Universidade Lusófona de Humanidades e Tecnologias
Fundamentos de Programação
2022/2023
Brena Lima, Bruno Saraiva, Lúcio Studer, Pedro Alves, Rodrigo Correia
v1.0.0

⛺ Jogo das tendas ⛺

Enunciado do projecto prático - Parte 2

Nota prévia

Na parte 2 do projeto devem manter os grupos criados na parte 1. Em casos excepcionais, mediante autorização prévia do professor das práticas e com uma justificação adequada, é possível dividir grupos na parte 2 (ou seja, cada elemento do grupo passa a entregar sozinho). Não é possível formar novos grupos.

Objectivo

Mantém-se o objetivo da parte 1, desenvolver um programa em Kotlin que implemente o jogo das tendas.

Deverão, portanto, continuar o projeto entregue na parte 1.

Este é o enunciado da Segunda Parte.

Objectivos da Segunda Parte

Na segunda parte, graças à matéria nova que foi entretanto leccionada, já será possível implementar o jogo completo.

O terreno inicial passa a mostrar árvores espalhadas em vez de estarem todas encostadas à direita. Essa configuração inicial vai variar consoante o tamanho do terreno mas terá sempre que ser um jogo válido. Ou seja, tem que ser possível terminar o jogo com sucesso, respeitando as regras do mesmo. Por essa razão, haverá um conjunto de ficheiros previamente criados com a configuração inicial para cada tamanho de terreno. É com base nesses ficheiros que irão preencher o terreno.

O jogo também passa a atualizar o terreno, à medida que o jogador vai introduzindo as coordenadas das várias tendas. Ou seja, será portanto um ciclo de jogadas (introduzidos pelo jogador), em que o terreno vai sendo atualizado em cada iteração, terminando quando o jogador coloca tantas tendas quantas o número de árvores e respeita as regras do jogo.

Relembramos que as regras do jogo são:

- As tendas têm que estar num quadrado adjacente (horizontal ou vertical) a uma árvore
- As tendas não se podem tocar - nem mesmo na diagonal
- Os números fora da grelha indicam o número total de tendas nessas linhas ou colunas

Para ajudar nesta implementação, há um conjunto de novas funções obrigatórias para implementar, que acrescem às funções que tinham que implementar na parte 1. Algumas das funções da parte 1 tiveram que sofrer adaptações na parte 2.

O menu inicial assim como as perguntas que levam ao jogo mantêm-se inalterados da parte 1. No entanto, o programa passa a ser um ciclo que só termina quando se escolhe a opção Sair.

Ou seja, após terminar um jogo, deverá voltar ao menu. No entanto, a qualquer momento, o jogador pode desistir do jogo atual, e nesse caso também volta ao menu.

Metodologia Recomendada

Mantém-se a metodologia recomendada na parte 1 do projeto.

Domínio do Problema

A seguir descrevem-se alguns conceitos importantes para a compreensão do enunciado.

Terreno

Mantém-se o terreno definido na parte 1, mas agora existe informação adicional sobre o número de tendas em cada linha/coluna.

Ou seja, o exemplo apresentado para um terreno 6x5 da parte 1, agora já devidamente inicializado, passa a ser:

		2	1	1		3
		A	B	C	D	E
	1	△				△
3	2		△			
	3	△				△
2	4					
	5		△			
2	6				△	

(nota: as cores são meramente ilustrativas, no jogo não serão utilizadas)

Temos aqui 4 conceitos que serão importantes para a compreensão do enunciado:

- **Legenda horizontal**, a cinzento (já existia na parte 1) - Consiste na 2ª linha apresentada, com letras maiúsculas sequenciais, a começarem em 'A'.
- **Legenda vertical**, a cinzento (já existia na parte 1) - Consiste na na 2ª coluna apresentada, com números sequenciais a começarem em 1.
- **Contadores verticais**, a amarelo (novo) - Consiste na 1ª linha apresentada. Indica o número de tendas que devem ser colocadas em cada coluna. No exemplo apresentado, na coluna 'A', devem ser colocadas 2 tendas.

- **Contadores horizontais**, a amarelo (novo) - Consiste na 1ª coluna apresentada. Indica o número de tendas que devem ser colocadas em cada linha. No exemplo apresentado, na linha 2, devem ser colocadas 3 tendas.

Representação do Terreno

O terreno irá ser representado por um array bidimensional de Strings que podem ser null, com as dimensões dependentes daquilo que o jogador escolher nestas perguntas:

```
Quantas linhas?
6
Quantas colunas?
6
```

Usando o exemplo anterior (é apenas um exemplo, tem que funcionar para quaisquer dimensões), teremos o seguinte array (em que l = linha e c = coluna):

l = 0, c = 0	l = 0, c = 1	l = 0, c = 2	l = 0, c = 3	l = 0, c = 4	l = 0, c = 5
l = 1, c = 0	l = 1, c = 1	l = 1, c = 2	l = 1, c = 3	l = 1, c = 4	l = 1, c = 5
l = 2, c = 0	l = 2, c = 1	l = 2, c = 2	l = 2, c = 3	l = 2, c = 4	l = 2, c = 5
l = 3, c = 0	l = 3, c = 1	l = 3, c = 2	l = 3, c = 3	l = 3, c = 4	l = 3, c = 5
l = 4, c = 0	l = 4, c = 1	l = 4, c = 2	l = 4, c = 3	l = 4, c = 4	l = 4, c = 5
l = 5, c = 0	l = 5, c = 1	l = 5, c = 2	l = 5, c = 3	l = 5, c = 4	l = 5, c = 5

É importante que não troquem a linha com a coluna, caso contrário o jogo não funcionará bem. Ou seja, o array deve estar construído como **um array de linhas em que cada linha é um array de colunas**. Por isso, imaginando que a variável `terreno` é o array representado na figura, a célula azul estará em `terreno[1][3]`.

Agora que está definida a estrutura base, é preciso perceber o que vai estar dentro de cada célula/posição. Cada posição pode estar ocupada ou vazia. Se estiver ocupada, pode ser uma árvore ("`A`") ou uma tenda ("`T`"). Se estiver vazia será `null`.

Notem que a posição (1,A) corresponde à posição (0,0) no array. Falaremos mais em detalhe sobre isto mais à frente.

Criação do Terreno

Após o jogador introduzir os dados para a criação do terreno, é necessário criar o tal array, com as dimensões corretas e devidamente preenchido com as árvores. Também deverão ser preenchidos os contadores horizontais e verticais.

Podíamos usar o `random()` para gerar árvores aleatoriamente mas isso poderia produzir jogos impossíveis de resolver. Além disso, ainda tínhamos que gerar contadores horizontais e verticais corretos. Por isso, decidimos facilitar e pré-criar configurações iniciais de terrenos, guardando essas configurações em ficheiros.

Existirá 1 ficheiro por cada dimensão possível (as dimensões possíveis são as mesmas que foram indicadas no enunciado da parte 1), com o seguinte formato:

<numLinhas>x**<numColunas>**.txt

Por exemplo, o ficheiro que guarda a configuração de um terreno 6 por 5, será:
`6x5.txt`

Estes ficheiros estão disponíveis aqui:
https://github.com/ULHT-FP-2022-23/ficheiros-jogo-tendas

Os ficheiros devem ser colocados na raiz do projeto.

Formato dos ficheiros

Cada ficheiro com a configuração inicial do terreno terá várias linhas com significados diferentes. Apresenta-se um exemplo desse ficheiro para o terreno 6x5 apresentado anteriormente.

```
2,1,1,0,3
0,3,0,2,0,2
0,0
0,4
1,1
2,0
2,4
4,1
5,3
```

Linha 1 - Contadores verticais, separados por vírgula. Esta linha terá tantos números quanto o número de colunas do terreno.

Linha 2 - Contadores horizontais, separados por vírgula. Esta linha terá tantos números quanto o número de linhas do terreno

Linhas 3-N - As restantes linhas do ficheiro representam as coordenadas das árvores no terreno. Cada linha corresponde a uma árvore. Por exemplo, podemos ver que existe uma árvore nas coordenadas (0,0), outra nas coordenadas (0,4), etc..

Este formato será essencial para conseguirem implementar as funções `leContadoresDoFicheiro` e `leTerrenoDoFicheiro` (descritas na secção com as funções de implementação obrigatória)

Apresentação do terreno

Uma vez que o terreno inicial passa a ser dinâmico (lido de um ficheiro), a função `criaTerreno` passa a ser responsável por transformar um array bidimensional com a estrutura definida na secção "Representação do terreno" numa String passível de ser escrita no ecrã.

Por exemplo, considerem o seguinte array bidimensional:

```
val terreno = arrayOf(
                    arrayOf(null, "A", null, "A"),
                    arrayOf(null, "T", null, null),
                    arrayOf("A", null, null, null))
```

A função `criaTerreno` deverá transformar este array nesta String (assumindo que não queremos mostrar os contadores nem a legenda):

```
     |   | △ |   | △
     |   | T |   |
     | △ |   |   |
```

Notem que os nulls foram transformados em espaços e os "A" foram transformados em "△". Notem também que há sempre 5 espaços até ao primeiro "|". Isto é uma mudança relativamente à parte 1, motivada pela necessidade de reservar espaço para os contadores horizontais.

Apresenta-se de seguida um exemplo completo (com contadores e legenda):

```
                2       2
     | A | B | C | D | E
1  1 |   |   | T |   |
   2 |   |   | △ |   |
   3 |   |   |   |   |
   4 |   |   |   |   |
1  5 | △ | T |   |   |
   6 |   |   |   |   |
```

Testar a criação do terreno

Se, de cada vez que quisermos testar a construção do terreno, tivermos que iniciar um jogo (introduzindo o tamanho, eventualmente a data de nascimento, etc.), os testes serão chatos e morosos.

Tal como descrito na secção Metodologia, usem a função `main` para testar partes específicas do jogo. Neste caso, queremos testar a criação do terreno, já com árvores e contadores horizontais e verticais, algo deste género.

```
fun main() {

    // banana() << esta é a função auxiliar onde coloquei o código do main

    val contadoresVerticais = leContadoresDoFicheiro(6, 6, true)
    val contadoresHorizontais = leContadoresDoFicheiro(6, 6, false)
    val terreno = leTerrenoDoFicheiro(6, 6)

    print(criaTerreno(terreno,
                      contadoresVerticais, contadoresHorizontais,
                      true, true))
}
```

Lembrem-se apenas que estes testes têm que estar comentados quando submeterem no Drop Project, pois ele está à espera que o `main` inicie o jogo normalmente (ou seja, chame a função `banana`).

Introdução de coordenadas da tenda

Mantêm-se o comportamento para a introdução e validação de coordenadas definido na parte 1, nomeadamente a função `processaCoordenadas`.

Agora, após a validação com sucesso das coordenadas será colocada uma tenda nessas coordenadas, através da função `colocaTenda` (mais informação na secção de funções de implementação obrigatória).

Esta função deve começar por validar:

- Se a posição pretendida está livre
- Se a posição pretendida está adjacente a uma árvore (usando a função `temArvoreAdjacente`).
- Se a posição pretendida não está adjacente a uma tenda (usando a função `temTendaAdjacente`).

Se falhar alguma validação deve apresentar a mensagem “Tenda nao pode ser colocada nestas coordenadas”, voltar a mostrar a mensagem “Coordenadas da tenda? (ex: 1,B)” e ficar à espera que o utilizador novas coordenadas.

Se tudo estiver ok, então a função deve alterar o terreno colocando lá a tenda.

Retirar uma tenda previamente colocada

É possível retirar uma tenda previamente colocada, introduzindo as coordenadas dessa tenda. Ou seja, a função colocaTenda tem também que verificar se já lá está uma tenda e, nesse caso, retirá-la.

Terminar o programa a meio do jogo

A qualquer momento, o jogador pode terminar o jogo usando a palavra “sair” em vez das coordenadas. Desta forma:

```
Coordenadas da tenda? (ex: 1,B)
sair
```

Neste caso, o programa não escreve mais nada e termina. Note-se que não deverão usar a instrução exitProcess(), break ou similar para terminar abruptamente o programa. Deverão usar a instrução return e eventualmente alguma variável boolean de forma a perceberem que têm que terminar o ciclo principal do programa.

Deteção de fim de jogo

Caso o jogador tenha colocado todas as tendas corretamente, o jogo deve escrever a frase “Parabens! Terminou o jogo!” e voltar ao menu principal.

Para perceber se o jogo terminou, devem implementar a função terminouJogo. Essa função vai validar que, para cada árvore, existe uma tenda corretamente posicionada. Deve também verificar que os contadores horizontais e verticais estão corretos.

Funções de Implementação Obrigatória

Tal como na parte 1, existirão funções de implementação obrigatória. Juntas constituem as peças do puzzle que, se fôr bem montado, torna trivial a construção do jogo.

Algumas das funções da parte 1 mantêm-se inalteradas. Outras terão que sofrer adaptações. Finalmente, há diversas funções novas que aparecem na parte 2. As alterações relativamente à parte 1 estão sinalizadas a amarelo.

Nota importante: Estas funções não devem escrever nada no ecrã (ou seja, não devem usar o print()/println()).

Assinatura	**Comportamento**
`fun criaMenu(): String`	Retorna uma string das opções do menu (“Novo jogo” e “Sair”) assim como o “Bem vindo ao jogo das tendas”, tal como é apresentado na secção “Menus do Jogo”.
`fun validaTamanhoMapa(numLinhas: Int, numColunas: Int): Boolean`	Retorna true caso a configuração apresentada seja uma configuração válida, de acordo com as regras apresentadas na secção “Terreno”.
`fun validaDataNascimento(data: String?) : String?`	Retorna null caso a String passada como parâmetro contenha uma data válida. Caso a data seja inválida, deve retornar uma de 2 Strings: ● “Data invalida” ● “Menor de idade nao pode jogar” Verifiquem as validações da data na secção “Validação da idade”
`fun criaLegendaHorizontal(numColunas: Int): String`	Devolve a string da legenda horizontal (as letras de A a Z que estão acima do terreno apresentado), sem espaços à volta. Exemplo: `“A | B | C”`
`fun criaLegendaContadoresHorizontal(contadoresHorizontal: Array<Int?>): String`	Transforma o array de contadores horizontais passada por parâmetro numa String, para ser depois usada na criação do terreno (1ª linha). A String retornada não deve ter espaços à volta. Caso o contador para uma certa coluna esteja a null, deve ser transformado num espaço. Exemplo para o array `[2,1,1,null,3]`: `“2  1  1  3”` Nota: O array `contadoresHorizontais` é obtido através da função `leContadoresDoFicheiro`, descrita a seguir.
`fun leContadoresDoFicheiro(numLines: Int, numColumns: Int, verticais: Boolean): Array<Int?>`	Abre o ficheiro correspondente à configuração passada por parâmetro (`numLinhas` e `numColunas`) e lê a partir dele os respectivos contadores. Note-se que a função pode ser usada para ler os contadores horizontais ou verticais, sendo a diferenciação feita através do parâmetro `verticais`. O array retornado será constituído por inteiros positivos ou null (caso esse contador seja zero).

	Pode assumir que o ficheiro existe. Isto é, não é preciso validar previamente se o ficheiro existe. Note que a função poderá ser chamada com uma dimensão não permitida pelo jogo (ex: 3x4). Desde que exista um ficheiro `3x4.txt`, ela deverá aceitar essa dimensão e retornar o resultado esperado.
`fun leTerrenoDoFicheiro(numLines: Int, numColumns: Int): Array<Array<String?>>`	Cria uma matriz (array bidimensional) com base nas dimensões passadas por parâmetro. De seguida, abre o ficheiro correspondente a essas dimensões e lê a partir dele as coordenadas das árvores. Preenche de seguida as posições correspondentes no array bidimensional com a String “A”. Notem que não preenche com “△”, essa transformação será feita posteriormente na função `criaTerreno`. Pode assumir que o ficheiro existe. Isto é, não é preciso validar previamente se o ficheiro existe. Note que a função poderá ser chamada com uma dimensão não permitida pelo jogo (ex: 3x4). Desde que exista um ficheiro `3x4.txt`, ela deverá aceitar essa dimensão e retornar o resultado esperado.
~~`fun criaTerreno(numLinhas: Int, numColunas: Int, mostraLegendaHorizontal: Boolean, mostraLegendaVertical: Boolean): String`~~ `fun criaTerreno(terreno: Array<Array<String?>>, contadoresVerticais: Array<Int?>?, contadoresHorizontais: Array<Int?>?, mostraLegendaHorizontal: Boolean, mostraLegendaVertical: Boolean): String`	Esta função muda bastante da parte 1 para a parte 2, passando a transformar um terreno pré-criado (o array bidimensional `terreno` que é passado por parâmetro) numa String com a representação “gráfica” do mesmo, pronta a ser escrita no ecrã. Esse parâmetro `terreno` é obtido através da função `leTerrenoDoFicheiro`, explicada anteriormente. Recebe também os parâmetros `contadoresVerticais` e `contadoresHorizontais`, que são obtidos através da função `leContadoresDoFicheiro`, explicada anteriormente. Se esses parâmetros tiverem o valor null, não devem ser mostrados os respetivos contadores. Tal como na parte1, existem ainda dois

	parâmetros que controlam se deve ser mostrada a legenda horizontal e ou vertical. Esta função deve chamar as funções `criaLegendaHorizontal(...)` e `criaLegendaContadoresHorizontal(...)` Importante: A função deve estar preparada para ser executada apenas com 4 ou 5 parâmetros, usando valores por omissão. Os valores por omissão para o 4º e 5º parâmetros são true e true respetivamente.
`fun processaCoordenadas(coordenadasStr: String?, numLines: Int, numColumns: Int): Pair<Int,Int>?`	Verifica se as coordenadas introduzidas pelo jogador são válidas, tendo em conta a configuração do terreno. Nesta parte, além de validar, passa a retornar um Pair com as coordenadas no array bidimensional. Por exemplo, se `coordenadasStr` tiver o valor "2,C", deve retornar o Pair (1,2). Caso as coordenadas sejam inválidas, deve retornar null.
`fun temArvoreAdjacente(terreno: Array<Array<String?>>, coords: Pair<Int, Int>) : Boolean`	Esta função verifica se existe uma árvore adjacente à posição representada pelo parâmetro `coords`, tendo em conta o conteúdo do array bidimensional `terreno`. Neste caso, considera-se adjacente uma posição que esteja encostada a uma árvore **na horizontal ou na vertical**. Pode assumir que as coordenadas estarão sempre dentro do terreno, numa posição vazia.
`fun temTendaAdjacente(terreno: Array<Array<String?>>, coords: Pair<Int, Int>) : Boolean`	Esta função verifica se existe uma tenda adjacente à posição representada pelo parâmetro `coords`, tendo em conta o conteúdo do array bidimensional `terreno`. Neste caso, considera-se adjacente uma posição que esteja encostada a uma tenda **na horizontal, vertical ou diagonal**. Pode assumir que as coordenadas estarão sempre dentro do terreno, numa posição vazia.
`fun contaTendasColuna(terreno:`	Retorna o número de tendas na coluna

`Array<Array<String?>>, coluna: Int): Int`	passada por parâmetro. A numeração das colunas começa em zero. Caso não exista a coluna indicada, deve retornar zero.
`fun contaTendasLinha(terreno: Array<Array<String?>>, linha: Int): Int`	Retorna o número de tendas na linha passada por parâmetro. A numeração das linhas começa em zero. Caso não exista a linha indicada, deve retornar zero.
`fun colocaTenda(terreno: Array<Array<String?>>, coords: Pair<Int, Int>): Boolean`	Coloca uma tenda no `terreno`, nas coordenadas indicadas pelo parâmetro `coords`. Antes de colocar a tenda, verifica que a jogada é válida - caso não seja retorna false. Se fôr válida, retorna true. Ver mais informação na secção “Introdução de coordenadas da tenda”.
`fun terminouJogo(terreno: Array<Array<String?>>, contadoresVerticais: Array<Int?>, contadoresHorizontais: Array<Int?>): Boolean`	Retorna true, caso tenham sido colocadas todas as tendas (uma por árvore) e os contadores horizontais e verticais batam certo. Caso contrário, retorna false.

Podem e devem implementar funções adicionais se isso ajudar a organizar o vosso código. Funções com demasiadas linhas de código (incluindo o main()) levarão a penalizações na componente de qualidade de código.

Lembrem-se que a implementação de certos comportamentos em funções adicionais (em vez do main) pode ser mais fácil pois podem usar o `return` para terminar imediatamente o fluxo de execução.

Entrega e Avaliação

Artefactos a Entregar

A entrega deve ser feita através do Drop Project usando um ficheiro comprimido (formato .zip) dentro do qual se encontrem:

- Uma pasta com o nome "src" com todo o código necessário para compilar e executar o projeto.
- Um ficheiro de texto (chamado AUTHORS.txt) contendo os nomes e números de aluno).

Formatos de Entrega

O ficheiro .zip deve seguir a estrutura que se indica de seguida:

```
AUTHORS.txt (contém linhas NUMERO_ALUNO;NOME_ALUNO, uma por aluno do grupo)

+ src
|--- Main.kt
|--- ...   (outros ficheiros kotlin do projeto)
```

Nota: Não precisam de incluir os ficheiros 6x6.txt, etc…

Restrições Técnicas

Na parte 2 deixam de existir algumas das restrições técnicas da parte 1. É proibida a utilização de estruturas dinâmicas como listas, maps e sets. Também não é permitido criarem classes. Podem usar constantes globais mas não variáveis globais. Por serem uma má prática de programação (aliás, nem sequer se fala disso nas aulas), também é proibido utilizar o `break`, o `continue` e o `exitProcess()`. Tudo o resto é permitido - no entanto, salientamos que **o projeto pode ser implementado apenas com as instruções ensinadas nas aulas teóricas**. Não serão esclarecidas dúvidas sobre código que use instruções não ensinadas nas aulas.

A versão do Kotlin ensinada nas aulas e que é oficialmente suportada pelo Drop Project é a versão 1.7.X.

Como Entregar

O projeto será entregue via Drop Project (DP), através do link:

https://deisi.ulusofona.pt/drop-project/upload/fp-projeto-22-23-p2

Não serão aceites entregas por outra via.

Os alunos são incentivados a testar o projeto no DP à medida que o vão implementando. Podem e devem fazer quantas submissões acharem necessárias. Para efeitos de avaliação será considerada a melhor submissão feita dentro do prazo.

Prazos de Entrega

A data limite de entrega é o dia **8 de Janeiro de 2022**, pelas **23h00** (hora de Lisboa, Portugal).

Os alunos que entreguem depois do prazo, até dia 9 de Janeiro, às 08h00, **terão uma penalização de 5 valores** na nota final desta parte do projeto.

Não serão aceites entregas após dia 9 de Janeiro às 08h00. Nesse caso os alunos terão nota zero no projeto, **reprovando na componente prática da disciplina (1ª época)**.

As defesas decorrerão presencialmente entre os dias 9 e 13 de Janeiro e serão obrigatórias.

Avaliação

Mantêm-se as regras da primeira parte, às quais se acrescenta:

- Na **nota final** do projeto existe a nota mínima de 9.5 (nove e meio) valores.
- Há um conjunto mínimo de funcionalidades implementadas para que o projecto possa ser defendido com sucesso. Ou seja, alguns dos testes vão ser **obrigatórios**.
    - Estes testes estarão identificados com o sufixo "_OBG" (ex: test_10_criaTerrenoCom3Linhas_OBG)
    - Os alunos que entreguem projectos que não passem **todos os testes obrigatórios** serão reprovados em primeira época.

Cotações da Segunda Parte

A avaliação do projecto será feita através dos seguintes tópicos:

Tópico	Descrição	Pontuação (0..20)
Qualidade de código	O código cumpre as boas práticas de programação ensinadas nas aulas (nomes apropriados para as variáveis e funções em camelCase, utilização do val e do var, código bem indentado, funções que não sejam demasiado grandes, evitar usar o !!, etc.).	3
Testes automáticos	Será aplicada uma bateria de testes automáticos cujo relatório poderá ser consultado após cada submissão. Quantos mais testes passarem, melhor nota terão nesta componente.	14
Avaliação manual da aplicação	Os professores farão testes adicionais assim como inspeção de código para avaliar esta componente	3

Cópias

Trabalhos que sejam identificados como cópias serão anulados e os alunos que os submetam terão nota zero em ambas as partes do projeto (quer tenham copiado, quer tenham deixado copiar). Para evitar situações deste género, recomendamos aos alunos que nunca partilhem ou mostrem o código do seu projeto a pessoas fora do grupo de trabalho.

A decisão sobre se um trabalho é uma cópia cabe exclusivamente aos docentes da unidade curricular.

Outras Informações Relevantes

Todas as descritas no enunciado da parte 1.

Exemplo de interação

```
Bem vindo ao jogo das tendas

1 - Novo jogo
0 - Sair

1
Quantas linhas?
10
Quantas colunas?
10
Qual a sua data de nascimento? (dd-mm-yyyy)
10-10-1990
            2           3   1   2   3       3
      | A | B | C | D | E | F | G | H | I | J
   1 |   |   |   | △ |   |   | △ |   |   |
   2 |   |   |   |   |   |   |   | △ | △ |
2  3 |   | △ |   |   |   |   |   |   |   |
   4 | △ |   |   |   |   |   |   |   |   |
   5 | △ |   |   |   | △ |   |   | △ |   | △
   6 |   | △ |   |   | △ |   |   | △ |   | △
2  7 |   |   |   |   |   |   |   |   |   |
   8 | △ |   |   |   |   |   |   |   |   |
3  9 |   |   | △ |   |   |   |   | △ |   |
2 10 |   |   |   |   | △ | △ |   |   |   | △
Coordenadas da tenda? (ex: 1,B)
1,A
Tenda nao pode ser colocada nestas coordenadas
Coordenadas da tenda? (ex: 1,B)
3,A
            2           3   1   2   3       3
      | A | B | C | D | E | F | G | H | I | J
   1 |   |   |   | △ |   |   | △ |   |   |
   2 |   |   |   |   |   |   |   | △ | △ |
2  3 | T | △ |   |   |   |   |   |   |   |
   4 | △ |   |   |   |   |   |   |   |   |
   5 | △ |   |   |   | △ |   |   | △ |   | △
   6 |   | △ |   |   | △ |   |   | △ |   | △
2  7 |   |   |   |   |   |   |   |   |   |
   8 | △ |   |   |   |   |   |   |   |   |
3  9 |   |   | △ |   |   |   |   | △ |   |
2 10 |   |   |   |   | △ | △ |   |   |   | △
Coordenadas da tenda? (ex: 1,B)
1,E
            2           3   1   2   3       3
      | A | B | C | D | E | F | G | H | I | J
   1 |   |   |   | △ | T |   | △ |   |   |
   2 |   |   |   |   |   |   |   | △ | △ |
2  3 | T | △ |   |   |   |   |   |   |   |
```

```
   4 | △ |   |   |   |   |   |   |   |   |
   5 | △ |   |   |   | △ |   |   | △ |   | △
   6 |   | △ |   |   | △ |   |   | △ |   | △
2  7 |   |   |   |   |   |   |   |   |   |
   8 | △ |   |   |   |   |   |   |   |   |
3  9 |   |   | △ |   |   |   |   | △ |   |
2 10 |   |   |   |   | △ | △ |   |   |   | △
Coordenadas da tenda? (ex: 1,B)
(...)
Coordenadas da tenda? (ex: 1,B)
10,G
Parabens! Terminou o jogo!

Bem vindo ao jogo das tendas

1 - Novo jogo
0 - Sair

0
```