Table of Contents

About

Version of document: 12th May 2020

Before you start!

Remember, if you have any question at all – if anything doesn't make sense, either in the instructions or in the code – please email g.j.mills@rug.nl. The program is designed as a general tool for researchers, so any feedback concerning its use will really help us write better instructions.

Introduction:

The toolkit is a text-based chat tool for carrying out experiments on dialogue. The basic idea of the chat tool is that it 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.
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.

Equipment required to run the software

The following equipment is required to run experiments using the software:
3 or more computers, which can be any combination of Windows PC, Linux or Apple machines:

- One computer runs as server, and is used by the experimenter to observe the conversation. This computer runs the server software, which is typically scripted by the experimenter to record and manipulate participants' turns.
- Two or more computers run the client software. This program runs on each participant's computer. It provides an interface that displays the conversation and allows the participants to type text to each other.

Network: The clients must be able to connect to the server. The server has to be able to accept incoming connections on at least one port (you might need to ask systems admin. to "open" a port on the firewall). The default port setting is 20000. Possible (tried and tested) setups include.

- WIFI: using the existing infrastructure of the institution
- WIFI: participants bring their own laptops to the experiment (all laptops should be able to run the client software and connect to the server without having to install any new software)
- WIFI: The experimenter uses own WIFI Router / switch to create a wireless network for running the experiment. This has proved especially useful, not least because this makes it easy to block access to the wider internet, to ensure that participants can't read emails /or surf the web while taking part in the experiment.
- Ethernet: Participants sit at desktop computers in institution's computer laboratory)
- Mixed: any combination of WIFI / Ethernet should work as long as the Server can accept incoming connections from the clients
- From home: Participants download the chat client and run it on their own home computer
- On mobiles: We are currently programming a version that can run on participants' mobile phones.

Software required:

- To run an experiment the software needs to use Java runtime (This is usually installed as default on most computers). A version of java (for windows machines) is included in the installation zip file.
- To program the chat tool (see the “programmer’s guide” on github) , unless you use a different IDE, it is strongly recommended to use Netbeans as a platform for development. The source code is distributed as a self-contained Netbeans project with all the libraries included.

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

Recorded data

The chat tool automatically records the following data of each turn: the producer, the recipient(s), each word (frequency, recency), typing speed, number of edits , turn formulation time and typing overlap. This data is available at runtime to be used for generating experimental interventions that are sensitive to the immediate conversational context.

Integration with R/SPSS/Excel/LibreOffice

All the data from the experiments is saved to a CSV file (Comma Separated Values) that can be easily loaded into statistical software.

Naturalistic interventions

In order to create more plausible, naturalistic interventions, the chat tools allows the interventions to “spoof” participants’ typing speeds, and also introduce mistypings (typos) that are sensitive to the keyboard layout.

Interfaces

The chat tool includes a variety of customizable interfaces:

- An interface similar to existing instant messaging apps, e.g .whatsapp. Participants type text into a private window and press ENTER to send their turn.
- An interface similar to UNIX Talk that displays participants' text in separate windows.
- An incremental interface (WYSIWYG: What You See Is What You Get) that is especially useful for investigating incremental phenomena in dialogue.

Program versions and download

The chat tool is programmed entirely in Java, and runs on all (Linux, Windows, Mac) platforms.
It is released under a GPL license, and the latest version is available to download from github.

History

This is essentially the 5th incarnation of the chat tool. It grew out of the ROSSINI project (P.G.T. Healey Queen Mary University), was used by Matt Purver for his thesis, was rebuilt for Greg Mills's thesis, and then programmed as a stand-alone experimental resource as part of the DiET project (EPSRC). It has been modified and added to as part of the DynDial project (Kings College and Queen Mary University) and also as part of ERIS (FP7 EU Project, G.J Mills).

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"

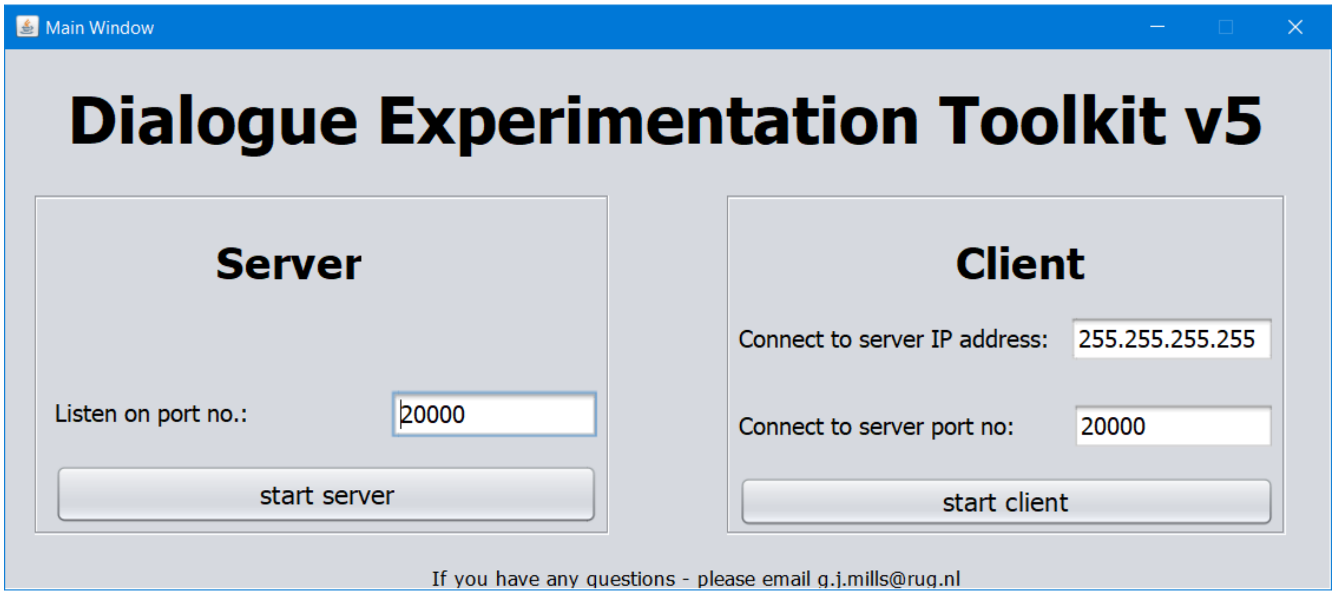

Main Window
Dialogue Experimentation Toolkit v5
Server
Listen on port no.:
20000
start server
Client
Connect to server IP address:
255.255.255.255
Connect to server port no:
20000
start client
If you have any questions - please email g.j.mills@rug.nl

Figure 1 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 1 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 1 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 9 (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 9, 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:

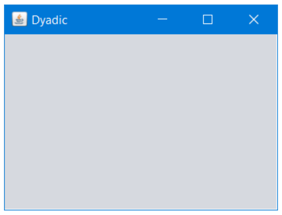
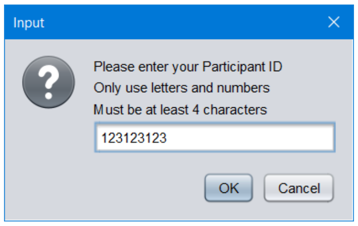


Figure 2 Window on server for observing what participants are typing in real-time

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:

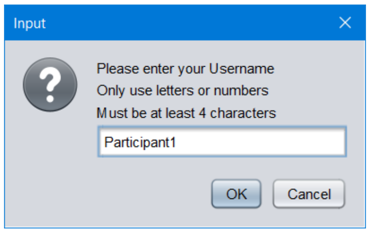


Please enter your Participant ID
Only use letters and numbers
Must be at least 4 characters
123123123

Figure 3 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:



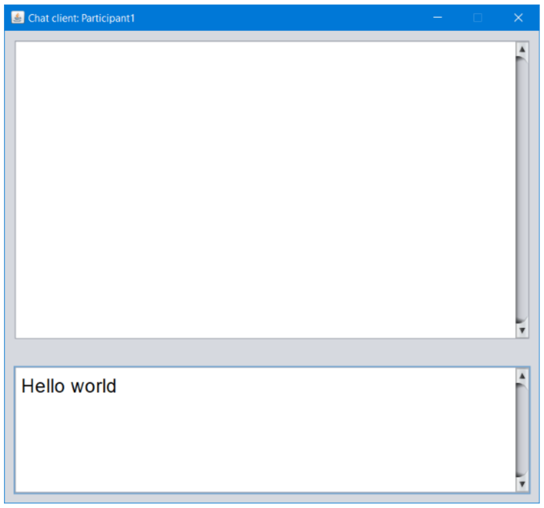
Please enter your Username
Only use letters or numbers
Must be at least 4 characters
Participant1

Figure 4 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 5 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).



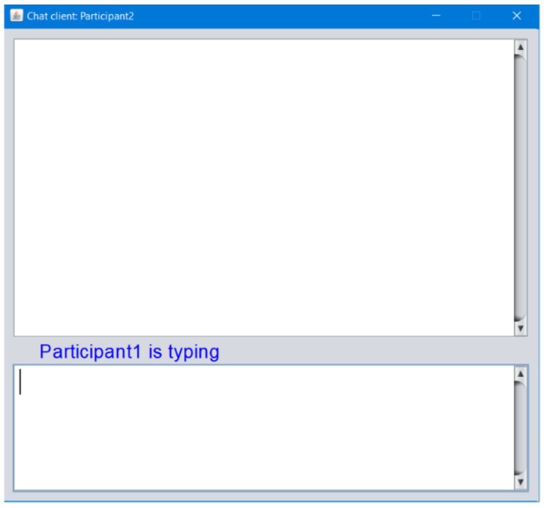


Figure 5 Simple Chat-interfaces. Each chat interface has an upper part that displays the conversation history, and a lower part where the participant types text. The title-bar of the window displays the username of the participant.

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 6). 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!)

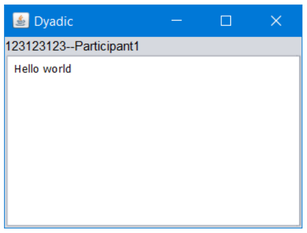
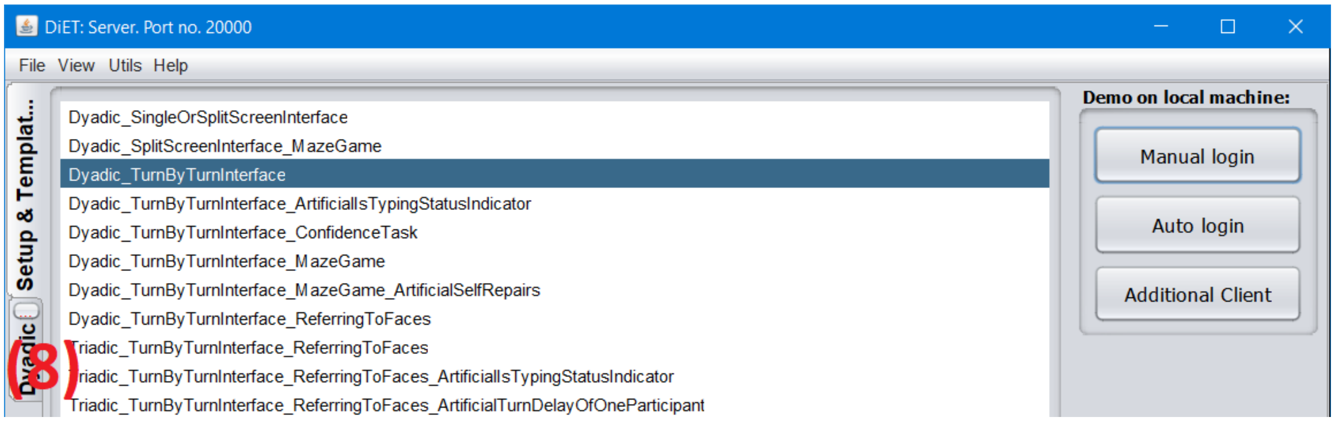


Figure 6 Server window showing participants' typing activity

Next, on the server, press the tab "Dyadic", marked (8) in Figure 7 and in Figure 10, to activate the "experimental control" interface.

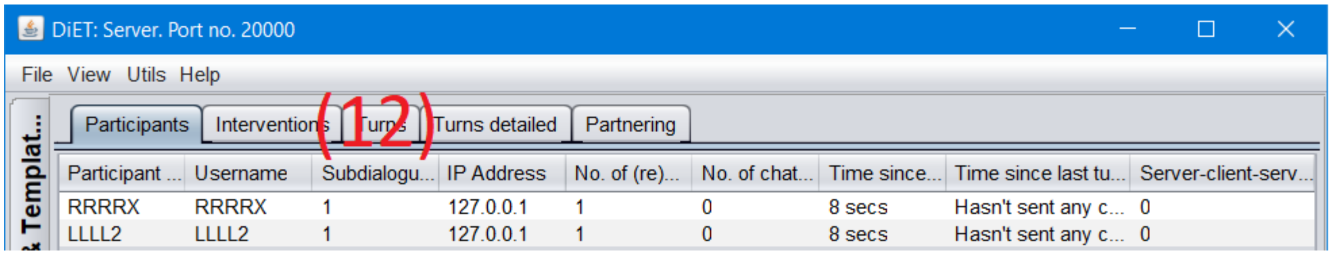


Dyadic_SingleOrSplitScreenInterface
Dyadic_SplitScreenInterface_MazeGame
Dyadic_TurnByTurnInterface
Dyadic_TurnByTurnInterface_ArtificialIsTypingStatusIndicator
Dyadic_TurnByTurnInterface_ConfidenceTask
Dyadic_TurnByTurnInterface_MazeGame
Dyadic_TurnByTurnInterface_MazeGame_ArtificialSelfRepairs
Dyadic_TurnByTurnInterface_ReferringToFaces
Triadic_TurnByTurnInterface_ReferringToFaces
Triadic_TurnByTurnInterface_ReferringToFaces_ArtificialIsTypingStatusIndicator
Triadic_TurnByTurnInterface_ReferringToFaces_ArtificialTurnDelayOfOneParticipant

Figure 7 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 8).



Participant ...	Username	Subdialogu...	IP Address	No. of (re)...	No. of chat...	Time since...	Time since last tu...	Server-client-serv...
RRRRX	RRRRX	1	127.0.0.1	1	0	8 secs	Hasn't sent any c...	0
LLLL2	LLLL2	1	127.0.0.1	1	0	8 secs	Hasn't sent any c...	0

Figure 8 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 “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.

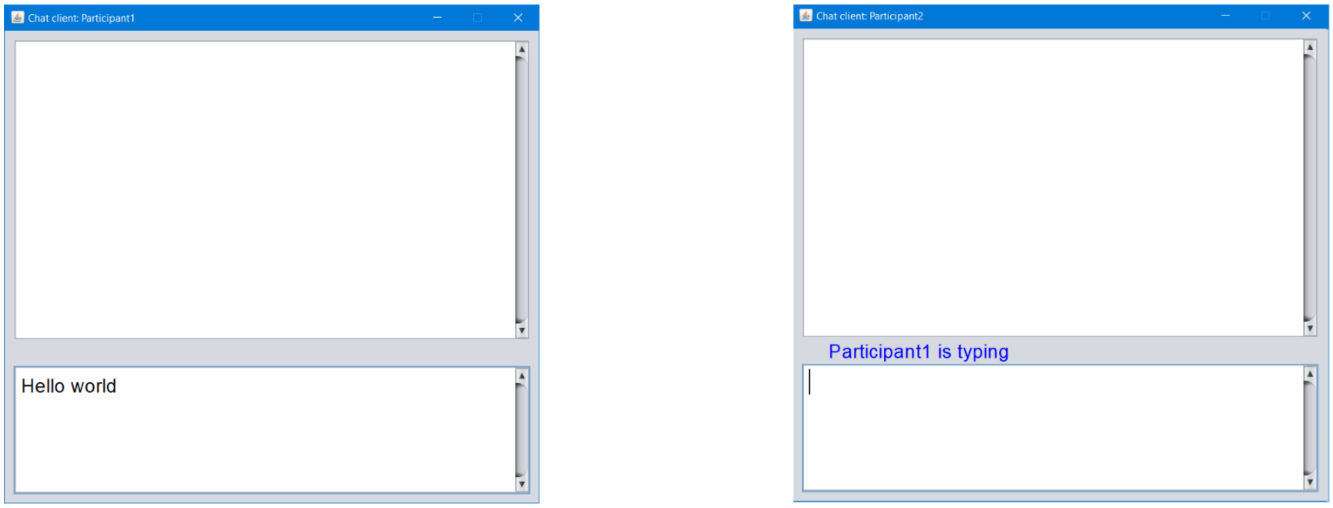


Figure 22 Turn By Turn Interface

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..

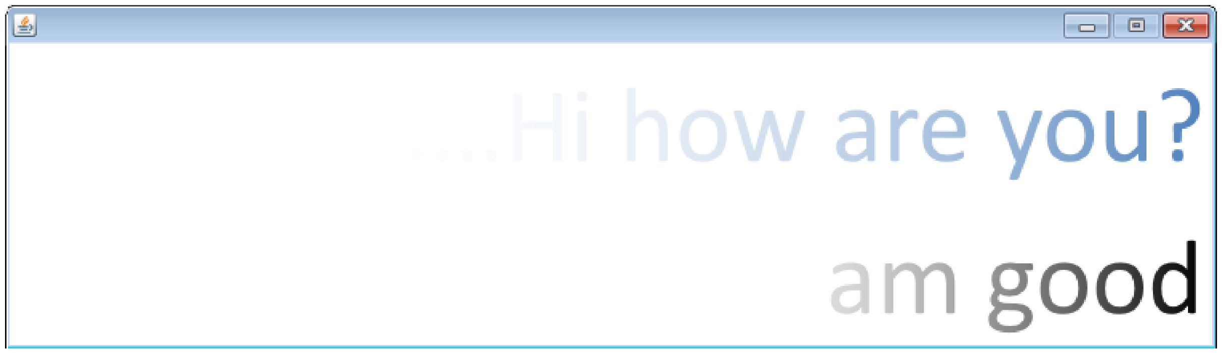


Figure 23 WYSIWYG Interface with each participant on a separate row

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

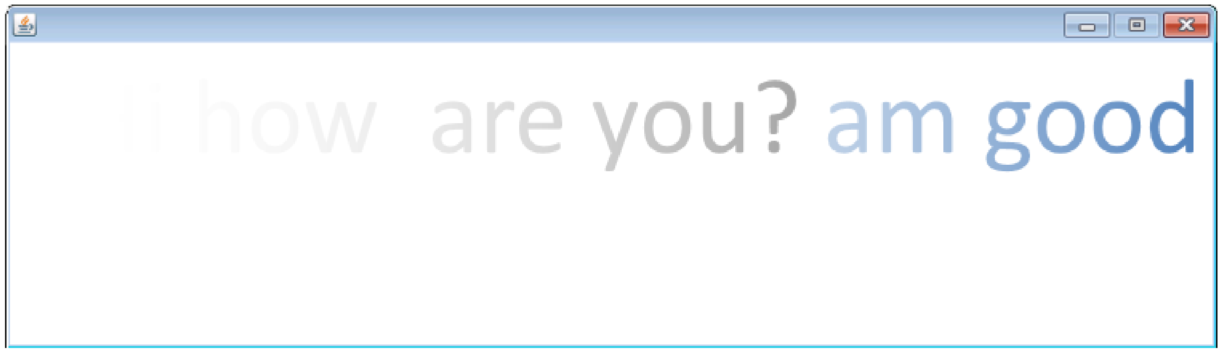


Figure 24 WYSIWYG Interface: Both participants on the same row

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:

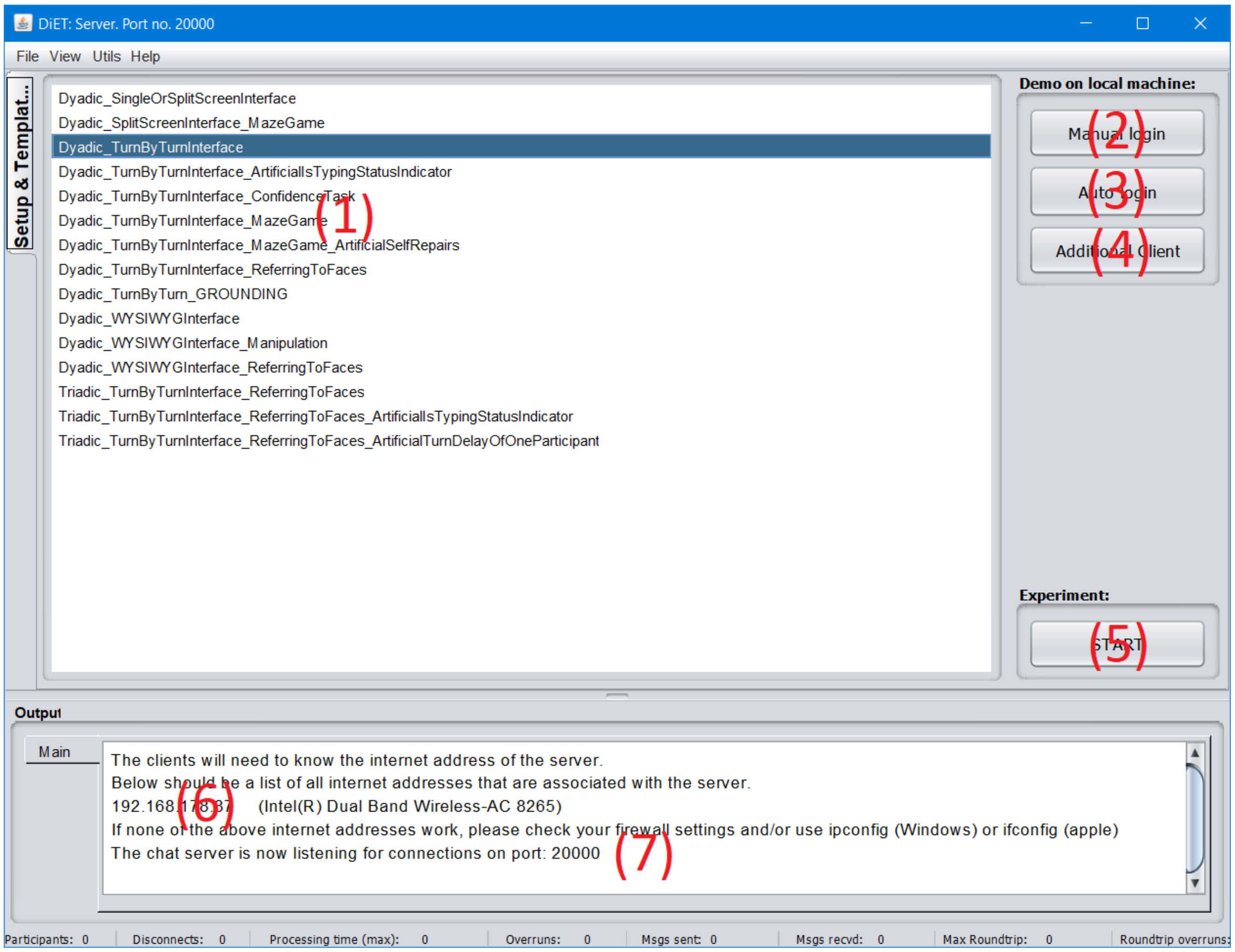
- Dyadic_WYSIWYGInterface   (the default implementation)
- Dyadic_WYSIWYGInterface_Manipulation
  (an experimental template that inserts fake clarification requests into the interaction)
- Dyadic_WYSIWYGInterface_ReferringToFaces (a joint reference experiment)

Data saved by the chattool on p. 27 below).

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

The server interface

Launching experiments



```
Dyadic_SingleOrSplitScreenInterface
Dyadic_SplitScreenInterface_MazeGame
Dyadic_TurnByTurnInterface
Dyadic_TurnByTurnInterface_ArtificialIsTypingStatusIndicator
Dyadic_TurnByTurnInterface_ConfidenceTask
Dyadic_TurnByTurnInterface_MazeGame
Dyadic_TurnByTurnInterface_MazeGame_ArtificialSelfRepairs
Dyadic_TurnByTurnInterface_ReferringToFaces
Dyadic_TurnByTurn_GROUNDING
Dyadic_WYSIWYGInterface
Dyadic_WYSIWYGInterface_Manipulation
Dyadic_WYSIWYGInterface_ReferringToFaces
Triadic_TurnByTurnInterface_ReferringToFaces
Triadic_TurnByTurnInterface_ReferringToFaces_ArtificialIsTypingStatusIndicator
Triadic_TurnByTurnInterface_ReferringToFaces_ArtificialTurnDelayOfOneParticipant
```

```
The clients will need to know the internet address of the server.
Below should be a list of all internet addresses that are associated with the server.
192.168.178.37     (Intel(R) Dual Band Wireless-AC 8265)
If none of the above internet addresses work, please check your firewall settings and/or use ipconfig (Windows) or ifconfig (apple)
The chat server is now listening for connections on port: 20000
```

```
Participants: 0 | Disconnects: 0 | Processing time (max): 0 | Overruns: 0 | Msgs sent: 0 | Msgs recvd: 0 | Max Roundtrip: 0 | Roundtrip overruns:
```

Figure 9 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.

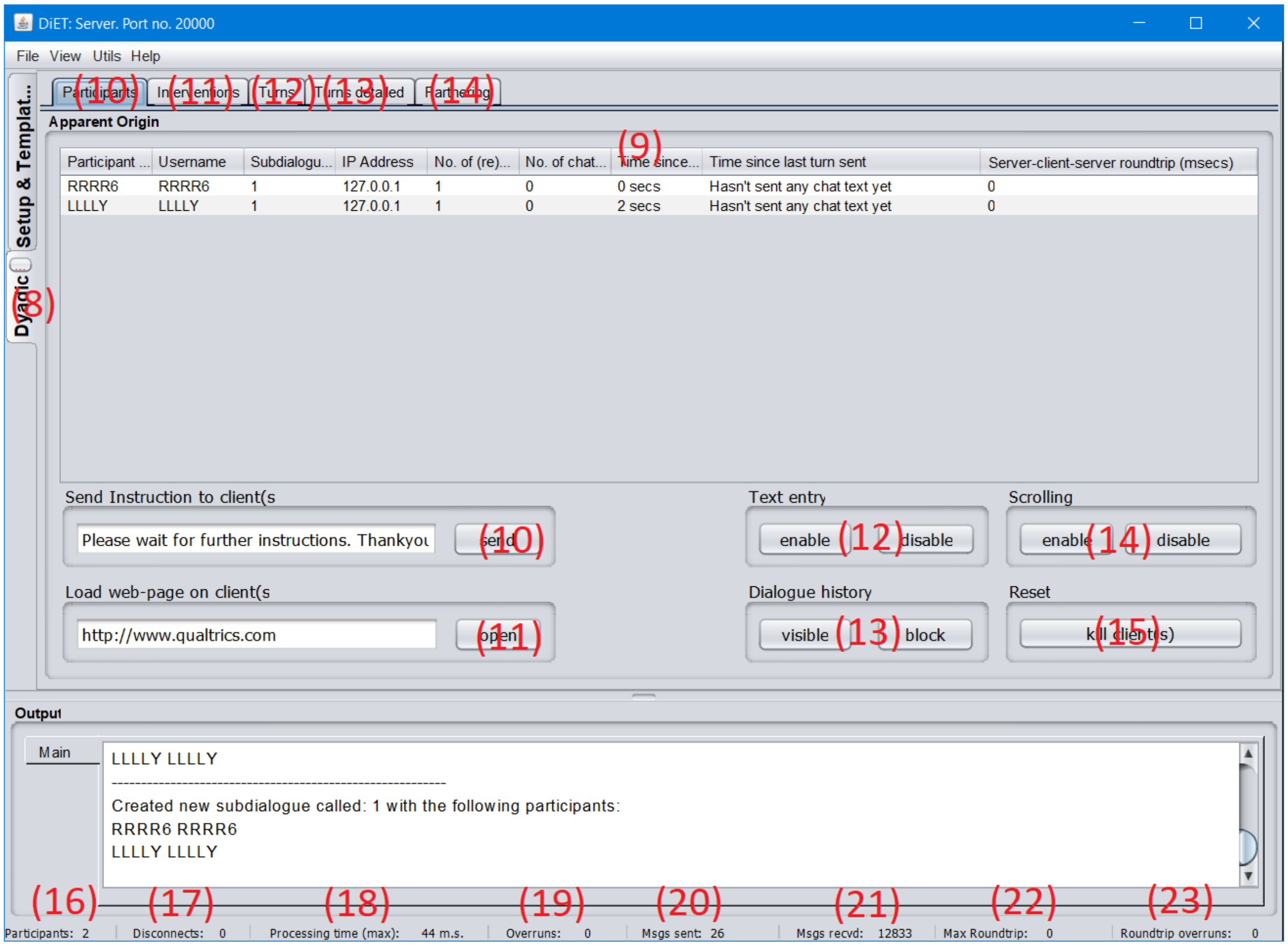
(4) This starts an additional client interface and connects to the experimental intervention. This is useful, e.g. if you are programming an intervention that can have a variable number of participants (E.g. a large group-chat). To add a new participant you just need to press this button.

(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 - Participants



Participant ...	Username	Subdialogu...	IP Address	No. of (re)...	No. of chat...	Time since...	Time since last turn sent	Server-client-server roundtrip (msecs)
RRRR6	RRRR6	1	127.0.0.1	1	0	0 secs	Hasn't sent any chat text yet	0
LLLLY	LLLLY	1	127.0.0.1	1	0	2 secs	Hasn't sent any chat text yet	0

Please wait for further instructions. Thankyou

http://www.qualtrics.com

```
LLLLY LLLLY
------------------------------------------------------------
Created new subdialogue called: 1 with the following participants:
RRRR6 RRRR6
LLLLY LLLLY
```

Participants: 2 | Disconnects: 0 | Processing time (max): 44 m.s. | Overruns: 0 | Msgs sent: 26 | Msgs recvd: 12833 | Max Roundtrip: 0 | Roundtrip overruns: 0

Figure 10 During an experiment

(8) Press this tab to activate the window above

(9) 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
- **Subdialogue:** 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 “subdialogue”.
- **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.

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

(11) 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 (12) and (13) can be used together, e.g. to make participants pay attention to another window (e.g. when filling out a questionnaire) (See Figure 1Figure 11). Note that this functionality only works with the "Turn By Turn" interface, not with the WYSIWYG Interface (see the section on interfaces on page 27)

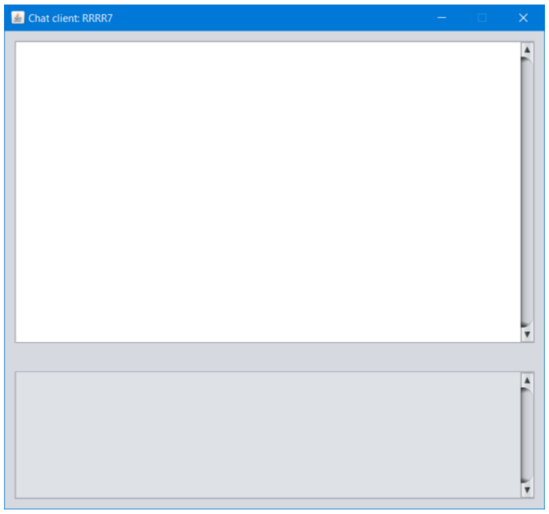


Figure 11 A chat client interface with text entry blocked

(13) 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 (12) and (13) can be used, e.g. to make participants pay attention to another window (e.g. when filling out a questionnaire) (see Figure 12). Note that this functionality only works with the "Turn By Turn" interface, not with the WYSIWYG Interface (see the section on interfaces on page 27)

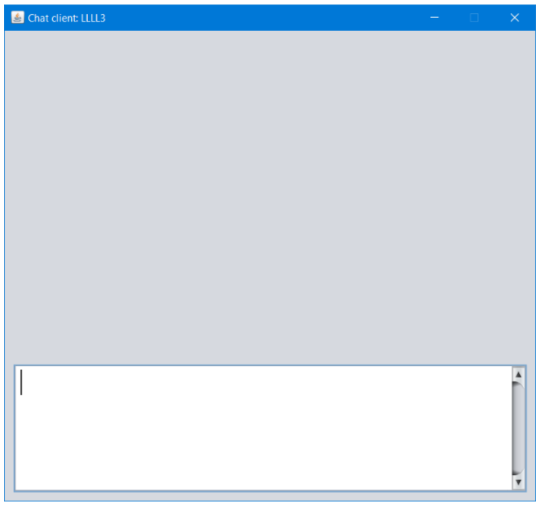


Figure 12 An interface with conversation history blocked

(14) 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 27).

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

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

(17) 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.

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

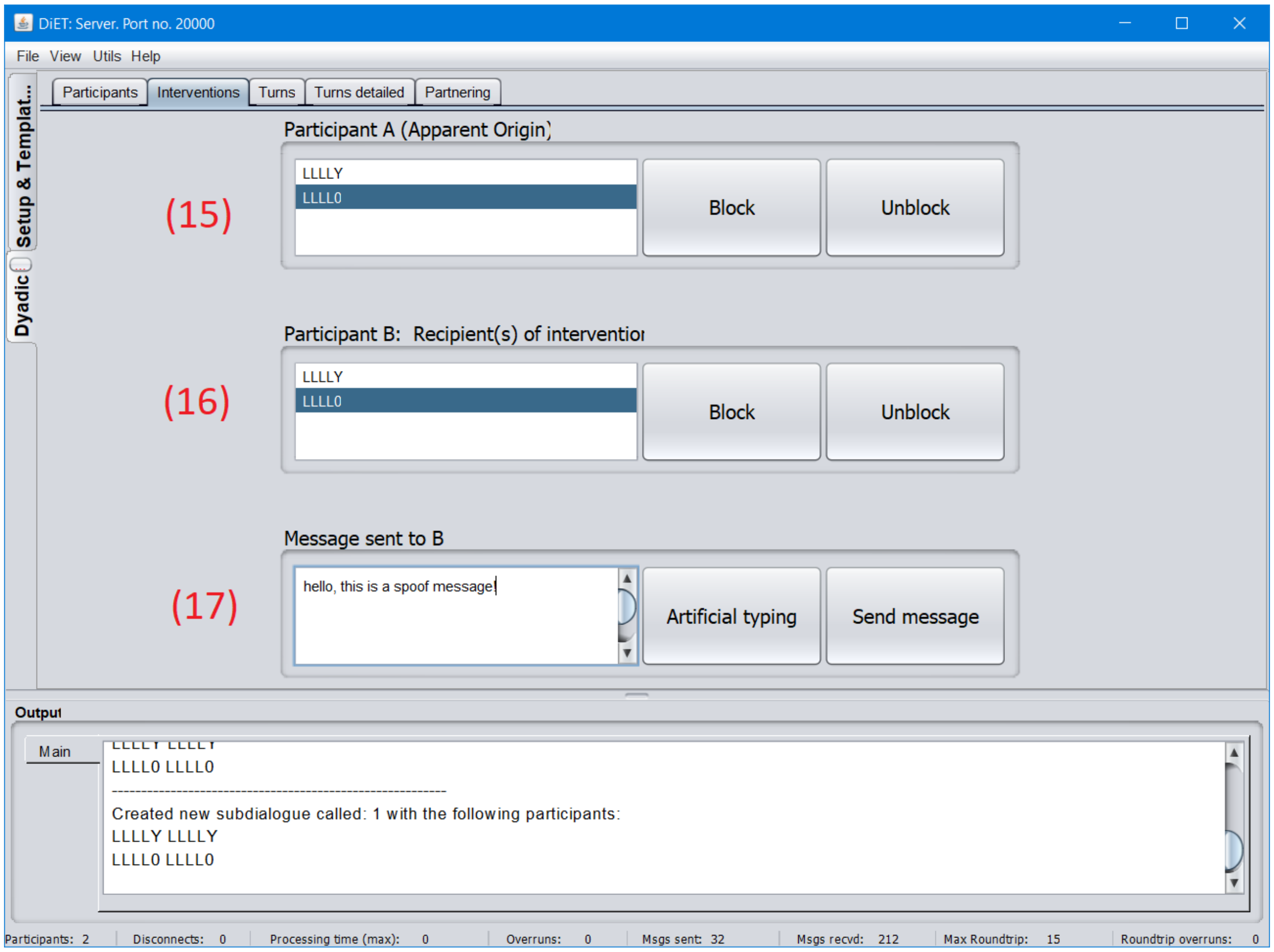
(19) Messages sent: This records how many messages were sent by the server to all clients. A message is not the same as a turn – messages can be pings, as well as instructions to show "is typing" , disable/enable textentry, etc.

(20) 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.

(21) 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 - Interventions



```
LLLLY LLLLY
LLLL0 LLLL0
--------------------------------------------------------
Created new subdialogue called: 1 with the following participants:
LLLLY LLLLY
LLLL0 LLLL0
```

```
Participants: 2 | Disconnects: 0 | Processing time (max): 0 | Overruns: 0 | Msgs sent: 32 | Msgs recvd: 212 | Max Roundtrip: 15 | Roundtrip overruns: 0
```

Figure 13 Generating Fake Turns

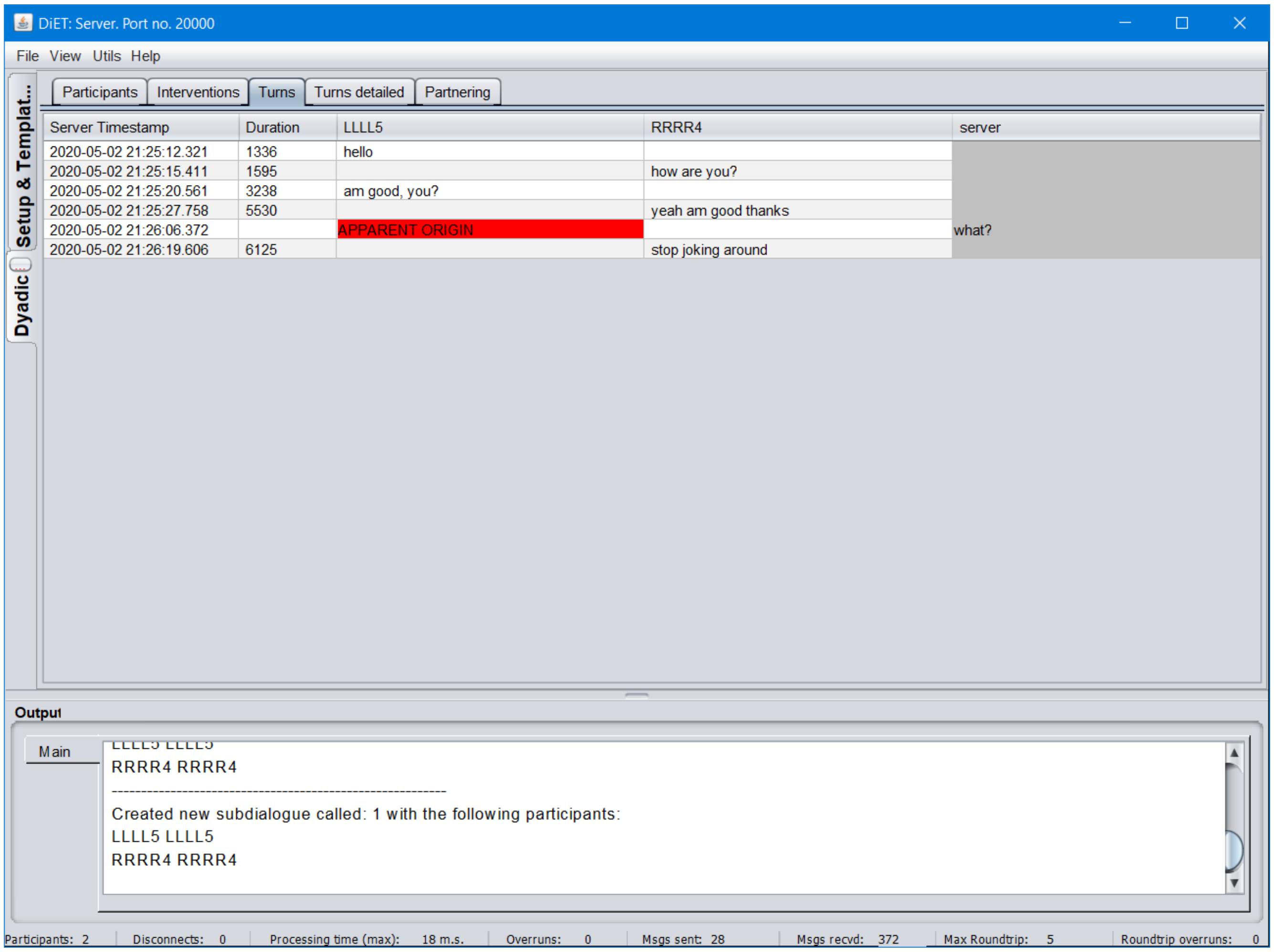
This is an interface for sending "spoofed" turns. The explanation below describes the steps involved in sending a fake turn to Participant B that appears to B, to originate from Participant A. The steps are typically:

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!" (17) to the participant with ID LLLL0 (16), which appears to come from the participant with ID LLLL0 (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 (17) before sending the turn (17). 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" (15). 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 (15,16).

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 - Turns



Server Timestamp	Duration	LLLL5	RRRR4	server
2020-05-02 21:25:12.321	1336	hello		
2020-05-02 21:25:15.411	1595		how are you?	
2020-05-02 21:25:20.561	3238	am good, you?		
2020-05-02 21:25:27.758	5530		yeah am good thanks	
2020-05-02 21:26:06.372		APPARENT ORIGIN		what?
2020-05-02 21:26:19.606	6125		stop joking around	

```
LLLL5 LLLL5
RRRR4 RRRR4
------------------------------------------------------------
Created new subdialogue called: 1 with the following participants:
LLLL5 LLLL5
RRRR4 RRRR4
```

```
Participants: 2   Disconnects: 0   Processing time (max): 18 m.s.   Overruns: 0   Msgs sent: 28   Msgs recvd: 372   Max Roundtrip: 5   Roundtrip overruns: 0
```

Figure 14 Conversation history

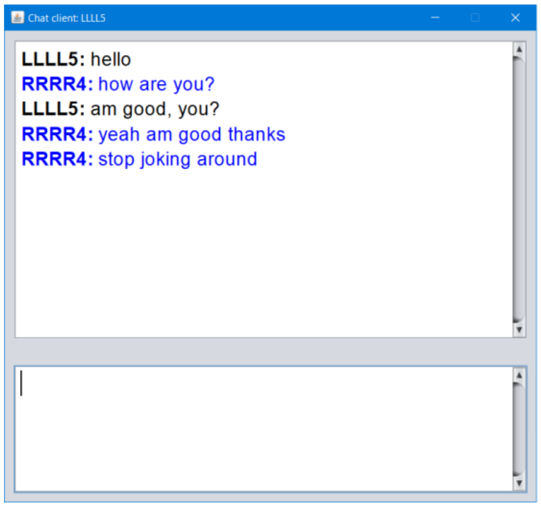
This interface displays the turns that were typed by the participants.

"**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.

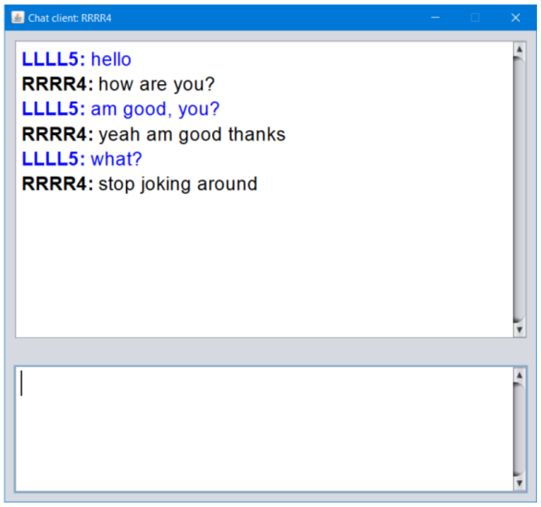
"**Duration**" displays how long it took the participant to formulate the turn, in milliseconds (this is the duration from the first keypress till when they pressed enter to send the message).

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 15 Figure 15 Dialogue with fake inserted text "what?" by the serverbelow to see what this looks like on the participants' screens.


Chat client: LLLL5

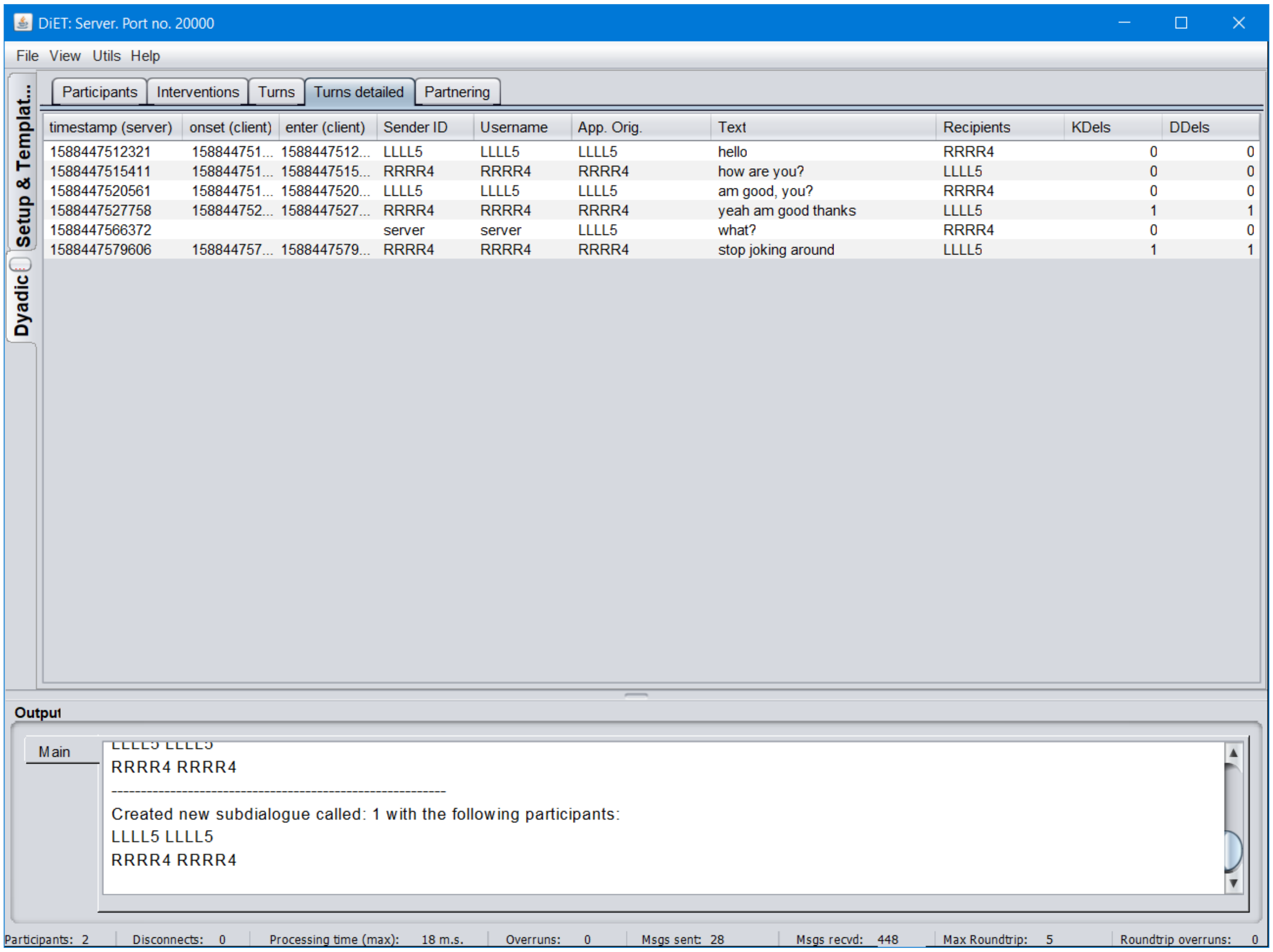
LLLL5: hello
RRRR4: how are you?
LLLL5: am good, you?
RRRR4: yeah am good thanks
RRRR4: stop joking around


Chat client: RRRR4

LLLL5: hello
RRRR4: how are you?
LLLL5: am good, you?
RRRR4: yeah am good thanks
LLLL5: what?
RRRR4: stop joking around

Figure 15 Dialogue with fake inserted text "what?" by the server

While running an experiment - Turns detailed



timestamp (server)	onset (client)	enter (client)	Sender ID	Username	App. Orig.	Text	Recipients	KDels	DDels
1588447512321	158844751...	1588447512...	LLLL5	LLLL5	LLLL5	hello	RRRR4	0	0
1588447515411	158844751...	1588447515...	RRRR4	RRRR4	RRRR4	how are you?	LLLL5	0	0
1588447520561	158844751...	1588447520...	LLLL5	LLLL5	LLLL5	am good, you?	RRRR4	0	0
1588447527758	158844752...	1588447527...	RRRR4	RRRR4	RRRR4	yeah am good thanks	LLLL5	1	1
1588447566372			server	server	LLLL5	what?	RRRR4	0	0
1588447579606	158844757...	1588447579...	RRRR4	RRRR4	RRRR4	stop joking around	LLLL5	1	1

```
LLLL5 LLLL5
RRRR4 RRRR4
------------------------------------------------------------
Created new subdialogue called: 1 with the following participants:
LLLL5 LLLL5
RRRR4 RRRR4
```

Participants: 2 | Disconnects: 0 | Processing time (max): 18 m.s. | Overruns: 0 | Msgs sent: 28 | Msgs recvd: 448 | Max Roundtrip: 5 | Roundtrip overruns: 0

Figure 16 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,

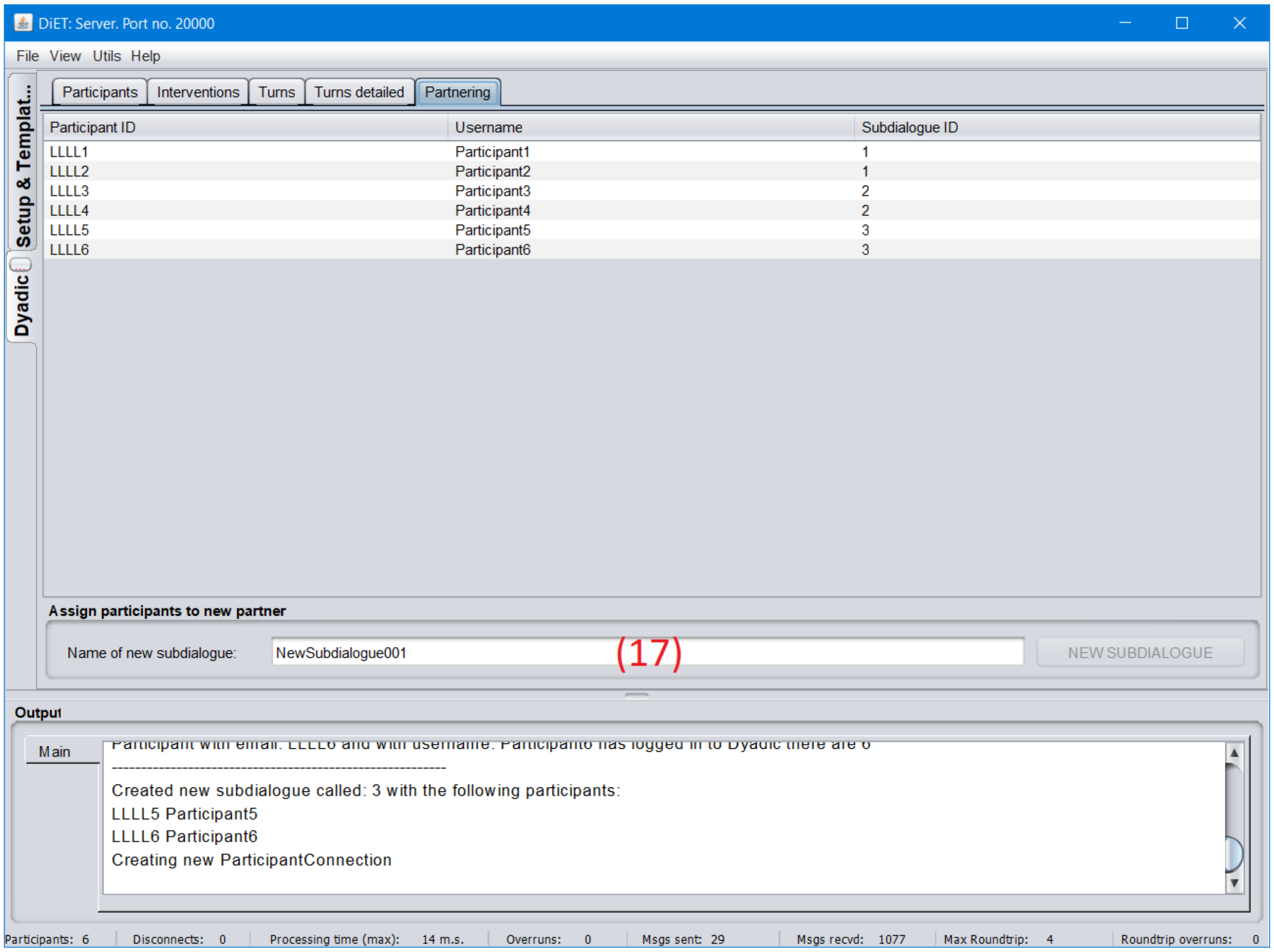
if **timestamp (server)** and **enter (client)** have the same value, this is no guarantee that they are in fact simultaneous. However, if **enter (client)** is 4000ms more than **onset (client)**, this does mean that the turn almost certainly took 4 seconds to formulate.

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 - Participant Partnering



Participant ID	Username	Subdialogue ID
LLLL1	Participant1	1
LLLL2	Participant2	1
LLLL3	Participant3	2
LLLL4	Participant4	2
LLLL5	Participant5	3
LLLL6	Participant6	3

```
Participant with email: LLLL6 and with username: Participant6 has logged in to Dyadic there are 6
------------------------------------------------------------
Created new subdialogue called: 3 with the following participants:
LLLL5 Participant5
LLLL6 Participant6
Creating new ParticipantConnection
```

```
Participants: 6 | Disconnects: 0 | Processing time (max): 14 m.s. | Overruns: 0 | Msgs sent: 29 | Msgs recvd: 1077 | Max Roundtrip: 4 | Roundtrip overruns: 0
```

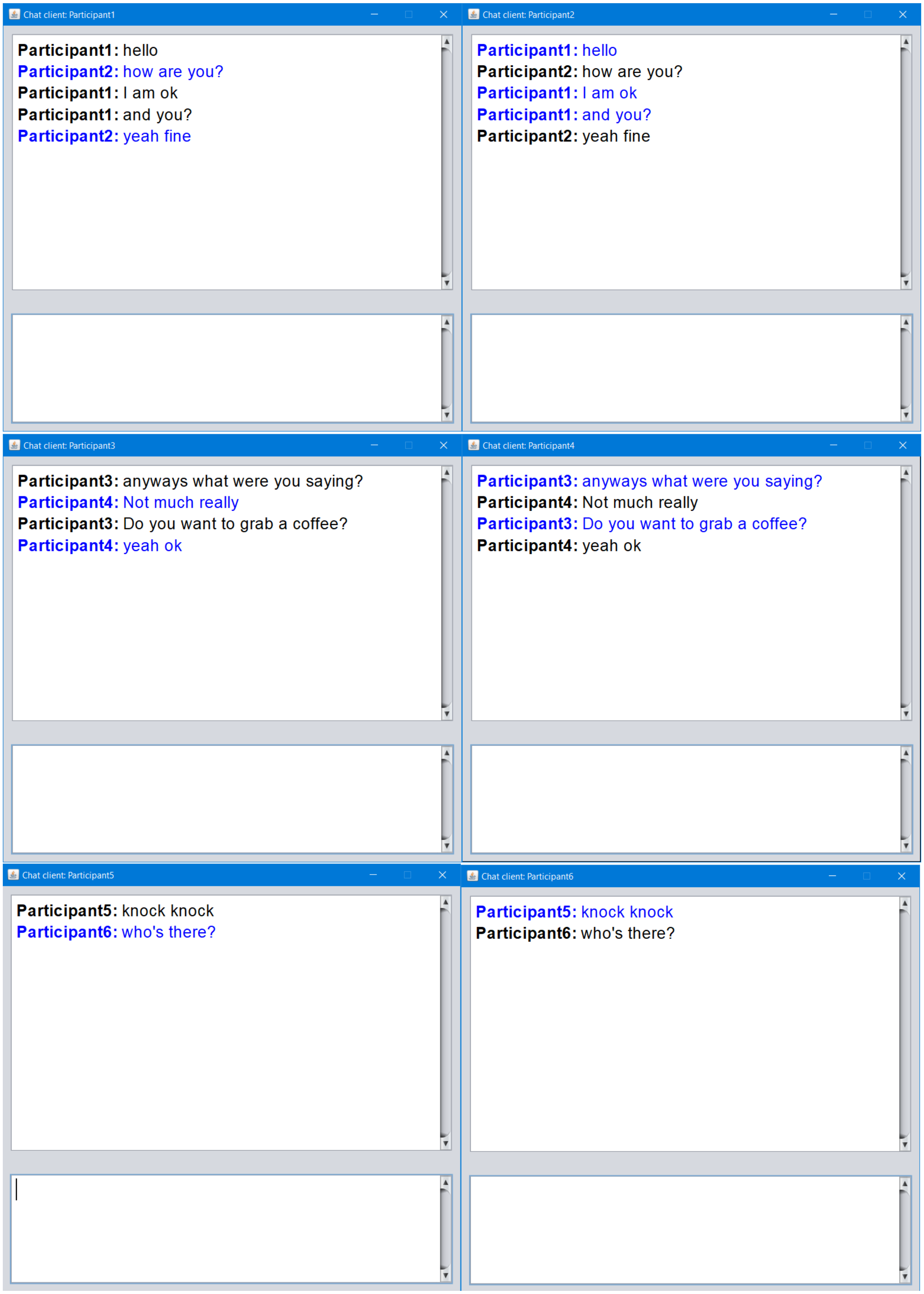
Figure 17 Participant subdialogues

This interface allows you to see who is speaking with whom.

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

The column "Subdialogue ID" shows who is speaking with whom. Particpants with the same subdialogueID are interacting with each other. In this table it shows that Participant1 and Participant2 have subdialogueID 1, Participant2 and Participant3 have subdialogue ID 2, and Participant 5 and Participant 6 have Subdialogue ID 3. In other words, this shows that there are three subdialogues (i.e. there are three dyads) having separate conversations. See Figure 18 below

(17) This allows the experimenter to dynamically assign participants to different partners. At the moment Figure 9 and Figure 10 show three separate dyads. Figure 19 and Figure 20 (below) show how to re-assign participants to different partners.
* N.B. *The term "sub-dialogue" is somewhat ambiguous. This term was chosen instead of "conversation" because the chat-tool allows the constitution of these subdialogues to be experimentally manipulated while participants are interacting. "Conversation" implies something more static.*


Chat client: Participant1

Participant1: hello
Participant2: how are you?
Participant1: I am ok
Participant1: and you?
Participant2: yeah fine

Chat client: Participant2

Participant1: hello
Participant2: how are you?
Participant1: I am ok
Participant1: and you?
Participant2: yeah fine

Chat client: Participant3

Participant3: anyways what were you saying?
Participant4: Not much really
Participant3: Do you want to grab a coffee?
Participant4: yeah ok

Chat client: Participant4

Participant3: anyways what were you saying?
Participant4: Not much really
Participant3: Do you want to grab a coffee?
Participant4: yeah ok

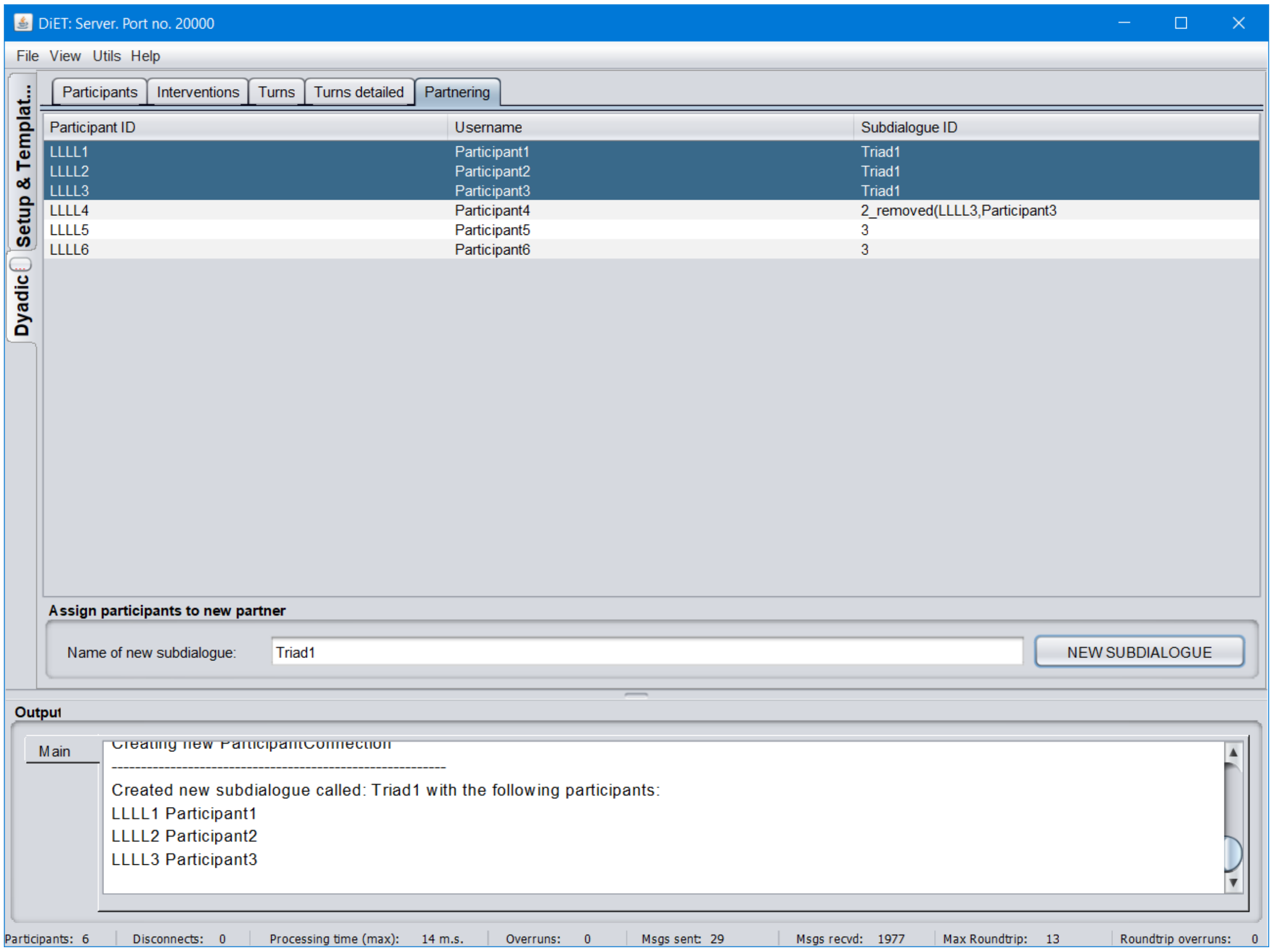
Chat client: Participant5

Participant5: knock knock
Participant6: who's there?

Chat client: Participant6

Participant5: knock knock
Participant6: who's there?

Figure 18 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.



Participant ID	Username	Subdialogue ID
LLLL1	Participant1	Triad1
LLLL2	Participant2	Triad1
LLLL3	Participant3	Triad1
LLLL4	Participant4	2_removed(LLLL3,Participant3
LLLL5	Participant5	3
LLLL6	Participant6	3

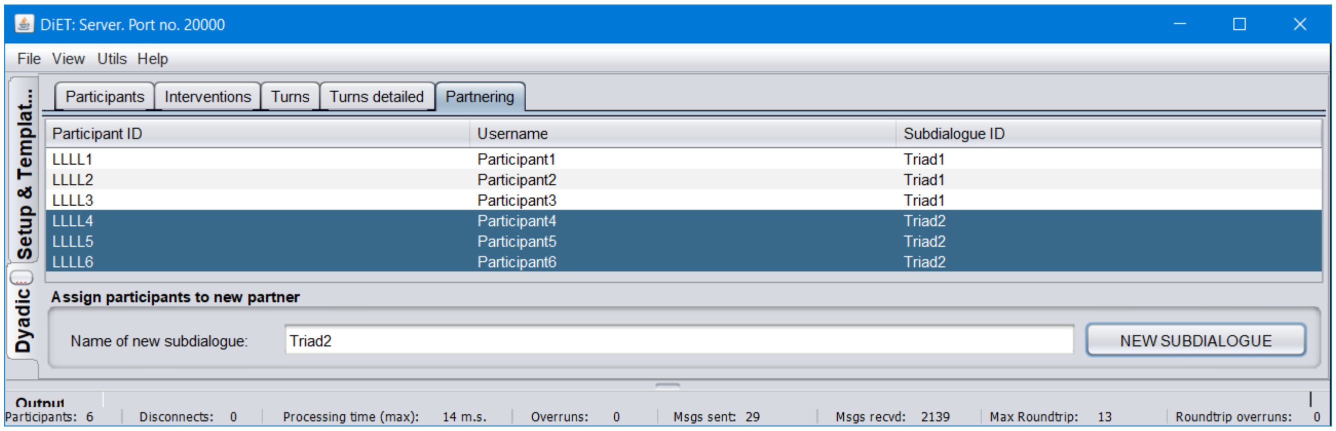
Name of new subdialogue: Triad1

```
Creating new ParticipantConnection
------------------------------------------------------------
Created new subdialogue called: Triad1 with the following participants:
LLLL1 Participant1
LLLL2 Participant2
LLLL3 Participant3
```

```
Participants: 6 | Disconnects: 0 | Processing time (max): 14 m.s. | Overruns: 0 | Msgs sent: 29 | Msgs recvd: 1977 | Max Roundtrip: 13 | Roundtrip overruns: 0
```

Figure 19 Reassining participants to different groups - in this example assigning participant1, participant2 and participant3 to a triadic interaction.

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 Figure 19) and then press “NEW SUBDIALOGUE”. The results are shown in Figure 19 above. Figure 20 below shows what happens when another Triad is created: Triad2 contains participant 4, participant 5 and participant 6.

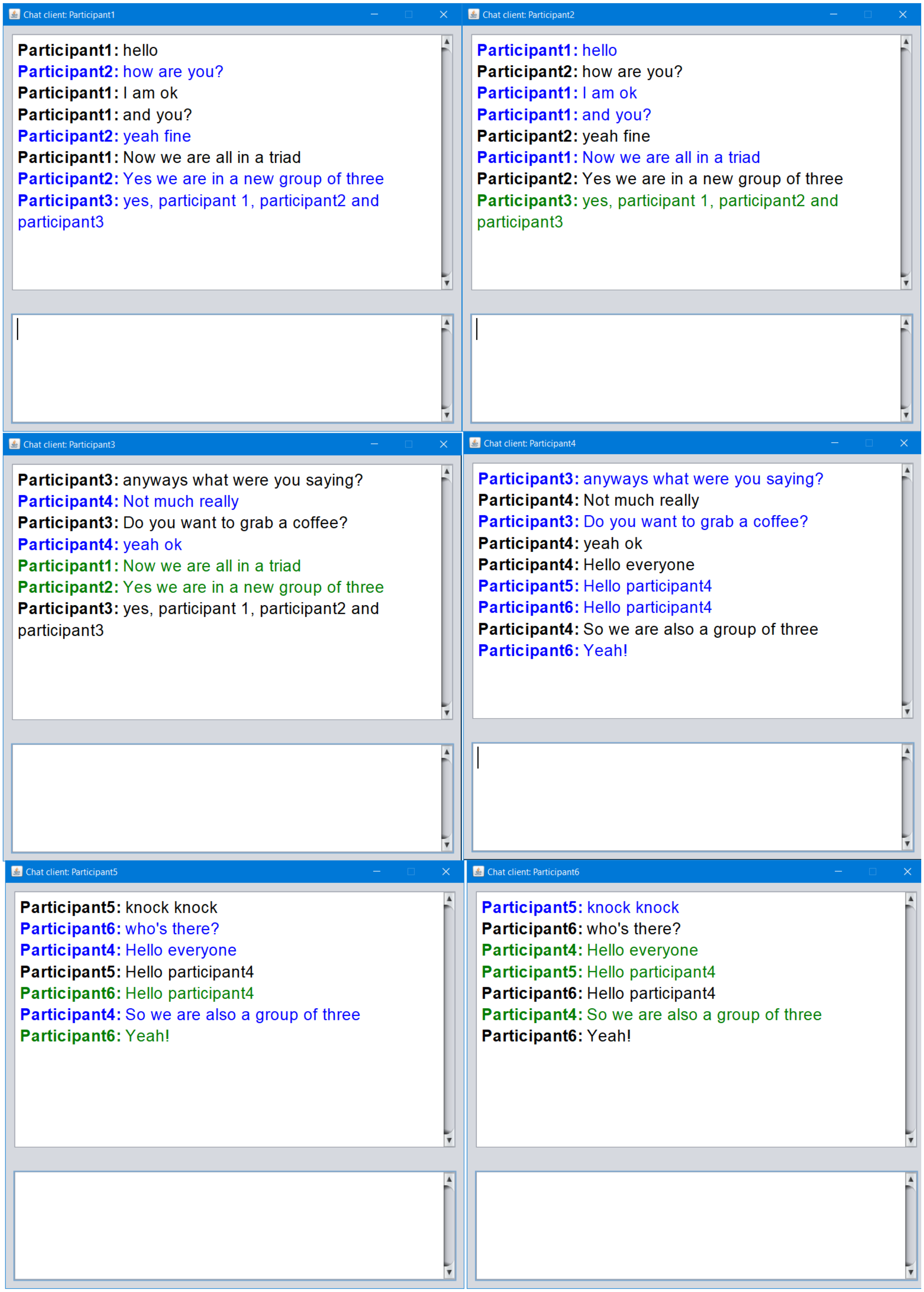


Participant ID	Username	Subdialogue ID
LLLL1	Participant1	Triad1
LLLL2	Participant2	Triad1
LLLL3	Participant3	Triad1
LLLL4	Participant4	Triad2
LLLL5	Participant5	Triad2
LLLL6	Participant6	Triad2

Name of new subdialogue: Triad2

```
Participants: 6 | Disconnects: 0 | Processing time (max): 14 m.s. | Overruns: 0 | Msgs sent: 29 | Msgs recvd: 2139 | Max Roundtrip: 13 | Roundtrip overruns: 0
```

Figure 20 Reassinging participants to different groups - in this example assigning participant4, participant5 and participant6 to a triadic interaction.


Chat client: Participant1
Chat client: Participant2
Chat client: Participant3
Chat client: Participant4
Chat client: Participant5
Chat client: Participant6

```
Participant1: hello
Participant2: how are you?
Participant1: I am ok
Participant1: and you?
Participant2: yeah fine
Participant1: Now we are all in a triad
Participant2: Yes we are in a new group of three
Participant3: yes, participant 1, participant2 and
participant3
```

```
Participant1: hello
Participant2: how are you?
Participant1: I am ok
Participant1: and you?
Participant2: yeah fine
Participant1: Now we are all in a triad
Participant2: Yes we are in a new group of three
Participant3: yes, participant 1, participant2 and
participant3
```

```
Participant3: anyways what were you saying?
Participant4: Not much really
Participant3: Do you want to grab a coffee?
Participant4: yeah ok
Participant1: Now we are all in a triad
Participant2: Yes we are in a new group of three
Participant3: yes, participant 1, participant2 and
participant3
```

```
Participant3: anyways what were you saying?
Participant4: Not much really
Participant3: Do you want to grab a coffee?
Participant4: yeah ok
Participant4: Hello everyone
Participant5: Hello participant4
Participant6: Hello participant4
Participant4: So we are also a group of three
Participant6: Yeah!
```

```
Participant5: knock knock
Participant6: who's there?
Participant4: Hello everyone
Participant5: Hello participant4
Participant6: Hello participant4
Participant4: So we are also a group of three
Participant6: Yeah!
```

```
Participant5: knock knock
Participant6: who's there?
Participant4: Hello everyone
Participant5: Hello participant4
Participant6: Hello participant4
Participant4: So we are also a group of three
Participant6: Yeah!
```

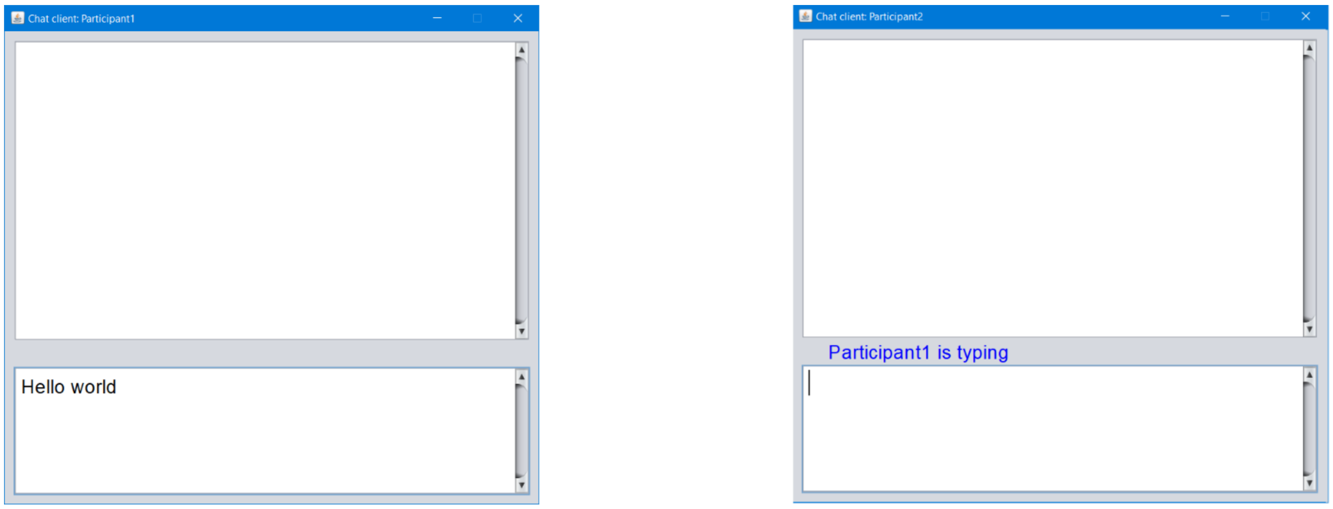
Figure 21 There are now two groups: Group 1 contains Participant 1, Participant2 and Participant 3. Group 2 contains Participant 4, Participant 5 and Participant 6

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.



Hello world

Participant1 is typing

Figure 22 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..

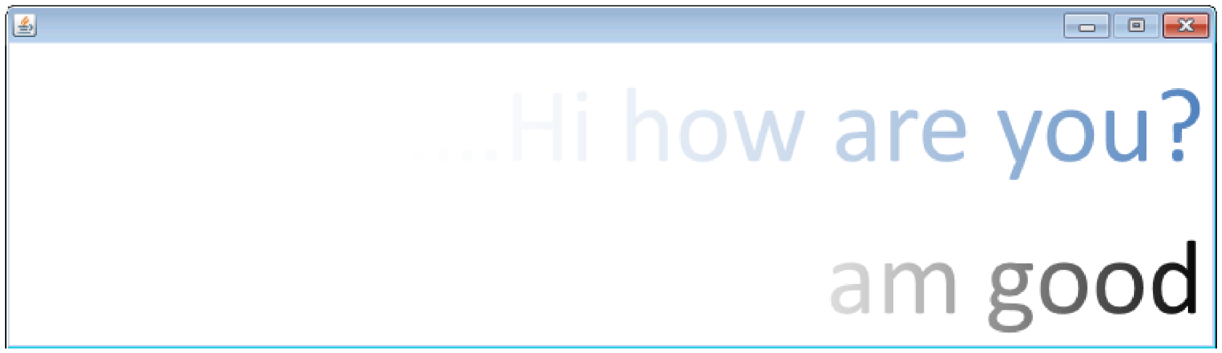


Figure 23 WYSIWYG Interface with each participant on a separate row

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

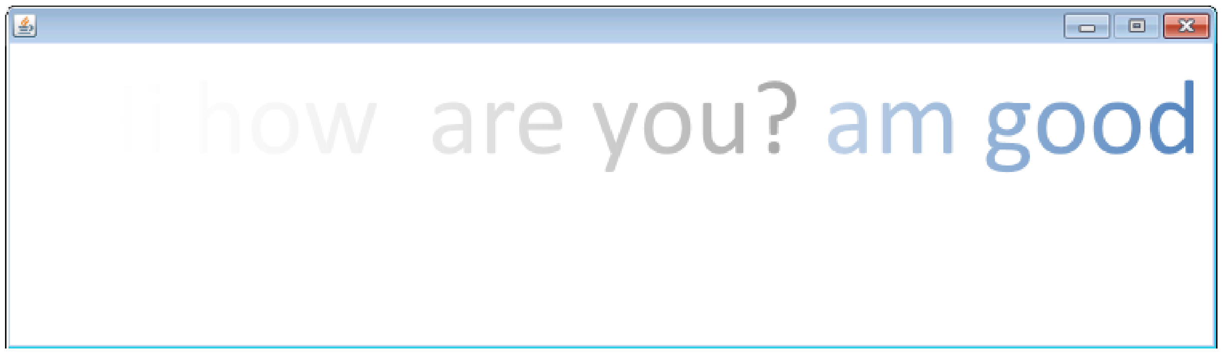


Figure 24 WYSIWYG Interface: Both participants on the same row

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.

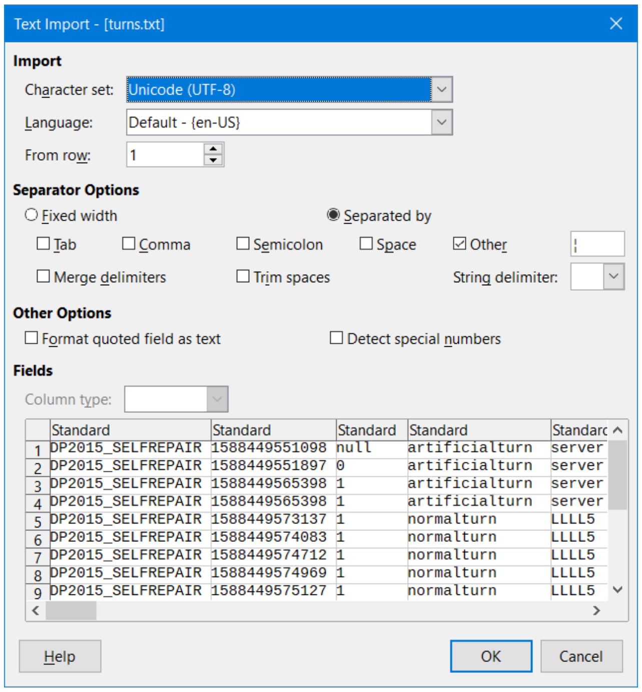
The saved data – turns.txt

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 25. (*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*)


Text Import - [turns.txt]

Import

Character set: Unicode (UTF-8)

Language: Default - {en-US}

From row: 1

Separator Options

Fixed width [✓ Separated by]

☐ Tab ☐ Comma ☐ Semicolon ☐ Space [✓ Other] ¦

☐ Merge delimiters ☐ Trim spaces String delimiter:

Other Options

☐ Format quoted field as text ☐ Detect special numbers

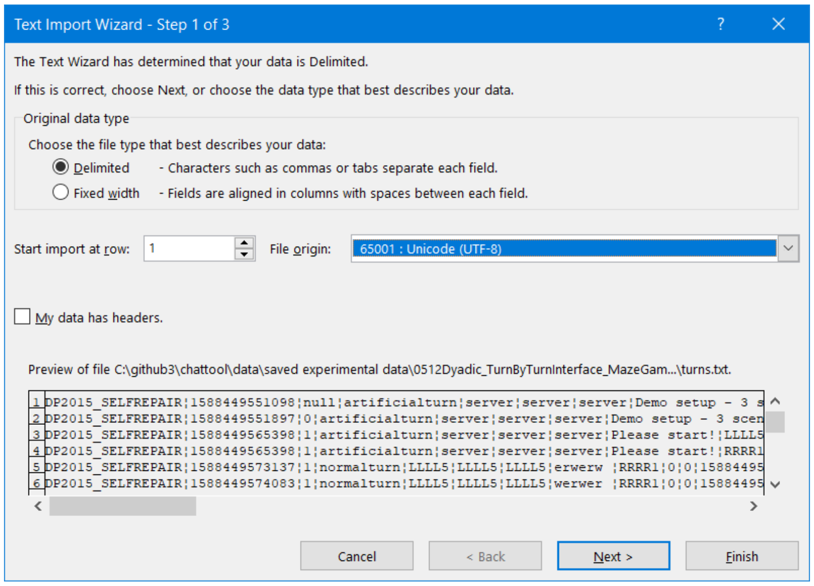
Fields

Column type:

	Standard	Standard	Standard	Standard	Standard
1	DP2015_SELFREPAIR	1588449551098	null	artificialturn	server
2	DP2015_SELFREPAIR	1588449551897	0	artificialturn	server
3	DP2015_SELFREPAIR	1588449565398	1	artificialturn	server
4	DP2015_SELFREPAIR	1588449565398	1	artificialturn	server
5	DP2015_SELFREPAIR	1588449573137	1	normalturn	LLLL5
6	DP2015_SELFREPAIR	1588449574083	1	normalturn	LLLL5
7	DP2015_SELFREPAIR	1588449574712	1	normalturn	LLLL5
8	DP2015_SELFREPAIR	1588449574969	1	normalturn	LLLL5
9	DP2015_SELFREPAIR	1588449575127	1	normalturn	LLLL5

Figure 25 Loading data with LibreOffice

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


Text Import Wizard - Step 1 of 3

The Text Wizard has determined that your data is Delimited.

If this is correct, choose Next, or choose the data type that best describes your data.

Original data type

Choose the file type that best describes your data:

[✓ Delimited] - Characters such as commas or tabs separate each field.

Fixed width - Fields are aligned in columns with spaces between each field.

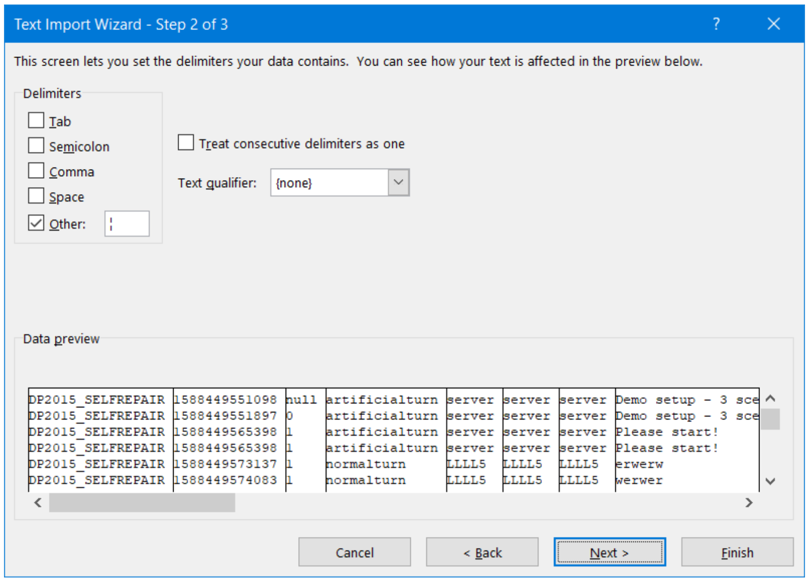
Start import at row: 1 File origin: 65001 : Unicode (UTF-8)

☐ My data has headers.

Preview of file C:\github3\chattool\data\saved experimental data\0512Dyadic_TurnByTurnInterface_MazeGam...\turns.txt.

```
1 DP2015_SELFREPAIR¦1588449551098¦null¦artificialturn¦server¦server¦server¦Demo setup - 3 s
2 DP2015_SELFREPAIR¦1588449551897¦0¦artificialturn¦server¦server¦server¦Demo setup - 3 scen
3 DP2015_SELFREPAIR¦1588449565398¦1¦artificialturn¦server¦server¦server¦Please start!¦LLLL5
4 DP2015_SELFREPAIR¦1588449565398¦1¦artificialturn¦server¦server¦server¦Please start!¦RRRR1
5 DP2015_SELFREPAIR¦1588449573137¦1¦normalturn¦LLLL5¦LLLL5¦LLLL5¦erwerw ¦RRRR1¦0¦0¦15884495
6 DP2015_SELFREPAIR¦1588449574083¦1¦normalturn¦LLLL5¦LLLL5¦LLLL5¦werwer ¦RRRR1¦0¦0¦15884495
```

Figure 26 Importing data in excel, step 1



Text Import Wizard - Step 2 of 3

This screen lets you set the delimiters your data contains. You can see how your text is affected in the preview below.

Delimiters
- [ ] Tab
- [ ] Semicolon
- [ ] Comma
- [ ] Space
- [x] Other: ¦

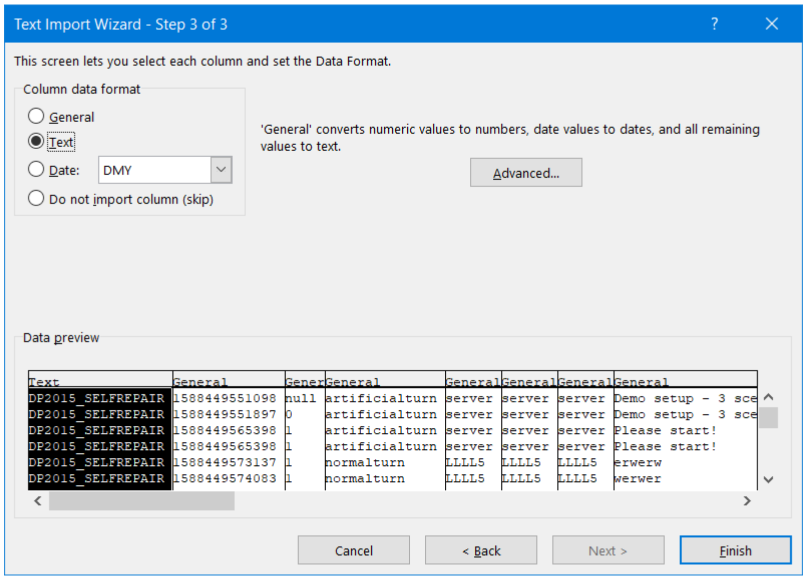
- [ ] Treat consecutive delimiters as one

Text qualifier: {none}

Data preview

```
DP2015_SELFREPAIR 1588449551098 null artificialturn server server server Demo setup - 3 sce
DP2015_SELFREPAIR 1588449551897 0    artificialturn server server server Demo setup - 3 sce
DP2015_SELFREPAIR 1588449565398 1    artificialturn server server server Please start!
DP2015_SELFREPAIR 1588449565398 1    artificialturn server server server Please start!
DP2015_SELFREPAIR 1588449573137 1    normalturn     LLLL5  LLLL5  LLLL5  erwerw
DP2015_SELFREPAIR 1588449574083 1    normalturn     LLLL5  LLLL5  LLLL5  werwer
```

Figure 27 Importing data in excel step 2



Text Import Wizard - Step 3 of 3

This screen lets you select each column and set the Data Format.

Column data format
- ( ) General
- (●) Text
- ( ) Date: DMY
- ( ) Do not import column (skip)

'General' converts numeric values to numbers, date values to dates, and all remaining values to text.

Data preview

```
Text              General       GenerGeneral        GeneralGeneralGeneralGeneral
DP2015_SELFREPAIR 1588449551098 null artificialturn server server server Demo setup - 3 sce
DP2015_SELFREPAIR 1588449551897 0    artificialturn server server server Demo setup - 3 sce
DP2015_SELFREPAIR 1588449565398 1    artificialturn server server server Please start!
DP2015_SELFREPAIR 1588449565398 1    artificialturn server server server Please start!
DP2015_SELFREPAIR 1588449573137 1    normalturn     LLLL5  LLLL5  LLLL5  erwerw
DP2015_SELFREPAIR 1588449574083 1    normalturn     LLLL5  LLLL5  LLLL5  werwer
```

Figure 28 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:

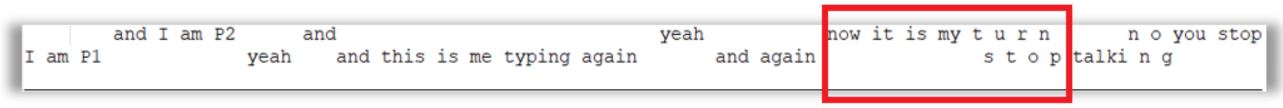
A: now it is my t
B: s
A: u
B: t
A: r
B: o
A: p talking

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

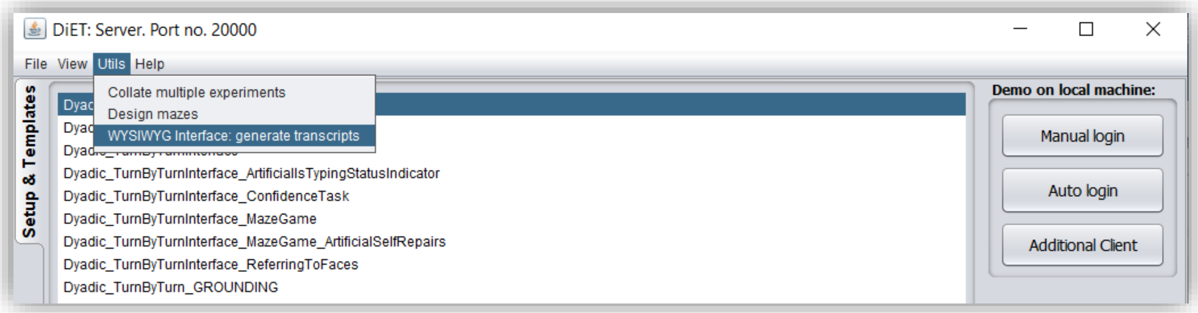


```
        and I am P2     and                            yeah           now it is my t u r n       n o you stop
I am P1            yeah    and this is me typing again      and again                s t o p talki n g
```

Figure 29 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”



```
Dyadic_TurnByTurnInterface_ArtificialIsTypingStatusIndicator
Dyadic_TurnByTurnInterface_ConfidenceTask
Dyadic_TurnByTurnInterface_MazeGame
Dyadic_TurnByTurnInterface_MazeGame_ArtificialSelfRepairs
Dyadic_TurnByTurnInterface_ReferringToFaces
Dyadic_TurnByTurn_GROUNDING
```

Figure 30 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

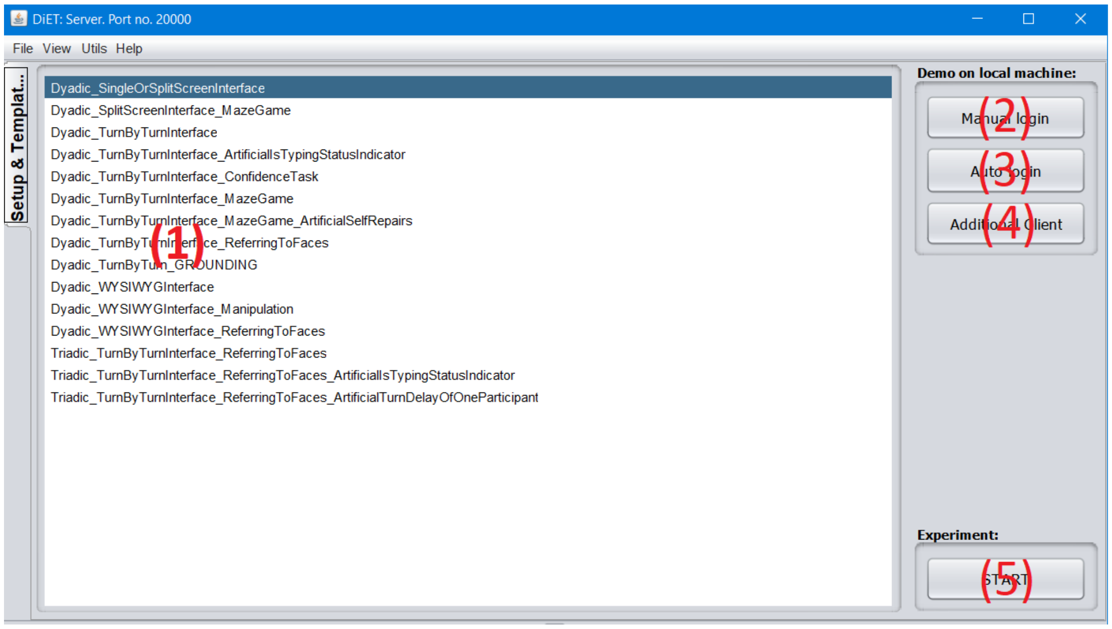
1. Double-click on "chattool.jar" . If this works, this means you have java installed.

2. If double-clicking doesn't work, open a Terminal window and navigate to the folder where "chattool.jar" is located, then type **java -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 1), select "start server".

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

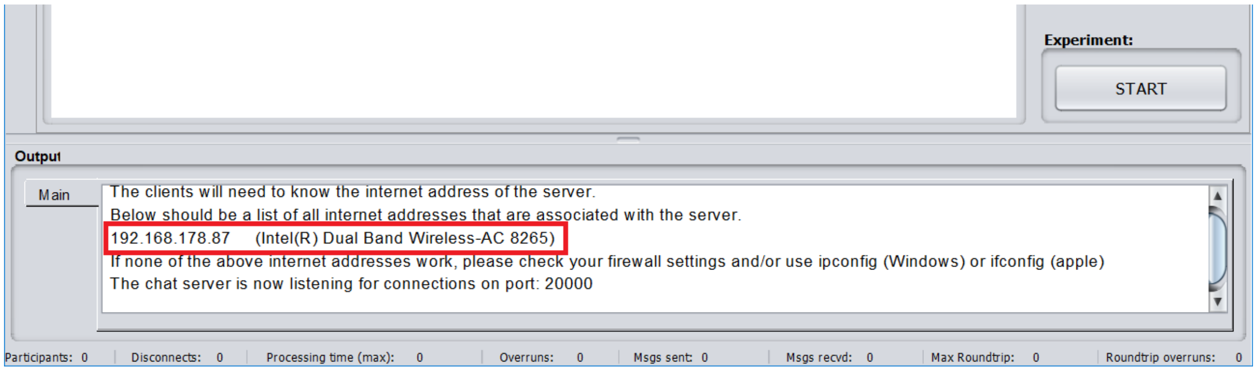


Dyadic_SingleOrSplitScreenInterface
Dyadic_SplitScreenInterface_MazeGame
Dyadic_TurnByTurnInterface
Dyadic_TurnByTurnInterface_ArtificialIsTypingStatusIndicator
Dyadic_TurnByTurnInterface_ConfidenceTask
Dyadic_TurnByTurnInterface_MazeGame
Dyadic_TurnByTurnInterface_MazeGame_ArtificialSelfRepairs
Dyadic_TurnByTurnInterface_ReferringToFaces
Dyadic_TurnByTurn_GROUNDING
Dyadic_WYSIWYGInterface
Dyadic_WYSIWYGInterface_Manipulation
Dyadic_WYSIWYGInterface_ReferringToFaces
Triadic_TurnByTurnInterface_ReferringToFaces
Triadic_TurnByTurnInterface_ReferringToFaces_ArtificialIsTypingStatusIndicator
Triadic_TurnByTurnInterface_ReferringToFaces_ArtificialTurnDelayOfOneParticipant

Figure 31 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:



```
The clients will need to know the internet address of the server.
Below should be a list of all internet addresses that are associated with the server.
192.168.178.87     (Intel(R) Dual Band Wireless-AC 8265)
If none of the above internet addresses work, please check your firewall settings and/or use ipconfig (Windows) or ifconfig (apple)
The chat server is now listening for connections on port: 20000
```

```
Participants: 0 | Disconnects: 0 | Processing time (max): 0 | Overruns: 0 | Msgs sent: 0 | Msgs recvd: 0 | Max Roundtrip: 0 | Roundtrip overruns: 0
```

Figure 32 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.

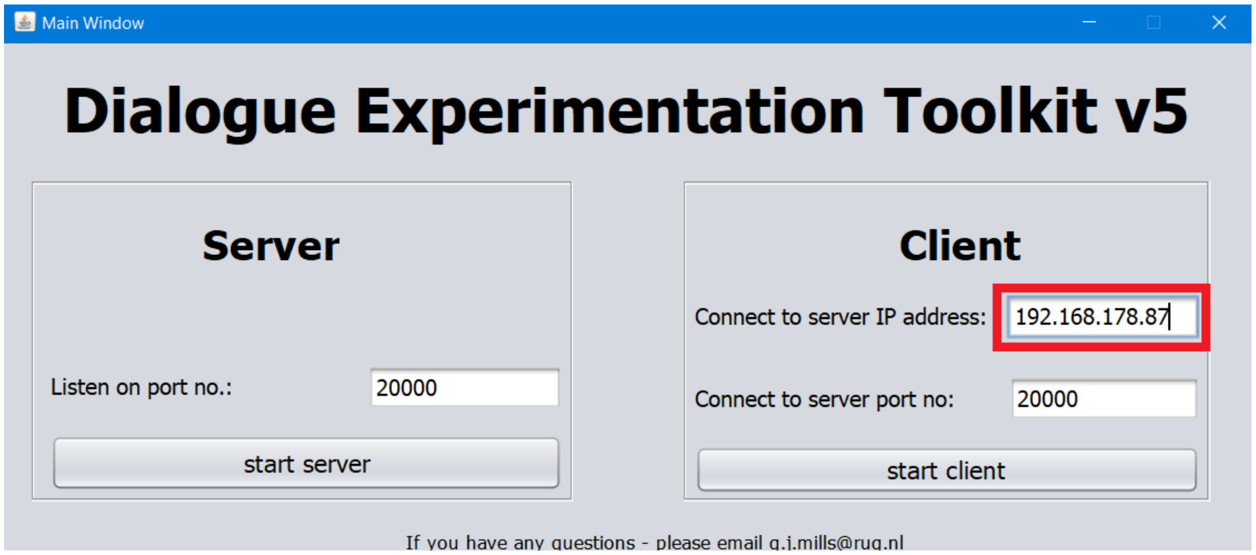
Mac / Linux

Double-click on "chattool.jar" or open a Terminal window and navigate to the folder where "chattool.jar" is located, then type **java -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 32 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:


Main Window

Dialogue Experimentation Toolkit v5

Server

Listen on port no.: 20000

Client

Connect to server IP address: 192.168.178.87

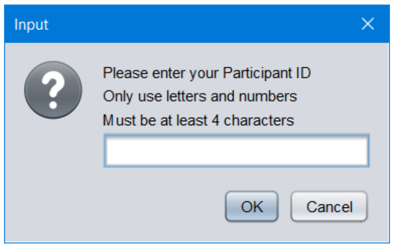
Connect to server port no: 20000

If you have any questions - please email g.j.mills@rug.nl

Figure 33 Starting the client – entering the IP address of the server

Then press “start client”

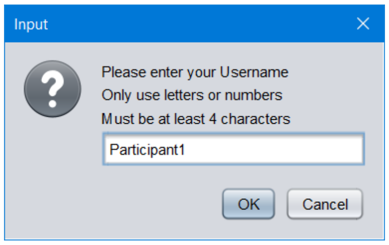
You should see a popup menu asking for a participant ID


Input

Please enter your Participant ID
Only use letters and numbers
Must be at least 4 characters

Figure 34 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:


Input

Please enter your Username
Only use letters or numbers
Must be at least 4 characters

Participant1

Figure 35 Starting the client Entering the username

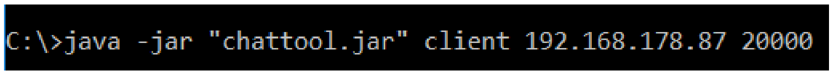
Repeat this process on another computer – (choosing a different Participant ID and Username).

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:



```
C:\>java -jar "chattool.jar" client 192.168.178.87 20000
```

Figure 36 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.

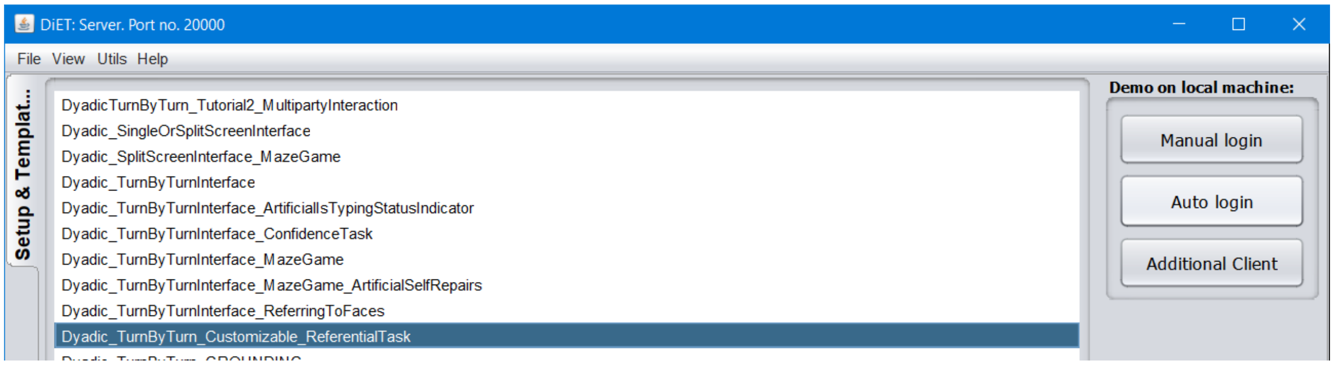
Running your own referential task (e.g. “tangram” joint reference task)

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”



DyadicTurnByTurn_Tutorial2_MultipartyInteraction
Dyadic_SingleOrSplitScreenInterface
Dyadic_SplitScreenInterface_MazeGame
Dyadic_TurnByTurnInterface
Dyadic_TurnByTurnInterface_ArtificialIsTypingStatusIndicator
Dyadic_TurnByTurnInterface_ConfidenceTask
Dyadic_TurnByTurnInterface_MazeGame
Dyadic_TurnByTurnInterface_MazeGame_ArtificialSelfRepairs
Dyadic_TurnByTurnInterface_ReferringToFaces
Dyadic_TurnByTurn_Customizable_ReferentialTask

Figure 37 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 38, 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 38) and the server also has a window (Figure 40) 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.

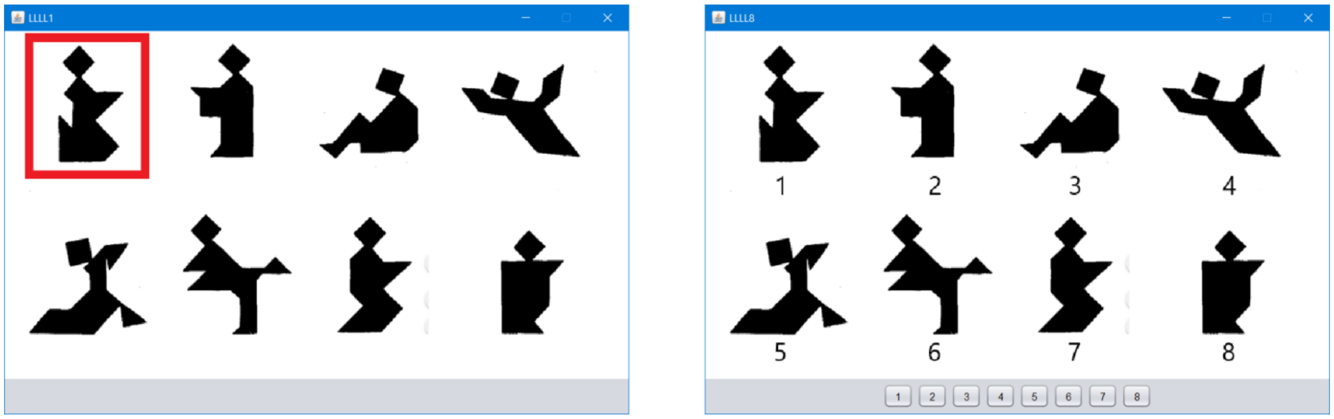


Figure 38 The stimuli windows of both participants. The left window shows the director. The right window shows the matcher. Note how the matcher has eight buttons underneath which can be pressed to make a selection.

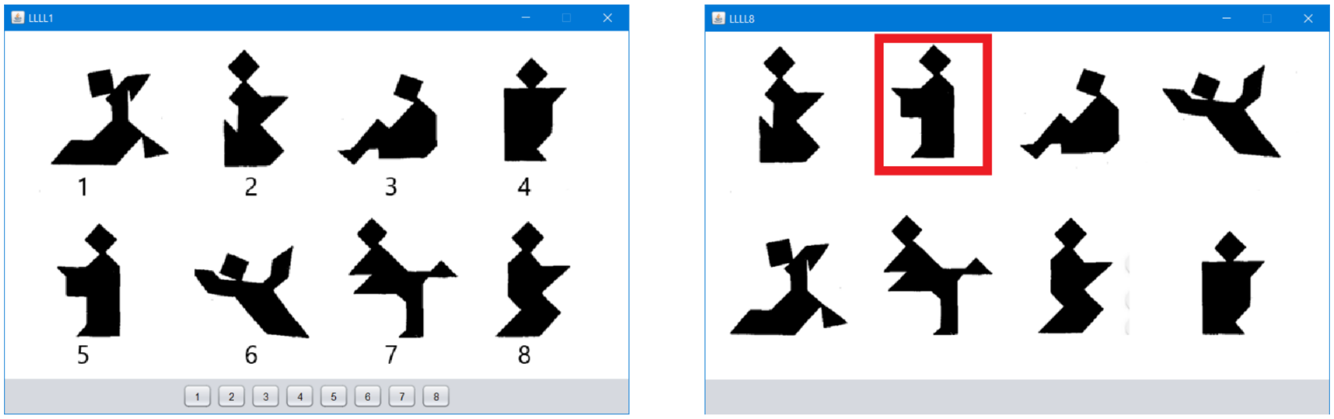
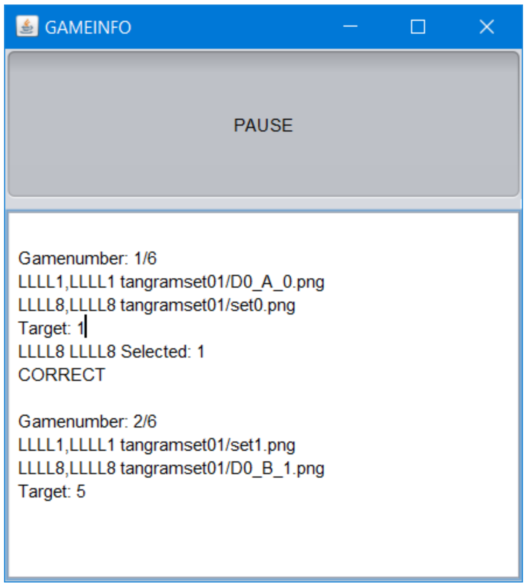


Figure 39 The stimuli window of both participants in game 2. Notice how the participant on the right is now the Director. Notice also how the participant on the right no longer has buttons - these are displayed on the matcher's screen on the left



```
Gamenumber: 1/6
LLLL1,LLLL1 tangramset01/D0_A_0.png
LLLL8,LLLL8 tangramset01/set0.png
Target: 1
LLLL8 LLLL8 Selected: 1
CORRECT

Gamenumber: 2/6
LLLL1,LLLL1 tangramset01/set1.png
LLLL8,LLLL8 tangramset01/D0_B_1.png
Target: 5
```

Figure 40 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:

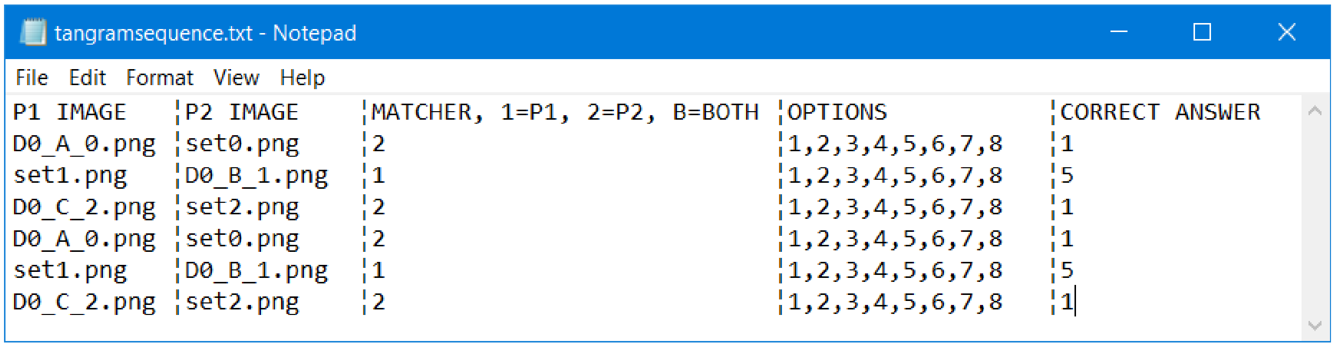
/experimentresources/stimuli/facerealornot/face-fakeorreal.txt

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*



```
P1 IMAGE   ¦P2 IMAGE   ¦MATCHER, 1=P1, 2=P2, B=BOTH ¦OPTIONS         ¦CORRECT ANSWER
D0_A_0.png ¦set0.png   ¦2                           ¦1,2,3,4,5,6,7,8 ¦1
set1.png   ¦D0_B_1.png ¦1                           ¦1,2,3,4,5,6,7,8 ¦5
D0_C_2.png ¦set2.png   ¦2                           ¦1,2,3,4,5,6,7,8 ¦1
D0_A_0.png ¦set0.png   ¦2                           ¦1,2,3,4,5,6,7,8 ¦1
set1.png   ¦D0_B_1.png ¦1                           ¦1,2,3,4,5,6,7,8 ¦5
D0_C_2.png ¦set2.png   ¦2                           ¦1,2,3,4,5,6,7,8 ¦1
```

Figure 41 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.

, above, of how to format the text file. Fill each row out for each game. It is best to start with a couple of games first to make sure that it works.

7. You can save multiple scripts to the same folder – use them for different dyads.