IMD0040 Trabalho 3

Introdução

Este trabalho foi projetado para demonstrar seu entendimento no material apresentado pelo capítulo 10. Além disso, sugerimos a leitura do capítulo 11 antes de começar a fazer o trabalho. Você vai entregar o trabalho eletronicamente via SIGAA.

Este trabalho pode ser desenvolvido utilizando-se de quaisquer ferramentas de desenvolvimento (IDE) que você quiser.

Qualquer método escrito por você deverá ser totalmente comentado utilizando tags javadoc.

Prazo de entrega: 23:59 19 de Outubro de 2017

Plágio e Duplicação de Material

O trabalho que você submeter deverá ser somente seu. Iremos rodar programas verificadores em todos os trabalhos submetidos para identificar plágio, e adotar as medidas disciplinares cabíveis quando for o caso. Algumas dicas para evitar problemas de plágio:

> Um dos motivos mais comuns para problemas de plágio em trabalhos de programação é deixar para fazer o trabalho de última hora. Evite isso, e tenha certeza de descobrir o que você tem que fazer (que não significa necessariamente como fazer) o mais cedo o possível. Em seguida, decida o que você precisará fazer para completar o trabalho. Isto provavelmente envolverá alguma leitura e prática de programação. Se estiver em dúvida sobre o que foi pedido pelo trabalho, pergunte ao professor da disciplina.
> Outra razão muito comum é trabalhar em conjunto com outros alunos da disciplina. Não faça trabalhos de programação em conjunto, ou seja, não utilizem um único PC, ou sentem lado a lado, principalmente, digitando código ao mesmo tempo. Discutam as diversas partes do trabalho, mas não submetam o mesmo código.
> Não é aceitável a submissão de código com diferenças em comentários e nomes de variáveis, por exemplo. É muito fácil para nós detectar quando isso for feito, e iremos verificar esse caso.
> Nunca deixe outra pessoa ter uma cópia de seu código, não importando o quão desesperado eles possam estar. Sempre aconselhe alguém nesta situação a buscar ajudar com o professor da disciplina.

A Tarefa

Este trabalho consiste na implementação de uma simulação caça/caçador.

O cenário é de um ambiente marítimo, consistindo principalmente de água, mas poderá incluir outras áreas que não sejam água (terra, pedras, etc). Neste ambiente existem pelo menos os seguintes participantes: sardinhas (Sardine), atuns (Tuna), tubarões (Shark) e algas. Cada um deles apresenta um comportamento específico.

Seu ponto de partida é um projeto funcional, que apresenta uma simulação contínua do ambiente marítimo, e que produz uma saída gráfica e animada. A simulação é configurada e iniciada a partir da classe `Simulator`. A classe `Ocean` possui uma coleção com todos os atores (sardinhas, atuns e tubarões) que participam da simulação. Alem disto, `Ocean` contém um loop para realizar a iteração nos atores desta coleção, chamando o método act de cada ator.

O sistema reproduz um ciclo completo do ecosistema marinho.

- Todos os peixes comem, se reproduzem e nadam pelo oceano
- Peixes podem morrer de fome ou por idade.
- Sardinhas comem algas, enquanto que os demais peixes são predadores e comem outros peixes.
- Cada localização (célula no mundo) possui um valor de alga na faixa [0..10]. Alga se regenera regularmente, e se espalha, se não for consumida por um peixe.

Sua implementação deverá atender aos seguintes requisitos abaixo. Para tanto, adicione quaisquer outras classes e método que julgar necessário.

- Faça com que as algas apareçam visualmente, sem ocultar os peixes no oceano.
- Cada célula do oceano deve ter a sua quantidade de algas e um peixe, e não todo o oceano; Ela deve impedir que o peixe mova para ela, caso não esteja vazia.
- Cada peixe não deve saber em que célula está;
- Ao invés do método act, cada peixe deve possuir os métodos abaixo. Altere o programa se mudar o comportamento esperado dos peixes.
  - eat (List<Cell> neighborhood)
  - isAlive()
  - move((Cel current, List<Cell> neighborhood)
  - breed(List<Cell> neighborhood)
- Hoje se as sardinhas comerem todas as algas de uma célula, as algas não crescem mais. Altere o programa para que o crescimento de algas tenha 50% de influência da célula atual, e 50% de suas células vizinhas (norte, sul, leste e oeste).
- Hoje, os peixes predadores comem quaisquer outros peixes. Por exemplo, um atum pode comer um tubarão e se entalar; Faça com que os peixe só comam outros menores que eles. Desta maneira, Atuns comem sardinhas. Tubarões comem sardinha ou atum, mas eles preferem atum.
- Tubarões são solitários – eles preferem não nadar próximos um dos outros.
- Elabore um painel de controle para sua simulação. Esse painel de controle deverá permitir ao usuário:
  - Iniciar simulação;
  - Pausar ou resumir uma simulação.
  - Reiniciar uma simulação do começo.
  - Controlar a velocidade que a simulação executa.
  - Modificar alguns dos parâmetros da simulação (ao alterar um parâmetro no painel de controle não precisa alterar os peixes já existentes, mas deverá alterar os parâmetros dos peixes que forem criados a partir desse momento.

Você pode escolher quais parâmetros permitir alteração, por exemplo, o peso (ou nível de comida) com que peixes são criados, a porcentagem de peso que um peixe perde se não comer nada durante um ciclo da simulação, a idade em que um peixe pode procriar ou morrer, etc.

Requisitos de desafio 1

Sardinhas exibem um comportamento de flocking, isto é, tendem a nadar juntos (com algumas exceções ocasionais), tendem a manter uma mesma direção (com algumas exceções). Se elas percebem um predador se aproximando, elas entram em pânico e se espalham.

Requisitos de desafio 2

Persistir o estado atual do projeto continuamente (por exemplo, a cada 10 ciclos), para poder recomeçar deste ponto na próxima vez que o programa for iniciado (ver serialização de objetos). Adicione ao painel de controle uma opção para recomeçar a simulação do último ponto salvo.

Pontuação e Qualidade de código

Parte da nota (30%) será atribuída simplesmente por completar uma solução básica, considerando o design e documentação da sua solução. 30% da nota será baseada no painel de controle. 20% da nota será baseada no requisito de desafio 1. 20% da nota será baseada no requisito de desafio 2.

- Escreva código orientado a objeto. Pense sobre design baseado em responsabilidade, focando em baixo acoplamento e alta coesão.
- Pense bem onde colocar código (ex: classe pai ou sub-classe? Qual a relação entre as classes?). Evite duplicação ou que uma classe saiba muito sobre outra.
- Use os recursos apropriados da linguagem Java quando necessário (ex: pacotes, tipos genéricos, classes abstratas e interfaces, final e enumerações).
- Mantenha um estilo de código consistente.
- Documente suas classes e métodos usando javadoc (veja http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html)

Finalmente

Antes de submeter seu trabalho, teste exaustivamente o projeto inteiro para se certificar de que nada do que foi acrescentado recentemente quebrou algo adicionado anteriormente. Se você não conseguir completar o projeto - mesmo se você não conseguir compilar – submeta o que você tem, pois é provável que você vai ter pelo menos algum ponto por isso.

V2, 02/10/2017