on. After this, the messages that were typed and sent in the Telegram mobile app are now received by the chat tool and then sent to the other participant.
[N.B. If you don't want to do more sophisticated login control, see section X, e.g. on how to reuse a subset of participants in a subsequent experiment.]

Using Telegram

Telegram directory structure

All Telegram information is saved in the /tg/ directory.
All files in this directory are saved in the UTF-8 Text format.

This folder contains the following files:

botname.txt

This file contains the credentials that the server uses to register as a bot on the Telegram network.
The first row contains the name of the bot, e.g. MyDialogueBot
The second row contains the token of the bot.

telegrambotio.txt

This file contains a log of every Telegram message that was sent and received by the server
Each row consists of three fields that are separated with the "|" character:

- A timestamp of when the message was saved. The timestamp is saved in the unix epoch time format (https://www.epochconverter.com/)
- Information about whether the message was sent by a clients or by the server:
  FROM means the message was sent FROM the server to the clients
  TO means the message was sent TO the server by the clients
- The message. This text saves the raw message format that is used by the Telegram service.

adminids.txt:

This file contains a list of telegram IDs of experiment administrators who can control the chattool remotely via the Telegram app [See below]

**********.log

The folder contains a log file for each telegram client that sends a message to the server. Whenever the server receives a message from a telegram client, it creates a new file where the filename is the telegram ID of the client, with a ".log" suffix. All telegram messages sent to and received from the client are also saved in this log file. When the client logs into the server, the server also saves the ParticipantID and the Username to this file.
IMPORTANT: Once a participant has logged in successfully using a logincode and has received a valid set of credentials (i.e. a participant ID and username), this ID and username will always be associated with that participant – unless you delete the logfile.

blockedids.txt:

If you want to block particular telegram users from connecting, enter their telegram ID (one per row).

A closer look at the login process

This section assumes that you have already completed the quick start (above). This section explains the login process in more detail. It explains what is going on "under the hood" when a participant logs in.

The chattool, internally, associates each participant with a unique participant ID and a unique username. The username is the term that is prefixed to participants' turns (This username can also be manipulated experimentally). Also, the experimental templates might place constraints on permissible participant IDs and usernames (e.g. a referential task might have the constraint that participant IDs must be prefixed with "D" or "M" for director or matcher, respectively. In addition, each participant will have a unique Telegram ID that is associated with the participant's mobile phone.

First of all the chattool needs to prevent non-participants from connecting to experiments via the Telegram service. This is done by giving participants a logincode that they need to provide when they first connect to the server. There are two types of logincode – generic and specific.

Generic login codes:

- The *generic code* is intended to make it as easy as possible to sign up an unknown/unlimited number of participants: The experimenter gives the same login code to all participants. When the participants provide this code, the chattool automatically creates a new, unique participant ID and username for each participant
- Consider a situation where you have a large (possibly unknown sized) group of participants, e.g. you have a large class of students (200 or so) who need to participate for course credit during an allocated timeslot. Or you have an audience of people, some of whom have their mobile phones with them and you tell the audience that they can participate if they wish. Here time constraints are quite important. It also often isn't practical giving each participant a unique logincode – if you give them a spreadsheet they often use the wrong code….and sending 200 separate emails is also quite time-consuming. There will always be one or two students who aren't on the roster, or who bring their friends and who you would like to participate. In this situation you can use a single generic logincode – the experimenter gives the *same* login code to all participants, which they then use to login. The server creates unique credentials for each telegram ID.

Specific login codes

- The *specific code* is intended so that that experimenter can assign participants to specific participant IDs. Consider another situation – you have an experiment where participants fill out a questionnaire before or after the experiment. In this case you do need to associate their ID in the chattool with the ID on the questionnaire. If this is the case, use specific logincodes. The directory "\experimentresources\permittedparticipantids\permittedparticipantids.txt" contains a list of 250 unique participant IDs that can be used. You can also edit this file and choose your own.
- Some experimental templates which have particular roles (E.g. director / matcher or instructor vs. follower) might allow the experimenter to assign the role based on the logincode. E.g. it could be that if the participant enters a sequence of six numbers as a logincode that the participant is assigned the role of director, whereas if the participant enters a different sequence of six numbers as a logincode, the participant is assigned the role of matcher. If this is the case then you need to use specific login codes. But this depends on the template.

The very first logon:

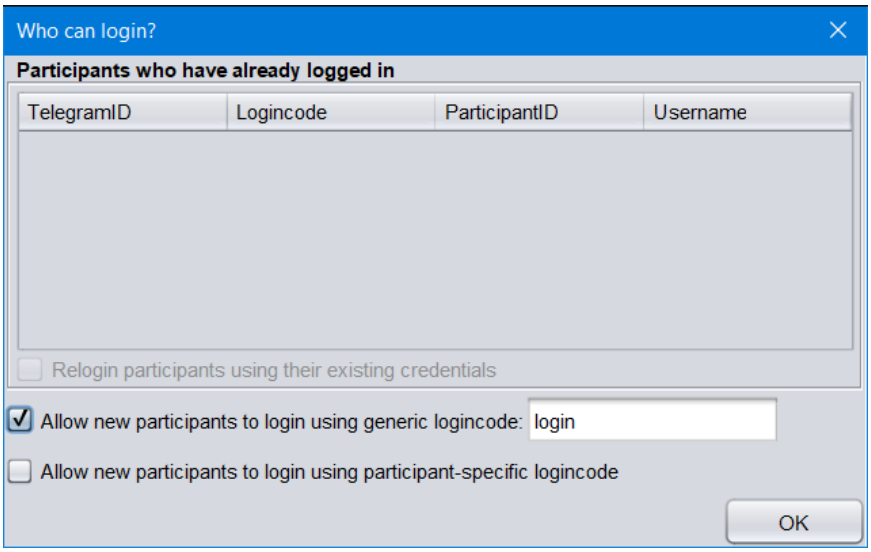


Figure 7 The options that are displayed when starting a Telegram script, before any participant has logged on (notice how the table of "Participants who have already logged in" is empty. This screen allows experimenter to enable generic and specific logincodes

1. The participant downloads and installs the Telegram client to their phone
2. In the telegram client, the participant finds the Bot that was created by the experimenter on the Telegram service [see figure X]
3. In the telegram client the participant connects to the Bot (by pressing START on the bot's page [see figure]
4. In the telegram client the participant sends a message containing the logincode.
5. This message is received by the Telegram service and relayed to the DiET server
6. The DiET server receives the message from the Telegram service. The server saves the message to /tg/telegrambotio.txt
7. The DiET server checks that the message is not sent from a telegram ID that is blocked (i.e. a telegram ID that is contained in /tg/blockedids.txt)
8. The server creates a new logfile in the /tg/ directory. The filename of the logfile is the Telegram ID. So for example if the telegram ID of the client is 1234567890, then the DiET server would create a logfile called /tg/1234567890.log
9. The server then saves the incoming message to the logfile created in the previous step.
10. The server checks with the currently running experimental template whether the logincode is valid. There are two types of logincode – generic and specific.
    The generic code is a single logincode that the experimenter gives to all participants.
    The partner-specific codes are given to particular participants.
11. If the logincode is invalid the server sends the client a message requesting that the participant sends a valid login code, and then the server goes to (5.)
12. If the logincode is valid, the experimental template creates a set of credentials for the client: It creates a unique Participant ID and a unique Username.
    IMPORTANT: Once a participant has logged in successfully using a logincode and has received a valid set of credentials (i.e. a participant ID and username), this ID and username will always be associated with that participant – unless you delete the logfile!
13. The new Participant ID and Username are stored in the participant's logfile in the /tg/directory
14. When the client subsequently sends a message to the server, the server can associate the Telegram ID of the incoming message with the correct set of credentials)

Logging users back in

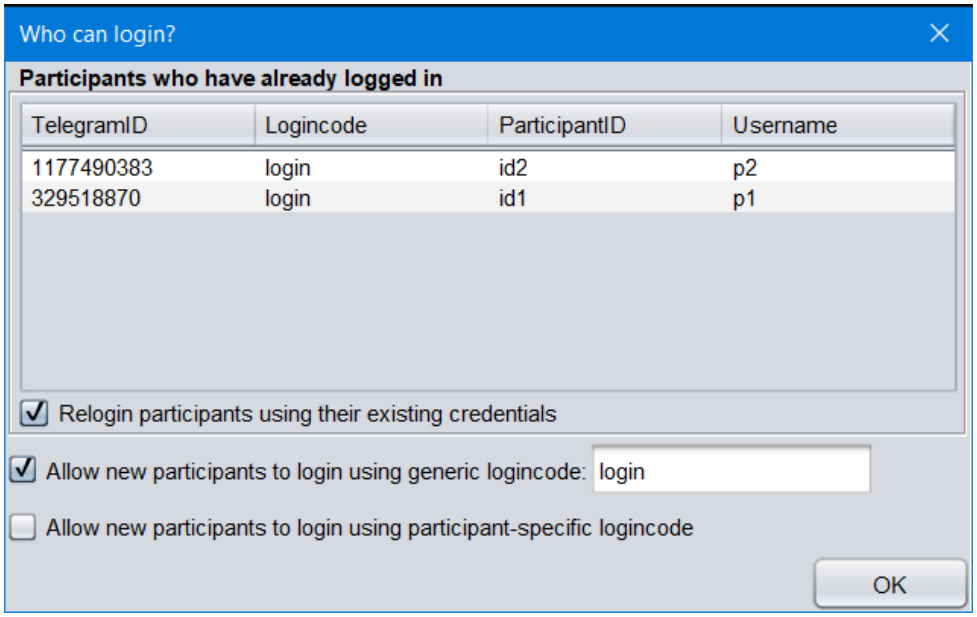


Figure 8 The login options allow the experimenter to specify whether previously logged in participants should be logged back in using their existing credentials

1. The experimenter starts the chattool server
2. The experimenter starts an experimental template
3. Before starting the template, the server looks through the /tg/ directory for logfiles. It then looks through each logfile to see if that logfile contains a participant ID, username and logincode.
4. The server then displays a list of those participants that have already logged in, and asks the experimenter [See figure X] whether these previously logged in participants should be logged in again.
5. If the experimenter selected that previously logged in participants should be relogged in AND if there are participants that have already successfully logged in, then the chattool attempts to login those participants back in, using the credentials that were stored in their logfile. Note that if you are using a different experimental template than the one that was used when the user's credentials were created, there is no guarantee that the user will be able to login. For example, suppose a participant with Telegram ID 1234567890 first logged in to the (fictional) template "`Telegram – Dyadic Task oriented dialogue 20 mins`", which assigned the participant the ID "ID01" and username "P01". Then, the next day, the experimenter starts the server and runs a new (fictional) template called "`Telegram – Second-language-learning`" and selects that the user. It could be that this new experimental template only allows participants with ID "Instructor1" and "Learner" to participate and the participant will be blocked.
6. If the experimenter didn't select the option that previously logged in participants should be logged back in, then they will be prevented from logging in later.
7. Many experimental templates perform a script when a participant is logged back in – e.g. it could be that the script (re)sends the current image stimulus in a psycholinguistic experiment, or sends the participant the current score in the game. What happens depends entirely on the experimental template!
8. The server then waits for new participants to login

How to use a different/new bot

You might want to connect the chattool to a new/different bot (e.g. if you want to run multiple different experiments in parallel or on different occasions). To do this,

1. Inside the telegram app on a mobile phone create a new Telegram Bot. Make a note of the Bot name and token.
2. Close down the chattool
3. Make a backup copy of the telegram directory
4. Edit /tg/botname.txt so that the firstline contains the bot name and the second line contains the bot token.
5. Optional: You might also want to delete all the ********.log files. If you don't delete the logfiles, when you start the experimental template, it will
6. Start the chattool

Messages sent by the participants while the server isn't running

If for any reasons the chat server is not running (i.e. if there is no Template [see X] that is actively running), then all messages sent by the participants are enqueued on Telegram's servers, and are sent to the DiET server as soon as the DiET server connects to Telegram.

How to change a participant's credentials (participant ID / username)

1. Switch off the server
2. Backup and then delete the user's logfile in the /tg/ directory
3. Start the server

Withdraw a user's access

If there is a participant who continues to try to connect, even if they shouldn't you can simply add their telegram ID to the /tg/blockedids.txt file and restart the server.

Change a user's participant ID / Username

Probably the easiest way to do this is to delete the user's log file, restart the server, and get them to log in again.

Data saved.

When running an experiment, the chattool automatically creates a new subdirectory of /data/saved/experimental data where all data that is associated with the experiment is saved.

In addition, when running Telegram experiments, all Telegram-specific information is saved to various files in the /tg/ subdirectory. This information is useful, e.g. if you want to reuse participant credentials on multiple experiments.

Telegram data

/tg/telegramio.txt
This file contains a log of every Telegram message that was sent and received by the server. Each row consists of three fields that are separated with the "|" character:

- A timestamp of when the message was saved. The timestamp is saved in the unix epoch time format (https://www.epochconverter.com/)
- Information about whether the message was sent by a clients or by the server:
  FROM means the message was sent FROM the server to the clients
  TO means the message was sent TO the server by the clients
- The message. This text saves the raw message format that is used by the Telegram service.

This contains low-level information. Note that this file will contain ALL information from every single telegram experiment that was run using the chattool.

/tg/********.log

The folder contains a log file for each telegram client that has sent a message to the server. Whenever the server receives a message from a telegram client, it creates a new file where the filename is the telegram ID of the client, with a ".log" suffix. All telegram messages sent to and received from the client are also saved in this log file. When the client successfully logs into the server (i.e. by using the correct logincode), the server saves the ParticipantID, Username and logincode to the logfile. When the server restarts and asks if the user should be logged back in, the chattool looks up a user's credentials (participantID, username, logincode) in this file.

IMPORTANT: Once a participant has logged in successfully using a logincode and has received a valid set of credentials (i.e. a participant ID and username), this ID and username will always be associated with that participant – unless you delete the logfile.

Experiment data

When an experiment is started, it automatically creates a new subdirectory of
/data/saved experimental data/

This directory contains all data that is associated with that specific experiment.

turns.txt

This is a human-readable transcript of the interaction. It is a CSV file, formatted in UTF8, with each field separated by the "¦" character. Each row is a separate turn. It records the turns produced by the participants, as well as the instructions sent by the server to the participants. The fields are:

ExperimentID

This is the name of the experimental script, as displayed in the interface.

ServerTimestampOfSavingToFile

The server saves the timestamp (in unix epoch format) of when the message was saved to the data file on the server.

GroupID

This is the (sub-)group of the participant. [See section] for how to assign participants to different groups

Turntype

This specifies what the type of the turn is, e.g. whether it is a turn that was produced normally by a participant or whether the turn was intercepted (blocked/modified/delayed) by the server, or whether the turn was an instruction by the server that was sent to a client.

SernderID

The participant ID of the sender (this is unique for each participant)

SenderUsername

The username of the sender (this is unique for each participant)

ApparentSender

This field is only relevant in experiments where the identity of participants is spoofed. This field shows who the message appears to be sent from.

Text

The text of the message sent by the participants / sent by the server to the participants

Recipient(s)

A list of the usernames of the participants who received the messages.

NoOfDocumentDeletes

You can ignore this – this field is only used in the built-in Turn By Turn Interface. In the Turn By Turn Interface the interface captures how many characters were deleted while the participant formulated a turn. This finegrained information isn't recorded by the Telegram app (if your experiment needs this information, you should use the built-in TBT interface).

NoOfKeypressDeletes

You can ignore this – this field is only used in the built-in Turn By Turn Interface. In the Turn By Turn Interface the interface captures how many times the participant pressed the delete key while formulating the turn. This finegrained information isn't recorded by the Telegram app (if your experiment needs this information, you should use the built-in TBT interface).

ClientTimestampONSET

You can ignore this field – this field is only used in the built-in interface. In the built-in Turn By Turn interface, the interface captures the timestamp of the first keypress of each turn. However this information is not registered by the Telegram app. This field is set to the same value as ClientTimestamENTER

ClientTimestampENTER

This field shows the timestamp when the Telegram service registered the message sent by the participant.

ServerTimestampOfReceiptAndOrSending

This field records the exact timestamp when the message was received by the server. It should be a few milliseconds before the field ServerTimestampOfSavingToFile

TextAsformulatedTIMING

You can ignore this – this field is only used in the built-in Turn By Turn interface. This field represents how the turn was incrementally constructed. This fine-grained information isn't recorded by the Telegram app.

TextAsFormulatedLOGSPACE

You can ignore this – this field is only used in the built-in Turn By Turn interface. This field represents how the turn was incrementally constructed. This fine-grained information isn't recorded by the Telegram app.

Additional data

This field contains any additional data that was recorded. Sometimes telegram produces additional status messages – these are saved in this field. Also, some experimental scripts save additional data (e.g. the maze game script also saves game-related information such as the location of the position marker, and the current maze number). This data field saves data formatted as attribute-value pairs inside square brackets, i.e.

[name1, value] [name2, value2] [name3, value3] [name4, value4] …….. [nameN, valueN]

One of the attribute-value pairs is the raw Telegram message, e.g

[telegramrawdata, Update{updateId=941023011, message=Message{messageId=169, from=User{id=329518870, firstName='G', isBot=false, lastName='null', userName='null', languageCode='en', canJoinGroups=null, canReadAllGroupMessages=null, supportInlineQueries=null}, date=1597926857…]

turnasattribvals.txt

This file is essentially the same as “turns.txt” (see above). The only difference is that this file automatically creates a new field (i.e. a new column) for each of the attribute-value pairs in the Additional data column.

N.B. The trade-off for prettifying the output in this way is that it isn’t possible to place the column headers (i.e. the attributes) in the first row, as is typically done with CSV files (since an experiment might always generate a new type of attribute-value pair).

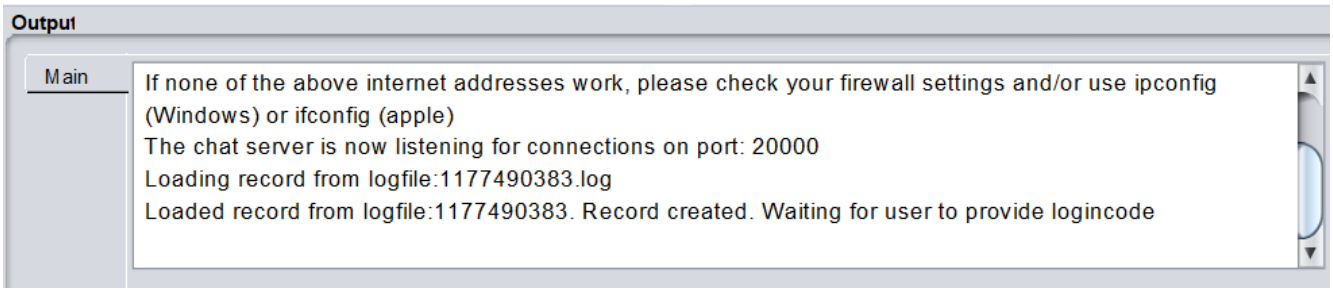
The column headers are saved in a separate file - **turnasattribvals.txt_HEADER.txt**

turnasattribvals.txt_HEADER.txt

See previous explanation for **turnasattribvals.txt**

window_Main.txt

All the status updates that are displayed in the output window (see figure below) are saved to file.



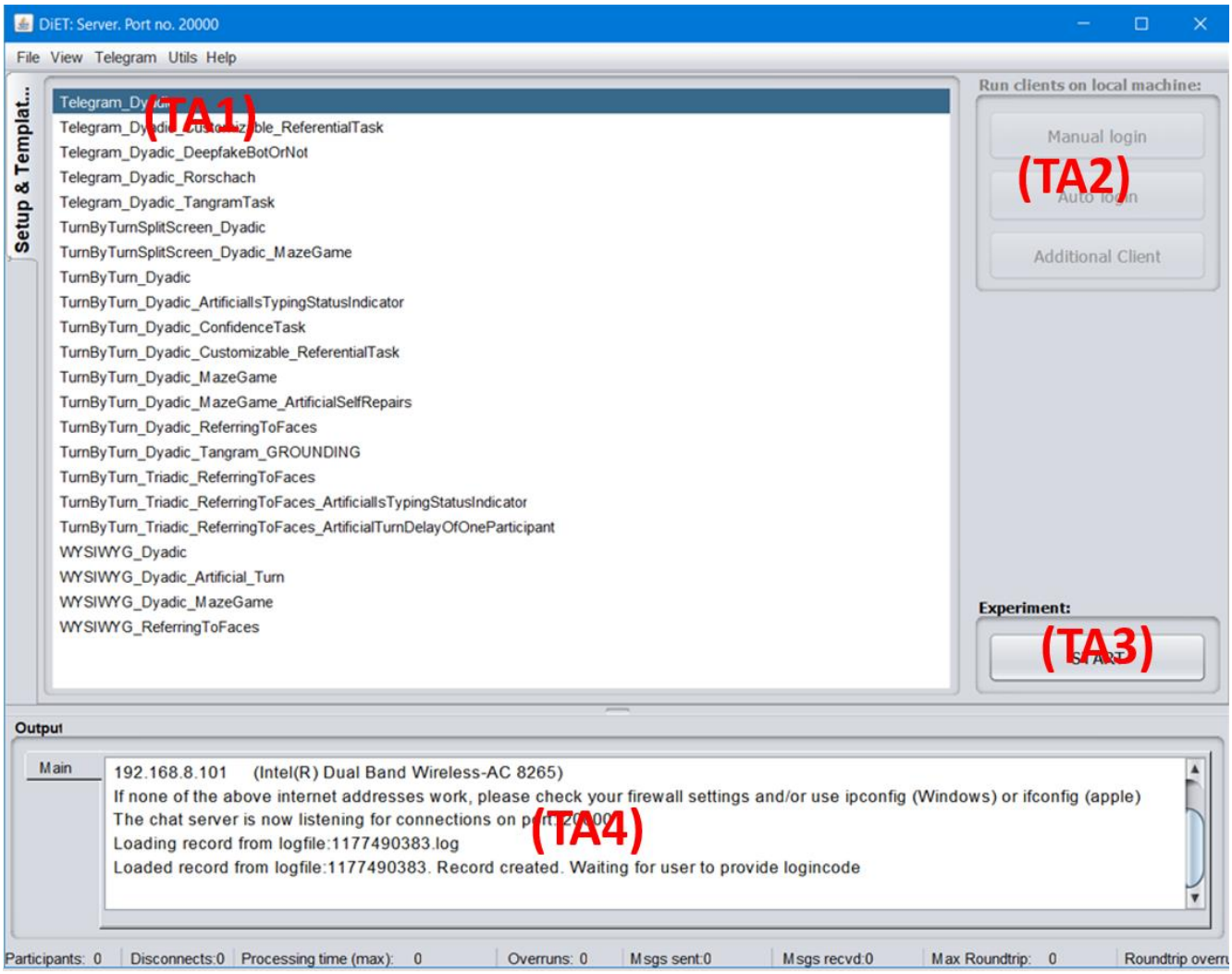
turnsserialized.obj

You don’t need this file unless you are programming the chattool. This file contains “serialized” versions of the participants’ turns (see the programming manual)

messages.obj

You don’t need this file unless you are programming the chattool and using the built-in client. This file contains “serialized” versions of the messages sent between the server and the built-in client.

Telegram – Server – Script Launcher



This is the main window that loads when the server starts

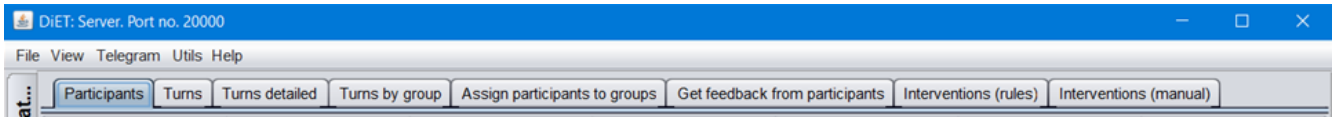
(TA1) This is the list of experimental templates that can be launched. The chattool contains many different experimental templates. When you program your own intervention, it will appear here. N.B. Because we are constantly creating and adding new interventions, this list might not be identical in the current version.

(TA2) are controls for testing experiments that use the built-in interfaces. These options are greyed out for Telegram experiments.

(TA3) This option initializes the selected script and connects to the Telegram services to wait for connections / messages from the participants.

(TA4) This window displays information about participants connecting to the chattool. The contents of this window are saved to **windowMain.txt** in the experiment directory.

While running a Telegram experiment



When an experiment is started it loads up a new set of windows which can be accessed by selecting their respective tab. The eight different windows are:

Participants

This window displays which participants are currently logged in, together with the participants' credentials.

Turns

This window displays the turns produced by participants.

Turns detailed

This window displays detailed information about the turns

Turns by group

The chattool allows multiple parallel conversations by different (sub-)groups. This window shows the conversation of each group in a separate window, making it much easier to keep track of who said what to whom.

Assign participants to groups

This window allows the experimenter to dynamically assign participants to different groups – i.e. to assign participants to different conversational partners

Assign participants to groups

This window allows the experimenter to send a "poll" to all participants.

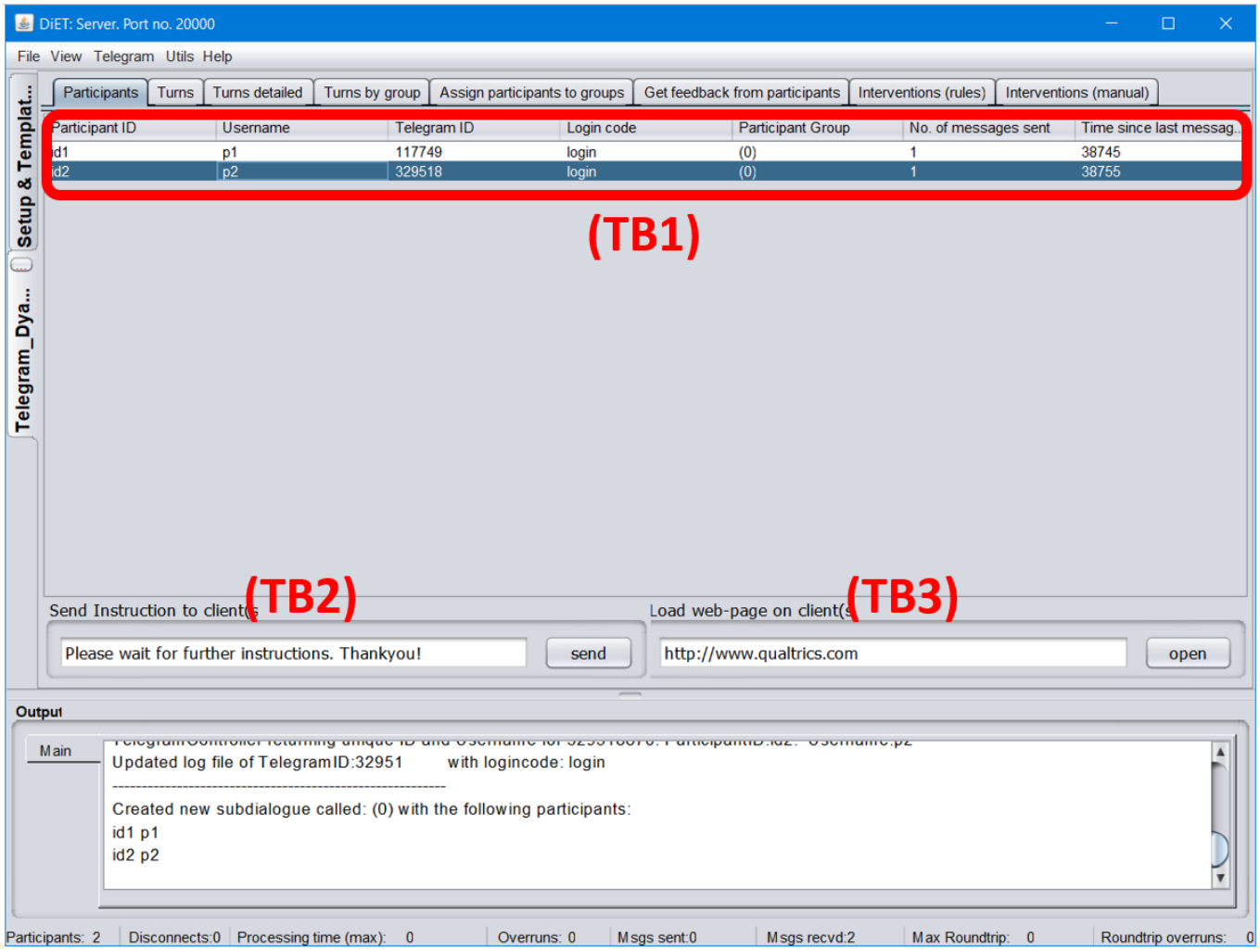
Interventions (rules)

This window allows the experimenter to specify rules for experimentally manipulating the interaction, i.e. specify rules for blocking, transforming or delaying turns.

Interventions (manual)

This window allows the experimenter to directly manipulate participants' turns.

Telegram – Server GUI – **Participants**



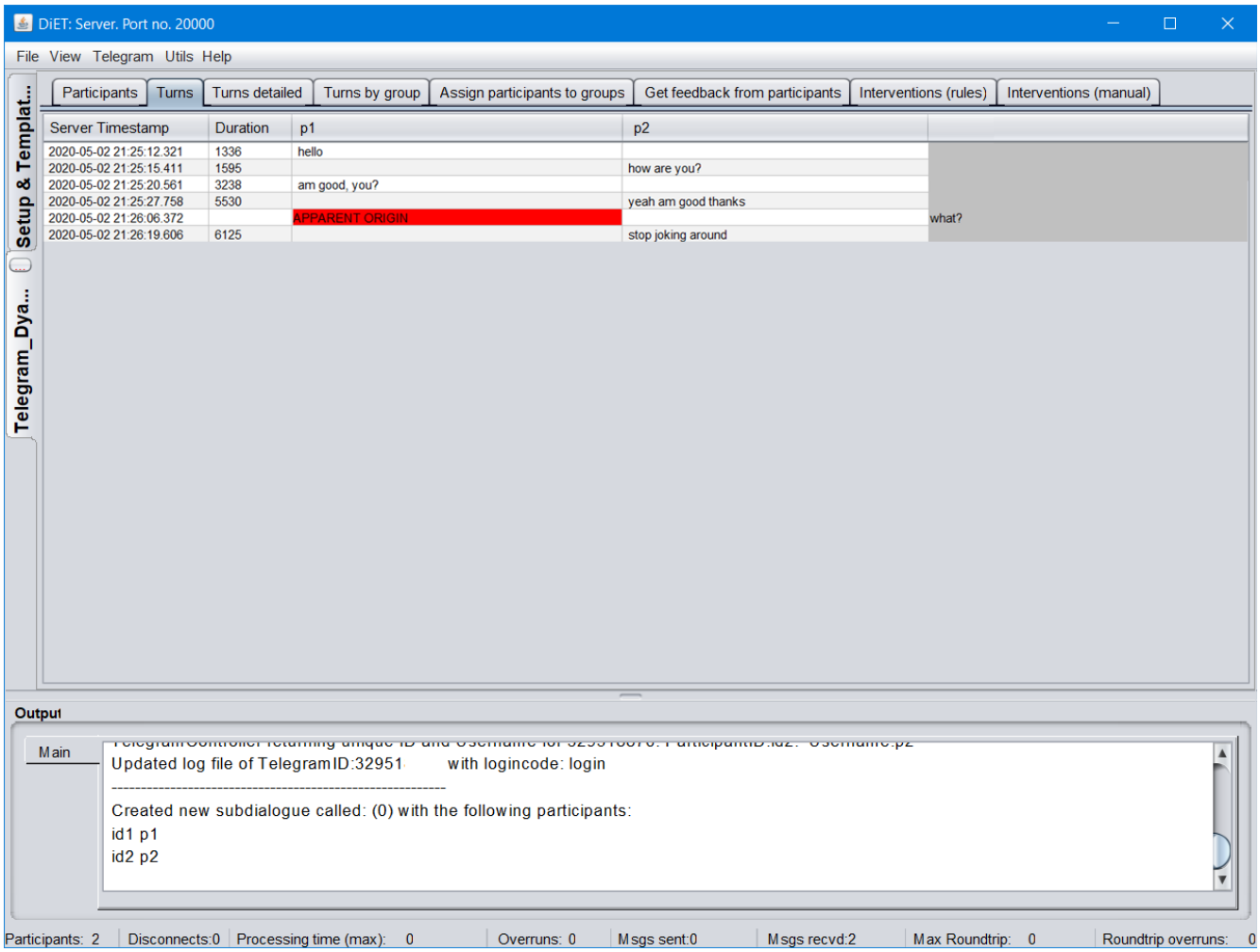
(TB1) This table contains the credentials of all participants who are logged into the chattool, as well as information about which participant group the participant is assigned to, how many messages the participant has sent during the experiment and the number of seconds since the last message.

(TB2) This is an option for the experimenter to send instructions to participants. When the experimenter presses SEND, it sends the message (in this case "Please wait for further instructions") to the participants. The experimenter can choose who to send the message(s) to by selecting the relevant participants in the table (TB1).

(TB3) This is an option to send a link to a webpage to the participants. When the experimenter presses SEND, it sends the URL (in this case http://www.qualtrics.com) to the participants. The experimenter can choose who to send the message(s) to by selecting the relevant participants in the table (TB1).

IMPORTANT! This screen is slightly different from the "Participants" screen when using the built-in interfaces (i.e non-Telegram). For example in the table (TB1) "Telegram ID" and "login code" are specific to Telegram experiments. Also, some options for controlling the interface are not possible when using the Telegram interface.

Telegram – Server GUI - Turns



This interface displays the turns that were typed by the participants.

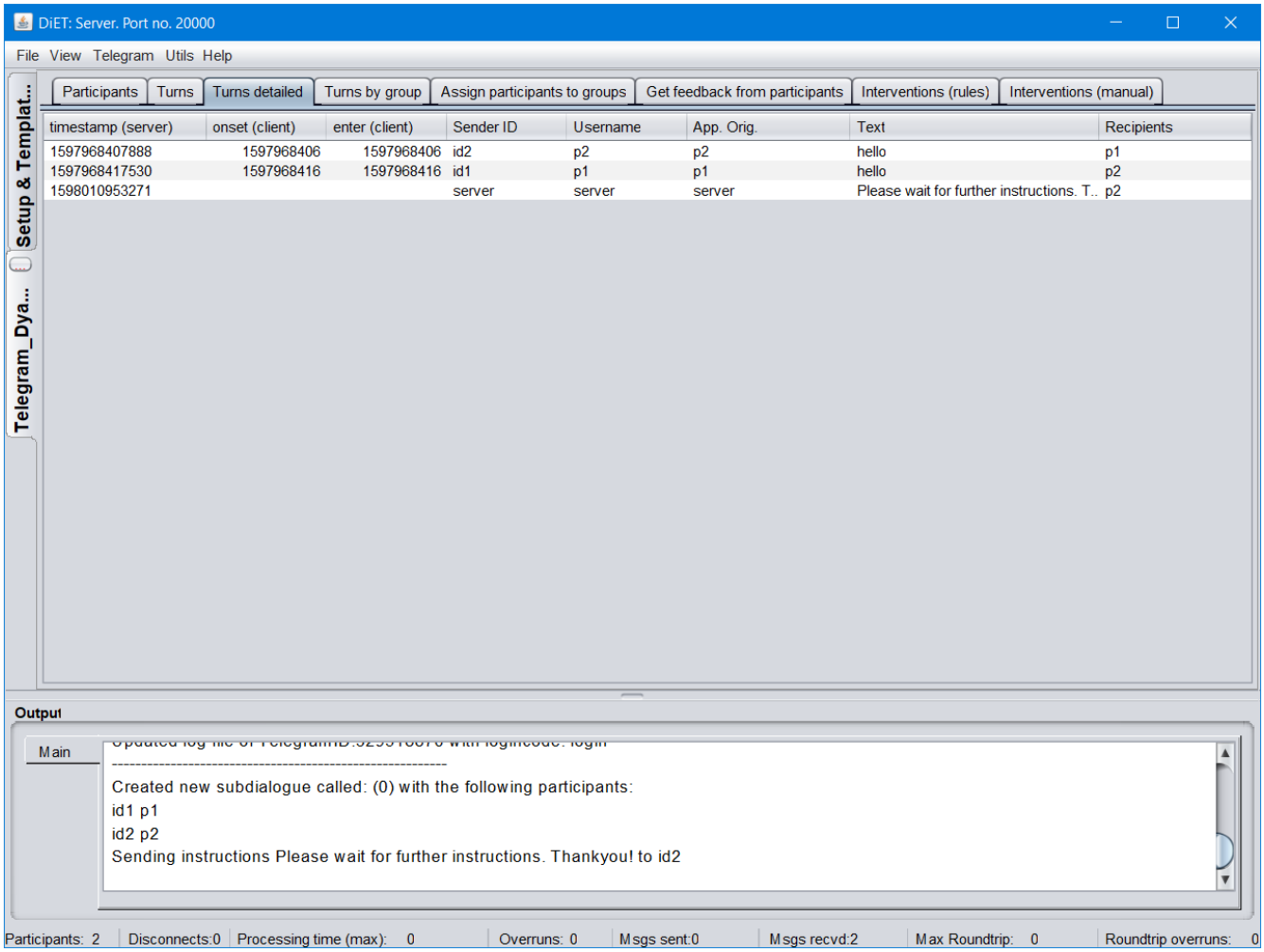
"**Server Timestamp**" shows the time when the message was received by the server. The messages are ordered in the order in which they were received by the server.

"**Duration**" displays how long it took the participant to formulate the turn, in milliseconds (this is the duration from the first keypress till when they pressed enter to send the message).

The subsequent columns are the turns produced by each participant. The turns produced by each participant are displayed in separate columns. This makes it easier to follow who typed what. This table displays a sequence of interaction between two participants, with usernames p1 and p2.

Notice that there is also a column for the "server". This allows you to see the artificial "spoof turns"!. In this example the server sent a spoof turn "what?" that appeared to originate from participant with username p1.

Telegram – Server GUI – Turns detailed



This interface displays a more detailed view of the turns.

Timestamp (server): This is the time when the message was received by the server.

Onset (client)/ Enter (client):
This is the time, as recorded by Telegram when the message was received by the server. Note that the times are recorded in "unix" / epoch time – this is one of the most standard time formats in computers – the number of milliseconds that have elapsed since 1970. It is very straightforward to convert to human-readable dates/times, see e.g. https://www.epochconverter.com/

Sender ID: This is the participant ID of the person who sent the turn

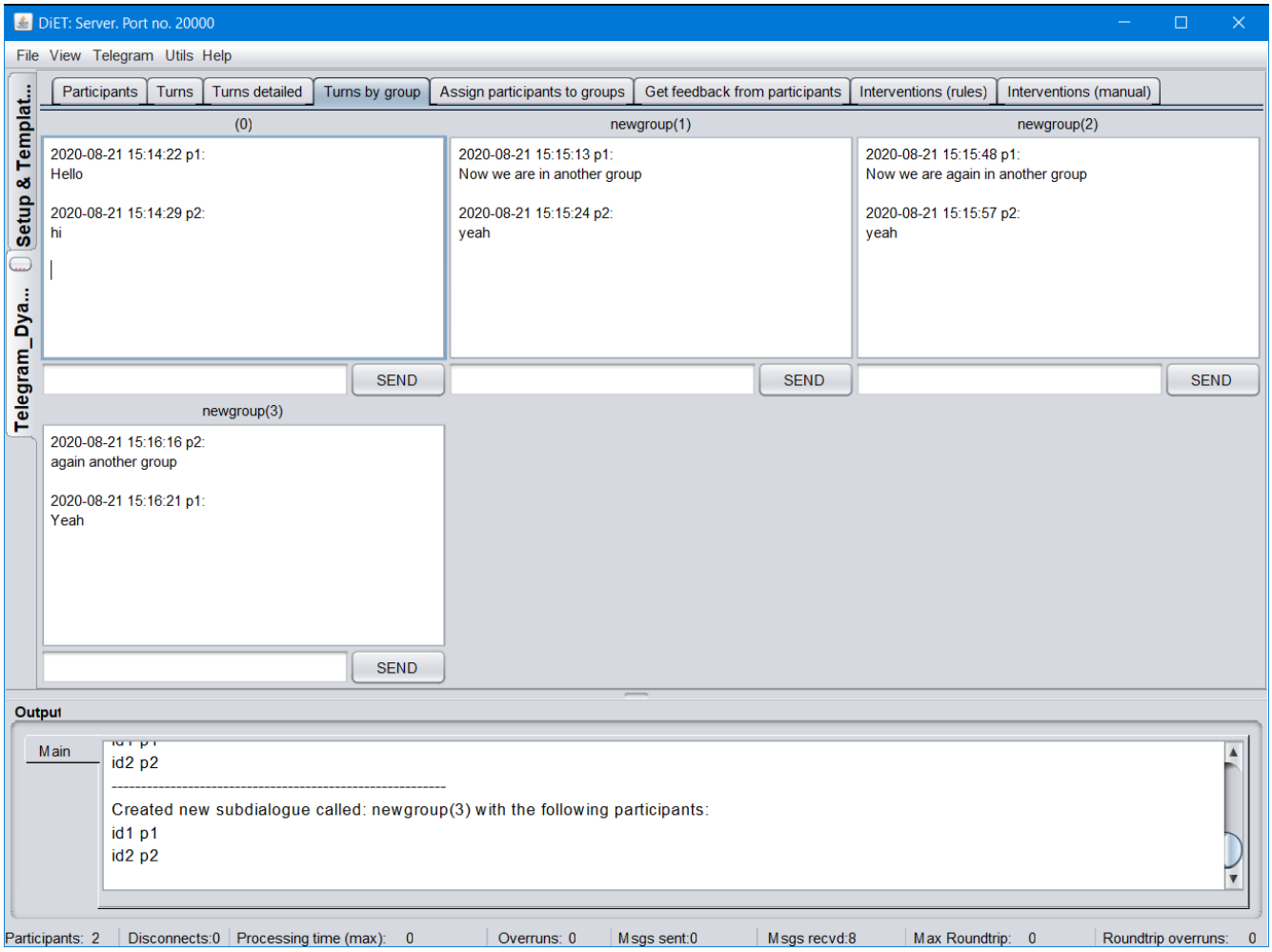
Username: This is the username of the person who sent the turn.

Apparent Origin: This is who the message appears to be sent from. Ordinarily, this is the same as "Username", but is different for fake messages.

Recipients: This shows who received the message.

Note that the last row shows that the server sent instructions to p2 to "Please wait for further instructions…"

Telegram – Server GUI – Turns by group

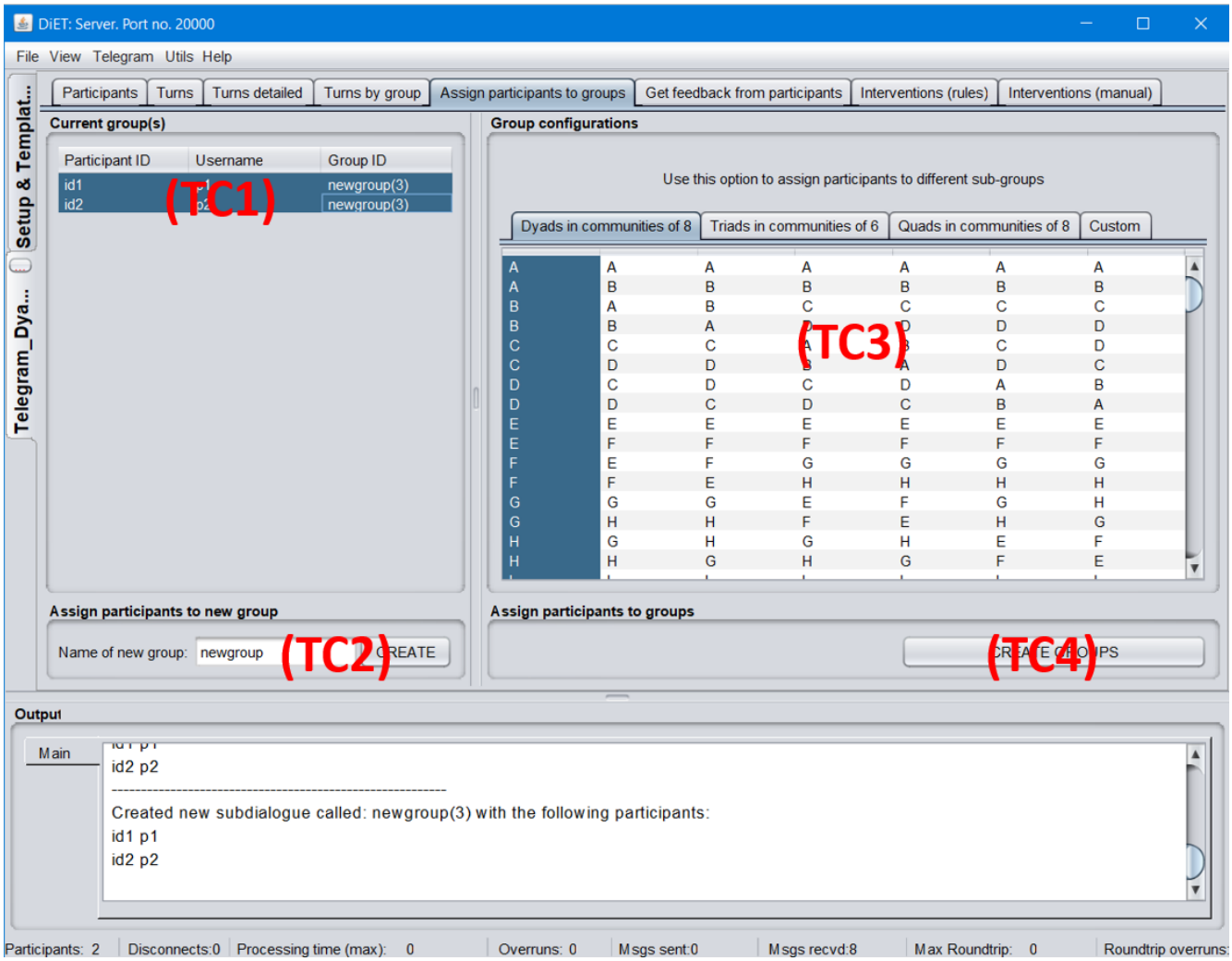


This screen shows the conversations of each group separately. Participants can be assigned to different groups manually by the experimenter [Using the screen “Assign participants to groups”] or by the experimental script. Notice how in this screen there are 4 separate windows showing the conversations of four different groups.

Whenever a new group is created, this screen automatically adds a new window showing the conversation of that group.

Notice also that each group has a textfield and a SEND button. This allows the experimenter to send instructions to specific groups

Telegram – Server GUI – Assign participants to group



This screen allows the experimenter to manually assign participants to new groups. There are two ways of creating new groups:

Assigning a list of participants to one single new group.

(TC1) displays a list of all participants who are logged into the chat tool and which group they are currently part of. To assign participants to a new group, select the participants in table (TC1), type the new group name (in this case the new group name is "newgroup") and press CREATE. This will automatically create the new group.

Simultaneously assigning ALL participants to multiple groups.

Consider an experimental design where you have 100 participants who are paired in 50 dyads. Now imagine you wish to pair the 100 participants in new dyads. It would take 50 steps to do this using the method described above.

Table (TC3) contains a list of group configurations. Notice how the first column contains "A", "A", "B", "B", "C", "C", "D", "D", "E", "E", "F", "F", "G", "G", "H", "H"

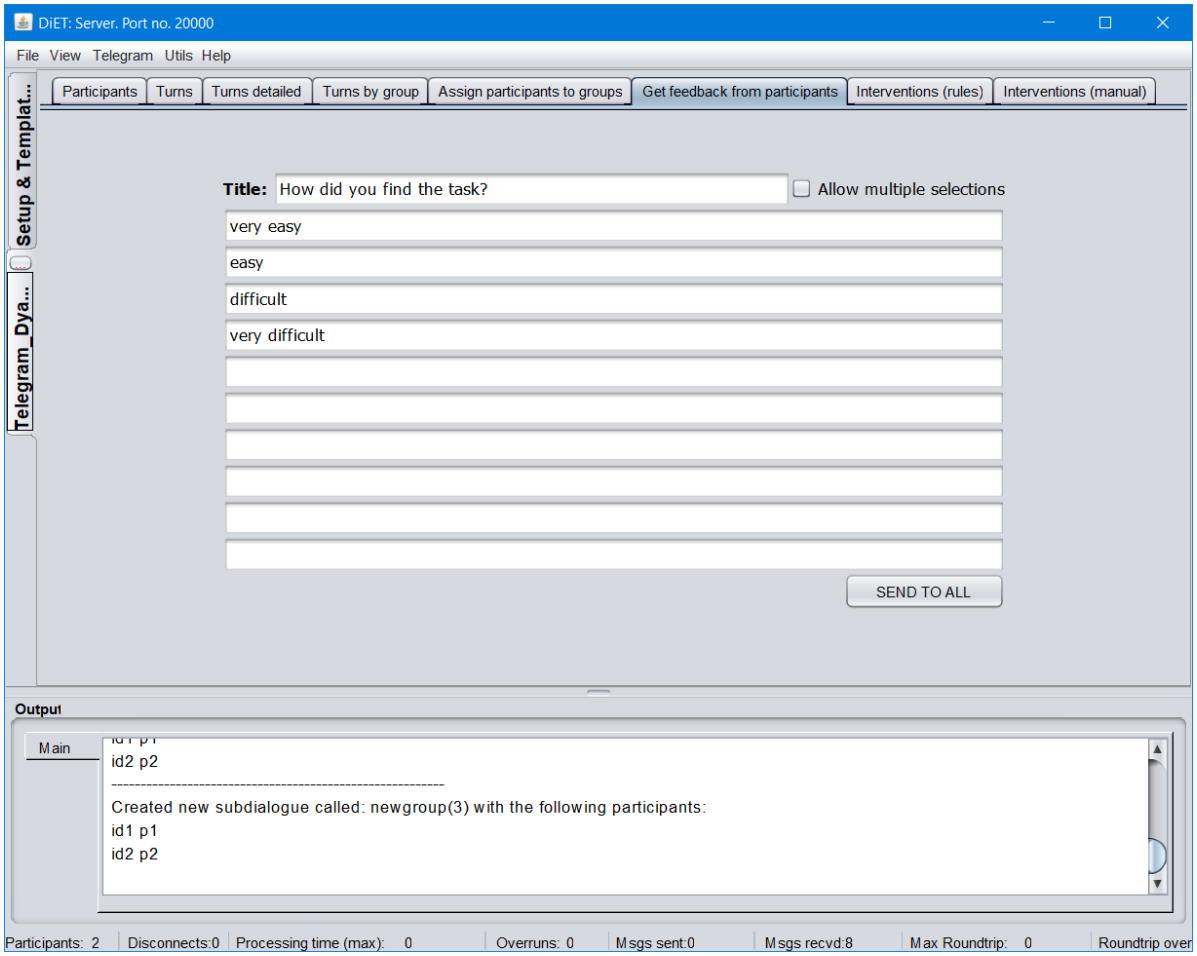
What this means that if the experimenter presses (TC4) “CREATE GROUPS” then the first participant in table (TC1) (i.e. p1) will be assigned to a new group “A”. The second participant in table (TC1) (p2) will also be assigned to group A. In this example there are only two participants (p1 and p2), but if there were more participants - the third participant would be assigned to group “B”. The fourth participant would be assigned to group “B”, etc.

In other words, selecting the first column and pressing “CREATE GROUPS” ends up creating new dyads where the first participant is paired with the second, the third participant is paired with the fourth, the fifth participant is paired with the sixth, etc.

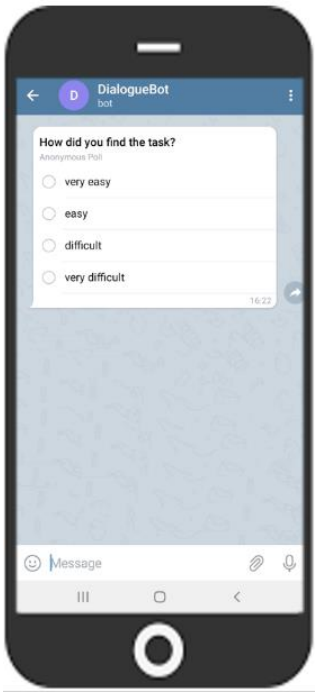
But other pairings are possible! Now suppose you select the second column. Notice how the column contains “A”, “B”, “A”, “B”, “C”, “D”, “C”, “D”, “E”, “F”, “E”, “F” etc.
If you were to select this column and press (TC4) “CREATE GROUPS” then the first participant in the list (TC1) would be paired with the third participant in a group called “A”. The second participant would be paired with the third participant in a group called “B”, etc.

Of course groups can consist of any number of participants. The chattool provides a few configurations for groups of three (triads) and groups of four (quads). If you wish to use your own, custom list of groups, you can select “Custom” and load your own list. See, e.g. …\experimentresources\groupconfigurations\dyads.csv

Telegram – Server GUI – getting input from the user

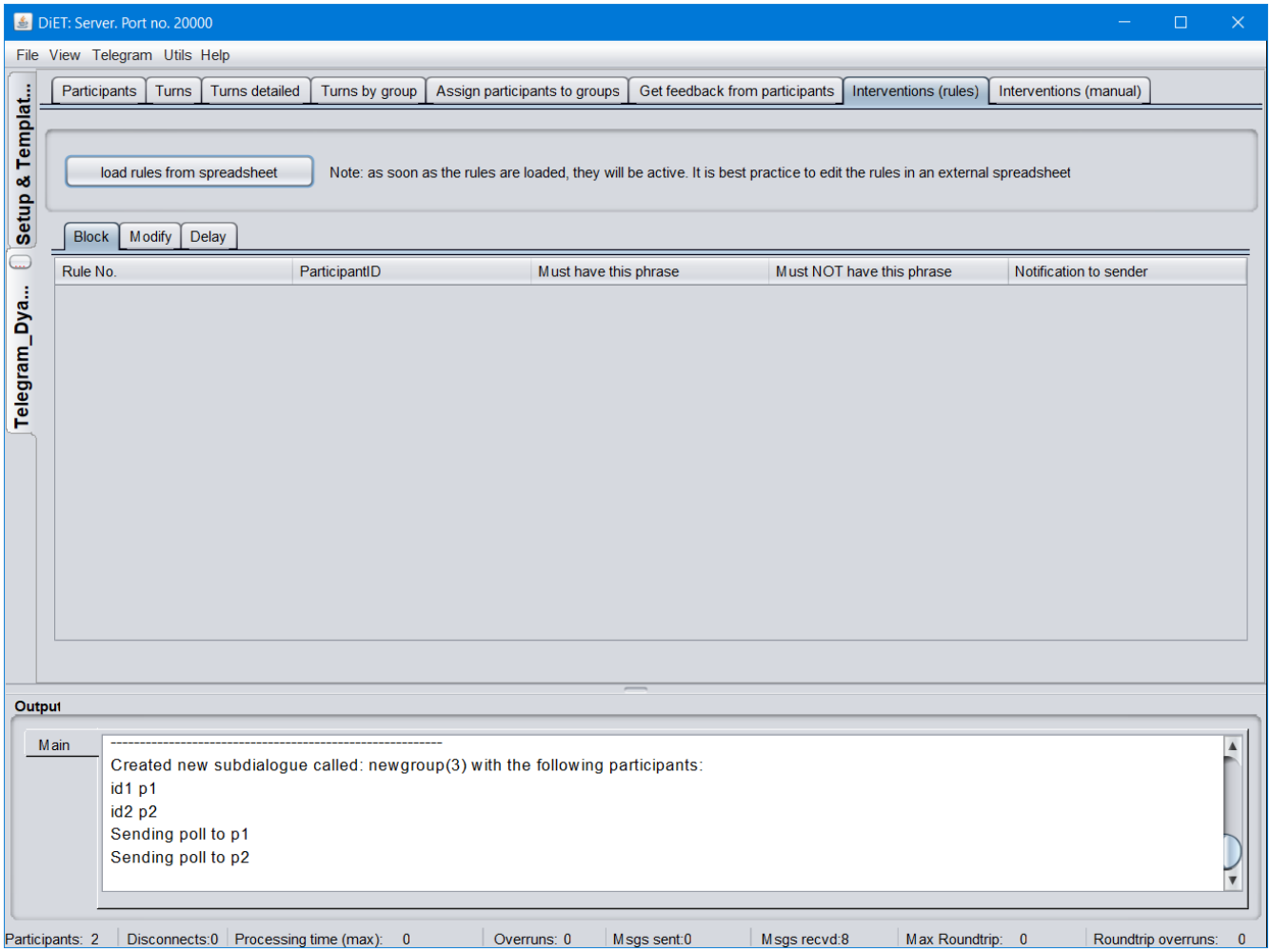


Use this option to create and send a Telegram poll to all the participants in the experiment (see https://telegram.org/blog/polls-2-0-vmq) The poll will be displayed in the Telegram app on the participant's mobile phone.



Note that this is only an option when running a Telegram experiment!

Telegram – Interventions (rules)

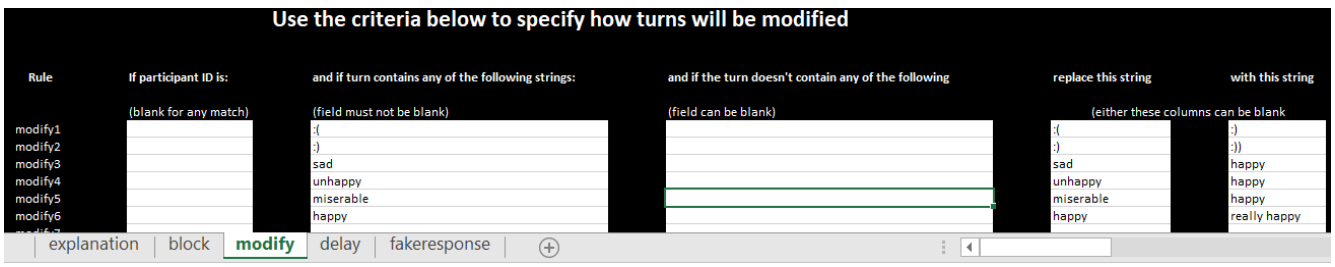


This screen makes it possible to run (simple) experiments that manipulate the content and timing of participants' turns.

This screen allows the experimenter to specify sets of rules for artificially blocking (i.e. "shadowbanning"), modifying and delaying participants' turns, without requiring any programming by the experimenter.

The rules are stored in excel spreadsheets in the directory:
.\experimentresources\interventionsauto

For example, the file ".\experimentresources\interventionsauto\makehappy.xlsx" is included as a "toy" example consisting of a few rules that artificially make a conversation "happier". To see how it works, first load the file ".\experimentresources\interventionsauto\makehappy.xlsx" into a spreadsheet (e.g. excel or LibreOffice) and then select the sheet "modify". This will show the excel template consisting of six rules:



- The first rule converts the emoticon :(to :)
- The second rule converts the emoticon :) to :))
- The third rule converts the word “sad” to “happy”
- The fourth rule converts the word “unhappy” to “happy”
- The fifth rule converts the word “miserable” to “happy”
- The sixth rule converts the word “happy” to “really happy”

If you select “load rules from spreadsheet” in the chattool and select “makehappy.xlsx”, these rules will be displayed in the chattool:

Block | Modify | Delay

Rule No.	ParticipantID	Must have this phrase	Must NOT have this phrase	Phrase to be removed	Phrase to be inserted
modify1		:(		:(	:)
modify2		:)		:)	:))
modify3		sad		sad	happy
modify4		unhappy		unhappy	happy
modify5		miserable		miserable	happy
modify6		happy		happy	really happy

Now if one of the participants sends a message containing one of the target phrases, it will be automatically transformed by the server, e.g. consider this dialogue

Notice how the chattool transforms “I am happy” to “I am really happy”.

The spreadsheet template contains three sheets that allow the specification of rules for blocking, modifying and delaying participants' turns. Each row of each sheet is a separate rule. The chattool looks through the spreadsheet attempting to see if any of the rules match. The first rule that matches is the rule that gets invoked.

Blocking turns.

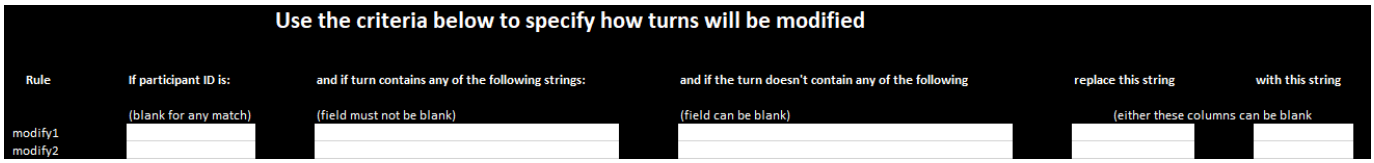
The first column (optional) allows you to specify the ID of the participant whose turns will be blocked. If you want to use this functionality it is probably easiest to use [partner-specific participant IDs].

The second column are the positive criteria for triggering the intervention. If a turn contains the sequence of characters in that column, the intervention is triggered.

The third column (optional) contains the negative criteria. You can prevent a rule from being triggered if it contains sequences of characters in that column

The fourth column (optional) allows you to specify a message that gets sent to the sender. Suppose you wish to build an experiment that tests people's reactions to being censored – you could put a message here such as "You are not allowed to send this message".

Modifying turns



The first column (optional) allows you to specify the ID of the participant whose turns will be blocked. If you want to use this functionality it is probably easiest to use [partner-specific participant IDs].

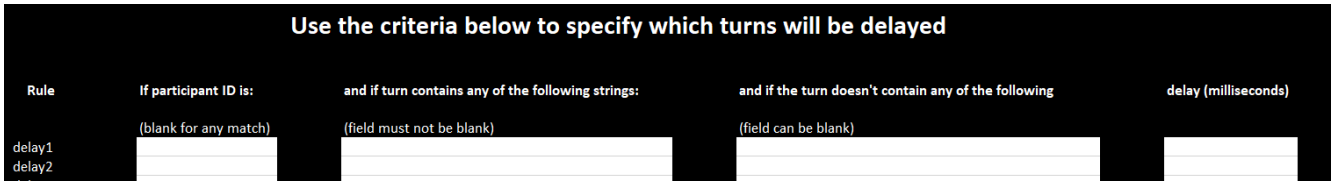
The second column are the positive criteria for triggering the intervention. If a turn contains the sequence of characters in that column, the intervention is triggered.

The third column (optional) contains the negative criteria. You can prevent a rule from being triggered if it contains sequences of characters in that column

The fourth column contains the sequence of words that will be replaced. Typically the entry here is the same as the entry in the second column

The fifth column contains the new sequence of words that will replace the sequence of words described in the fourth column

Delaying turns



The first column (optional) allows you to specify the ID of the participant whose turns will be blocked. If you want to use this functionality it is probably easiest to use [partner-specific participant IDs].

The second column are the positive criteria for triggering the intervention. If a turn contains the sequence of characters in that column, the intervention is triggered.

The third column (optional) contains the negative criteria. You can prevent a rule from being triggered if it contains sequences of characters in that column

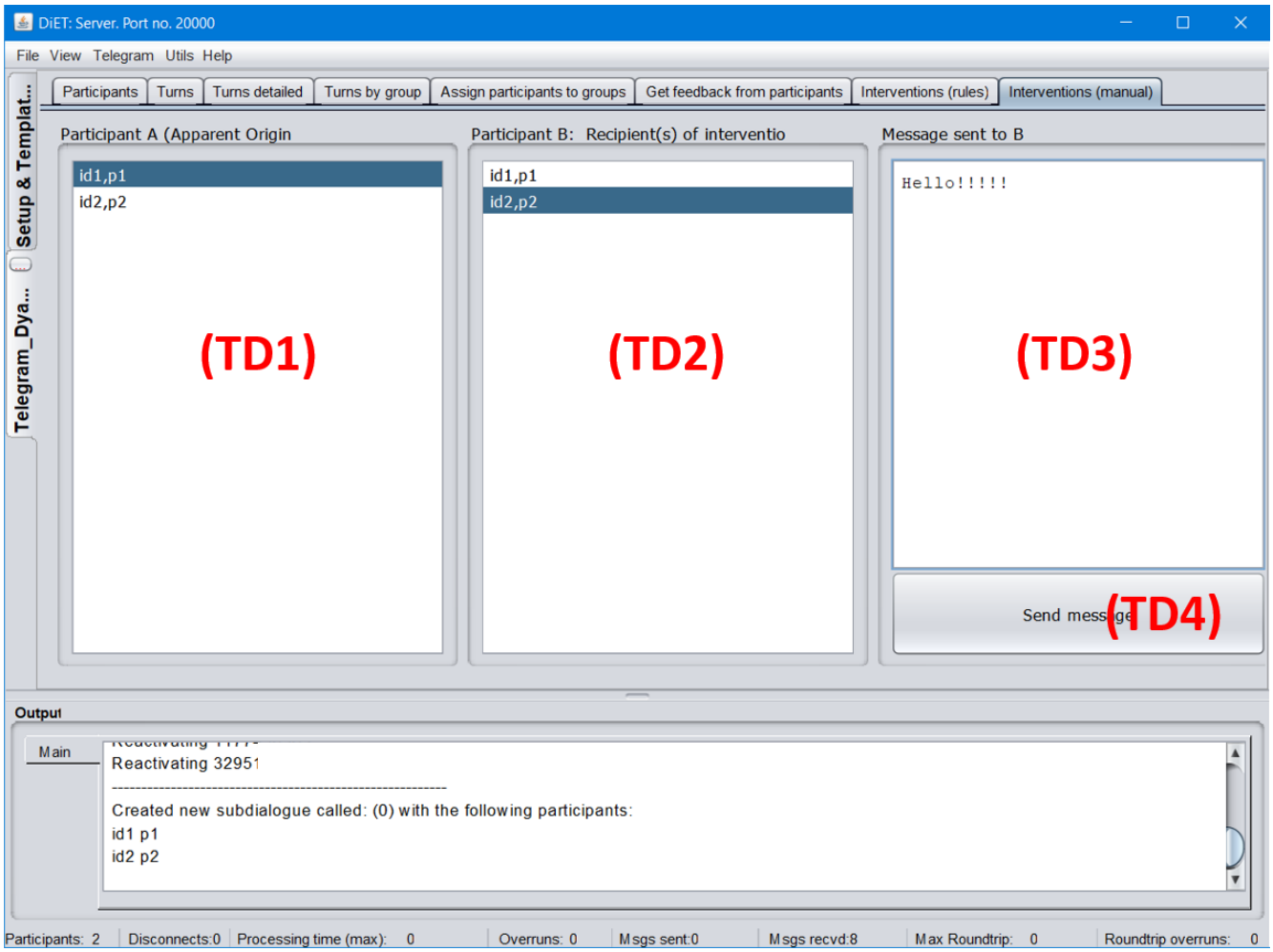
The fourth column (optional) contains the duration in milliseconds that the turn will be delayed by.

Other manipulations

It is possible to write more sophisticated scripting tools for generating interventions – we are currently programming another sheet that allows you to specify when to insert entirely fake turns.

If you have any suggestions / desired manipulations – do let us know!!

Telegram GUI – Interventions (manual)



This screen allows experimenters to manually generate artificial turns (i.e. experimental manipulations).

(TD1) is who the fake message appears to originate from (Apparent Origin)

(TD2) is who receives the message

(TD3) is the text of the message

(TD4) is the button to send the message (in this case “Hello!!!!!”)

Telegram – Tutorial: Creating and running your own, custom referential task

This tutorial assumes you have already completed the Telegram Quick Start guide.

The Telegram service allows bots to send images (similarly to whatsapp) directly to the Telegram chat window. The chattool uses this service to send experimental stimuli to participants. The chattool contains a template for scripting sequences of stimuli. This template also allows the experimenter to specify which options are presented to the participant(s).

Note that when using Telegram to send stimuli to participants, because everything is controlled from the chattool program, the timing of the presentation of the stimuli is not accurate to the millisecond. When the server sends stimuli to the participants they first have to be sent via the Telegram service. This can take 1-2 seconds. If you want fine-grained, millisecond control over the presentation of stimuli, you should use the built-in interfaces (Turn-By-Turn or What You See is What You Get).

Running the tangram task

1. First, start the server
2. Select the template "Telegram_Dyadic_TangramTask"
3. If you have already logged in two Telegram clients in the "Quick start" section, select the option "Relogin participants". Otherwise select the option "allow new participant to login" and then connect two participant mobile phones to the Bot as described in the "Quick start" guide
4. Select "START"
5. Select "keep order"
6. Select the default game duration of 60 seconds.
7. Select the default stimuli presentation time of 60 seconds.
8. Select the option to have buttons underneath the stimuli on the clients
9. Select the option to keep the stimuli in the telegram window after each trial

This should load the following interface on the server:

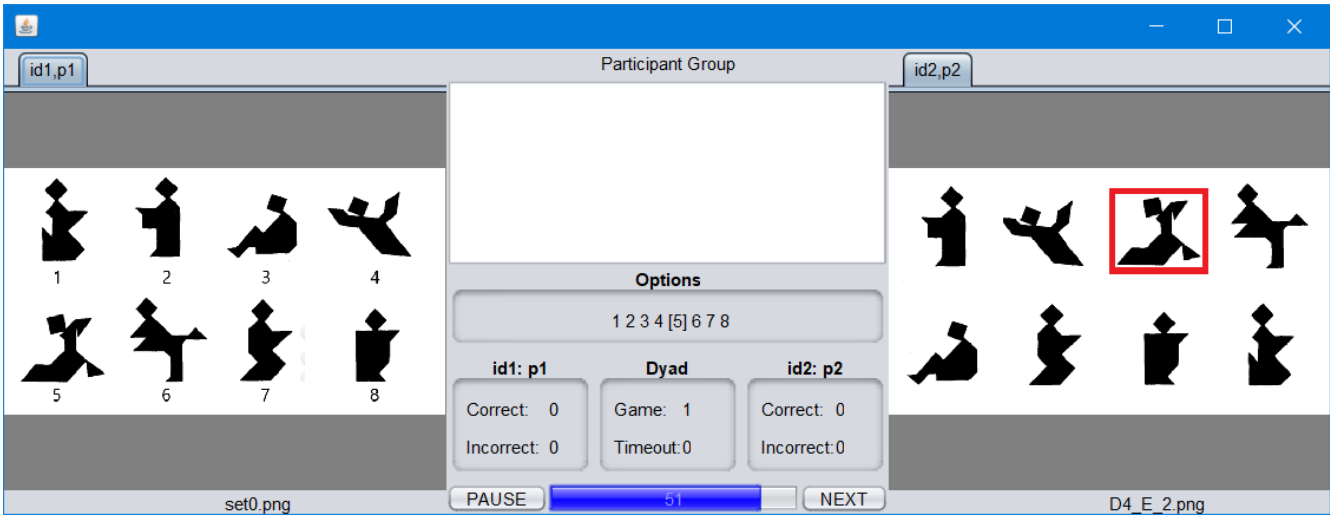
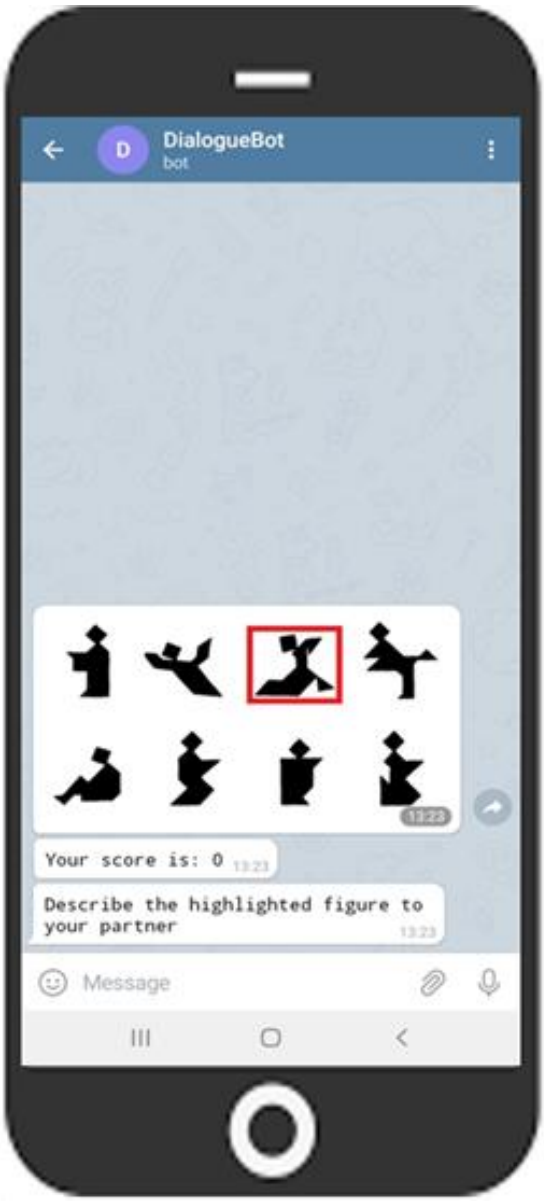


Figure 9 Console window that is used to display and control the sequence of stimuli presented to participants

[The figure above] shows the console that is used to control / view the presentation of stimuli in dyadic experiments. The screen on the left shows the stimuli of participant with participant ID “id1” and username “p2”. The screen on the right shows the stimuli of participant with participant ID “id2” and username “p2”. Both participants see the same six tangram figures, however they are ordered differently. Also, the participant “id1, p1” has numbers underneath the tangrams, while the participant with id “id2” and username”p2” has one tangram with a red square around it. This is because in this round id2,p2 is the Director and id1, p1 is the Matcher (this is specified in the configuration file – [see below]). Notice how the target tangram figure (the one with the red square round it) is number 5 in the matcher’s stimuli set. This information is also available in the middle of the console – the panel for **Options** which shows “1 2 3 4 [5] 6 7 8” means that there are these eight options, and the option enclosed in a square bracket is the correct option.

If you look on the participants’ screens, you can see that the stimuli are displayed in the chat window. Also, underneath the stimuli on the Matcher’s screen are the eight options that the participant can press to select the correct tangram figure.



In order to see how this is scripted, open the file
/experimentresources/stimuli/tangramset01/tangramsequence.txt
This is the script of the experiment you have just started.

P1 IMAGE	P2 IMAGE	MATCHER:1=P1,2=P2,B=BOTH	OPTIONS	CORRECT ANSWER	P1TEXT	P2TEXT
set0.png	D4_E_2.png	1	1,2,3,4,5,6,7,8	5	Identify which shape your partner is describing	Describe the highlighted figure to your partner
D6_F_4.png	set4.png	2	1,2,3,4,5,6,7,8	4	Describe the highlighted figure to your partner	Identify which shape your partner is describing
D0_A_0.png	set0.png	2	1,2,3,4,5,6,7,8	1	Describe the highlighted figure to your partner	Identify which shape your partner is describing
set1.png	D0_B_1.png	1	1,2,3,4,5,6,7,8	5	Identify which shape your partner is describing	Describe the highlighted figure to your partner
D0_C_2.png	set2.png	2	1,2,3,4,5,6,7,8	1	Describe the highlighted figure to your partner	Identify which shape your partner is describing
D0_A_0.png	set0.png	2	1,2,3,4,5,6,7,8	1	Describe the highlighted figure to your partner	Identify which shape your partner is describing
set1.png	D0_B_1.png	1	1,2,3,4,5,6,7,8	5	Identify which shape your partner is describing	Describe the highlighted figure to your partner
D0_C_2.png	set2.png	2	1,2,3,4,5,6,7,8	1	Describe the highlighted figure to your partner	Identify which shape your partner is describing

This script has 7 columns, all separated by a "¦" character.
Each row represents a single round/trial in the referential task.

- The first column contains the name of the jpg or png file that is shown on the first participant's screen
- The second column contains the name of the jpg or png file that is shown on the second participant's screen
- The third column specifies who is the matcher (i.e. who can make selections)
    - A value of "1" means that only the first participant is the matcher
    - A value of "2" means that the second participant is the matcher
    - A value of "B", means that both participants are the matcher (This is useful in referential tasks where both participants have to establish whether they are looking at the same referent or not)
- The third column specifies a list of options, separated by commas.
- The fourth column specifies what the correct answer (i.e. which of the options is the correct one)
- The fifth column shows the text that is sent to the first participant at the start of the round/trial
- The sixth column shows the text that is sent to the second participant at the start of the round/trial.

Part 2: Using the built-in interfaces: Quick start

This section is designed to give you an idea of the chat tool without requiring any programming.
The instructions below are for setting up a simple experiment using one of the pre-existing templates from the library of experiments. If, you want to try it out on a single computer, simply carry out all the steps listed below on the same computer.

Downloading and starting the software:

First, download the latest zip file from github. The latest version (May 3rd 2020 is available at)
https://github.com/dialoguetoolkit/chattool/releases/download/4.6.2/chattool_runme.zip

Unzip the contents of the folder into a subfolder. For the purposes of these instructions, it is assumed that you unzipped the contents into a folder on the desktop: **/desktop/chattool/**
Verify you have the following files

/desktop/chattool/data
/desktop/chattool/experimentresources/
/desktop/chattool/jre-10.0.2/
/desktop/chattool/chattool.jar
/desktop/chattool/readme.txt
/desktop/chattool/start.bat

Windows

Double-click on "start.bat". This should start the program.
Press the "start server" button (See Figure 1 below)

Mac / Linux

1. Double-click on "chattool.jar" . If this works, this means you have java installed.
2. If double-clicking doesn't work, open a Terminal window and navigate to the folder where "chattool.jar" is located, (i.e. **/desktop/chattool/**) then type j**ava -jar "chattool.jar**

If that doesn't work, you need to install java. Follow the instructions on installing the latest version of java runtime for your computer (see e.g. https://support.apple.com/en-us/HT204036), and then retry steps 1 and 2 listed above.

When you see the screen in Figure 1 (below), select "start server"

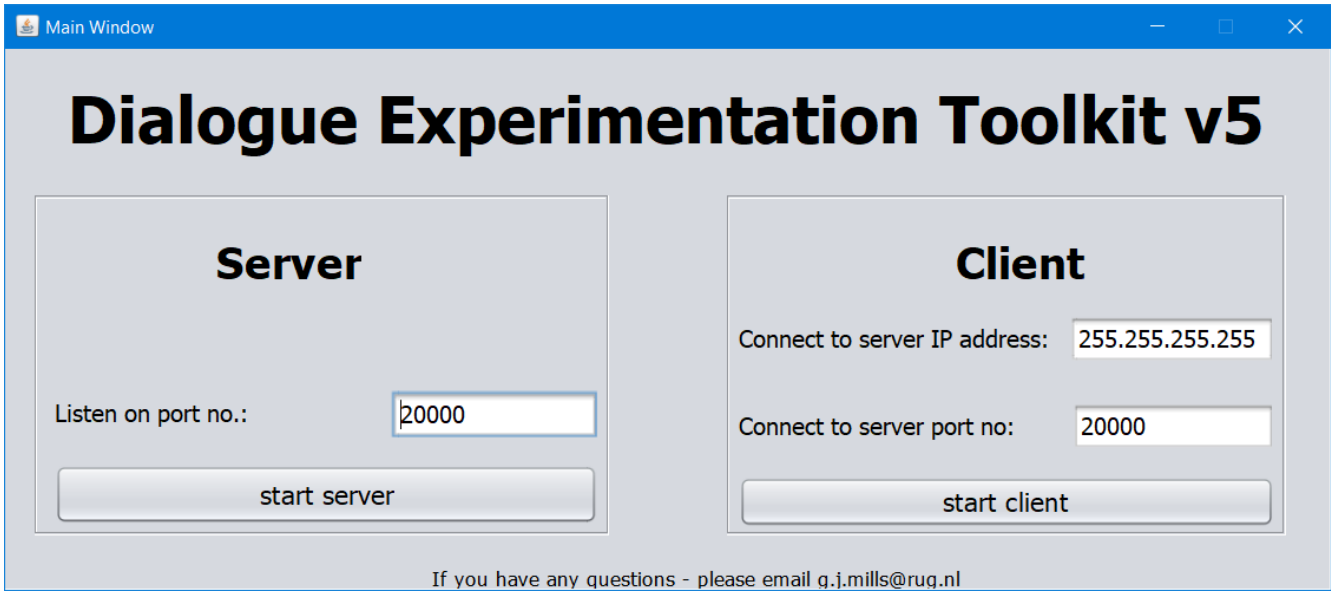


Figure 10 When you start the software without specifying any parameters, it lets you start the server or the client

Sorting out the networking and starting the server and client.

If you are trying this software out for the first time, you can ignore this section, because it is possible to run the server and clients on the same machine. You only need to run server and clients on separate machines when piloting or running the experiment.

Figure 10 shows the first interface that appears when you start the app. Notice that you can choose whether to start the app as a server or as a client. If you start the server you have to specify on what port the server listens for incoming connections from the clients (the default port number is 20000). Conversely, if you start the client, it needs to know what the IP address is of the server – so that it can connect to it. The example in Figure 10 has a field 255.255.255.255. (See e.g. **Error! Reference source not found.**).

When you run the client you will need to enter the IP address of the server. See Figure 18 (6) below, which shows where to look for the server's IP address.

Running an experiment (From the preinstalled library of template)

After you select "start server", the main window should open (see Figure 18, below). In the main window (1), select "Dyadic_Turn_By_turn_Interface" so that it is highlighted, and then select (2) "Demo: Manual login". This option does two things – it initializes the experimental script and then starts two clients. Ordinarily these clients would be running on separate machines, in separate booths. This method allows the experimenter to test interventions locally before running them over a network.

You should see two windows appear. One window appears in the top-left of the screen:

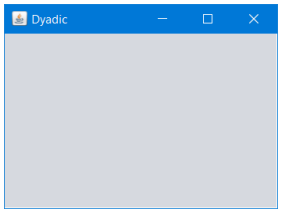


Figure 11 Window on server for observing what participants are typing in real-time

This is a window which shows the experimenter what participants are typing, while they are typing. (i.e. the experimenter can see what participants are typing privately before they send it as a turn). Because the participants haven't logged in or typed anything yet, it is blank.

In the middle of the screen another window also appears:

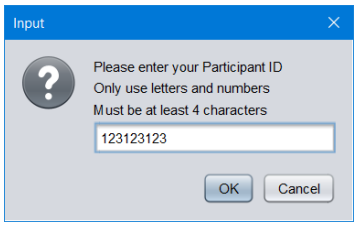


Figure 12 Logging in a participant - entering participant ID

When a client connects to the server, the server requests a Participant ID. Simply enter a random string of numbers/letters. This particular demo doesn't check whether a participant ID is on a whitelist/blacklist (you can set this if you want, but this does require a few lines of code in java).

This will lead to the next step in the login process. The server requests a username:

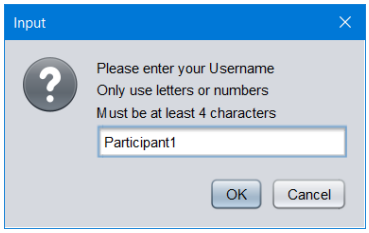


Figure 13 Logging in a participant - entering a username

Type in a username, e.g. Participant1.

When you have entered a participant ID and Username for the first participant, it will show the same two windows again for the second participant. Repeat these steps using a different Participant ID and different username.

You should now see two chat windows (See Figure 14 below). The chat windows are analogues of instant-messaging interfaces, e.g. whatsapp etc. When you have verified that you have these two chat interfaces, type some text in the lower window of both interface. Notice how the Participant 2 receives notifications about Participant 1's typing activity (and vice versa).

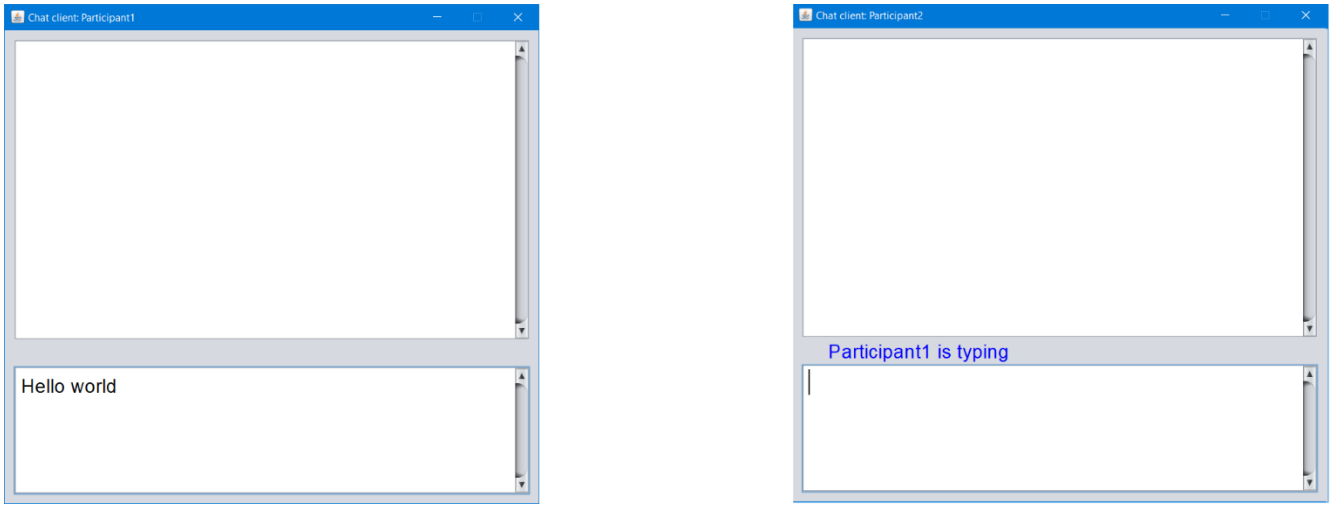


Figure 14 Simple Chat-interfaces. Each chat interface has an upper part that displays the conversation history, and a lower part where the participant types text. The title-bar of the window displays the username of the participant.

Now that the clients are logged in and connected to each other, look at the server interface. Notice the window in the top left-hand corner of the screen (Figure 15). This window shows Participant 1's text on the server before participant 1 has even sent it (It is also possible to program interventions that respond to people's turns before they have sent them!)

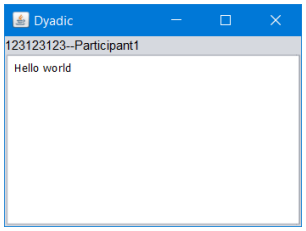


Figure 15 Server window showing participants' typing activity

Next, on the server, press the tab "Dyadic", marked (8) in Figure 16 and in Figure 19, to activate the "experimental control" interface.

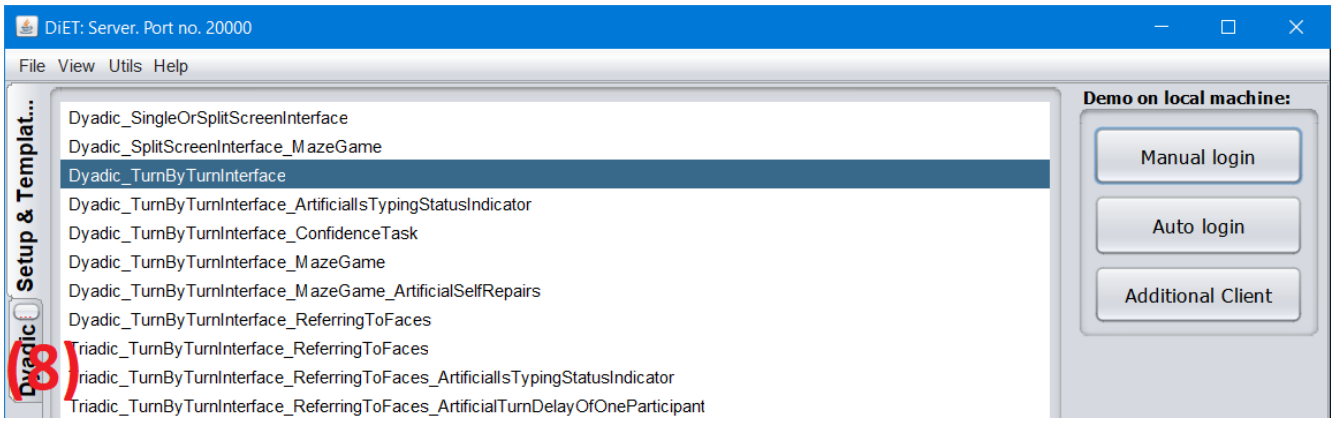


Figure 16 Activating the "experimental control" interface

This should open a new window with five tabs (Participants, Interventions, Turns, Turns detailed, Partnering). These windows are explained in more detail in the section The server interface below.

For now, choose the tab "Turns" **(12)** (Figure 17).

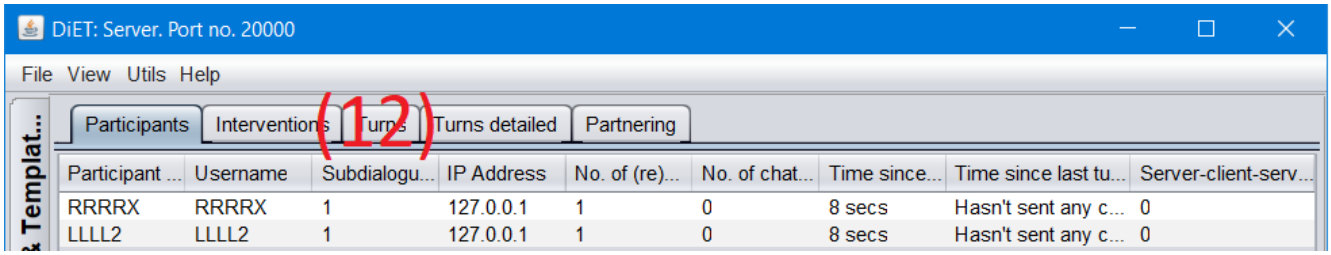


Figure 17 Selecting the option to look at the turns typed by the participants

Then type text e.g. “hello” and then press ENTER in the client’s chat interfaces and observe how it is displayed on the server.

(see the sections below for more information about each of the interfaces)

After typing and sending a few turns between the participants you can check to see how the turns have been saved.

Open the directory **/desktop/chattool/data/Saved experimental data/**

Each experiment is saved to a subdirectory. The most recently created directory contains the data from the experiment you have just carried out. The main file containing the data is “turns.txt”. You can open it in Excel or LibreOffice. Make sure you use “UTF8” as the text format and the ¦ character as a separator.

To see detailed information about how to open the data files, see the section on p. 72 below).

When you have completed this quick start, try out some of the other experiment templates.

The server interface

Launching experiments

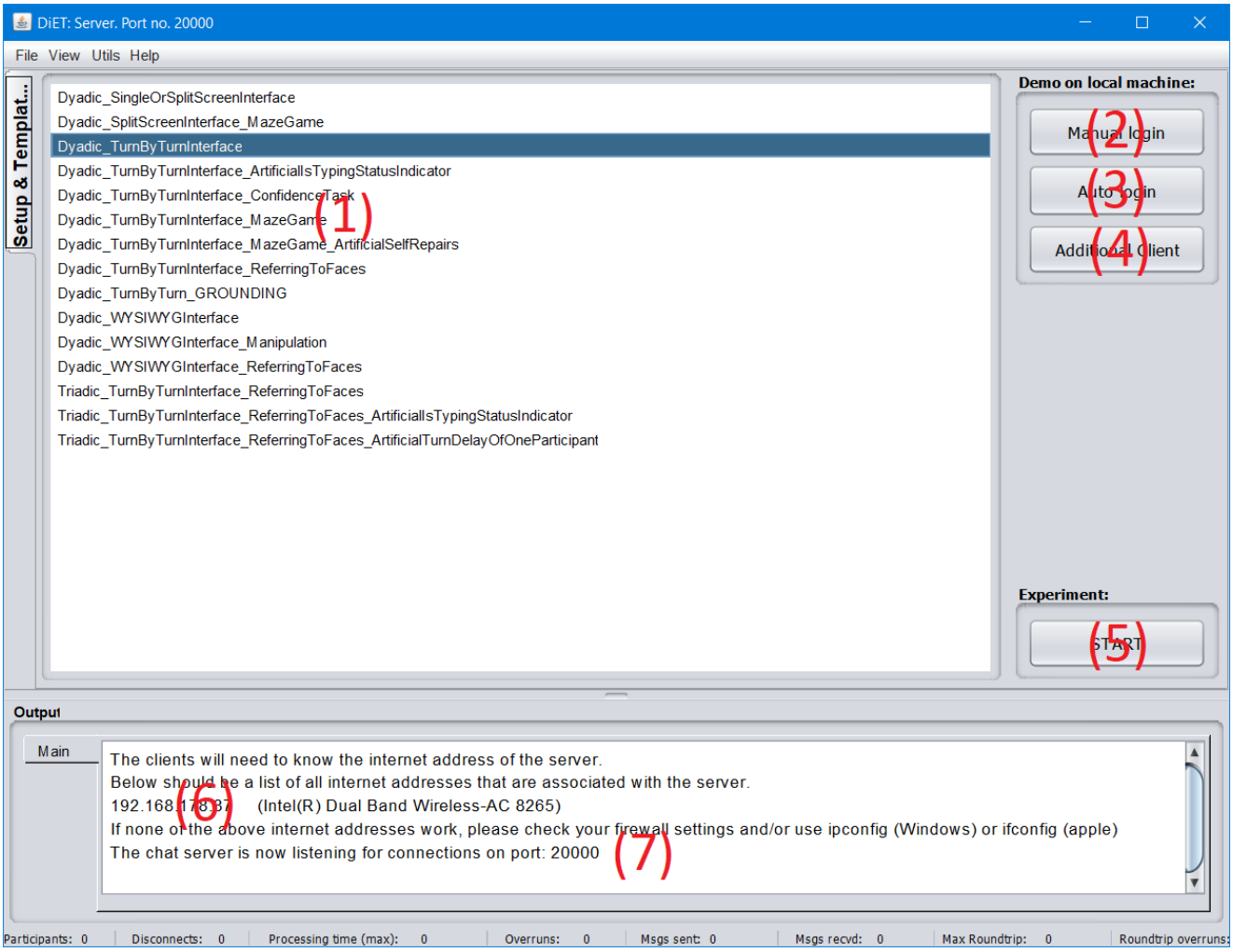


Figure 18 Launcher window

This is the main window that loads when the server starts

(1) This is the list of experimental templates that can be launched. The chattool contains many different experimental templates. When you program your own intervention, it will appear here. N.B. Because we are constantly creating and adding new interventions, this list might not be identical in the current version.

(2) (3) (4) are controls for testing (demoing) an experiment. They allow you to run all the code locally on the same computer (instead of having to copy code to the client computers and get them to connect via the network). They also make the logging in process easier.

(2) Manual login will start the selected intervention from list (1) , and will start two clients and connect them to the template, as in the quick start guide above. . So for example in this screen, when the user presses (2), the chattool will start the intervention called “Dyadic_TurnByTurnInterface” and then start two clients.

(3) This option creates pairs of clients and automatically logs them in – using preset IDs and usernames. This saves a lot of time and bypasses the tedious process of entering participantIDs and usernames for participants.

(4) This starts an additional client interface and connects to the experimental intervention. This is useful, e.g. if you are programming an intervention that can have a variable number of participants (E.g. a large group-chat). To add a new participant you just need to press this button.

(5) This button is intended for when you want to run an experiment. It starts the experimental setup that is highlighted in (1) and waits for clients to connect.

(6) The chattool attempts to find the local network address of the machine. When you run an experiment, participants are typically on separate computers connected to the server via the network. The clients need to know how to connect to the server. This is the internet address that you need to enter on the clients (See e.g. **Error! Reference source not found.**)

(7) This is the port number on which the chattool listens for incoming connections from the clients. The default is port 20000.

While running an experiment with built in interface: Participants

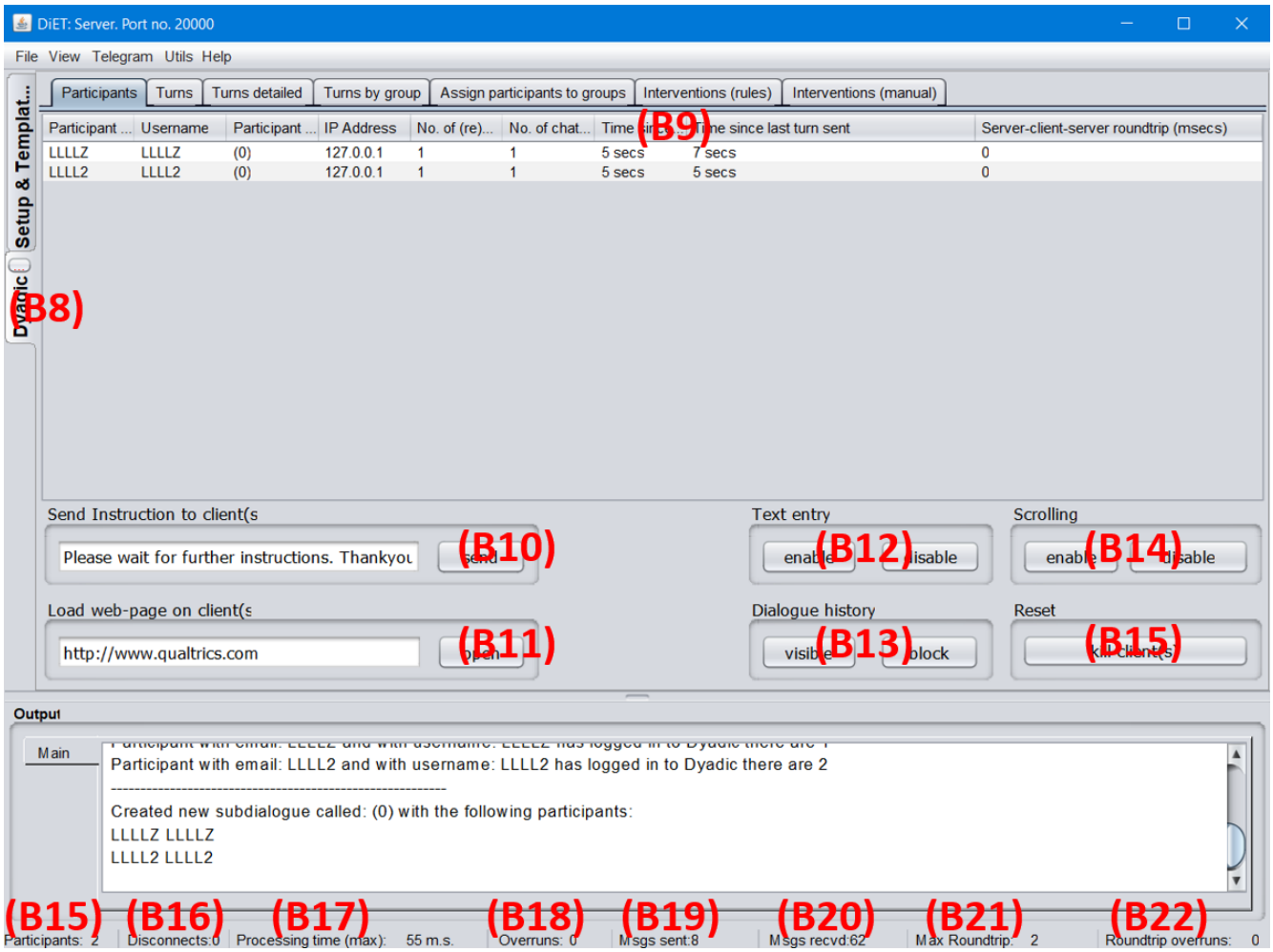


Figure 19 During an experiment

(B8) Press this tab to activate the window above

(B9) This table contains information about all the participants who are logged into the chattool. The columns are, from left to right:

- **ParticipantID:** This is the unique ID to identify the participant
- **Username:** This is the name that other participants see prefixed before each turn. This can be preset by the experimenter, or participants can be allowed to choose their own at login
- **Participant Group:** You might want to run an experiment that assigns participants to different partners / subgroups during the interaction. This leads to experimental designs where there are multiple dialogues happening in parallel. Each one of these conversations is stored as a "participant group".
- **IP address:** This is the internet address of the client, as reported by the client software.
- **No of (re)connects:** The chat-tool is designed to reconnect if there are network errors. To help debug/diagnose network issues, this column records how many times the client has connected/reconnected to the server.

- **No. of chat turns:** This records how many chat turns (i.e. how many contributions) were sent by each participant

- **Time since last message:** Every few seconds the clients ping the server to make sure that the connection is still active. This records the time since the last ping was recorded. This is useful for diagnosing errors with the connection

- **Time since last turn sent:** this records how many seconds have elapsed since the participant last sent a turn. This is also useful for diagnosing connectivity issues. It is also helpful for keeping an overview to identify participants who haven't typed anything in a while (perhaps because they don't understand the instructions)

- **"Server-Client-Server** roundtrip time. Every 10 seconds the server "pings" the clients and gets a response. This column shows the most recent value for the roundtrip.

(B10) This allows the experimenter to send a message to all the clients.

(B11) This instructs the web browser on the clients to open a particular webpage. This can be used to e.g. get participants to fill out a questionnaire after the experiment.

(12) This functionality can be used to prevent participants from entering text in the text-formulation window. This stops participants typing. Both (B12) and (B13) can be used together, e.g. to make participants pay attention to another window (e.g. when filling out a questionnaire)[(See Figure 10Figure 20).] Note that this functionality only works with the "Turn By Turn" interface, not with the WYSIWYG Interface [(see the section on interfaces on page 72)]

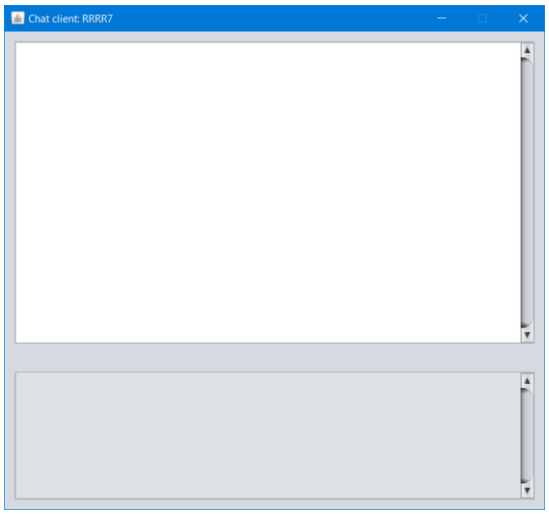


Figure 20 A chat client interface with text entry blocked

(B13) Dialogue history: This enables / disables the upper window on the chat clients. Selecting Disable greys out the chat history – preventing participants from reading the chat history. Both (B12) and (B13) can be used, e.g. to make participants pay attention to another window (e.g. when filling out a questionnaire) (see Figure 21). Note that this functionality only works with the "Turn By Turn" interface, not with the WYSIWYG Interface (see the section on interfaces on page 72)

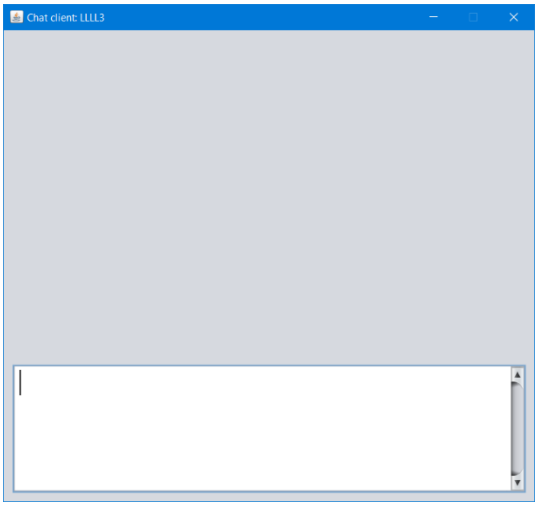


Figure 21 An interface with conversation history blocked

(B14) This enables/disables the scrolling in the clients' conversation history windows. This can be useful, e.g. if you wish to prevent participants from scrolling backwards and forwards in the history, especially in task-oriented dialogues.). Note that this functionality only works with the "Turn By Turn" interface, not with the WYSIWYG Interface Note that this functionality only works with the "Turn By Turn" interface, not with the WYSIWYG Interface (see the section on interfaces on page 72).

(B15) This sends an instruction to the clients to shut down.

(B16) This shows at a glance how many participants are logged in.

(B17) Processing time (max) When a message is received from one of the clients it has to relay those messages to all the other clients so that they receive it. Ordinarily this is effectively instantaneous. However, when programming experimental interventions that manipulate the turns in some way, the server does some processing (E.g. parsing or task-related processing). It is essential that this code doesn't take too long to execute – as this will create lag in the interaction. This variable records the maximum recorded processing time during an experiment. A good rule of thumb is that it shouldn't go over 300ms.

(B18) Overruns. This records how many times the processing time (see 17) goes over a preset maximum threshold. Currently the default is 500ms.

(B19) Messages sent: This records how many messages were sent by the server to all clients. A message is not the same as a turn – messages can be pings, as well as instructions to show "is typing" , disable/enable textentry, etc.

(B20) Messages received: This records the number of messages received by the server. Each keypress from the client is sent as a separate message. Each update from the interface is saved as a separate message.

(B21) Max. Roundtrip: Every 10 seconds the chat tool sends a "ping" message to each client. It records the time of sending and when it gets a response. This is the roundtrip. It measures how slow the network connection is. This records the maximum recorded roundtrip.

(Note: There are many other features of the client interface that can be controlled, but that does require scripting the interventions in java)

While running an experiment with a built-in interface: Turns

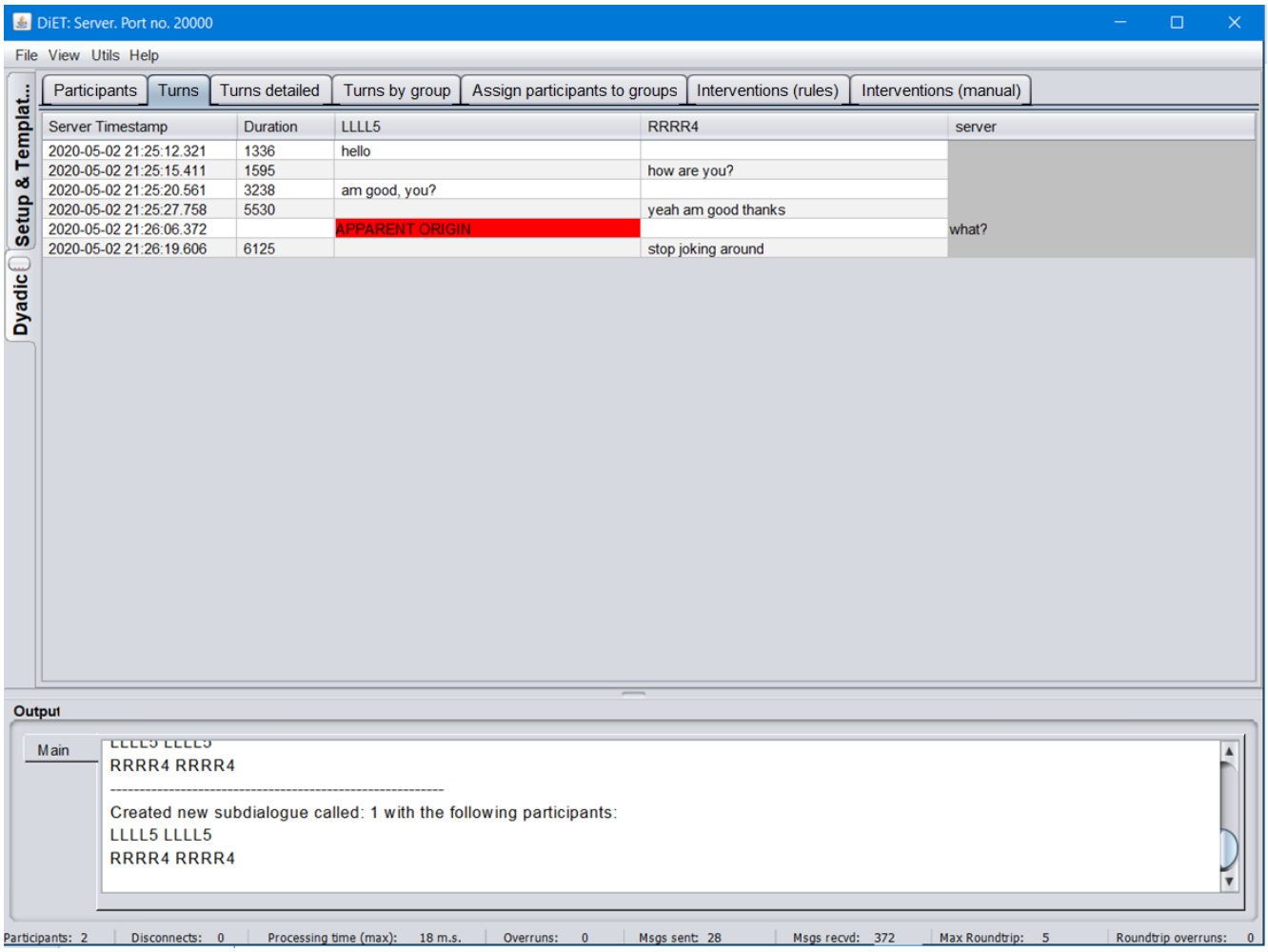


Figure 22 Conversation history

This interface displays the turns that were typed by the participants.

"**Server Timestamp**" shows the time when the message was received by the server. The messages are ordered in the order in which they were received by the server.

"**Duration**" displays how long it took the participant to formulate the turn, in milliseconds (this is the duration from the first keypress till when they pressed enter to send the message).

The subsequent columns are the turns produced by each participant. The turns produced by each participant are displayed in separate columns. This makes it easier to follow who typed what. This table displays a sequence of interaction between two participants, with ID RRRR6 and RRR4.

Notice that there is also a column for the "server". This allows you to see the artificial "spoof turns"!. In this example the server sent a spoof turn "what?" that appeared to originate from participant with ID LLLL5. See Figure 23 Figure 23 Dialogue with fake inserted text "what?" by the serverbelow to see what this looks like on the participants' screens.

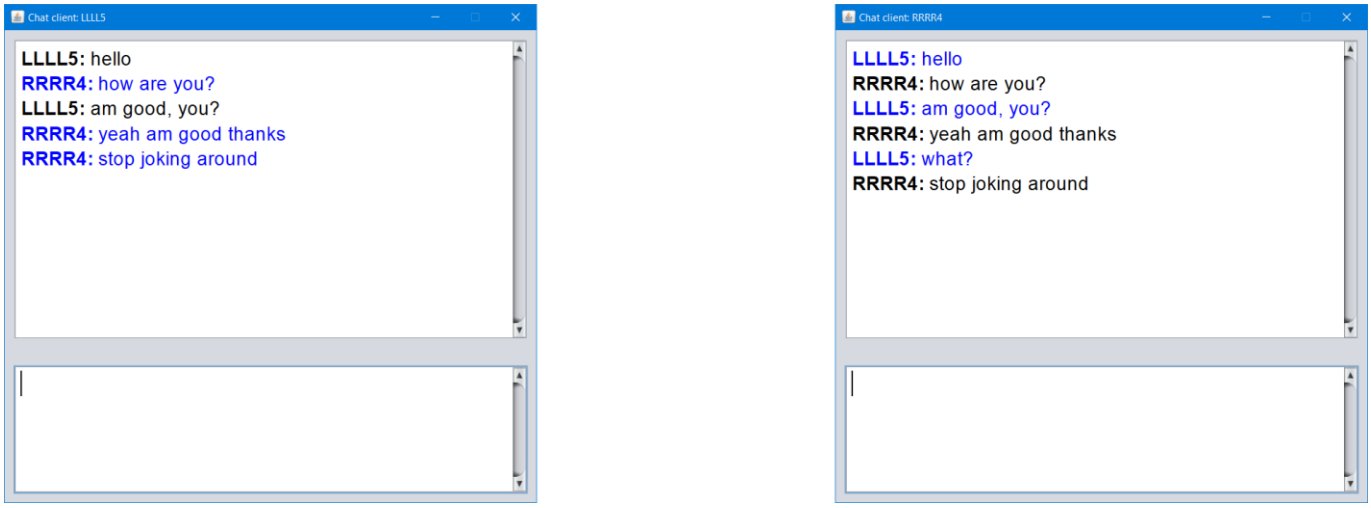


Figure 23 Dialogue with fake inserted text "what?" by the server

While running an experiment with the built in interface: Turns detailed

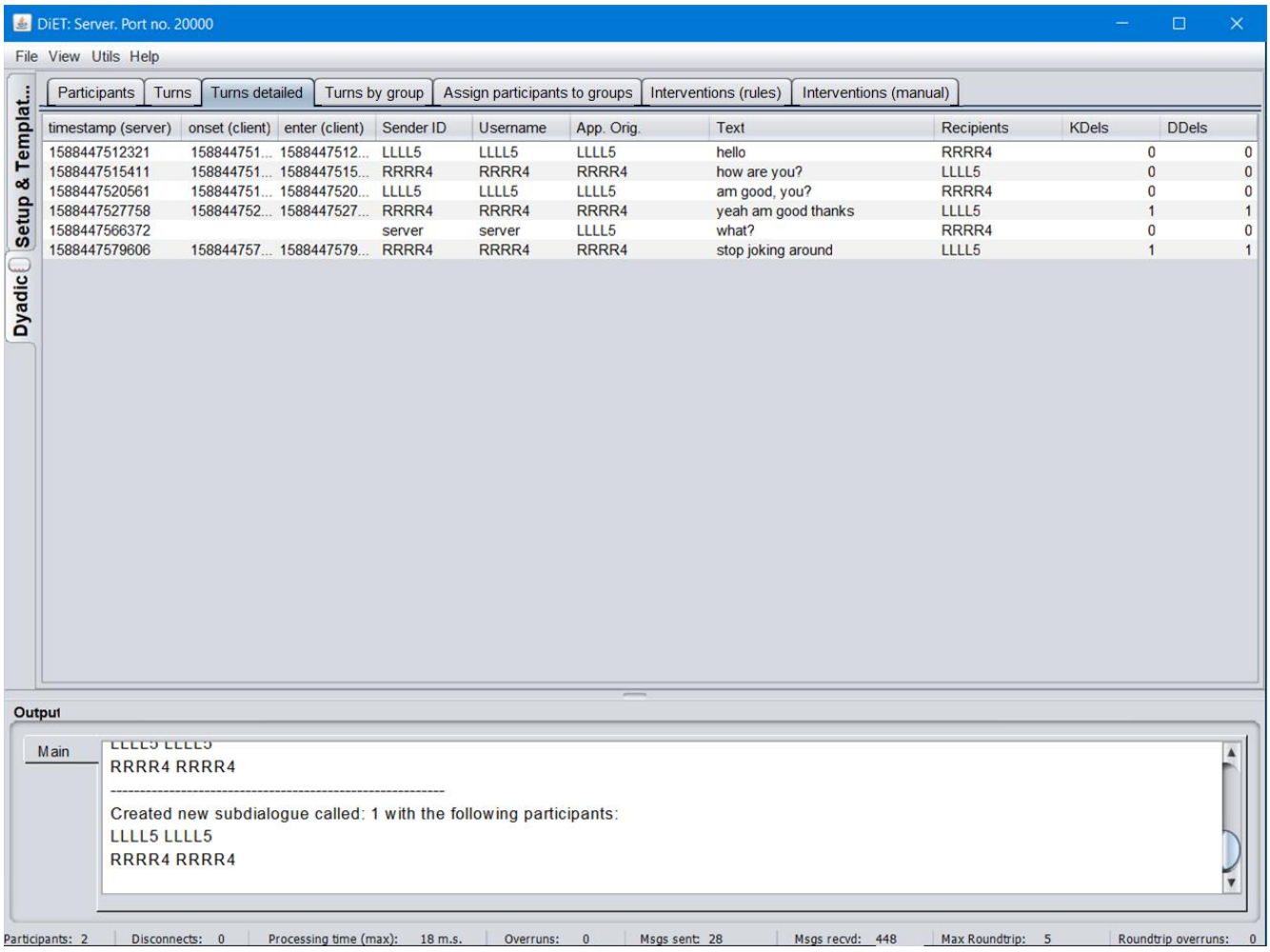


Figure 24 Detailed view of turns

This interface displays a more detailed view of the turns.

Timestamp (server): This is the time when the message was received by the server.

Onset (client): This is the time, as recorded on the participant's computer of the first keypress

Enter (client): This is the time, as recorded on the participant's computer, when they pressed enter and sent the message. In principle, assuming that the clocks of the server and clients are perfectly synchronized, you could calculate the network lag by subtracting **enter-client** from **timestamp(server)**.

Note that the times are recorded in "unix" / epoch time – this is one of the most standard time formats in computers – the number of milliseconds that have elapsed since 1970. It is very straightforward to convert to human-readable dates/times, see e.g. https://www.epochconverter.com/

Important:
Although most computers are synchronized to an atomic clock via the operating system, some computers aren't! There is no guarantee that all clocks of all computers are synchronized. It is extremely likely, though

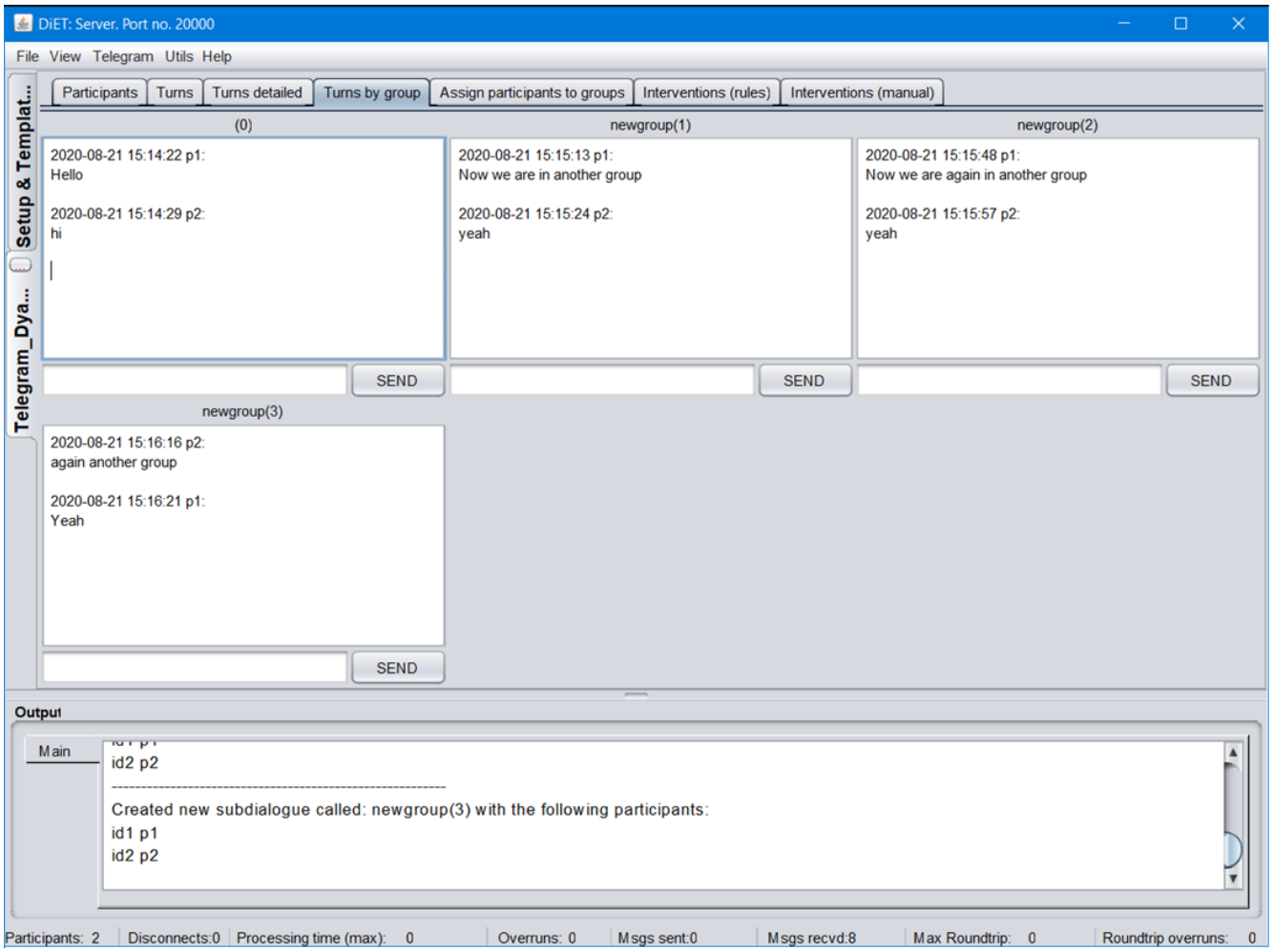
that even if the clocks aren't synchronized, that each computer's clock is internally consistent. In other words, if **timestamp (server)** and **enter (client)** have the same value, this is no guarantee that they are in fact simultaneous. However, if **enter (client)** is 4000ms more than **onset (client)**, this does mean that the turn almost certainly took 4 seconds to formulate.

Sender ID: This is the participant ID of the person who sent the turn

Username: This is the username of the person who sent the turn.

Apparent Origin: This is who the message appears to be sent from. Ordinarily, this is the same as "Username", but is different for fake messages. Notice that the second-last row of Figure 8 shows the fake turn "What?", which is generated by the server, but that appears to come from participant LLLL5.

While running an experiment using the built-in interface: Turns by groups



This screen shows the conversations of each group separately. Participants can be assigned to different groups manually by the experimenter [Using the screen “Assign participants to groups”] or by the experimental script. Notice how in this screen there are 4 separate windows showing the conversations of four different groups.

Whenever a new group is created, this screen automatically adds a new window showing the conversation of that group.

Notice also that each group has a textfield and a SEND button. This allows the experimenter to send instructions to specific groups

While running an experiment using built-in interface: Manual Interventions

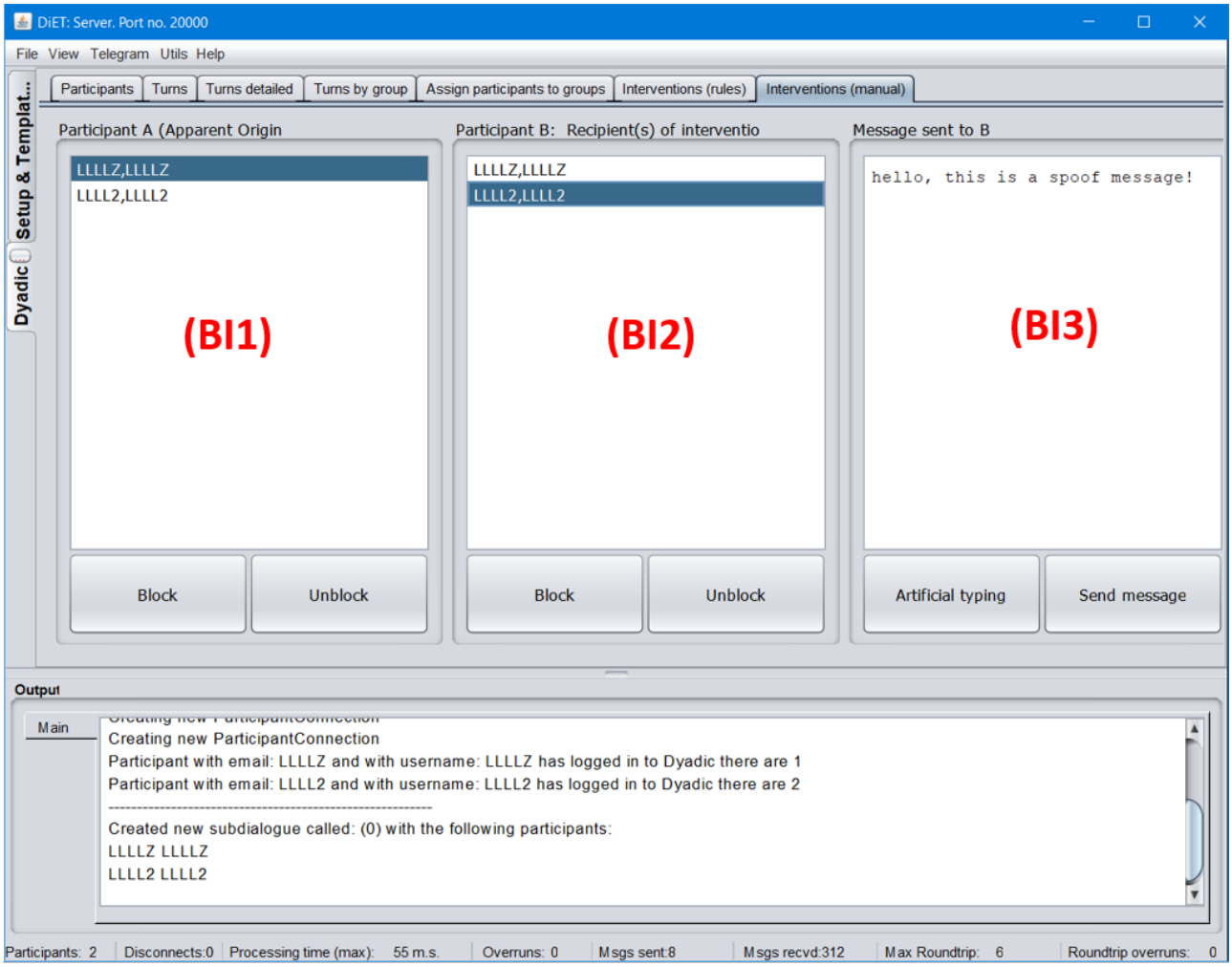


Figure 25 Generating Fake Turns

This is an interface for sending "spoofed" turns. The explanation below describes the steps involved in sending a fake turn to Participant B that appears to B, to originate from Participant A. The steps are typically:

1. Server sends fake "is typing" activity from Participant A to Participant B
2. Server sends fake message from Participant A to Participant B
3. Server blocks Participant A from sending a message to B, giving B time to respond to the fake message
4. Server unblocks Participant A after receiving a response from B or after a timeout

The interface above shows the experimenter in the process of sending a fake turn "hello, this is a spoof message!" (BI3) to the participant with ID LLLL2 (BI2), which appears to come from the participant with ID LLLLZ (15). In order to make the intervention more plausible, the experimenter can add fake "is typing" activity before sending the post using the "Artificial typing" button (BI3) before sending the turn (BI3). In order to prevent the other participant A from sending a message to participant B, while B is responding to the fake turn, the interface allows the experimenter to block participant A's textentry interface (see Figure 4, above) using the button "Block" (BI1). In order to ensure that participants don't get suspicious of their textentry box getting disabled, it's important to randomly block both people's interfaces during the experiment for short periods (BI1,BI2). It is much easier to program this automatically in the chat-tool, but this interface can be used for piloting different types of intervention

While running an experiment - Participant Partnering

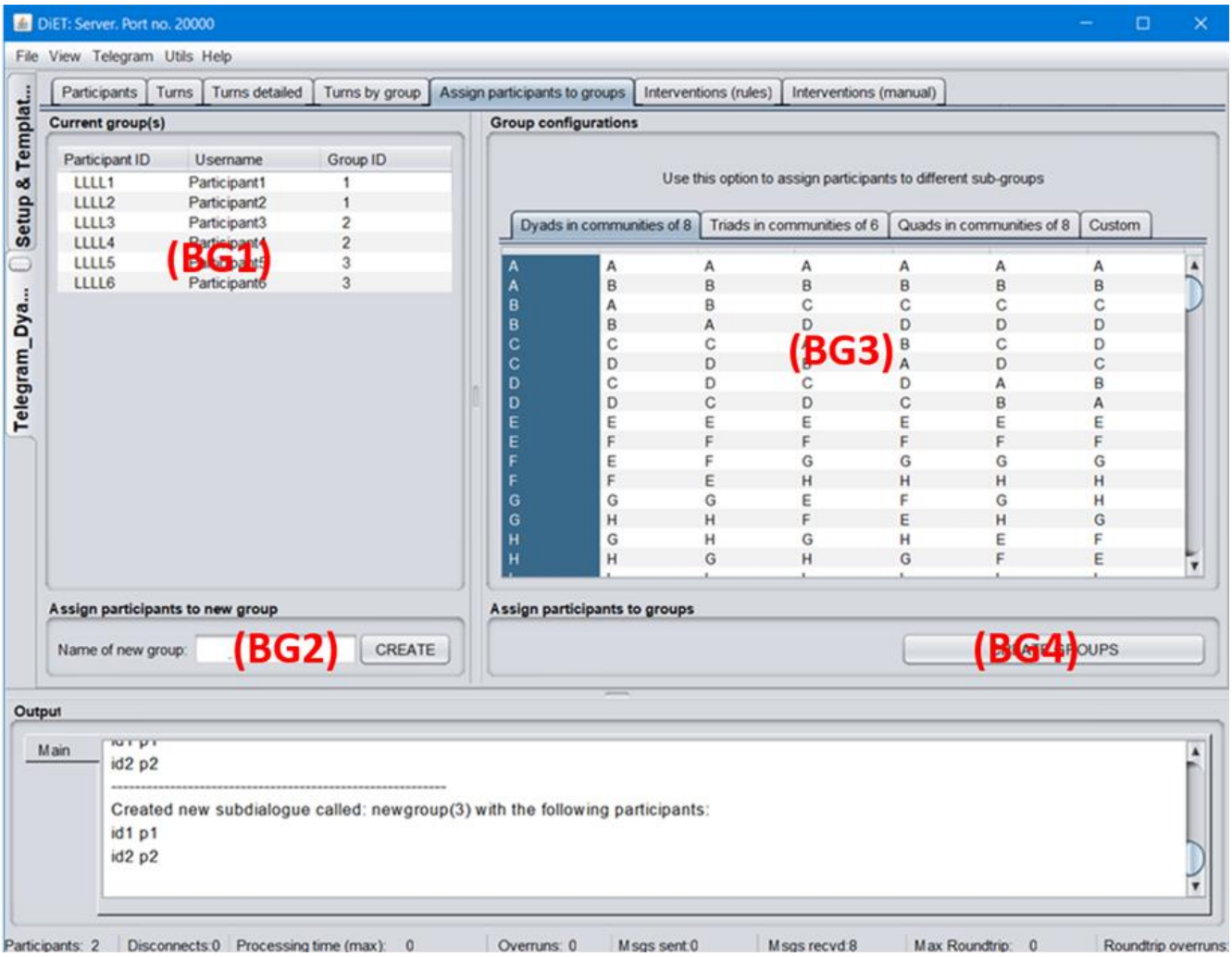


Figure 26 Participant subdialogues

This screen allows the experimenter to manually assign participants to new groups. There are two ways of creating new groups:

Assigning a list of participants to one single new group.

(BG1) displays a list of all participants who are logged into the chat tool and which group they are currently part of. To assign participants to a new group, select the participants in table (BG1), type the new group name (in this case the new group name is “newgroup”) and press CREATE. This will automatically create the new group.

On this screen you can see there are 6 participants who are logged in, Participant1, Participant, Participant3, Participant4, Participant5, Participant6.

The column “Group ID” shows who is speaking with whom. Particpants with the same Group ID are interacting with each other. In this table it shows that Participant1 and Participant2 have Group ID 1, Participant2 and Participant3 have Group ID 2, and Participant 5 and Participant 6 have Group ID 3.
In other words, this shows that there are three groups (i.e. there are three dyads) having separate conversations. See **Error! Reference source not found.** below

(BG2) This allows the experimenter to dynamically assign participants to different partners. At the moment Figure 9 and Figure 10 show three separate dyads.[**Error! Reference source not found.** and **Error! Reference source not found.**](below) show how to re-assign participants to different partners.

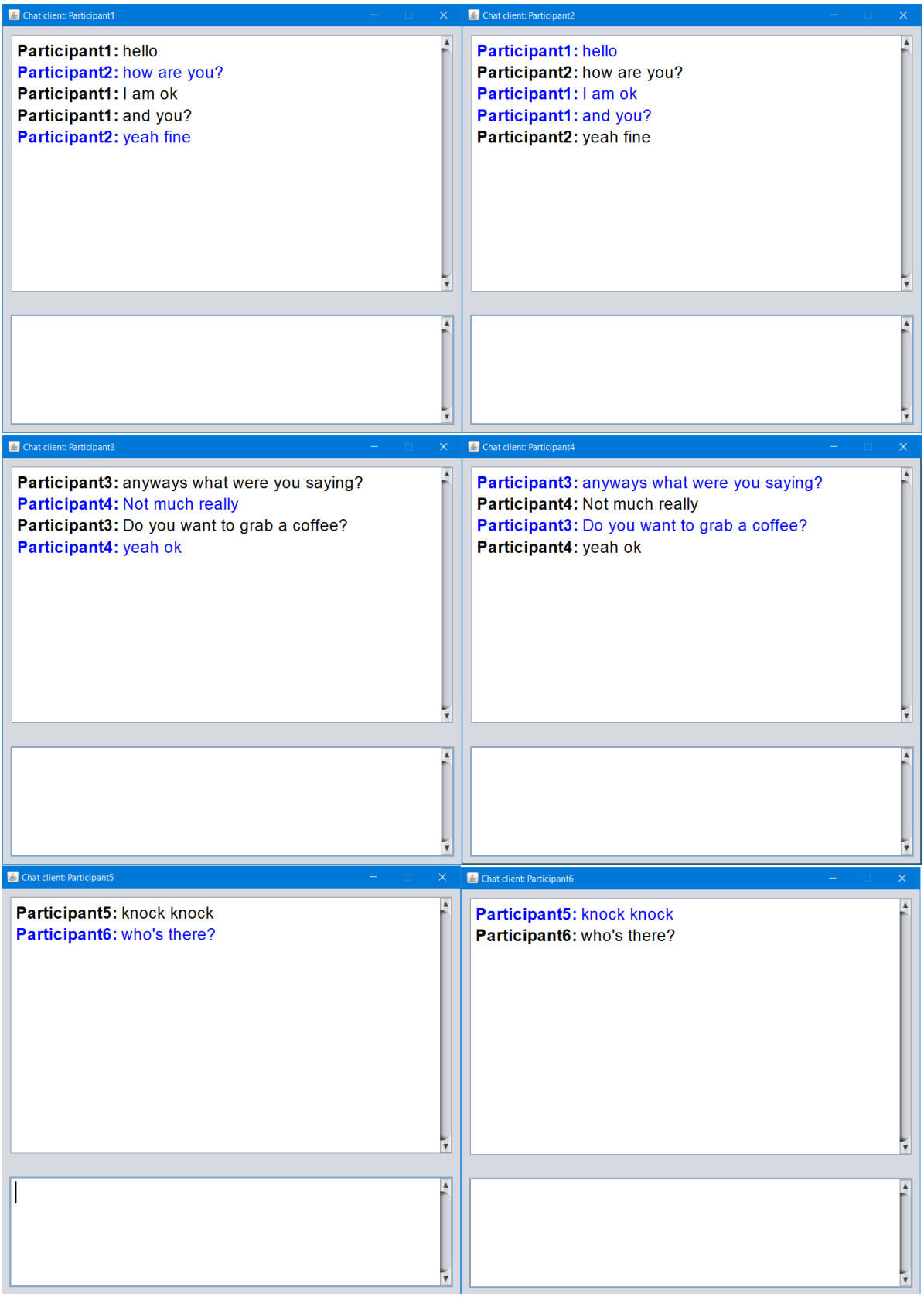


Figure 27 Six participants are logged in, there are three "subdialogues": (participant 1 and participant 2, (participant 3 and participant 4) , (participant 5 and participant6). Each participant only speaks with one other participant.

To create a new group of participants, in this case to reassign participant1, participant2, participant 3 to a triadic group, choose a name of the new group, e.g. “Triad 1” and type it into the textfield “Name of new subdialogue” (in **Error! Reference source not found.**) and then press “CREATE”. Then select participant 4, p articipant 5 and participant 6, enter “Triad 2” as the new name and press “CREATE”

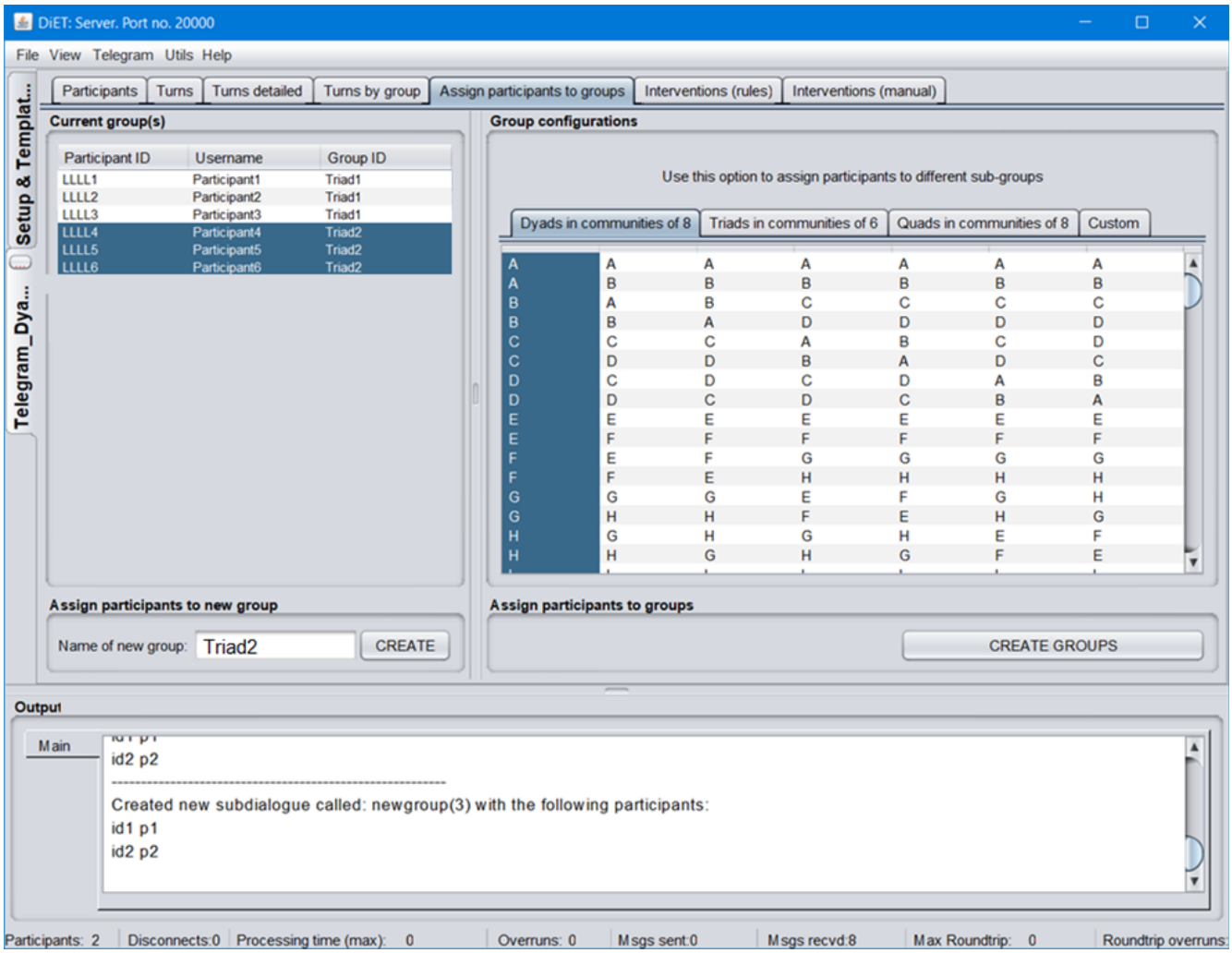


Figure 28 Reassinging participants to different groups - in this example assigning participant4, participant5 and participant6 to a triadic interaction.

The results are shown below.

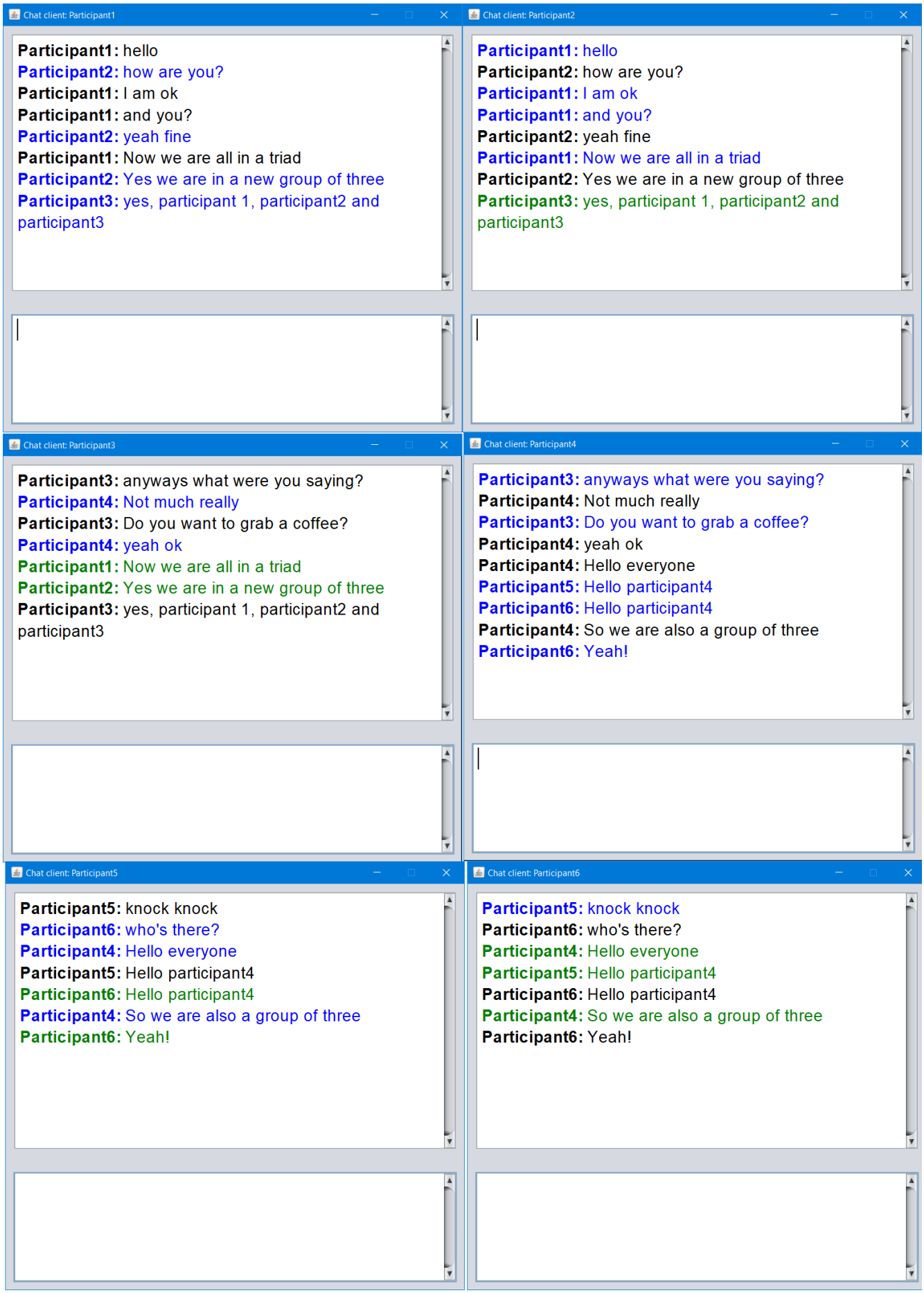


Figure 29 There are now two groups: Group 1 contains Participant 1, Participant2 and Participant 3. Group 2 contains Participant 4, Participant 5 and Participant 6

Simultaneously assigning ALL participants to multiple groups.

Consider an experimental design where you have 100 participants who are paired in 50 dyads. Now imagine you wish to pair the 100 participants in new dyads. It would take 50 steps to do this using the method described above.

Table (BG3) contains a list of group configurations. Notice how the first column contains
"A", "A", "B", "B", "C", "C", "D", "D", "E", "E", "F", "F", "G", "G", "H", "H"

What this means that if the experimenter presses (BG4) "CREATE GROUPS" then the first participant in table (BG1) (i.e. p1) will be assigned to a new group "A". The second participant in table (BG1) (p2) will also be assigned to group A. In this example there are only two participants (p1 and p2), but if there were more participants - the third participant would be assigned to group "B". The fourth participant would be assigned to group "B", etc.

In other words, selecting the first column and pressing "CREATE GROUPS" ends up creating new dyads where the first participant is paired with the second, the third participant is paired with the fourth, the fifth participant is paired with the sixth, etc.

But other pairings are possible! Now suppose you select the second column. Notice how the column contains
"A", "B", "A", "B", "C", "D", "C", "D", "E", "F", "E", "F" etc.
If you were to select this column and press (BG4) "CREATE GROUPS" then the first participant in the list (BG1) would be paired with the third participant in a group called "A". The second participant would be paired with the third participant in a group called "B", etc.

Of course groups can consist of any number of participants. The chattool provides a few configurations for groups of three (triads) and groups of four (quads). If you wish to use your own, custom list of groups, you can select "Custom" and load your own list. See, e.g. …\experimentresources\groupconfigurations\dyads.csv

Interfaces

The chattool contains a number of different interfaces.

Turn By Turn Interface

This is the standard interface that is used in most instant messaging apps, e.g. whatsapp, signal etc.

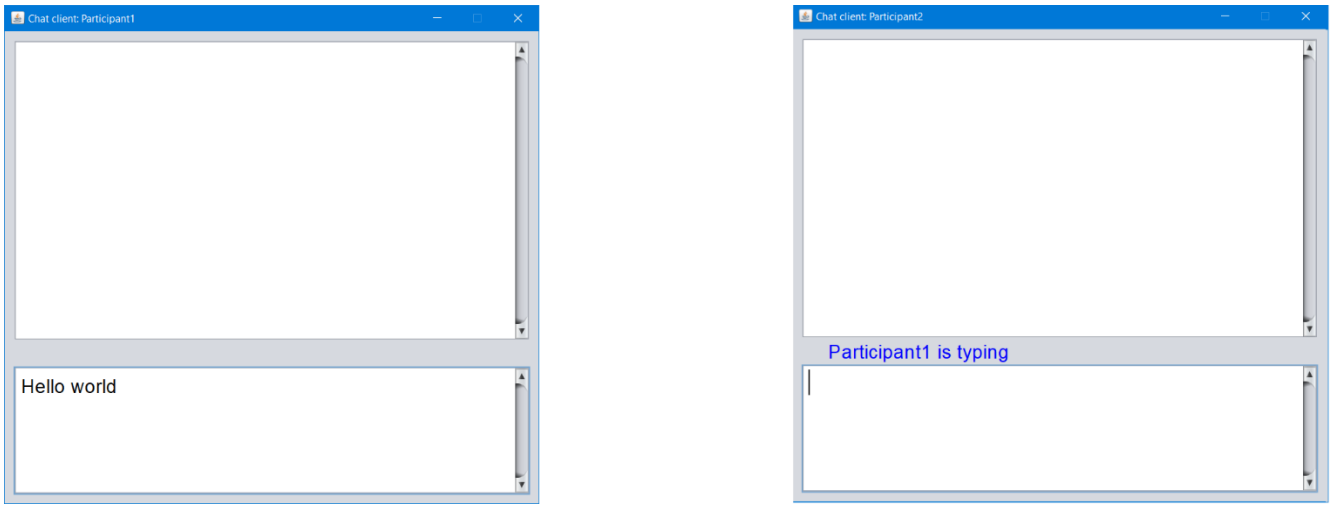


Figure 30 Turn By Turn Interface

What You See is What You Get (WYSIWYG) Interface

The chattool also contains a couple of WYSIWYG Interfaces where participants can see each other type in real-time. The text appears from right to left (Similarly to streaming news pixel board displays). The interface allows the experimenter to specify how long the text is displayed before it fades out..

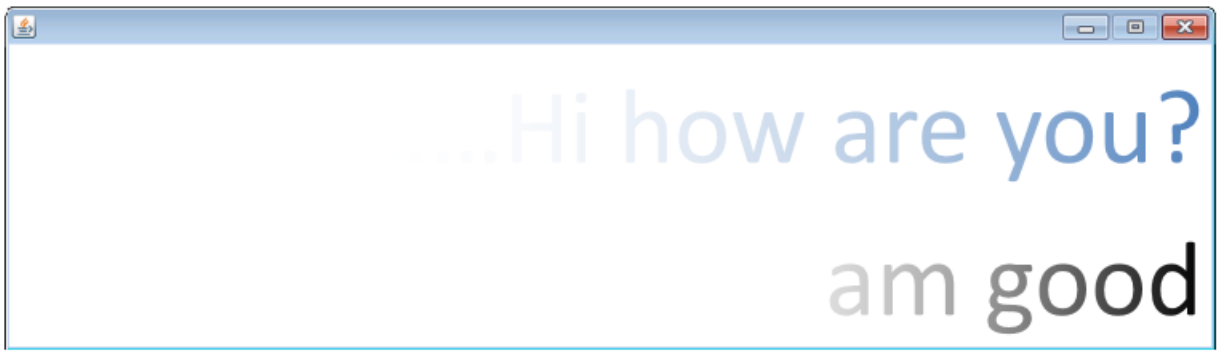


Figure 31 WYSIWYG Interface with each participant on a separate row

In this interface, each participant's text appears on a separate row. This allows both participants to type simultaneously, without leading to incoherent text.

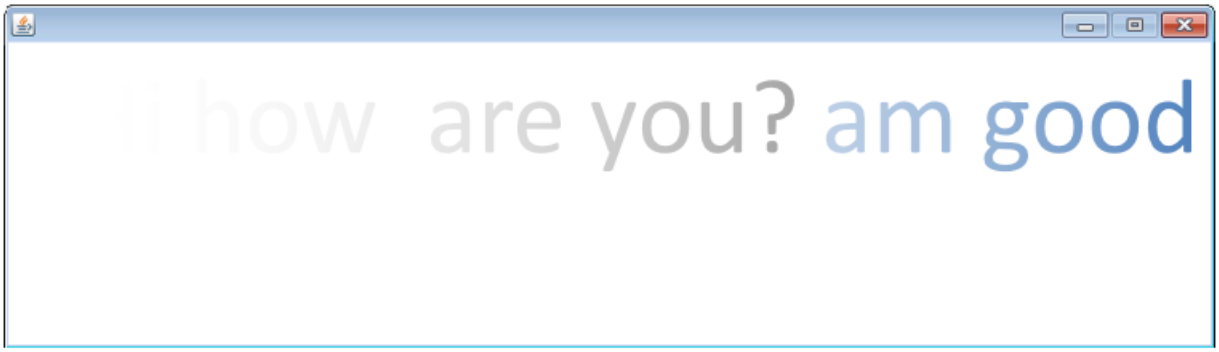


Figure 32 WYSIWYG Interface: Both participants on the same row

In this interface both participants' text appears on the same row. This forces both participants to coordinate on turn-taking..

These interfaces are very powerful – they are arguably much more analogous to spoken conversation. First, they don't allow participants to delete their turns privately – the patterns of self-repair look very similar to spoken dialogue repairs. Secondly, participants are observed to give feedback while the other person types (e.g. "typing "ok", "Yeah", etc while the other types.) Third, the timing of participants'turns is much more important here. Crucially, this set of interfaces allows much more fine-grained experimental control over moment-by-moment coordination phenomena.

These interfaces are implemented in the experiment templates:

- Dyadic_WYSIWYGInterface (the default implementation)
- Dyadic_WYSIWYGInterface_Manipulation
  (an experimental template that inserts fake clarification requests into the interaction)
- Dyadic_WYSIWYGInterface_ReferringToFaces (a joint reference experiment)

Data saved by the chattool

The folder location

When you start a new experiment, e.g. you select an item in the list (1) and press "start", the chattool creates a new subdirectory where all experiment-related data is saved. Suppose you have the chattool stored in **c:\chattool**, then the data will be stored in a subdirectory **c:\chattool\data\saved experimental data**

The subdirectory is created automatically, e.g.
c:\chattool\data\saved experimental data\0001DyadicMazeGame
The chattool takes the name of the intervention you selected from the list in (1) and prepends a numerical prefix, starting at 0001 and incrementing for each new experiment.

N.B. If you are running the code from a java jar file and there is no "/data/saved experimental data/" directory, the code will automatically create a directory.

The saved data – turns.txt

The main file containing the information from the experiment is in a file "turns.txt", e.g.

c:\chattool\data\saved experimental data\0001DyadicMazeGame\turns.txt

This is a CSV file, stored in UTF8, with the "¦" character as a separator.

To open this file using open-source software, you can use LibreOffice Calc (https://www.libreoffice.org/).
In calc, choose, "File – Open", then select "**turns.txt**". It will show an "Import" menu like this. Use the settings shown in Figure 33. (*N.B. You can also use turnsattrbivals.txt" – this contains the same information. This is the newer method, currently in development, which in future will also save the data in JSON attribute/value pairs, making it more amenable for analysis/integration with python and javascript*)

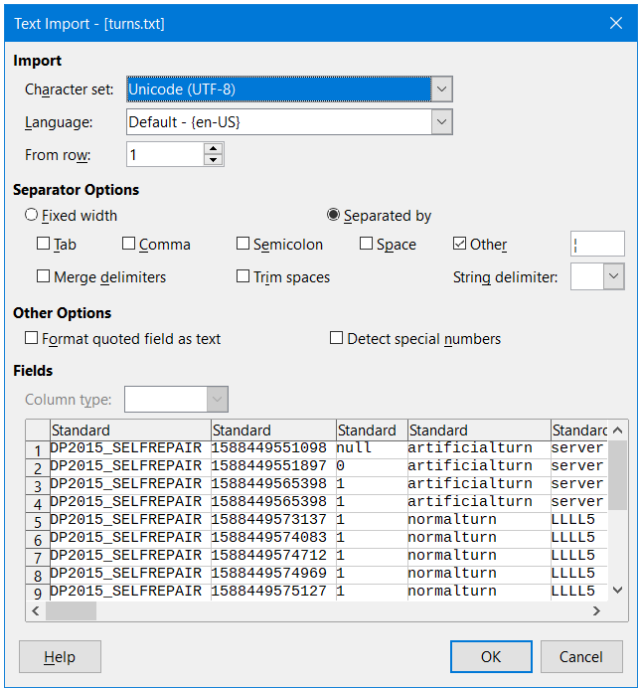


Figure 33 Loading data with LibreOffice

Alternatively, in excel, choose “Data” from the ribbon, then choose “From Text”, then select these options:

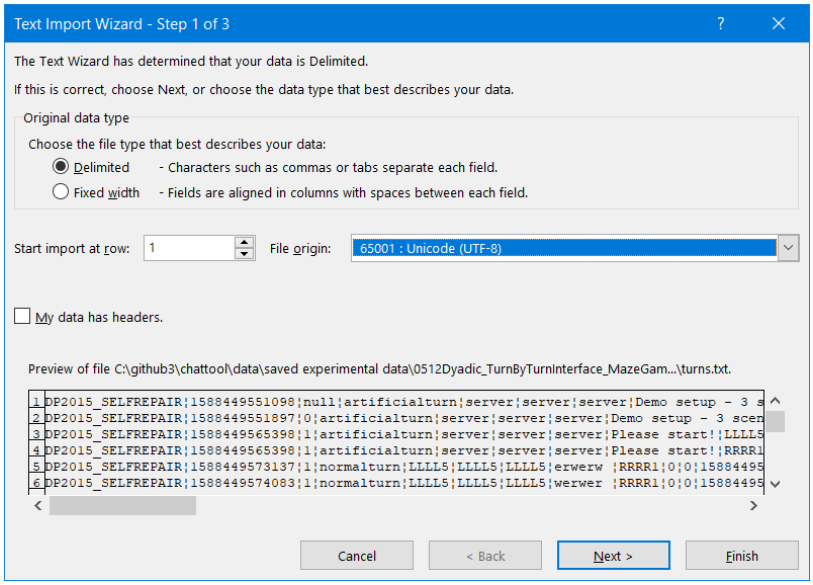


Figure 34 Importing data in excel, step 1

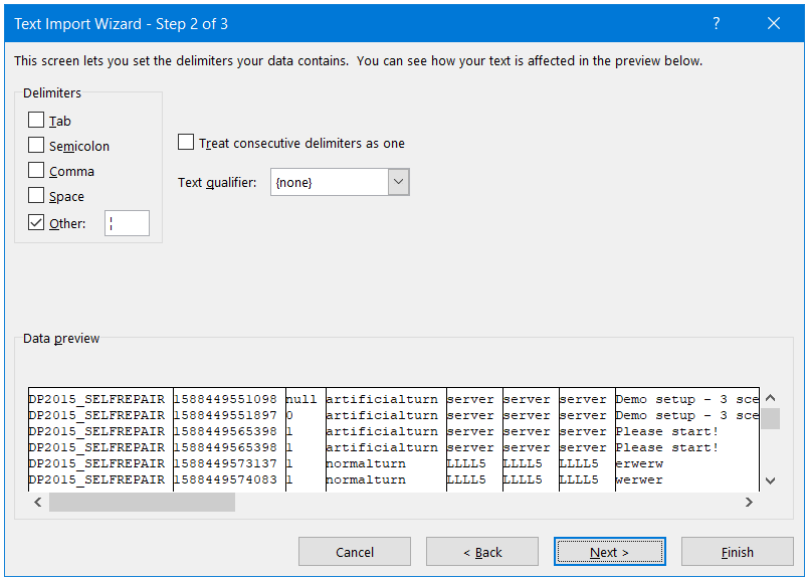


Figure 35 Importing data in excel step 2

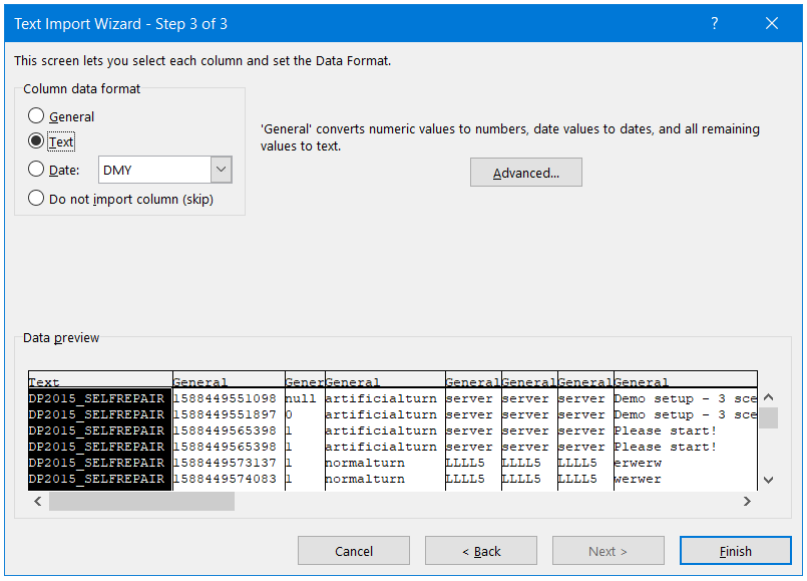


Figure 36 Importing data in excel step 3

Each row in this spreadsheet is a turn that is either produced by one of the participants or produced by the server. It is essentially a more detailed version of the table in Figure 8, above.
The columns contain specific information about individual turns

Important:
The "turn headers" (i.e. the names of each column) are stored in a separate file called "**turnsasattribvals.txt_HEADER.txt**" in the data folder.

(For any programmers reading this) This is more unwieldy than having the column names in the first row of the CSV file. However, it is necessary, because the chat-tool has to be able to save additional data in other columns: The first 16 columns are common to all interventions. However, some interventions, especially those involving task-oriented dialogue save additional data, e.g. in the maze game: which maze they are solving, whether the maze was correct, etc. These additional columns are always saved to the right of the first 16 columns. If the CSV file had the names in the first row, all the column names need to be fixed statically at the start of the experiment (As it is really tricky going back in a file and editing individual strings, instead of simply

appending to the end of the file). This makes programming interventions much easier, because it allows the experimenter to save an open-ended number of attribute-value pairs to the file, and each attribute will be given its own column.

Column A : ExperimentID An identifier of the type of experiment (This is automatically generated by the ConversationController object)

Column B: ServerTimestampOfSavingToFile This is the timestamp, recorded on the server, when the row of data was saved to the CSV file

Column C: SubdialogueID This identifies the "subdialogue" in the interaction. This is only relevant for experiments which involve multiparty interactions where there are multiple groups. Each group is assigned a separate subdialogueID.

Column D: Turntype This identifies what kind of data is stored in that particular role in the CSV file. Turns produced by participants are saved as "normalturn". There are other types of data, e.g. "servermessage" - which are instruction messages that were sent to the clients from the server.

Column E: SenderID This is the Participant ID of the Participant who sent the message.

Column F: SenderUsername This is the username of the Participant.

Column G: ApparentSender This is who the participant appears to be to the recipient of the message. This is only relevant for turns that are spoofed. For example, if Participant C receives a message that was created by Participant A, but appears to be sent from Participant B, then the ApparentSender would be Partiicpant B.(This shows who the recipient thinks sent the message)

Column H: Text This is the text of the turn that was sent.

Column I: Recipient(s) The participant(s) that received the message.

Column J: NoOfDocumentDeletes This is the number of characters that were deleted in the text formulation window. Usually this is the same as NoOfKeypressDeletes (below) - but some people select a large chunk of text and delete or replace it with other text - this captures these deletions.

Column K: NoOfKeypressDeletes This is the number of times the participant pressed the physical Delete key on the keyboard while formulating the turn.

Column L: ClientTimestampONSET This is the time (in msecs) of the first keypress, recorded on each client.

Column M: ClientTimestampENTER This is the time (in msecs), also recorded on the client, when the participant pressed ENTER and sent the message.

Column N: ServerTimestampOfReceiptAndOrSending This is the time (in msecs), recorded on the server, when the message from the client was received on the server.

Column O: TextAsformulatedTIMING This shows, character by character, how a turn was produced. In order to display this information a simple notation is used that is both human-readable and can also be easily parsed by a computer script. Each keypress is prefixed with superscript representing the time that has elapsed since the previous keypress. Backspace keyspresses are represented with a left-

pointing arrow. It also records whether the participant produced their message while the other was typing or not. This notation is explained below:

Recording the time of each keypress:
So for example, suppose a participant types and sends "Hello" , this could yield:
^{0}H ^{110}e ^{289}l ^{182}l ^{348}o ^{6821}ENTER
This shows that "e" was typed 110 ms after the "H". The first "l"was typed 289ms after the "e". The second "l" was typed 182ms after the first "l", the "o" was typed 348 msec after the "l". Finally, it shows that the turn was sent (i.e. Enter was pressed) 6821 msecs after the "o" was typed

Recording the backspace keypress::
Suppose a participant types "dig" and then uses backspace to edit the turn to "dog".
This could yield:
^{0}d ^{110}o ^{289}g ^{182}← ^{348}← ^{300}i ^{289}g
Notice each backspace is recorded as a left-pointing arrow ←

Recording the "is typing" notifications that are displayed on the participant's screen. This makes it possible to investigate whether participants start/continue/stop typing when they see their partner start/stop typing. Suppose a participant is typing a turn, and wants to write "this shape is red", but half-way through, while the participant is typing "shape" the other participant starts typing, which displays an "is typing" notification on the interface. This could yield:

^{0}T ^{110}h ^{289}is ^{182} ^{348}s ^{800}h ^{110}a ^{90}⊆ ^{289}p ^{182}e ^{5800}⊇ ^{600}i ^{500}s ^{490} ^{300}r ^{289}e ^{100}d ^{300}ENTER

The start of an "is typing" notification is represented with "⊆" and the end of the "is typing" notification is represented with ⊇. The text below highlights these components for illustrative purposes:

^{0}T ^{110}h ^{289}is ^{182} ^{348}s ^{800}h ^{110}a ^{90}⊆ ^{289}p ^{182}e ^{5800}⊇ ^{600}i ^{500}s ^{490} ^{300}r ^{289}e ^{100}d ^{300}ENTER

Notice, that from this representation you can see that

- the participant received the "is typing"notification just before typing ^{289}p
- after finishing typing "shape", the participant stopped typing for 5.8 seconds, only resuming 600 milliseconds after the "is typing" notification stopped being displayed

Recording the incoming turns. Because text-chat is asynchronous, this means that at any point, while a participant is formulating a turn, it can happen that the other person sends a turn. This is recorded by the chat-tool. Suppose, a participant is typing "The yellow circle", but half-way through typing "yellow", the other person sends their turn, e.g. "now?", this could yield:

^{0}T ^{110}h ^{289}e ^{182} ^{348}y ^{800}e ^{110}l ^{307}【Participant1: now?】 ^{289}l ^{182}o ^{58}w ^{600} ^{500}c ^{490}i ^{300}r ^{289}c ^{100}l ^{300}e

This shows that 307 milliseconds after the participant typing the turn typed the second "l" of "yellow", the participant received the turn "now?" from Participant1.

TextAsFormulatedLOGSPACE

This prettifies the output of TextAsFormulatedTIMING. It removes the superscript numerals and replaces them with spaces (calculated logarithmically)- e.g. a gap of 100ms is one space. A gap of 1second is two spaces, a gap of 10 seconds is three spaces, a gap of 100 seconds is four spaces, etc.

istypingtimeout The parameter that determines how long the "is typing" indicator when a participant presses a key.

Data saved from the WYSIWYG Interface

The data from the WYSIWYG interface is more complex than data from the turn-by-turn interface. This is because in the WYSIWYG Interface "turns" are not clearly defined. The chattool saves the data in two formats:

Turns.txt file

The way turns.txt creates the transcript is the following: All characters typed by the participants are sent to the server, in order to be relayed to the other client(s). The server analyzes this stream of incoming characters.

1. Suppose at the start of the experiment participant 1 types a character. This is received by the server and treated by the server as the first character of Participant 1's turn.

2. participant 1 types more characters, all subsequent characters are treated as being part of the same turn.

3. If another participant (e.g. participant 2) sends a character, the server records a turn-change – now participant 2 "has the floor". All subsequent characters produced by participant 2 are treated as being part of participant 2's turn.

One problem with this approach is what to do with overlap. Imagine participant 1 types "it is my turn" while the other person types "stop talking", this could lead to this transcript:

A: now it is my t
B: s
A: u
B: t
A: r
B: o
A: p talking

This is not really human-readable and also, these "turns" don't really correspond to what participants themselves might treat as turns. But there is no rigorously principled way of *automatically* abstracting turns out of these streams of characters. A further problem with this approach is that it ignores network delay and doesn't necessarily reflect what participants themselves see on their screen. This is less of a problem with turn-by-turn interfaces, but with character-by-character interfaces it can be problematic. Consider if both participants type exactly the same text at the same time – due to inevitable network delay they will both perceive themselves typing the letters before their partners…So, to avoid this, the chattool records *on the clients what the participants saw*, and then sends that to the server, which then formats it to make it human-readable.

This means that there is now no longer one single objective view of the interaction. Each client records what the participant on that client saw! (in practice, if network latency is low this shouldn't make much of a difference). When generating the transcripts, for each client, it generates 5 files. The data from each pair is stored in four separate files which store the data associated with each keypress in a CSV format (with the "¦" character as separator):

To generate the WYSIWYG file format

The WYSIWYG output file looks like this. Compare the content of the red rectangle with the “turns.txt” transcript above. It is much clearer

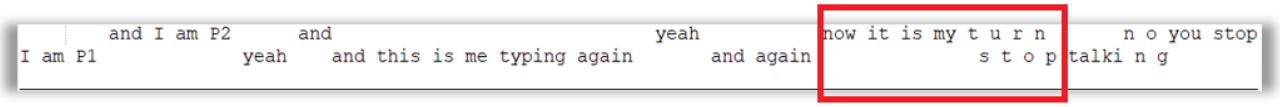


Figure 37 WYSIWYG Example transcript

To generate this transcript for WYSIWYG experiments you need to post-process the data after the experiment:

In the chattool, select the menu option “utils”, then select “WYSIWYGInterface: generate transcripts”

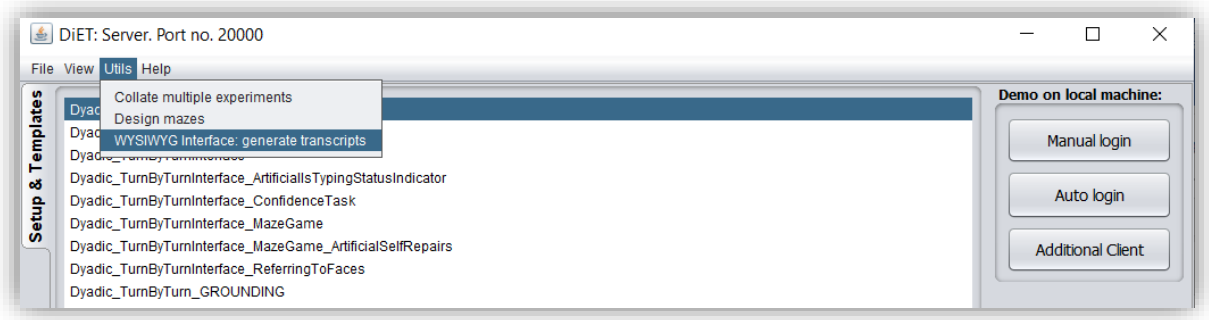


Figure 38 Generating WYSIWYG Transcript

Run it and select the folder(s) in /data/saved/experimental data/**FOLDER NAME(S)** where your data is. This should process the recorded data and create a set of new files. The two files you want are:

- TRANSCRIPT_POSTPROCwysiwyg_cie_***P1ID_P1USERNAME_P2ID_P2USERNAME***.txt (participant 1’s view of the interaction)
- TRANSCRIPT_POSTPROCwysiwyg_cie_***P2ID_P2USERNAME_P1ID_P1USERNAME***.txt (participant 2’s view of the interaction)

Where

- ***P1ID*** is Participant1’s ID
- ***P1USERNAME*** is Participant1’s Username
- ***P2ID*** is Participant2’s ID.
- ***P2USERNAME*** is Participant2’s Username

Important: make sure you load these files with a text-editor that has linewrap disabled! Otherwise the formatting won’t work, and it will look all jumbled up!

The next section describes how this transcript is created from the keypresses.

The WYSIWYG data captured on the clients:

Consider a hypothetical experiment between two participants

Participant 1: Participant ID: LLLL0 and Username: Michael
Participant 2: Participant ID: RRRR0 and Username: Jenny
The following files will be generated:

POSTPROCwysiwyg_cie_LLLL0_Michael_s_.txt
This file contains all the characters typed by Participant 1 (ID=LLLL0 with Username=Michael)
The “cie” in the filename stands for “clientinterfaceevents” and the “s” in the filename stands for “self”
The first few characters might be:
h¦e¦¦¦¦¦¦h¦e¦l¦l¦o¦w¦h¦y¦ ¦w¦e¦r¦e¦

POSTPROCwysiwyg_cie_LLLL0_Michael_o_RRRR0.txt
The file contains all the keypresses from the other participant in the pair, in this case the person with participant ID RRRR0 (The “o” in the filename stands for “other”)
The first few characters in this file might be:
¦¦h¦e¦l¦l¦o¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦

POSTPROCwysiwyg_cie_LLLL0_Michael_t_.txt
The file contains the time elapsed between each character and the preceding character (captured on the client interface – not captured on the server!)
(The “t” in the filename stands for “time”). You should ignore the first two values in this file.
The first few characters in this file are
¦10¦238¦11¦182¦7¦468¦447¦1851¦145¦131¦117¦183¦9586¦137¦94¦121¦152¦104¦152¦

POSTPROCwysiwyg_cie_LLLL0_Michael_w_.txt
This file contains information which interface was being used.
If the single line interface was being used, all values are “-“
If the double line interface was being used, all values are “-“ or “_”
- means that the text appeared on the top row.
_means that the text appeared on the bottom row.
The first few characters in this file are:
-¦-

TRANSCRIPTPOSTPROCwysiwyg_cie_LLLL0_Michael_o_RRRR0.txt
This is a readable transcript which is calculated from the other files in the following way:
Putting the files together (i.e. one file per row of text) gives this:

```
¦¦h¦e¦l¦l¦o¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
h¦e¦¦¦¦¦¦h¦e¦l¦l¦o¦w¦h¦y¦ ¦w¦e¦r¦e¦
10¦238¦11¦182¦7¦468¦447¦1851¦145¦131¦117¦183¦9586¦137¦94¦121¦152¦104¦152¦
-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-¦-
```

If padding is added so that the timestamps can be displayed, it gives:

```
    ¦    ¦h   ¦e   ¦l   ¦l   ¦o   ¦    ¦    ¦    ¦    ¦    ¦    ¦    ¦    ¦    ¦    ¦    ¦    ¦    ¦
h   ¦e   ¦    ¦    ¦    ¦    ¦    ¦h   ¦e   ¦l   ¦l   ¦o   ¦w   ¦h   ¦y   ¦    ¦w   ¦e   ¦r   ¦e   ¦
-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦-   ¦
10  ¦238 ¦11  ¦182 ¦7   ¦468 ¦447 ¦1851¦145 ¦131 ¦117 ¦183 ¦9586¦137 ¦94  ¦121 ¦152 ¦104 ¦152 ¦112 ¦
```

This is then used to generate the transcript, e.g.

```
  Hello        were
he     hellowhy
--------------------
```

N.B. Because of the fade-out of the interface, participants often don't use spaces if there is a longer pause.
E.g. there is no space character between "hello" and "why" in the example above.
This is because "hello" would have faded out before the participant typed "why".

Running an experiment

This takes you through the steps of running an experiment. Many of the steps are the same as in the quick start guide. They are duplicated here so that you don't need to scroll backwards and forwards in this document. Before running this make sure you have run the "quick start" instructions at the start of this document.

Starting the server and loading the intervention

Windows

Double-click on "start.bat". This should start the program.

Press the "start server" button (See Figure 1 below)

Mac / Linux

1. Double-click on "chattool.jar" . If this works, this means you have java installed.

2. If double-clicking doesn't work, open a Terminal window and navigate to the folder where "chattool.jar" is located, then type j**ava -jar "chattool.jar**

If that doesn't work, you need to install java. Follow the instructions on installing the latest version of java runtime for your computer, and then retry steps 1 and 2 listed above.

When you see the startup screen (Figure 10), select "start server".

In the main window, select the template from the list (1) and then select (5) "START"

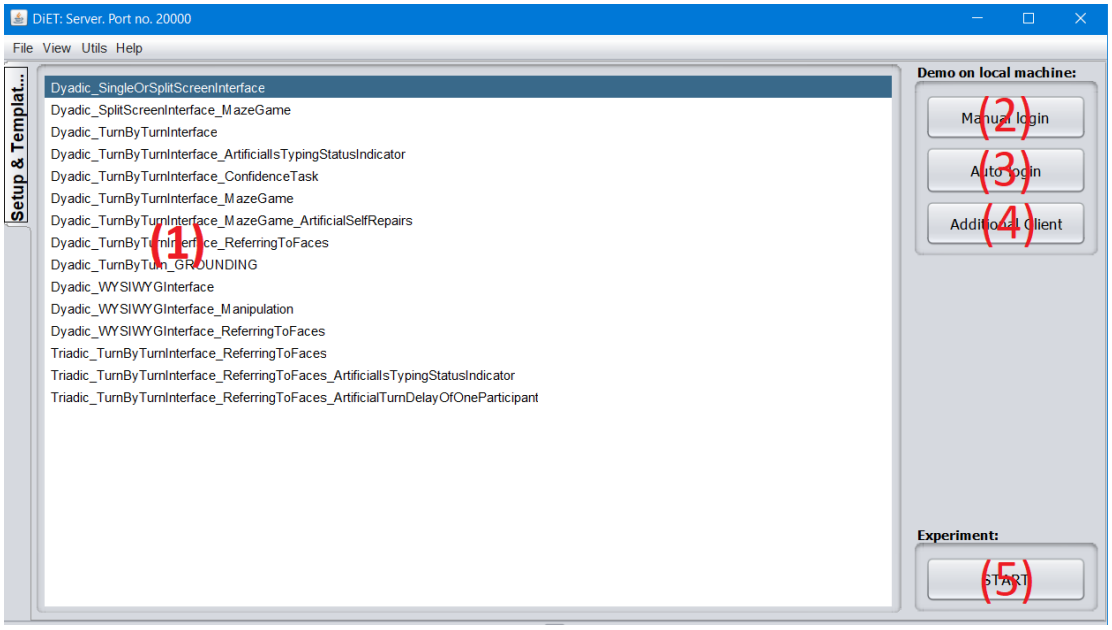


Figure 39 Main window for launching experiments

This will start the template. Now the server is waiting for the client to login.

Now you need to connect the clients to the server. The lower window of the server has the information you need:

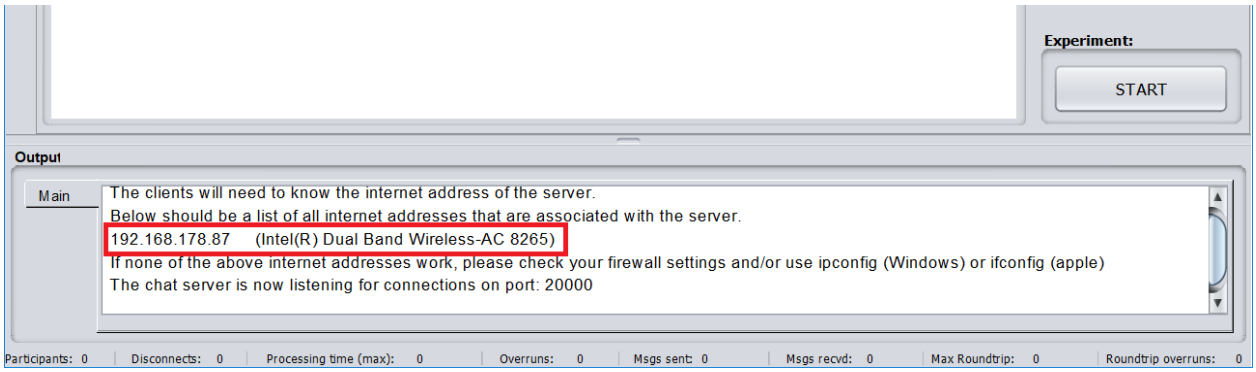


Figure 40 IP address of server that client needs to connect to

Make a note of this IP address.

Connecting the clients to the server

To connect to the server, copy the folder containing the chattool code to the clients.
Important – it is probably not a good idea to run the code from a USB stick. The reason is that if there is a blip where for a few milliseconds the operating system can't read/write to the USB stick, this could lead to the code crashing.

Windows

Double-click on "start.bat". This should start the program.

Mac / Linux

Double-click on "chattool.jar" or open a Terminal window and navigate to the folder where "chattool.jar" is located, then type j**ava -jar "chattool.jar**

This should open the startup screen. Now you need to tell the software what the IP address is of the server. You can find this IP address from the GUI of the server (see Figure 40 above) :

The server automatically tries to identify the IP address. Sometimes you might see more than one IP address (this is because each network adapter – ethernet, wifi, etc. typically has its own IP address). If this is the case, try each one to see which one works. In this example the IP address of the server is 192.168.178.87.

Next, enter this IP address into the client, as highlighted in below:

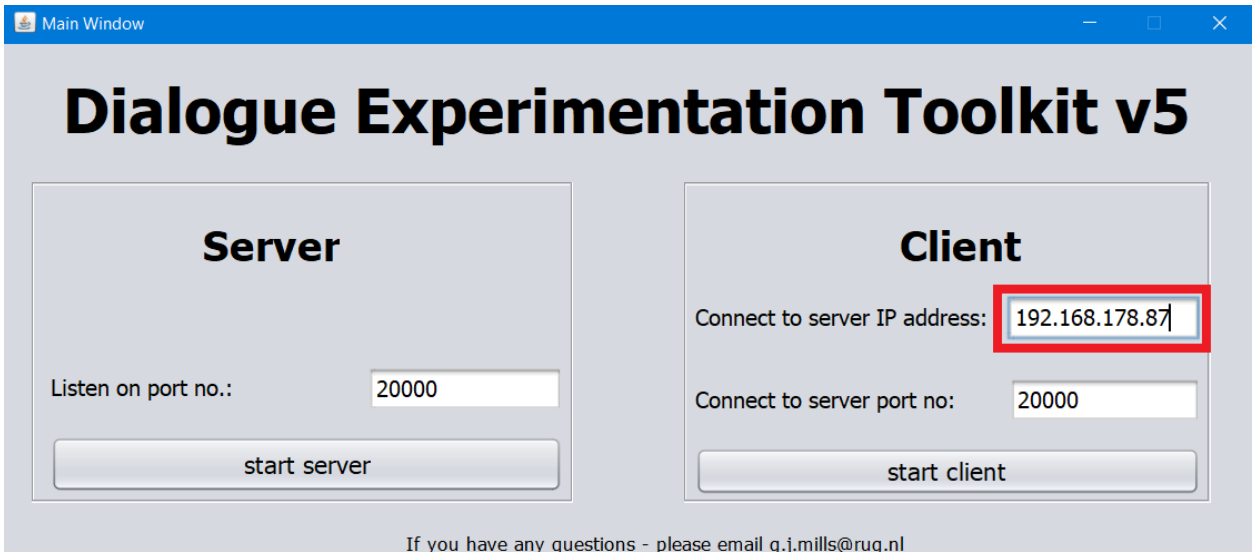


Figure 41 Starting the client – entering the IP address of the server

Then press "start client"

You should see a popup menu asking for a participant ID

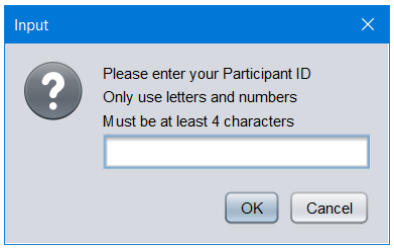


Figure 42 Starting the client – entering the Participant ID

Enter a participant ID (most of the demos allow any participant ID).
Then it should show the next step in the login process of requesting a username:

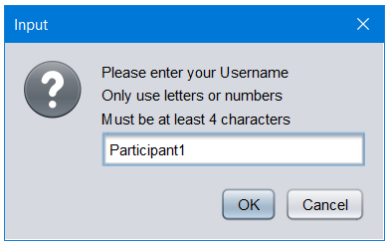


Figure 43 Starting the client Entering the username

Repeat this process on another computer – (choosing a different Participant ID and Username).

Speeding up the login process

It can get quite tedious entering the IP address manually each time. Once you know what the IP address is of the server, you can start the clients so that they automatically connect to the server's IP address, by opening a terminal window and typing

```
java -jar "chattool.jar" client IPADDRESS PORTNO
```

where IPADDRESS is the IP address of the server, and PORTNO is the port number (default 20000). On windows this would look like this:

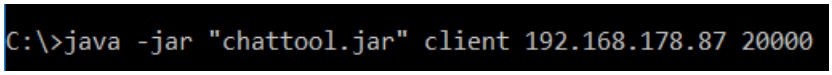


Figure 44 Starting the client – commandline instruction to connect to server

You can save this command to a batch file in windows / script in Mac / Linux and run it to start the server.

Running your own referential task (e.g. “tangram” joint reference task)

This section shows how to create custom interventions. The toolkit contains a template for specifying the stimuli in an experiment. To use your own stimuli you need a set of images (.jpg or .png) and a text file listing the sequence of stimuli to be presented to the participants. These need to be placed in the correct location. To explain how it works it is best to start using a set of stimuli that are included in the chattool.

Using included stimuli – the tangram task

First, start the server

Select the template “Dyadic_TurnByTurn_Customizable_ReferentialTask”

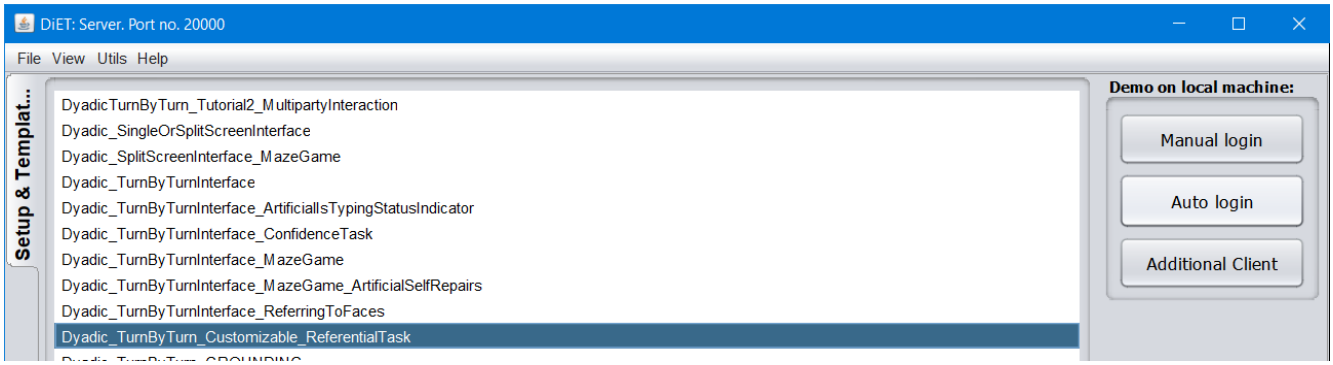


Figure 45 Selecting joint reference template

Then select “auto login”. (manual login will also work)

The template will ask a sequence of questions

1. How long should the stimuli be displayed for?
This option allows you to control how long the stimuli are displayed for. E.g. a value of 5000 means that at the start of each round the stimulus is displayed for 5000 milliseconds. Thus, on each round, after 5 seconds the participants need to describe stimuli from memory. If you want to make sure that the stimuli are displayed for the duration of each round, use a value that is longer than the duration of the round.
Select “60000”

2. Do you want to show buttons underneath the stimuli on the clients? Select
This template allows two different ways of letting users select referents:
(1) By pressing a corresponding button underneath the stimulus (See Figure 46, below)
(2) By entering a command in the chat interface, e.g. typing and sending “/1” to select the first item in an image.

The advantage of (1) is that it is more intuitive than (1). However (1) means that you give the Matcher a verbatim, necessarily closed list of items to select (Since the options have to be displayed on the buttons). In some experimental designs you might not want to enable this.
Select "Buttons"

3. Does the director receive feedback from the task about success/failure?
Most referential tasks give either the Director or Matcher or both, feedback about whether their selection was correct/incorrect.
Select *"Receives feedback"*

4. Does the matcher receive feedback from the task about success/failure?
Most referential tasks give either the Director or Matcher or both, feedback about whether their selection was correct/incorrect.
Select *"Receives feedback"*

5. How long is a game?
This asks how long (in milliseconds) participants can spend on any referent before there is a timeout out the experiment moves to the next referent.
Select the default value

6. Popup message
No input required – read and press OK.

7. File chooser menu
The file chooser menu should open the subfolder /experimentresources/stimuli/
You should see a list of directories, including

- facerealornot
- rorschachset01
- rorschachset02
- tangramset01

Each of these folders contains a different stimuli set.
Each folder also contains one or more text files that contain the order in which stimuli are presented, as well as what the correct answers are (see next section for a detailed explanation of how to customize this).
Navigate inside the "tangramset01" folder
Select "tangramsequence.txt"
Select "load the file"

8. Press OK to start
The chattool should have loaded the stimuli sequences .
Press OK

9. The experiment
The server loads the chat client interfaces as expected. It also opens three additional windows . Both participants have a stimuli window (Figure 46) and the server also has a window (Figure 48) which displays game-related information (And allows the experimenter to pause the experiment). Notice how the matcher has buttons underneath the stimuli. For game 1, the correct answer is "1", so *press the "1" button.*
You will now see the chattool give feedback that the selection was correct, and then move to the next image. Notice that on the subsequent game that the roles are swapped. The participant on the left is now the

matcher, and the participant on the right is the director. Also, now the participant on the left has the buttons (because this participant is the matcher).

ou can also let participants select items directly in the chat window: In the matcher's turn-formulation window *type "/5" and press ENTER*. This will select the 5th choice.

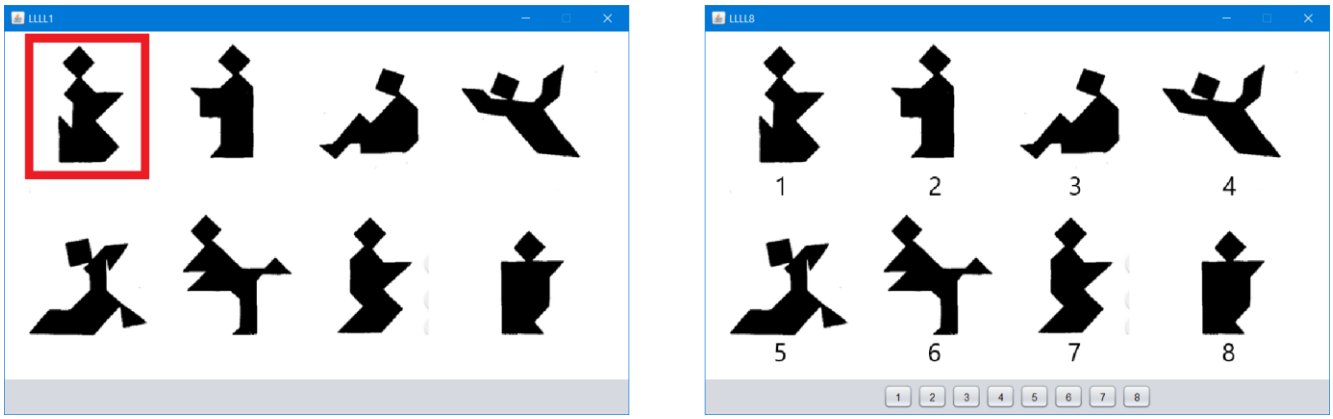


Figure 46 The stimuli windows of both participants. The left window shows the director. The right window shows the matcher. Note how the matcher has eight buttons underneath which can be pressed to make a selection.

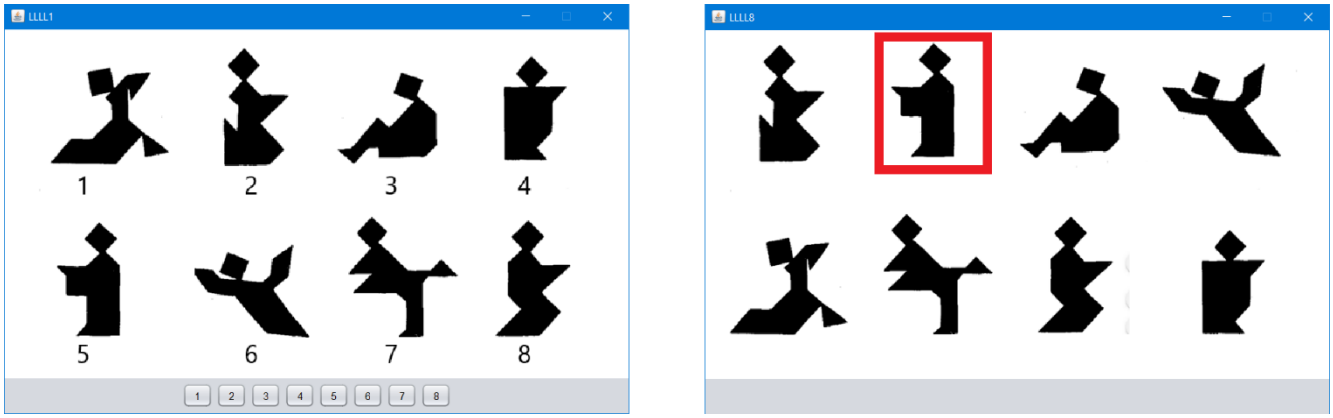


Figure 47 The stimuli window of both participants in game 2. Notice how the participant on the right is now the Director. Notice also how the participant on the right no longer has buttons - these are displayed on the matcher's screen on the left

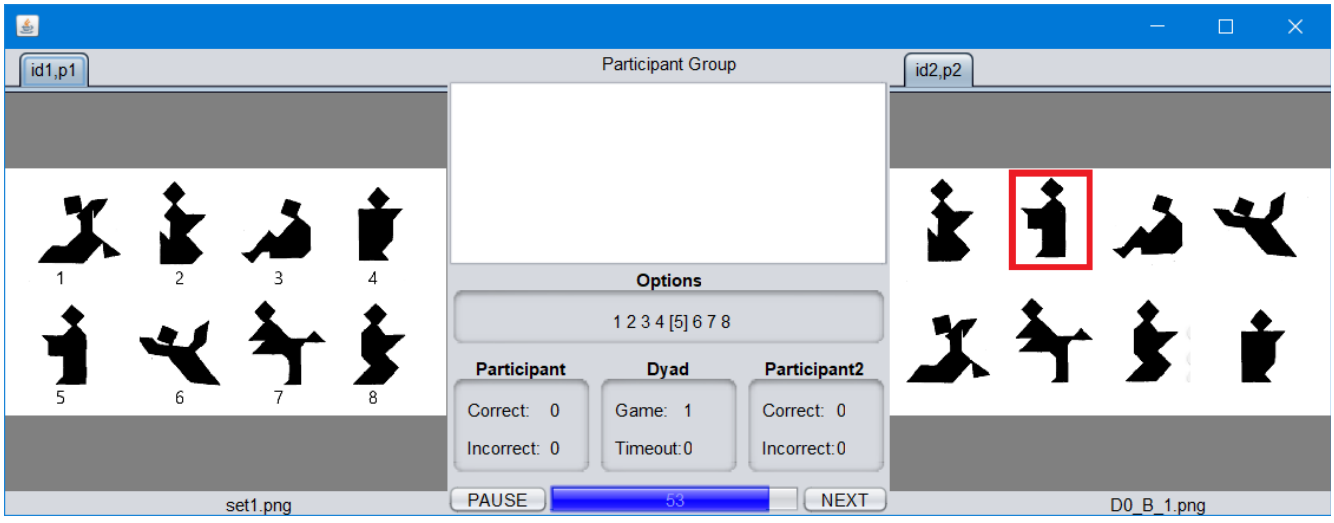


Figure 48 Server window for the experimenter to monitor game-related information. The top part has a pause button allowing the experimenter to pause the experiment..

After trying out this experiment, repeat this section, but instead of loading /experimentresources/stimuli/ tangramset01/tangramsequence.txt

Try the following:

Rorschach test:

/experimentresources/stimuli/rorschachset02/rorschachsequence01

In this setup each participant is presented with a single image. Half of the time they are presented with the same image. Half of the time they are presented with a different image. Their task is to work out whether they are looking at the same image or at a different image. Here the options are “S” for same and “D” for different.

Real vs. Fake faces:

/experimentresources/stimuli/facerealornot/face-fakeorreal.txt

In this task, both participants are shown the same image. Images are either of real people or of artificially generated faces. Their task is to decide, together, whether the faces are real or not. Note how in this game, both participants always see the same stimuli.

Scripting the sequence of stimuli

After trying out the tangram example from the previous section open the folder containing the stimuli in: /experimentresources/stimuli/tangramset01/

Notice how there are many separate sets of of stimuli for the director (prefixed with D) and for the matcher (set0 – set8.png). **The filenames are not important for the scripting**. The scripting is specified in a separate file. *Open the file /experimentresources/stimuli/tangramset01/tangramsequence.txt*

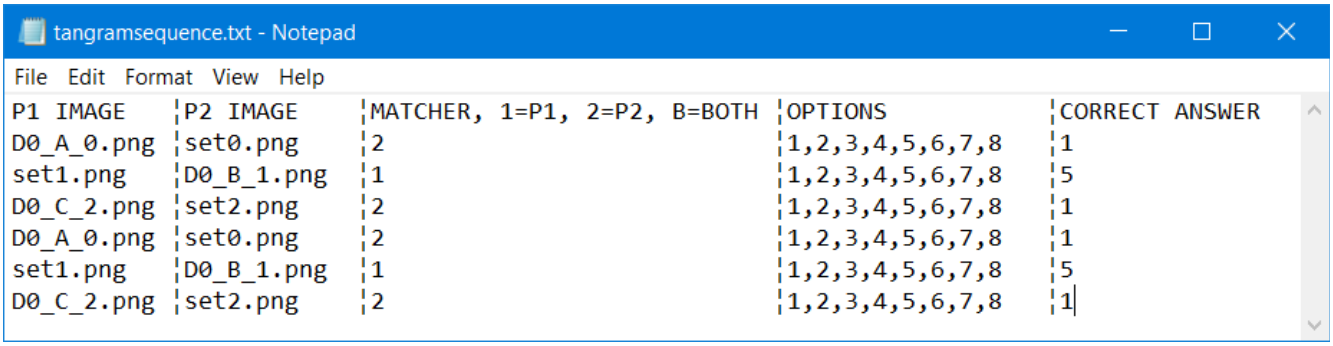


Figure 49 Text file specifying the sequence of stimuli

The stimuli scripts contain 5 columns. Each column is delimited by the "¦" character.
The fourth column is a list of items that are delimited by a comma, and the last item is not followed by a comma. Spaces are ignored by the algorithm – spaces are used in this script to make it easier to read.
The first row contains a description of the information contained in each column. Each row is a separate game.

- The first column specifies what image Participant 1 sees
- The second columns specifies what image Participant 2 sees
- The third column specifies which participant is the matcher.
    - A value of "1" means Participant 1 is the matcher.
    - A value of "2" means Participant 2 is the matcher.
    - A value of "B" means both participants can make selections.
- The fourth column specifies what the options are that the matcher can choose.
    - The options can be words, numbers (but no spaces!)
    - If buttons are enabled, these are displayed on the buttons.
    - N.B. This whitelist of options is essential. The reason is that if participants enter the options via the turnformulation window, e.g. selecting "/1" then the system has to be able to distinguish between incorrect and invalid choices - i.e. if a participant makes a typo and selects an option that doesn't exist, this shouldn't result in an incorrect selection.
- Correct answer.
    - If the participant selects this option it is recorded as the correct answer.
    - Note – the "correct answer" also has to be in the list of options.

Note: This setup controls for each game, which participant is the director and which is the matcher. It does this by distinguishing between P1 (first column) and P2 (second column). If you want a participant to be treated as P1 (i.e. to receive the sequence of images in the first column) assign that participant a participant ID that contains a "1" digit. For example if two participants login, one with participant ID "3456135" and the other with ID "657464", the first participant would be assigned to P1. Otherwise roles (P1 vs. P2) are randomly assigned.

Running a “tangram” joint reference task – using your own stimuli

To use your own stimuli you need to:

1. Make sure that all stimuli are in jpg or png format

2. Make sure that all stimuli have the same dimensions
(the chattool uses the width/height of the first image in the folder to determine the size of the stimuli window)

3. Create a subdirectory of /experimentresources/stimuli/ e.g. /experimentresources/stimuli/mynewstimuli
It is essential that you use this subdirectory.

4. Copy your stimuli into this folder

5. Create the text file containing the instructions for the experiment. Probably the easiest way is to copy an existing set of instructions and edit them. Copy
/experimentresources/stimuli/rorschachset02/rorschachsequence01.txt
to
/experimentresources/stimuli/rorschachset02/mynewstimuli/mynewstimulisequence.txt

6. Make sure you have read the explanation in

After trying out this experiment, repeat this section, but instead of loading
/experimentresources/stimuli/ tangramset01/tangramsequence.txt

Try the following:

Rorschach test:

/experimentresources/stimuli/rorschachset02/rorschachsequence01

In this setup each participant is presented with a single image. Half of the time they are presented with the same image. Half of the time they are presented with a different image. Their task is to work out whether they are looking at the same image or at a different image. Here the options are “S” for same and “D” for different.

Real vs. Fake faces:

/experimentresources/stimuli/facerealornot/face-fakeorreal.txt

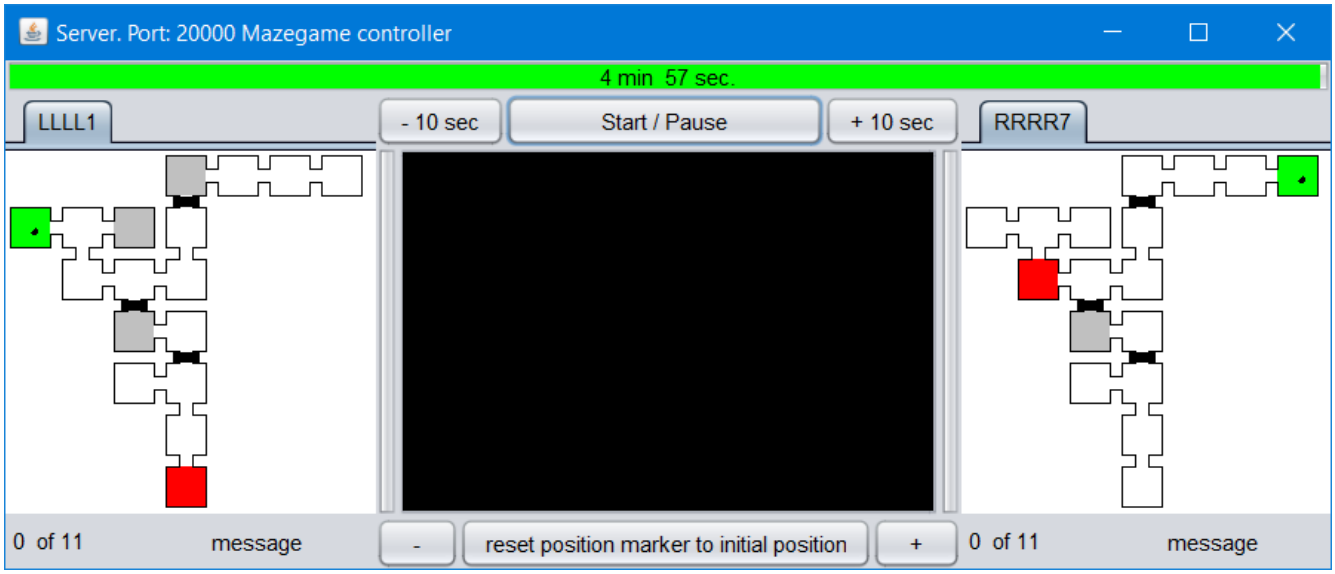
In this task, both participants are shown the same image. Images are either of real people or of artificially generated faces. Their task is to decide, together, whether the faces are real or not. Note how in this game, both participants always see the same stimuli.

The maze game

Running the maze game

Select one of the templates, e.g. **TurnByTurn_Dyadic_MazeGame**
Select **Auto login** (to run the template locally as a demo)
Select one of the sets of mazes, e.g. **mazeset_12_mazes**
Select the button "**Select this button when everyone has logged in"**

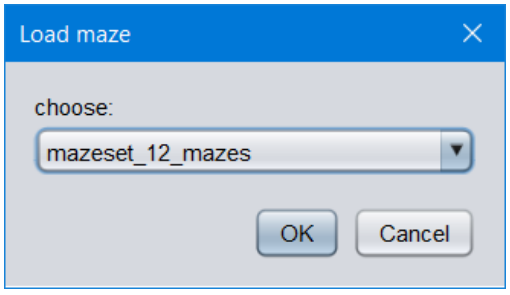
This should then open a screen as below:



This screen is used by the experimenter to observe/control the maze game window. The maze on the left is one participant's maze (Participant with ID LLLL1). The maze on the right is the other participant's maze (Participant with ID RRRR7). The black rectangle in the middle displays the conversation history of the dyad. Using Start/Pause and -10 and +10 sec it is possible to control the maximum time that is allocated to each round of the game.

Configuring the maze game

When starting the maze game you see a set of options:



These options are preconfigured sets of mazes.

If you look in the directory ../experimentresources/mazegame
You will see the corresponding directories, i.e.
../experimentresources/mazegame/mazeset_9_mazes
../experimentresources/mazegame/mazeset_12_easy_mazes

Each of these folders contains a maze configuration. So for example if you navigate to the folder:
../experimentresources/mazegame/mazeset_9_mazes

There are 2 sets of two files:
cl1mzes.txt Contains the maze configuration for one participant
cl2mzes.txt Contains the maze configuration for the other participant

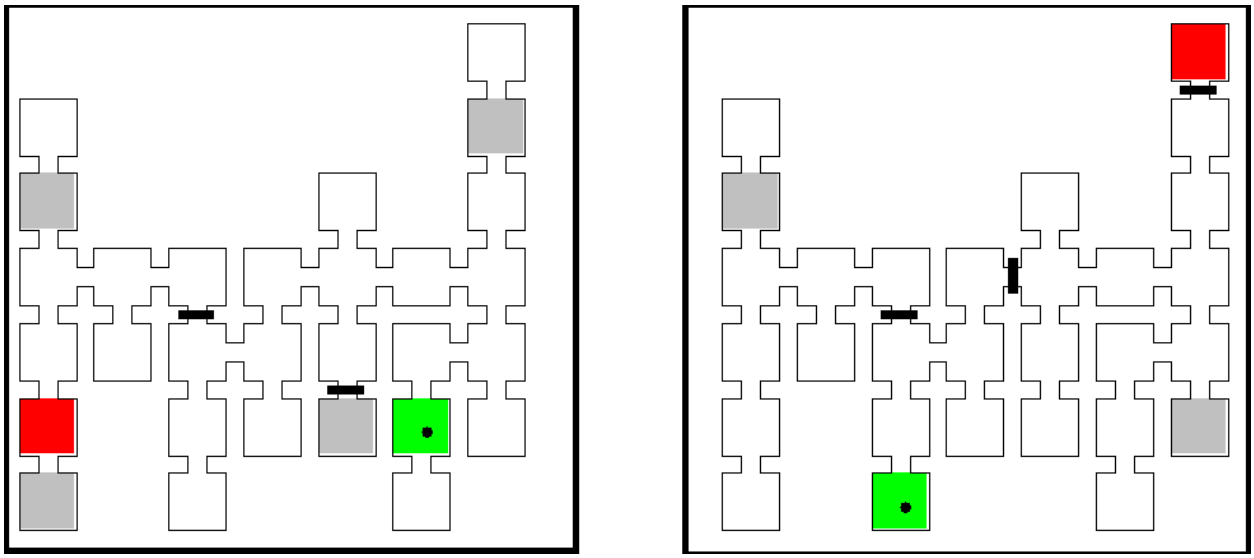
Each participant has a separate configuration files, because both participants have separate gates, switches and goals. (The files cl1mzes1.v and cl2mzes.v are serialized versions of the maze configurations and can be ignored.)

If you look e.g. inside **cl1mzes.txt** you can see:

MAZE No: 0
[-3 0]Link North Link South
[-3 -1]Link North is a gate. Link East Link South
[-3 -2]Link South is a gate. Link West
[-4 -2]Link East Link South Link West
[-4 -1]Link North
[-2 -1]Link North Link South Link West
[-2 -2]Link East Link South
[-1 -2]Link North Link East Link South Link West
[-1 -1]Link North Link South is a gate.
[-1 -3]Link South
[-1 0]SWITCH Link North is a gate.
[0 -2]Link East Link West
[1 -2]Link North Link South Link West
[1 -1]Link North Link South Link West
[0 -1]Link East Link South
[0 0]BEGIN Link North Link South
[-5 -2]Link North Link East Link South
[-2 0]Link North
[-5 -1]Link North Link South
[-5 0]FINISH Link North Link South
[-5 1]SWITCH Link North
[1 0]Link North
[1 -3]Link North Link South
[-3 1]Link North
[-5 -3]SWITCH Link North Link South
[-5 -4]Link South
[0 1]Link North

[1 -4]SWITCH Link North Link South
[1 -5]Link South

The maze described by this text is represented in the left maze of the figure below:



The file format is the following. Each of the numbers corresponds to a location in the grid. The location marked [0 0] is always the start location (i.e. where the position marker is initialized at the start of the game. [1 1] is one location below and to the right of the origin. For each location in the maze, the configuration records whether there is a link to to the node that is above (North), to the right (East), below (South) and to the left (West). It records whether there is a gate that blocks the link, and also records whether the location in the maze is a switch (i.e. a grey square).

How to read the configuration file:

[0 0]BEGIN specifies where the position marker is initialized at the start of the game.
In this case you can see [0 0]BEGIN Link North Link South. Notice how the green square in the attached figure has a connection above and below. The square below which is a "dead end" in the maze is described in the line "[0 1]Link North". The square above is "[0 -1]Link East Link South"

Creating new mazes

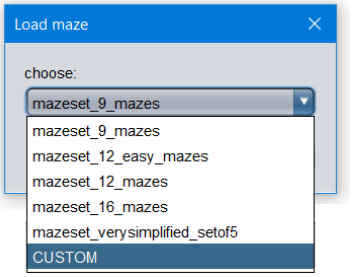
The steps below detail how to create a really simple “toy” set of mazes. If you want to create your own mazes, follow these steps below to familiarize yourself with the process. Then design your mazes on a grid on a piece of paper and follow this process to use the mazes in an experiment:

1. Create a new subdirectory, e.g.
/experimentresources/mazegame/mynewmazeset

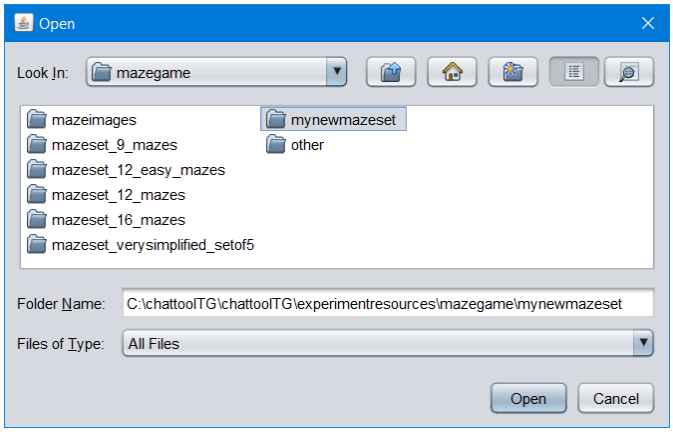
2. Create a file cl1mzes.txt containing
MAZE No: 0

[0 0]BEGIN Link South
[0 1]FINISH Link North

3. Create a file cl2mzes.txt containing
MAZE No: 0
[0 0]BEGIN Link North
[0 -1]FINISH Link South
4. Start the chattool and run one of the maze game experiments
5. Select "CUSTOM"



6. Select the folder containing the mazes



This should then load the new set of mazes.

Utility for viewing/creating mazes.

The chat tool does include a utility for semi-automatically creating mazes - and for previewing the mazes in the configuration files.

N.B. This interface was designed as a quick hack for a project 10 years ago – it has awful usability. It can be quicker to design mazes by hand (as described in the previous section)

If you load the chattool and select "Utils" and then "Design mazes" in the top menu bar, it should show the utility. There is an option "load from internal library" - if you select one of the names in the list e.g. "expmazes20" it will load the corresponding configuration. You can see the different mazes by selecting "maze number" . In this case there are 12 different mazes. When you select, e.g. "maze number 4", the left maze shows that maze for the first participant and the right maze shows the maze for the second participant.

This utilitiy is hard-coded to read configuration files from

/experimentresources/mazegame/

(N.B. because all the locations are defined relative to the start location, unless both participants start in the same location, both participants will have different coordinates for their mazes, even though they have the same underlying maze structure)

Now, when you run the maze game you should be able to load the simple maze configuration. If you want to make more complex mazes you can simply edit cl1mzes.txt and cl2mzes.txt

1. In the utility select the option "Load from external directory"
Specify the folder .. /experimentresources/mazegame/mazeset_9_mazes

This should load all the mazes into the utility and you should be able to view them.

The utility also allows you to create mazes (though it is probably easiest to sketch them on paper and then enter the coordinates directly in the text file).

To create a new configuration, press the "gene..." button (standing for "generate new maze pair"). Notice to the right of the "gene" button are two numbers 7 and 7 - these are the dimensions of the grid. "duplicate paths" is set to 0. This means that there are no cycles in the maze - there is only ever one route from one location to any other location (i.e. the mazes are effectively trees). if you increase this parameter (between 0 and 1) it increases the probability of creating a duplicate path for any link. The gates parameter specifies how many gates are created.

you can customize the maze configuration. Underneath each maze are four text textfields where you can enter the coordinates of locations in the maze.

So for example suppose you have a randomly generated configuration where there is a maze location underneath the start location - if you enter

0 0 / 0 1

and press "change link status" this will add a new gate between these two locations. If you press "change link status it will remove the gate.

If you enter:

0 1 /

and then select "switch" this will add a switch (a grey area) to [0 1] (this option ignores the second coordinate after the "/")

You can also specify the "finish" location (the square highlighted in red), as well as the start location. Note that if you specify the start location the coordinate scheme will change - all coordinates are relative to the start!

When you are done, press "save"

Just select the default save location. It will save in a subfolder (named with the unix epoch timestamp)

If you navigate to this folder you will see four files:

You can ignore cl1mzes.v and cl2mzes.v - these are legacy serialized versions of the mazes)

The text files

cl1Mazes.txt
cl2Mazes.txt

contain the maze configurations you created. To use them in an experiment follow the same instructions as above, i.e.

1. Create a new subdirectory, e.g.
 /experimentresources/mazegame/anotherdataset

2. Add "anotherdataset" to the list of options displayed
  when the maze game starts. I.e. in
  diet.task.MazeGameLoadMazesFromJarFile.MazeGameLoadMazesFromJarFile()

  add this line:

  setNames.addElement("anotherdataset")

3. Rename cl1Mazes.txt and cl2Mazes.txt to
  cl1mzes.txt and cl2mzes.txt
  and copy them into "anotherdataset"

4. Run the maze game and load the configuration "anotherdataset"