Programação Estruturada

Aula 10 - Jogo do labirinto parte 1 – uso de matriz

Apresentação

Na aula passada, você estudou conceitos de recursão aplicada à programação. No contexto de nossa disciplina, a recursão é a implementação de funções ou procedimentos usando sua própria definição. Essa forma de implementação pode simplificar o desenvolvimento de alguns programas e isso é o que você verá nesta e na próxima aula. Você desenvolverá um jogo de labirinto que será desenhado na tela de modo que o próprio computador terá que achar o caminho de saída.

Nesta primeira aula, desenvolveremos a parte básica do jogo, para na próxima aula você aplicar a recursão para implementar a busca pela saída do labirinto. Tenha uma boa leitura e uma boa programação do jogo!

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Implementar os recursos básicos de um jogo de labirinto.

1. Introdução

Vamos iniciar agora uma aula bem prática, com o desenvolvimento de um jogo de labirinto. Durante esta aula, nós iremos definir o que queremos fazer em termos de telas do jogo a serem desenvolvidas. Isso lhe dará uma ideia do que precisa ser implementado. Em seguida, você verá o código correspondente ao comportamento básico do jogo. Para completar o seu aprendizado, você irá resolver exercícios que estendem o comportamento do jogo de labirinto, cujo código nós iremos lhe mostrar.

A primeira decisão é como implementar a interface do jogo com o usuário. Por enquanto, vamos trabalhar apenas no modo texto, ok? Isso porque você está desenvolvendo o primeiro programa com maior complexidade visto neste módulo. Para criar esse jogo, nomearemos uma classe chamada de Labirinto, como mostrado a seguir.

```
1 public class Labirinto {
2    // aqui vem o código do jogo
3 }
```

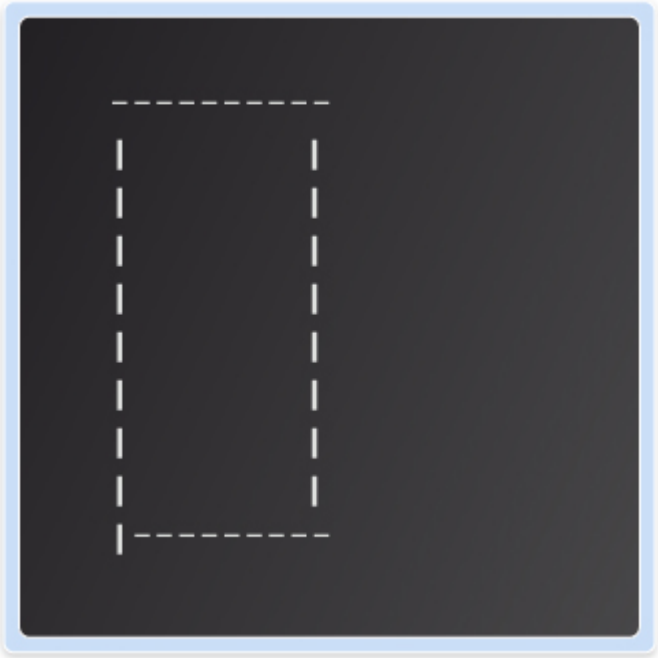
O código dessa classe, ou seja, do jogo, será apresentado para você de forma incremental nas próximas seções, visando facilitar seu entendimento.

2. Tabuleiro do Jogo

Para criar um jogo de labirinto usando o modo texto, precisamos escolher caracteres para formar o que iremos chamar de tabuleiro do jogo, ou seja, do labirinto, do caminho percorrido etc.

Para delimitar a extensão do tabuleiro, usemos os caracteres “|” e “-”. No caso, o caractere “|” será usado para delimitar as paredes externas verticais do tabuleiro. O caractere “-”, por sua vez, irá delimitar as paredes externas horizontais do tabuleiro. Isso quer dizer que teremos um tabuleiro como o mostrado a seguir.

Figura 01 - Tabuleiro em branco



O tabuleiro mostrado tem dimensão 10 x 10, ou seja, 10 linhas e 10 colunas. Aproveitando a oportunidade, existe alguma estrutura de dados que é dimensionada em termos de quantidade de linhas e colunas?

Se você não se lembrou das matrizes, ou se tem alguma dúvida na definição e uso de matrizes, releia a aula de vetores e matrizes, ok? Os conceitos apresentados naquela aula serão de grande valia para nós implementarmos o jogo do labirinto.

Vamos então começar com a parte do programa responsável por imprimir o tabuleiro vazio. Primeiro, temos que definir uma estrutura de dados para representá-lo. Essa estrutura, como já dada a dica anteriormente, é uma matriz. No caso, uma matriz de caracteres para poder armazenar os caracteres que formam o tabuleiro. Veja a seguir como isso fica no código.

```
1 public class Labirinto {
2    private static char[][] tabuleiro;
3    // aqui vem o código do jogo
4 }
```

Temos aí a definição de uma variável de nome tabuleiro. Como ela não está sendo declarada dentro de uma rotina (procedimento ou função), isso quer dizer que ela pode ser usada por qualquer rotina definida na classe *Labirinto*. Vamos

escrever o programa de forma a montar e imprimir esse tabuleiro? Veja como fica.

```
public class Labirinto {
   private static final char PAREDE_VERTICAL = '|';
   private static final char PAREDE_HORIZONTAL = '-';
   private static final char VAZIO = ' ';
   private static final char TAMANHO = 10;
   private static char[][] tabuleiro;

   public static void inicializarMatriz() {
      int i, j;
      for (i = 0; i < TAMANHO; i++) {
         tabuleiro[i][0] = PAREDE_VERTICAL;
         tabuleiro[i][TAMANHO - 1] = PAREDE_VERTICAL;
         tabuleiro[0][i] = PAREDE_HORIZONTAL;
         tabuleiro[TAMANHO - 1][i] = PAREDE_HORIZONTAL;
      }
      for (i = 1; i < TAMANHO - 1; i++) {
         for (j = 1; j < TAMANHO - 1; j++) {
            tabuleiro[i][j] = VAZIO;
         }
      }
   }

   public static void imprimir() {
      for (int i = 0; i < TAMANHO; i++) {
         for (int j = 0; j < TAMANHO; j++) {
            System.out.print(tabuleiro[i][j]);
         }
         System.out.println();
      }
   }

   public static void main(String Arg[]) {
      tabuleiro = new char[TAMANHO][TAMANHO];
      inicializarMatriz();
      imprimir();
   }
}
```

Vídeo 02 - Tabuleiro 1/3

Você conseguiu entender o funcionamento do código apresentado? Vamos a uma explicação detalhada de seu funcionamento. Primeiro, note o uso de várias constantes (variáveis declaradas com a palavra **final**).

```
1 private static final char PAREDE_VERTICAL = '|';
2 private static final char PAREDE_HORIZONTAL = '-';
3 private static final char VAZIO = ' ';
```

Essas constantes são utilizadas para representar os caracteres especiais que utilizaremos para escrita do tabuleiro na tela em modo texto. É recomendado usar constantes nesses casos porque você evita que esses caracteres sejam espalhados pelo código do programa. Dessa forma, se quisermos mudar do caractere “-” para o sublinhado (“_”), é só fazer essa alteração na declaração da constante que todo o resto do programa já irá automaticamente usar esse novo caractere definido.

Além das constantes mostradas, temos a constante a seguir definida para representar o tamanho da matriz, considerando que a mesma sempre será quadrada. A ideia é então que o tamanho do tabuleiro (ou seja, da matriz) possa ser modificado apenas mudando o valor dessa constante. De fato, esse valor poderia até ser informado pelo usuário no início da execução do programa. Temos só que garantir que o número a ser informado pelo usuário seja maior do que 4, para gerar um tabuleiro que possa ser trabalhado realmente como um labirinto, mesmo que simples. Valores menores do que 4 gerariam tabuleiros pequenos demais para se construir um jogo de labirinto.

```
1 private static final char TAMANHO = 10;
```

Em seguida, temos a definição do procedimento *inicializarMatriz()*, cujo objetivo é inicializar a matriz que representa o tabuleiro com caracteres que representem o conteúdo e as paredes do labirinto. Observemos a primeira parte do código desse procedimento:

```
1 int i, j;
2 for (i = 0; i < TAMANHO; i++) {
3    tabuleiro[i][0] = PAREDE_VERTICAL;
4    tabuleiro[i][TAMANHO - 1] = PAREDE_VERTICAL;
5    tabuleiro[TAMANHO - 1][i] = PAREDE_HORIZONTAL;
6    tabuleiro[0][i] = PAREDE_HORIZONTAL;
7 }
```

Você consegue perceber quais células da matriz *tabuleiro* serão preenchidas com cada caractere especial de parede que foi definido (*PAREDE_VERTICAL ou PAREDE_HORIZONTAL*)? Observe que a variável i vai do valor 0 até o tamanho da matriz (constante *TAMANHO*) menos um. Considerando o tamanho 10 definido na constante *TAMANHO*, o valor de *i* vai de 0 a 9. Veja que atribuições são feitas com relação às paredes verticais:

```
No laço i = 0:
   tabuleiro[0][0] = PAREDE_VERTICAL;
   tabuleiro[0][9] = PAREDE_VERTICAL;

No laço i = 1:
   tabuleiro[1][0] = PAREDE_VERTICAL;
   tabuleiro[1][9] = PAREDE_VERTICAL;

...

No laço i = 9:
   tabuleiro[9][0] = PAREDE_VERTICAL;
   tabuleiro[9][9] = PAREDE_VERTICAL;
```

Perceba que o que se está variando é o número da linha, da primeira até a última, considerando sempre a primeira e a última coluna. Dessa forma, o caractere "|" usado na parede vertical é usado para preencher a primeira e a última coluna da matriz. Processo similar é utilizado para a parede horizontal:

```
No laço i = 0:
   tabuleiro[0][0] = PAREDE_HORIZONTAL;
   tabuleiro[9][0] = PAREDE_HORIZONTAL;

No laço i = 1:
   tabuleiro[0][1] = PAREDE_HORIZONTAL;
   tabuleiro[9][1] = PAREDE_HORIZONTAL;

No laço i = 2:
   tabuleiro[0][2] = PAREDE_HORIZONTAL;
   tabuleiro[9][2] = PAREDE_HORIZONTAL;

...

No laço i = 9:
   tabuleiro[0][9] = PAREDE_HORIZONTAL;
   tabuleiro[9][9] = PAREDE_HORIZONTAL;
```

Note que o que muda em cada iteração são as colunas. Dessa forma, todas as colunas da primeira e da última linha são preenchidas com o caractere “-”, que representa uma parede horizontal do tabuleiro.

Em seguida, temos o seguinte código para preencher os elementos internos do tabuleiro com posições em branco, ou seja, caracteres vazios (“ ”):

```
for (i = 1; i < TAMANHO - 1; i++) {
    for (j = 1; j < TAMANHO - 1; j++) {
        tabuleiro[i][j] = VAZIO;
    }
}
```

Note que temos um laço dentro de outro laço. Você se lembra por que temos esse tipo de estrutura? Quando trabalhamos com matrizes e precisamos percorrer elementos de diferentes linhas e colunas, precisamos de dois laços, um para controlar o índice da linha e outro para controlar o índice da coluna. No caso, tanto os índices da linha (variável i) como da coluna (variável j) começam do segundo elemento (índice 1) e vão até o penúltimo elemento (índice tamanho – 2). Isso porque não queremos escrever os caracteres em branco nas bordas.

Além do procedimento de inicializar a matriz, o código do Labirinto tem um procedimento chamado **imprimir()** que imprime a matriz, ou seja, que percorre seus elementos e os imprime na tela. Por fim, para podermos rodar o jogo, precisamos do procedimento **main()** já mostrado, que inicializa a matriz com o tamanho certo e que “chama” os procedimentos de inicializar e de imprimir a matriz.

Atividade 01

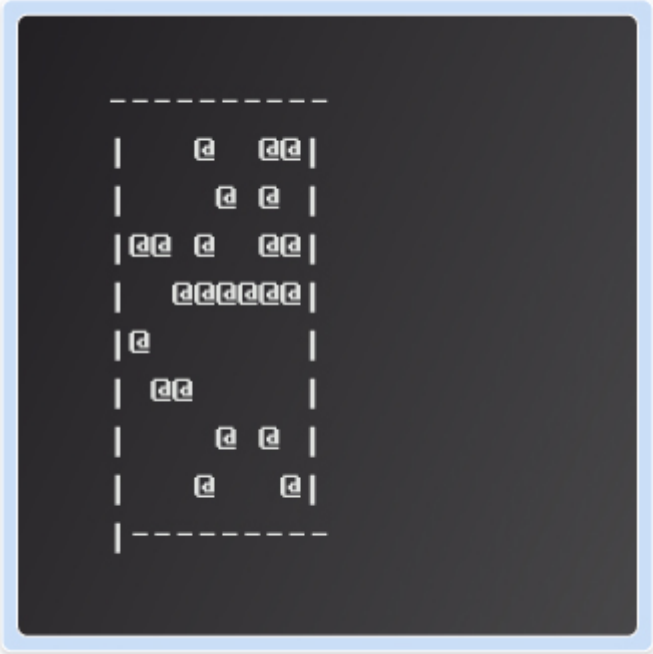
---

1. Implemente o programa Labirinto, de acordo com a versão mostrada, só para apresentar o tabuleiro vazio. Tente não copiar o código mostrado, mas fazer sem olhar o material da aula. Relate o que você fez de diferente em termos de implementação ou se a sua implementação ficou igual à mostrada na aula.

2. Altere o valor da constante que representa o tamanho do tabuleiro. Relate o que houve com o tabuleiro mostrado pelo programa e explique como a alteração dessa constante afetou o resto do programa.

Muito bem, temos com os caracteres “|” e “-” um delineamento da extensão do tabuleiro do jogo, porém, o mesmo encontra-se em branco. Precisamos criar paredes internas dentro do tabuleiro, certo? Que tal usarmos o caractere “@” para criar elementos do tabuleiro que formam as paredes internas do jogo no tabuleiro? Dessa forma, teremos algo como:

Figura 02 - Tabuleiro do jogo, incluindo-se paredes internas (obstáculos)



Agora, veja que o tabuleiro possui espaços em branco que formam os possíveis caminhos do labirinto e os “@” indicando obstáculos, ou seja, paredes que impedem a passagem da pessoa no labirinto. Porém, como podemos fazer a escolha dos locais a serem colocados esses caracteres “@”? Existem algumas formas, como deixar fixo no código, deixar o usuário escolher onde colocar os obstáculos internos ou posicioná-los de forma aleatória. Vamos escolher esta última opção?

O posicionamento aleatório dos caracteres “@” é basicamente colocar esses caracteres em posições diferentes todas as vezes que executarmos o programa Labirinto. Para isso, precisamos alterar o código inserindo primeiro uma constante para representar o caractere da parede interna (obstáculo).

```
1 private static final char PAREDE_INTERNA = '@';
```

Agora, para posicionar esse caractere de forma aleatória, podemos fazer a seguinte alteração no procedimento *inicializarMatriz()*. Alteramos o código que coloca os caracteres vazios no tabuleiro, passando de

```
for (i = 1; i < TAMANHO - 1; i++) {
   for (j = 1; j < TAMANHO - 1; j++) {
      tabuleiro[i][j] = VAZIO;
   }
}
```

para o seguinte trecho de código:

```
for (i = 1; i < TAMANHO - 1; i++) {
   for (j = 1; j < TAMANHO - 1; j++) {
      if (Math.random() > PROBABILIDADE) {
         tabuleiro[i][j] = PAREDE_INTERNA;
      } else {
         tabuleiro[i][j] = VAZIO;
      }
   }
}
```

Note a inclusão de um comando condicional. Isso porque agora nem sempre uma posição interna do tabuleiro será inicializada com o caractere vazio (“ ”), pois pode agora receber o caractere de parede interna (“@”). A expressão booleana usada no comando condicional *if* faz uso da função **Math.random()**. Essa função retorna um número aleatório entre 0 e 1. Se você usar essa função 50 vezes, ela provavelmente irá lhe retornar 50 números distintos entre 0 e 1. Comparamos então esse valor gerado aleatoriamente com uma constante probabilidade definida da seguinte forma:

```
private static final double PROBABILIDADE = 0.7;
```

Comparar o valor gerado aleatoriamente com o valor 0.7 quer dizer que quando o número gerado for maior que 0.7, a posição em questão da matriz receberá um caractere que representa uma parede interna. Caso contrário, receberá um caractere vazio. O valor 0.7, na prática, determina que a expressão irá ser verdadeira em apenas 30% das vezes. Isso porque os números que são maiores que 0.7 representam 30% dos possíveis valores que podem ser sorteados. Como resultado, aproximadamente 30% dos caracteres do tabuleiro serão do tipo parede interna (obstáculo) e os demais serão espaços vazios.

Vídeo 03 - Tabuleiro 2/3

Vídeo 04 - Tabuleiro 3/3

Atividade 02

1. Altere sua implementação do programa Labirinto para apresentar o tabuleiro com as paredes internas (obstáculos), como mostrado nessa segunda parte da aula. Tente não copiar o código mostrado, mas fazer sem olhar o material da aula. Relate o que você fez de diferente em termos de implementação ou se a sua implementação ficou igual à mostrada na aula.

2. Execute várias vezes o programa Labirinto e relate se houve mudanças na posição e quantidade de obstáculos que formam as paredes internas do labirinto.

3. Teste o uso de outros caracteres para as constantes usadas no programa e indique se você encontrou valores mais adequados que os mostrados na aula (deixam o tabuleiro mais bonito, são mais simples de serem visualizados etc.).

4. Altere o valor da constante PROBABILIDADE e relate o que acontece quando aumentamos ou diminuímos o seu valor.

3. Resumo

Nesta aula, você aprendeu a desenvolver os recursos básicos de um jogo de labirinto desenhado na tela através de caracteres especiais. Os caracteres especiais foram definidos através de constantes, o que facilita sua troca por outros. Além disso, você aprendeu a utilizar uma função de geração de número aleatório em Java para montar o tabuleiro colocando obstáculos (paredes internas) de forma aleatória.

Para a próxima aula, você verá o desenvolvimento da funcionalidade de busca do caminho no labirinto através de recursão. Até lá!

4. Autoavaliação

1. Qual a estrutura de dados que se pode utilizar para representar o tabuleiro do jogo de labirinto?
2. Como podemos preencher os obstáculos (paredes internas) no tabuleiro do jogo, de forma que toda vez que se jogue, um novo tabuleiro seja mostrado?

5. Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores**: algoritmos, Pascal, C/C++ e Java. São Paulo: Editora Pearson, 2008.

THE JAVA tutorials. Disponível em: <http://download.oracle.com/javase/tutorial/>. Acesso em: 6 dez. 2011.