Introdução às Tecnologias *Web*

Guião da Aula Teórico-Prática sobre JavaScript (Parte 2)

Antes da aula

Realizar as seguintes lições do tutorial Learn JavaScript do Codecademy: Scope, Arrays, Loops, e Objects.

Durante a aula

Comece por descarregar o ficheiro de material de apoio à aula teórico-prática, o qual contém `index.html`, `apresentacao.css` (na pasta de `estilos`), e `comportamento.js` (em `scripts`). O documento HTML referencia o ficheiro de CSS, bem como o ficheiro com código JavaScript, sendo este último carregado e automaticamente executado pelo *browser*.

O objetivo da aula é completar o desenvolvimento de um **jogo de adivinha o número**. Tanto o ficheiro HTML como o CSS podem ser considerados finalizados, sendo todos os exercícios realizados no código JavaScript, que já tem uma base de trabalho mas ainda não está pronto.

Exercício 1: Analise a estrutura e conteúdo do documento HTML num editor de código, em particular: a) a secção de configuração do jogo, com uma tabela que guarda o máximo de tentativas para adivinhar o número, bem como o intervalo de valores admissíveis para o número aleatório gerado pelo computador; e b) a secção de tentativas realizadas pelo utilizador durante um jogo, com uma tabela que indica o número e valor da tentativa, e se acertou ou não.

Exercício 2: Ainda no `index.html`, repare nos identificadores dos elementos HTML, incluindo secções, células da tabela de configuração do jogo, e botões. Estes últimos são usados para invocar funções JavaScript, uma por cada botão, indicadas nos atributos `onclick`, nomeadamente para alterar as opções de configuração, gerar um número aleatório e iniciar um jogo, e fazer uma nova tentativa de acertar no número. A seguir, abra o ficheiro `comportamento.js` num editor de código, comprove a existência das funções invocadas a partir dos botões, e repare que estão definidas constantes para todos os identificadores de elementos HTML.

As constantes permitem o uso de nomes inteligíveis, tornando o código mais fácil de manter, e poupam trabalho caso haja mudanças nos valores que guardam. Por exemplo, se o nome de um identificador de botão HTML mudar, basta fazer a atualização na declaração da respetiva constante, em vez de em todas as partes do código JavaScript em que o botão fosse usado.

Para o exercício seguinte, é útil saber que existe uma variável global chamada `configuracao`, que guarda a configuração do jogo em três propriedades: `numeroTentativas`, `minimoAleatorio`, e `maximoAleatorio`. Os valores por omissão são, respetivamente, 10, 1, e 100, definidos nas constantes `NUMERO_TENTATIVAS_OMISSAO`, `MINIMO_ALEATORIO_OMISSAO`, e `MAXIMO_ALEATORIO_OMISSAO`. A secção de configuração do jogo no documento HTML deve estar sincronizada com o conteúdo desta variável global, caso contrário o utilizador pode ser induzido em erro.

Exercício 3: Abra o ficheiro HTML no *browser*. Nota alguma incoerência na tabela de configuração do jogo? Falta o valor aleatório máximo! Confirme em `index.html` que a célula que deveria guardar esse valor tem o identificador `tdMaximoAleatorio`, procure a constante correspondente em `comportamento.js`, e corrija o código na função `mostraConfiguracaoJogo()`.

Antes de um jogo começar, o utilizador deve poder carregar nos botões de alteração da configuração. Contudo, se experimentar mudar o número de tentativas permitidas, em vez de ser solicitado um novo valor ao utilizador, aparece simplesmente `undefined`. Isto acontece porque o clique no botão (daí o atributo `onclick` no elemento `button`) chama a função `pedeNumeroTentativas()`, que, por sua vez, invoca `pedeNumeroInteiro()`, mas esta última está… vazia!

Exercício 4: Codifique a função `pedeNumeroInteiro(minimo, maximo, proposito)` que pede ao utilizador um número inteiro entre `minimo` e `maximo`, mostrando o propósito do pedido. O pedido deve ser feito usando `prompt()`, as vezes necessárias até ser obtido um valor válido, sendo mostradas mensagens informativas caso surjam erros. Para converter de texto para um número inteiro usa-se `parseInt()` e para verificar que é mesmo um número pode usar-se `Number.isNaN()`. No ficheiro `comportamento.js` estão mais detalhes sobre `pedeNumeroInteiro()`.

Exercício 5: Com base no código de `pedeNumeroTentativas()`, programe as funções `pedeMinimoAleatorio()` e `pedeMaximoAleatorio()`. Certifique-se que invoca `pedeNumeroInteiro()` com os argumentos adequados. Por exemplo, para redefinir o valor aleatório mínimo, não faz sentido que este possa ser maior do que o atual valor aleatório máximo, e note que existem as constantes `PROPOSITO_MINIMO_ALEATORIO` e `PROPOSITO_MAXIMO_ALEATORIO`. Certifique-se que no final do código de ambas as funções é chamada `mostraConfiguracaoJogo()`, para atualizar a configuração do jogo no documento HTML.

Caso o utilizador carregue no botão de gerar um número aleatório para começar um jogo, deve apenas ficar ativo o botão de fazer uma nova tentativa para adivinhar o número, pois não é possível mudar a configuração do jogo enquanto este estiver a decorrer.

Exercício 6: Experimente iniciar um jogo. Ficou algum botão de mudar a configuração ativo? Ups… Confirme que, em `index.html`, a função invocada após o clique no botão de começar o jogo tem o nome de `iniciaJogo()` e corrija o respetivo código.

Para o exercício final, é conveniente conhecer as funções `Math.random()` e `Math.floor()`. A primeira serve para gerar números aleatórios de vírgula flutuante entre 0 e 1, e a segunda devolve o maior número inteiro menor ou igual ao argumento que lhe for passado. Por exemplo, `Math.floor(0.555 * 100 + 1)` tem como resultado `56`.

Exercício 7: Programe a função `geraNumeroInteiroAleatorio(minimo, maximo)` que gera um número inteiro aleatório entre `minimo` e `maximo`, inclusive.

Neste ponto da aula, deverá ter o jogo totalmente funcional. Há várias funções em `comportamento.js` que não foram abordadas neste guião, tais como `fazTentativa()` e `terminaJogo()`, mas cuja análise é recomendada para enriquecimento da aprendizagem.