Departamento de Ciência da Computação – IME-USP
MAC 110 – Introdução à Computação

Primeiro Semestre 2006 – BCC – Prof. Alfredo Goldman

Terceiro Exercício-Programa

Data de Entrega: xx/06/2006
Local de entrega: http://paca.ime.usp.br/

Esse é o enunciado do terceiro exercício-programa (EP) e apesar de ser maior do que os enunciados anteriores, esse EP não está mais difícil do que os outros. Qualquer alteração no enunciado ou nos métodos auxiliares serão comunicadas através do forum do PACA. Leia com atenção o enunciado e não esqueça de identar e comentar o seu código, pois nesse EP o desconto para esses erros será maior.

1. Introdução

1.1. O que é sudoku

Sudoku é um jogo que exige raciocínio e lógica baseado em regras simples. Na sua forma mais popular o sudoku utiliza um tabuleiro de 9 x 9 *células*, subdividido em 9 *caixas* de 3 x 3 células. Cada tabuleiro possui portanto 9 *linhas*, 9 *colunas* e 9 *caixas*, conforme representado na Figura 1.

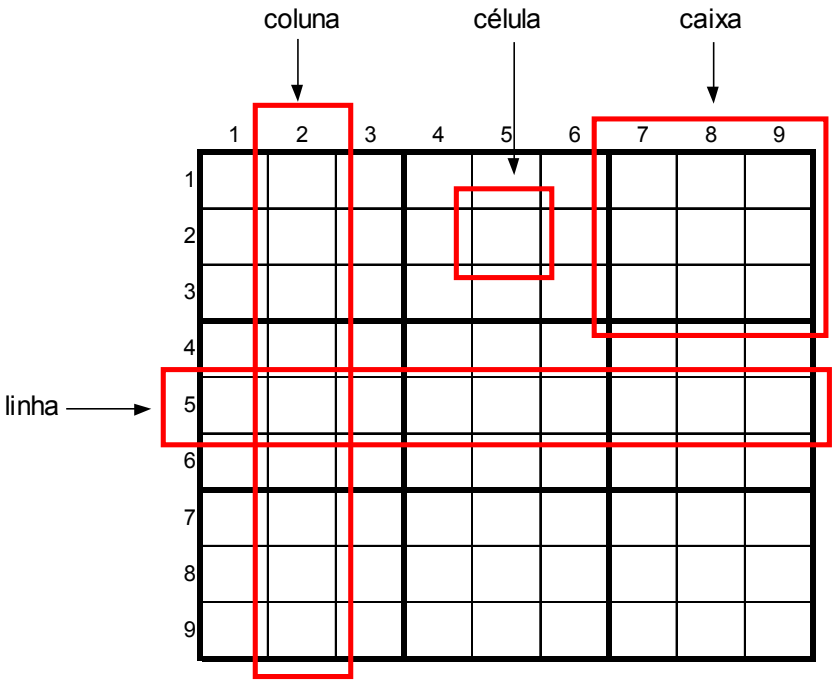

Figura 1 – Tabuleiro Padrão do sudoku 9x9

Observe na Figura 1 que as linhas e as colunas estão numeradas de 1 a 9. Normalmente os tabuleiros de sudoku não estão numerados, mas utilizaremos este artifício para facilitar a explicação das regras do jogo. Por convenção vamos nos referir a cada célula por um par de números inteiros: (*linha, coluna*). Assim, à célula indicada na Figura 1 nos referimos como (2,5).

O objetivo do jogo é preencher todas as células do tabuleiro com números inteiros de 1 a 9 obedecendo as seguintes regras:

1. Não pode haver números repetidos em uma mesma linha
2. Não pode haver números repetidos em uma mesma coluna
3. Não pode haver números repetidos em uma mesma caixa

Seguindo estas regras, ao final do jogo cada caixa 3x3, cada linha e cada coluna deve conter números entre 1 e 9, cada número aparecendo uma e somente única vez. A ordem na qual os números aparecem nas linhas, nas colunas e nas caixas não é relevante.

O jogo começa a partir de um tabuleiro parcialmente preenchido, conforme Figura 2, e a sua tarefa é preencher as demais células restantes seguindo as 3 regras básicas estabelecidas acima. Para cada jogo existe uma **única** solução. O grau de dificuldade do jogo é definido de acordo com o número de células previamente preenchidos no tabuleiro inicial. A Figura 2 apresenta um exemplo fácil, pois várias células estão preenchidas e portanto é mais fácil determinar como preencher as células restante. Vamos utilizar este exemplo para exemplificar o que deve ser feito no EP.

	1	2	3	4	5	6	7	8	9
1				1	2			3	4
2				5	3			6	7
3	8	3	2	4					
4			7	2	6		9		
5		5	1		9		3	8	
6			4		8	5	1		
7						3	4	5	9
8	6	4			5	2			
9	1	9			4	7			

Figura 2 – Tabuleiro inicial

1.2. Analisando uma Jogada Válida

Vamos analisar uma possível jogada válida. Considere a caixa no canto superior esquerdo da Figura 2, cujas linhas e colunas estão numeradas de 1 a 3. Nesta caixa vemos que as células (3,1), (3,2) e (3,3) já estão preenchidas respectivamente com os números 8, 3 e 2, portanto, para completá-la podemos utilizar apenas os números 1, 4, 5, 6, 7 e 9. Vamos iniciar decidindo, por exemplo, onde podemos colocar o número 4.

O número 4 não pode ser colocado na linha 1 pois na célula (1,9) já existe o número 4 e portanto estaríamos violando a regra do jogo. Pelo mesmo motivo não poderíamos colocar o 4 na linha 3 pois a célula (3,4) já contém o número 4. O número 4 também não pode ser colocado nas colunas 2 e 3 pois os as células (8,2) e (6,3) foram preenchidas com o número 4. Desta forma, a única posição válida para o número 4, e portanto na qual ele poderia ser colocado, é apenas a célula (2,1). O raciocínio feito esta ilustrado na Figura 3.

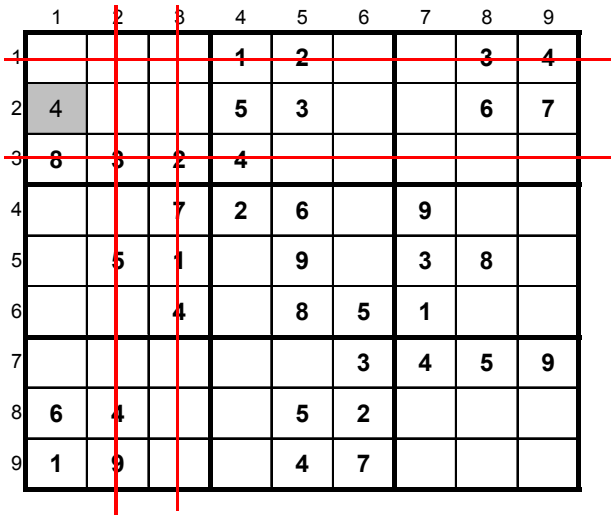

Figura 3 – Exemplo de jogada válida

Fácil não ?!

Não exatamente....Esta jogada foi fácil pois o fato de já existirem vários números previamente preenchidos reduziu o número de opções de algarismos para preencher a célula (2,1). Para entender o que isto significa, repita o mesmo raciocínio tente determinar na caixa do canto superior esquerdo a posição do número 9. Você verá que a princípio ele poderá estar em mais de uma célula e que portanto sua posição não esta determinada nesta etapa do jogo pois ela dependerá do preenchimento das células das outras linhas, colunas e caixas. Quanto menos números preenchidos no tabuleiro inicial maiores os graus de liberdade de cada célula, conforme detalhada no item 1.4, e portanto mais difícil o jogo se torna.

1.3. Analisando Jogadas Não Válidas

As figuras abaixo apresentam jogadas não válidas.

	1	2	3	4	5	6	7	8	9
1				1	2	9	8	3	4
2	4		9	5	3	8		6	7
3	8	3	2	4	7	6	5		
4		8	7	2	6	1	9	4	5
5		5	1	7	9		3	8	
6	9		4	3	8	5	1	7	2
7		2	8		1	3	4	5	9
8	6	4	9		5	2		1	8
9	1	9			4	7		2	

Figura 4 – Jogada inválida pois repete o número 9 na coluna 3

	1	2	3	4	5	6	7	8	9
1				1	2	9	8	3	4
2	4		9	5	3	8		6	7
3	8	3	2	4	7	6	5	2	
4		8	7	2	6	1	9	4	5
5		5	1	7	9		3	8	
6	9		4	3	8	5	1	7	2
7		2	8		1	3	4	5	9
8	6	4			5	2		1	8
9	1	9			4	7			

Figura 5 – Jogada inválida pois repete o número 2 na linha 3

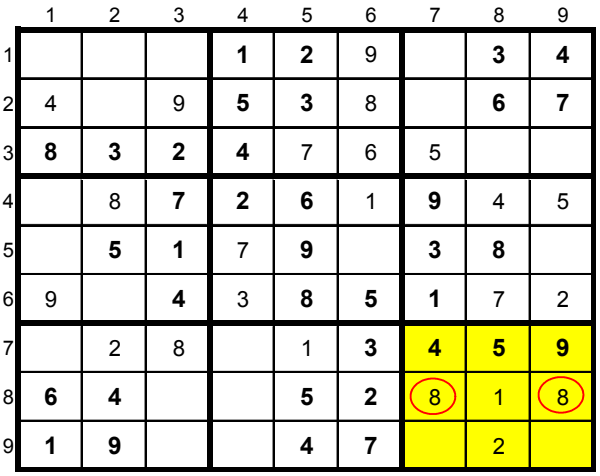

Figura 6 – Jogada inválida pois repete o número 8 na caixa do canto inferior direito

1.4. Graus de Liberdade

Existem várias estratégias para preencher o tabuleiro do sudoku. Uma destas estratégias é a cada jogada, identificar para cada célula as respectivas restrições e determinar quantos números ela pode assumir. Em seguida escolher para preencher uma das células ainda vazias e que tenha o menor número de opções possíveis. A esta quantidade, que informa o número de possíveis maneiras de preencher uma célula, denominaremos *grau de liberdade* da célula.

Para calcular o grau de liberdade da célula (*i, j*) partimos de dois princípios básicos: i) se não há nenhuma restrição em um tabuleiro 9 x 9 cada célula pode assumir valores de 1 a 9, e portanto o grau de liberdade máximo será 9; ii) o grau de liberdade de células já preenchidas é zero.Em seguida verificamos a ocorrência de números distintos na caixa a que a célula (*i, j*) pertence, na i-ésima linha e na j-ésima coluna. Subtraímos a quantidade de números distintos encontrados do grau de liberdade máximo para determinar o grau de liberdade da célula. Quanto mais números colocamos, mais restrições são criadas e portanto, conforme preenchemos o tabuleiro os graus de liberdade das células diminuem.

Por exemplo, vamos calcular o grau de liberdade da célula (3,5), acompanhe a lógica observando a Figura 7

Números da linha 3: 2, 3, 4, 8

Números da coluna 5: 2, 3, 4, 5, 8, 9

Números da Caixa: 1, 2, 3, 4, 5

Números distintos nas restrições: 1, 2, 3, 4, 5, 6, 8, 9 = 8 números distintos

Graus de liberdade célula (3,5): 1 = 9 –8

Números possíveis para célula (3,5): 7

Como o grau de liberdade da célula é 1, significa que para esta célula, apenas um número é válido, no caso será o número 7, pois é o único número que não aparece nas restrições.

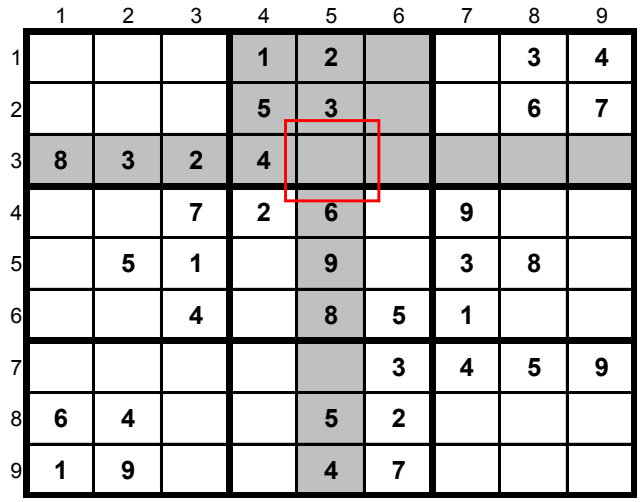

Figura 7 – Grau de liberdade da célula (3,5)

No entanto, se o grau de liberdade for maior que 1, mais de um número poderá ser colocado na célula. Por exemplo, observe a Figura 8 que utilizaremos para calcular os graus de liberdade da célula (9,9).

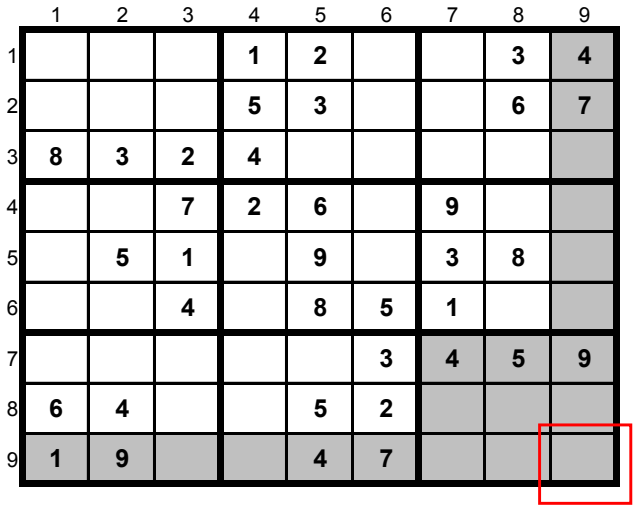

Figura 8 – Grau de liberdade da célula (9,9)

Para a célula (9,9) temos as seguintes restrições:

Números da linha 9: 1, 4, 7, 9

Números da coluna 9: 4, 7, 9

Números da Caixa: 4, 5, 9

Números distintos nas restrições: 1, 4, 5, 7, 9= 5 números distintos

Graus de liberdade célula (9,9): 4 = 9 –5

Números possíveis para célula (9,9): 2, 3, 6, 8

Portanto a célula (9,9), nesta etapa do jogo tem 4 graus de liberdade e portanto 4 números possíveis.Qual dos números escolher depende seu raciocínio.

A estratégia de preencher a cada jogada as células vazias que tenham o menor grau de liberdade é um indicativo de como completar o sudoku, mas para os problemas mais difíceis ela não é suficiente. Neste caso faz-se necessário um raciocínio mais elaborado e esta estratégica básica deve ser associada a outras.

2. Exercício Programa

2.1. Objetivos

O objetivo deste EP é fazer com que os alunos aprendam a utilizar variáveis com várias dimensões, como arrays e matrizes. Para isto o aluno deverá implementar três classes (**Sudoku, Jogada e GuardadorDeJogadas).** A primeira irá carregar um tabuleiro de sudoku a partir de um arquivo texto, inserido via teclado ou gerar um aleatório, e em seguida interagir com o jogador no intuito de preencher corretamente o tabuleiro. A classe Jogada é auxiliar, e a última, GuardadorDeJogadas, servirá para implementar a funcionalidade de desfazer jogadas (*Undo*).

Para tanto, o aluno deverá implementar pelo menos os métodos descrito nas próximas secões, os quais cumprem as funções mínimas necessárias para inicializar o tabuleiro, interação com o jogador e verificação do preenchimento do tabuleiro.

2.2. Classe Sudoku

Os seguintes métodos são obrigatórios e deverão ser implementados exatamente como definido a seguir. Dentro de poucos dias disponibilizaremos uma interface gráfica – GUI – (e dessa vez com o codigo fonte :-)). Essa interface irá chamar os métodos da sua classe da maneira como estão descritos aqui:

Construtor: `Sudoku()`
Função: Inicializar os atributos da classe que implementa o jogo sudoku
Parâmetros: Não há parâmetros

Método: `void leTabuleiroArquivo(String nomeDoArquivo)`
Função: Carregar o tabuleiro especificado em um arquivo texto em um padrão predefinido descrito no item 2.5.3
Parâmetros: String nomeDoArquivo: nome do arquivo a ser lido
Retorno: Não há valor de retorno

Método: `void leTabuleiroTeclado()`
Função: Carregar o tabuleiro diretamente do teclado conforme descrito no item 2.6., este método deverá validar a entrada via teclado de acordo com as dimensões do tabuleiro. Por exemplo, se o tabuleiro é 9x9 não deverá permitir que o usuário tente inserir um valor na coluna 10.
Parâmetros: Não há parâmetros de entrada
Retorno: Não há valor de retorno

Método: `void geraTabuleiroAleatorio()`
Função: Método que gera um tabuleiro aleatóriamente conforme descrito no item 2.6.
Parâmetros: Não há parâmetros de entrada
Retorno: Não há valor de retorno

Método: boolean `verificaValidadeTabuleiro()`
Função: Método que verifica a validade do tabuleiro. Ele deve ser acionado toda vez que um novo tabuleiro for carregado para verificar se o mesmo não apresenta inconsciências que infrinjam as regras básicas estabelecidas no item 1.1. Caso seja o inicio do jogo e o tabuleiro não seja válido o jogar deve abandonar o jogo em andamento, iniciar um novo e carregar um novo tabuleiro válido.
Este método pode também ser invocado a qualquer instante do jogo para verificar a consistência do tabuleiro ou de uma jogada do usuário.
Parâmetros: Não há parâmetros de entrada
Retorno: *true* se o tabuleiro for válido, e *false* se o tabuleiro for inválido.

Método: `int[][] devolveTabuleiro()`
Função: Método que retorna a representação do tabuleiro do jogo (será usado pela interface gráfica)
Parâmetros: Não há parâmetros de entrada
Retorno: Matriz com os valores já preenchidos no tabuleiro

Método: `void limpaTabuleiro()`
Função: Voltar o tabuleiro à representação original, isto é, remover todas as jogadas. (**Dica**: você também precisará limpar o guardadorDeJogadas, descrito no item 2.4)
Parâmetros: Não há parâmetros de entrada
Retorno: Não há valor de retorno

Método: `boolean executaJogada(Jogada novaJogada)`
Função: Método que o jogador irá utilizar para fazer as jogadas para preencher as células. A cada jogada proposta pelo usuário o programa deverá verificar se a jogada é válida, observando as regras descritas no item 1.1 e além disto, verificando se o jogador não esta tentando alterar uma célula que faça parte do tabuleiro inicialmente proposto. O jogador não pode ao longo do jogo alterar as células que vierem preenchidas no tabuleiro inicial. A cada jogada este método deve informar o jogador se a jogada foi válida ou se ocorreu algum conflito. Opcionalmente, pode informar qual o conflito. (**Dica**: você precisará criar uma Jogada com o valor anterior da célula e inserir no guardadorDeJogadas, descrito no item 2.4)
Parâmetros: Jogada novaJogada: objeto do classe Jogada (descrito no item 2.3, que contém: linha, coluna e valor a ser preenchido)
Retorno: *true* se a jogada for válida, e *false* se a jogada não for válida.

Método: `void imprimeTabuleiro()`
Função: Imprimir o tabuleiro no formato texto de uma forma amigável para que o jogador possa acompanhar o jogo. Quanto mais "bonito" visualmente melhor será, mas as recomendações mínimas são: i) cada linha do tabuleiro seja impressa em uma linha na tela; ii) os números sejam separados por espaços ou outro caractere; iii) os espaços vazios sejam preenchidos com 0 (zero) ou algum outro caractere de sua preferência.
Um exemplo de impressão seria:

```
-  -  -  1  2  -  -  3  4
-  -  -  5  3  -  -  6  7
8  3  2  4  -  -  -  -  -
-  -  7  2  6  -  9  -  -
-  5  1  -  9  -  3  8  -
-  -  4  -  8  5  1  -  -
-  -  -  -  -  3  4  5  9
6  4  -  -  5  2  -  -  -
1  9  -  -  4  7  -  -  -
```

Parâmetros: Não há parâmetros de entrada
Retorno: Não há valor de retorno

Método: `int[][] calculaGrausDeLiberdade()`
Função: Método que calcula os graus de liberdade de cada célula do tabuleiro corrente do jogo para ajudar na tomada de decisão do jogador. Este método poderá ser invocado pelo jogador a qualquer momento, desde que algum tabuleiro tenha sido carregado. A forma como a informação será passada para o jogador fica a critério do aluno mas uma dica é utilizar um método semelhante `imprimeTabuleiro` descrito acima para mostrar visualmente os graus de liberdade de cada célula para o jogador.
Parâmetros: Não há parâmetros de entrada
Retorno: Uma matriz, com as dimensões iguais a do tabuleiro (9x9), contendo o grau de liberdade de cada posição do tabuleiro.

Método: `boolean verificaFimDoJogo()`
Função: Método que verifica se ainda existem células a serem preenchidas pelo jogador.
Parâmetros: Não há parâmetros de entrada
Retorno: *true* se o jogo acabou, e *false* caso contrário.

Método: `Jogada sugereProximaJogada()`
Função: Procurar pela existência de alguma jogada válida, vocês podem implementar este método de maneira simplista, isto é, procurar por alguma célula que tenha grau de liberdade = 1.
Parâmetros: Não há parâmetros de entrada
Retorno: objeto do tipo Jogada contendo a posição (linha,coluna) e o valor da jogada válida, caso não exista, devolve *null*

Método: `Jogada[] calculaPossiveisJogadas(int linha, int coluna)`
Função: Procura por possíveis valores que possam ser colocados na célula (linha, coluna)
Parâmetros: Não há parâmetros de entrada
Retorno: vetor de objetos do tipo Jogada uma lista de jogadas possíveis para uma determinada célula, devolve um e valor de uma jogada válida. Devolve *null* caso não existam jogadas possíveis.

Método: `boolean desfazerJogada()`
Função: Volta o tabuleiro para um estado anterior ao atual (isto é, desfaz a última jogada). Este método poderá ser chamado várias vezes seguidas, e a camada chamada, volta uma jogada. Para implementar esta funcionalidade, vocês deverão usar uma classe auxiliar **GuardadorDeJogadas** como descrita no item 2.4.
Parâmetros: Não há parâmetros de entrada
Retorno: devolve *true* se a jogada foi refeita, e *false* caso não existam mais jogadas a desfazer

Outros métodos poderão ser incluídos com objetivo de melhorar a interface com o jogador ou automatizar procedimentos.

2.3. Classe Jogada

Vocês também deverão implementar uma classe auxiliar de nome **Jogada**, que será usada para armazenar os paramêtros de uma jogada. Esta classe deverá conter exatamente três atributos: int linha; int coluna; int jogada; Atenção: Os atributos desta classe deverão ser particulares, só acessiveis internamente (i.e. *private*), e portanto, os seguintes métodos acessores deverão existir:

Construtor: `Jogada(int l, int c, int v)`
Funcão: Inicializa os atributos linha, coluna e valor do objeto

Método: `int leLinha()`
Retorno: devolve o número da linha correspondente a esta jogada

Método: `int leColuna()`
Retorno: devolve o número da coluna correspondente a esta jogada

Método: `int leValor()`
Retorno: devolve o número (de 1 a 9)

2.4. Classe GuardadorDeJogadas

A classe GuardadorDeJogadas será usada para implementar o método 'desfazerJogada()' do Sudoku (i.e. função *Undo*). A quantidade de movimentos que serão armazenados será definido no construtor da classe (por exemplo, 15 jogadas). Um atributo essencial desta classe é um *array* de objetos '`Jogada`'. Outro atributo importante é um marcador da última posição usada do *array*. Segue a lista de métodos necessários (**Dica:** Implemente esta parte somente depois que o restante do jogo esteja funcionando Ok!)

Constutor: `GuardadorDeJogadas(int capacidade)`
Função: Inicializa os atributos do guardador (cria um array de tamanho 'capacidade' e inicializa o ponteiro que indica a utilizacao do array). O valor do parâmetro capacidade deverá ser maior que 10.
Parâmetros: int capacidade: o número máximo de jogadas que serão armazenadas pelo guardador (>= 10)

Método: `void insere(Jogada jogada)`
Função: Insere 'jogada' no array de jogadas do guardador (**Note que:** esta não é exatamente a jogada feita pelo usuário, mas sim, uma jogada ficticia que contém o valor anterior da célula. Portanto, antes de alterar o valor da célula, vcs deverão criar um objeto do tipo Jogada para guardar este valor anterior). **Atenção:** vocês também precisam tomar cuidado com o limite de jogadas a ser armazenado, caso o vetor fique cheio, a jogada mais antiga será jogada fora!.
Parâmetros: Não há parâmetros de entrada
Retorno: Não há parâmetros de retorno

Método: `Jogada devolveUltima()`
Função: Remove e devolve o último objeto inserido no array de jogadas do guardador.
Parâmetros: Não há parâmetros de entrada
Retorno: Devolve a última Jogada inserida no guardador, caso o guardador esteja vazio, devolve *null*

Método: `void limpa()`
Função: Apaga todas as jogadas que estão no guardador (lembre-se de reposicionar o marcador)
Parâmetros: Não há parâmetros de entrada
Retorno: Não há parâmetros de retorno

2.5. Leitura do Tabuleiro

Para carregar o tabuleiro no início do jogo, o aluno poderá fazê-lo a partir de um arquivo texto ou através do teclado, informando manualmente o conteúdo inicial da célula. Para facilitar esta parte disponibilizamos um arquivo no Paca (*Leitor.java*) que contem a interface do *Leitor* e duas classes que a implementam: *LeitorDeArquivo* e *LeitorDeTeclado*. Para utilizar estas classes o aluno deve baixar e compilar o arquivo *Leitor.java* no DrJava ou copiar o conteúdo do arquivo para dentro do seu próprio arquivo.

A interface tem, entre outros, o método:

Método: `int[] lerCelula()`
Função: Método que lê o conteúdo de uma célula do tabuleiro, seja do arquivo texto ou do teclado.
Parâmetros: Não há parâmetros de entrada
Retorno: O método retorna um array com 3 números inteiros. O primeiro número corresponde à coordenada da linha, o segundo número à coordenada da coluna e o terceiro ao valor propriamente dito da célula. Por exemplo, o array (6,5,8) indica que o conteúdo da célula (6,5) é 8. Caso o arquivo tenha chegado ao fim ou o jogador tenha inserido uma linha em branco, no caso da entrada via teclado, o método retorna *null* indicado que a entrada de dados chegou ao fim.

2.5.1. Leitura do Tabuleiro do Arquivo Texto

Para implementar a leitura do tabuleiro a partir do arquivo texto o aluno pode se basear no código abaixo:

```
(...)
// Inicializa um objeto leitor de arquivos
Leitor leitor = new LeitorDeArquivo("sudoku01");
int[] celula = new int[3];
(...)
// Lê linha a linha o arquivo e processa a informação
while ((celula = leitor.leCelula())!=null) {

   // Trata o conteúdo da célula lido
   System.out.println("linha:"+celula[0]+" coluna:"+celula[1]+" valor:"+celula[2]);

}
(...)
leitor.fechaEntrada();
```

2.5.2. Detalhamento da Entrada via Teclado

Para implementar a leitura do tabuleiro de forma interativa através do teclado o usuário terá que fornecer a cada para cada célula o número a coordenada da linha, a coordenada da coluna e o valor da célula. O objeto leitor irá fazer perguntas específicas e o usuário terá que informar um número inteiro e pressionar a tecla <Enter>. Para terminar a entrada de dados, basta o usuário informar uma linha vazia, ou seja, pressionar a tecla <Enter> sem fornecer informação alguma. Entradas que não sejam números serão ignoradas.

Por exemplo, considerando que os número em itálico corresponde aos números digitados pelo usuário, o objeto Leitor irá fazer as seguintes perguntas:

```
Digite o número da linha: 6
Digite o número da coluna: 5
Digite o valor da célula: 8
(...)
Digite o número da linha: 9
Digite o número da coluna: 6
Digite o valor da célula: 7
// Para teminar deixe a linha vazia
```
Digite o número da linha:

```
// Fim da entrada de dados.
```

Para implementar a leitura do tabuleiro a partir do teclado o aluno pode se basear no código abaixo:

```
(...)
// Inicializa um objeto leitor de arquivos
Leitor leitor = new LeitorDeTeclado();
int[] celula = new int[3];
(...)
// Lê o valor da célula informada via tecladao e processa a informação
while ((celula = leitor.leCelula())!=null) {

   // Trata o conteúdo da célula lido
   System.out.println("linha:"+celula[0]+" coluna:"+celula[1]+" valor:"+celula[2]);

}
(...)
leitor.fechaEntrada();
```

2.5.3. Detalhamento do Arquivo de Entrada

A título de exemplo, será disponibilizado no Paca o arquivo *sudoku01.txt* com a definição do arquivo tabuleiro inicial que utilizamos como exemplo. Outros tabuleiros podem ser propostos pelo aluno, o arquivo de entrada com a descrição do tabuleiro é um arquivo texto ("*plain-text*") que pode ser criado utilizando o *vi*, *emacs* ou o "Bloco de notas" do Windows, não tente usar o *Word* ou o *WordPad*.

O arquivo texto tem várias linhas, cada linha corresponde à informação sobre uma célula na forma de 3 números inteiros separados por espaço. O primeiro número corresponde à coordenada da linha, o segundo número à coordenada da coluna e o terceiro ao valor propriamente dito da célula. Por exemplo a linha em negrito do pedaço de arquivo texto abaixo:

```
(...)
3 4 4
6 5 8
5 7 3
(...)
```

Indica que o conteúdo da célula (6,5) é 8.

2.6. Detalhamento da Geração do Tabuleiro

Uma propriedade interessante do sudoku é que a partir de uma solução completa inicial, ou seja, um tabuleiro já totalmente preenchido e válido segundo as regras do sudoku, se pegarmos duas colunas de caixas (ver figura 9 abaixo) quaisquer e fizermos cada uma ter os valores da outra, o tabuleiro continua válido.O mesmo conceito pode ser aplicado para linhas de caixas. Isto ocorre porque troca de colunas ou linhas de caixas mantém todos os números diferentes nas colunas ou linhas, e não altera os valores das caixas movidas.

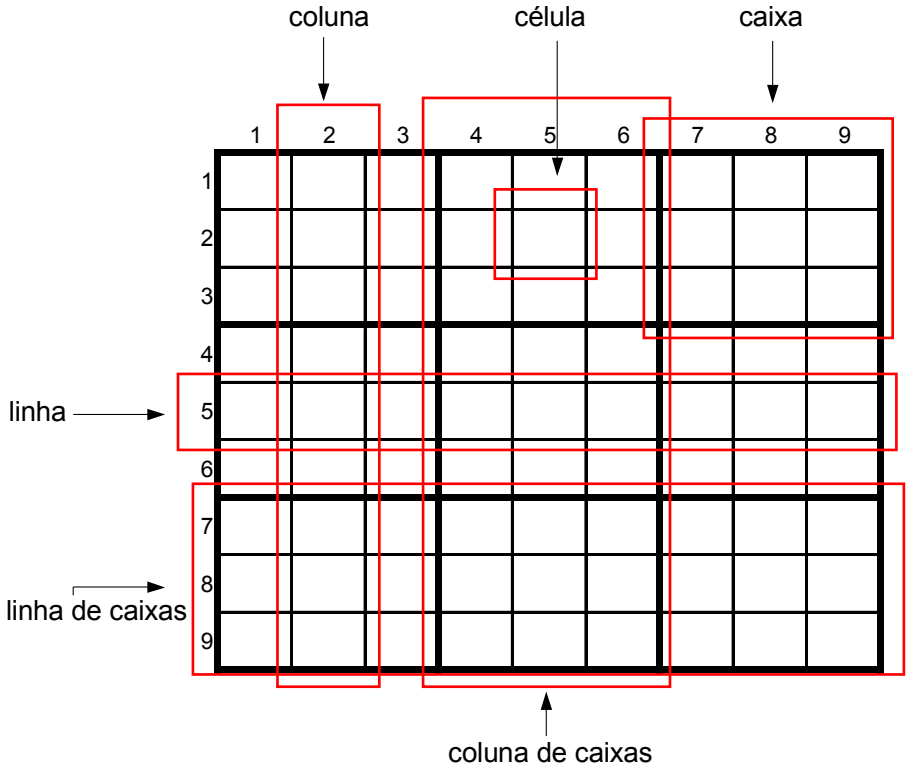

Figura 9 – Identificação de linha e coluna de caixas

Outra propriedade interessante é que dentro uma coluna de caixas, se trocarmos duas colunas entre si, a solução também continuará valida. O mesmo conceito pode ser aplicado para as linhas dentro uma linha de caixas.Isto

ocorre porque trocas de colunas ou linhas dentro de colunas e linhas de caixa, respectivamente, mantém todos os números diferentes nas linhas e colunas.

Porém, notem que não podemos trocar colunas entre colunas de caixas diferentes ou linhas de linhas de caixa diferentes, pois apesar de mantermos o fato de valores diferentes para linhas e colunas, a terceira regra pode vir a ser violada.

Como parte deste ep, pede-se para retornar um tabuleiro aleatório não completo inicial a partir de uma solução válida inicial.

Uma solução válida inicial seria simplesmente o seguinte abaixo, mas poderia ser qualquer outro tabuleiro válido.

Exemplo

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	4	5	6	7	8	9	1	2	3
3	7	8	9	1	2	3	4	5	6
4	2	3	4	5	6	7	8	9	1
5	5	6	7	8	9	1	2	3	4
6	8	9	1	2	3	4	5	6	7
7	3	4	5	6	7	8	9	1	2
8	6	7	8	9	1	2	3	4	5
9	9	1	2	3	4	5	6	7	8

Figura 10 – Tabuleiro base para geração de tabuleiros aleatórios

Para o método em questão, será obrigatório haver 5 fases. A primeira para a troca de colunas de caixas, a segunda para a troca de linhas de caixas, a terceira para trocas de colunas dentro de colunas de caixas, a quarta para a troca de linhas dentro de linhas de caixas, e a quinta para apagar aleatoriamente alguns valores do tabuleiro para torná-lo jogável.

Para a primeira fase, basta criar um vetor de tamanho 3 que contenha uma permutação aleatória da seqüência de valores 0, 1 e 2. Tomando o índice do vetor como o número da coluna de faixas, os valores contidos no vetor corresponderão à nova permutação das colunas de caixas. A segunda fase segue o mesmo conceito, assim como a terceira e quarta fases.

Na quinta fase, deve-se apagar aleatoriamente pelo menos metade dos valores contidos no tabuleiro. Porém, deve-se tomar cuidado para que dentro de uma linha ou coluna de caixas não haja mais de uma linha ou coluna, respectivamente, sem nenhum valor. Caso haja, estaremos criando um tabuleiro com mais de uma solução. Imagine por exemplo se duas linhas de uma linha de caixas ficam sem nenhum valor depois de apagados os valores. Como podemos trocar linhas, haveria 2 soluções diferentes pelo fato dessas duas linhas poderem ser intercambiáveis.

Portanto, deve-se obter um modo de evitar isso. Para tal, cria-se um vetor de tamanho 9 cujos valores sejam uma permutação aleatória da seqüência de 0 a 8. Deste modo, na posição 0(zero) do vetor, por exemplo, terá-se o valor do índice da coluna cujo valor não pode ser removido na linha 0(zero). Assim, com os 9 valores desse vetor, todas as linhas e colunas terão pelo menos um valor. **Opcional:** Uma outra característica de alguns

tabuleiros é a simetria dos valores prenchidos/branco das caixas opostas em relacão ao centro. Repare no exemplo no final do enunciado como a caixa superior esquerda é 'espelhada' com a do canto inferior direito, como desafio, vocês podem gerar tabuleiros com essa característica.

Para representar uma célula que não tenha nenhum valor, coloquem o valor –1, e não 0 (zero), para evitar qualquer dúvida que seja uma célula sem valor.

2.7. O que deve ser entregue?

Cada aluno, ou dupla de alunos, deverá entregar um arquivo compactado contendo 2 arquivos:

1. Arquivo Java com a implementação da classe **Sudoku, Jogada e GuardadorDeJogadas**
2. Arquivo texto: SudokuSimula.txt

O nome do arquivo deve necessariamente seguir o padrão:

Sudoku_NomeDoAluno_SiglaDoCurso.zip

No caso de duplas:

Sudoku_NomeDoAluno01_ NomeDoAluno02_SiglaDoCurso.zip

A simulação deverá conter 3 jogos:

1. Jogo carregado do arquivo exemplo *sudoku01.txt*
2. Jogo inserido pelo teclado, pode ser o mesmo jogo *sudoku01*
3. Jogo de tabuleiro gerado aleatóriamente

Ao realizar a simulação o aluno não precisa imprimir o **tabuleiro** e **os graus de liberdade** a cada jogada, mas deverá fazê-lo necessariamente: i) No início do jogo logo após carregar o tabuleiro; ii) ao fim do jogo, com ele já totalmente preenchido; iii) 2 vezes ao longo de cada jogo.

O aluno deve simular também 3 jogas inválidas. Por exemplo tentar violar as regras do sudoku ou sobrescrever uma informação do tabuleiro inicial.

Este é o terceiro e EP e vocês já estão bem *grandinhos*, portanto, sugerimos seguir as seguintes recomendações:

1. Antes de começar gaste um tempo planejando como irá implementar seu código, depois que seu projeto estiver “pronto”, só então passe para a implementação. Assim seu código ficará mais limpo e provavelmente você poupará seu tempo.
2. Capriche a indentação do seu programa pois, além de ser uma prática elegante, ela será avaliada
3. Capriche nos comentários. Seja conciso mas informativo. Descreva de forma sucinta os atributos da classe. Para os métodos, você pode seguir mais ou menos o padrão acima e descrever para cada um, a função, os parâmetros de entrada e o retorno.

3. Exemplo e Solução de um Tabuleiro

Para facilitar a implementação e os testes, a Figura 11 apresenta o tabuleiro inicial e a solução para o problema proposto no arquivo *sudoku01.txt*.

	1	2	3	4	5	6	7	8	9
1				**1**	**2**			**3**	**4**
2				**5**	**3**			**6**	**7**
3	**8**	**3**	**2**	**4**					
4			**7**	**2**	**6**		**9**		
5		**5**	**1**		**9**		**3**	**8**	
6			**4**		**8**	**5**	**1**		
7						**3**	**4**	**5**	**9**
8	**6**	**4**			**5**	**2**			
9	**1**	**9**			**4**	**7**			

Figura 11 – Tabuleiro inicial descrito no arquivo *sudoku01.txt*

A Figura 12 é a respectiva solução para o problema proposto, como é possível observar, os números em negrito correspondem aos números inicialmente preenchidos no tabuleiro.

	1	2	3	4	5	6	7	8	9
1	5	7	6	**1**	**2**	9	8	**3**	**4**
2	4	1	9	**5**	**3**	8	2	**6**	**7**
3	**8**	**3**	**2**	**4**	7	6	5	9	1
4	3	8	**7**	**2**	**6**	1	**9**	4	5
5	2	**5**	**1**	7	**9**	4	**3**	**8**	
6	9	6	**4**	3	**8**	**5**	**1**	7	2
7	7	2	8	6	1	**3**	**4**	**5**	**9**
8	**6**	**4**	3	9	**5**	**2**	7	1	8
9	**1**	**9**	5	8	**4**	**7**	6	2	3

Figura 12 – Tabuleiro resolvido

Enunciado versão 1 (06/06/06)