# Puzzle de 8

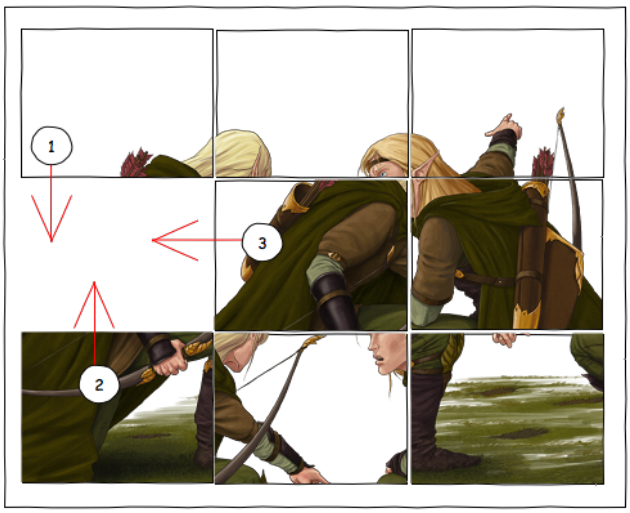
| | |
|---|---|
| **Formação** | **Jogo do Rato** |
| **Local** | Marinha Grande |
| **Formador** | Sérgio Lopes, knitter.is@gmail.com [mailto:knitter.is@gmail.com] |
| **Ficha** | **[RESOLUÇÃO] Jogo: Puzzle de 8** |

Puzzle de 8 é um jogo onde o utilizador deve reconstruir uma imagem que se encontra dividida em 8 blocos dispostos de forma aleatória num tabuleiro de 9 posições. O jogo começa os 8 blocos que formam a imagem dispostos pela ordem correcta e casa vazia no canto inferior esquerdo. Os blocos são baralhados e o jogador deve mover as peças de forma a repor a ordem correcta.

Para completar o jogo apenas é possível mover uma peça de cada vez, sempre para a zona vazia, e apenas na horizontal ou na vertical. Como se pode ver pelas imagens seguintes, as regras são bastante simples.

## Figura 1. Posição Inicial

## Figura 2. Jogadas Possíveis



Na imagem anterior são apresentadas as 3 jogadas possíveis para a disposição das peças:

1. A peça na posição **1:1** pode ser movida para baixo;
2. A peça na posição **3:1** pode ser movida para cima;
3. A peça na posição **2:2** pode ser movida para a esquerda;

### Figura 3. Área de Jogo


# Funcionalidades

O projeto deve fornecer ao utilizador todas as funcionalidades que permitam jogar o jogo com sucesso. Além disso é necessário que seja criada uma opção para baralhar as peças, escolher entre, pelo menos, três imagens. Devem ser apresentadas, ao utilizador, o número de jogadas já feitas e se o jogo se encontra concluído com sucesso.

Como extra pode ser implementada a funcionalidade de upload de novas imagens.

# Recursos

1. jQuery, http://jquery.com
2. jQueryUI, http://jqueryui.com
3. Imagens de utilização livre, http://opengameart.org

# Implementação

Começa por criar um novo projecto no NetBeans IDE, com o nome **puzzle8**, dentro da tua pasta de trabalho. Cria as seguintes pastas dentro do projecto de modo a organizar o trabalho: **classes**, **css**, **images** e **js**. Faz download do ficheiro ZIP com os recursos a usar para este exercício e coloca as pastas das três imagens exemplo na pasta **images** e o ficheiro de jQuery na pasta **js**.

O projecto vai ser dividido em script de arranque (*index.php)*, *script de aspecto (_layout.php*) e classes da aplicação (todas as classes na pasta **classes**).

Olhando para a descrição do jogo facilmente se percebe que são necessários objectos para representar o jogo, o tabuleiro e uma peça que é usada pelo tabuleiro. Se pensármos o problema com mais cuidado vemos que é necessário guardar informação sobre as imagens disponíveis e, no caso da peça, será útil ter uma forma de representar a sua posição dentro do tabuleiro, assim vamos precisar de mais duas classes: uma para representar os dados de uma imagem e outra para representar uma posição num tabuleiro.

### Tabela 1. Descrição das classes

| | |
|---|---|
| Game | Esta será a classe principal e representa um jogo completo, regista o número de jogadas, o tabuleiro e permite ajudar na selecção de imagens e nas acções de iniciar o jogo, resolver o puzzle e outras que surjam. |
| Board | A classe *Board*, como o nome sugere, representa um tabuleiro. Este tabuleiro não é mais que um conjunto de 9 casas (3x3) onde estão organizadas as diversas peças que constituem a imagem. |
| Piece | Cada peça tem uma posição onde se encontra (linha/coluna), uma imagem associada e, para facilitar a verificação das posições, um número. Este número indica a posição correcta da peça quando o puzzle estiver concluído. |
| Image | Esta é uma das duas classes auxiliares e permite registar os dados das imagens. Uma imagem é constituída por um nome (a pasta onde estão os pedaços), uma tamanho (altura e largura), uma posição em branco (correspondente ao peço em falta) e um título que ajude na apresentação ao utilizador. |
| Position | Para terminar, a classe *Position* representa uma posição no tabuleiro, composta pela linha e pela coluna. |

# Ficheiros Auxiliares

Dentro de cada uma das pastas onde estão colocadas as imagens cria um ficheiro chamado *details.info* e coloca nele os dados correspondentes a cada conjunto de imagens.

| Nome da Pasta | Dados |
|---|---|
| sleeping | title: Sleeping Cat<br>name: sleeping<br>empty: 20<br>width: 640<br>height: 360 |
| small | title: Small Cat<br>name: small<br>empty: 22<br>width: 640<br>height: 400 |
| wizard | title: Female Wizard<br>name: wizard<br>empty: 02<br>width: 640<br>height: 400 |

# Criar as Classes

Cria os ficheiros necessários para as classes decritas acima, tendo o cuidado de colocar os ficheiros na pasta correcta. Abre cada um dos ficheiros criados e segue a implementação sugerida de seguinda. Tem também atenção que, em alguns casos, será necessário fazer as inclusões correctas de modo a que o PHP encontre as classes usadas uma vez que não estamos a usar o mecanismo de *autoload*.

**Classe Position.** De longe uma das classes mais simples de implementar, esta classe tem apenas dois atributos privados com o nome *row* e *column*. Usando o atalho do NetBeans IDE (ctrl + I) cria um construtor e os métodos *get* necessários para aceder aos atributos.

**Classe Image.** Esta classe terá de conter atributos que permitam guardar todos os dados que estão nos ficheiros *details.info*, por isso será necessário criar atributos privados com os nomes: *name*, *title*, *emptyRow*, *emptyColumn*, *width*, *height*.

Para facilitar a utilização o valor da casa vazia foi dividido em duas partes que terão de ser criadas durante a leitura dos dados do ficheiro. Veremos como mais abaixo. Tal como na classe anterior é neessário criar um construtor e métodos *get*.

**Classe Piece.** Uma peça, como foi descrito antes, tem um ficheiro correspondente ao pedaço de imagem (file), uma posição (position) e um número (number). Cria os atributos privados, o construtor e os métodos *get* necessários para esta classe.

**Classe Board.** Para esta classe usa o código sugerido de seguinda e termina de implementar os métodos que estão assinalados.

```
private $pieces;
private $playing;
```

```
function __construct(Image $image) {
    $this->playing = false;

    $skipRow = $image->getEmptyRow();
    $skipColumn = $image->getEmptyColumn();

    $this->pieces = array();
    for ($number = 1, $row = 0; $row < 3; $row++) {
        for ($column = 0; $column < 3; $column++, $number++) {
            if ($row == $skipRow && $column == $skipColumn) {
                $this->pieces[$row][$column] = new Piece(new Position($row, $column)
                        , null, 0);
            } else {
                $file = 'images/' . $image->getName() . '/' . $row . $column . '.png';
                $this->pieces[$row][$column] = new Piece(new Position($row, $column)
                        , $file, $number);
            }
        }
    }
}

public function getPiece($row, $column) { /*IMPLEMENTAR*/ }

public function isPlaying() {
    return $this->playing;
}

public function shuffle() {
    //IMPLEMENTAR PARTE EM FALTA

    $this->playing = true;
}

public function isComplete() { /*IMPLEMENTAR*/ }

public function move($row, $column) {
    if (isset($this->pieces[$row][$column])) {
        $piece = $this->pieces[$row][$column];
        $oRow = $row;
        $oColumn = $column;
        $swap = false;

        if (isset($this->pieces[$row - 1][$column])
                && $this->pieces[$row - 1][$column]->getNumber() == 0) {
            $oRow = $row - 1;
            $swap = true;
        } else if (isset($this->pieces[$row + 1][$column])
                && $this->pieces[$row + 1][$column]->getNumber() == 0) {
            $oRow = $row + 1;
            $swap = true;
        } else if (isset($this->pieces[$row][$column - 1])
                && $this->pieces[$row][$column - 1]->getNumber() == 0) {
            $oColumn = $column - 1;
            $swap = true;
        } else if (isset($this->pieces[$row][$column + 1])
                && $this->pieces[$row][$column + 1]->getNumber() == 0) {
            $oColumn = $column + 1;
            $swap = true;
        }

        if ($swap) {
            $other = $this->pieces[$oRow][$oColumn];
            $other->setPosition(new Position($row, $column));
            $this->pieces[$row][$column] = $other;

            $piece->setPosition(new Position($oRow, $oColumn));
```

```
            $this->pieces[$oRow][$oColumn] = $piece;

            return (object) array('row' => $oRow, 'column' => $oColumn);
        }
    }

    return null;
}
```

**Classe Game.** A classe que representa um jogo tem a seguinte estrutura:

```
protected $board;
    protected $moveCount;
    protected $images;
    protected $selectedImage;

function __construct() {
    $this->images = array();

    $basePath = realpath(dirname(__FILE__) . '/../images');
    $dh = opendir($basePath);
    if ($dh) {
        while (($imageDir = readdir()) !== false) {
            if ($imageDir != '.' && $imageDir != '..' && is_dir($basePath . '/'
                    . $imageDir)) {
                $detailsFile = $basePath . '/' . $imageDir . '/details.info';

                $temp = array();
                if (is_file($detailsFile) && ($fh = fopen($detailsFile, 'r'))) {
                    while (($line = fgets($fh)) != false) {
                        $pieces = explode(':', $line);
                        if (count($pieces) == 2) {
                            $temp[trim($pieces[0])] = trim($pieces[1]);
                        }
                    }
                    fclose($fh);
                    $this->images[$temp['name']] = new Image($temp['name'], $temp['title']
                             , $temp['empty']{0}, $temp['empty']{1}, intval($temp['width'])
                             , intval($temp['height']));
                }
            }
        }
        closedir($dh);
    }
}

public function getBoard() {
    return $this->board;
}

public function isPlaying() {
    return $this->board != null && $this->board->isPlaying();
}

public function exec() {
    if (isset($_SESSION['game'])) {
        $this->board = unserialize($_SESSION['game']);
        $this->selectedImage = unserialize($_SESSION['selectedImage']);

        $this->moveCount = $_SESSION['moves'];
    }

    if (!empty($_REQUEST)) {
        if (isset($_REQUEST['suffle'])) {
            $this->suffle();
        } else if ($this->isPlaying() && isset($_REQUEST['play'])
```

```
                && isset($_REQUEST['origin'])) {
            $coordinates = explode(',', $_REQUEST['origin']);
            if (count($coordinates) == 2) {
                $this->play($coordinates[0], $coordinates[1]
                    , isset($_REQUEST['encode']) ? : false);
            }
        }
    }

    $this->show();
}

private function suffle() {
    $this->moveCount = 0;

    $name = $_REQUEST['image'];
    $this->selectedImage = (object) array(
                'name' => $name,
                'width' => $this->images[$name]->getWidth(),
                'height' => $this->images[$name]->getHeight()
    );

    $this->board = new Board($this->images[$this->selectedImage->name]);
    $this->board->shuffle();

    $_SESSION['game'] = serialize($this->board);
    $_SESSION['selectedImage'] = serialize($this->selectedImage);
    $_SESSION['moves'] = 0;
}

private function play($row, $column, $encode = false) {
    $result = $this->board->move($row, $column);

    $_SESSION['game'] = serialize($this->board);
    if ($result) {
        $this->moveCount++;
        $_SESSION['moves'] = $this->moveCount;
    }

    if ($encode) {
        if ($result) {
            echo json_encode(array(
                'moved' => 1,
                'to' => $result,
                'moves' => $this->moveCount
            ));
        } else {
            echo json_encode(array('moved' => 0));
        }

        exit();
    }
}

private function show() {
    ob_start();
    include 'layout.php';
    echo ob_get_clean();
}
```

Nesta classe não é necessário acrescentar nada bastando transcrever correctamente o código.

# Interface

Cria um ficheiros chamado *styles.css* e coloca nele o seguinte código CSS:

```
/*RESET*/
html, body, div, span, table, tbody, tfoot, thead, tr, th, td {
    margin:0;
    padding:0;
    border:0;
    outline:0;
    font-size:100%;
    vertical-align:baseline;
    background:transparent;
}

body {
    line-height: 1;
}

table {
    border-collapse:collapse;
    border-spacing:0;
}

input, select {
    vertical-align:middle;
}
/*END RESET*/
```

O código anterior permite remover margens e espaços para termos elementos mais uniformes ao longo dos diversos browsers. De seguinda adiciona o CSS que permite dar algum corpo e aspecto ao jogo:

```
body {
    font-size: 12pt;
    font-family: Geneva, Helvetica, Verdana, Arial, sans-serif;
}

#wrapper {
    width: 680px;
    margin: 0 auto;
}

#header { height: 80px; }

#tools {}
#playfield {}

#puzzle {
    margin: 2em auto;
    padding: 0;
    width: 640px;
    min-height: 420px;
}

#nopuzzle {
    text-align: left;
    padding-top: 2em;
    margin: 0 0 5em 0;
}

#gameinfo { }

#footer {
    margin: 2em 0;
```

```
    padding: 0;
    text-align: center;
}

table {
    margin: 0 auto;
    background: #FFF;
    border: 1px solid #999;
}

td {
    background-color: #FFF;
    text-align: center;
    vertical-align: middle;
    padding: 1px;
}

td img { vertical-align: middle; }

input, select {
    border: 1px solid #0A0;
    background-color: #FFF;
}

input { margin-left: 3em; }

form { display: inline; }
```

Cria o ficheiro *index.php* se ainda não o tens criado, se já existir apaga o seu conteúdo, cria também o ficheiro *layout.php*. Ambos devem ser criados na raiz do projecto de modo a estarem facilmente acessíveis ao browser.

No ficheiro *layout.php* coloca o seguinte código:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <link rel="stylesheet" type="text/css" href="css/styles.css" />

        <script type="text/javascript" src="js/jquery-1.7.1.min.js"></script>
        <script type="text/javascript" src="js/script.js"></script>

        <title>Puzzle de 8</title>
    </head>
    <body>
        <div id="wrapper">
            <div id="header"></div>
            <div id="tools">
                <?php if (count($this->images)) { ?>
                    <form action="index.php" method="post">
                        <select name="image" id="image">
                            <!-- IMPLEMENTAR GERAÇÃO DE OPÇÕES -->
                        </select>
                        <input type="submit" value="Baralhar" name="suffle" />
                    </form>

                    <form action="index.php" method="post">
                        <input type="submit" name="solve" value="Resolver" />
                    </form>
                <?php } else { ?>
                    Não existem imagens disponíveis para jogar.
                <?php } ?>
            </div>
            <div id="playfield">
```

```
                <?php if ($this->isPlaying()) { ?>
                    <div id="puzzle">
                        <table>
                            <?php for ($row = 0; $row < 3; $row++) { ?>
                                <tr>
                                    <?php
                                    for ($column = 0; $column < 3; $column++) {
                                        $piece = $this->board->getPiece($row, $column);
                                        ?>
                                        <td id="<?php echo 'P', $row, $column; ?>">
                                            <?php
                                            if ($piece->getNumber() != 0) {
                                                ?>
                                            <img id="<?php echo 'I', $row, $column; ?>"
                                                 src="<?php echo $piece->getFile(); ?>"
                                                 onclick="move(this)"
                                                 data-row="<?php echo $row; ?>"
                                                 data-column="<?php echo $column; ?>" />
                                            <?php } ?>
                                        </td>
                                    <?php } ?>
                                </tr>
                            <?php } ?>
                        </table>
                    </div>
                <?php } else { ?>
                    <div id="nopuzzle">
                        Escolha uma imagem e baralhe para iniciar o jogo.
                    </div>
                <?php } ?>
                <div id="gameinfo">
                    <?php if ($this->isPlaying()) { ?>
                        Número de Jogadas:
                        <strong id="movecount">
                            <?php echo $this->moveCount; ?>
                        </strong>
                    <?php } ?>
                </div>
            </div>
            <div id="footer">
                &copy; <?php echo date('Y'); ?> Jogo do Rato
            </div>
        </div>
    </body>
</html>
```

Finalmente cria o ficheiro *script.js* na pasta **js** e completa o código seguinte (usa como base a documentação em http://api.jquery.com/jQuery.ajax/):

```
function move(elem) {
    var me = $(elem);

    $.ajax({
        url: /*IMPLEMENTAR*/,
        dataType: /*IMPLEMENTAR*/,
        type: /*IMPLEMENTAR*/,
        data: {
            play: 'yes',
            origin: me.data('row') + ',' + me.data('column'),
            encode: 1
        },
        success: function(response) {
            if(response.moved) {
                me.data('row', response.to.row)
                .data('column', response.to.column)
                .appendTo($('#P' + response.to.row + response.to.column));
```

```
                $('#movecount').text(response.moves);
            }
        }
    });
}
```

# Extras

Completa o HTML e CSS de modo a melhorar o aspecto do jogo e da interface criada.

Implementa o upload de novas imagens de modo a permitir novos puzzles.