

Programação Orientada a Objetos

Padrões de Projeto

Game of Life

Rodrigo Bonifácio

22 de maio de 2014

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

Proposto pelo matemático John Conway (Princeton 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.

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.

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).

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).

Neste curso, disponibilizei uma implementação Java semelhante a uma anterior, em C++ e que me tomou algo em torno de 6 horas da noite de um sábado

A implementação contém várias falhas de design

Neste curso, disponibilizei uma implementação Java semelhante a uma anterior, em C++ e que me tomou algo em torno de 6 horas da noite de um sábado

A implementação contém várias falhas de design (propositais).

- Vocês precisam melhorar essa implementação
- Usando como guia alguns padrões de projeto

Neste curso, disponibilizei uma implementação Java semelhante a uma anterior, em C++ e que me tomou algo em torno de 6 horas da noite de um sábado

A implementação contém várias falhas de design (propositais).

- Vocês precisam melhorar essa implementação
- Usando como guia alguns padrões de projeto

Usaremos a abordagem *Problem Based Learning* para discutirmos sobre padrões de projeto.

A decomposição *Model View Controller* da implementação não está adequada.

A decomposição *Model View Controller* da implementação não está adequada. **Desafio:** Reestruturar de acordo com o padrão MVC conforme discutido por:

- Glenn E. Krasner and Stephen T. Pope, *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*, Journal of Object-Oriented Programming, Volume 1 Issue 3, Aug./Sept. 1988

Existe um forte acoplamento entre a classe *GameOfLife* e a classe *Statistics*. Basicamente, a instância da classe *Statistics* precisa ser notificada quando uma célula é morta ou ressuscitada. Mas outros objetos poderiam ter interesse neste tipo de notificação, associada a mudança de estado do jogo.

Existe um forte acoplamento entre a classe *GameOfLife* e a classe *Statistics*. Basicamente, a instância da classe *Statistics* precisa ser notificada quando uma célula é morta ou ressuscitada. Mas outros objetos poderiam ter interesse neste tipo de notificação, associada a mudança de estado do jogo. **Desafio:** Usar o padrão de projeto *Observer* para diminuir esse acoplamento, permitindo que instâncias de outras classes possam se registrar nesse tipo de evento.

As regras do jogo (derivação de uma nova geração) estão definidas no método *nextGeneration()* da classe *GameOfLife*. 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 vivas¹.

¹Existem várias outras estratégias:

http://www.mirekw.com/ca/rullex_life.html

As regras do jogo (derivação de uma nova geração) estão definidas no método *nextGeneration()* da classe *GameOfLife*. 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 vivas¹.

Desafio: Usar os padrões de projeto *Strategy* e *Template Method* para modularizar as regras do jogo, de tal forma que possamos ter diferentes estratégias de implementação. Necessário implementar pelo menos três novas estratégias e comparar as soluções.

¹Existem várias outras estratégias:

http://www.mirekw.com/ca/rullex_life.html

A transição de estado de uma célula está implementada na classe *GameOfLife*, basicamente porque temos apenas dois estados para células (viva / morta). Por outro lado, existem variações do jogo *GameOfLife* que suportam mais estados. Por exemplo, na variação *ImigrationGame*, uma célula ressuscitada tem as mesmas características genéticas das células mais frequentes em relação as suas vizinhas.

A transição de estado de uma célula está implementada na classe *GameOfLife*, basicamente porque temos apenas dois estados para células (viva / morta). Por outro lado, existem variações do jogo *GameOfLife* que suportam mais estados. Por exemplo, na variação *ImigrationGame*, uma célula ressuscitada tem as mesmas características genéticas das células mais frequentes em relação as suas vizinhas. **Desafio:** Usar o padrão *State* para controlar a transição de estados das células, bem como evoluir a aplicação para considera dois tipos de células vivas “o” e “x”.

A implementação não oferece meios para retornar a uma geração anterior. Por outro lado, isso é útil quando desejamos comparar as evoluções entre duas gerações de forma mais precisa. A idéia é implementar essa nova *feature*, de tal forma que o usuário possa retornar *n*-gerações, onde $1 \leq n \leq 5$. **Sugestão:** Usar o padrão de projeto *Memento*, de tal forma que o estado de um objeto seja encapsulado, facilitando operações de *undo*. Implementar também a operação de *redo*.

A interface com o usuário atual é bastante rudimentar, por outro lado, facilita a portabilidade. Gostaríamos de ter uma implementação do *GameOfLife* usando alguma interface gráfica, baseada em frameworks como Java Swing. Esses frameworks utilizam padrões de projeto como *Observer* e *Command* para notificar e tratar eventos. **Desafio:** Implementar uma interface gráfica utilizando algum framework existente, mas isso de forma flexível, coexistindo com a interface baseada em *shell*.

Resolver 4 das seis falhas listadas anteriormente, e apresentar as resoluções como parte da nota da segunda prova (equivalente a 40% da nota)

- 1 A solução final deve considerar um *mundo infinito* e ter a opção de transições sucessivas sem a intervenção do usuário.

- 1 A solução final deve considerar um *mundo infinito* e ter a opção de transições sucessivas sem a intervenção do usuário.
- 2 Pensar em como compor as diferentes resoluções, algumas podem ser inconsistentes, caso compostas em ordens específicas.