Ex 10 (Micro vs macro: um estudo in silico)

Parte B

Alunos:
Giovanni Cangiano n°USP: 10705892
Jonas Rodrigues n°USP: 10734391
Matheus Morroni n°USP: 10350160

Primeiro, este documento tem este nome por termos feito ele depois do dia 23/6.

Processo:

O processo de criação do programa surgiu durante uma conversa com o professor Yoshiharu, inicialmente, nosso grupo estava pensando em fazer um demônio de Maxwell, mas conforme seguiu a conversa, o professor falou que um programa que fizesse as partículas retornarem pro seu estado inicial provavelmente seria algo interessante de ser estudado e rodado. Conversando com outros alunos descobrimos que já existiam outros grupos desejando fazer o demônio de Maxwell e estávamos começando a nos apegar à sugestão feita pelo professor.

O processo de criação inicial foi bem simples, para fazer as partículas retrocederem basta inverter suas velocidades e rodar por determinado tempo. Deste modo, quando a apresentação do programa ia ocorrer, conversamos novamente com o professor e ele nos sugeriu estudar coisas relacionadas ao efeito borboleta, porque assim poderíamos compreender melhor porque alguns casos quando retrocedidos dão errado e outros não.

Seguindo isso, implementamos as ideias que ele nos deu e criamos outras. As principais ideias estão contidas em dois "grupos": efeito borboleta e divergência. A ideia principal do efeito borboleta seria a de analisar duas configurações levemente diferentes e ver quanto tempo demora até que elas passem um certo "limiar" de diferença entre as configurações. Já a ideia central do divergência seria ver por quanto tempo uma configuração consegue rodar, de modo que ela consiga retornar para sua posição inicial com um certo épsilon de êxito.

Resultados:

Como resultado de fazer o retrocesso, o programa funciona bem para casos pequenos e erra em casos com muitas partículas; casos como *billiards4.txt* conseguem rodar por cerca de 8,5 segundos sem que ocorram problemas no processo de retroceder. Fizemos uma seleção de casos que funcionavam relativamente bem, alguns testes mais "pesados" também foram incluídos.

Os resultados das funções relativas ao efeito borboleta ou divergência foram bem interessantes, temos agora um maior controle e informação sobre os sistemas que queremos analisar. Podemos até utilizar os métodos que utilizam dados aleatórios para tentar encontrar algum padrão na mudança de épsilon ou de limiar, para ver se podemos encontrar alguma informação interessante.

Conclusões:

Acreditamos que as simulações que não retrocedem com êxito ocorrem devido um certo efeito borboleta, uma das primeiras ideias que veio à nossa mente foi a de que os erros derivam de arredondamentos, de modo que são acumulados durante o restante da simulação e cada vez mais provocam um distanciamento em relação ao estado inicial. Outro fator que acreditamos que pode influenciar os resultados do nosso ep é o fato do código *CollisionSystem.java* ter um comportamento relativamente estranho em alguns casos, ele assume eventos com valores negativos; acreditamos que isto pode ter algum efeito à longo prazo, de modo que ou a primeira simulação ou a segunda poderiam ocorrer por mais tempo do que o desejado, o que teria um impacto no resultado final.

Execução:

No início do código temos uma rápida explicação, mas ela cobre poucos casos. Simulações do tipo aleatório (ac, at, ag) utilizam partículas criadas aleatoriamente, função já existente em *Particles.java*; testes do tipo padrão (pc, pt, pg) recebem arquivos do mesmo tipo de *CollisionSystem.java* e funções randômicas (rc, rt, rg) são parecidas com as padrões, só que estas não recebem velocidade, sua velocidade é definida aleatoriamente, de modo que se é sorteado um número entre zero e um e este é dividido por um número, que foi passado pelo usuário. Testes que tem o sufixo "c", como: ac, pc e rc, são simulação que retrocedem quando o usuário clica; aqueles com "t" retrocedem depois que a variável "t" assume um valor igual ou maior que um tempo determinado pelo usuário; e aqueles com "g" exibem um gráfico da distância média das partículas em relação à sua posição inicial pelo tempo de execução.

java-introcs Asimov ac número_de_partículas
java-introcs Asimov at número_de_partículas tempo
java-introcs Asimov ag número_de_partículas tempo

java-introcs Asimov pc < entrada.txt
java-introcs Asimov pt divisor tempo < entrada.txt
java-introcs Asimov pg tempo < entrada.txt

java-introcs Asimov rc divisor < entrada.txt
java-introcs Asimov rt divisor tempo < entrada.txt

java-introcs Asimov rg divisor tempo < entrada.txt

Testes do tipo efeito borboleta e divergência são extremamente parecidos, nas suas chamadas. Todos estes testes têm uma variante padrão e uma aleatória; a parte que não foi dita antes é das funções que alteram épsilon/limiar e as do tipo N. As que alteram limiar/épsilon servem para estudar o comportamento dos sistemas quando aumentamos ou diminuímos a exigência dele serem semelhantes, no caso do borboleta, semelhante ao outro sistemas, no caso da divergência, semelhante à sua configuração inicial. Os testes do tipo N são testes que criam N configurações e as rodam, de modo a obter um gráfico com a análise final do comportamento de tais sistemas.

java-introcs Asimov b limiar diferença_entre_os_sistemas delta < entrada.txt
java-introcs Asimov ba número_de_partículas limiar diferença_entre_os_sistemas delta
java-introcs Asimov bl número_de_iterações limiar diferença fator_que_multiplica_limiar delta < entrada.txt
java-introcs Asimov bla número_de_iterações número_de_partículas limiar diferença fator_que_multiplica_limiar delta
java-introcs Asimov bn número_de_iterações número_de_partículas limiar diferença delta

java-introcs Asimov d épsilon delta < entrada.txt
java-introcs Asimov da número_de_partículas épsilon delta
java-introcs Asimov de número_de_iterações épsilon fator_que_multiplica_épsilon delta < entrada.txt
java-introcs Asimov dea número_de_iterações número_de_partículas épsilon fator_que_multiplica_épsilon delta
java-introcs Asimov dn número_de_iterações número_de_partículas épsilon delta