Ciências ULisboa | Informática

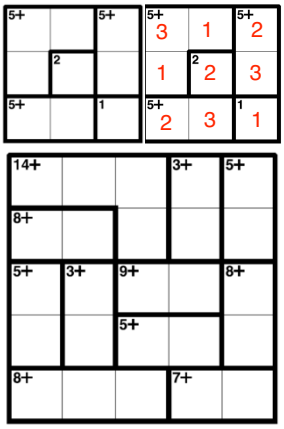
Introdução à Programação

Licenciatura em Engenharia Informática

Trabalho: 2ª parte

2024/2025

Sumdoku



O trabalho de programação que vos é proposto em IP é sobre um jogo que passa por resolver puzzles com números, semelhante ao Sudoku e batizado, para efeitos do trabalho, *Sumdoku*.

Neste jogo, o objetivo é preencher uma grelha quadrada com dígitos de tal forma que, cada linha e cada coluna contenham todos os dígitos de *1* a *n*, onde *n* é o tamanho da grelha (um valor entre 3 e 9). Adicionalmente, as casas estão agrupadas em grupos disjuntos e a soma tem de dar o valor indicado para o grupo. Cada puzzle tem apenas uma solução.

Ao lado apresenta-se um exemplo de um puzzle de tamanho 3 com 5 grupos e a respetiva solução e um puzzle de tamanho 5 com 11 grupos.

Em que consiste o trabalho?

A vossa tarefa é desenvolver de novo código Java que permita jogar o *Sumdoku*. A solução pretendida nesta fase já não faz a leitura de puzzles, mas dá a possibilidade ao utilizador de jogar vários puzzles diferentes. Adicionalmente, a solução pretendida nesta fase está sujeita a um conjunto de restrições diferentes. Estas restrições têm como objetivo garantir que praticam o desenvolvimento de soluções que envolvem utilizar vetores e programar classes que definem tipos. A vossa solução vai permitir jogar o *Sumdoku* tanto com uma interface gráfica (fornecida) como com uma interface textual.

Concretamente, a vossa tarefa é programar:

- Uma classe **`SumdokuGrid`** cujos objetos representam grelhas de *Sumdoku*, cujas casas podem estar ou não preenchidas.
- Uma classe **`SumdokuPuzzle`** cujos objetos representam puzzles de *Sumdoku* construídos à custa de uma matriz definindo a que grupo pertence cada casa da quadrícula e um vetor que define quanto somam as casas de cada grupo.
- Uma classe **`SumdokuRandomPuzzle`** cujos objetos representam geradores de puzzles *Sumdoku*
- Uma classe **`SumdokuTxt`** que implementa uma interface textual do jogo

A vossa solução deve usar a nova versão da classe **`SumdokuSolver`** fornecida nesta fase. Tal como na fase anterior, devem usar apenas o que está disponível na versão 8 do Java e foi lecionado na disciplina (isto significa que, por exemplo, não podem usar listas ou outras coleções). De seguida descreve-se em mais detalhe os requisitos específicos de cada classe.

`SumdokuGrid`. Esta é a classe que foi fornecida na primeira fase do trabalho e que agora vão ter de implementar. Os objetos desta classe são mutáveis e representam grelhas quadradas de um determinado tamanho, dado na construção. As casas destas grelhas podem não estar preenchidas ou ter um número entre 1 e o tamanho da grelha. Para mais detalhes sobre os construtores e métodos a implementar nesta classe devem consultar a documentação fornecida na primeira fase.

SumdokuPuzzle. Os objetos desta classe representam puzzles de *Sumdoku*. A classe deve incluir os seguintes construtores e métodos públicos:

- `public static boolean definesPuzzle(int[][] groupMembership, int[] groupsValues)` que é uma função que verifica se `groups` e `groupsValues` fornecidos como argumento definem um puzzle.

  Para isto acontecer `groupMembership` deve ser uma matriz definindo a que grupo pertence cada casa da quadrícula (com grupos numerados a partir de `0`) e `groupsValues` deve ser um vetor que define quanto somam as casas de cada grupo. Mais especificamente a função deve verificar que:

  1. `groupMembership` é uma matriz quadrada *NxN* com *N* entre `3` e `9`
  2. `groupsValues` é um vetor cujo tamanho é pelo menos 1 e não superior a *NxN* e que está preenchido com valores entre `1` e $(N^3+N^2)/2$
  3. todos as entradas de `groupMembership` estão entre `0` e `groupsValues.length-1`
  4. existe pelo menos uma entrada de `groupMembership` com o valor `g`, para todo o `0 ≤ g < groupsValues.length`
  5. o puzzle admite uma e uma só solução
- `public SumdokuPuzzle(int[][] groupMembership, int[] groupsValues)`, que constrói um puzzle baseado nos argumentos dados, assumindo estes definem efetivamente um puzzle
- `public int size()`, que dá o tamanho da grelha do puzzle
- `public int numberOfGroups()`, que dá o número de grupos do puzzle
- `public int groupNumber(int col, int row)`, que dá o número do grupo da casa na linha e coluna dada, assumindo que `1 ≤ col <= size() && 1 ≤ row <= size()`; o valor retornado é um valor entre `1` e `numberOfGroups()`
- `public int valueGroup(int group)`, que dá quanto somam as casas do grupo dado, assumindo que `1 ≤ group ≤ numberOfGroups()`
- `public boolean isSolvedBy(SumdokuGrid playedGrid)`, que verifica se `playedGrid` é a solução do puzzle, assumindo que `playedGrid` não é null e tem o tamanho certo.
- `public boolean isPartiallySolvedBy(SumdokuGrid playedGrid)`, que verifica se as casas já preenchidas de `playedGrid` têm os valores da solução do puzzle, assumindo que `playedGrid` não é null e tem o tamanho certo.
- `public String cluesToString()`, que dá uma representação textual das pistas do puzzle como exemplificado abaixo para o exemplo do puzzle de tamanho 3 na página anterior

```
 1 1 2
 1 3 2
 4 4 5
G1 = 5 G2 = 5 G3 = 2 G4 = 5 G5 = 1
```

- `public String toString()`, que dá uma representação textual do puzzle (à sua escolha)

SumdokuRandomPuzzle. Os objetos desta classe representam geradores aleatórios de puzzles de *Sumdoku* para grelhas de um determinado tamanho, dado na construção.

Uma vez que a verdadeira geração de puzzles é um problema complicado, a classe deve implementar esta geração à custa de ter, para cada tamanho da grelha, um conjunto de puzzles *built-in* e limitar-se a escolher aleatoriamente a ordem pela qual os fornece. Para efeitos do trabalho deve pelo menos ter os dois puzzles de tamanho 3 e o puzzle de tamanho 5 usados nos testes (ver a classe **SumdokuTest**).

A classe deve incluir os seguintes construtores e métodos públicos:

- `public RandomSumdokuPuzzle(int size)`, que constrói um gerador de puzzles com grelha do tamanho dado, que se assume ser um valor entre 3 e 9
- `public int size()`, que indica o tamanho dos puzzles gerados por este gerador
- `public boolean hasNextPuzzle()`, que indica se há mais algum puzzle para fornecer
- `public SumdokuPuzzle nextPuzzle()`, que dá o próximo puzzle, assumindo que este existe

SumdokuTxt. Classe com um método `main`, que deverá obter o tamanho da grelha do puzzle a partir dos argumentos da consola e permitir que o jogador jogue um ou mais puzzles com esse tamanho. Deve criar um objeto **SumdokuRandomPuzzle** que gera puzzles desse tamanho e usar este objeto para obter mais puzzles enquanto isso for possível e o jogador estiver interessado em continuar a jogar. Além do `main`

deve definir nesta classe um método `static void play(SumdokuPuzzle puzzle, int maxAttempts, Scanner reader)` que, usando o `reader` fornecido, trata da interação com o jogador durante o jogo do `puzzle`, limitado por o número de tentativas `maxAttempts`.

Como posso testar o meu código? Uma alternativa é usar a classe **SumdokuTest** fornecida, a qual exercita os vários métodos das classes a programar em alguns cenários. Pode ainda exercitar o seu código através da execução da classe **SumdokuTxt**.

É fornecida ainda uma classe **SumdokuGUI**. Esta classe, juntamente com o resto dos recursos fornecidos, permite jogar o *Sumdoku* com uma interface gráfica (GUI). No final do enunciado é dito como. Só deve procurar correr esta interface quando tiver o resto praticamente terminado, a passar todos os testes da **SumdokuTxt** e já for possível jogar usando a interface textual.

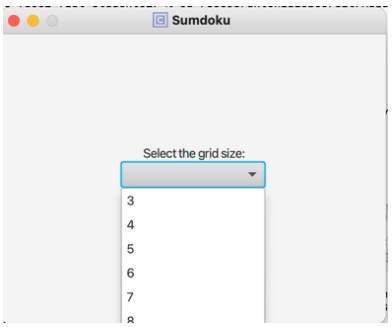
Exemplos ilustrativos da Interface Textual e Gráfica

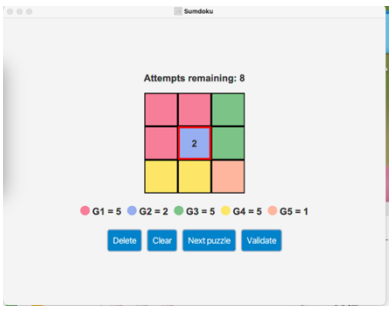
Para facilitar a compreensão dos exemplos, apresenta-se o input a vermelho e omitiram-se algumas coisas (representado por ...).

```
$ java SumdokuTxt 3

Bem vindo ao jogo Sumdoku!
As pistas do puzzle:
 1 1 1
 1 1 2
 1 2 2
G1 = 14 G2 = 4
Tens 9 tentativas para o resolver. Boa sorte!
Casa a preencher? 1
Valor a colocar? 1
 1 . .
 . . .
 . . .
Casa a preencher? 2
Valor a colocar? 2
 1 2 .
 . . .
 . . .

...

Casa a preencher? 8
Valor a colocar? 2
 1 2 3
 1 2 3
 1 2 .
Casa a preencher? 9
Valor a colocar? 3
 1 2 3
 1 2 3
 1 2 3
Opps, tentativas esgotadas!
Queres tentar resolver um novo puzzle (true/false)? false
Espero que tenhas gostado. Volta sempre!
```

```
$ java SumdokuTxt 6
Não existem mais puzzles de tamanho 6 para jogar.
```

```
$ java SumdokuGUI[1]
```





O que entrego? Os ficheiros **SumdokuGrid.java**, **SumdokuPuzzle.java**, **SumdokuRandomPuzzle.java** e **SumdokuTxt.java**. Não há relatório a entregar porque o vosso software é a vossa documentação. Assim, não se esqueçam de comentar a vossa classe. Devem incluir no início da classe um cabeçalho *javadoc* com **@author** (nome e número dos alunos que compõem o grupo). Para cada classe e método público há que preparar um cabeçalho incluindo a sua descrição, e, se for caso disso, **@param**, **@requires**, **@ensures** e **@return**. Apresentem código que siga as normas de codificação em Java, bem alinhado e com um número de colunas adequado.

Como entrego o trabalho? Um dos alunos do grupo entrega o trabalho através da ligação que, para o efeito, existe na página da disciplina no *moodle*. O prazo de entrega é dia 17 de Dezembro às 23h.

Quanto vale o trabalho? Esta segunda parte do trabalho é cotada para 10 valores e irá somar à nota da primeira parte.

---

[1] Ver variações necessárias desta instrução no final.

Correr a Interface Gráfica

A interface que vos é fornecida foi desenvolvida usando umas bibliotecas que nem todos os Java têm (os mais recentes não têm). Assim, a forma de correr a interface gráfica vai depender não só do Sistema Operativo mas também do Java instalado. As instruções a seguir apenas contemplam o caso do Linux, que é o SO suportado em IP. No Linux podem ver que Java têm instalado com `java -version`

Máquina Virtual do DI. A máquina virtual do DI tem um **linux** com o **Java 8 da Oracle** instalado que tem todas as bibliotecas necessárias. Neste caso, basta colocar os ficheiros com o código desenvolvido juntamente com o que é fornecido (ou seja, na mesma diretoria). Nessa diretoria, depois de terem compilado o vosso código normalmente, basta executar com `java SumdokuGUI`.

```
$ javac *.java
$ java SumdokuGUI
```

Laboratórios do DI. As máquinas dos laboratórios do DI tem um **linux** com o **Java 21** instalado. Neste caso, basta colocar os ficheiros com o código desenvolvido juntamente com o que é fornecido (ou seja, na mesma diretoria). Nessa diretoria, precisam de compilar e executar com os comandos abaixo (que vão dizer onde estão as biblotecas necessárias).

```
$ javac --module-path fx --add-modules javafx.controls,javafx.fxml *.java
$ java  --module-path fx --add-modules javafx.controls,javafx.fxml SumdokuGUI
```

Estas instruções servirão (em princípio) também para quem tem outras versões recentes do Java e usa Linux. Quem tiver um mac recente (com os processadores da Apple) e um Java ainda mais recente que 21, para quem isto não funcionar, deve colocar um post no forum a descrever a situação e o erro.