nodeGame

social experiments in the browser

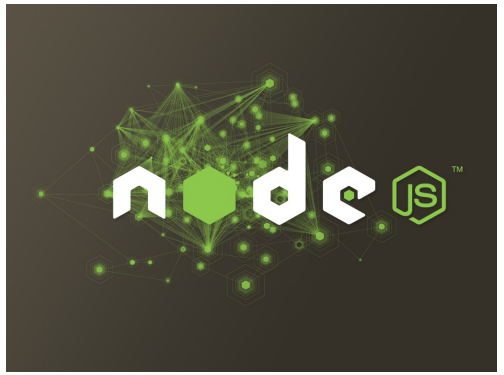
v.0.5

nodegame.org

Stefano Balietti - sbalietti@ethz.ch - ETH Zurich, SOMS Chair of Sociology, Modeling and Simulations

nodeGame

NodeGame is a free, open source JavaScript framework to design and conduct social experiment *online* in the *browser window.*



nodeGame

- It is a **modular** framework
- Clients/server architecture
- **Websockets** guarantees scalability and real-time response
- **Event-based** (it's JavaScript!)
- Message-passing
- **API** to support most common tasks
- Works on mobile and tablet devices

Let's look under the hood...

```
<!doctype html>
<title>Ultimatum Game</title>

<!-- Loading  nodeGame libraries and CSS -->
<script src="/socket.io/socket.io.js"></script>
<script src="/javascripts/nodegame-full.js" charset="utf-8"></script>
<link rel='stylesheet' href='/stylesheets/player.css'></link>
<!-- end -->

<!-- Loading the Ultimatum game -->
<script src="./Ultimatum.js" charset="utf-8"></script>
<!-- end -->

<body>
<div id="root"></div>
<script>
  window.onload = function () {

     // Create the Client
     var conf = {
     url: "/ultimatum",
          verbosity: 10,
            player: {
               name: "P_" + Math.floor(Math.random()*100)
          }
     };

      node.play(conf, new Ultimatum());
}
</script>
</body>
```

```
<!doctype html>
<title>Ultimatum Game</title>
```

import the nodeGame library

```
<!-- Loading  nodeGame libraries and CSS -->
<script src="/socket.io/socket.io.js"></script>
<script src="/javascripts/nodegame-full.js" charset="utf-8"></script>
<link rel='stylesheet' href='/stylesheets/player.css'></link>
<!-- end -->

<!-- Loading the Ultimatum game -->
<script src="./Ultimatum.js" charset="utf-8"></script>
<!-- end -->

<body>
<div id="root"></div>
<script>
  window.onload = function () {

    // Create the Client
    var conf = {
    url: "/ultimatum",
        verbosity: 10,
          player: {
             name: "P_" + Math.floor(Math.random()*100)
        }
    };

     node.play(conf, new Ultimatum());
}
</script>
</body>
```

```
<!doctype html>
<title>Ultimatum Game</title>

<!-- Loading  nodeGame libraries and CSS -->
<script src="/socket.io/socket.io.js"></script>
<script src="/javascripts/nodegame-full.js" charset="utf-8"></script>
<link rel='stylesheet' href='/stylesheets/player.css'></link>
<!-- end -->

<!-- Loading the Ultimatum game -->
<script src="./Ultimatum.js" charset="utf-8"></script>
<!-- end -->

<body>
<div id="root"></div>
<script>
  window.onload = function () {

    // Create the Client
    var conf = {
    url: "/ultimatum",
        verbosity: 10,
          player: {
             name: "P_" + Math.floor(Math.random()*100)
        }
    };

     node.play(conf, new Ultimatum());
}
</script>
</body>
```

import the specific game

```
<!doctype html>
<title>Ultimatum Game</title>

<!-- Loading  nodeGame libraries and CSS -->
<script src="/socket.io/socket.io.js"></script>
<script src="/javascripts/nodegame-full.js" charset="utf-8"></script>
<link rel='stylesheet' href='/stylesheets/player.css'></link>
<!-- end -->

<!-- Loading the Ultimatum game -->
<script src="./Ultimatum.js" charset="utf-8"></script>
<!-- end -->

<body>
<div id="root"></div>
<script>
  window.onload = function () {

    // Create the Client
    var conf = {
    url: "/ultimatum",
        verbosity: 10,
          player: {
             name: "P_" + Math.floor(Math.random()*100)
        }
    };

     node.play(conf, new Ultimatum());
}
</script>
</body>
```

configure nodeGame

```
<!doctype html>
<title>Ultimatum Game</title>

<!-- Loading  nodeGame libraries and CSS -->
<script src="/socket.io/socket.io.js"></script>
<script src="/javascripts/nodegame-full.js" charset="utf-8"></script>
<link rel='stylesheet' href='/stylesheets/player.css'></link>
<!-- end -->

<!-- Loading the Ultimatum game -->
<script src="./Ultimatum.js" charset="utf-8"></script>
<!-- end -->

<body>
<div id="root"></div>
<script>
  window.onload = function () {

    // Create the Client
    var conf = {
    url: "/ultimatum",
        verbosity: 10,
          player: {
             name: "P_" + Math.floor(Math.random()*100)
        }
    };

    node.play(conf, new Ultimatum());
}
</script>
</body>
```

start the game

Game's anatomy

Global settings:

the conditions for advancing through the game states
the minimum number of players, etc.

Init Function:

executed before the game starts (special meaning)

Game Loop:

contains the functions to be executed sequentially as the game advances

```
function Ultimatum () {
```

global settings

```
    this.name = 'Ultimatum Game';
    this.description = 'The receiver of the offer can accept or reject.';

    this.auto_step = false;
    this.auto_wait = true;

    this.minPlayers = 2;
    this.maxPlayers = 10;

    this.init = function() {
        // init the game
    };

    this.loop = {

        1: {
            state: function() {
                // Do something, e.g. shows the instructions
            }
        },

        2: {
            state: function() {
                // Do something else, e.g. make a proposal to the player
            }
        }

        // more states here
    };
}
```

```
function Ultimatum () {

    this.name = 'Ultimatum Game';
    this.description = 'The receiver of the offer can accept or reject.';

    this.auto_step = false;
    this.auto_wait = true;

    this.minPlayers = 2;
    this.maxPlayers = 10;

    this.init = function() {
        // init the game
    };

    this.loop = {

        1: {
            state: function() {
                // Do something, e.g. shows the instructions
            }
        },

        2: {
            state: function() {
                // Do something else, e.g. make a proposal to the player
            }
        }

        // more states here
    };
}
```

init function

```
function Ultimatum () {

    this.name = 'Ultimatum Game';
    this.description = 'The receiver of the offer can accept or reject.';

    this.auto_step = false;
    this.auto_wait = true;

    this.minPlayers = 2;
    this.maxPlayers = 10;

    this.init = function() {
        // init the game
    };
```

game loop

```
    this.loop = {

        1: {
            state: function() {
                // Do something, e.g. shows the instructions
            }
        },

        2: {
            state: function() {
                // Do something else, e.g. make a proposal to the player
            }
        }

        // more states here
    };
}
```

How to write the game loop

- User behavior is captured through standard JS **events listeners**
- Is encapsulated in a **game message**
- Game messages are **exchanged** between players
- A game is a set of rules that specify **what to do** upon **which event**
- NodeGame provides an API for dealing with most of the common tasks

Ultimatum Bidder

```
// Load the screen for the bidder
W.loadFrame('html/bidder.html', function () {

    // Sends a message to another player
    node.say(75, 'OFFER', 'RESPONDENT_ID');

    // Stores the value of the offer in the server's memory
    node.set(75, 'OFFER');

    // Waits for the reply
    node.onDATA('ACCEPT', function (msg) {
        W.write('Your offer was accepted.');
        node.DONE();
    });

    node.onDATA('REJECT', function (msg) {
        W.write('Your offer was rejected.');
        node.DONE();
    });

});
```

load the screen and executes the callback when ready

Ultimatum Bidder

```
// Load the screen for the bidder
W.loadFrame('html/bidder.html', function () {

    // Sends a message to another player
    node.say(75, 'OFFER', 'RESPONDENT_ID');

    // Stores the value of the offer in the server's memory
    node.set(75, 'OFFER');

    // Waits for the reply
    node.onDATA('ACCEPT', function (msg) {
        W.write('Your offer was accepted.');
        node.DONE();
    });

    node.onDATA('REJECT', function (msg) {
        W.write('Your offer was rejected.');
        node.DONE();
    });

});
```

send message

Ultimatum Bidder

```
// Load the screen for the bidder
W.loadFrame('html/bidder.html', function () {

    // Sends a message to another player
    node.say(75, 'OFFER', 'RESPONDENT_ID');

    // Stores the value of the offer in the server's memory
    node.set(75, 'OFFER');

    // Waits for the reply
    node.onDATA('ACCEPT', function (msg) {
        W.write('Your offer was accepted.');
        node.DONE();
    });

    node.onDATA('REJECT', function (msg) {
        W.write('Your offer was rejected.');
        node.DONE();
    });

});
```

This message will be saved in the memory object together with the timestamp, the state, and the sender.

Ultimatum Bidder

```
// Load the screen for the bidder
W.loadFrame('html/bidder.html', function () {

    // Sends a message to another player
    node.say(75, 'OFFER', 'RESPONDENT_ID');

    // Stores the value of the offer in the server's memory
    node.set(75, 'OFFER');

    // Waits for the reply
    node.onDATA('ACCEPT', function (msg) {
        W.write('Your offer was accepted.');
        node.DONE();
    });

    node.onDATA('REJECT', function (msg) {
        W.write('Your offer was rejected.');
        node.DONE();
    });

});
```

This message will be saved in the memory object together with the timestamp, the state, and the sender.

It will also be part of the output file when the experiment is finished.

Ultimatum Bidder

```
// Load the screen for the bidder
W.loadFrame('html/bidder.html', function () {

    // Sends a message to another player
    node.say(75, 'OFFER', 'RESPONDENT_ID');

    // Stores the value of the offer in the server's memory
    node.set(75, 'OFFER');

    // Waits for the reply
    node.onDATA('ACCEPT', function (msg) {
        W.write('Your offer was accepted.');
        node.DONE();
    });

    node.onDATA('REJECT', function (msg) {
        W.write('Your offer was rejected.');
        node.DONE();
    });

});
```

Waits for a reply from the Respondent and then terminates the state (DONE)

Ultimatum Respondent

```
// Load the screen for the bidder
W.loadFrame('html/respondent.html', function () {

    // Waits for the offer from another player
    node.onDATA('OFFER', function(msg){
        W.write('You received an offer of ' + msg.data);
    });

    // Listen on the user clicks
    accept.onclick = function() {
        node.say('ACCEPT', 'ACCEPT');
        node.set('ACCEPT');
        node.DONE();
    };

    reject.onclick = function() {
        node.say('REJECT', 'REJECT');
        node.set('REJECT');
        node.DONE();
    };
});
```

Ultimatum Respondent

```
// Load the screen for the bidder
W.loadFrame('html/respondent.html', function () {

    // Waits for the offer from another player
    node.onDATA('OFFER', function(msg){
        W.write('You received an offer of ' + msg.data);
    });

    // Listen on the user clicks
    accept.onclick = function() {
        node.say('ACCEPT', 'ACCEPT');
        node.set('ACCEPT');
        node.DONE();
    };

    reject.onclick = function() {
        node.say('REJECT', 'REJECT');
        node.set('REJECT');
        node.DONE();
    };
});
```

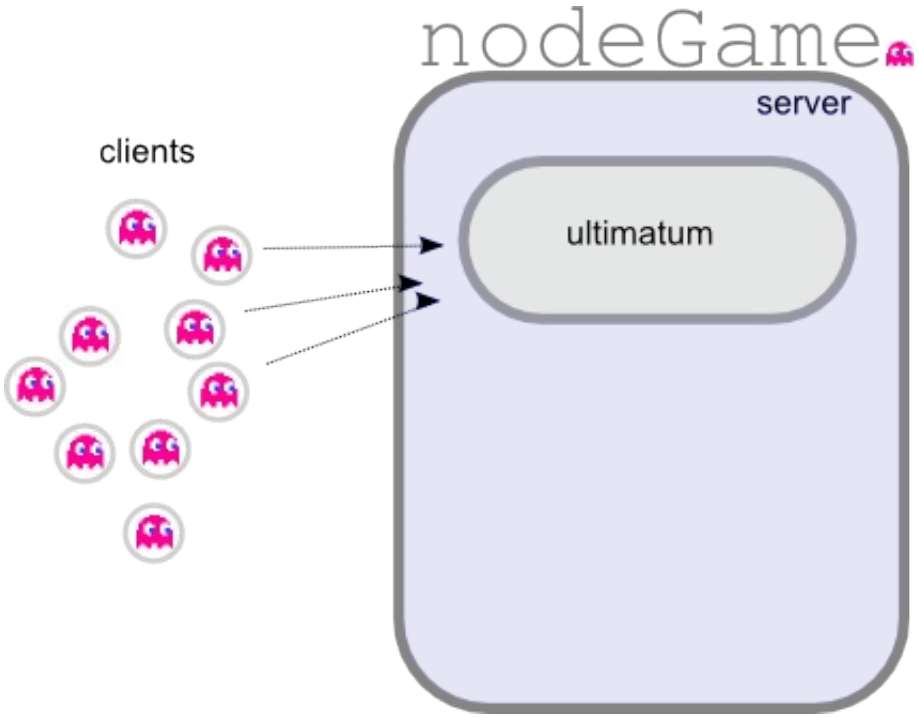
Waits for an incoming offer

Ultimatum Respondent

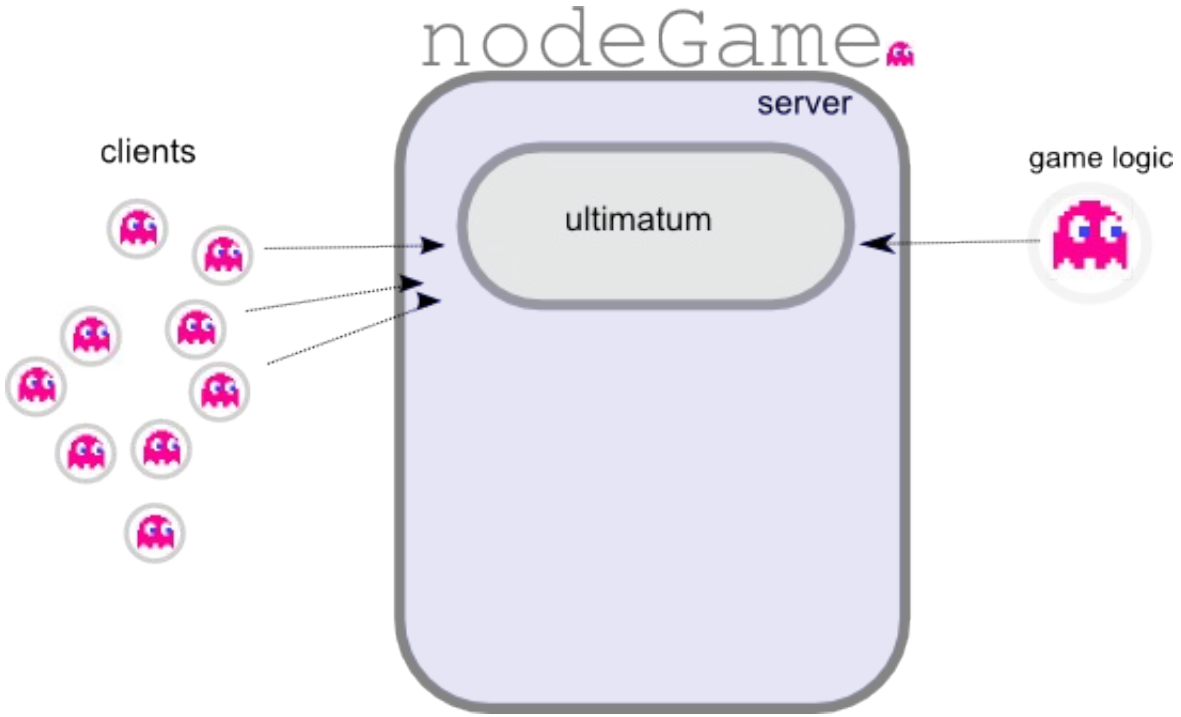
```
// Load the screen for the bidder
W.loadFrame('html/respondent.html', function () {

    // Waits for the offer from another player
    node.onDATA('OFFER', function(msg){
        W.write('You received an offer of ' + msg.data);
    });

    // Listen on the user clicks
    accept.onclick = function() {
        node.say('ACCEPT', 'ACCEPT');
        node.set('ACCEPT');
        node.DONE();
    };

    reject.onclick = function() {
        node.say('REJECT', 'REJECT');
        node.set('REJECT');
        node.DONE();
    };
});
```

Waits for the user click, then
Sends the reply to the player,
Store the reply in the server memory,
And terminates the state (DONE)

Summary



Summary



Game Logic

- Game-logic follows the **same structure of the game clients**
- It connects to the server from localhost
- **Keep tracks of the state** of all the players
- Has a **database** object containing all the stored valued
- Has access to **higher privilege** functions on the server, e.g. :
  - can advance the game
  - can redirect players to other game rooms

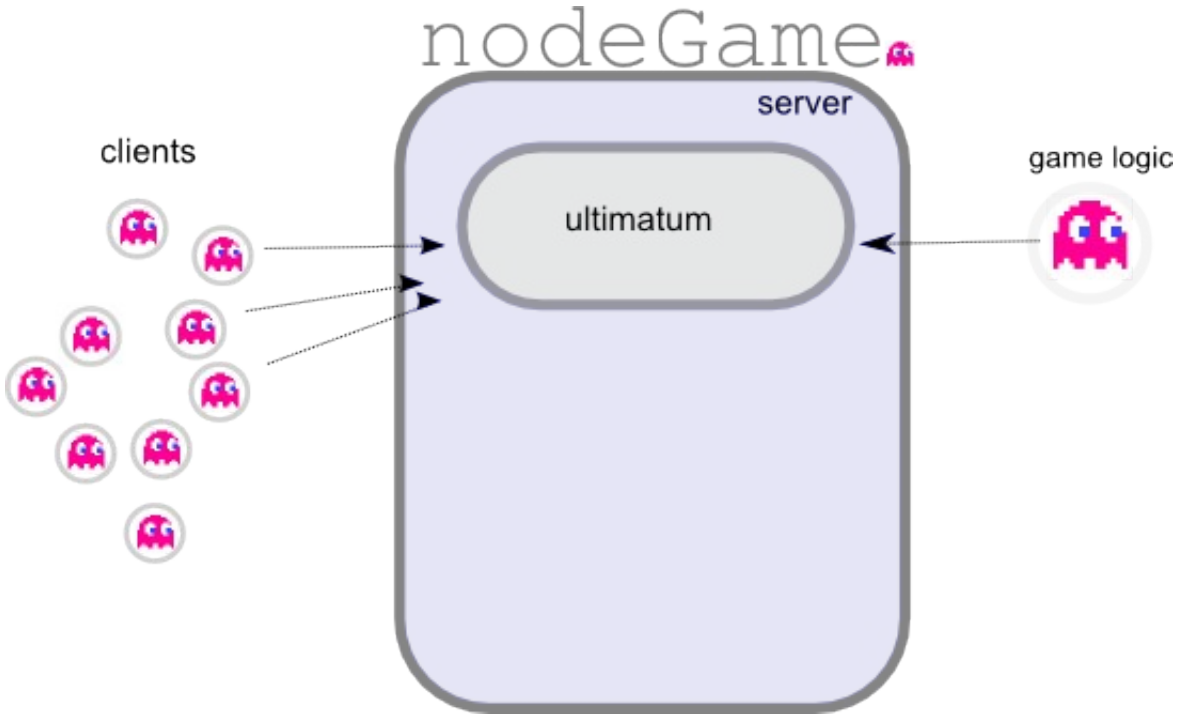
Commands in the Game Logic

```
// Pairs all players // a
var groups = node.game.pl.getGroupsSizeN(2);
```

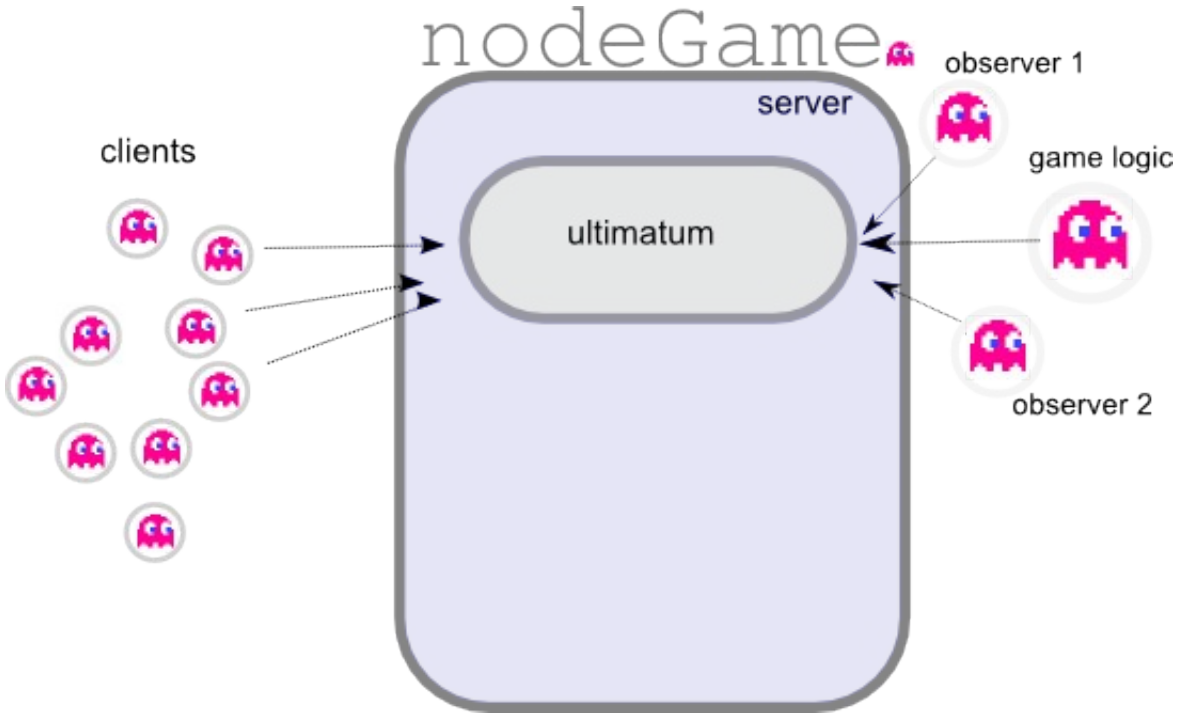
```
// communicates the winning to each player
node.game.pl.each(function(p) {
   node.say(p.win, 'WIN', p.id);
});
```

```
// save the results of the game to file system
// as JSON file
node.game.memory.save('./results.nddb');
```

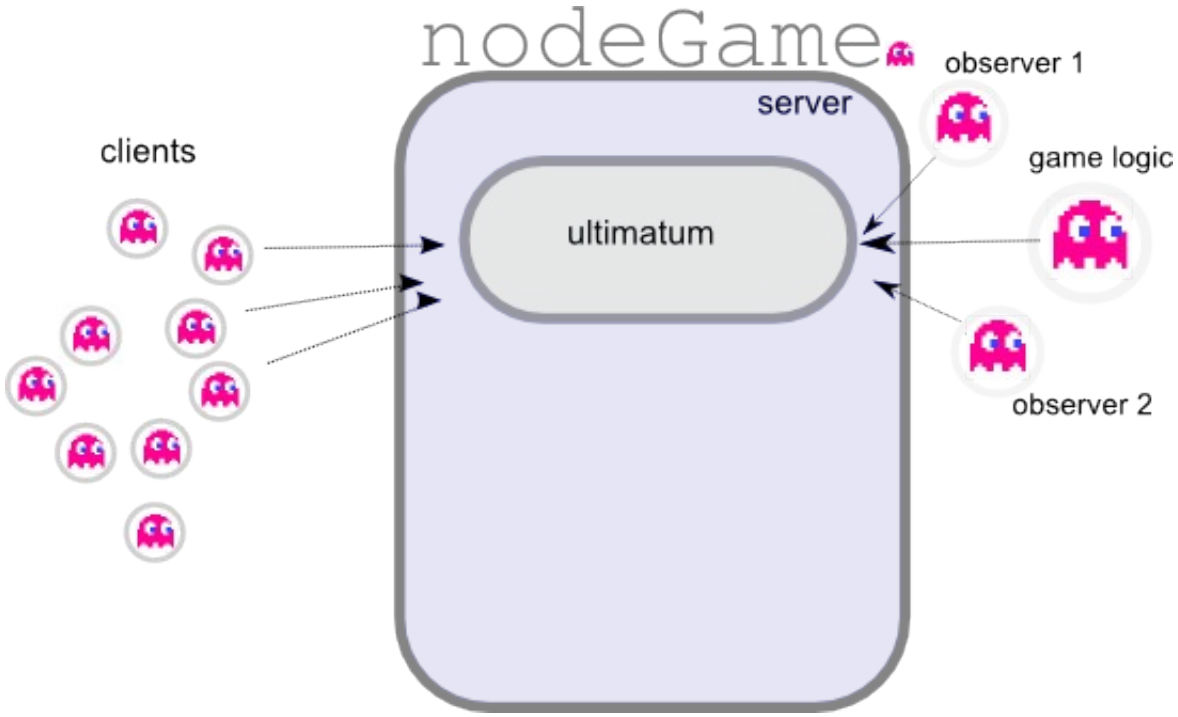
Summary



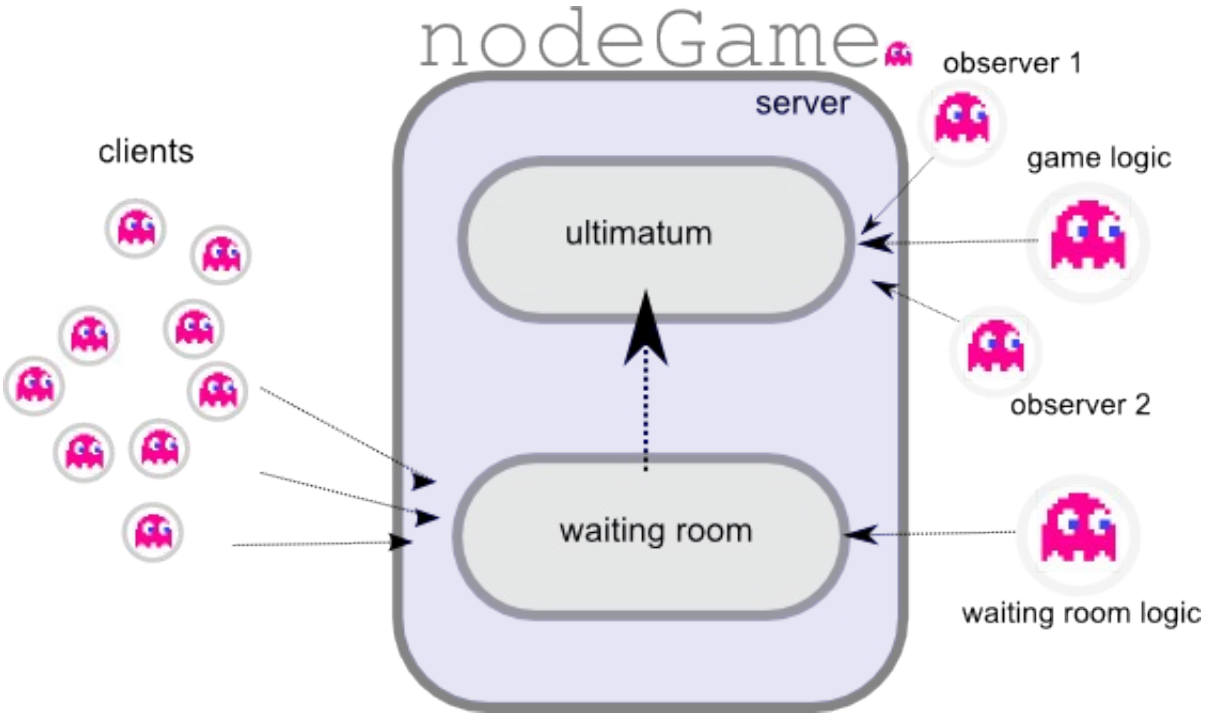
Summary


nodeGame
server
clients
ultimatum
observer 1
game logic
observer 2

Summary


nodeGame
server
clients
ultimatum
observer 1
game logic
observer 2

Summary


nodeGame
server
clients
ultimatum
waiting room
observer 1
game logic
observer 2
waiting room logic

nodeGame: the good parts

- Get access to a **wider subject pool**

- Run **large-scale experiments** (tested in the lab with over one thousand bot-players!)

- Write once, run the same code **online and in the lab**

Integrate in normal web pages for conducting **field studies**

nodeGame: the good parts

- **Open source** and open standard (**HTML5**)
- Server can run multiple games and **waiting rooms** at the same time
- Works on **mobile devices and tablets**
- No installation required on clients
- Integrates smoothly with other libraries and web services, such as Amazon **Mechanical Turk**