Design and Implementation of Online Experiments

nodeGame.org

Stefano Balietti

MZES and Heidelberg

More nodeGame

@balietti
@nodegameorg
stefanobalietti.com@gmail.com

Game's Anatomy

Configuration and game files are separated different folders

- auth
- channel
- data
- game
- levels
- public
- requirements
- views
- waitroom
- .gitignore
- package.json

- Define treatments
- Define the sequence of stages and steps of the experiment
- Implement the rules of interactions among participants
- General setup

- Define the graphical contents of the game

- Define the rules about how to assign treatments to game rooms, and participants to treatments

Folder waitroom/

Waiting-Room-v5

Change waiting room options

Start 2 groups simultaneously with random treatment assignment (hint: POOL_SIZE)

waitroom.settings.js

```
// How many clients must connect before groups are formed
POOL_SIZE: 2,

// The size of each group
GROUP_SIZE: 2,
```

Folder waitroom/

Waiting-Room-v5

Change waiting room options

Start 2 groups simultaneously with random treatment assignment (hint: POOL_SIZE)

waitroom.settings.js

```
// How many clients must connect before groups are formed
POOL_SIZE: 2,

// The size of each group
GROUP_SIZE: 2,
```

A larger pool size allows you to:

- Reshuffle groups after each experiment
- Ensure that randomization is effective (e.g., distribute fast participants across treatments, or other forms of stratified assignment).

Reading Errors

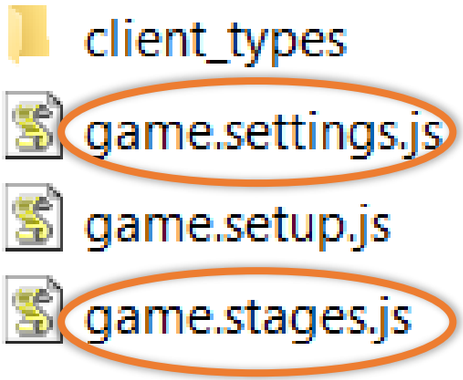
```
error: GameLoader.buildWaitRoomConf: error reading waitroom.settings file. Game: ultimatum. Err:
C:\Users\balistef\www\nodegame-v5.0.0-dev\games\ultimatum-game\waitroom\waitroom.settings.js:46
    GROUP_SIZE: 2,
    ^^^^^^^^^^

SyntaxError: Unexpected identifier
    at Module._compile (internal/modules/cjs/loader.js:721:23)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:787:10)
    at Module.load (internal/modules/cjs/loader.js:653:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
    at Function.Module._load (internal/modules/cjs/loader.js:585:3)
    at Module.require (internal/modules/cjs/loader.js:690:17)
    at require (internal/modules/cjs/helpers.js:25:18)
    at GameLoader.buildWaitRoomConf (C:\Users\balistef\www\nodegame-v5.0.0-dev\node_modules\nodegame-server\lib\GameLoader.js:926:16)
    at GameLoader.loadWaitRoomDir (C:\Users\balistef\www\nodegame-v5.0.0-dev\node_modules\nodegame-server\lib\GameLoader.js:987:17)
    at GameLoader.addGame (C:\Users\balistef\www\nodegame-v5.0.0-dev\node_modules\nodegame-server\lib\GameLoader.js:163:10)
```

File Name and Line Number

Note! The line number is not always where the error actually lies. In fact, it is often on a subsequent line. Here, without the comma, the compiler does not even know that a line ended. When you forget a parenthesis, the errored line number can be the last line of the file, which makes it very difficult to find the error's actual position.

Folder game/



Define game variables and group them into treatments

Define the sequence of stages and steps of the experiment

Stages Definition

Stager-API-v5

game.stages.js

We use the stager API to define the sequence (the order matters here!)

A sequence contains stages, and stages contain steps

```
stager
    .next('id_of_stage')

stager
    .step('id_of_step1_within_stage')
    .step('id_of_step2_within_stage')

stager
    .repeat('id_stage_to_repeat', 3)
```

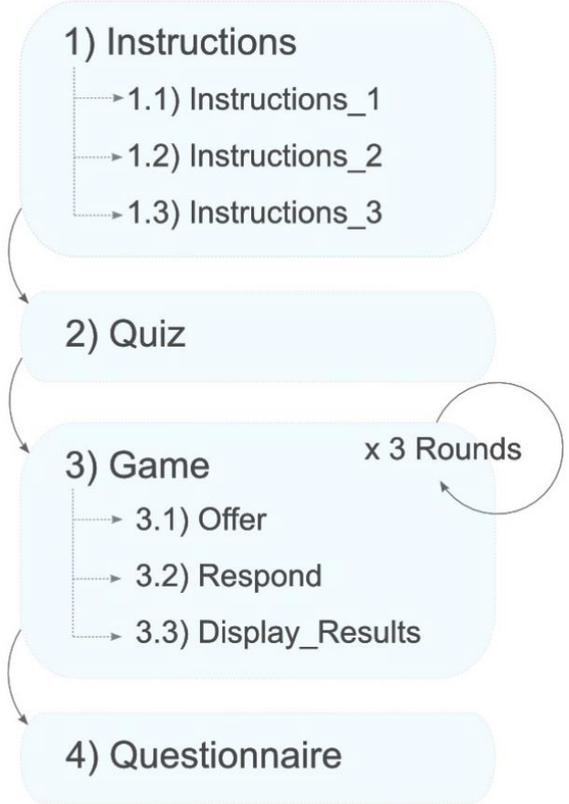
Here, we "chain" two method calls together. We can do it, because each method is returning a `stager` object, so it is a more compact way of writing:

```
stager.step('id_of_step1_within_stage');
stager.step('id_of_step2_within_stage');
```

Stages Definition

Stager-API-v5

Game Sequence



Code Snippet

```
stager.stage("instructions")
      .step("instructions_1")
      .step("instructions_2")
      .step("instructions_3");

stager.stage("quiz");

stager.repeat("game", 3)

stager.stage("questionnaire");

stager.extendStage("game", {
    steps: [ "offer",
             "respond",
             "display_results" ]
});
```

Skipping Stages and Steps

- Skip some stages of the game sequence (very useful for debugging)

game.stages.js

```
// Skip stages from the sequence.
stager.skip('precache');
stager.skip('selectLanguage');
stager.skip('quiz');
stager.skip('instructions');
stager.skip('mood');
// Skip the step from the sequence.
stager.skip('ultimatum', 'bidder');
```

Settings and Treatments

Settings-and-Treatments-v5

game.settings.js

Multiline Comments

```
/**
 * # Game settings: Ultimatum Game
 * Copyright(c) 2018 Stefano Balietti <ste@nodegame.org>
 * MIT Licensed
 *
 * http://www.nodegame.org
 */

module.exports = {

    // Minimum number of players that must be always connected.
    MIN_PLAYERS: 2,

    // Number or rounds to repeat the bidding. *
    REPEAT: 2,
```

Node.JS makes the content of this object available outside of the file

Game variables (these two are game-specific)

Settings and Treatments

Settings-and-Treatments-v5

game.settings.js

```
// TIMER.
// If the name of a key of the TIMER object matches the name of one
// of the steps or stages, its value is automatically used as the
// value of the `timer` property of that step/stage.
//
// The timer property is read by `node.game.timer` and by VisualTimer
// widgets, if created. It can be:
//
//  - a number (in milliseconds),
//  - a function returning the number of milliseconds,
//  - an object containing properties _milliseconds_, and _timeup_
//      the latter being the name of an event to emit or a function
//      to execute when the timer expires. If _timeup_ is not set,
//      property _timeup_ of the game step will be used.
TIMER: {
    selectLanguage: 30000,
    instructions: 90000,
    quiz: 60000,
    mood: 60000,
    questionnaire: 90000,
    bidder: 30000,
    respondent: 30000
},
```

- Variable ***TIMER*** is read by nodeGame's engine
- It defines the max duration (in milliseconds) of each step
- The names of the properties are the ids of the steps in the sequence

Settings and Treatments

Settings-and-Treatments-v5

game.settings.js

```
// Available treatments:
treatments: {

    standard: {
        description: "More time to wait and no peer pressure.",
        WAIT_TIME: 20,
        instructionsPage: 'instructions.html'
    },

    pp: {
        description:
            "Introduces peer pressure to players to not disconnect.",
        WAIT_TIME: 10,
        instructionsPage: 'instructions_pp.html'
    }
}
```

- Property ***treatments*** is read by nodeGame's engine
- It is an object indexed by the names of the treatments
- Each treatment contains *all the variables defined outside* of the treatments object, *plus what it is defined inside.*

Treatment names: they will appear in the selection box in the waiting room, together with the description. You are free to choose any name is appropriate for your experiment.

Settings and Treatments

Settings-and-Treatments-v5

game.settings.js

By assigning a property with the same name, but with different values we can define *controlled differences* in treatments.

```
// Available treatments:
treatments: {

    standard: {
        description: "More time to wait and no peer pressure.",
        WAIT_TIME: 20,
        instructionsPage: 'instructions.html'
    },

    pp: {
        description:
            "Introduces peer pressure to players to not disconnect.",
        WAIT_TIME: 10,
        instructionsPage: 'instructions_pp.html'
    }
}
```

`WAIT_TIME` controls how much time to wait for a disconnected player to reconnect. This property is read by nodeGame, which automatically adjusts the reconnect timer.

`instructionsPage` is a game variable that will use later when extending the steps.

Hands On 5

- Change the number of repetition of the ultimatum stage

game.settings.js

```
// Change the number of repetition to 1.
REPEAT: 1,
```

Hands On 5

game.settings.js

```
// Change the number of repetition to 1.
REPEAT: 1,
```

Reads

game.stages.js

```
module.exports = function(stager, settings) {

  stager
    .next('precache')
    .next('selectLanguage')
    .next('instructions')
    .next('quiz')
    .next('mood')
    .repeat('ultimatum', settings.REPEAT)
    .next('questionnaire')
    .next('endgame')
    .gameover();
```

Folder game/client_types/

- client_types
- game.settings.js
- game.setup.js
- game.stages.js

- Client types implement the sequence
- The same sequence can be implemented differently, depending by who is playing or where the code is going to be executed

- autoplay.js
- bot.js
- logic.js ← Computer (orchestrator), server
- player.js ← Human, web browser

Client Type player.js

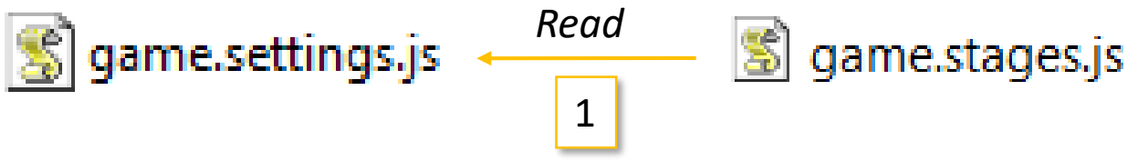
player.js

- Contains code that will be executed on the client (web browser)
  - Has access to all standard JS objects AND nodeGame client API
  - However, it is “compiled” on the server, so it also has access to some server objects

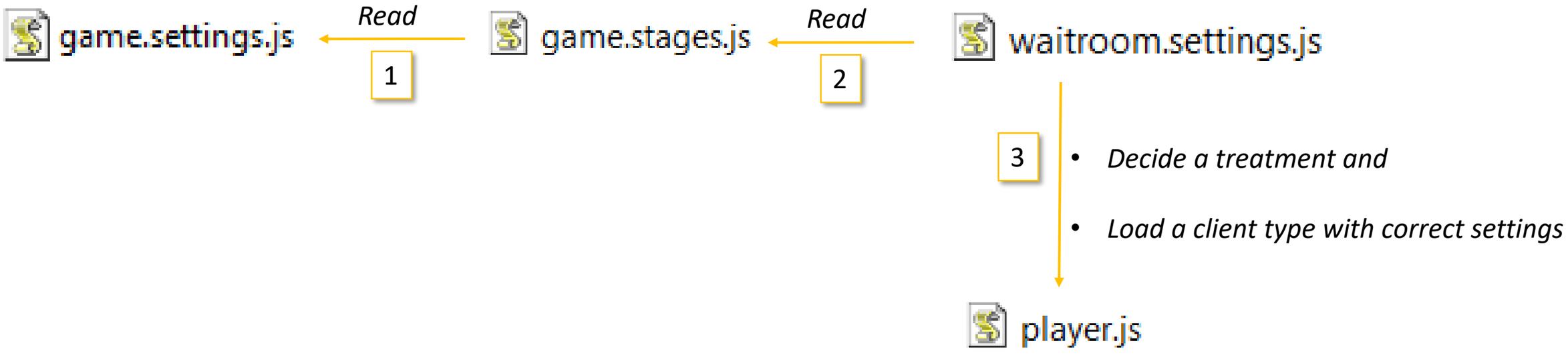
Information Flow for Client Type

game.settings.js

Information Flow for Client Type



Information Flow for Client Type



Information Flow for Client Type

game.settings.js ← *Read* (1) — game.stages.js ← *Read* (2) — waitroom.settings.js

(3)
- *Decide a treatment and*
- *Load a client type with correct settings*

player.js

Has available

```
module.exports = function(treatmentName, settings, stager, setup, gameRoom) {
```

(3) (1) (2)

How To Implement a Sequence

Client-Types-v5

player.js

- Use Stager API `stager.[method]`
  - **Initialize game**

```
stager.setOnInit(function() {
    // For instance, create Header and Frame, add widgets, etc.
});
```

  - **Add properties to stages and steps**

```
stager.extendStage("stageId", {
    foo: bar
});
stager.extendStep("stepId", {
    foo: bar2
});
```

Remember foo bar right?

https://en.wikipedia.org/wiki/Foobar

How To Implement a Sequence

Client-Types-v5

player.js

- Use Stager API `stager.[method]`

  - **Initialize game**

```
stager.setOnInit(function() {
    // For instance, create He
});
```

  - **Add properties to stages and steps**

```
stager.extendStage("ultimatum", {
    foo: bar
});
stager.extendStep("bidder", {
    foo: bar2
});
```

The name of the stage/step **must match** what is defined in the sequence.

A property defined at the stage level is shared with all the steps inside the same stage.

In this example, the step "bidder" is overwriting the value of foo defined at the stage level. *What is the value of foo in step "respondent"?*

For instance, game.stages.js

```
stager
    .next('selectLanguage')
    .next('instructions')
    .repeat('ultimatum', 2)
    .step('bidder')
    .step('respondent')
    .next('questionnaire')
```

The step-property `frame`

Client-Types-v5

Controls which HTML page is loaded in the *frame* for each step.

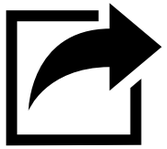
```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html'
});
```

Here, we fix the name to a file, which is in `public/`.

```
stager.extendStep('instructions', {
    frame: settings.instructionsPage
});
```

Here, we take the value from the settings object, so that the actual frame loaded is treatment-dependent.

The step-property `cb`



Client-Types-v5

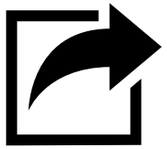
Cb is a shorthand for "Callback," which simply means function.
After the frame has been loaded, the callback is executed doing something on the page.

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
          // Let's do something in here. What?
     }
});
```

Don't forget commas, this is an object.

Note! `extendStage` cannot have a cb property

The step-property `cb`



Client-Types-v5

Important! Although the function is created on the server, **it is sent and executed on the client.** So, it has access to the DOM tree, the default JS objects as well as nodeGame objects.

Examples of JavaScript objects and methods we have already seen in this course.

```
document.body
document.createElement('div')
document.getElementById('myId')
location.href
alert
```

Full list available:

https://www.w3schools.com/jsref/obj_window.asp

The step-property `cb`

Client-Types-v5

Important! Although the function is created on the server, **it is sent and executed on the client.** So, it has access to the DOM tree, the default JS objects as well as nodeGame objects.

Examples of JavaScript objects and methods we have already seen in this course.

```
document.body
document.createElement('div')
document.getElementById('myId')
location.href
alert
```

Full list available:
https://www.w3schools.com/jsref/obj_window.asp

Main nodeGame objects

`J (JSUS = JS UtilS)`

- **Collection of helper functions, e.g. random integer numbers.**

`W (Window)`

- **Methods for manipulating the graphical interface**

`node (nodeGame)`

- **The entire nodeGame client API**
- `node.game` **contains all game-related methods and objects, including sequence, treatment settings, etc.**
- `node.widgets` **contains method to create widgets**

Adding Widgets to the Page

Widgets-v5

Here, we add a nodeGame Widget to select the language (as the step id suggests).

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
         node.game.lang = node.widgets.append('LanguageSelector',
                                                W.getFrameDocument().body);
     }
});
```

`node.widgets.append` takes 3 parameters: the name of the widget, where it should be appended, and an optional configuration object with options for the widget (not used here).

Adding Widgets to the Page

Widgets-v5

Here, we add a nodeGame Widget to select the language (as the step id suggests).

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
         node.game.lang = node.widgets.append('LanguageSelector',
                                              W.getFrameDocument().body);
     }
});
```

`node.widgets.append` takes 3 parameters: **the name of the widget**, where it should be appended, and an optional configuration object with options for the widget (not used here).

Adding Widgets to the Page

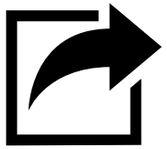
Widgets-v5

Here, we add a nodeGame Widget to select the language (as the step id suggests).

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
         node.game.lang = node.widgets.append('LanguageSelector',
                                                W.getFrameDocument().body);
     }
});
```

`node.widgets.append` takes 3 parameters: the name of the widget, **where it should be appended**, and an optional configuration object with options for the widget (not used here).

Adding Widgets to the Page



Widgets-v5

Here, we add a nodeGame Widget to select the language (as the step id suggests).

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
         node.game.lang = node.widgets.append('LanguageSelector',
                                                W.getFrameDocument().body);
     }
});
```

`node.widgets.append` takes 3 parameters: the name of the widget, where it should be appended, and an optional configuration object with options for the widget (not used here).

It returns a reference to the widget object, which we store with the name `lang` inside the `node.game` object. **Important!** `node.game` stores information that might be needed across steps. If you need to access the widget only within the same step, you might as well use a local variable (var lang = …) or avoid any assignment altogether.

Adding Widgets to the Page

Widgets-v5

Here, we add a nodeGame Widget to select the language (as the step id suggests).

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
         node.game.lang = node.widgets.append('LanguageSelector',
                                                W.getFrameDocument().body);
     }
});
```

`W` (stands for Window) is an object that helps to manipulate the user interface. Here, `W` is returning the `body` tag.

Why can't we use `document.body`? We could, but it would not be the `body` we expect. Let's learn what's going on behind the user interface.

Understanding the User Interface

Stage 2 / 7 **Time Left** 00:41 Done

Instructions of the Ultimatum Game

Please read them carefully.

This game is played in rounds by two human players randomly paired.

In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

The RESPONDENT can either accept or reject the offer of the BIDDER. If he / she accepts, both players split 100 ECU accordingly, else both get 0.

The game is repeated 2 rounds.

Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining ones will be paid only the show up fee.

If you understood the instructions correctly press the DONE button to proceed to the game.

Understanding the User Interface

Header

- **It is created at initialization**
- **It stays throughout the whole game**

Stage 2 / 7 **Time Left** 00:41 Done

Instructions of the Ultimatum Game

Please read them carefully.

This game is played in rounds by two human players randomly paired.

In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

The RESPONDENT can either accept or reject the offer of the BIDDER. If he / she accepts, both players split 100 ECU accordingly, else both get 0.

The game is repeated 2 rounds.

Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining ones will be paid only the show up fee.

If you understood the instructions correctly press the DONE button to proceed to the game.

Understanding the User Interface

Header

- **It is created at initialization**
- **It stays throughout the whole game**
- **Widgets such as timers, stage counters, and done button are generally added here**

Understanding the User Interface

Frame

- **It is created at initialization**
- **Its content is updated at every step according to the value of the `frame` step-property**

Stage 2 / 7 **Time Left** 00:41 Done

Instructions of the Ultimatum Game

Please read them carefully.

This game is played in rounds by two human players randomly paired.

In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

The RESPONDENT can either accept or reject the offer of the BIDDER. If he / she accepts, both players split 100 ECU accordingly, else both get 0.

The game is repeated 2 rounds.

Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining ones will be paid only the show up fee.

If you understood the instructions correctly press the DONE button to proceed to the game.

The Frame

Stage 2 / 7 **Time Left** 00:41 Done

Instructions of the Ultimatum Game

Please read them carefully.

This game is played in rounds by two human players randomly paired.

In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

The RESPONDENT can either accept or reject the offer of the BIDDER. If he / she accepts, both players split 100 ECU accordingly, else both get 0.

The game is repeated 2 rounds.

Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining one will be paid only the show up fee.

If you understood the instructions correctly press the DONE button to proceed to the game.

```
<!DOCTYPE html>
<html> event
▶ <head> ··· </head>
▼ <body>
  ▶ <div id="ng_header" class="ng_header_position-horizontal-t"> ··· </div>
  ▼ <iframe id="ng_mainframe" class="ng_mainframe-header-horizontal-t"
    name="ng_mainframe" scrolling="no" style="padding-top: 69px; min-
    height: 959px;" frameborder="0">
    ▼ #document
        <!DOCTYPE html>
        ▼ <html> event
          ▶ <head> ··· </head>
          ▼ <body>
            ▼ <div id="container">
                <h1>Instructions of the Ultimatum Game</h1>
                <div class="subtitle margin-bottom">
                Please read them carefully.</div>
              ▶ <div id="instructions"> ··· </div>
                If you understood the instructions correctly press the DONE
                button to proceed to the game.
              </div>
              ::after
            </body>
          </html>
    </iframe>
  ▶ <div id="ng_waitScreen" style="display: none;"> ··· </div>
    ::after
  </body>
</html>
```

Note! The iframe has a separate `document` object.

In the HTML language, the frame is an IFRAME tag, that is a completely separate HTML page within the parent page.

The Frame

Stage 2 / 7 **Time Left** 00:41 Done

Instructions of the Ultimatum Game

Please read them carefully.

This game is played in rounds by two human players randomly paired.

In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

The RESPONDENT can either accept or reject the offer of the BIDDER. If he / she accepts, both players split 100 ECU accordingly, else both get 0.

The game is repeated 2 rounds.

Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining one will be paid only the show up fee.

If you understood the instructions correctly press the DONE button to proceed to the game.

```
<!DOCTYPE html>
<html> event
  <head> ··· </head>
  <body>
    <div id="ng_head
    <iframe id="ng_
      name="ng_mainfra
      height: 959px;"
      #document
        <!DOCTYPE html>
        <html> event
          <head> ··· </head>
          <body>
            <div id="container">
              <h1>Instructions of the Ultimatum Game</h1>
              <div class="subtitle margin-bottom">
              Please read them carefully.</div>
              <div id="instructions"> ··· </div>
              If you understood the instructions correctly press the DONE
              button to proceed to the game.
            </div>
            ::after
          </body>
        </html>
      </iframe>
    <div id="ng_waitScreen" style="display: none;"> ··· </div>
    ::after
  </body>
</html>
```

The `node` object lives inside the parent page (because it is a stable environment, does not change at every step).

Therefore, `document.body` refers to the parent `body`.

In the HTML language, the frame is an IFRAME tag, that is a completely separate HTML page within the parent page.

The Frame

Stage 2 / 7 **Time Left** 00:41 Done

Instructions of the Ultimatum Game

Please read them carefully.

This game is played in rounds by two human players randomly paired.

In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

The RESPONDENT can either accept or reject the offer of the BIDDER. If he / she accepts, both players split 100 ECU accordingly, else both get 0.

The game is repeated 2 rounds.

Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining one will be paid only the show up fee.

If you understood the instructions correctly press the DONE button to proceed to the game.

```
<!DOCTYPE html>
<html> event
▶ <head> ··· </head>
▼ <body>
  ▶ <div id="ng_head
  ▼ <iframe id="ng_m
    name="ng_mainfra
    height: 959px;"
    ▼ #document
      <!DOCTYPE ht
      ▼ <html> event
        ▼ <body>
            <h1>Instructions of the Ultimatum Game</h1>
            <div class="subtitle margin-bottom">
            Please read them carefully.</div>
          ▶ <div id="instructions"> ··· </div>
            If you understood the instructions correctly press the DONE
            button to proceed to the game.
          </div>
          ::after
        </body>
      </html>
    </iframe>
  ▶ <div id="ng_waitScreen" style="display: none;"> ··· </div>
  ::after
</body>
</html>
```

The `node` object lives inside the parent page (because it is a stable environment, does not change at every step).

Therefore, `document.body` refers to the parent `body`.

To access elements of the frame, we need to use the `W` object, which takes care of most issues for us.

In the HTML language, the frame is an IFRAME tag, that is a completely separate HTML page within the parent page.

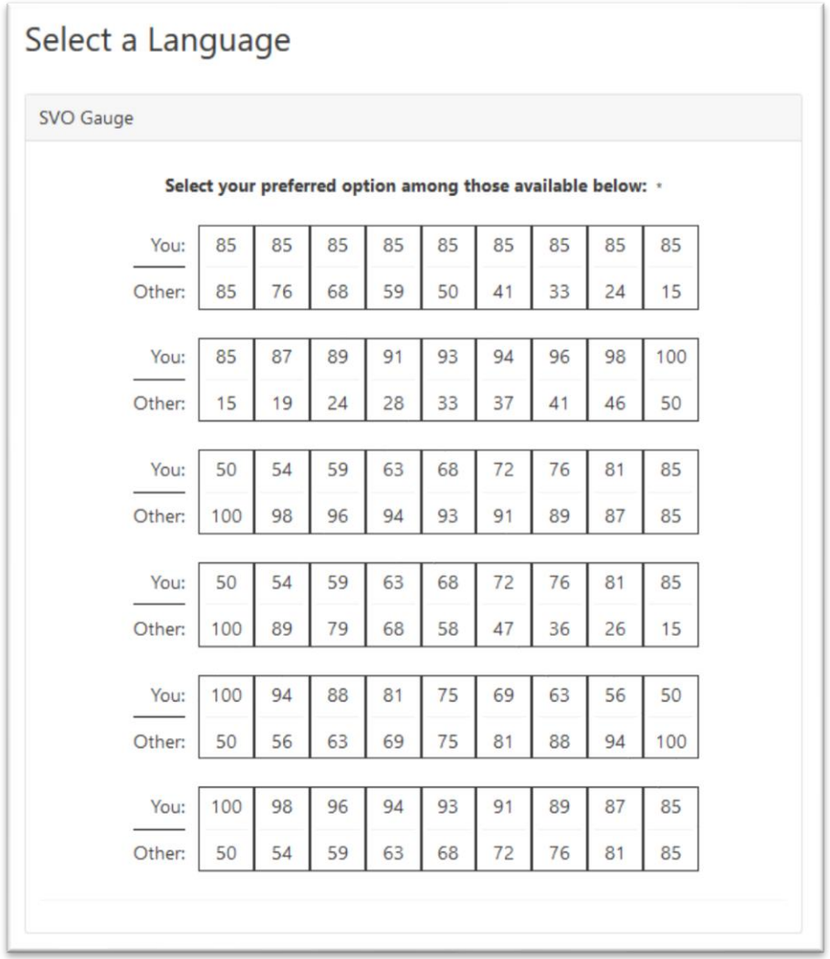
Adding Widgets to the Page

Widgets-v5

Here, we remove the LanguageSelector widget and we try out the **SVOGauge** widget, which measures social value orientation.

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
          // Store a reference to the body.
          var body = W.getFrameDocument().body;
          node.widgets.append('SVOGauge', body);
     }
});
```

Note! Here, we do not store the SVO widget in a variable, we simply add it to the page.



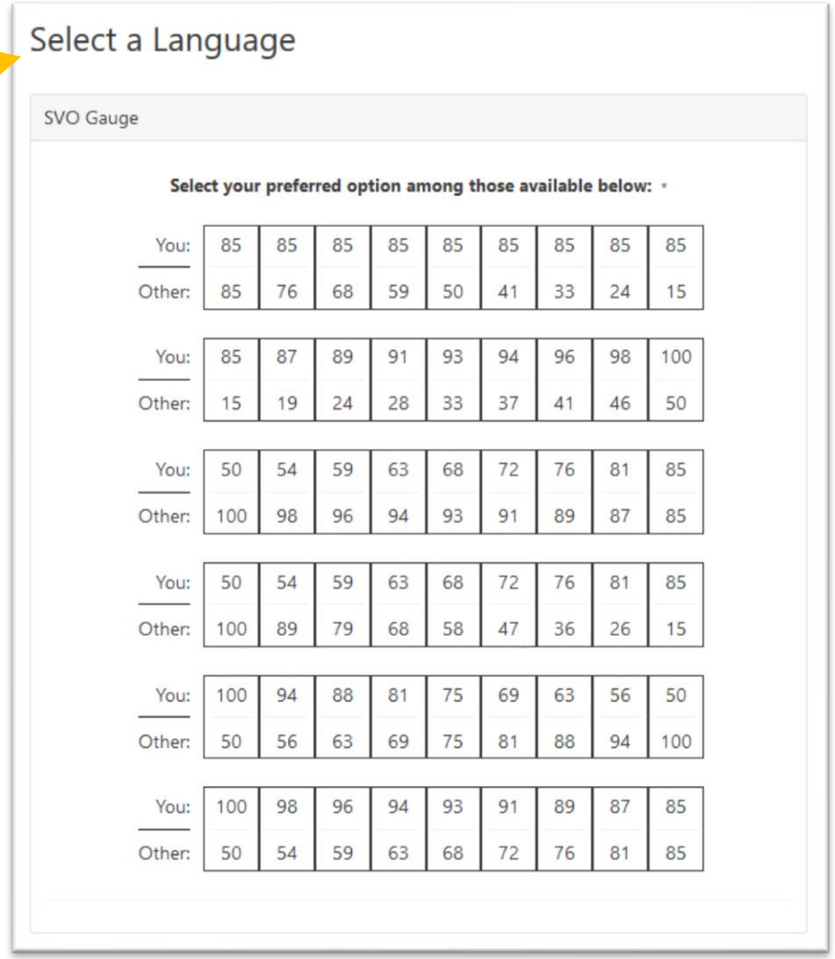
Adding Widgets to the Page

Widgets-v5

Here, we remove the LanguageSelector widget and we try out the **SVOGauge** widget, which measures social value orientation.

However, the title "Select a Language" is no longer appropriate. Let's change it in the HTML page.

```
stager.extendStep(
     frame: 'langua
     cb: function() {
          // Store a reference to the body.
          var body = W.getFrameDocument().body;
          node.widgets.append('SVOGauge', body);
     }
});
```



Modifying a Frame

- Implement Game Sequence: Change contents of frame to select the language
  - Go inside folder `public/`
  - Modify file `public/en/languageSelection.html`

Original content:

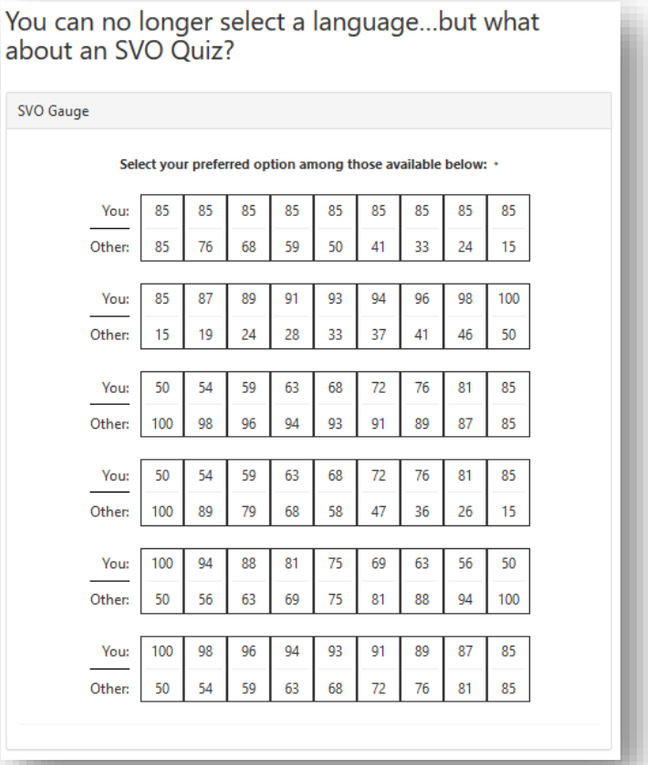
```
<!DOCTYPE html>
<title>Select a Language</title>
<link rel="stylesheet" type="text/css" href="/lib/bootstrap/bootstrap.min.css"/>
<link rel="stylesheet" type="text/css" href="/stylesheets/nodegame.css"/>
<link rel="stylesheet" type="text/css" href="../css/style.css"/>
<body class="centered">
    <h1 class='margin-bottom'>Select a Language</h1>
</body>
```

Modifying a Frame

```
<!DOCTYPE html>
<title>Select a Language</title>
<link rel="stylesheet" type="text/css" href="/lib/bootstrap/bootstrap.min.css"/>
<link rel="stylesheet" type="text/css" href="/stylesheets/nodegame.css"/>
<link rel="stylesheet" type="text/css" href="../css/style.css"/>
<body class="centered">
    <h1 class='margin-bottom'>Select a Language</h1>
</body>
```

You can no longer select a language…but what about an SVO Quiz?

The `h1` tag is a display tag for "headings," i.e., titles. There are different heading size from 1 the biggest, to 6 the smallest.



Treatment-Dependent Display

We want to display some information which depends on the chosen treatment.
Let's add a variable called `salutation` in each treatment.

game.settings.js

```
// Available treatments:
// (there is also the "standard" treatment, using the options above)
treatments: {

    standard: {
        description: "More time to wait and no peer pressure.",
        WAIT_TIME: 20,
        salutation: 'Hi there!',
        instructionsPage: 'instructions.html'
    },

    peerPressure: {
        description:
            "Introduces peer pressure to players to not disconnect.",
        WAIT_TIME: 10,
        salutation: 'Good Evening Mr.',
        instructionsPage: 'instructions_pp.html'
    }
}
```

Treatment-Dependent Display

We want to display some information which depends on the chosen treatment.
Let's add a variable called `salutation` in each treatment.

game.settings.js

```
// Available treatments:
// (there is also the "standard" treatment, using the options above)
treatments: {

    standard: {
        description: "More time to wait and no peer pressure.",
        WAIT_TIME: 20,
        salutation: 'Hi there!',
        instructionsPage: 'instructions.html'
    },

    peerPressure: {
        description:
            "Introduces peer pressure to players to not disco
        WAIT_TIME: 10,
        salutation: 'Good Evening Mr.',
        instructionsPage: 'instructions_pp.html'
    }
}
```

When the **waiting room** dispatches a new **game room**, it will randomly select a treatment, build the full settings object, and send it to the browser, which stores it under `node.game.settings`

JavaScript console on the browser

```
» node.game.settings
← ▾ {…}
    COINS: 100
    EXCHANGE_RATE: 0.0005
    EXCHANGE_RATE_INSTRUCTIONS: 0.01
    MIN_PLAYERS: 2
    REPEAT: 1
  ▸ TIMER: Object { selectLanguage: 3000000, instructions: 90000, quiz: 60000, … }
    WAIT_TIME: 20
    description: "More time to wait and no peer pressure."
    instructionsPage: "instructions.html"
    name: "standard"
    salutation: "Hi there!"
    treatmentName: "standard"
  ▸ <prototype>: Object { … }
```

Treatment-Dependent Display

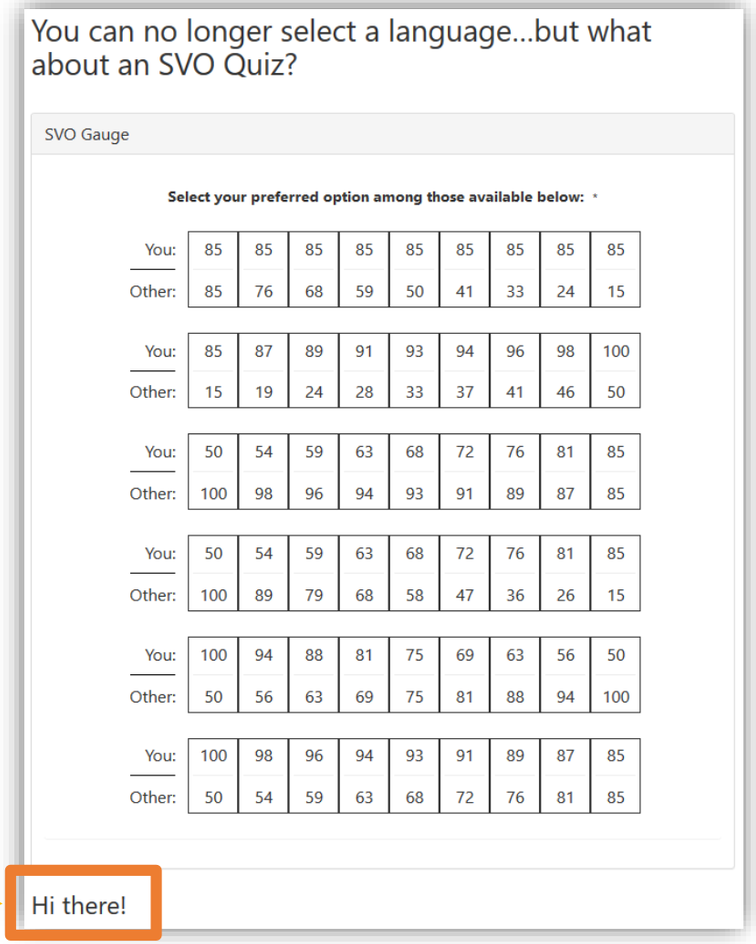
Here, we create a new DIV, we add the treatment dependent variable salutation from the settings object, and we append the DIV to the body of the frame.

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
         // Store a reference to the body.
         var body = W.getFrameDocument().body;
         node.widgets.append('SVOGauge', body);
         // Add a new element to the page.
         var div = document.createElementById('div');
         // Fill in treatment-dependent content.
         div.innerHTML = node.game.settings.salutation;
         // Append div to the body.
         body.appendChild(div);
     }
});
```

Treatment-Dependent Display

Here, we create a new DIV, we add the treatment dependent variable salutation from the settings object, and we append the DIV to the body of the frame.

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
          // Store a reference to the body.
          var body = W.getFrameDocument().body;
          node.widgets.append('SVOGauge', body);
          // Add a new element to the page.
          var div = document.createElementById('div');
          // Fill in treatment-dependent content.
          div.innerHTML = node.game.settings.salutation;
          // Append div to the body.
          body.appendChild(div);
```



The value of this text will be different for the two treatments. *Why is it at the bottom?* Because **appendChild** always append at the end of the element.

Creating a Page Structure

Here, we create two new HTML DIV elements, which will host our data.

```
<!DOCTYPE html>
<title>Select a Language</title>
<link rel="stylesheet" type="text/css" href="/lib/bootstrap/bootstrap.min.css"/>
<link rel="stylesheet" type="text/css" href="/stylesheets/nodegame.css"/>
<link rel="stylesheet" type="text/css" href="../css/style.css"/>
<body class="centered">
    <h1 class='margin-bottom'>You can no longer select a language…but what about an SVO Quiz?</h1>
    <div id="above"></div>
    <div id="below"></div>
</body>
```

We also give an id so that they can be easily fetched by JavaScript.
Note! The DIV elements are by default displayed as "blocks," that is one below the other. With a SPAN element the display might be different.

Treatment-Dependent Display

Here, we create a new DIV, we add the treatment dependent variable salutation from the settings object, and we append the DIV to the body of the frame.

```
stager.extendStep('selectLanguage', {
     frame: 'languageSelection.html',
     cb: function() {
         // Store a reference to the above and below elements.
         var above = W.getElementById('above');
         var below = W.gid('below'); // Shorthand for getElementById
         // Append the SVO widget below.
         node.widgets.append('SVOGauge', below);
         // Add a new element to the page.
         var div = document.createElementById('div');
         // Fill in treatment-dependent content.
         div.innerHTML = node.game.settings.salutation;
         // Append in the above element.
         above.appendChild(div);
     }
});
```

Treatment-Dependent Display

As a result, the salutation is inserted above the SVO widget.

```
<!DOCTYPE html>
<html> event
▶ <head> ··· </head>
▼ <body>
  ▶ <div id="ng_header" class="ng_header_position-horizontal-t"> ··· </div>
  ▼ <iframe id="ng_mainframe" class="ng_mainframe-header-horizontal-t" name="ng_mainframe"
    scrolling="no" style="padding-top: 69px; min-height: 1208px;" frameborder="0">
    ▼ #document
        <!DOCTYPE html>
      ▼ <html> event
        ▶ <head> ··· </head>
        ▼ <body class="centered">
          ▶ <h1 class="margin-bottom"> ··· </h1>
          ▼ <div id="above">
              <div>Hi there!</div>
            </div>
          ▼ <div id="below">
            ▶ <div class="ng_widget panel panel-default svogauge"> ··· </div>
            </div>
            ::after
          </body>
        </html>
    </iframe>
  ▶ <div id="ng_waitScreen" style="display: none;"> ··· </div>
    ::after
  </body>
</html>
```