Uma Abordagem Paralela na Implementação do Jogo "Asteroids"

Victor Pedro C. da Silva

victorp@r7.com

Departamento de Ciência da Computação - Instituto Multidisciplinar - Universidade Federal Rural do Rio de Janeiro

1. Introdução

O jogo asteroids, lançado em 1979, é composto de um mapa simples onde existem vários asteróides e uma espaçonave, cujo objetivo é destruir os asteróides sem sem atingido por nenhum.

Neste trabalho, foi abordada uma técnica diferente de implementação do jogo com o objetivo de melhorar seu desempenho.

2. Programação do Jogo

Neste trabalho, o jogo foi implementado utilizando o pacote AWT da biblioteca padrão da linguagem Java e também o pacote Swing para confecção da janela do jogo.

2.1. Orientação a Objetos

Para facilitar o entendimento e desenvolvimento do jogo, cada elemento que aparece na tela é um Objeto Java, definido pelas suas respectivas classes.

A primeria classe, *Entity*, contém todas as informações comuns nos dois tipos de elementos (Espaço-nave e Asteróide). Esta classe foi desenhada desta forma para que não houvesse repetição de código e possibilitar, também, a uilização de polimorfismo na implementação dos métodos que manipulam os atributos de cada elemento na tela.

As outras duas classes são *Ship* e *Asteroid*. Nestas classes estão os métodos específicos de atualização das informações de cada nave e o método de tratamento de colisões.

2.2. Lógica de Programação

O jogo está dividido, basicamente, em duas classes principais, de elementos: *Ship* e *Asteroid*. Em cada um, há informações relacionadas à fisica do jogo, tais como aceleração, rotação, velocidade, etc. Desta forma, a atualização destes dados é feita pelo próprio objeto, evitando assim uma sobrecarga na classe do jogo e um encapsulamento destas informações.

Para atualizar estas informações, foram criados dois métodos: **update()** e **handleColision()**, que serão descritos com mais detalhes abaixo.

2.2.1. Método update()

Este método é responsável por fazer o cálculo que define a próxima posição da espaço-nave dentro do mapa do jogo. Ele também atualiza as posições dos projéteis atirados pela espaço-nave.

Ele é chamado pela classe principal do jogo, que é responsável pelo *game loop* e é totalmente desacoplado desta classe, separando, assim, a lógica e os cálculos do jogo das operações mais básicas de manter o *loop* rodando

2.2.2. Método handleColision()

Este método recebe como parâmetro outro objeto do tipo **Entity**, verifica o tipo deste outro objeto e toma as ações necessárias.

Tanto este método quanto o método **update()** têm sua lógica e dados independentes do fluxo principal do programa

3. Paralelização

Para distribuir o código do jogo em threads foi utilizado o conceito de Thread Pooling, onde o programa principal tem um conjunto pré-definido de threads e as tarefas vão sendo alocadas nestas threads pré-definidas de acordo com a necessidade do programa.

3.1 Definição das Tarefas Paralelas

Foram escolhidos, para ser paralelizados, os dois métodos descritos anteriormente, **update()** e **handleColision()**, pois estes métodos são executados uma vez para cada elemento na tela, isto é,

dependendo do nível de dificuldade e da quantidade de asteróides na tela, estes métodos podem ser executados um número considerável de vezes e em sequência. Por isso o objetivo deste trabalho é fazer com que estes métodos, em cada objeto, sejam executados simultaneamente.

3.2. Programação Paralela em Java

A linguagem Java, fornece, em sua biblioteca padrão, ferramentas para a utilização de threads e paralelismo. Vamos descrever alguns conceitos:

- Thread: É um executor de tarefas que as executa sempre paralelamente e quando seu método **start()** é chamado.
- Callable: É uma interface que contém um método chamado **call()**. Esta interface garante à thread um padrão de execução. Ou seja, todas as classes que têm tarefas que podem ser executadas em paralelo, devem implementar a interface **call** e seu método **call()**. E é neste método que fica contida a lógica do programa que deve ser executada em paralelo.
- Executor e ExecutorService: Estas classes fornecem uma implementação de Thread Pooling pronta, o que facilita o desenvolvimento da aplicação e agiliza o processo de criação e utilização de threads.

3.3. Implementação

A paralelização dos métodos **update()** e **handleColision()** foi feita através de dois *wrappers*, ou seja, uma classe que encapsula um objeto do tipo **Ship** ou **Asteroid** e implementa a interface **Callable()**. Dentro do método **call()** de cada um destes *wrappers* está a chamada do método **udapte()** ou **handleColision()**.

Isto foi implementado desta forma pois depois de ser executada em uma thread, o objeto “morre” e é recolhido pelo Garbage Collector, porém, para que o jogo continue rodando é necessário que estes objetos continuem existindo, principalmente a espaço-nave do jogador.

Os *wraperrs* são as classes **ConcurrentHandleColision** e **ConcurrentUpdater**.

E graças à forma como o jogo foi implementado, não foi necessário tratar nenhuma concorrência ou condição de corrida, nem a utilização de semáforos, pois todas as operações feitas nos métodos paralelos afetam apenas os atributos de seu próprio objeto.

4. Resultados

Para avaliar o desempenhos dos programas sequencial e paralelo foi utilizada uma métrica que permite calcular o tempo de execução em milissegundos.

Porém esta métrica se mostrou ineficiente pois a diferença de tempo de exeução do programa sequencial para o programa paralelo foi insignificante.

5. Conclusão

Apesar de parecer uma implementação mais sofisticada e que trará vantagens para o desempenho da aplicação, nem sempre as threads ainda não são uma “bala de prata” que resolverá os problemas de desempenho de qualquer programa.

Há também a necessidade de desenvolver e utilizar métricas mais apuradas para comparação do desempenho de aplicações sequenciais e paralelas.

6. Referências

Vogella.com,. Java concurrency (multi-threading) - Tutorial. Disponível em: <http://www.vogella.com/tutorials/JavaConcurrency/article.html>. Acesso em: jul. 2015.

Docs.oracle.com,. ExecutorService (Java Platform SE 7). Disponível em: <http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html>. Acesso em: jul. 2015.