Design and Implementation of Online Experiments

nodeGame.org

Stefano Balietti

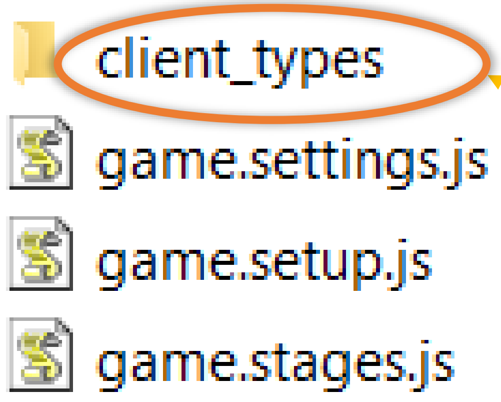
MZES and Heidelberg

nodeGame

Intermediate

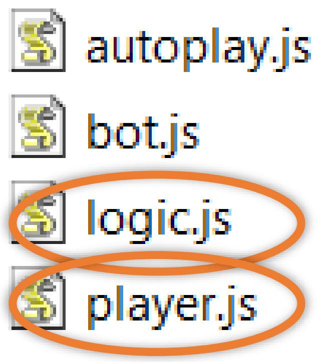
@balietti
@nodegameorg
stefanobalietti.com@gmail.com

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
player.js

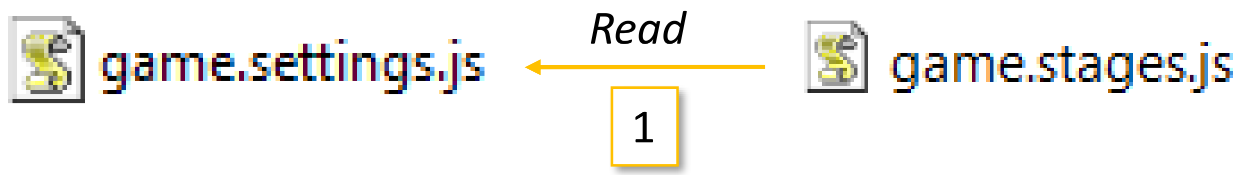
Computer (orchestrator), server

Human, web browser

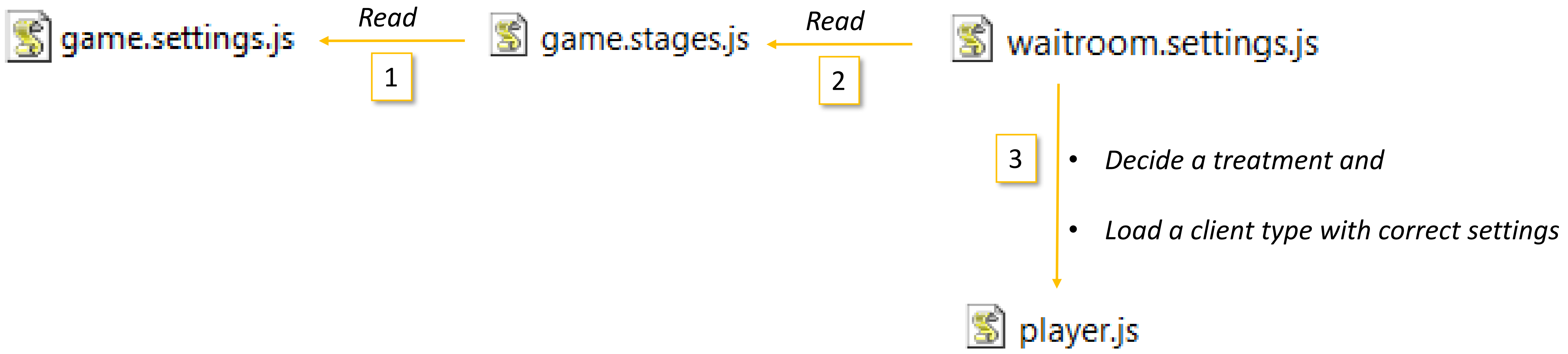
Information Flow for Client Type

game.settings.js

Information Flow for Client Type

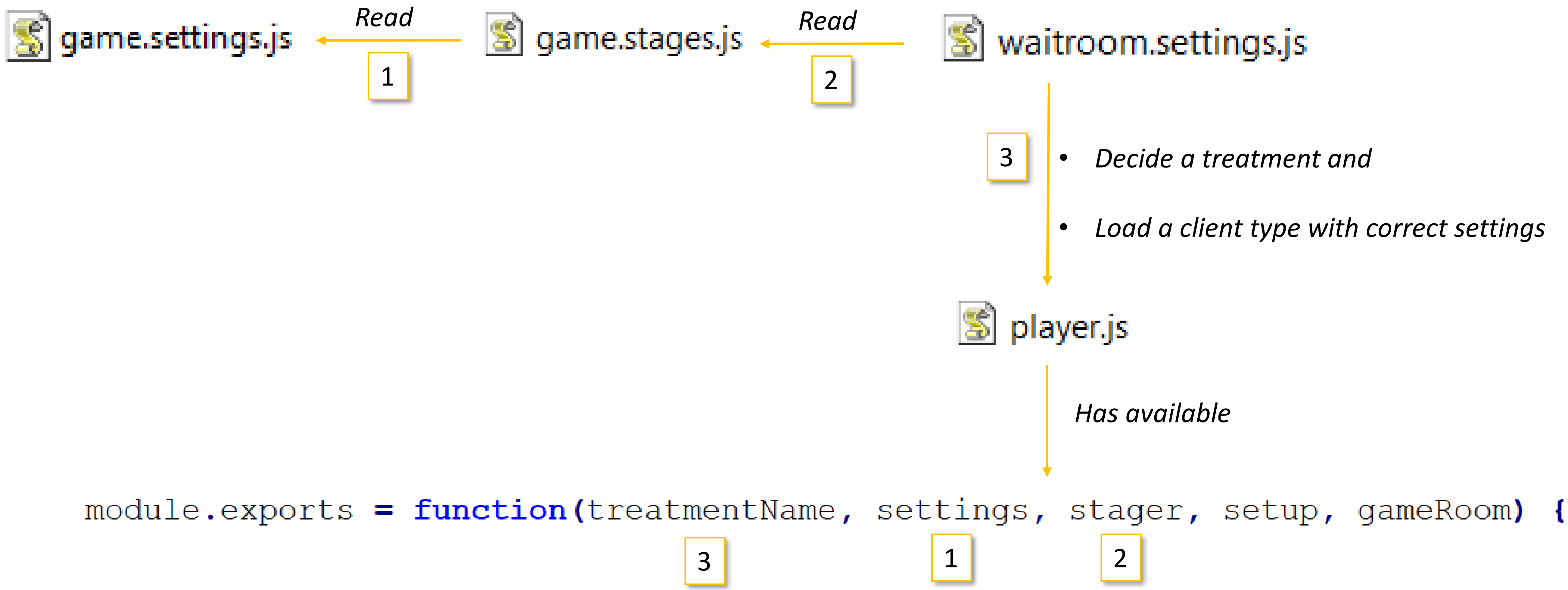


Information Flow for Client Type



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

Information Flow for Client Type



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

How To Implement a Sequence

Client-Types-v5

- 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]

- 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 be found** in the game 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"?*

game.stages.js

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

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 comma, 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

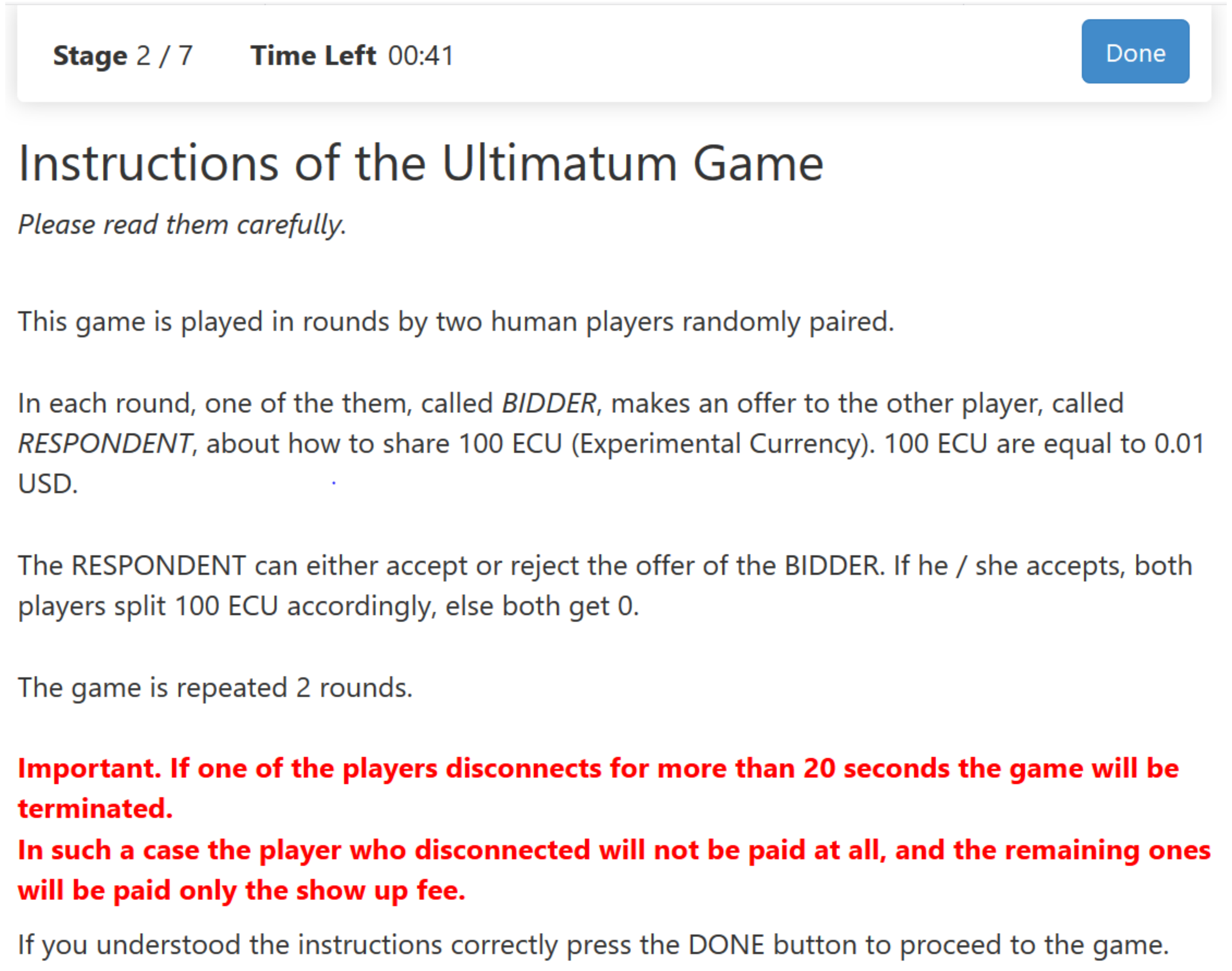
Here, we add a nodeGame Widget to select the language (as the step id suggests).

```
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



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.

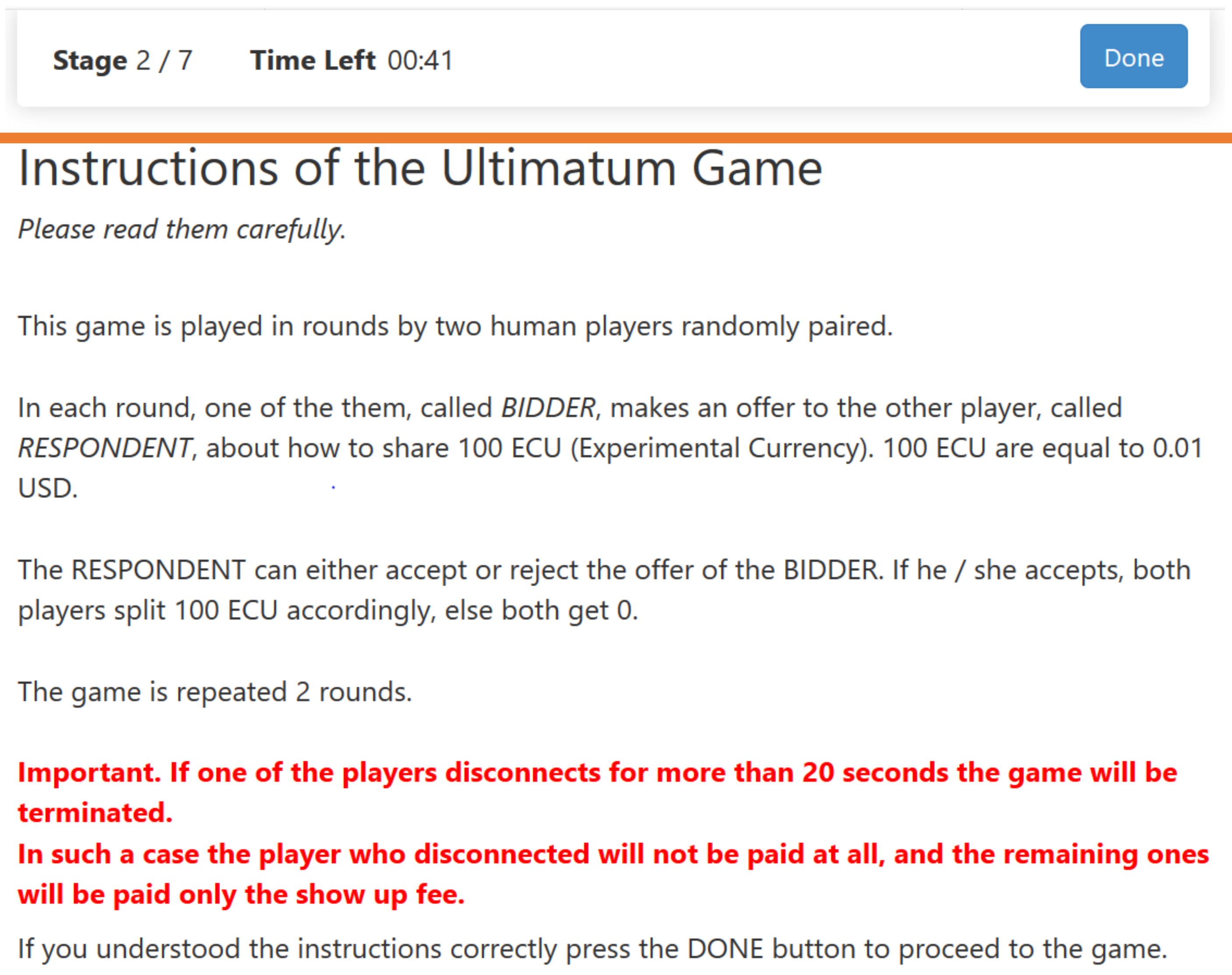
Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining ones will be paid only the show up fee.

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.

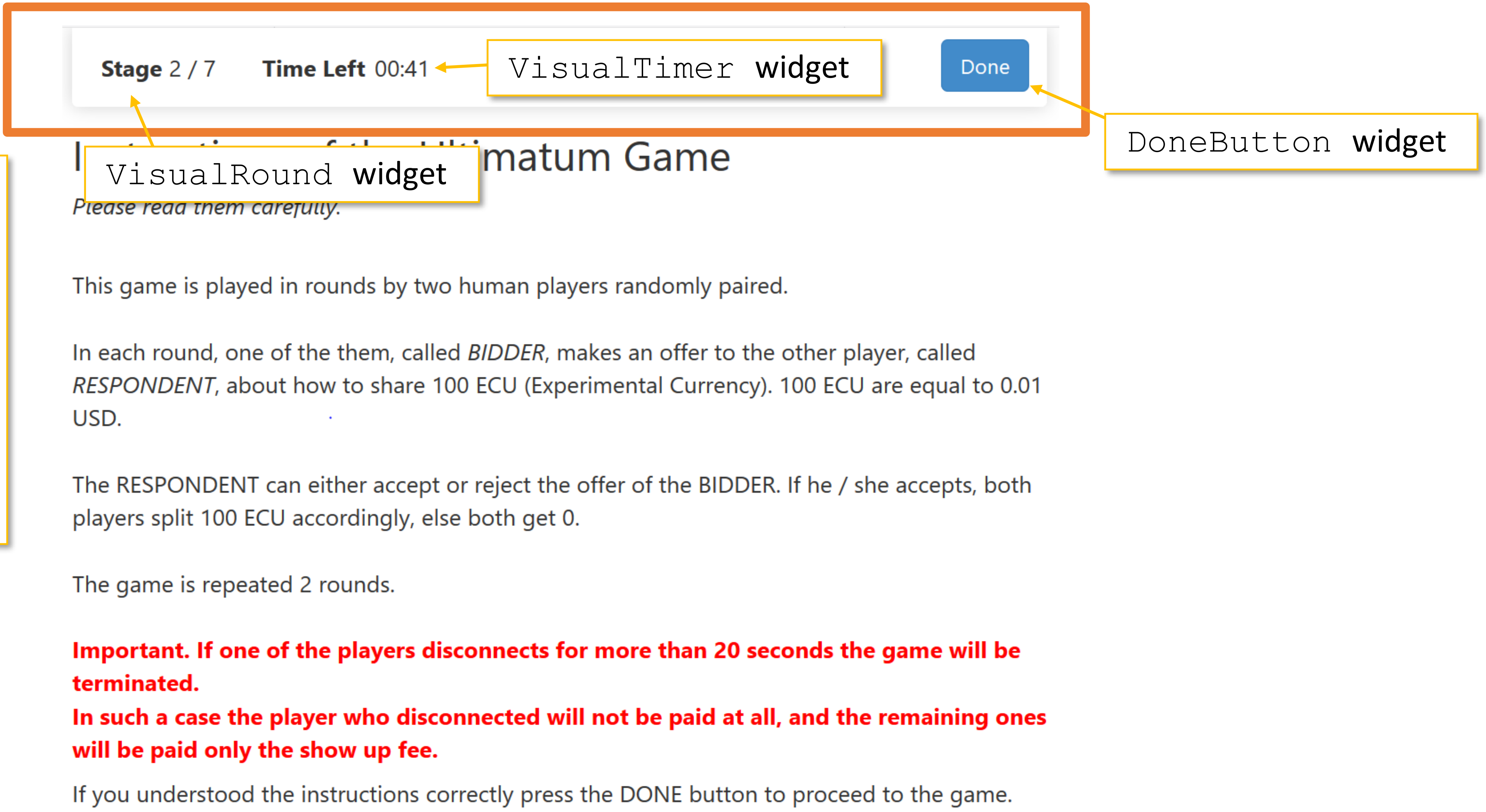
Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining ones will be paid only the show up fee.

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



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.

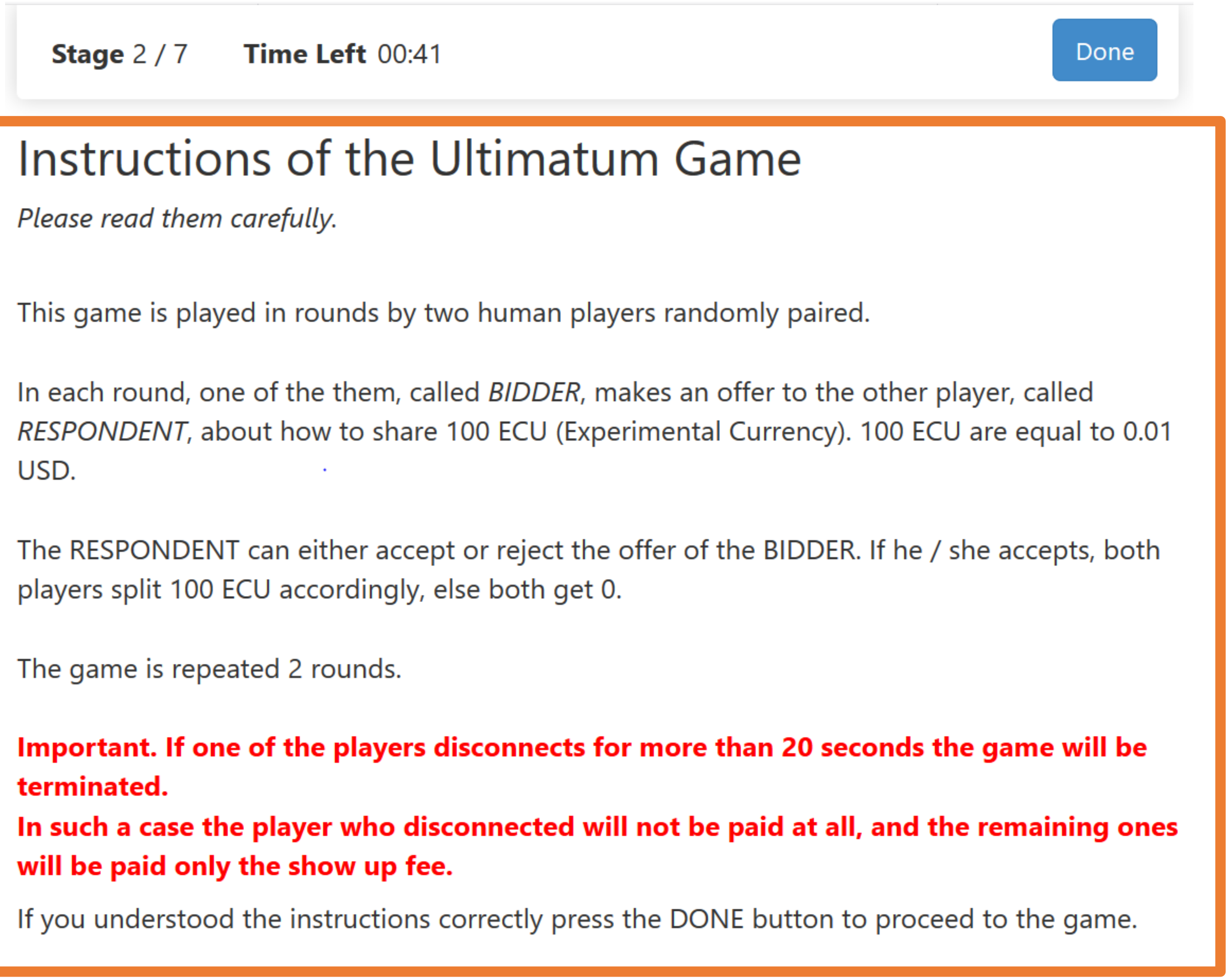
The RESPONDENT can either accept or reject the offer of the BIDDER. If he / she accepts, both players split 100 ECU accordingly, else both get 0.

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


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.

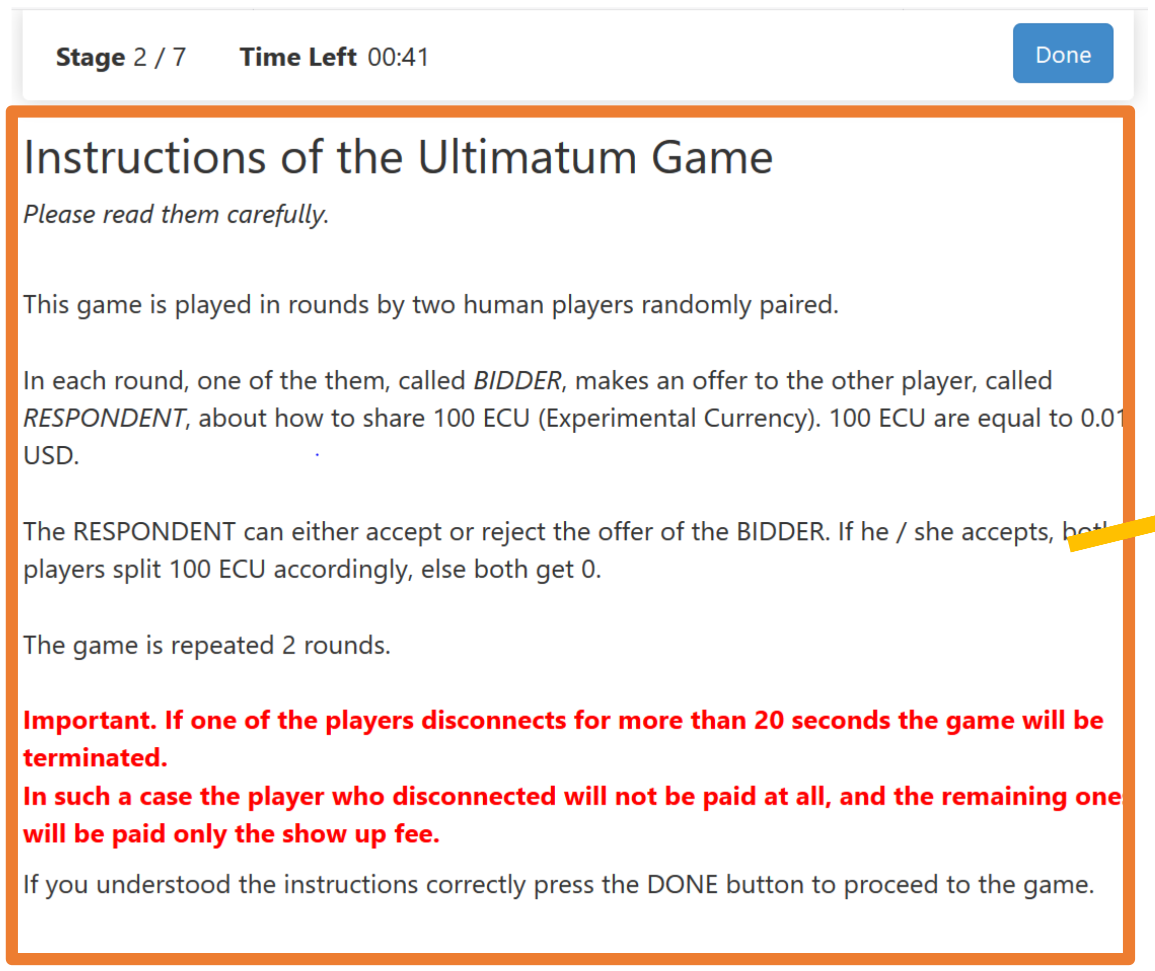
Important. If one of the players disconnects for more than 20 seconds the game will be terminated.
In such a case the player who disconnected will not be paid at all, and the remaining ones will be paid only the show up fee.

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

Frame

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

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.

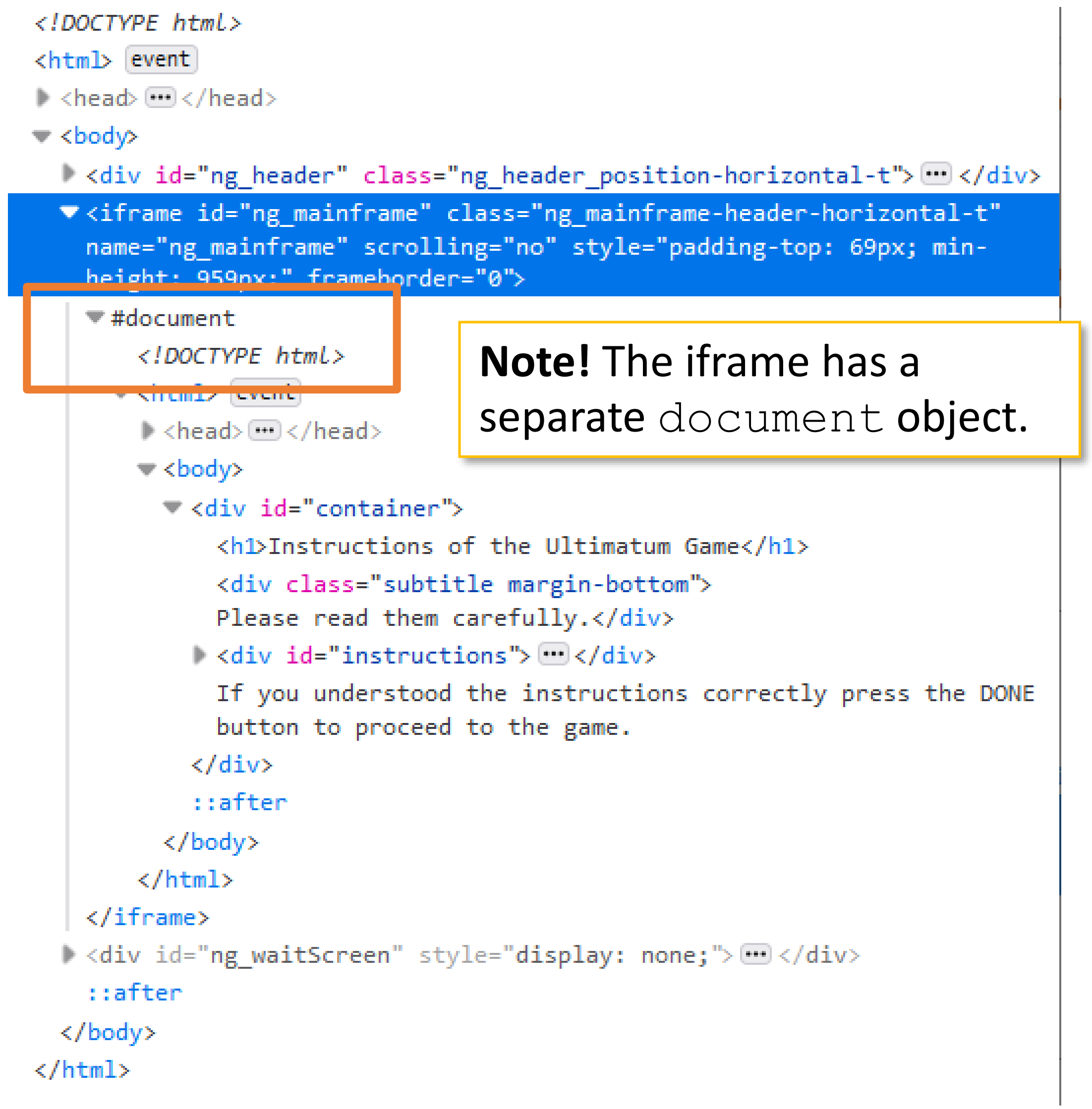
In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

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 understand the instructions correctly press the DONE button to proceed to the game.

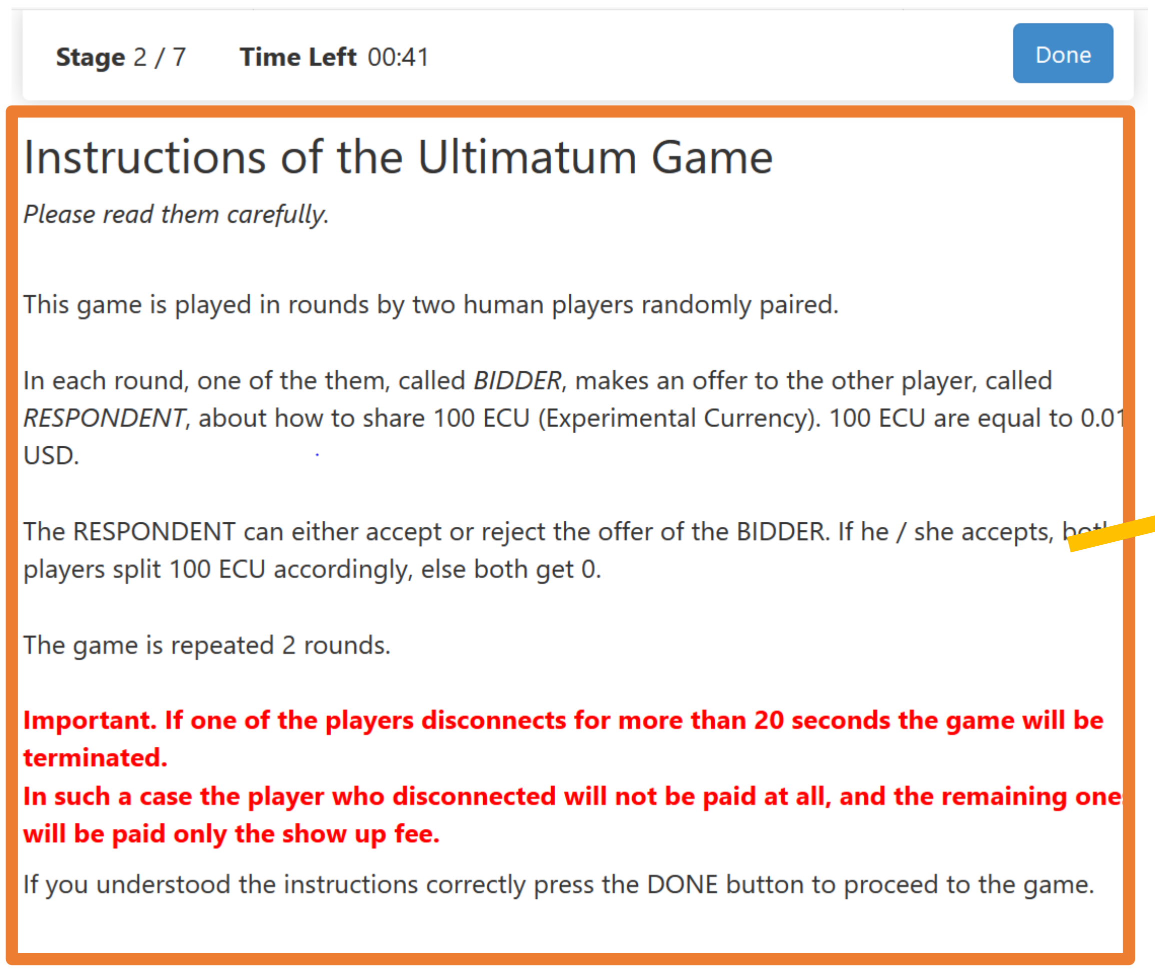


```
<!DOCTYPE html>
<html> event
  <head> … </head>
  <body>
    <div id="ng_header" class="ng_header_position-horizontal-t"> … </div>
    <iframe id="ng_mainframe" class="ng_mainframe-header-horizontal-t"
    name="ng_mainframe" scrolling="no" style="padding-top: 69px; min-
    height: 959px;" frameborder="0">
      #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.

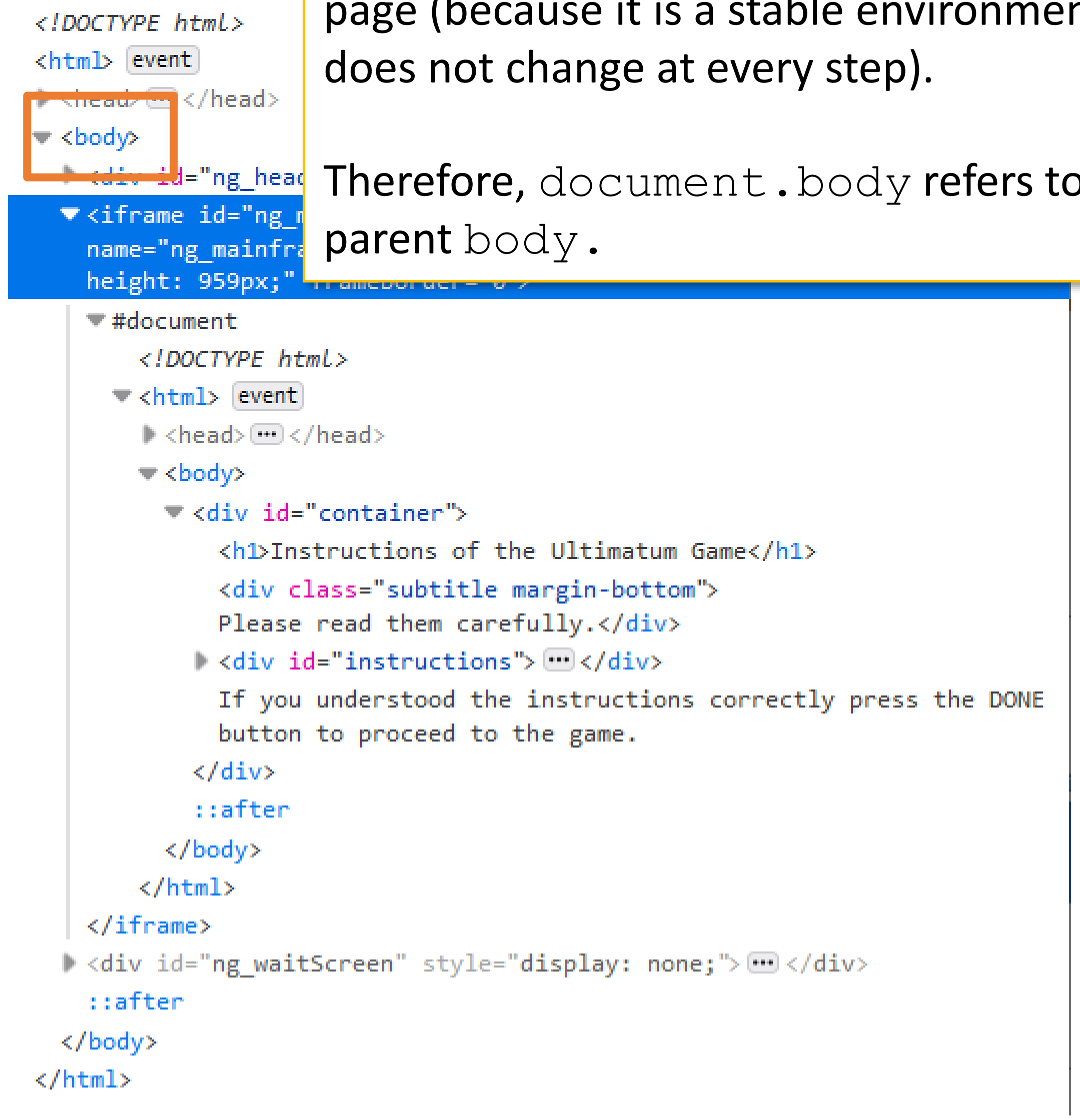
In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

The RESPONDENT can either accept or reject the offer of the BIDDER. If he / she accepts, bo 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.



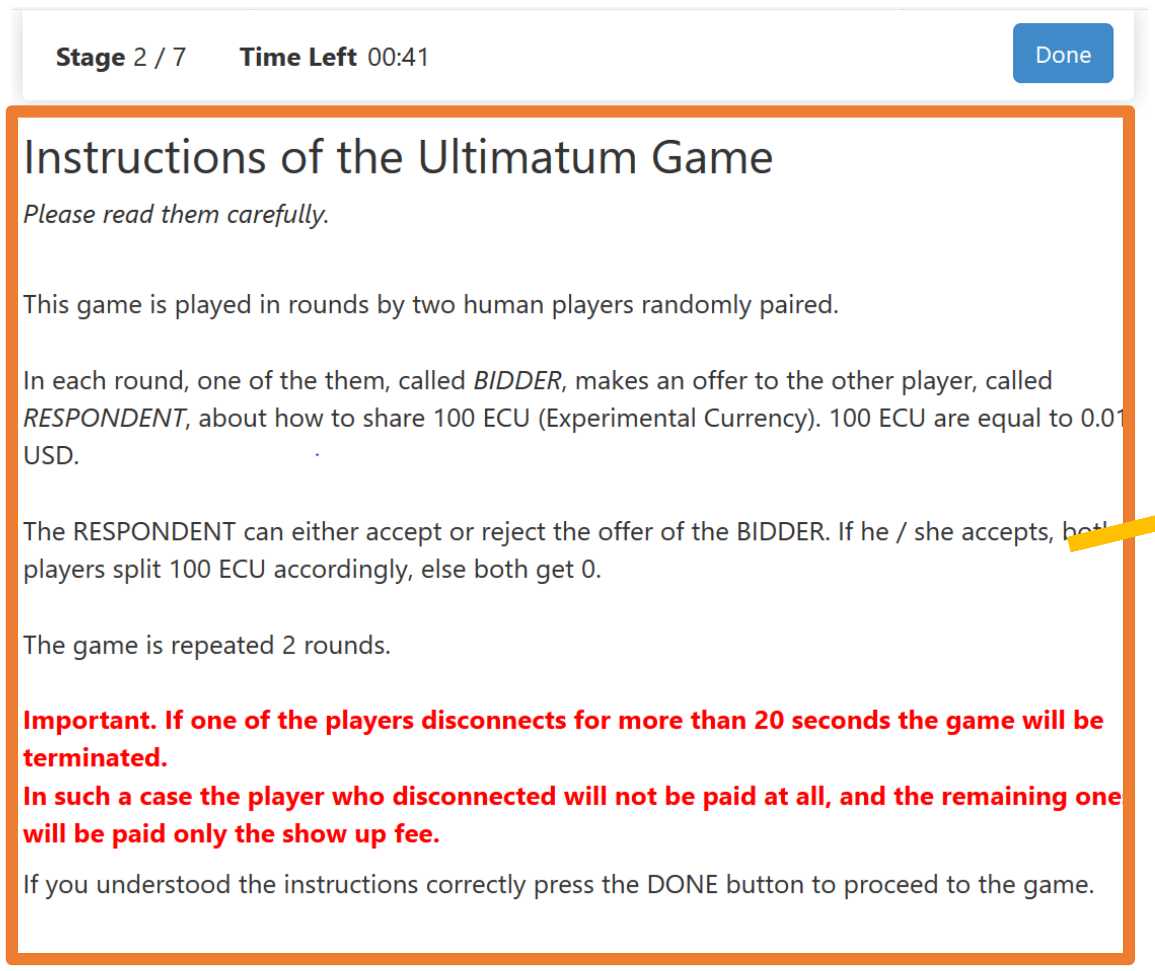
```
<!DOCTYPE html>
<html> event
  <head> … </head>
  <body>
    <div id="ng_head
    <iframe id="ng_m
    name="ng_mainfra
    height: 959px;"
      #document
        <!DOCTYPE html>
        <html> event
          <head> … </head>
          <body>
            <div id="container">
              <h1>Instructions 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.

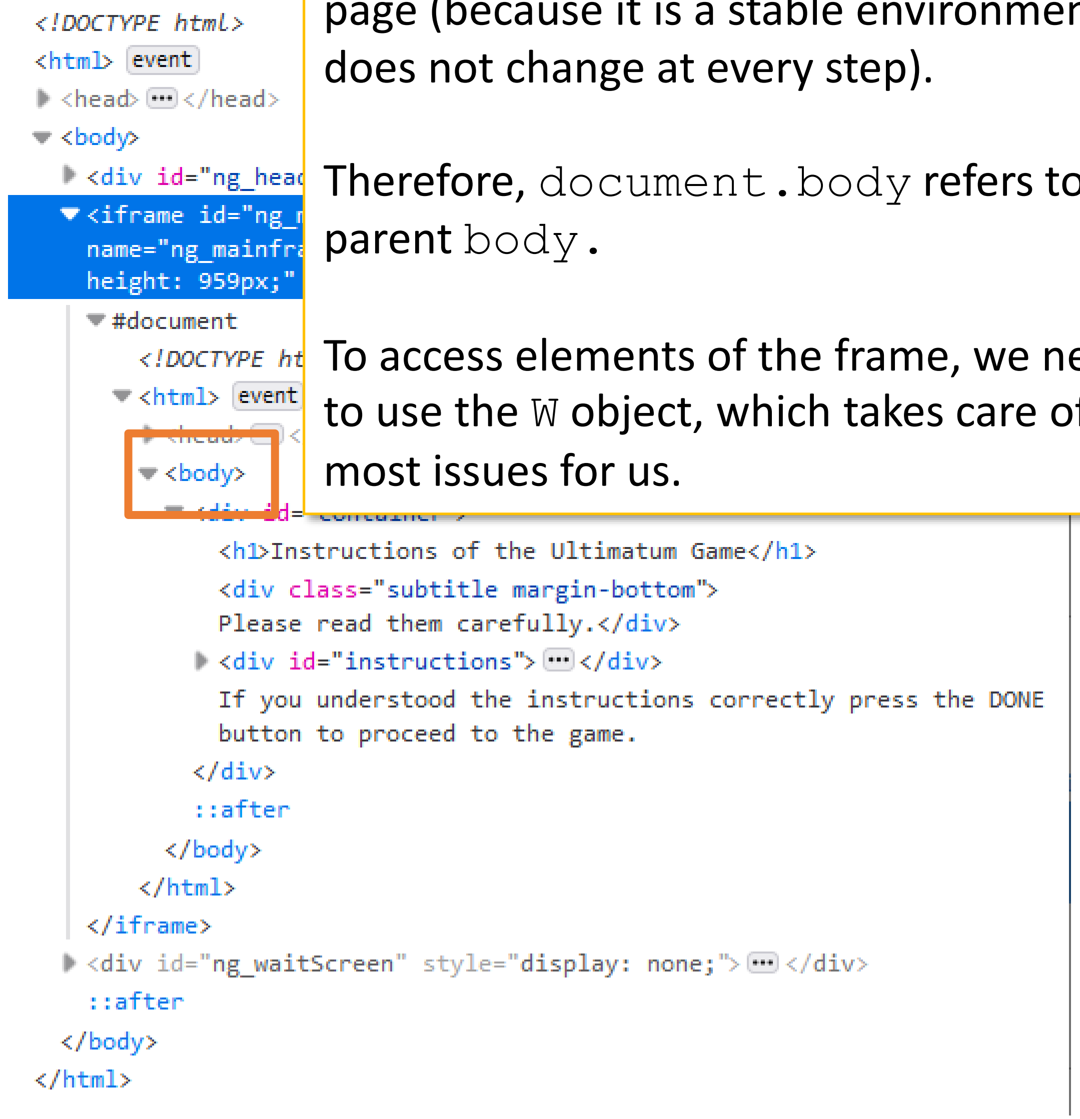
In each round, one of the them, called *BIDDER*, makes an offer to the other player, called *RESPONDENT*, about how to share 100 ECU (Experimental Currency). 100 ECU are equal to 0.01 USD.

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_heac
  <iframe id="ng_r
  name="ng_mainfra
  height: 959px;"
   #document
    <!DOCTYPE ht
    <html> event
     <head> ⋯ <
     <body>
      <div id=
        <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).

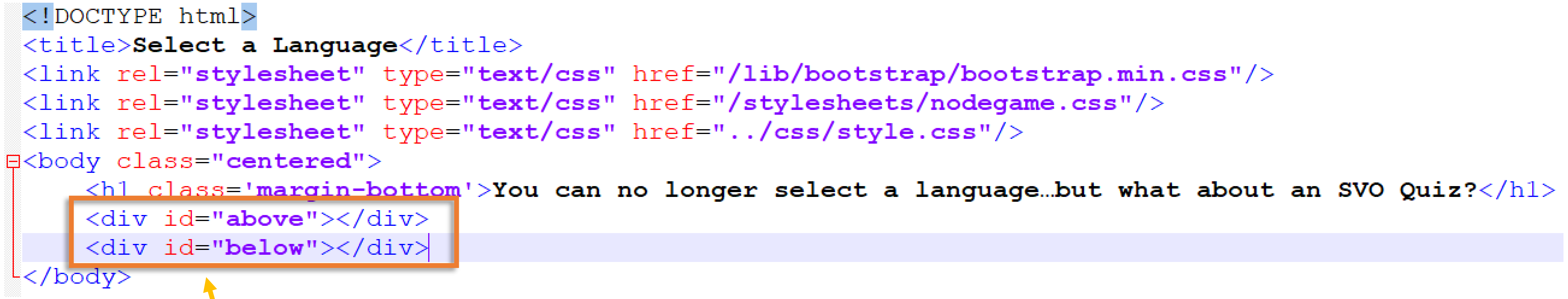
Therefore, `document.body` refers to the parent `body`.

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.

Creating a Page Structure

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.



```
<!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 add two div elements, and we give them an id so that they cab be easily be fetched by JavaScript.
Note! The DIV elements are by default displayed as "blocks," that is one below the other. With a SPAN element it might be different.

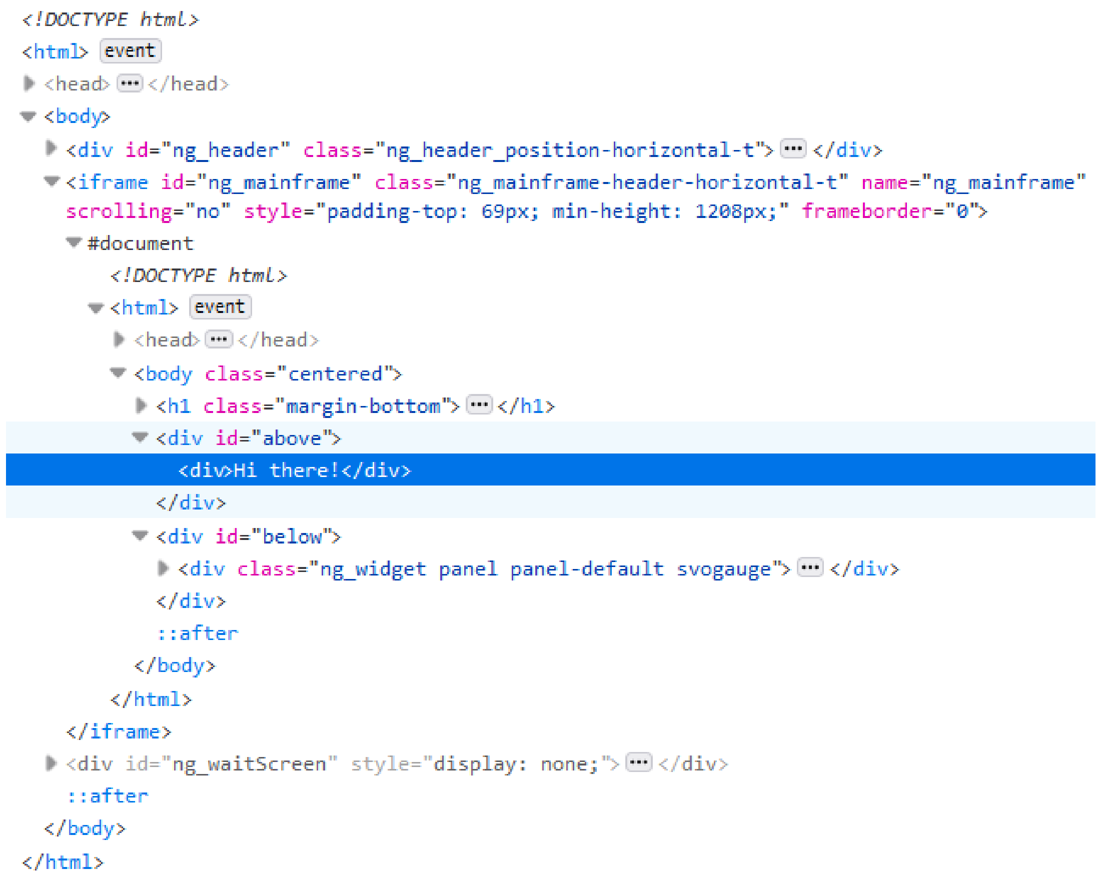
Treatment-Dependent Display

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.

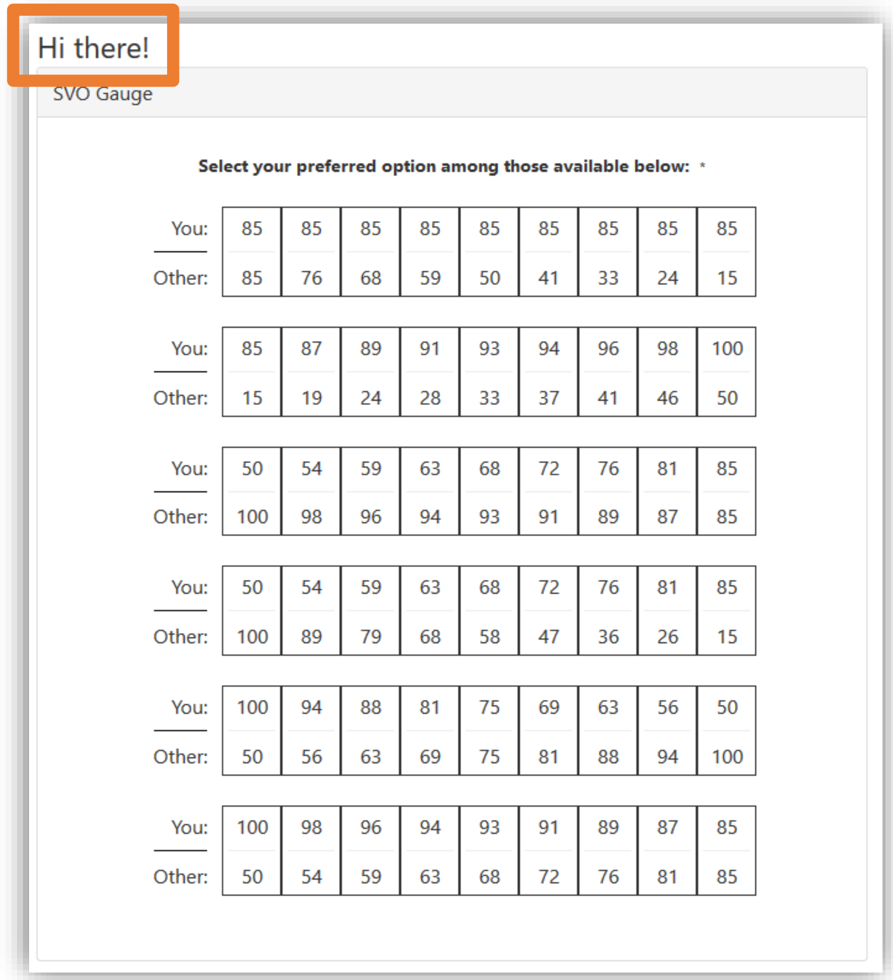
```
stager.extendStep('selectLanguage', {
      frame: 'languageSelection.html',
      cb: function() {
           // Store a reference to the above and below elements.
           var above = W.getElementById('above');
           var below = W.gid('below'); // Shorthand for getElementById
           // 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>
```



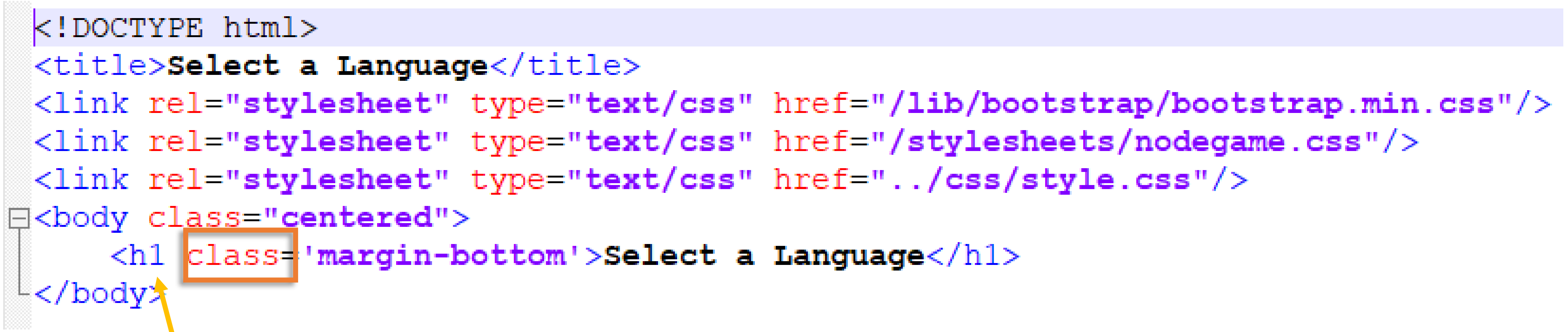
Hi there!

SVO Gauge

Select your preferred option among those available below: *

You:	85	85	85	85	85	85	85	85	85
Other:	85	76	68	59	50	41	33	24	15
You:	85	87	89	91	93	94	96	98	100
Other:	15	19	24	28	33	37	41	46	50
You:	50	54	59	63	68	72	76	81	85
Other:	100	98	96	94	93	91	89	87	85
You:	50	54	59	63	68	72	76	81	85
Other:	100	89	79	68	58	47	36	26	15
You:	100	94	88	81	75	69	63	56	50
Other:	50	56	63	69	75	81	88	94	100
You:	100	98	96	94	93	91	89	87	85
Other:	50	54	59	63	68	72	76	81	85

External CSS Files

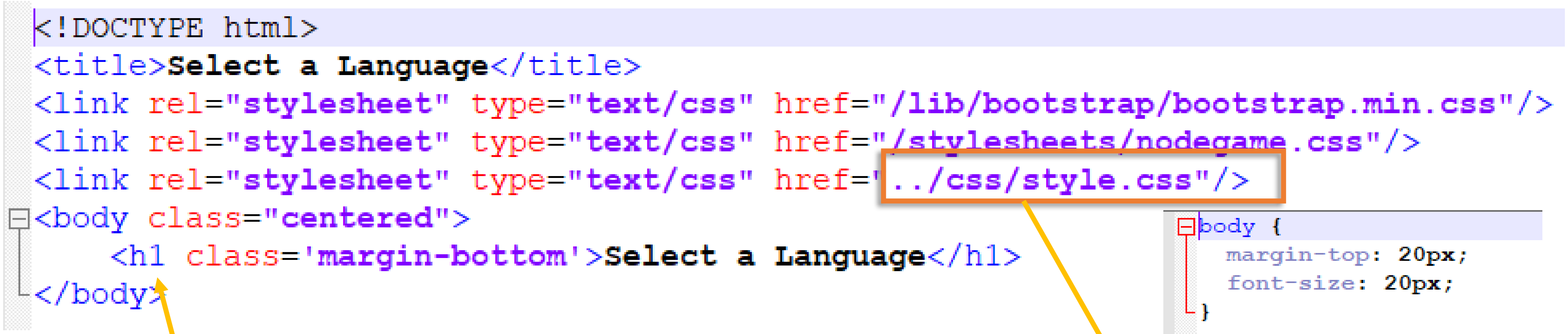


```
<!DOCTYPE html>
<title>Select a Language</title>
<link rel="stylesheet" type="text/css" href="/lib/bootstrap/bootstrap.min.css"/>
<link rel="stylesheet" type="text/css" href="/stylesheets/nodegame.css"/>
<link rel="stylesheet" type="text/css" href="../css/style.css"/>
<body class="centered">
   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

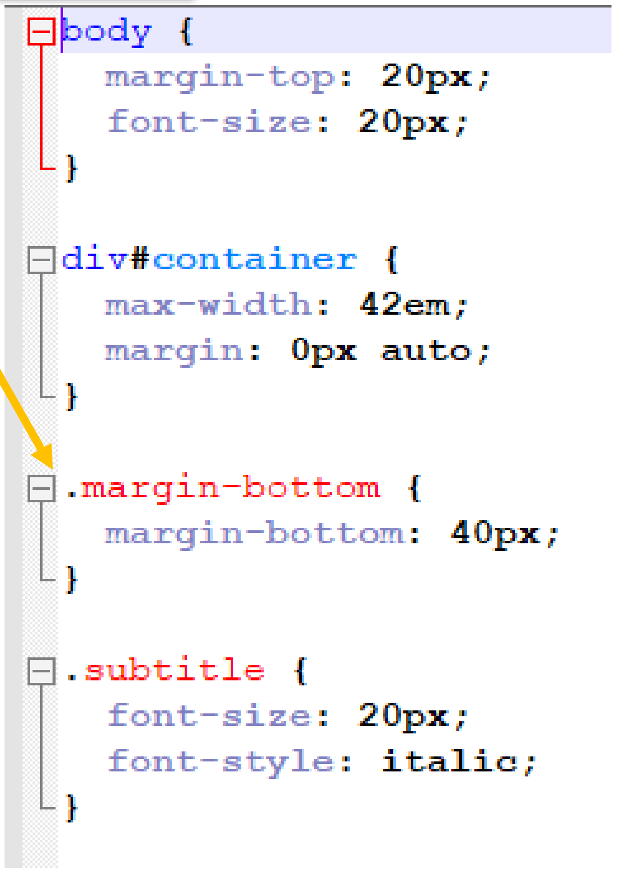
The `h1` tag is a display tag for "headings," i.e., titles. There are different heading size from 1 the biggest, to 6 the smallest.

Our `h1` tag as a class attribute equals to `margin-bottom`. What does it mean? In one of the stylesheets above there is a class named `margin-bottom` with some rules defined. **Can you find it?**

External CSS Files



```
<!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>
```



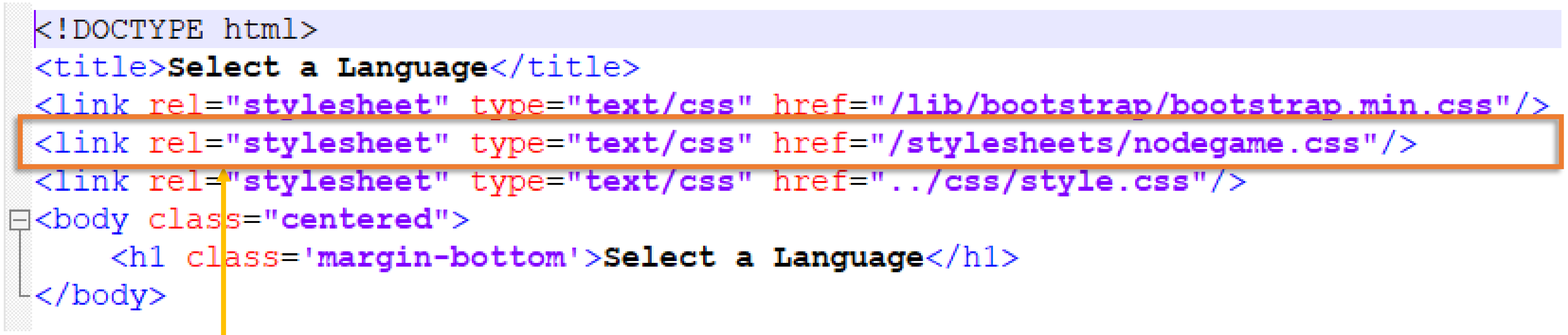
```
body {
  margin-top: 20px;
  font-size: 20px;
}

div#container {
  max-width: 42em;
  margin: 0px auto;
}

.margin-bottom {
  margin-bottom: 40px;
}

.subtitle {
  font-size: 20px;
  font-style: italic;
}
```

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.

Our `h1` tag as a class attribute equals to `margin-bottom`. What does it mean? In one of the stylesheets above there is a class named `margin-bottom` with some rules defined. **Can you find it?**

An easy entry-point to CSS rules can be found here:
https://www.w3schools.com/Css/

External CSS Files

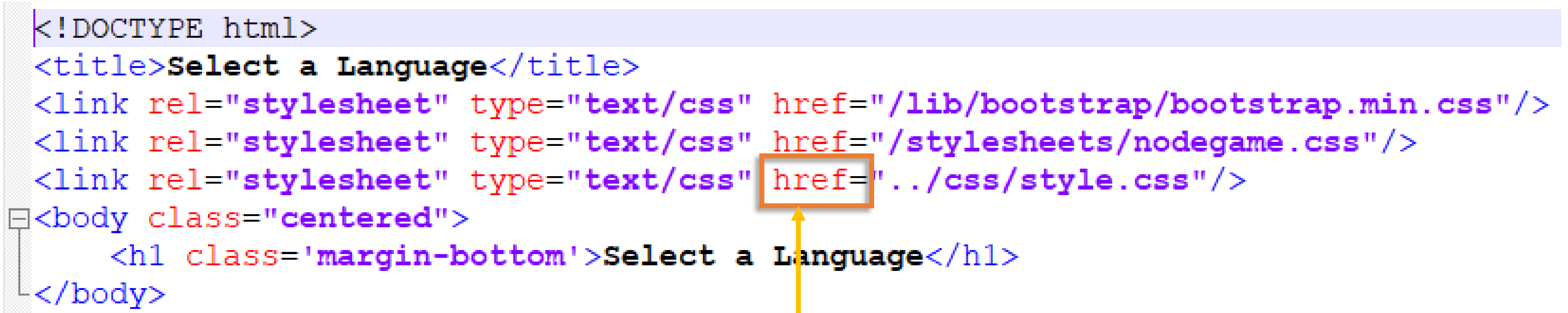


```
<!DOCTYPE html>
<title>Select a Language</title>
<link rel="stylesheet" type="text/css" href="/lib/bootstrap/bootstrap.min.css"/>
<link rel="stylesheet" type="text/css" href="/stylesheets/nodegame.css"/>
<link rel="stylesheet" type="text/css" href="../css/style.css"/>
<body class="centered">
   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

<link> tags import CSS rules to style the display of page elements
Notice! Link tags are self-closing, that is: there is no </link> at the end

What other HTML tags are self closing?
Hint: A self-closing does not need to contain something else.

External CSS Files

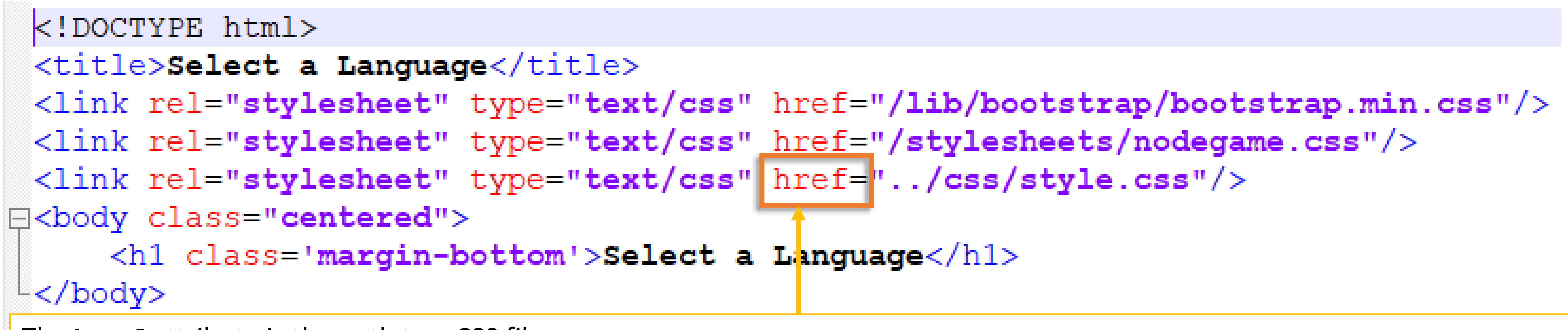


```
<!DOCTYPE html>
<title>Select a Language</title>
<link rel="stylesheet" type="text/css" href="/lib/bootstrap/bootstrap.min.css"/>
<link rel="stylesheet" type="text/css" href="/stylesheets/nodegame.css"/>
<link rel="stylesheet" type="text/css" href="../css/style.css"/>
<body class="centered">
   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

The `href` attribute is the path to a CSS file.

Notice! The path for the first two files begins with `/` meaning that these files are to be found at the *very top* of the directory structure, that is the nodeGame server. To view it, open your browser at the address: `http://localhost:8080/stylesheets/nodegame.css`

External CSS Files



```
<!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>
```

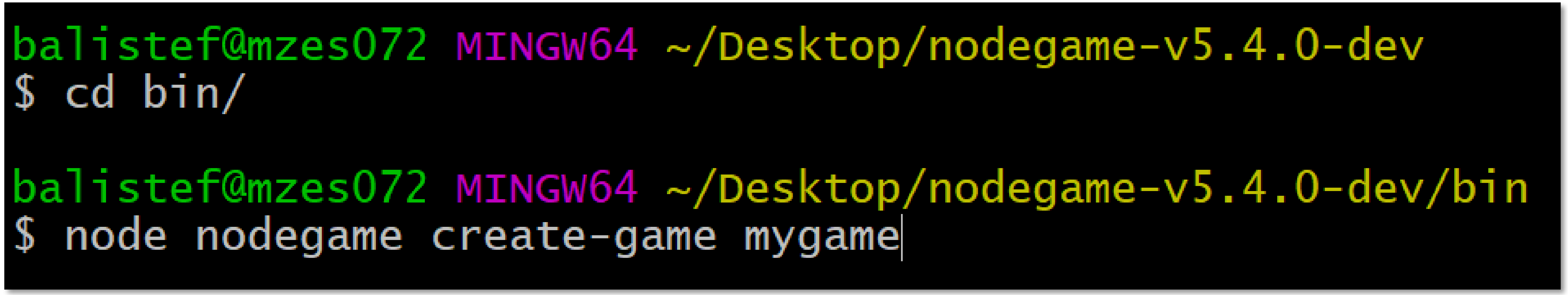
The `href` attribute is the path to a CSS file.

Notice! The path for the first two files begins with `/` meaning that these files are to be found at the *very top* of the directory structure, that is the nodeGame server. To view it, open your browser at the address:
`http://localhost:8080/stylesheets/nodegame.css`

The last CSS file does not start with `/` meaning that it is a file *local to your game*. Where it is? Do you remember the commands to navigate the file system in the terminal? `cd ..` means to go one directory above the current one, and here `..` Means to go one directory above the current one. We are in `public/en/`, so the file can be found in `public/css/style.css`
You can also view it with your browser at the address:
`http://localhost:8080/ultimatum/css/style.css`

Create a New Game

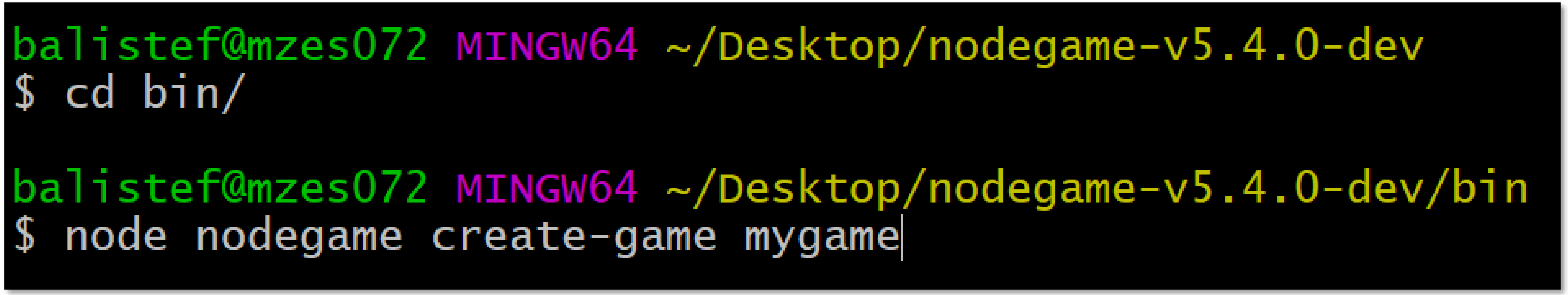
1



```
balistef@mzes072 MINGW64 ~/Desktop/nodegame-v5.4.0-dev
$ cd bin/

balistef@mzes072 MINGW64 ~/Desktop/nodegame-v5.4.0-dev/bin
$ node nodegame create-game mygame
```

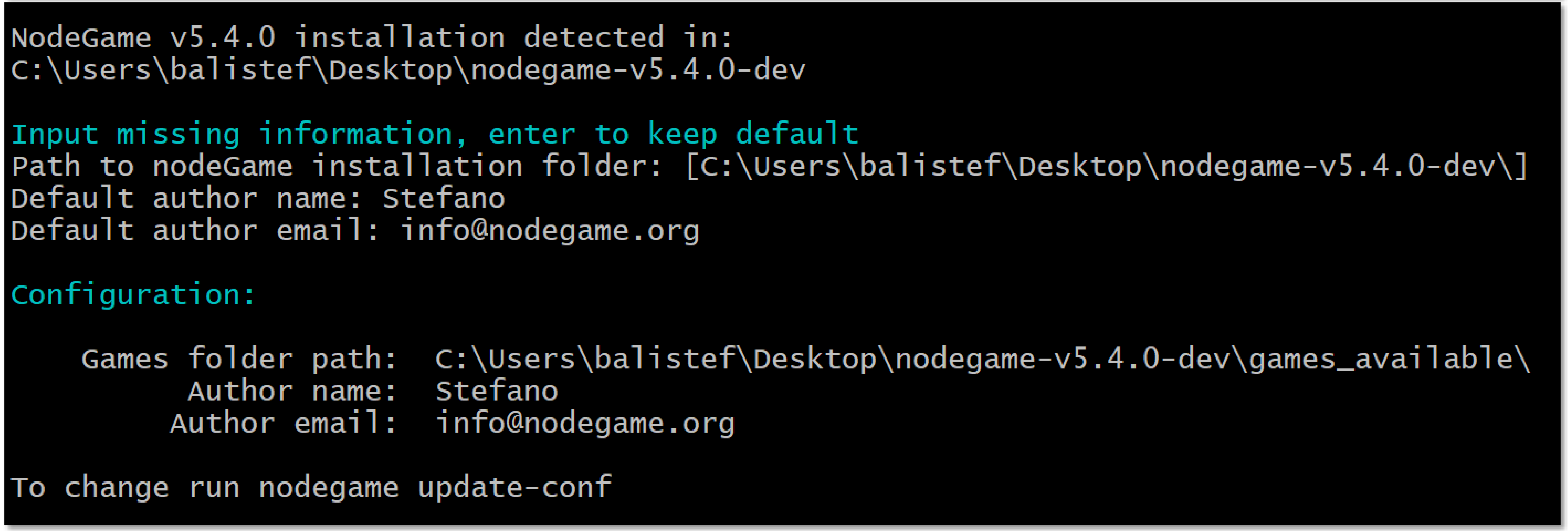
Create a New Game

1



```
balistef@mzes072 MINGW64 ~/Desktop/nodegame-v5.4.0-dev
$ cd bin/

balistef@mzes072 MINGW64 ~/Desktop/nodegame-v5.4.0-dev/bin
$ node nodegame create-game mygame
```

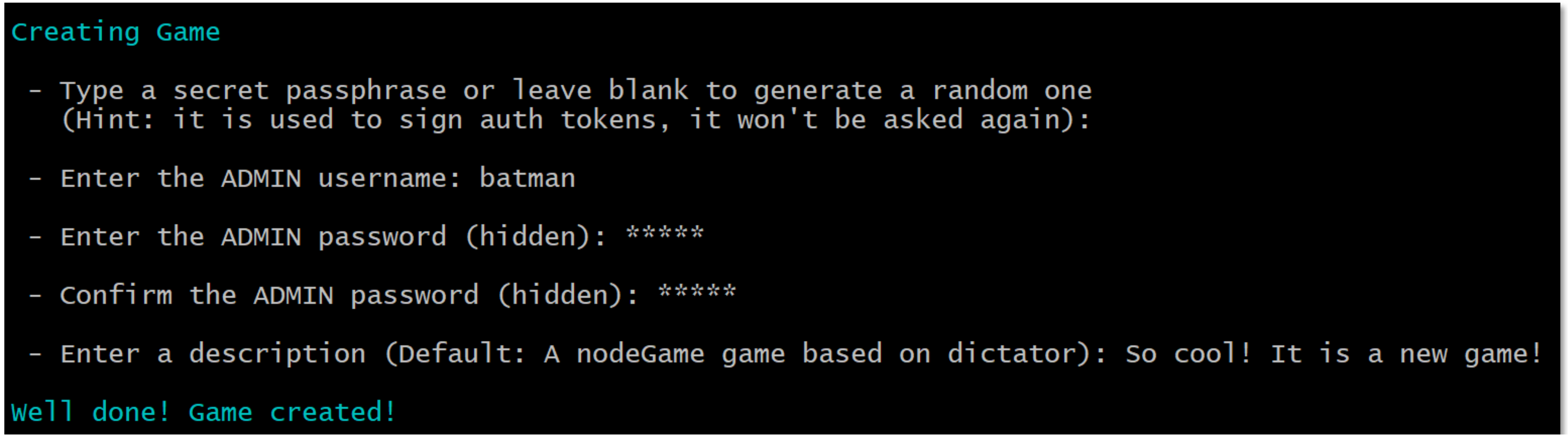
2



```
NodeGame v5.4.0 installation detected in:
C:\Users\balistef\Desktop\nodegame-v5.4.0-dev

Input missing information, enter to keep default
Path to nodeGame installation folder: [C:\Users\balistef\Desktop\nodegame-v5.4.0-dev\]
Default author name: Stefano
Default author email: info@nodegame.org

Configuration:

    Games folder path:  C:\Users\balistef\Desktop\nodegame-v5.4.0-dev\games_available\
          Author name:  Stefano
         Author email:  info@nodegame.org

To change run nodegame update-conf
```

Create a New Game

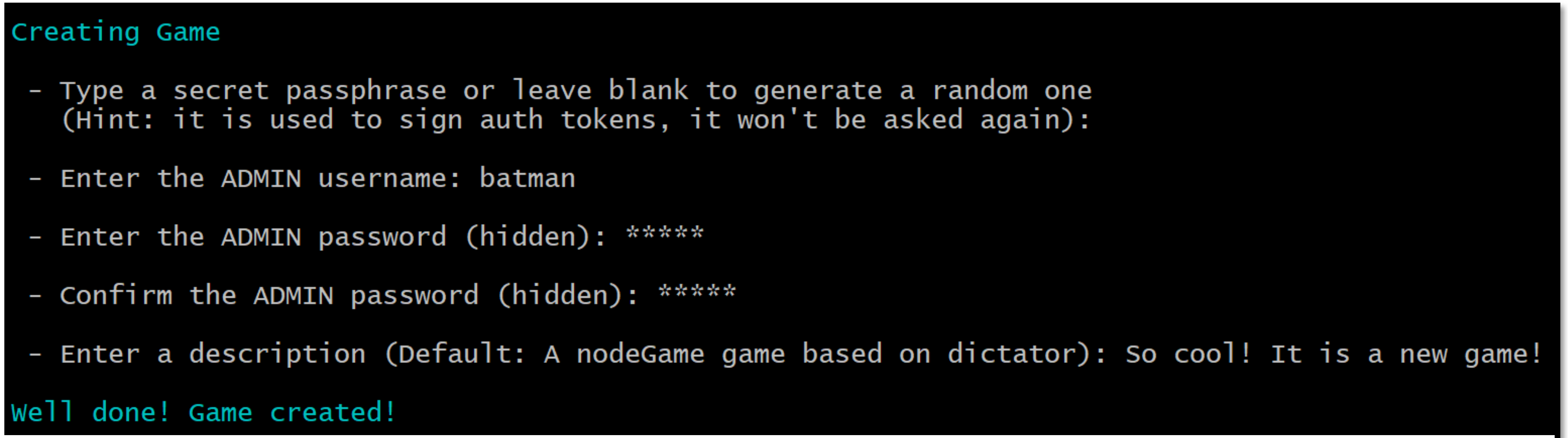
3



```
Creating Game

 - Type a secret passphrase or leave blank to generate a random one
   (Hint: it is used to sign auth tokens, it won't be asked again):

 - Enter the ADMIN username: batman

 - Enter the ADMIN password (hidden): *****

 - Confirm the ADMIN password (hidden): *****

 - Enter a description (Default: A nodeGame game based on dictator): So cool! It is a new game!

Well done! Game created!
```

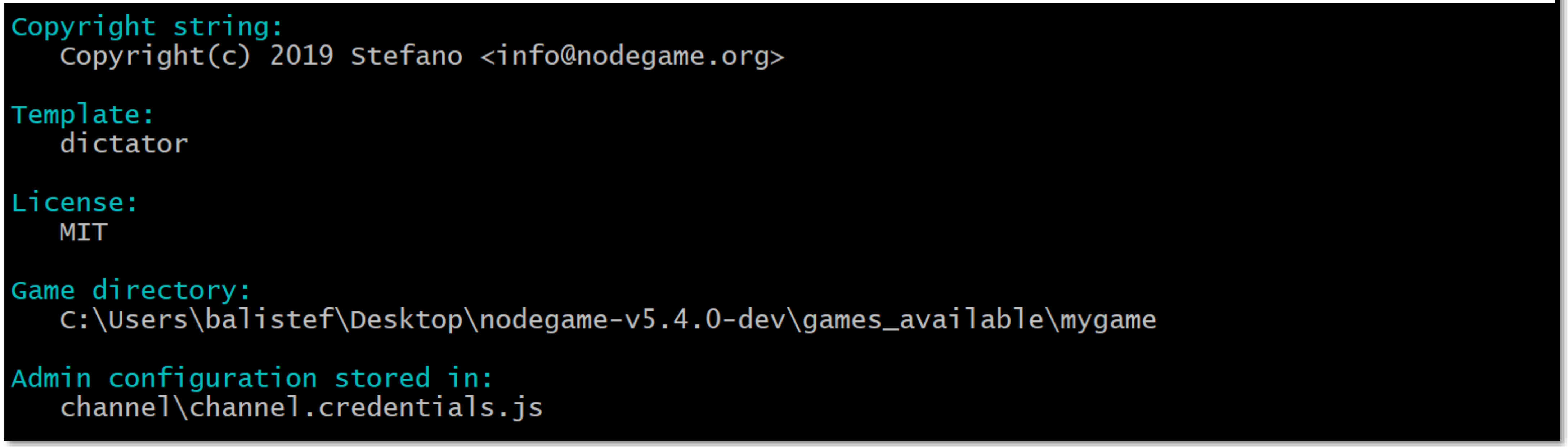
Create a New Game

3



```
Creating Game

 - Type a secret passphrase or leave blank to generate a random one
   (Hint: it is used to sign auth tokens, it won't be asked again):

 - Enter the ADMIN username: batman

 - Enter the ADMIN password (hidden): *****

 - Confirm the ADMIN password (hidden): *****

 - Enter a description (Default: A nodeGame game based on dictator): So cool! It is a new game!

Well done! Game created!
```

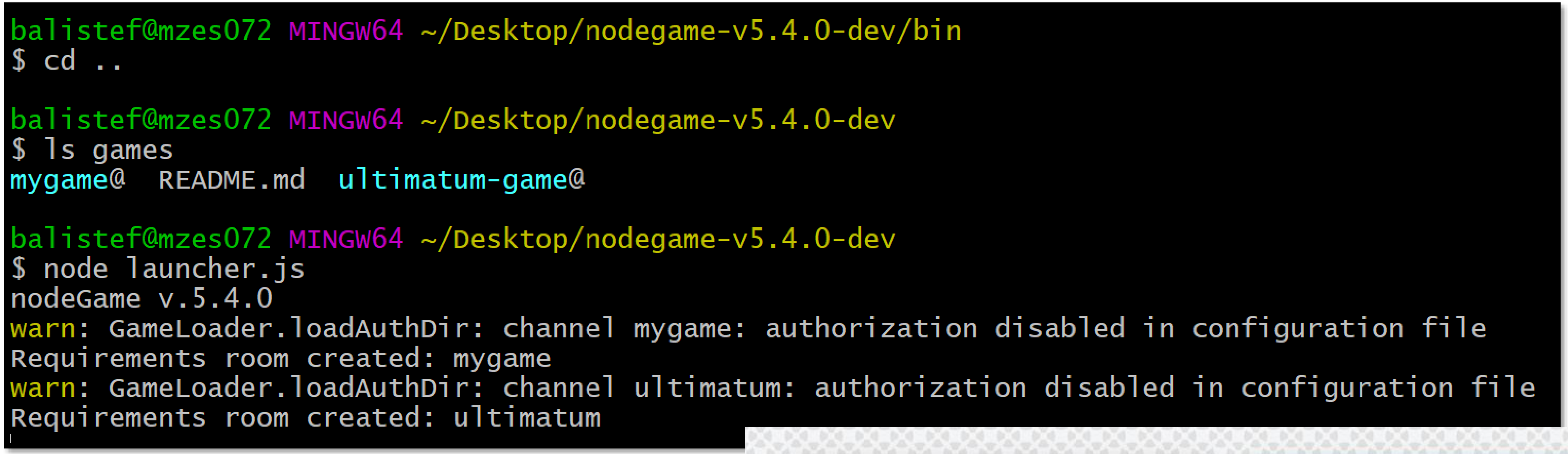
4



```
Copyright string:
   Copyright(c) 2019 Stefano <info@nodegame.org>

Template:
   dictator

License:
   MIT

Game directory:
   C:\Users\balistef\Desktop\nodegame-v5.4.0-dev\games_available\mygame

Admin configuration stored in:
   channel\channel.credentials.js
```

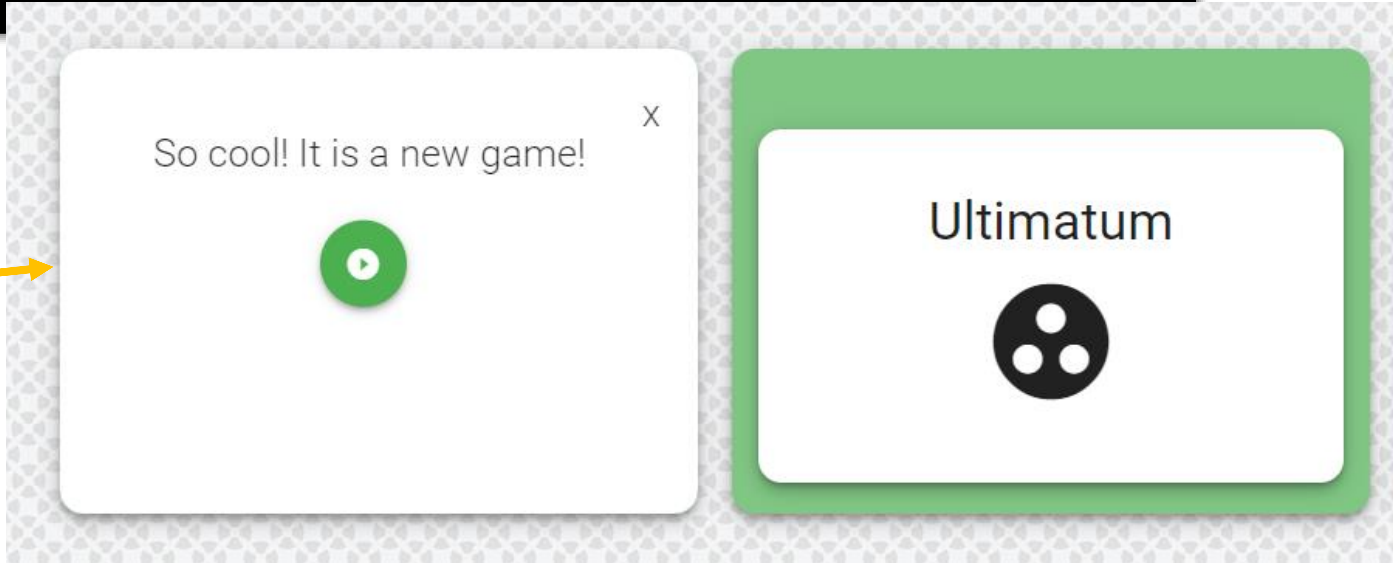
Create a New Game

5



```
balistef@mzes072 MINGW64 ~/Desktop/nodegame-v5.4.0-dev/bin
$ cd ..

balistef@mzes072 MINGW64 ~/Desktop/nodegame-v5.4.0-dev
$ ls games
mygame@  README.md  ultimatum-game@

balistef@mzes072 MINGW64 ~/Desktop/nodegame-v5.4.0-dev
$ node launcher.js
nodeGame v.5.4.0
warn: GameLoader.loadAuthDir: channel mygame: authorization disabled in configuration file
Requirements room created: mygame
warn: GameLoader.loadAuthDir: channel ultimatum: authorization disabled in configuration file
Requirements room created: ultimatum
```

http://localhost:8080
Our new game is there!



The Default Template of a New Game

Dictator Game

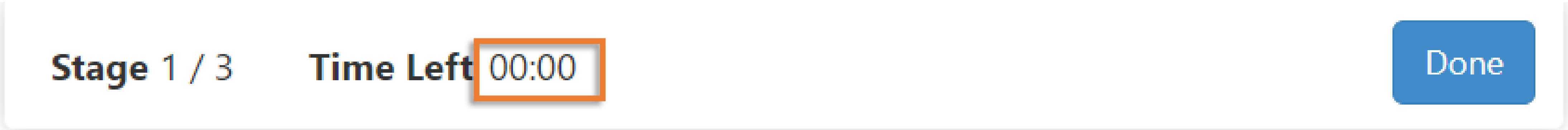
- The default template is a basic **Dictator Game**
- The Dictator game is like an Ultimatum, but even simpler: after an *offer* is made, the other player cannot reply, he or she must merely *observe.*
- It is appropriate to study *fairness* and *altruism*
- A rational player should always offer 0, however non-zero offers are common in experiments
- Framing makes a difference: *taking from* vs *giving to* others

The Default Template of a New Game

How can we improve this basic game?

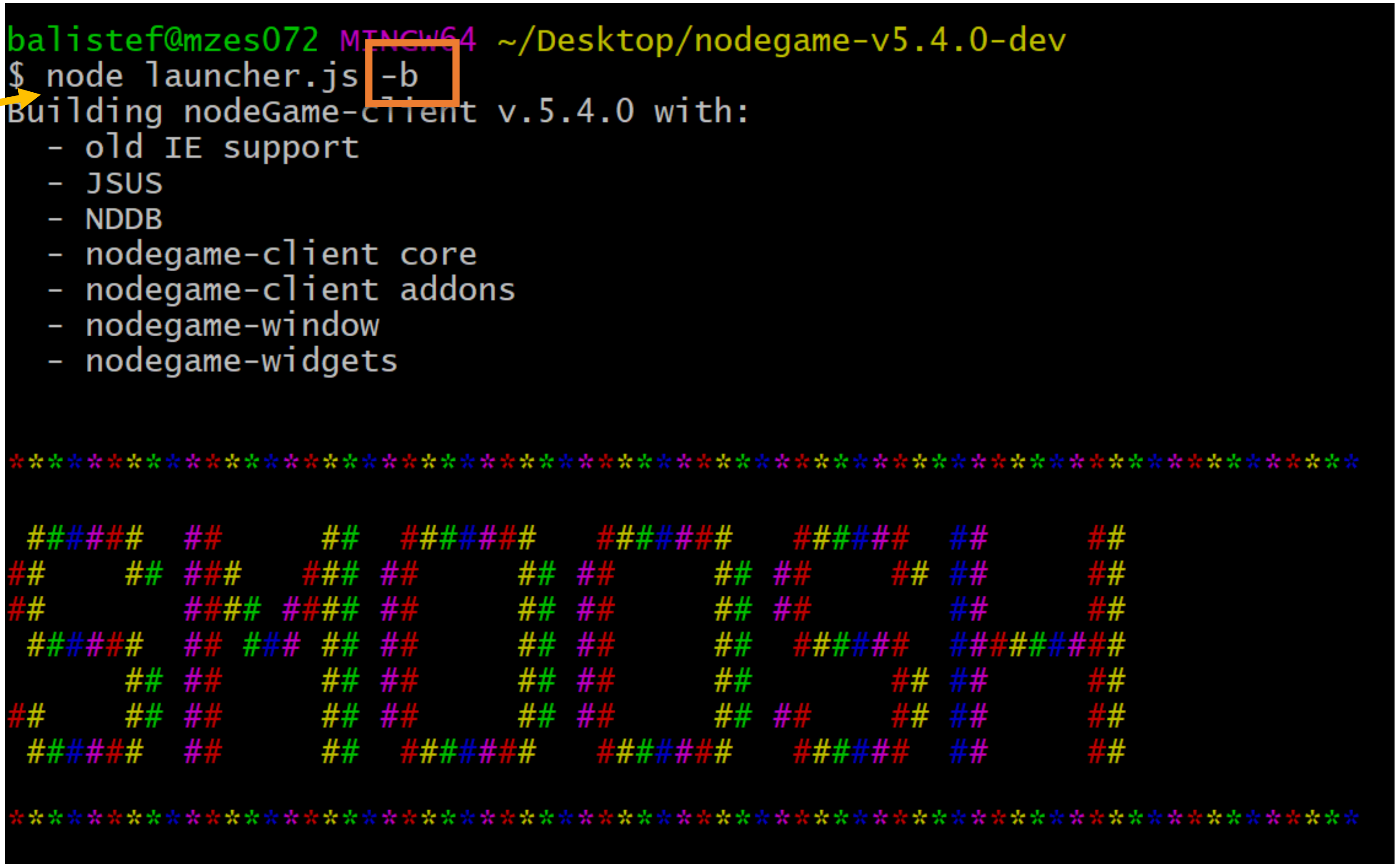
1. Add a **feedback form** at the end of the experiment
2. Add an **understanding quiz** after the instructions
3. Add a **bot client type**
4. Fix the timer issue?

Is Timer Always 00:00?



If so, follow these steps once:

1. Stop the server (Ctrl-C)
2. Start it with –b option to rebuild (smoosh) the client
3. TA DA!
4. *Still no Timer?* Sometimes browsers cache resources. Clear the cache, open a "Private Mode" tab, or try another browser.



```
balistef@mzes072 MINGW64 ~/Desktop/nodegame-v5.4.0-dev
$ node launcher.js -b
Building nodeGame-client v.5.4.0 with:
  - old IE support
  - JSUS
  - NDDB
  - nodegame-client core
  - nodegame-client addons
  - nodegame-window
  - nodegame-widgets


**********************************************************************

 ######  ##     ##  #######   #######   ######  ##     ##
##    ## ###   ### ##     ## ##     ## ##    ## ##     ##
##       #### #### ##     ## ##     ## ##       ##     ##
 ######  ## ### ## ##     ## ##     ##  ######  #########
      ## ##     ## ##     ## ##     ##       ## ##     ##
##    ## ##     ## ##     ## ##     ## ##    ## ##     ##
 ######  ##     ##  #######   #######   ######  ##     ##

**********************************************************************
```

Modify Game Sequence

game.stages.js

```
stager
.next('instructions')
.next('quiz')
.repeat('game', settings.REPEAT)
.next('feedback')
.next('end')
.gameover();
```

- Let's start by adding two new stages: 'feedback' and 'quiz'.
- **Remember!** When you develop a new game you can you skip stages with `stager.skip`

Implement The Feedback Stage

Feedback-Widget

player.js

```
stager.extendStep('feedback', {
   widget: {
      name: 'Feedback',
      options: {
         mainText: 'Leave comments here',
         minChars: 100,
         minWords: 5,
         showSubmit: false
         requiredChoice: true,
      }
   }
});
```

Implement The Feedback Stage

Feedback-Widget

player.js

```
stager.extendStep
  widget: {
    name: 'Feedback',
    options: {
      mainText: 'Leave comment
      minChars: 100,
      minWords: 5,
      showSubmit: false
      requiredChoice: true,
    }
```

Here, we are defining a "**widget-step,**" that is one widget will be added to the page and govern its behavior.

The name of the "**widget**"

mainText is a text shown *before* the widget (many widgets have this option)

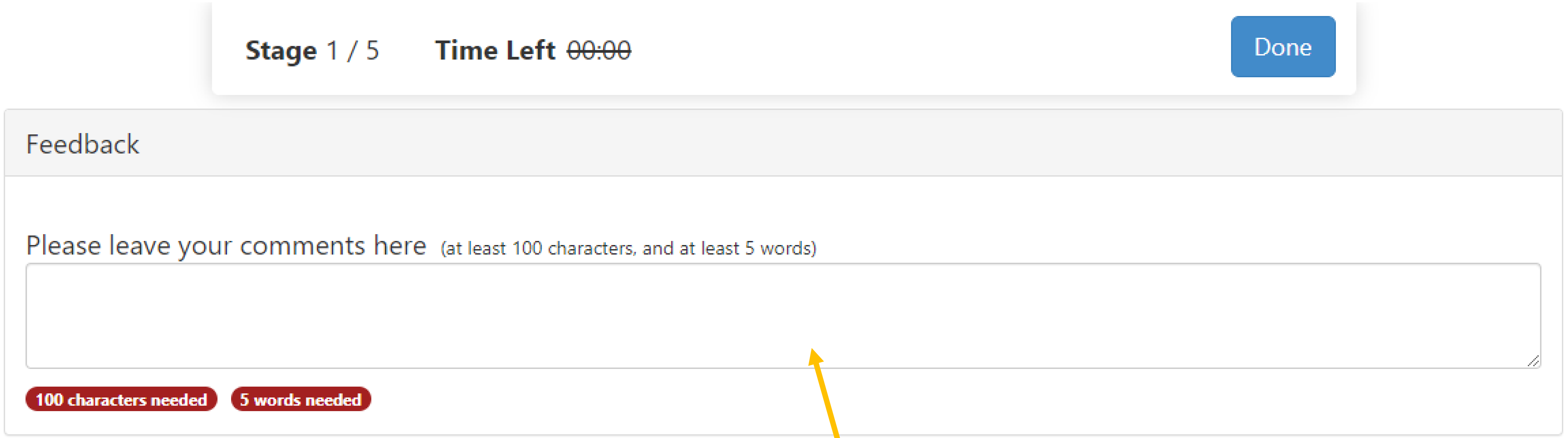
minChars and **minWords** are widget-specific options, and control how many characters and words must be typed in

showSubmit removes the submit button (we will use the Done button)

requiredChoice will prevent the user to continue if not enough input is typed in

Implement The Feedback Stage

[Feedback-Widget]


Stage 1 / 5 **Time Left** ~~00:00~~

Feedback

Please leave your comments here (at least 100 characters, and at least 5 words)

- The Feedback Widget is here! It requires to input at least 100 characters and 5 words.
- However, it look a bit ugly, because it stretches throughout the full page width. Let's make it centered.

Implement The Feedback Stage

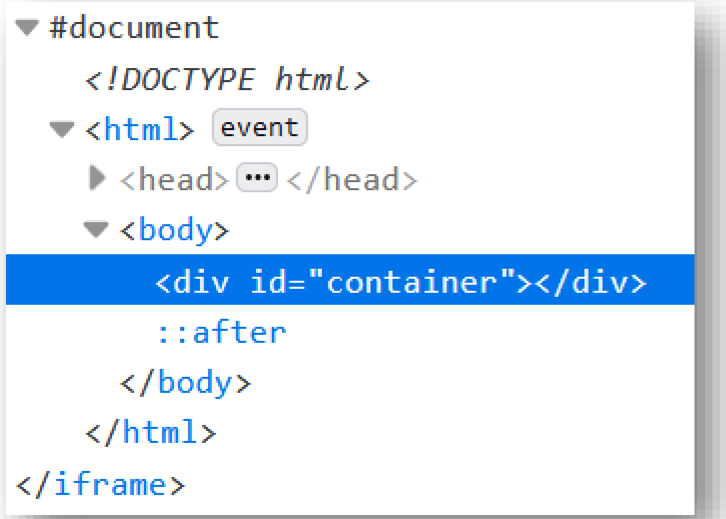
[Feedback-Widget]

player.js

```
stager.extendStep('feedback', {
    widget: {
        name: 'Feedback',
        root: 'container',
        options: {
            className: 'centered',
            mainText: 'Leave your comment
            minChars: 100,
            minWords: 5,
            showSubmit: false,
            requiredChoice: tr

        }
    }
});
```

root specified the id of element under which you want to append the widget. Here, we did not specify a *frame* step property, hence the default frame is used, which contains a DIV with id "container"

className makes sure to center the widget inside the its root element.



```
▼ #document
    <!DOCTYPE html>
  ▼ <html> event
    ▶ <head> ··· </head>
    ▼ <body>
        <div id="container"></div>
        ::after
      </body>
    </html>
</iframe>
```

Implement The Feedback Stage

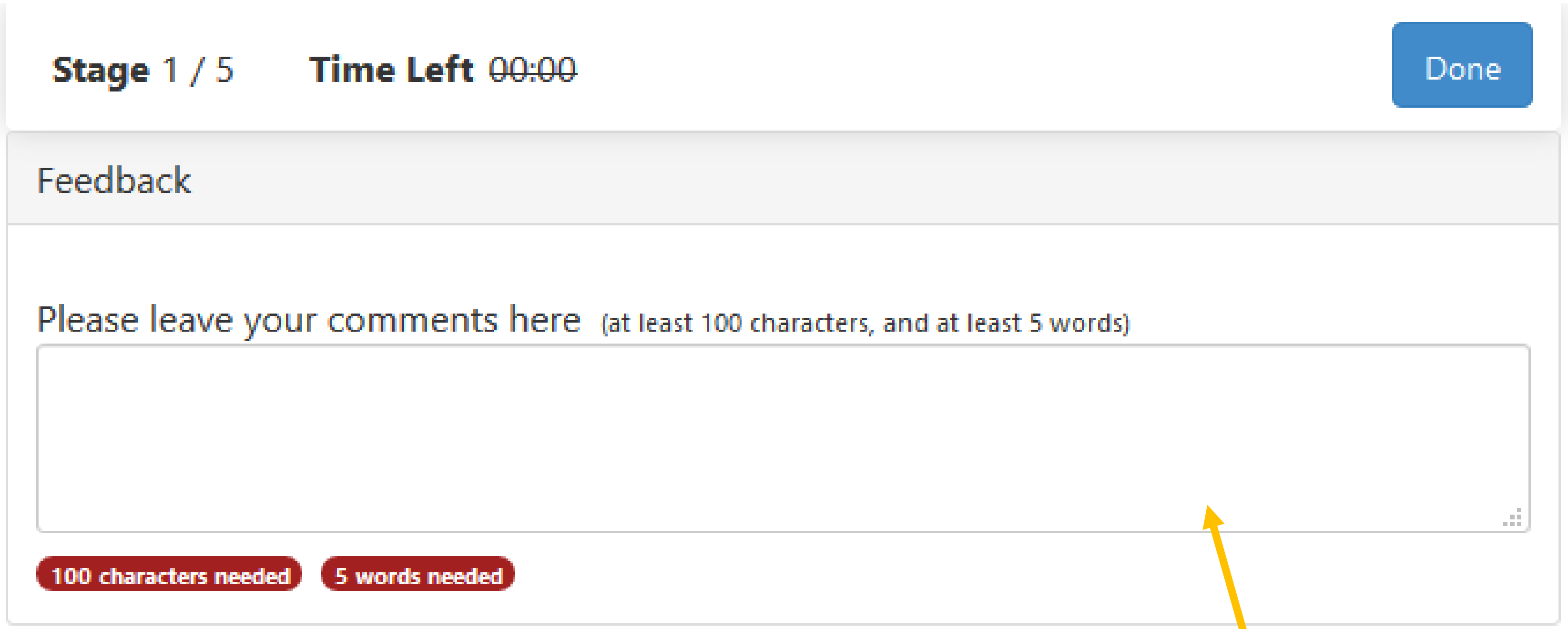
Feedback-Widget

player.js

```
stager.extendStep('feedback', {
    widget: {
        name: 'Feedback',
        root: 'container',
        options: {
            className: 'centered',
            mainText: 'Leave your comments here',
            minChars: 100,
            minWords: 5,
            showSubmit: false,
            requiredChoice: true

        }
    }
});
```

Implement The Feedback Stage

Feedback-Widget


Stage 1 / 5 **Time Left** ~~00:00~~

Feedback

Please leave your comments here (at least 100 characters, and at least 5 words)

100 characters needed

5 words needed

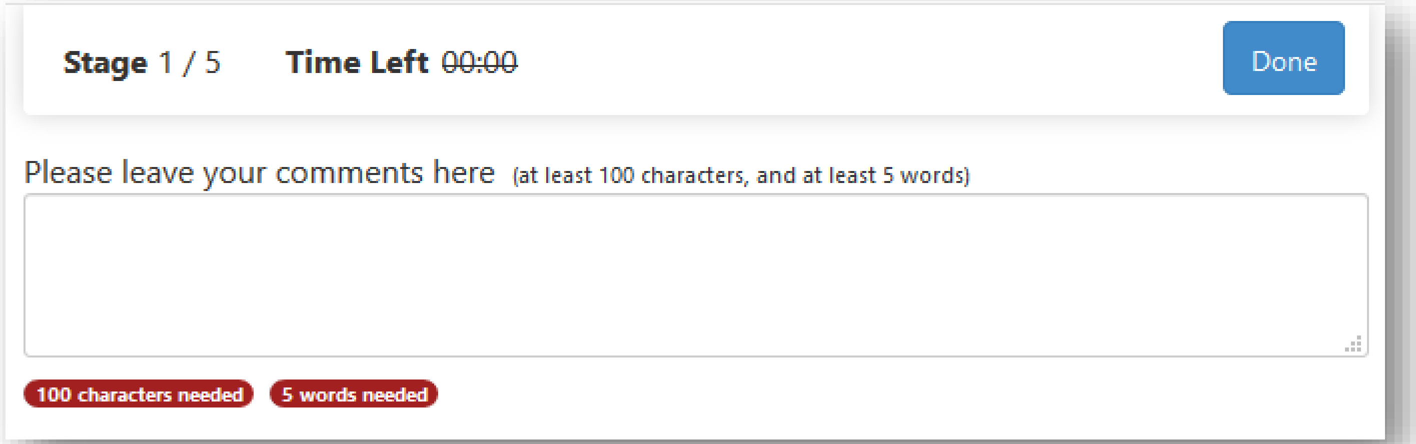
Well Done!
We can also make its appearance more sleek, by removing the panel and the title.

Implement The Feedback Stage

Feedback-Widget

player.js

```
stager.extendStep('feedback', {
    widget: {
        name: 'Feedback',
        root: 'container',
        options: {
            className: 'centered',
            mainText: 'Leave your comments here',
            minChars: 100,
            minWords: 5,
            requiredChoice: true,
            showSubmit: false,
            // For every widget.
            panel: false,
            title: false
        }
    }
});
```



Stage 1 / 5 **Time Left** ~~00:00~~ Done

Please leave your comments here (at least 100 characters, and at least 5 words)

100 characters needed 5 words needed

These options are valid for all widgets

How To Ask Feedback Properly?

- Eliciting feedback is **vital** when you pilot your experiment
- **Start with precise questions** to *collect the most important info first*
    - Was the purpose of the task clear?
    - Did you have enough time for **"Step X"?**
    - Did you follow a strategy for playing?

How To Ask Feedback Properly?

- Eliciting feedback is **vital** when you pilot your experiment
- **Start with precise questions** to *collect the most important info first*
    - Was the purpose of the task clear?
    - Did you have enough time for **"Step X"?**
    - Did you follow a strategy for playing?

Could be made more specific, i.e., checking for specific actions.

This is usually a crucial step (e.g., where the take an interactive decision)

Multiple choices are also appropriate here, but at a pilot stage, you might want to let them to answer with their own words

How To Ask Feedback Properly?

- Later on, move into **more general questions**
  - Did you feel the survey/game was boring/engaging/difficult?
  - Was the payment appropriate?
  - Was the task too long/too short?
  - Anything else you would like add.
- You may still keep the Feedback form in the main experiment if you have time/budget for it.

How To Ask Feedback Properly?

- Later on, move into **more general questions**
  - Did you feel the survey/game was boring/
  - Was the payment appropriate?
  - Was the task too long/too short?
  - Anything else you would like add.

Note! Basically, none will say *too short/too much money,* so you need to interpret their answers. On the other hand, if many say it was too long, you certainly have a problem.

- You may still keep the Feedback form in the main experiment if you have time/budget for it.

Implement The Quiz Stage

ChoiceManager-Widget

player.js

```
stager.extendStep('quiz', {
    widget: {
        name: 'ChoiceManager',
        root: 'container',
        options: {
            className: 'centered',
            mainText: 'A small quiz',
            forms: [

                // Here we add the questions.

            ]
        }
    }
});
```

Implement The Quiz Stage

ChoiceManager-Widget

player.js

```
stager.extendStep('quiz', {
    widget: {
        name: 'ChoiceManager',
        root: 'container',
        options: {
            className: 'centered',
            mainText: 'A small quiz',
            forms: [

                // Here we add the questions.

            ]
        }
    }
});
```

Here, we create a widget-step for widget **ChoiceManager.**
The choice manager contains and manages survey widgets ("choice" widgets).

forms is the array where we add the choice widgets, or objects specifying the options how to create them (and they are automatically created for us).

Implement The Quiz Stage

ChoiceManager-Widget

player.js

```
forms: [
{
   name: 'ChoiceTable',
   id: 'understand_roles',
   mainText: 'What are the roles in this game?',
   hint: 'I know you know it!',
   choices: [
      'Observer and Dictator',
      'Sancho and Pancho',
      'Batman and Robin',
      "I don't know",
      'I wish I\'d know it'
   ],
   correctChoice: 0,
   shuffleChoices: true
},
]
```

Implement The Quiz Stage

ChoiceManager-Widget

player.js

Here, we add a **ChoiceTable** widget, a table with clickable cells.

```
forms: [
{
    name: 'ChoiceTable',
    id: 'understand_roles',
    mainText: 'What are the roles in this game?',
    hint: 'I know you know it!',
    choices: [
        'Observer and Dictator',
        'Sancho and Pancho',
        'Batman and Robin',
        "I don't know",
        'I wish I\'d know it'
    ],
    correctChoice: 0,
    shuffleChoices: true
},
]
```

id is the name under which the answer is saved in the data (not displayed to the user)

hint is a small explanation after the main text

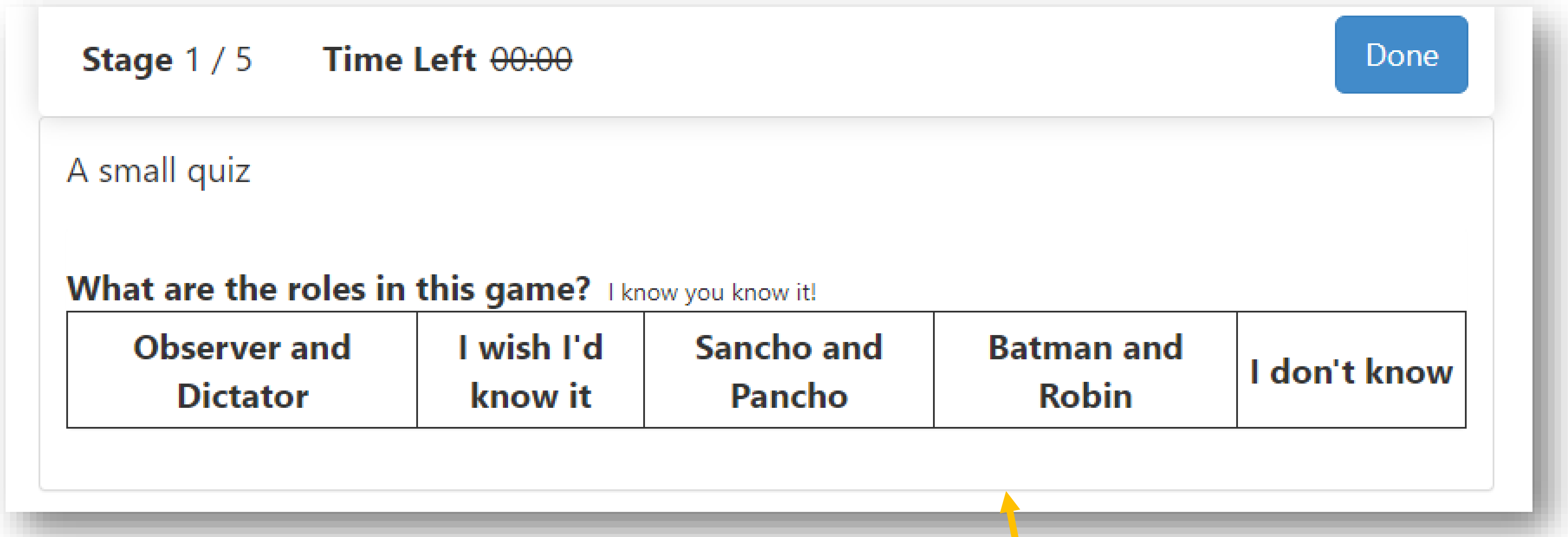
choices is an *array* containing texts or numbers that will be displayed in the table's cells.

Note! You need to escape quotes (\' or \"), or use a different quote to wrap the whole string.

correctChoice marks the position of the correct choice (0-indexed). **shuffleChoices** displays the choices in random order.

Implement The Quiz Stage

ChoiceManager-Widget



Stage 1 / 5 **Time Left** ~~00:00~~

A small quiz

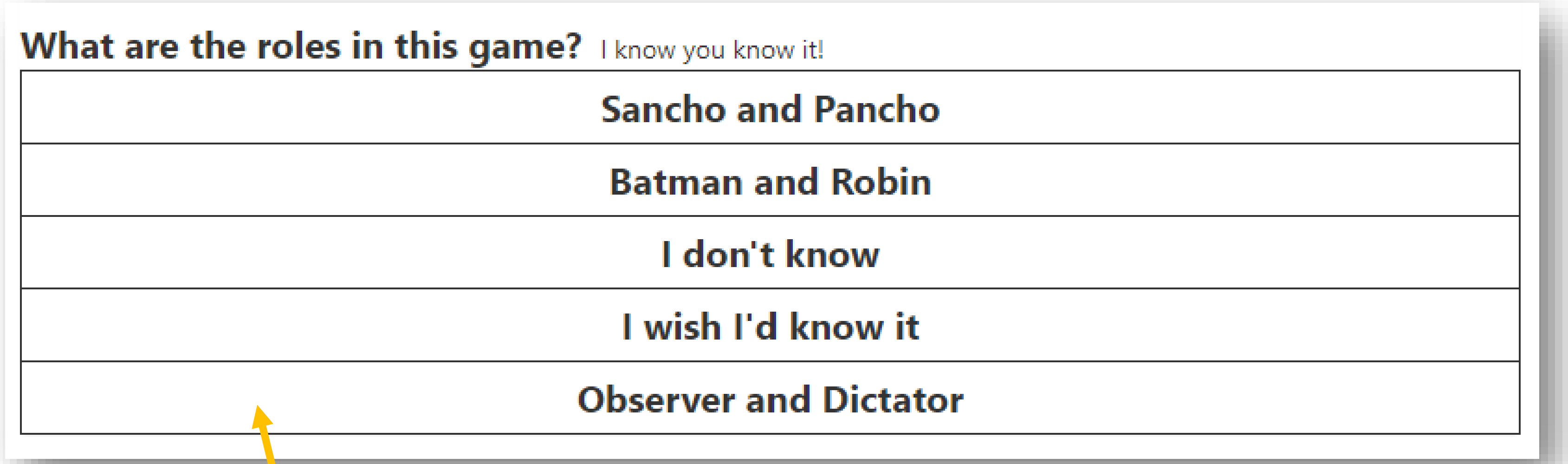
What are the roles in this game? I know you know it!

Observer and Dictator	I wish I'd know it	Sancho and Pancho	Batman and Robin	I don't know

By default there is no title and no panel around the choice widgets inside the ChoiceManager.

Implement The Quiz Stage

ChoiceManager-Widget


What are the roles in this game? I know you know it!

Sancho and Pancho

Batman and Robin

I don't know

I wish I'd know it

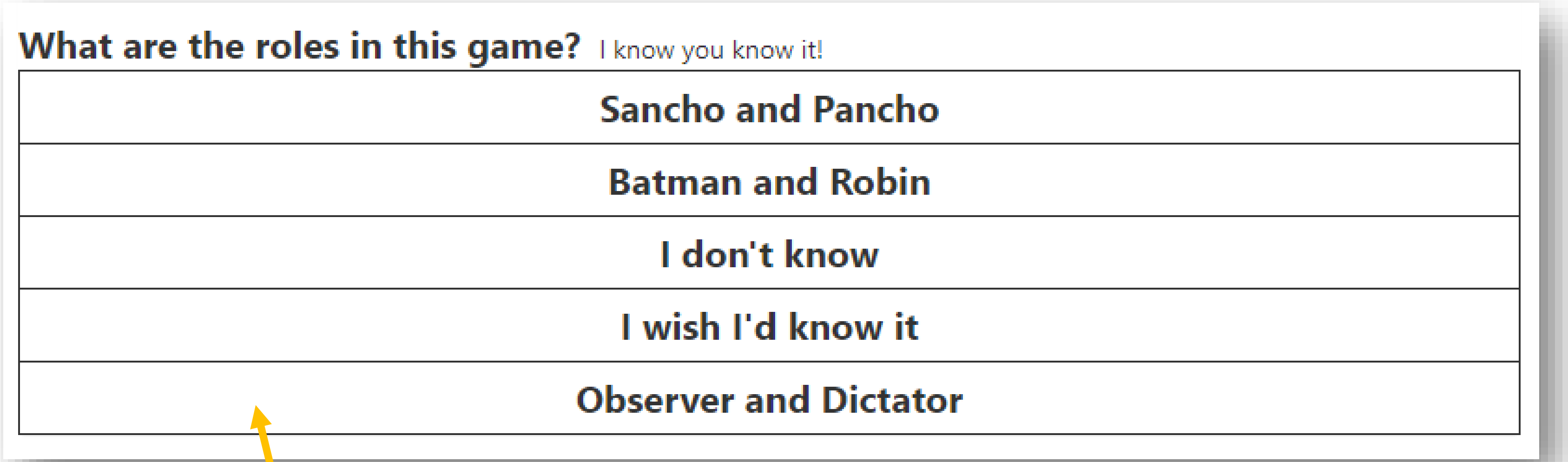
Observer and Dictator

```
orientation: 'V' // vertical
```

- You can set the vertical display adding the option **orientation.**
- This is useful when you have larger texts to be selected (e.g., when testing differences between treatments)

Implement The Quiz Stage

ChoiceManager-Widget



What are the roles in this game? I know you know it!

Sancho and Pancho
Batman and Robin
I don't know
I wish I'd know it
Observer and Dictator

How to align the text to the left?

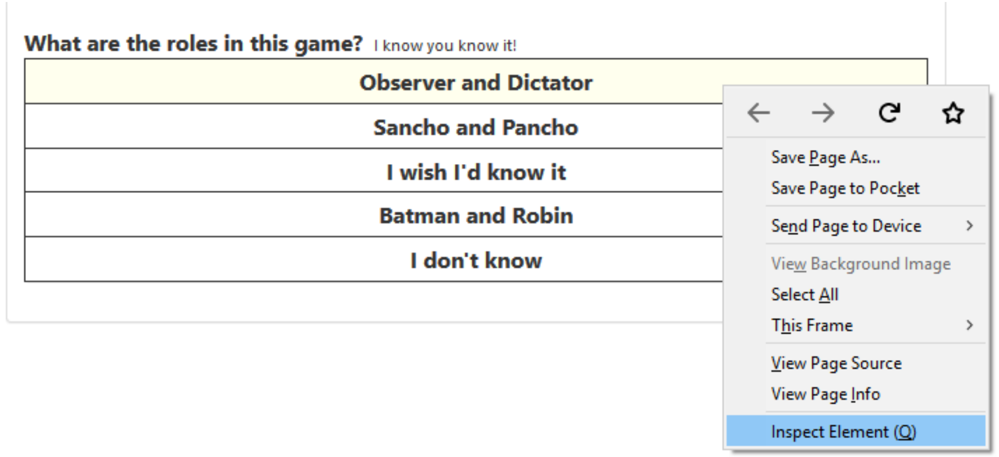
We need to specify a CSS rule! But How?

```
orientation: 'V' // vertical
```

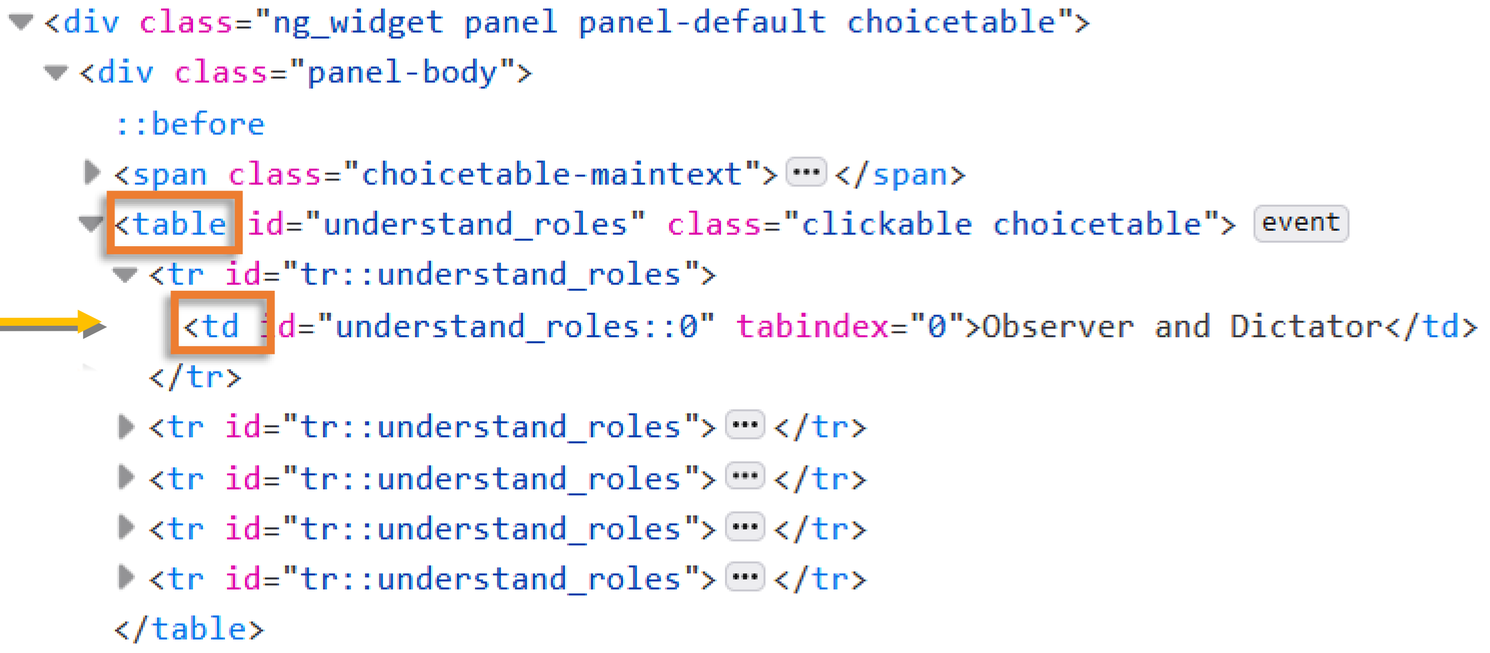
- You can set the vertical display adding the option **orientation.**
- This is useful when you have larger texts to be selected (e.g., when testing differences between treatments)

Adding a New CSS Rule

HTML Tables


What are the roles in this game? I know you know it!

Observer and Dictator
Sancho and Pancho
I wish I'd know it
Batman and Robin
I don't know


```
▼ <div class="ng_widget panel panel-default choicetable">
  ▼ <div class="panel-body">
      ::before
    ▶ <span class="choicetable-maintext"> ••• </span>
    ▼ <table id="understand_roles" class="clickable choicetable"> event
      ▼ <tr id="tr::understand_roles">
          <td id="understand_roles::0" tabindex="0">Observer and Dictator</td>
        </tr>
      ▶ <tr id="tr::understand_roles"> ••• </tr>
      ▶ <tr id="tr::understand_roles"> ••• </tr>
      ▶ <tr id="tr::understand_roles"> ••• </tr>
      ▶ <tr id="tr::understand_roles"> ••• </tr>
      </table>
```

Let's **inspect** the element we want to style.
We see that the ChoiceTable widget is indeed creating a <**TABLE**>.
Inside the table, each row is inside a <TR>, and each cell inside a <**TD**> element.
We need to style the *all* the TD tags inside the TABLE with id "understand_roles".

Adding a Custom CSS Rule to the Page

CSS Syntax

player.js

```
stager.extendStep('quiz', {
    widget: {
        name: 'ChoiceManager',
        // Further options hidden.
        }
    },
    cb: function() {

        W.cssRule('#understand_roles td { text-align: left; }');
    }
});
```

W.cssRule lets us add a quick modification to the default CSS rule of the page.

However, if you need many new rules, it makes sense to write a separate CSS file and import into the page with the <link> tag.

Adding a Custom CSS Rule to the Page

CSS Syntax

player.js

```
stager.extendStep('quiz', {
    widget: {
        name: 'ChoiceManager',
        // Further options hidden.
        }
    },
    cb: function() {

        W.cssRule('#understand_roles td { text-align: left; }');
    }
});
```

SELECTOR RULE

A CSS rule is composed of two parts:

- A **selector**, which finds the elements to which the rule applies
- The **rule** itself

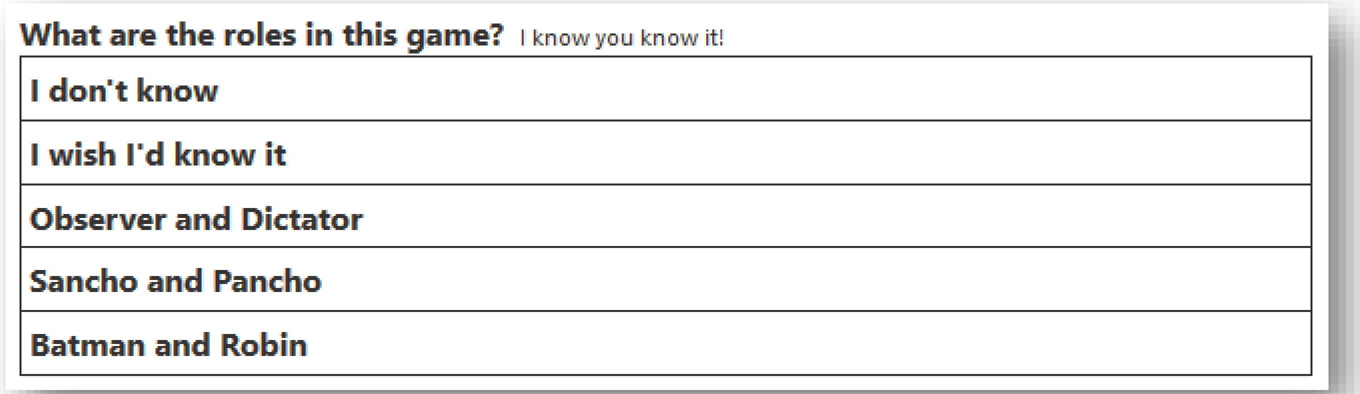
Adding a Custom CSS Rule to the Page

CSS Syntax

player.js

```
stager.extendStep('quiz', {
    widget: {
        name: 'ChoiceManager',
        // Further options hidden.
        }
    },
    cb: function() {

        W.cssRule('#understand_roles td { text-align: left; }');
    }
});
```

SELECTOR



What are the roles in this game? I know you know it!

I don't know
I wish I'd know it
Observer and Dictator
Sancho and Pancho
Batman and Robin

#understand_roles means the element with id "understand_roles"
#understand_roles td means all the TD elements inside the element with id "understand_roles"

Adding a Custom CSS Rule to the Page

CSS Syntax

player.js

```
stager.extendStep('quiz', {
    widget: {
        name: 'ChoiceManager',
        // Further options hidden.
        }
    },
    cb: function() {

        W.cssRule('#understand_roles td { text-align: left; }');
    }
});
```

SELECTOR

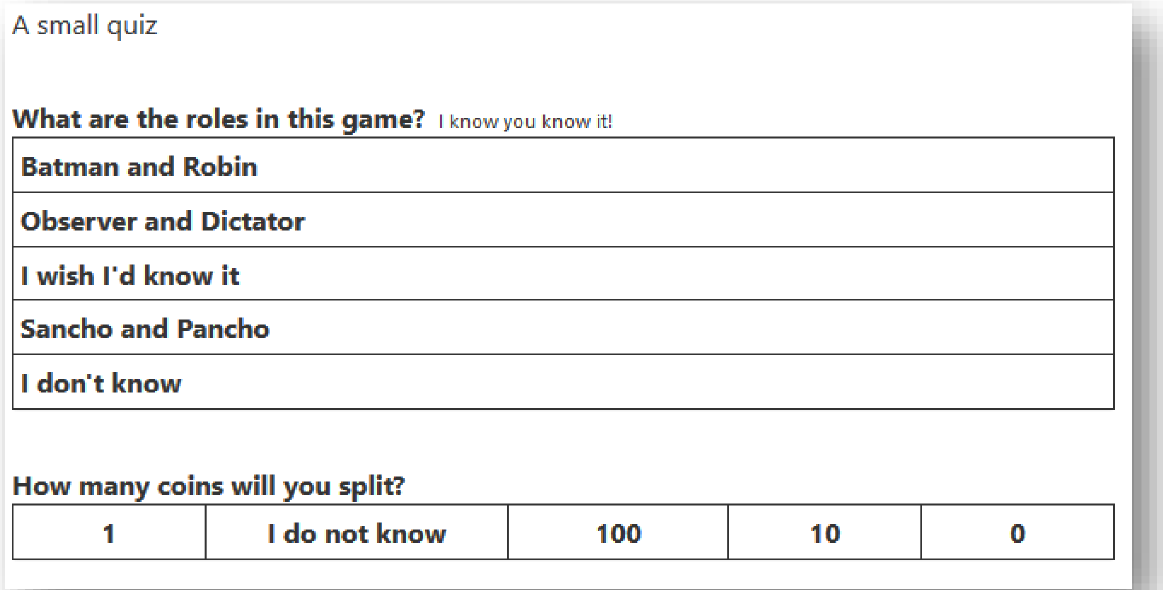
Note! This rule applies only the first question with id "understand_roles." If we have more quiz questions to style we could call W.cssRule several times with different ids, but a better strategy is to define **a more general rule.** *How?* **Hint.** Check the CSS Syntax link to learn how to select elements with a given class.

Add a Second Quiz Question

player.js

```
forms: [
{
  // First question (code hidden)
},
{
    name: 'ChoiceTable',
    id: 'understand_money',
    mainText: 'How many coins will you split?',
    choices: [
       0, 1, 10, 100, 'I do not know'
    ],
    correctChoice: 3,
    shuffleChoices: true
}
]
```

We just add a second object after the first one in the **forms** array.
Do not forget the **comma**!


A small quiz

What are the roles in this game? I know you know it!

Batman and Robin
Observer and Dictator
I wish I'd know it
Sancho and Pancho
I don't know

How many coins will you split?

1	I do not know	100	10	0

Adding a Bot to the Game

- A **bot** is a computer controlled player that goes through the same stages and steps
- A bot **does not need to visualize any HTML,** instructions, quizzes, etc.

- Important! There *is no need to implement a bot* if the computer-made decisions are simple and limited to one step only. You could code those decisions in the logic file.

- However, writing a bot has some *advantages:*
  - A bot gives a clearer separation of code
  - A bot lets you use the Play with Bots option from the waiting room
  - A bot can replace a human player that dropped out

Adding a Bot to the Game

bot.js

To implement a bot you need to add your code to file bot.js in folder client_types/

```
stager.setDefaultCallback(function() {
   this.node.timer.randomDone();
});
```

First, we are setting the default callback for every step.

In the callback, we are telling the bot to be **DONE** after a random time interval, using the **node.timer.randomDone()** method.

DONE is a special nodeGame event that ends the current step. After all players are DONE, the game will proceed to the next step in the sequence.

Adding a Bot to the Game

bot.js

To implement a bot you need to add your code to file bot.js in folder client_types/

```
stager.setDefaultCallback(function() {
   this.node.timer.randomDone();
});
```

Why are we using **this.node** instead of just **node** ?

The node object is a *global variable in the browser*, i.e., it is accessible from anywhere in the code.

However, *in the server node is not global,* i.e, you need a reference to access it. When the callback function is executed a special reference is added inside it, and you can access it through the **this** operator.

Adding a Bot to the Game

Next, you need to code the behavior of the bot for those steps that need a decision.
The easiest approach is to copy the relevant code from the player client type and adapt it.

```
stager.extendStep('game', {
roles: {
    DICTATOR: {
        cb: function() {
            this.node.done({ offer: 1 });
        }
    },
    OBSERVER: {
        cb: function() {
            // Store a local reference of node.
             var node = this.node;
            node.on.data('decision', function(msg) {
                setTimeout(function() {
                    node.done();
                }, 5000);
            });
        }
    }
}});
```

After we removed all the unnecessary HTML manipulation, this is the bare skeleton we are left with.

If the bot is a **dictator,** it will always make an offer of 1.

If the bot is an **observer,** it will wait for the offer and then simply call DONE after waiting exactly 5 seconds.

Adding a Bot to the Game

Next, you need to code the behavior of the bot for those steps that need a decision.
The easiest approach is to copy the relevant code from the player client type and adapt it.

```
stager.extendStep('game', {
roles: {
    DICTATOR: {
        cb: function() {
            this.node.done({ offer: 1 });
        }
    },
    OBSERVER: {
        cb: function() {
            // Store a local reference of node.
             var node = this.node;
            node.on.data('decision', function(msg) {
                setTimeout(function() {
                    node.done();
                }, 5000);
            });
        }
    }
}});
```

node.on.data is waiting for a message from the server.

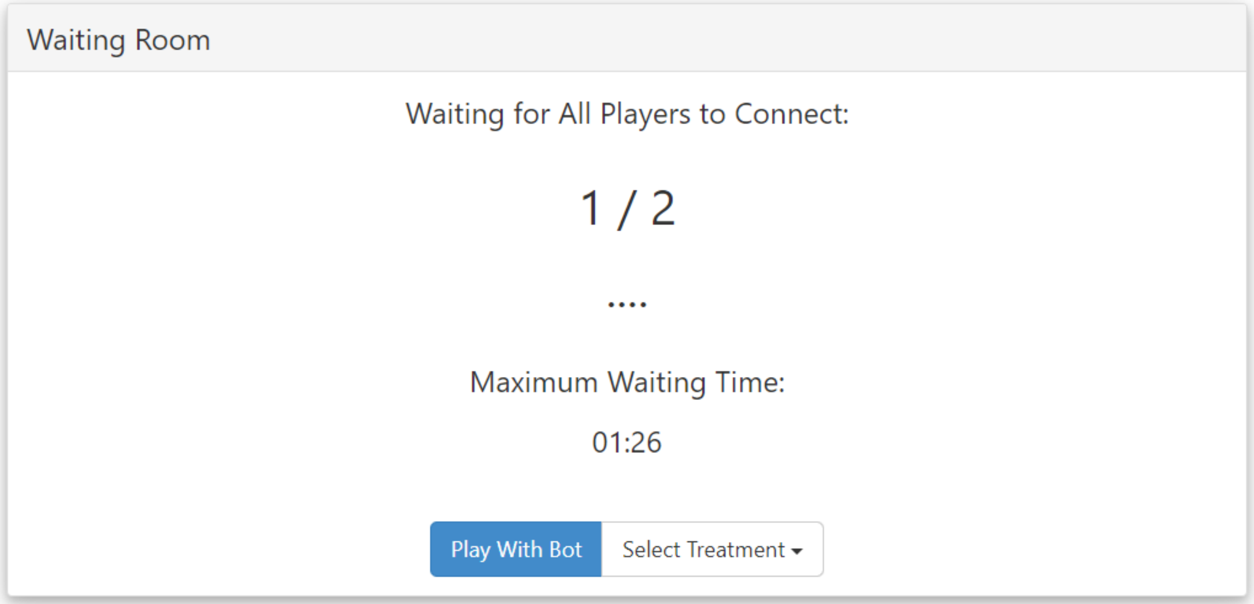
When a message arrives with the label **'decision'**, the callback function is executed with the message as input parameter.

A more complex bot in a more complex game could make use of this information to send a reply accordingly.

Enable Play with Bots in Waiting Room

The bot is ready, you can now test it!
Enable bots in the **waitroom.settings.js** file.

```
/** ### ALLOW_PLAY_WITH_BOTS
 *
 * Allows a player to request to start the game immediately with bots
 *
 * A button is added to the interface.
 */
ALLOW_PLAY_WITH_BOTS: true,

/** ### ALLOW_SELECT_TREATMENT
 *
 * Allows a player to select the treatment for the game
 *
 * This option requires `ALLOW_PLAY_WITH_BOTS` to be TRUE.
 *
 * A button is added to the interface.
 */
ALLOW_SELECT_TREATMENT: true
```



Waiting Room

Waiting for All Players to Connect:

1 / 2

....

Maximum Waiting Time:

01:26