oTree: Implementing websockets to allow for real-time interactions – A continuous double auction market as first application

Frauke von Bieberstein, Ann-Kathrin Crede, Jan Dietrich, Jonas Gehrlein, Oliver Neumann, Matthias Stürmer

Abstract

This article illustrates the implementation of websockets in oTree (Chen et al. 2016) to allow for real-time interactions. While oTree generally allows to overcome the need for participants to be at the same location to interact with each other, a real-time module in the sense that the user interface responds within milliseconds to actions from other participants is currently not available. We address this gap and further develop oTree by making real-time interactions between a large number of players with immediate updates possible. As a first application, we run a continuous double auction market on Amazon Mechanical Turk to validate its functionality. This ready-to-use software is of special interest for the research of large (online) markets and for teaching purposes. We provide the code open-source on GitHub, thereby encouraging users to develop it further and applying the websocket technology to other real-time settings.

Keywords: Experimental economics, oTree, double auction market, websockets, open-source

Acknowledgments:
We thank Christoph Zangger, Roden Safar, Annika Wyss, and Nicolas Hafner for excellent research assistance. We are grateful to Ilija Braun and Andrea Essl for very valuable comments.

1 Introduction

The use of oTree as a software to run experiments has a number of advantages that have been documented and tested since its introduction (e.g., Chen et al. 2016, Holzmeister & Pfurtscheller 2016, Holzmeister 2017). These advantages are especially visible in comparison to the widely used z-Tree (Fischbacher 2007). The main benefit is that only a web browser is necessary, thereby allowing that participants need not be at the same location to interact with each other. Especially for large-scale experiments where interaction between many subjects is necessary, oTree offers clear advantages. The native integration with Amazon Mechanical Turk provides an additional beneficial feature to run experiments online with non-standard subject pools. While oTree obviously offers new possibilities for running experiments, one important feature is not available so far: It is missing a real-time component in the sense that the user interface does not update within milliseconds to actions from other participants (Chen et al. 2016: 96).

Our novel implementation relies on websockets[1] allowing that the input of any kind of information of one participant is immediately transmitted to other participants without the need to refresh the current page. This automatic update allows participants to interact rapidly and in real-time. With respect to our application to a double auction (DA) market, the real-time component creates a realistic and authentic environment of a marketplace, where trading occurs fast and requires

[1]We use the term "websockets" to describe the technology in general, although more precisely the term "WebSockets" describes the protocol. The implementation of the websocket technology in oTree is done by Django channels (https://github.com/andrewgodwin), see section 2 for details.

quick decision-making. In this paper, we evaluate data gathered by a related project (Adrian et al. 2019)[2] to test the functionality of the DA market. We find that the module proved to work smoothly (see section 4).

We chose a DA market as first application to meet economists' ongoing interest in understanding and investigating markets and focus on market clearance through pure competition (for other mechanisms see Plott & Sunder 1988). In the 1960s, Vernon Smith was the first to run an oral DA market experiment with his students. In the basic version, participants are randomly assigned to a group of buyers or a group of sellers. Each buyer privately learns the maximum price he is willing to pay for one unit of the good. Each seller privately learns the minimum price at which he is willing to give away the good. Each buyer and each seller can trade once per round over several rounds in total. Despite his intention to reject competitive market theory, Smith found that the realized price under the DA market mechanism instead converged toward the competitive equilibrium even though basic assumptions (e.g., infinite number of buyers and sellers) were not met (Smith 1962). His work initiated a long wave of research and contributed to the promotion of laboratory experiments as a research method in economics (e.g., Smith 1976, Miller et al. 1977, Plott & Smith 1978, Williams 1980, Ketcham et al. 1984, Smith et al. 1988).

Today, investigating markets experimentally is common practice, especially in the context of financial markets (see Nuzzo & Morone 2017 for a recent overview). Most of the studies rely on some form of computer software providing the market

[2]The study by Adrian et al. (2019) investigates the influence of markets on moral decisions. In their experiment, the data collected on the market is not analyzed as it is not part of their research question. The DA only serves as stimulus for a subsequent moral decision.

infrastructure. While there is a large range of software for experimental markets, most researchers develop their own, custom-built solutions that are often not publically available (Palan 2015). However, there is a number of exceptions: VeconLab[3] and EconPort[4] provide online platforms that allow to (partly) configure and run different kinds of market games. GIMS (Graz-Innsbruck Market System) by Palan (2015) is a publically available market software, but is technically restricted to the features of the z-Tree application. ConG (Continuous Games) by Pettit et al. (2014) provides the fundament for running experiments with real-time interactions, but is not designed to run two-sided market institutions such as a continuous DA market (Pettit et al. 2014: 647). nodeGame by Balietti (2017) allows to run real-time experiments online with participants only needing a web browser access but does not provide a module for a DA market yet. Together, an experimental market software that is suitable for a wide range of research applications, easily usable and customizable and provided open-source is not available so far. We close this gap by offering an easy to implement and customizable DA market module for use within oTree. By providing the code open-source on GitHub, we want to encourage users to develop it further and apply the websocket technology to other real-time settings.

This paper proceeds as follows: Section 2 gives technical details on the use of websockets within oTree. Section 3 continues with the explanation of how to set up and use the DA market module. Section 4 shows the experimental design of the DA market, the procedural details and first results. Section 5 shortly concludes.

[3]See http://veconlab.econ.virginia.edu
[4]See http://econport.org/econport/request?page=web_home

2 Real-time interactions in oTree with websockets

Real-time interactions are currently not provided as default option in oTree. In the following section, we illustrate how the current framework operates and why this complicates the implementation of real-time interactions. We then illustrate an approach to solve this issue with examples from the DA market code.

Current limitations of oTree

The current oTree architecture is not suitable for real-time interactions as part of a DA market, since such applications need a persistent connection with uninterrupted communication between the clients and the server. Currently, the user requests a page from the oTree server, makes some inputs and clicks on the "next" button to save the state and advance to the next page (see Figure 1).

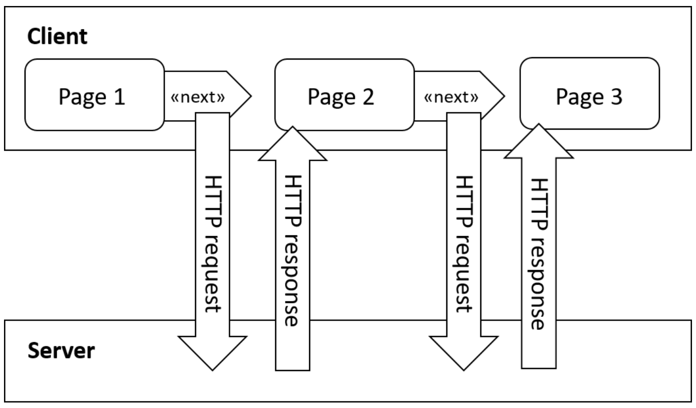


Figure 1: The current communication of an oTree application is based on HTTP requests

Technically, only the "client" can initiate each step with a new HTTP `GET` or `POST` request to the server in order to load an entirely new HTML page. It is therefore necessary to use a technical protocol that allows the client to simultaneously send and receive messages from the server.

Using websockets in oTree

The DA market module relies on the application of websockets to handle interactions among players. A websocket is a continuous connection between a client and a server. In the case of oTree, this means that the participant's web browser keeps an open internet connection to the oTree service where messages can be sent and received simultaneously.[5] A way to cope with missing websockets is offered by the underlying Django framework which provides an abstraction of websockets, called "Django channels". Such channels represent an active websocket connection to a client that can be used to send or receive messages and terminate its connection. Multiple channels can be grouped to send messages to multiple clients. Figure 2 illustrates this architecture.

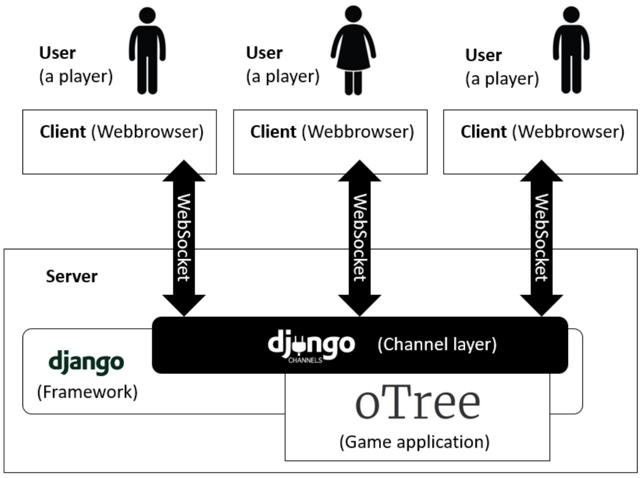


Figure 2: The websockets architecture within oTree using Django channels

[5]Websockets are already used in oTree to auto-advance players initiated by a process called "oTree worker": https://otree.readthedocs.io/en/latest/timeouts.html. However, applications to games are currently limited.

We show the use of websockets for four different forms of interactions: (1) connecting to a websocket, (2) sending a message, (3) receiving a message, and (4) disconnecting from the websocket. These four interactions, which are illustrated in Figure 3, require different implementations on the server- and client-side. For the server-side implementation, we provide various code snippets for further illustration using the DA market example.

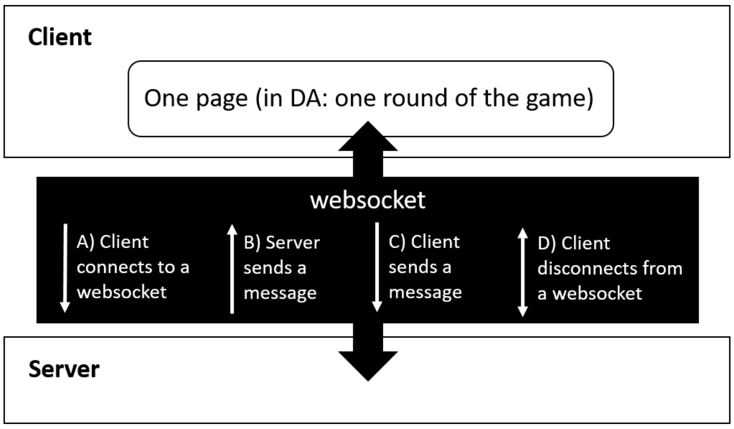

Client
One page (in DA: one round of the game)
websocket
A) Client connects to a websocket
B) Server sends a message
C) Client sends a message
D) Client disconnects from a websocket
Server

Figure 3: Websockets form a permanent communication channel between the client and the server

Server-side implementation

The following section focuses on the server-side implementation and provides (pseudo) code written in Python as an illustration. In the following examples a websocket consumer class is used to handle the server-side interaction.

A) Connecting to a websocket: Various functions can be defined on the server which specify the actions of a client once it connects to the websocket. In our application, the client receives all information regarding the current state of the market, i.e. all bids and asks with the corresponding contractors, as multiple messages. Additionally, the websockets of all players of a DA market are grouped together in order to be able to broadcast group messages which are relevant for all market participants. A sample code is:

```
def connect(connection, **connection_params):
    """
    client connects to websocket and is automatically added to
    group_channel
    """
    session_code, player, group_channel = get_infos(connection_params)
    self.send(current_game_state)
```

B) Sending a message: As mentioned above, either Django channels or Django channel groups can be used to send messages, such as "new bid of 40 of buyer 6". At the beginning of a round the connection is established and when a player changes his bid or ask, all other players need to be immediately informed about this change. The server informs all DA market players about the following actions:

- A player has updated his bid/ask,
- A player has deleted his bid/ask,
- A player has disconnected and is replaced by a bot,
- A player has reconnected and replaced the bot.

A sample code is:

```
# This sends a message to all DA market participants' websocket
group_send(group_channel, {
    "type": "match",
    "buyer": buyer_id,
    "seller": seller_id,
    "value": value
})
```

C) Receiving a message: The server collects all messages of user actions, calls the respective functions and replies with corresponding messages. In the DA market, there are multiple types of actions that a player can perform. The user can update or clear his entry or accept an offer/a bid from the "Current bids and asks" table. The server receives the message, evaluates the type of action, verifies its permissibility and acts accordingly, e.g. saves the changes to the database and informs the other players about the updated state. In case a message from a client is invalid, the server ignores it. The sample code is:

```
def receive(message, **conection_params):
    """
    When a message arrives the market state is updated according
    to the message.action_type
    """
    session_code, player, group_channel = get_infos(connection_params)
    if message.action_type == "clear":
        clear_bet(player.id)
    elif message.action_type == "seller" or action_type == "buyer":
        create_bid(message)
    ...
    endif
```

D) Disconnecting from the websocket: In case a client loses or closes the connection to the server, several game-specific functions can be implemented. The DA market module supports automated players that can replace a user who dropped the connection. If the bot service is enabled (see section on bots), all other players are informed about that dropout by labeling that player visually as a bot. The code for disconnecting is:

```
def disconnect(connection, **connection_params):
    """
    When a player disconnects he is automatically removed from
    group_channel
    """
    session_code, player, group_channel = get_infos(connection_params)
    replace_player_with_bot(player)
```

Once a connection is dropped, the websocket is discarded from the Django channel group.

Client-side implementation

As the client-side interaction is handled in the user's web browser, the corresponding websockets are implemented with built-in JavaScript functions. Possible actions correspond to those described for the server-side implementation.

A) Connecting with a websocket: After the page has loaded in the web browser, the client opens a websocket connection which stays open until the browser tab is closed or a new page is loaded. In our application, no actions are required on the client-side as the server handles the entire interaction.

B) Sending a message: If the user makes an input on the current page, a message will be sent through the websocket connection to inform the server of the change. In our application, a user can send three different types of messages:

- Update of a bid/ask
- Clearance of a bid/ask
- Accepting a bid/ask from another player

C) Receiving a message: An incoming message through the websocket leads to an update of the corresponding part of the webpage of the client. For example, a new message could be reading “new bid of 40 of buyer 6”. Accordingly, the “Current bids and asks” table is updated to show the new state of the market. Additionally, current values are analyzed to find suitable bids or asks and to show a corresponding “Accept” button.

D) Disconnecting from the websocket: When the connection is interrupted, the client is automatically disconnected from the websocket. In this case, the server is informed about the interruption. In our implementation, no specific action is required on the client-side as an irregular end of a connection (e.g. by closing the web browser) is handled by the server. The disconnected player can be replaced by a bot. It is also possible to disable the bot service in the settings. In this case, a player that drops out is labeled as “inactive”.

Bots

One option to handle drop-outs is to enable the bot service: When a player drops out of the DA market (due to attrition or technical issues), i.e. the websocket is closed, he is immediately replaced by a bot and all other players are notified of that change. Those players receive the label “bot” which is added to the player ID in the table “Market participants” (see the screenshot in the Appendix). A bot is programmed to place a bid / ask equal to his valuation / production costs at a randomized point of time (relative to when the websocket closed). To achieve this, a scheduled task is added to the “oTree worker”. Missing players can always return and instantaneously replace “their” bots by restoring the web session.

3 Setup and usage

To use the DA market module, the experimenter is required to install oTree, Python and Redis[6]. Participants of the DA market only require an active internet connection and a common web browser on a computer or mobile device such as a smartphone or tablet. A preconfigured version of the DA market can be downloaded from GitHub.[7] It consists of the oTree project setup, a copy of the DA market (which can be found in the subfolder double_auction) and a readme file with additional detailed information on the setup. Once the code is cloned, the command `otree devserver` starts the web application from a terminal (e.g. windows command prompt) with the oTree admin interface.[8] A session can be created by clicking on the game and the oTree web interface provides the links to start the experiment. To customize the DA market for individual needs, important variables can be either changed through the user interface or configured within the `settings.py` file:

```
bot_enable #Enables the bot service
delay_before_market_opens #Countdown before round starts (in seconds)
market_size #Maximum number of players in each market
num_of_test_rounds #Number of test rounds
production_costs_increments #Production costs increments of the seller
production_costs_max #Maximum production costs of seller
production_costs_min #Minimum production costs of seller
time_per_round #Time of one round (in seconds)
valuation_increments #Valuation increments of the buyer
valuation_max #Maximum valuation of buyer
valuation_min #Minimum valuation of buyer
participation_fee #Fixed compensation for participation
real_world_currency_per_point #Variable compensation for participation
```

[6]https://redis.io. Redis is a message broker that handles schedules tasks and that is necessary to make use of the bots.
[7]https://github.com/IOP-Experiments
[8]https://otree.readthedocs.io/en/latest/tutorial/intro.html

The series of valuations (production costs) is created as following: The minimum valuation (production costs) is incremented by the specified parameter until the maximum valuation (production costs) is reached. The thereby generated values are then randomly assigned to the players. Each value is only assigned once among sellers and among buyers. When the number of players exceeds the number of values, the additional players receive a draw from another series, which is generated as described above. Due to the current structure of oTree, the number of rounds cannot be adjusted from the web interface but rather has to be changed in `models.py` by changing the `num_rounds` variable. Generally, users of the DA market module are advised to read the provided readme file in the repository.

4 The Double Auction Market

In the following, we describe the DA market as conducted for the first time by Adrian et al. (2019). While in their paper, the DA market is only needed to expose participants to a market environment to examine its impact on moral decisions, here we focus on the DA market data. We describe the game with predetermined values for the particular parameters. However, these can be easily changed and adapted for individual purposes (see section 3).

Experimental design

In our experiment, we ran a continuous DA market consisting of 9 buyers and 9 sellers over 10 rounds (with 2 additional, non-incentivized test rounds). We assign subjects randomly to either the role of a buyer or seller. Subjects keep their role for the entire 12 rounds. In every round, they can trade at most once a fictional good for 60 seconds. At the beginning of each round, buyers privately learn their valuation of the good and sellers privately learn their production costs of the good. Valuations and costs are randomly drawn from the sets $v \in \{30, 40, 50, \ldots, 120\}$ and $c \in \{10, 20, 30, \ldots, 90\}$. In each round, every value can only appear once among the buyers and sellers. While the distribution of demand and supply is common knowledge, the realization of v (for a buyer) or c (for a seller) is private knowledge to each market participant.

Sellers can sell and buyers can buy one unit of the fictional good in each round. Once the market opens, sellers can submit asks, i.e. the price at which they are willing to sell the product. Buyers can submit bids, i.e. the price at which they are willing to buy the product. All asks and bids appear in the table "Current bids and asks" and are visible to all market participants (see Appendix for a screenshot). A trade occurs if a seller makes an ask that is lower than a current bid or if a buyer makes a bid that is higher than a current ask. The trade is closed at the price (of the bid or the ask) that was posted first. A trade is also possible by directly accepting a bid or ask that appears in the table. Sellers and buyers can modify their asks and bids until the market closes, as long as they did not trade yet. If a trade occurs, the payoff is $\pi_S = price - production\ costs$ for the sellers and $\pi_B = valuation - price$ for the buyers. Production costs only occur when trading, which means that it is not possible that a seller produces the good at a personal cost but cannot sell it on the market. Furthermore, buyers and sellers cannot make bids or asks that would lead to a negative payoff. After each round, sellers and buyers receive feedback and see a table with all trades and prices. The complete instructions can be found in the appendix. Competitive equilibrium theory predicts an average trading price of 60 with a frequency of trades between 5 and 6 per round (see Figure 4 below).

Procedural details

In the first test of conducting the DA market (Adrian et al. 2019), we ran 8 markets on Amazon Mechanical Turk (AMT). After publishing the HIT, we let participants queue in a waiting room until all 18 spots were filled. This took on average less than 5 minutes. After that, subjects were provided with the instructions and control questions. To facilitate the formation of a group of 18 participants in a reasonable time, the evening before running sessions we posted the starting times on platforms such as turkerhub.com, as done by Suri & Watts (2011). Participation in the whole experiment (DA market plus two additional parts) took approximately 40 minutes and participants earned on average \$ 5.40 (\$ 3.00 participation fee plus the payment from one randomly selected round, where one point was transferred to \$ 0.15). The sessions were conducted between November 2nd and November 7th, 2018.

Results

Overall, $N_p = 116$ (out of 144 potential) participants in $N_m = 8$ markets completed the DA market, i.e. out of 18 players, we had on average 3 - 4 bots in each market. We analyze the number of trades and the actual average market price. Table 1 provides an overview.

Round	Average number of trades	Average market price
test 1	5.63	57.62
test 2	6.19	57.39
1	6.00	58.33
2	6.00	58.27
3	5.94	59.37
4	6.13	59.76
5	6.00	59.58
6	6.25	59.56
7	6.25	58.58
8	6.25	58.08
9	5.88	58.57
10	5.88	59.62

Table 1: Aggregated market history

As the table shows, the number of trades fluctuates around the theoretically predicted number of trades of 5 - 6 starting from the first round and the average market price converges rapidly toward the theoretically predicted (market-clearing) price of 60. Figure 4 illustrates the evolution of market prices graphically.

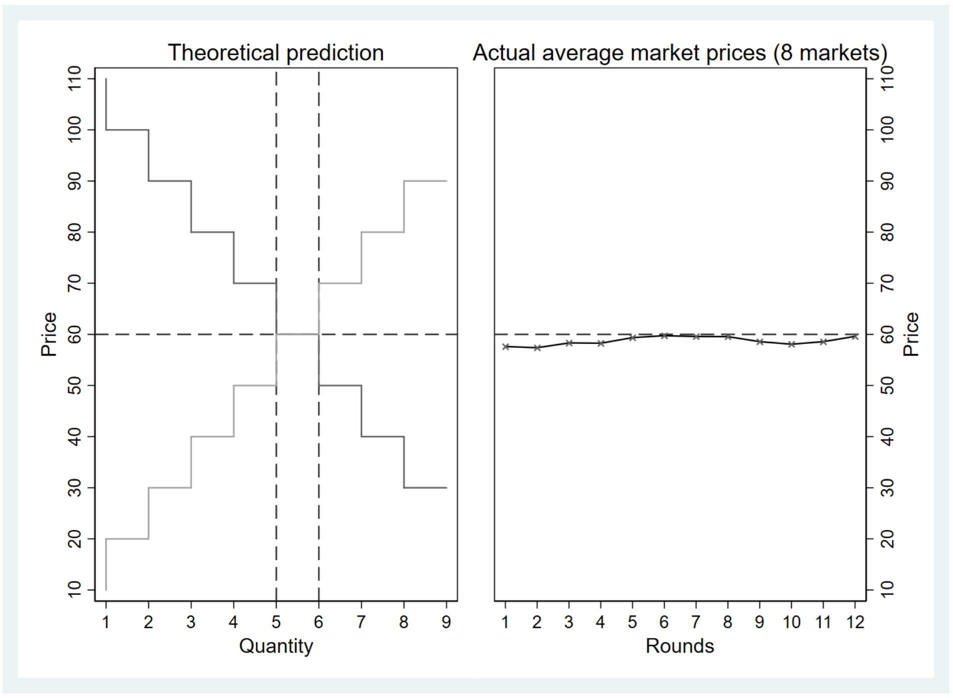


Figure 4: Theoretical prediction (left) and actual average market prices (right)

A comparison of our results to the existing literature shows a similar pattern of convergence with respect to the number of trades and the market price: In his first experimental markets, Smith (1962) shows that the predicted quantity and the predicted price arise within the first rounds of trading. Later studies consistently replicated that the DA mechanism is characterized by a fast convergence toward the theoretically predicted market price and highly efficient outcomes, especially in comparison to other market institutions such as a posted-offer market (e.g. Smith 1976, Smith et al. 1982, Ketcham et al. 1984). In addition, we find a comparably low variation of market prices as in Smith (1962). By running the DA market online on AMT and replicating results similar to previous laboratory experiments, we can further validate online subject pools as a reliable data source (for the discussion of the reliability of data from online experiments see e.g. Suri

& Watts 2011, Mason & Suri 2012, Arechar et al. 2018).

Attrition and Bots

One major challenge of conducting interactive games online on platforms such as AMT is that participants might leave the experiment at an early stage (even if this means that they are excluded from all payments), which the experimenter cannot influence. To cope with this problem of attrition, we implemented computerized players, so called bots.[9] These bots substitute human players once they close their web session. Bots are programmed such that they submit an ask equal to their production costs (as a seller) or a bid equal to their valuation (as a buyer) at a random point of time during the remaining seconds of the round. To comply with the rule of no-deception of study participants, the existence and trading strategy of bots is common knowledge to all participants and bots are indicated as such. By the use of bots, we can ensure that the experiment continues despite participant dropouts and allow the remaining players to continue without interruption. The code, however, allows to disable the bot service and thereby provides maximal flexibility for users who want to cope with attrition differently.[10] In this case, players who dropped out are labeled as "inactive" to other participants. Additionally, participants only received payment when they completed the whole experiment and we paid participants only for one randomly selected round to keep

[9]oTree commonly uses the term *bot* to express automated testing of the experiment by inputs done by computerized players. We use the term for automated players who are playing in real-time with human players.

[10]Including bots might lead to a higher number of trades than theoretically predicted, as the bots are programmed such that they do not make a surplus by trading but only make bids/asks equal to their valuations/production costs. Switching off the bot service therefore provides a tool to exclude this dynamic.

the importance of participating in every round high.

In the following, we examine attrition in our subject pool in greater detail. We can distinguish between two types of attrition: Attrition *before* the game, i.e. while participants are in the waiting room or reading the instructions, and attrition *during* the game, i.e. while participants are trading. Although participants were told at the beginning that the experiment would start after at most 15 minutes, most attrition (26 subjects or 18.06%) occurred during the time allocated to wait or read the instructions. The frequency of dropouts during the game is rather negligible. 116 of 118 participants (98.3%) participants who started the first round of the DA market also completed the last round. Of those 116 participants, only 4 participants were absent for some rounds during the game but reconnected before the last round. This suggests that subjects do not make noticeable use of dis- and reconnecting to the experiment in a strategic way.

As we experienced a varying influx of participants upon publishing the HIT, we chose to group participants into a market before giving access to the instructions. We wanted to avoid that after reading the instructions we could not start the market with sufficient participants and either pay for lost observations or deny the remaining participants payment. Based on our experience with low attrition during the game, a group formation after presenting the instructions might be an interesting alternative for future users to reach a high number of (human) participants.

5 Conclusion

In this paper, we illustrate the implementation of websockets in oTree that allow for real-time interactions. By running a DA market on AMT, we test both the replicability of basic results from the existing literature and the technical functionality of the DA market module. We find that participants from AMT behave comparably to participants in the laboratory, as we find similar convergence patterns in the DA market. In addition, we could verify that the technology works smoothly. We provide the code for the DA market module open-source on GitHub and encourage researchers and teachers to use and improve it to custom needs. For example, the extension to a multi-unit market would be a valuable contribution. In addition, we suggest to use websockets in oTree to investigate further synchronous, real-time games. Integrating websockets in games such as a dynamic public goods game or an asset market allows to run large-scale experiments and can thereby enrich behavioral experimental research.

References

Adrian, N., Crede, A.-K., & Gehrlein, J. (2019). Do markets foster consequentialist decisions? Evidence from an online experiment. *Working Paper*.

Arechar, A. A., Gaechter, S., & Molleman, L. (2018). Conducting interactive experiments online. *Experimental Economics*, *21*(1), 99–131.

Balietti, S. (2017). nodegame: Real-time, synchronous, online experiments in the browser. *Behavior Research Methods*, *49*(5), 1696–1715.

Chen, D. L., Schonger, M., & Wickens, C. (2016). oTree–An open-source platform for laboratory, online, and field experiments. *Journal of Behavioral and Experimental Finance*, *9*, 88–97.

Fischbacher, U. (2007). z-Tree: Zurich toolbox for ready-made economic experiments. *Experimental Economics*, *10*(2), 171–178.

Holzmeister, F. (2017). oTree: Ready-made apps for risk preference elicitation methods. *Journal of Behavioral and Experimental Finance*, *16*, 33–38.

Holzmeister, F., & Pfurtscheller, A. (2016). otree: The "bomb" risk elicitation task. *Journal of Behavioral and Experimental Finance*, *10*, 105–108.

Ketcham, J., Smith, V. L., & Williams, A. W. (1984). A comparison of posted-offer and double-auction pricing institutions. *The Review of Economic Studies*, *51*(4), 595–614.

Mason, W., & Suri, S. (2012). Conducting behavioral research on Amazon's Mechanical Turk. *Behavior Research Methods*, *44*(1), 1–23.

Miller, R. M., Plott, C. R., & Smith, V. L. (1977). Intertemporal competitive equilibrium: An empirical study of speculation. *The Quarterly Journal of Economics*, *91*(4), 599–624.

Nuzzo, S., & Morone, A. (2017). Asset markets in the lab: A literature review. *Journal of Behavioral and Experimental Finance*, *13*, 42–50.

Palan, S. (2015). GIMS–Software for asset market experiments. *Journal of Behavioral and Experimental Finance*, *5*, 1–14.

Pettit, J., Friedman, D., Kephart, C., & Oprea, R. (2014). Software for continuous game experiments. *Experimental Economics*, *17*(4), 631–648.

Plott, C. R., & Smith, V. L. (1978). An experimental examination of two exchange institutions. *The Review of Economic Studies*, *45*(1), 133–153.

Plott, C. R., & Sunder, S. (1988). Rational expectations and the aggregation of diverse information in laboratory security markets. *Econometrica*, *56*(5), 1085–1118.

Smith, V. L. (1962). An experimental study of competitive market behavior. *Journal of Political Economy*, *70*(2), 111–137.

Smith, V. L. (1976). Bidding and auctioning institutions: Experimental results. In Y. Amihud (Ed.), *Bidding and auctioning for procurement and allocation.* New York University Press.

Smith, V. L., Suchanek, G. L., & Williams, A. W. (1988). Bubbles, crashes, and endogenous expectations in experimental spot asset markets. *Econometrica*, *56*(5), 1119–1151.

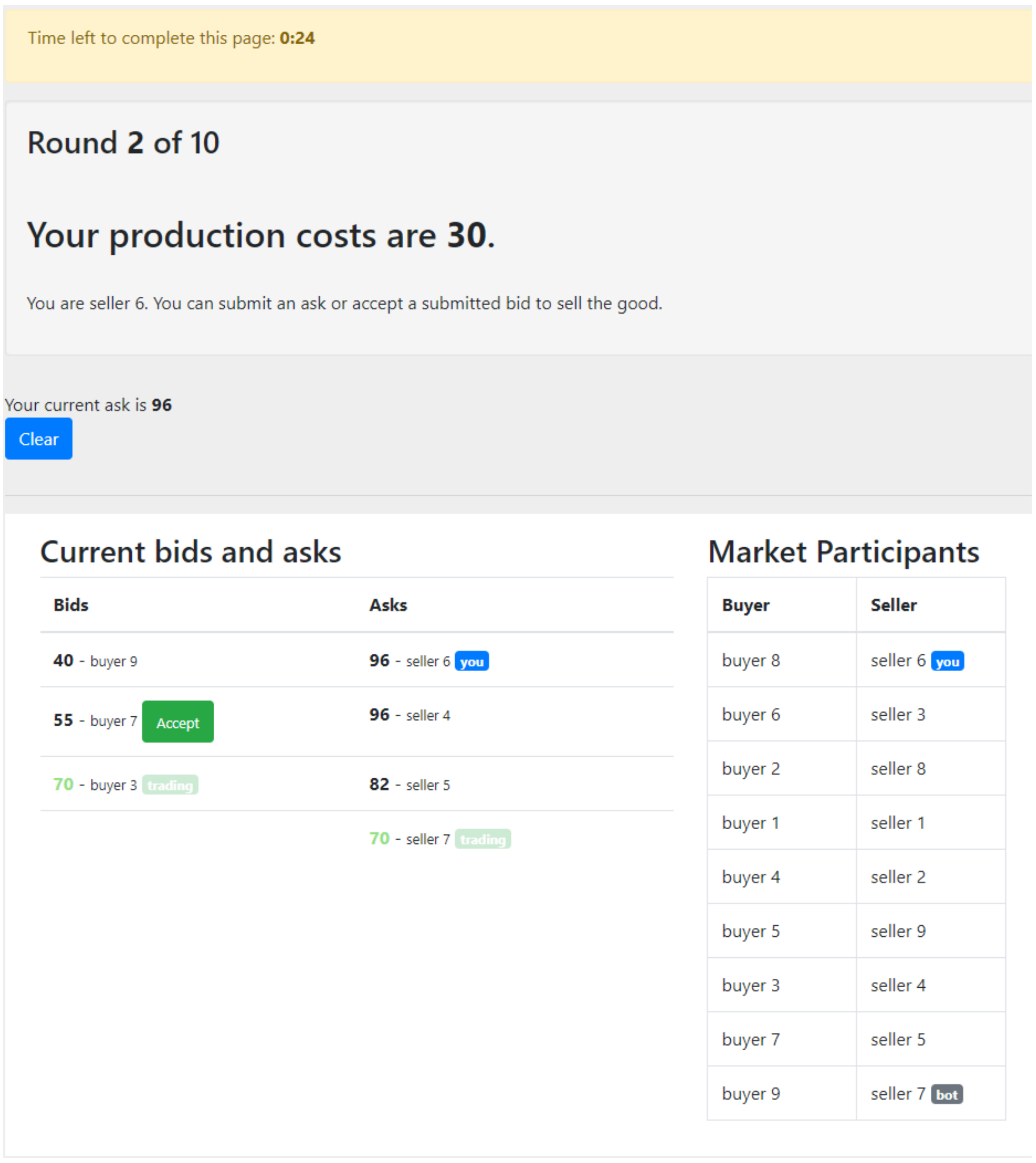
Smith, V. L., Williams, A. W., Bratton, W. K., & Vannoni, M. G. (1982). Competitive market institutions: Double auctions vs. sealed bid-offer auctions. *The American Economic Review*, *72*(1), 58–77.

Suri, S., & Watts, D. J. (2011). Cooperation and contagion in web-based, networked public goods experiments. *PloS one*, *6*(3), e16836.

Williams, A. W. (1980). Computerized double-auction markets: Some initial experimental results. *Journal of Business*, *53*(3), 235–258.

Appendix

Graphical User Interface of the DA market



Time left to complete this page: **0:24**

Round 2 of 10

Your production costs are 30.

You are seller 6. You can submit an ask or accept a submitted bid to sell the good.

Your current ask is **96**

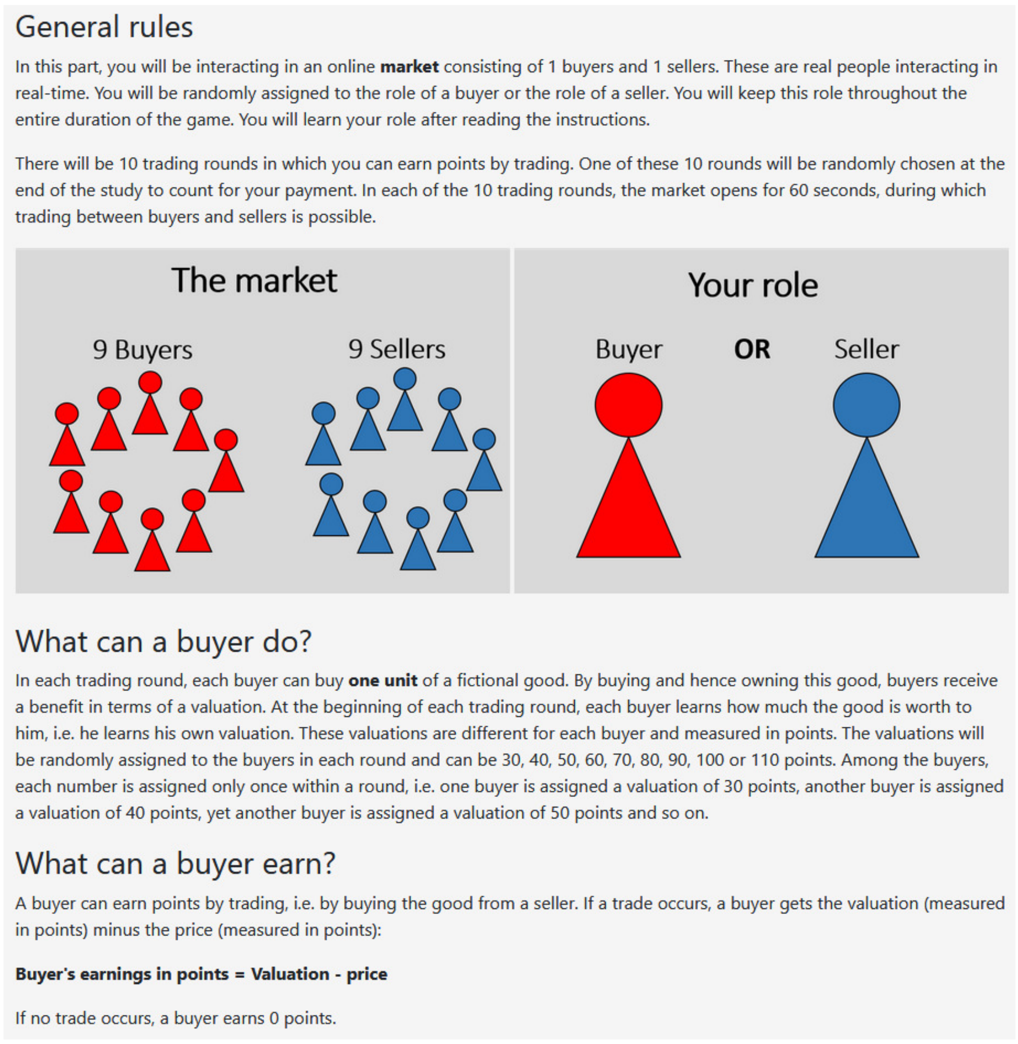
Current bids and asks

Bids	Asks
40 - buyer 9	**96** - seller 6 you
55 - buyer 7 Accept	**96** - seller 4
70 - buyer 3 trading	**82** - seller 5
	70 - seller 7 trading

Market Participants

Buyer	Seller
buyer 8	seller 6 you
buyer 6	seller 3
buyer 2	seller 8
buyer 1	seller 1
buyer 4	seller 2
buyer 5	seller 9
buyer 3	seller 4
buyer 7	seller 5
buyer 9	seller 7 bot

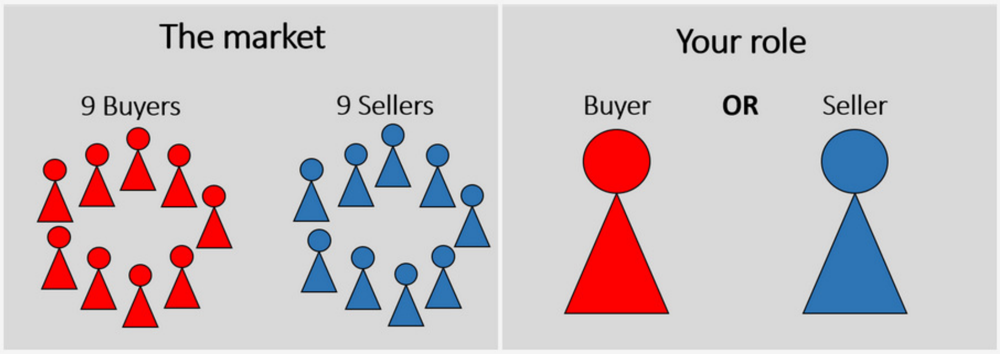
Instructions of the DA market



General rules

In this part, you will be interacting in an online **market** consisting of 1 buyers and 1 sellers. These are real people interacting in real-time. You will be randomly assigned to the role of a buyer or the role of a seller. You will keep this role throughout the entire duration of the game. You will learn your role after reading the instructions.

There will be 10 trading rounds in which you can earn points by trading. One of these 10 rounds will be randomly chosen at the end of the study to count for your payment. In each of the 10 trading rounds, the market opens for 60 seconds, during which trading between buyers and sellers is possible.



What can a buyer do?

In each trading round, each buyer can buy **one unit** of a fictional good. By buying and hence owning this good, buyers receive a benefit in terms of a valuation. At the beginning of each trading round, each buyer learns how much the good is worth to him, i.e. he learns his own valuation. These valuations are different for each buyer and measured in points. The valuations will be randomly assigned to the buyers in each round and can be 30, 40, 50, 60, 70, 80, 90, 100 or 110 points. Among the buyers, each number is assigned only once within a round, i.e. one buyer is assigned a valuation of 30 points, another buyer is assigned a valuation of 40 points, yet another buyer is assigned a valuation of 50 points and so on.

What can a buyer earn?

A buyer can earn points by trading, i.e. by buying the good from a seller. If a trade occurs, a buyer gets the valuation (measured in points) minus the price (measured in points):

Buyer's earnings in points = Valuation - price

If no trade occurs, a buyer earns 0 points.

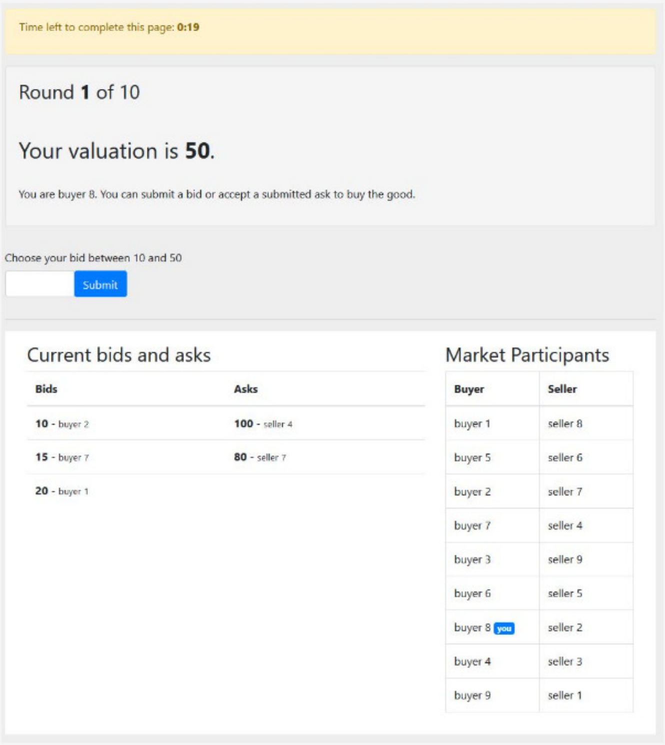
How does trading work for the buyer?

Trading is done on an online market platform. A buyer can trade in two possible ways:

1. He can accept an ask that has been submitted by a seller. The trade then occurs at the price of the ask.
2. Alternatively, he can submit a bid, i.e. the price at which he is willing to buy. If a seller accepts this bid or submits a lower ask, the trade occurs at the price of this bid.

The two possible ways of trading will be explained in more detail later on the screen.

The following screenshot shows how the online market platform looks like:



Time left to complete this page: **0:19**

Round **1** of 10

Your valuation is **50**.

You are buyer 8. You can submit a bid or accept a submitted ask to buy the good.

Choose your bid between 10 and 50

Current bids and asks

Bids	Asks
10 - buyer 2	**100** - seller 4
15 - buyer 7	**80** - seller 7
20 - buyer 1	

Market Participants

Buyer	Seller
buyer 1	seller 8
buyer 5	seller 6
buyer 2	seller 7
buyer 7	seller 4
buyer 3	seller 9
buyer 6	seller 5
buyer 8 you	seller 2
buyer 4	seller 3
buyer 9	seller 1

In each trading round, buyers are numbered consecutively from 1 to 9. The numbers change each round such that no buyer can be identified. In the example, the buyer has number 8. The valuation of the buyer in this round is 50, as you can see from the message on the screen "*Your valuation is 50*". You see a list with all market participants at the right side of the screen. Bids and asks of the buyers and sellers are displayed in the table "*Current bids and asks*".

At the beginning of each round, there is a countdown of 10 seconds during which each buyer learns his valuation. Then the market opens for 60 seconds. While the market is open, each buyer can trade one unit of the good by accepting an ask of a seller or by submitting a bid (these are the two possible ways of trading shortly described before):

1. **Each buyer can accept an ask** from the table "*Current bids and asks*". He does so by clicking on the accept button that shows up next to the lowest ask in the table. The good then trades for the price of the ask.

2. Alternatively, **each buyer can submit a bid**, i.e. a price at which he is willing to buy the good. In order to do so, he can enter a value and click on *Submit*. The bid then appears in the table "*Current bids and asks*" and is visible to all sellers and buyers. Within a trading round, a buyer can revise his bid as many times as he likes and replace it by a new one. If a seller accepts the bid of the buyer, trade occurs at the price of the bid. To avoid a loss, a buyer can only submit bids that are equal to or lower than his valuation.

   If a buyer submits a bid and there are lower asks in the table, trade occurs at the price of the lowest ask. In principle, it is the same as if the buyer had directly accepted the lowest (and thus currently best) ask in the table.

When the market closes, each buyer receives feedback about his payoff and all trades from that round.

What can a seller do?

In each trading round, each seller can produce **one unit** of a fictional good that he can sell in the market. At the beginning of each trading round, each seller learns how much it costs for him to produce this good, i.e. he learns his own production costs. These production costs are measured in points. They will be randomly assigned to the sellers in each round and can be 10, 20, 30, 40, 50, 60, 70, 80 or 90 points. Among the sellers, each number is assigned only once within a round, i.e. one seller is assigned production costs of 10 points, another seller is assigned production costs of 20 points, yet another seller is assigned production costs of 30 points and so on.

What can a seller earn?

A seller can earn points by trading, i.e. by selling the good to a buyer. If a trade occurs, a seller gets the price (measured in points) minus the production costs (measured in points):

Seller's earnings in points = Price – production costs

If no trade occurs, the good is not produced, i.e. the seller does not pay the production costs. Thus, if no trade occurs, a seller earns 0 points.

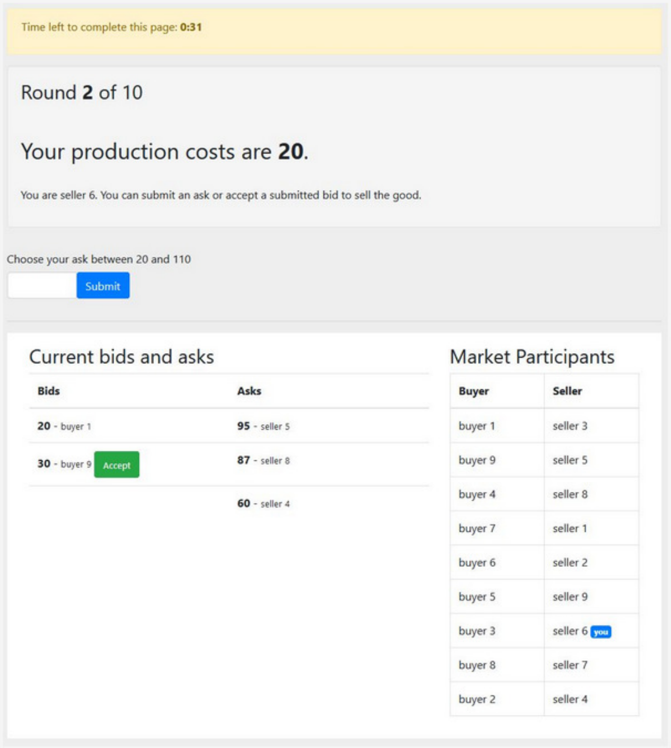
How does trading work for the seller?

Trading is done on an online market platform. A seller can trade in two possible ways:

1. He can accept a bid that has been submitted by a buyer. The trade then occurs at the price of this bid.
2. Alternatively, he can submit an ask, i.e. the price at which he is willing to sell. If a buyer accepts this ask or submits a higher bid, the trade occurs at the price of this ask.

The two possible ways of trading will be explained in more detail later on the screen.

The following screenshot shows how the online market platform looks like:



Time left to complete this page: **0:31**

Round **2** of 10

Your production costs are **20**.

You are seller 6. You can submit an ask or accept a submitted bid to sell the good.

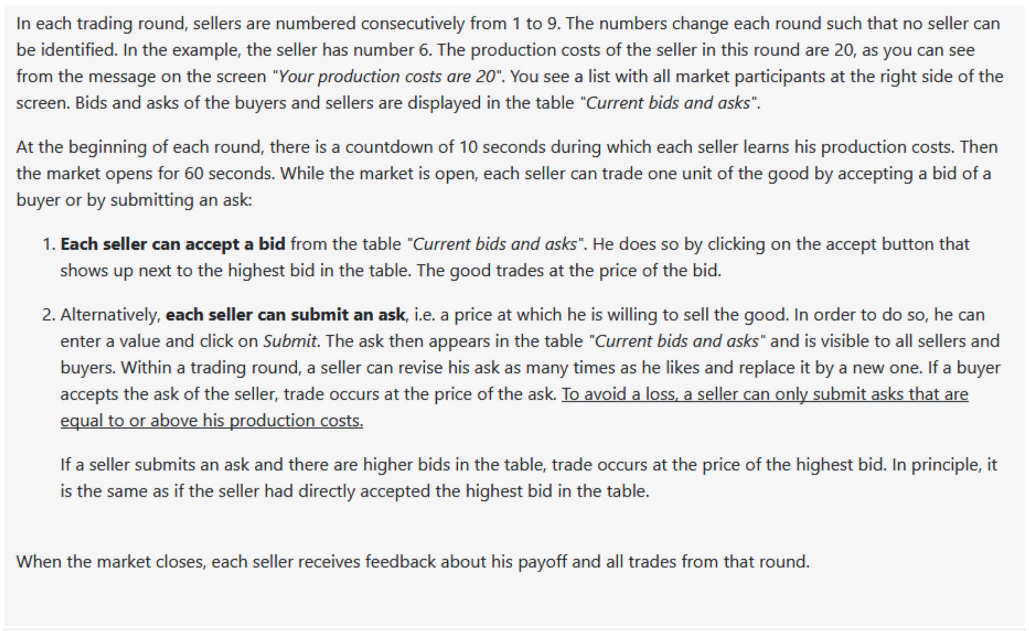
Choose your ask between 20 and 110

Current bids and asks

Bids	Asks
20 - buyer 1	**95** - seller 5
30 - buyer 9 Accept	**87** - seller 8
	60 - seller 4

Market Participants

Buyer	Seller
buyer 1	seller 3
buyer 9	seller 5
buyer 4	seller 8
buyer 7	seller 1
buyer 6	seller 2
buyer 5	seller 9
buyer 3	seller 6 you
buyer 8	seller 7
buyer 2	seller 4



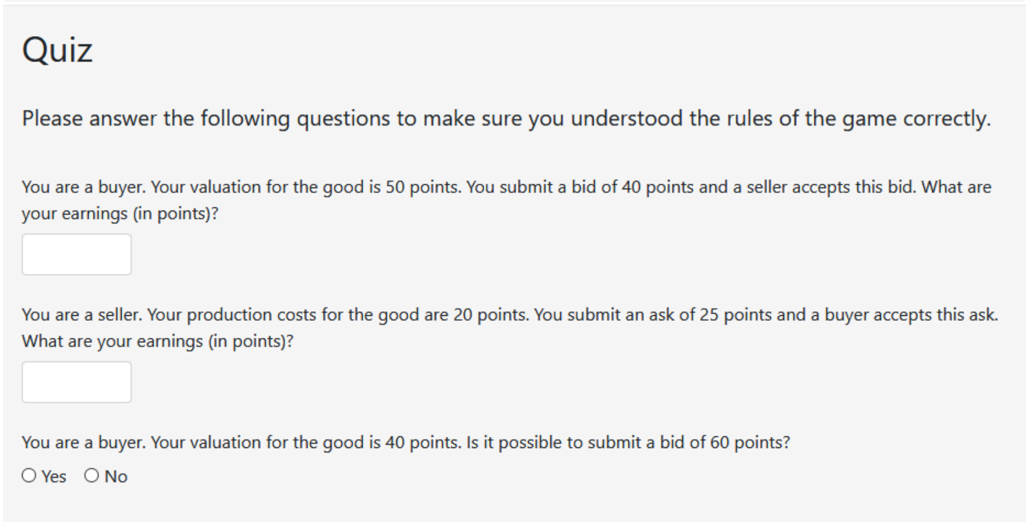
In each trading round, sellers are numbered consecutively from 1 to 9. The numbers change each round such that no seller can be identified. In the example, the seller has number 6. The production costs of the seller in this round are 20, as you can see from the message on the screen "*Your production costs are 20*". You see a list with all market participants at the right side of the screen. Bids and asks of the buyers and sellers are displayed in the table "*Current bids and asks*".

At the beginning of each round, there is a countdown of 10 seconds during which each seller learns his production costs. Then the market opens for 60 seconds. While the market is open, each seller can trade one unit of the good by accepting a bid of a buyer or by submitting an ask:

1. **Each seller can accept a bid** from the table "*Current bids and asks*". He does so by clicking on the accept button that shows up next to the highest bid in the table. The good trades at the price of the bid.
2. Alternatively, **each seller can submit an ask**, i.e. a price at which he is willing to sell the good. In order to do so, he can enter a value and click on *Submit*. The ask then appears in the table "*Current bids and asks*" and is visible to all sellers and buyers. Within a trading round, a seller can revise his ask as many times as he likes and replace it by a new one. If a buyer accepts the ask of the seller, trade occurs at the price of the ask. To avoid a loss, a seller can only submit asks that are equal to or above his production costs.

   If a seller submits an ask and there are higher bids in the table, trade occurs at the price of the highest bid. In principle, it is the same as if the seller had directly accepted the highest bid in the table.

When the market closes, each seller receives feedback about his payoff and all trades from that round.



Quiz

Please answer the following questions to make sure you understood the rules of the game correctly.

You are a buyer. Your valuation for the good is 50 points. You submit a bid of 40 points and a seller accepts this bid. What are your earnings (in points)?

You are a seller. Your production costs for the good are 20 points. You submit an ask of 25 points and a buyer accepts this ask. What are your earnings (in points)?

You are a buyer. Your valuation for the good is 40 points. Is it possible to submit a bid of 60 points?

○ Yes ○ No