Programação Orientada a Objeto

Game of Life

Rodrigo Bonifácio

19 de setembro de 2017

Game of Life

Proposto pelo matemático John Conway (Princenton University) em 1970, esse não corresponde a um jogo típico ...

Game of Life

Proposto pelo matemático John Conway (Princenton University) em 1970, esse não corresponde a um jogo típico . . .

- Não existem jogadores
- Não existem vencedores ou perdedores

Uma vez que as "peças" são posicionadas, as regras determinam tudo que acontecerá a seguir.

Regras

As células, dispostas em um tabuleiro em forma de grade bidimensional, podem estar vivas ou mortas. Uma célula viva é indicada por uma marca na posição específica do tabuleiro.

Regras

As células, dispostas em um tabuleiro em forma de grade bidimensional, podem estar vivas ou mortas. Uma célula viva é indicada por uma marca na posição específica do tabuleiro. Uma nova geração de células depende da vizinhança de cada célula específica (cada célula possui no máximo 8 células vizinhas).

Regras

As células, dispostas em um tabuleiro em forma de grade bidimensional, podem estar vivas ou mortas. Uma célula viva é indicada por uma marca na posição específica do tabuleiro. Uma nova geração de células depende da vizinhança de cada célula específica (cada célula possui no máximo 8 células vizinhas).

- Uma célula morta com exatamente três células vizinhas vivas se torna uma célula viva (nascimento).
- Uma célula viva com duas ou três células vizinhas vivas permanece viva (sobrevive).
- Em todos os outros casos, a célula morre ou continua morta (superpopulação ou solidão).

Implementação

Usaremos a abordagem *Problem Based Learning* para discutir alguns conceitos relacionados à abordagem de desenvolvimento orientada a objetos.

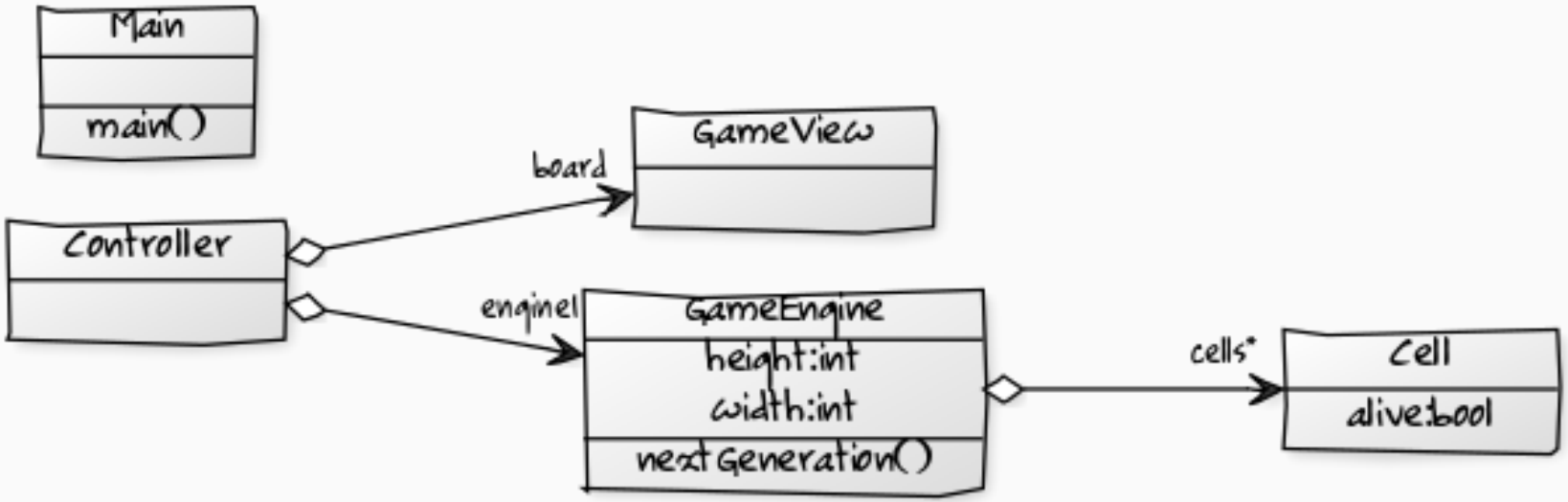
Implementação

Usaremos a abordagem *Problem Based Learning* para discutir alguns conceitos relacionados à abordagem de desenvolvimento orientada a objetos. Para esse tema da disciplina, o conceito mais importante é separação de preocupações, que recomenda a decomposição do software em termos de abstrações coesas, isto é, com responsabilidades bem definidas.

No GameOfLife, temos preocupações relativamente simples.

- elementos de UI (textual)
- elementos do domínio (céluas, ambiente, . . .)
- lógica que define o comportamento do jogo

Desenho da solução



Conceitos reforçados

Lembre-se: O principal interesse é pedagógico

Conceitos reforçados

Lembre-se: O principal interesse é pedagógico

- classes e objetos
- atributos, métodos e construtores
- associação entre classes
- programa principal

As regras do jogo (derivação de uma nova geração) estão definidas no método *nextGeneration()* da classe *GameEngine*. Por outro lado, existem diferentes estratégias de derivação de uma nova geração. Por exemplo, a variação *HighLife* sugere que uma célula morta seja ressuscitada caso tenha seis células vizinhas vivas[1].

[1] Existem várias outras estratégias:
http://www.mirekw.com/ca/rullex_life.html

Diferentes alternativas podem ser usadas para contornar esse problema. Mas, usaremos esse desafio para explorar técnicas de programação em OO, como padrões de projeto, herança e polimorfismo por subtipo— usando como guia duas alternativas consolidadas.

Questão 01 do Trabalho

Conforme mencionado, diferentes alternativas podem ser usadas para flexibilizar a implementação do cálculo das próximas gerações. Entre elas:

- Template Method
- Strategy

Usando o padrão Template Method, tornamos a classe `GameEngine` abstrata. Classes abstratas geralmente possuem pelo menos um método abstrato, não podem ser instanciadas e servem para estabelecer um tipo particular de contrato: as classes concretas que herdam de uma classe abstrata devem prover uma implementação para os métodos abstratos declarados na super classe. Os métodos `shouldRevive` e `shouldKeepAlive` devem ser declarados como abstratos e serem chamados pelo método concreto `nextGeneration` da classe `GameEngine` (caracterizando o padrão Template Method).

No caso do TemplateMethod, a implementação dos métodos `shouldRevive` e `shouldKeepAlive` é concretizada em classes que herdam de `GameEngine`. Note que, para mudar a regra de derivação durante uma partida, precisamos instanciar uma `GameEngine`

Usando o padrão Strategy, deve ocorrer uma extração da definição dos métodos `shouldRevive` e `shouldKeepAlive` para um trait (`EstrategiaDeDerivacao`) e fazer com que a classe `GameEngine` referencie uma instância desse trait. O método que computa a próxima geração faz as chamadas aos métodos `shouldRevive` e `shouldKeepAlive` extraídos para o trait.

A implementação dos métodos `shouldRevive` e `shouldKeepAlive` é concretizada em classes que implementam o trait `EstrategiaDeDerivacao`. Note que, para mudar a regra de derivação durante uma partida, precisamos apenas atualizar a referência a uma estrategia de derivação na classe `GameEngine`—não sendo necessário instanciar novamente essa classe.

Questão 02 do Trabalho

- Implemente uma interface gráfica para o GameOfLife reusando as classes existentes. A interface gráfica pose ser baseada na biblioteca LIBGDX.
- Torne a implementação mais flexível com o uso do padrão *injeção de dependência* (ID), de tal forma que os objetos que implementam os diferentes algoritmos para calcular as regras de derivação sejam injetados no programa (em vez de diretamente instanciados). Pesquise as estratégias para implementar ID na linguagem de programação Scala.

Questão 03 do Trabalho

Implemente uma nova opção de menu que faz com que próximas gerações sejam computadas automaticamente. Observe que a implementação atual não suporta a noção de ambiente infinito (as células próximas aos limites do tabuleiro não possuem oito células vizinhas). Corrija essa falha de implementação.

Questão 04 do Trabalho

- Estude o padrão de projeto `memento` e implemente um mecanismo de UNDO, de tal forma que seja possível retornar (desfazer) n gerações. Assuma $n = 10$.