Instituto Superior Técnico

MMA / MEEC

Programação Orientada por Objectos
Projecto 2017/2018

Catarina França Martins Nabais	78366
Filipe Baptista	78119
Kyrylo Yefimenko	78270
Madalena Nabais	78419

May 11, 2018

1 Introdução

O objetivo deste projeto é implementar, em Java, uma solução para o seguinte problema:
Dada uma grelha $n \times m$ e dada uma população, encontrar o individuo que faça o melhor caminho, i.e o caminho com menor custo entre um ponto inicial e um ponto final.

Ao percorrer a grelha, cada individuo poderá encontrar obstáculos e zonas de custo especial, influenciando assim o seu caminho e o seu custo final, respetivamente. Caso nenhum dos indivíduos tenha atingido o ponto final, seleciona-se o que apresenta melhor conforto (parâmetro atribuído a cada indivíduo).

Existem três tipos de eventos associados aos indivíduos que condicionam a sua evolução ao longo do tempo. Movimento, reprodução e morte.

No movimento, o indivíduo desloca-se para uma das suas posições adjacentes de forma equiprovável, para que assim o seu caminho seja o mais aleatório possível.

Na reprodução, o indivíduo "pai" gera um novo indivíduo "criança", cujo caminho será sempre um prefixo do caminho do indivíduo "pai".O tamanho inicial do caminho do indivíduo "criança" é constituído por 90% do tamanho do caminho do indivíduo "pai" e 10% do conforto do indivíduo "pai". Ao criar o indivíduo "criança" também lhe associamos três eventos: a sua morte, a sua primeira reprodução e o seu primeiro movimento.

Na morte, o indivíduo é removido da grelha.

Cada evento é agendado numa queue, onde o seu tempo de realização é gerado por uma variável exponencial aleatória com valor médio $(1 - \log(1 - \varphi(z)))\mu$, $(1 - \log(\varphi(z)))\rho$, $(1 - \log(\varphi(z)))\delta$, correspondendo respetivamente aos tempos dos eventos morte, reprodução e movimento e onde $\varphi(z)$ é o conforto do individuo z.

Em cada simulação deste problema, a queue de eventos será percorrida até deixar de ter elementos, uma vez que os eventos de cada individuo só são adicionados à queue se o seu tempo for inferior ao instante final.

Há de certo mais do que uma maneira de resolver o problema proposto e este relatório serve para apresentar algumas observações e escolhas feitas para a solução implementada.

2 Estrutura do Projeto

O projeto está separado em seis pacotes: **grid**, **main**, **maths**, **pec**, **population** e **simulation**. O pacote **grid** tem tudo o que se relaciona com a grelha, uma interface, que se implementou usando a classe Map e uma classe Point. O pacote **main**, contém apenas a classe Main que importa o pacote **simulation**. O pacote **maths**, que tem somente a classe QuickMaths (para facilitar a realização de contas) que importa os pacotes **simulation** e **population**.
O pacote **pec** tem todas as classes que se relacionam com os eventos. O pacote **population** tem duas classes, uma relacionada com os indivíduos (Individual) e outra que é o conjunto de todos os indivíduos, a Population. Por fim , o pacote **simulation**, onde está situada a classe que une todos os elementos do projeto, a Simulation, que para além de importar os pacotes relacionados diretamente com o problema (**grid**, **pec** , **population**, **maths**) ainda contém a classe Parser, que lê os parâmetros necessários para a concretização da simulação.

3 Observações

3.1 Eventos

Sendo que no enunciado do projeto estão descritos apenas três eventos para os indivíduos: movimento, reprodução e morte, pode ser confusa a implementação escolhida na PEC, apresentar mais um tipo de evento, o EventPrint. Este refere-se ao print das observações de $\tau/20$ em $\tau/20$ unidades de tempo. Esta escolha foi feita uma vez que, tal como nos eventos relacionados com os indivíduos, este print tem sempre um tempo associado no qual se concretiza. Assim, dividindo os eventos em eventos relacionados com indivíduos e eventos de print consegue-se ter todos na PEC onde, posteriormente serão executados de forma sequencial.

É também de notar que embora os eventos EventPrint não estejam relacionados com indivíduos, também se contam para a variável número de eventos (**numberOfEvents**) apresentada no print das observações.

3.2 Decisões de otimização

3.2.1 Grid

Foi criada uma interface Grid, de forma a que a grelha possa ser implementada de formas diferentes. Nesta solução, gera-se a grelha como um HashMap de forma a tirar proveito das suas diversas vantagens, como acesso não sincronizado

e o tempo das operações básicas ser constante (no caso de HashMap, os métodos **put** e **get**).

Estas vantagens destacam-se ao movimentar um individuo pois, basta aceder ao ponto em que está e exercer o método **get(Object key)**, de forma a obter as suas adjacências e escolher de forma equiprovável o próximo ponto no seu caminho, ao contrário de por exemplo, ter de percorrer uma lista até encontrar o ponto que corresponde à atual posição do individuo e de seguida retirar as suas adjacências.

Ao implementar a grelha, excluí-se de imediato os obstáculos e as suas arestas, de modo a que a lista de adjacências só inclua os pontos possíveis para os indivíduos se moverem. Ao contrário de ir verificar a cada movimento se o próximo ponto escolhido seria obstáculo ou não.

3.2.2 Inserção/Procura de Indivíduos

A procura dos indivíduos que sobrevivem à epidemia poderia ser um processo bastante demorado em termos computacionais, visto que possivelmente teríamos de percorrer a lista dos indivíduos toda cinco vezes para obter os cinco indivíduos com o melhor conforto, isto se a procura fosse sequencial. Escolheu-se não ter sempre esta lista organizada e fazê-lo somente quando ocorre uma epidemia, organizando de forma decrescente a lista dos indivíduos por conforto de modo a retirar os cinco primeiros.
Esta decisão foi tomada porque ao longo da simulação há mais reproduções e movimentos que epidemias, portanto a inserção contínua numa lista ordenada (no caso da reprodução) e a reorganização da lista a cada movimento (uma vez que sempre que um individuo se movimenta o seu conforto é recalculado) será, à partida, mais dispendiosa em termos de tempo computacional do que a ordenação de uma lista esporadicamente no programa.

3.2.3 PEC

A simulação dura τ unidades de tempo, logo só podem existir eventos até esse marco. De forma a poupar tempo computacional decidiu-se só adicionar eventos à PEC cujo tempo fosse igual ou menor ao τ, evitando-se, ao executar a simulação, a constante comparação entre o tempo do evento com o instante final, poupando-se também memória já que não se adicionam à PEC eventos que nunca se vão realizar. Relacionado com a poupança de memória, está também a decisão de se excluir da PEC qualquer evento já executado. Assim, a simulação é, de modo simplificado, a execução sequencial de eventos na PEC.

3.3 Decisões para testes

Com o intuito de provar que a solução escolhida é correta e eficiente para qualquer ficheiro **.xml**, criaram-se os seguintes testes:

- **test_1** : uma grelha 7×7 sem obstáculos e sem zonas de custo especial.
- **test_2** : uma grelha 9×8 que contém obstáculos e zonas de custo especial que se sobrepõe. Isto para mostrar que, caso o individuo passe numa aresta onde exista sobreposição de zonas, o custo superior é escolhido.
- **test_3** : uma grelha 10×12 com quatro zonas de custo especial e obstáculos colocados de forma a que o individuo tenha de passar por pelo menos duas das quatro zonas (se atingir o ponto final).
- **test_4** : uma grelha equivalente à gerada no **test_3**, mas usando uma população inicial de 1000, população máxima de 10000 e um parâmetro de morte de 20, com o propósito de testar a eficiência da solução quando estão muitos indivíduos a percorrer a grelha.
- **test_5** : uma grelha 500×600 , com obstáculos e zonas de custo especial geradas aleatoriamente, com a finalidade de testar a eficiência da solução quando é dada uma grelha de grandes dimensões.

4 Conclusões

Tendo em conta o problema apresentado, a implementação sugerida satisfaz todos os objetivos pretendidos para qualquer ambiente. O output deste programa respeita sempre o exigido, no aspeto em que só imprime o necessário e nada mais. O facto de a grelha ser implementada como uma interface torna esta solução versátil pois pode-se alterar com mais facilidade o funcionamento do programa para objetivos diferentes.