Centro Universitário de Belo Horizonte

2019

Relatório:

Simulador de Aeroporto

Aluno: Douglas José Tertuliano dos Santos

RA: 11513928

Ciência da Computação

douglasjtds@gmail.com

douglasjtds.github.io

1 – Introdução:

Este trabalho tem como objetivo simular os padrões de aterrissagem e decolagem em um aeroporto, conforme descrito pelo professor Lucas Schmidt em seu edital com instruções. Abaixo segue roteiro e descrição do que foi construído e como foi feito.

Foi utilizada a linguagem C# para construção de um *Console Application* com o .NET *Framework* 4.7.2. Para compilar e construir esse projeto foi utilizada a IDE *Visual Studio* 2019 na sua edição *Community* distribuída gratuitamente pela *Microsoft*.

2 – Descrição:

Foram utilizadas as classes de negócio Aviao.cs e Pista.cs; além das classes de domínio FilaEnum.cs e PistaEnum.cs que são enumeradores; e para a construção de toda a lógica onde foram criadas as funções e chamados os métodos, foi utilizada a classe Program.cs.

Cada classe Pista, tem as suas filas de pouso e decolagem de avião, sendo que a pista 3 é prioritariamente usada para decolagem.

Logo no início do programa são declaradas as seguintes variáveis globais:

```
#region [variáveis globais]
private static int numeroMaxIteracao = 100000;
private static int idMaximoAterrissagem = 1;
private static int idMaximoDecolagem = 2;
private static bool pousoEmergencialPista3NaInteracao = false;
private static bool aviaoDecolouPista1 = false;
private static bool aviaoDecolouPista2 = false;
private static List<Tuple<int, int, int?>> avioesCaidos = new List<Tuple<int, int, int?>>();
private static List<Tuple<int, int, int?>> avioesDecolados = new List<Tuple<int, int, int?>>();
private static List<Tuple<int, int, int?>> avioesPousados = new List<Tuple<int, int, int?>>();

private const int idPista1 = 1;
private const int idPista2 = 2;
private const int idPista3 = 3;

private static int iteracaoAtual = 1;
#endregion
```

Entramos então no "*core*" principal da aplicação, o método Main, onde são instanciadas as pistas 1, 2 e 3 passando seus determinados Ids conforme constantes na imagem acima. Temos então, ainda dentro do *Main*, a principal estrutura de repetição do programa, onde são chamadas todas as funções: um *while* que dura enquanto a iteracaoAtual for menor ou igual ao numeroMaxIteracao, sendo que o valor do número máximo de iterações (numeroMaxIteracao) pode ser alterado nos testes de acordo com a necessidade.

Essa estrutura ficou da seguinte forma:

```
    while (iteracaoAtual <= numeroMaxIteracao)
    {
        pousoEmergencialPista3NaInteracao = false;

        InsereAviao(pista1);
        InsereAviao(pista2);
        InsereAviao(pista3, false);

        ProcessarIteracao(pista1);
        ProcessarIteracao(pista2);
        ProcessarIteracao(pista3);

        BaixarNivelGasolina(pista1);
        BaixarNivelGasolina(pista2);

        iteracaoAtual++;
        ImprimirPeriodicamente();
    }
}
```

O método **InsereAviao** é chamado para as três pistas, sendo que pra pista 3 passa como *false* a variável gerarAvioesFilaDePouso, porque essa pista só será usada para pouso em caso de emergência.

Logo em seguida, é chamado o método **ProcessarIteracao**, também para as três pistas.

Depois de processada a Iteração para todos os métodos, é chamado um método que vai reduzir em (-1) o nível da gasolina de todos os aviões, em todas as filas, exceto a pista 3 porque ela é usada para pouso apenas em caso de emergência, logo, não tem uma fila específica para aterrissagem. Esse método é o **BaixarNivelGasolina**.

Explicarei esses três principais métodos, mais especificamente, logo abaixo.

2.1 – InsereAviao:

O método **InsereAviao** vai percorrer todas as filas de decolar e de pousar, utilizando um *foreach*, para gerar uma quantidade aleatória de aviões que pode ser de 0 a 3 por iteração para cada fila. Isso é feito dentro do *foreach*, chamando o método **GeraAvioes** onde cada um dos aviões recebe um idAviao, e caso for um id par será um avião decolando e ids ímpares, serão aviões pousando.

2.2 – ProcessarIteracao:

O método **ProcessarIteracao**, primeiro verifica se tem aviões com gasolina 0 em cada fila de cada pista (que seriam os aviões caídos) na iteração, chamando o método **VerificarAvioesCaidos,** e se tiver, adiciona na lista de avioesCaidos e removendo da fila anterior com o **RemoverAviaoCaidoFila**. Depois, o **ProcessarIteracao** confere se vai ser preciso realizar pouso de emergência na pista 3, chamando o método **RealizarPousoDeEmergenciaPista3**. Se for o caso, o **ProcessarIteracao** também realiza pouso de emergência na pista atual, chamando o método **RealizarPousoEmergencialPistaAtual**, mas só se a pista 3 já estiver sendo usada.

Continuando o **ProcessarIteracao**, temos um *switch case* que é onde é feita a lógica para **evitar a queda de aviões**. Para cada pista, o programa verifica se foi realizado pouso de emergência na pista atual, porque se tiver sido, a pista está ocupada e nem pode ser utilizada. Se não tiver pouso de emergência na pista atual ele segue com novas verificações. Foi construída uma lógica, para sempre que possível, alternar entre decolagens e pousos salvando o estado da última iteração. Para isso, é feita a verificação de qual foi o estado anterior e decidir remover avião da fila de pousar ou de decolar.

Se for pouso, como cada uma das duas primeiras pistas tem duas filas de pouso, foi feita a verificação para remover prioritariamente da fila que o primeiro avião tiver com menos gasolina, evitando assim que fiquem aviões com baixa gasolina por muito tempo na fila.

E para a pista 3, verifica o que a função **RealizarPousoDeEmergenciaPista3** salvou para a variável pousoEmergencialPista3NaInteracao para saber se vai precisar de pouso de emergência. Se não precisar, a pista é utilizada para decolar algum avião da fila de decolagem dessa pista.

2.3 – BaixarNivelGasolina:

O método **BaixarNivelGasolina** é bem simples. Toda vez que ele é chamado, ele roda para a pista 1 e para a pista 2, utilizando dois *foreach* que vão percorrer cada uma das filas dessas pistas abaixando em (-1) o nível de gasolina de cada avião. Isso é feito porque foi passada uma iteração sem tirar esses aviões das filas.

3 – Resultados:

Foram feitas algumas observações após vários testes dos resultados do simulador. Esses testes foram feitos alterando o valor da variável numeroMaxIteracao entre **100**, **1.000**, **10.000** e **100.000** iterações.

Inicialmente, a lógica utilizada no *switch case* do **ProcessarIteracao** era um pouco diferente do que foi explicado na seção **2.2**, o que acabou gerando resultados desagradáveis observada a grande queda de aviões. Esse tratamento era feito de forma que alternava a pista que seria pousado algum avião de acordo com as maiores filas, para tentar manter um padrão de tamanho das filas. E com isso era obtido um resultado insatisfatório de aviões caídos, conforme tabela abaixo:

numeroMaxIteracao	avioesDecolados	avioesPousados	avioesCaidos
100	173	123	1
1.000	1.810	1.109	6
10.000	18.201	11.103	130
100.000	175.804	122.945	646

Foi feita então uma refatoração de código no método de **ProcessarIteracao** para deixa-lo da forma que é explicado na seção **2.2**, deixou-se de levar em conta o tamanho das filas e passou a escolher de acordo com a fila que tivesse o primeiro avião com o menor nível de gasolina. E com isso, após vários testes semelhantes aos anteriores, passou a não cair nenhum avião mais. Conforme tabela abaixo:

numeroMaxIteracao	avioesDecolados	avioesPousados	avioesCaidos
100	198	102	0
1.000	1.977	937	0
10.000	19.846	9791	0
100.000	198.199	101.259	0