UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Licenciatura em Engenharia Informática

Unidade Curricular de
Programação Orientado a Objetos
(2º ano/2semestre – 2019 / 2020)

GAME OF LIFE

Relatório Técnico – Mini Projeto

Docente: José Valente de Oliveira

Discente:

PL1 Grupo nº4

David Fernandes, nº:58604

Guilherme Correia, nº:61098

Bruno Susana, nº:61024

Índice

Índice de Figuras

Glossário

Células partilhadas- são células que circundam mais que uma célula viva

Células vizinhas- são células que estão próximas uma da outra (1 inteiro de diferença)

LinkedListCircular-lista compoasta por nós que apontam para o nó seguinte chegando ao último nó e esse apontar devolta ao primeiro.

Introdução

O presente relatório pretende expor todo o trabalho realizado pelo PL1 grupo 4 relativamente ao mini-projeto: Game of Life.

O objetivo deste trabalho consistiu em recriar uma implementação que simule “Game of life” de John Horton Conway.

Este algoritmo consiste em, dadas várias células com um determinado estado (vivo ou morto) e com uma determinada distribuição inicial num tabuleiro e após várias iterações/gerações é mostrada a distribuição dessas mesmas células passados esse número de iterações/gerações.

As células regem-se por algumas regras que determinam se esta sobrevive de uma geração para a seguinte ou não.

Metodologia

Após uma primeira leitura, o maior dilema deparou-se com o facto de estarmos restringidos à forma de resolução indicada no enunciado, nomeadamente o uso de um array bidimensional esparso que apenas guarda as células vivas, este array esparso é constituído pois dois tipos de nós que são os sentinela e os de dados, formando os nós sentinela uma linked list circular em que quando o ultimo nó é alcançado, este aponta de imediato para o primeiro novamente criando assim um ciclo

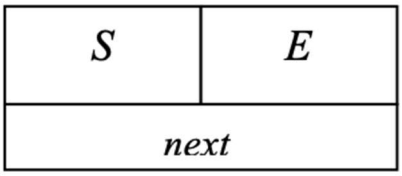


FIGURA 1- SENTINEL NODE

O responsável por apontar para o seguinte nó sentinela na lista é o campo “next” o campo “S” aponta para o primeiro nó de dados de uma determinada coluna ou então para o próprio nó caso não exista nenhum nó nessa mesma coluna por o outro lado o campo “E” faz exatamente o mesmo, mas para as linhas.

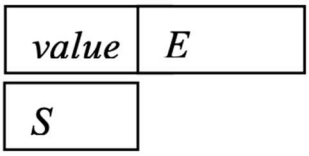


FIGURA 2 - DATA NODE

Relativamente aos nós de dados, possui os campos “S” e “E” que apontam para o próximo nó de dados diferente de zero ou para um nó sentinela em que o “S” é para as colunas e o “E” para as linhas, o campo “value” guarda um valor genérico que neste caso contem uma célula viva.

Material

-javadoc

-java-visual paradigma

-Eclipse/visual studio code

-junit 4

Classe aninhada

Uma classe aninhada é uma classe declarada dentro de uma outra classe externa a si. Permitindo assim agrupar diferentes classes que irão ser utilizadas no mesmo local, a classe aninhada tem acesso aos métodos da classe exterior e às variáveis de tipo primitivo, desta forma uma classe fica mais protegida pois encontra-se encapsulada noutra e cria desta forma cria-se um código de leitura mais fácil e com uma maior facilidade de manutenção.

Interface

Uma **interface** é uma classe abstrata que contém especificações que serão usadas por outras classes. Ou seja, ela tem por objetivo criar um contrato que deverá ser obedecido nas classes onde for implementada. Os métodos criados na interface não qualquer tipo de implementação, apenas é indicado o que deve ser implementado.

Herança

A herança é um mecanismo que permite criar classes a partir de classes já existentes, aproveitando-se das características existentes na classe a ser estendida (superclasse). Este mecanismo promove um grande reuso e reaproveitamento de código existente. As subclasses herdam todas as características das suas superclasses, tais como variáveis e métodos.

Os subtipos, além de herdarem todas as características dos seus supertipos, também podem adicionar mais características, seja na forma de variáveis e/ou métodos adicionais, bem como reescrever métodos já existentes na superclasse.

UML

Primeira Versão de UML

Numa abordagem inicial, fizemos este UML com a classe Cell ligada à Board, sendo a Board a representação da matriz esparsa. Ligamos também a esta classe, a classe List que é sucessivamente aninhada pela classe Node. Por sua vez a classe Node seria superclasse de DataNode e SentinelNode.

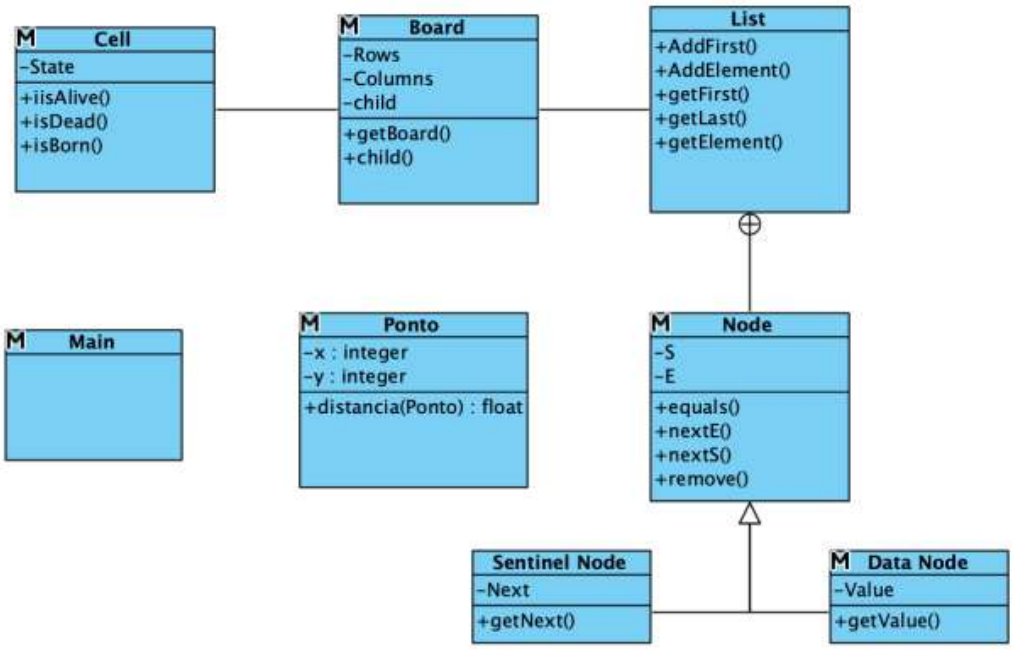


FIGURA 3 - 1º VERSÃO DO UML

Segunda versão do UML

Discutindo o nosso UML, decidimos fazer algumas alterações ao nosso UML, modificando a classe List para CircularList. A classe Board começou a implemetar a interface Matrix. Criamos também a classe GOL.

Classe CircularList:

- addfirst() que adiciona um elemento ao inicio da LinkedList
- addelement(), adiciona um elemento ao longo da LinkedList.
- getfirst(), vai buscar o primeiro elemento
- getlast(), vai buscar o ultimo elemento
- getelement(), vai buscar um elemento especifico

Classe Board:

- rows, representa os nós sentinelas que identificam as linhas
- Columns, representa os nós sentinelas que identificam as colunas
- Child, representa a board subsequente
- Child(), método que faz a próxima geração da board

Classe Node:

- S- Responsável por apontar para o primeiro nó de dados/sentinela de uma determinada coluna
- E- Responsável por apontar para o primeiro nó de dados/sentinela de uma determinada linhas
- Equals(), método que vê se nos são iguais
- remove (), método que remove o no selecionado

Classe SentinelNode:

- next- responsável por apontar para o seguinte nó sentinela

Classe DataNode:

- Value- guarda o ponto que contem uma célula viva.

Classe GOL:

- CreateBoard() – método que cria a board

Classe Cell:

- State- responsavel por guardar o estado, vivo ou morto da célula
- isAlive()- método responsável por ver se a célula, continua viva
- isDead()- método responsável por ver se a célula morre, mas é desnecessária, pois temos o isAlive()
- isBorn()- método responsável para ver se a célula, torna-se viva

Classe Ponto:

- X- Representa a coordenada do eixo das abcissas
- Y- Representa a coordenada do eixo das ordenadas
- distancia- método que calcula a distancia entre dois pontos

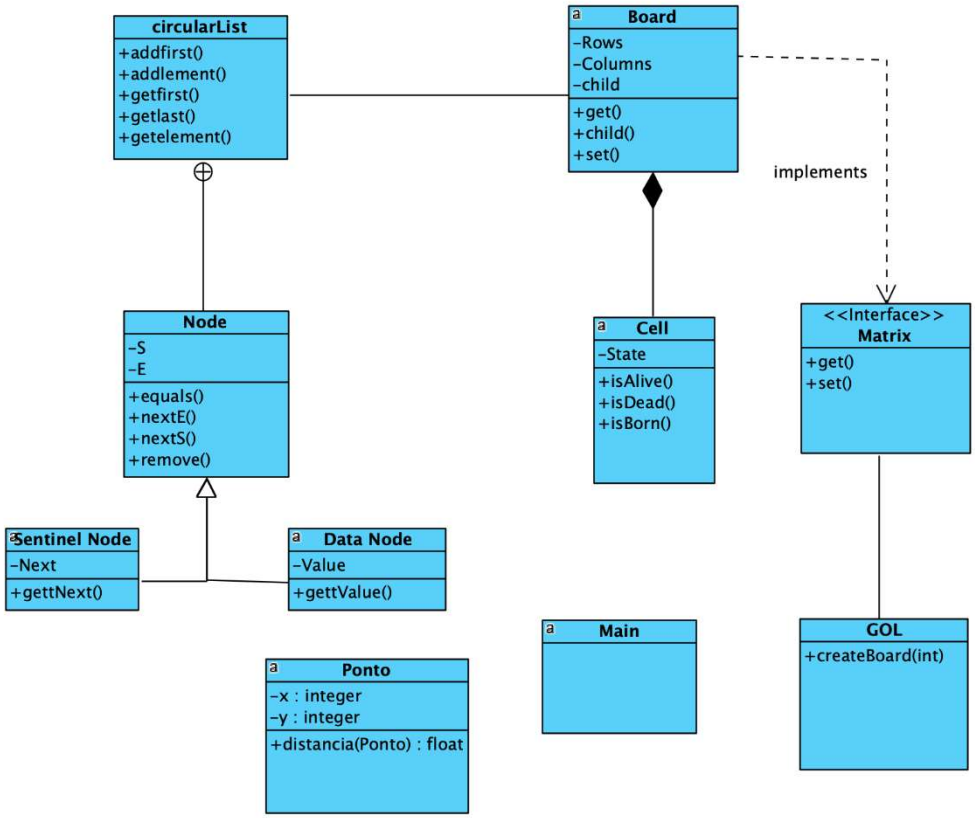


FIGURA 4 - 2º VERSÃO DO UML

UML de implementação

Ao realizar o código, reparamos que certas classes tinham de ser acrescentadas, nomeadamente a classe List. A classe List foi adicionada, para facilitar o trabalho na análise de vizinhos.

Foram adicionados métodos à classe Board e LinkedListCircular, para principalmente não existir a repetição de código e facilitar a leitura.

Os métodos adicionados foram

Classe Board:

- Createsetinels: Cria a estrutura da matriz esparsa, com sentinel nodes
- Neightboors_cell: Atualiza ou adiciona a célula vizinha
- CellAlreadyExists: verifica se já existe essa célula
- newCells: Verifica se existe condições de morrer ou nascer e dependente duma dessas ocasiões, adiciona ou remove da matriz esparsa

Classe LinkedListCircular:

- Col: inicia a LinkedListCircular que se orientara pelos nós na horizontal
- Addline : adiciona um sentinela node à LinkedlistCircular
- addfirstLine : adiciona à LinkedListCircular um sentinel node na primeira posição.
- Getlastline: devolve o último nó da LinkedListCircular
- getindexOf : encontra o index do objecto na linkedlistCircular associado neste a coluna em que se encontra
- getlinha: encontra o nó da linha dada
- getfirstindex: delvove o index do primeiro nó
- getElememementOf: devolve o valor do datanode que esta na coluna e na linha dados
- removeAssociationOf: remove um objeto da linha dada
- associateTo: Associar um objecto a linkedlistCircular linha e coluna consoante o índice dado
- nextcellHorizon: Encontra o no que esta a seguir ao objecto selecionado com orientação horizontal
- nextcellVertival: Encontra o no que esta a seguir ao objecto selecionado com orientação vertical
- contains: Verifica se o objecto selecionado na linha lineindex (orientação horizontal)

Classe List:

- size: dá o tamanho da lista
- clear: apaga a lista
- remove: apaga um elemento da lista
- add: adiciona um elemento à lista
- removeindex: remove o index da lista
- ensureCapacity: adiciona capacidade ao array
- contains: procura a existência de um objecto na lista
- clone: devolve uma copia da lista

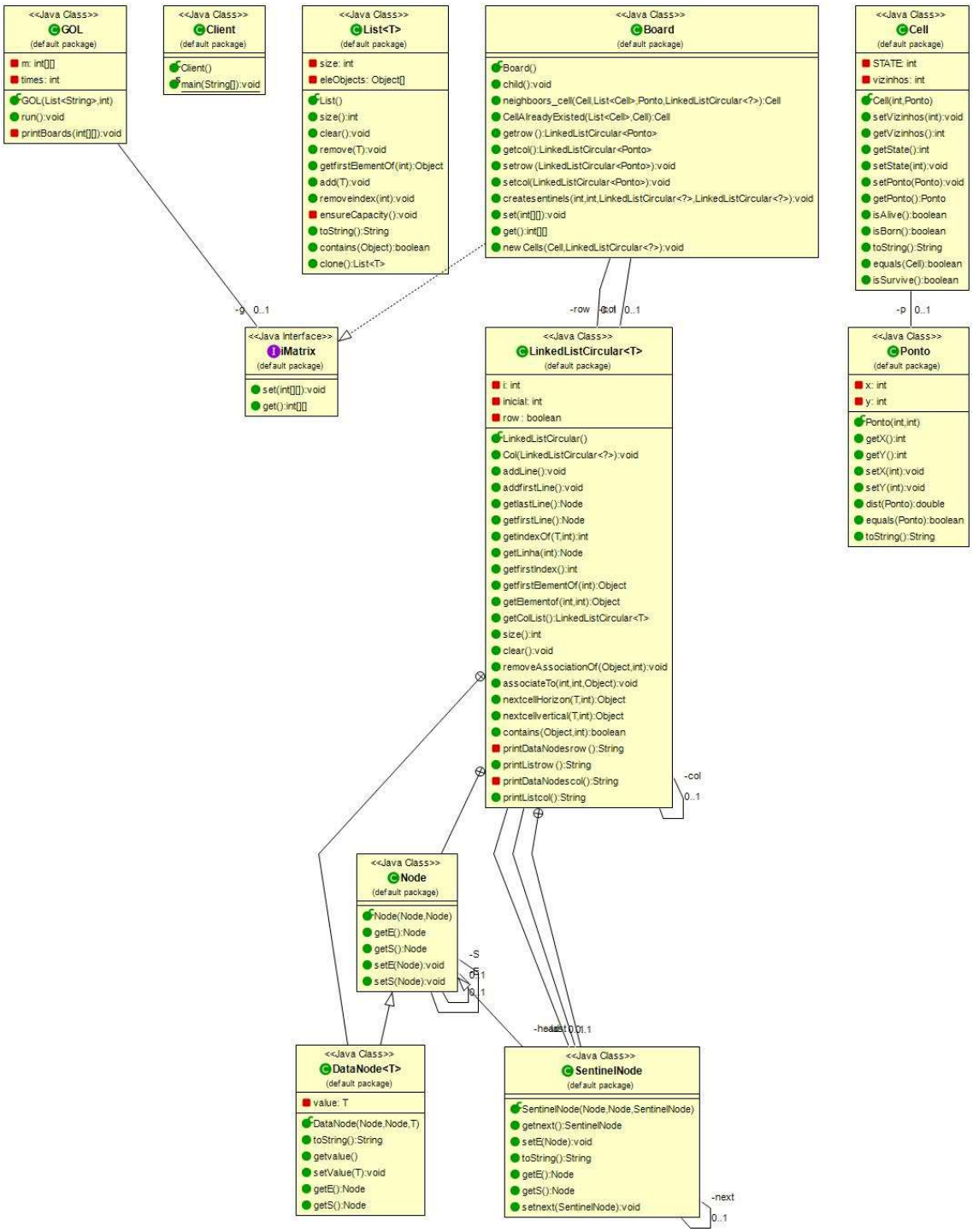


FIGURA 5- UML DE IMPLEMENTAÇÃO

Resolução

Inicio da implementação

Inicialmente foi criada a classe LinkedListCircular, durante o seu desenvolvimento foram realizadas algumas alterações e acrescentados alguns métodos que poderiam ser necessários tanto neste trabalho como para futuros onde poderá ser necessária esta LinkedList para criar matrizes esparsas, nomeadamente as funções Col e o getCol, que são funções que servem para identificar a LinkedList das colunas que esta a associada a LinkedList de forma a que seja mais fácil identificar qual a correspondente.

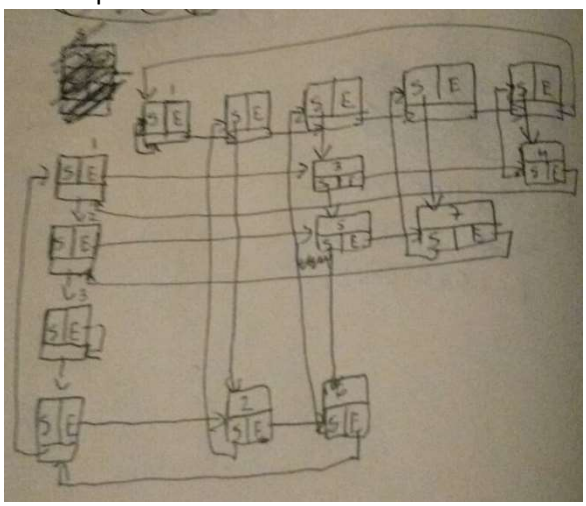


FIGURA 5-REPRESENTAÇÃO DA MATRIZ ESPARSA

Em conjunto, com o desenvolvimento da LinkedList, também estavam a ser desenvolvido as classes aninhadas Node, SentinelNode e DataNode. No início a classe DataNode foi desenvolvida para trabalhar unicamente com as coordenadas das células, para ter uma melhor compreensão do funcionamento da LinkedList. E também foram usadas as coordenadas das células para simplificar a pesquisa de determinada célula na matriz esparsa.

Terminada a LinkedList começou-se a trabalhar nas classes que iriam compor a resolução do problema em si nomeadamente a Cell e a Board. A classe Cell inicialmente tinha os métodos que tínhamos apontado no uml e a classe Board também. Isto também se aplica a classe Ponto, esta que foi reutilizada.

Primeiro método de resolução

A primeira maneira que pensamos para resolver o problema foi identificar as células vivas que se encontravam a Este, Oeste, Norte e Sul da célula viva onde nos encontrávamos, pensando que desta forma facilitaria encontrar as células que sobreviveriam. Na continuação deste método parecia-nos que existia padrões para nascer células.

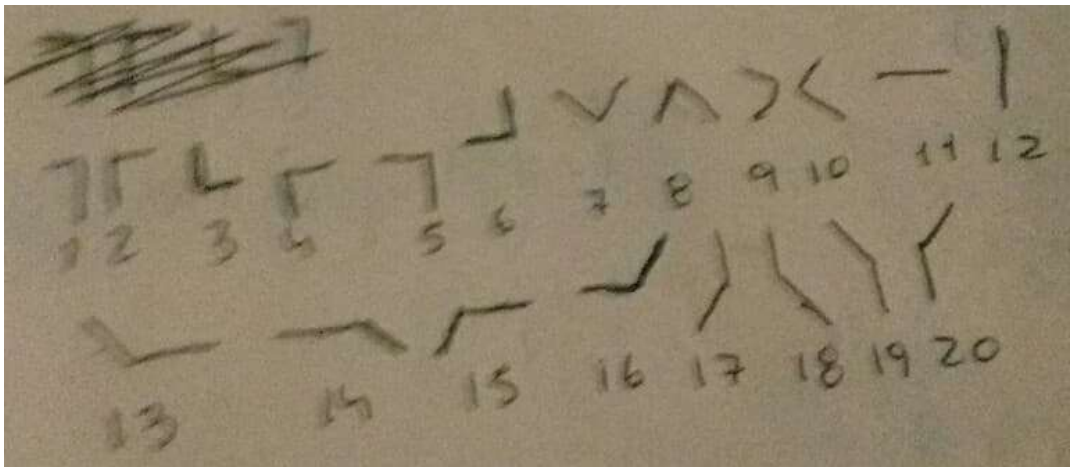


FIGURA 6-COMBINAÇÕES COM CÉLULAS VIZINHAS DE FORMA A NASCER UMA CÉLULA

Depois de identificado e feito o código relativamente às comparações, começou-se a pensar onde poderia nascer uma célula o que nos levou a fazer alguns cálculos de forma a saber mais rapidamente a posição do nascimento da célula.

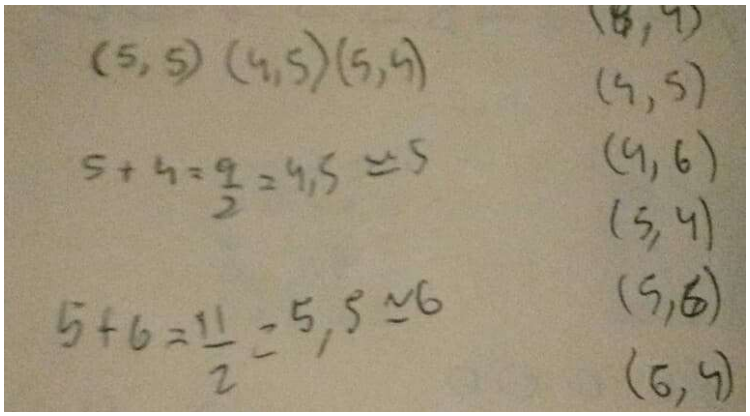


FIGURA 7-CÁLCULO PARA SABER A POSIÇÃO DE NACIMENTO DA CÉLULA

Apos algum tempo reparou-se que existia a possibilidade de nascerem células que se encontravam a 1 linha de distância, isto é, 2 células vivas numa linha e 1 noutra, levando assim a que houvesse ainda mais possibilidades de nascerem células. levando assim a que o código ficasse mais lento devido à quantidade enorme de comparações, de forma a identificar a posição correta das células vivas

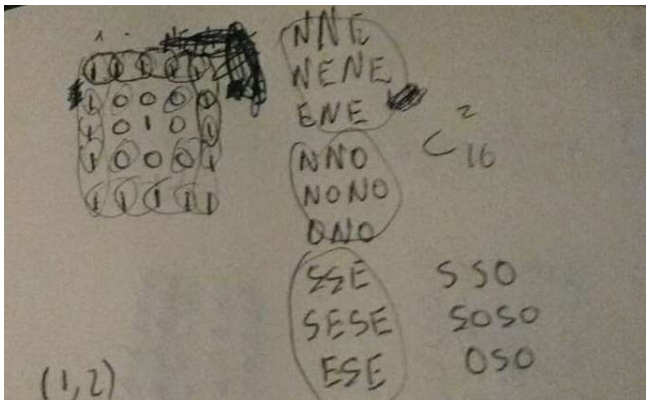


FIGURA 8-COMBINAÇOES COM CÉLULAS A 2 CÉLULAS DE DISTANCIA PARA NACIMENTO DE CÉLULAS

Isto levou a que nós abandonássemos esta forma de resolução.

Segundo método de resolução

Neste método, usando partes da ideia anterior tais como identificar o Norte, Sul, Este e Oeste, mas neste caso simplesmente adicionando 1 ou -1 às coordenadas da célula viva de forma a representar as células que estão em volta. Ao usar este método consideramos que os vizinhos dessa célula viva estão todos mortos.

Depois, vimos que existiam células vivas que por vezes partilham células mortas e desta forma vemos que o número de vizinhos vivos que essa célula morta tem pode ser maior que 1, o que levou a uma conclusão no grupo de arranjar uma maneira de transferir informação de células vizinhas para outras células vizinhas de uma célula viva.

Para a coluna não houve muitos problemas basta saber a distância das células vivas e dependendo de isso passar a informação das células vizinhas partilhadas.

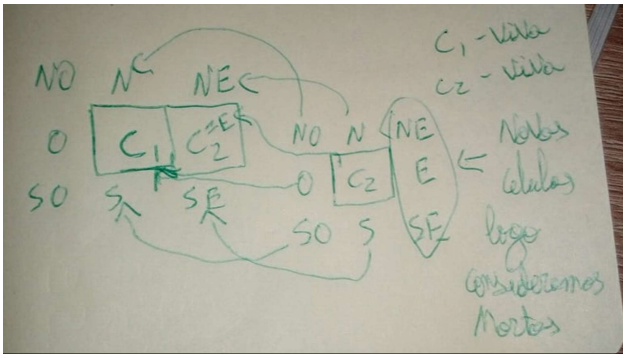


FIGURA 9-TRANSPORTE DE INFORMAÇÃO SOBRE CÉLULAS PARTILHADAS DE COLUNA PARA COLUNA

No caso da linha tínhamos de arranjar uma maneira de guardar as células vivas e vizinhas da linha, o que foi decidido foi usar uma lista de células onde podemos ter acesso à informação de forma rápida e simples. Assim foram criadas 2 listas uma para representar a lista da linha onde me encontro e a lista da linha a seguir onde iria ter só os vizinhos. Esta forma iria fazer o idêntico ao que é feito para ver se as células vizinhas são partilhadas numa linha.

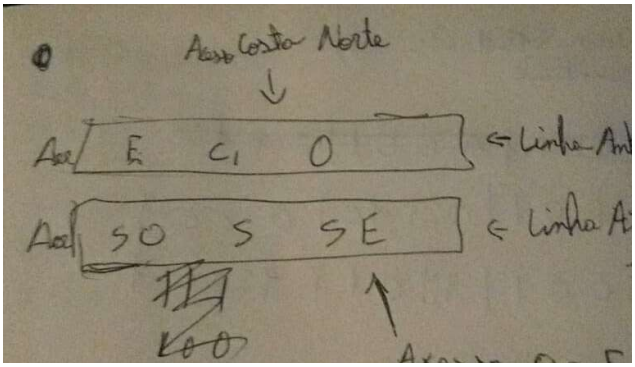


FIGURA 10-TRANSPORTE DE INFORMAÇÃO DE CÉLULAS PARTILHAS DE LINHA PARA LINHA

Para realizar esta parte foi criada uma Lista para armazenar esse conteúdo, com métodos simples e principalmente os que eram necessários para poder funcionar em paralelo com o programa.

Por fim, enquanto era analisada cada célula viva da board do início da iteração era também analisada cada célula vizinha da célula viva que que fosse depois fosse possível saber se a célula vizinha ou a viva iria aparecer na matriz esparsa final da iteração

Casos de uso

Os testes unitários utilizados forma determinantes para verificar os resultados obtidos durante a execução do programa. O ficheiro que contem estes testes chama-se “Tests.java” (Ver no Anexo).
Numa primeira fase criamos testes unitários, para testar todos os construtores criados e algumas funções da LinkedListCircular, entre as quais adicionar elementos, remover e verificar a existência de elementos. Depois deste foi também feito testes as funções da Cell, entre quais as funções de verificar se uma célula era viva, verificar se tem condições para nascer e se 2 células são iguais. Também forma feitos testes as funções da Board principalmente a child() para ver se a geração seguinte era a certa na representação da matriz esparsa. Por fim fizemos teste a classe List com os mesmo estilos de testes que forma feitos para a LinkedListCircular.

Nestes testes unitários utilizamos o método assertEquals() para comparar os resultados obtidos no programa com o verdadeiro resultado esperado.

Conclusão

Em suma, o objetivo do trabalho foi satisfeito, pois conseguimos implementar tudo o que foi expresso no enunciado deste miniprojecto, no entanto não foi implementado como o desejado inicialmente. Este trabalho fez com que consolidássemos o conhecimento adquirido ao longo do semestre na cadeira de Programação Orientada a Objetos, nomeadamente foi necessário a aplicação de vários conteúdos lecionados dos quais o funcionamento das coleções do java, que apesar de não serem usadas a sua linha de construção teve uma ajuda bastante benéfica aquando da implementação da nossa própria lista. Não obstando, também aplicamos outros conceitos, nomeadamente interface e classe aninhada.

Bibliografia

1. http://w3.ualg.pt/~mzacaria/tutorial-uml/index.html
2. http://w3.ualg.pt/~jvo/poo/2019-20/poo2019-20t15.pdf
3. http://w3.ualg.pt/~jvo/poo/2019-20/poo2019-20t14.pdf
4. https://www.baeldung.com/java-circular-linked-list
5. http://w3.ualg.pt/~jvo/poo2017-18/poo-Problem3.pdf
6. https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/
7. https://www.cs.usfca.edu/~galles/cs245S15/SparseArray/?fbclid=IwAR22Wyu7yd4T_zIQ4rtxLi3Bpw1E_W2scJirHDbBfiDFPmGxgK-UBrFxQRs
8. https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
9. https://github.com/guilhascorreia24/POO/blob/e6d92eccf8108726bf5899e34a3e5459c7f38950/lab5/Board.java

Anexo A

Mini-project for lab working groups: GoL

Deliverables:

- Submit your source code to http://deei-mooshak.ualg.pt/~jvo/ (Problem H) and
- your report (within a zip file) as TP3
  to http://www.deei.fct.ualg.pt/POO/Entregas/
  including:
  i) The problem id, your group number, and its elements;
  ii) Your description of the problem and the approach followed to address it;
  iii) The analysis UML class diagram
  iv) The unit tests developed
  v) All design options taken
  vi) Javadoc
  vii) The implementation UML class diagram (see www.objectaid.com)
  viii) Any relevant conclusion or remark
  ix) Bibliographic references used, if any

Up to May 11, 2020

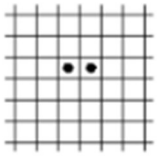
The problem

The Game of Life (GoL), introduced in 1970 by J. H. Conway, is a simulation of a borderless board, where each cell has a binary state of living or dead. Occupied cells are therefore addressed as *living* and free cells as *dead*.

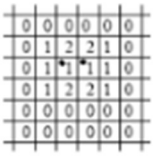
The board evolves at each generation according to the **transition rules**, in such a way that a cell can change its state according to the number of living neighbour cells.

Living neighbours

The neighbours of a cell are its 8 adjacent cells. The figure below shows in b) the counting of living neighbours for each cell, for a) a board with only 2 living cells represented by the black dots:



a)



b)

Transition rules

The next algorithm, computes the next generation of cells, given an initial cell distribution.

```
For each cell in the initial board,
    - IF living cell and the number of living neighbours is 2
      or 3 then the cell lives in the new board; it dies
      otherwise
    - IF dead cell and the number of living neighbours is 3
      then the cell lives in the new board; it remains dead
      otherwise
```

More information on GoL, as well as an online simulator, is available from:

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
https://bitstorm.org/gameoflife/

Task

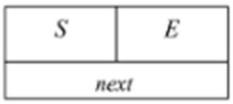
Develop a Java program that given an initial board (cell distribution) shows the successive distributions of cells when the GoL transition rules are applied generation after generation.

REQUIREMENTS

- Employ the principles and techniques of object-oriented programming.
- Use test-driven development.
- Unexpected conditions that lead to invalid states should resort to exceptions
- The game board might be naively implemented as a bidimensional array. However this will waste unnecessary space for large boards, so you should implement the board as a sparse bidimensional array of cell that only saves living cells.

- In particular, the sparse array should be implemented using sentinel and data nodes. The sentinel nodes will form a circular linked list. The *i*-th sentinel node SN_i will have the following *ad hoc* schematic representation:

SN_i



Where its members have the following meaning:

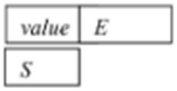
next - reference to the next sentinel node;

E – a reference to either the first data node of row *i* of the sparse array or to SN_i if row *i* does not have data nodes.

S – a reference to either the first data node of column *i* of the sparse array or to SN_i if column *i* does not have data nodes.

Therefore, the number of sentinel nodes equal the maximum of the number of rows and columns in the sparse array.

In turn, a data node will have, at least, the following information:



Where:

- *S* and *E* are references to either the next non-zero data node or to a sentinel node. The *E* of the last data node in a given row points back to the sentinel node of its row. Similarly, *S* of the last data node in a given column points back to the sentinel node of its column.
- value is of a *generic type* that in this particular problem holds a living cell.

NB: Any other approach to this problem, independently of its merit, will be marked with 0 (zero) values.

Evaluation of the unit tests

Multi-submissions to mooshak are allowed. However:

- Only the 3 first failed submission are free of charge; For the benefit of test driven development, you can submit as much accepted submissions as you like.
- After 3 failed submissions, the *i*-th failed submission has a penalty of 0,1*(*i-3)* over the final grade of this assignment; Example: a program was accepted at submission 6; i.e., it had 5 failed submissions with an associated penalty of 0+0+0+0,1+0,2 = 0,3v

Input

The first line of the input is a natural number *N* specifying the number of iterations or generations. After the first line, there will be as many input lines as the rows in the initial board. In each of these lines a 0 represents a dead cell and a 1 a living cell.

Output

The distribution of cells in every board of each generation up to generation *N*. One row for each board line; where a dead cell is presented by 0 while a living one is represented by a 1. Boards are separated by a blank line.

NB: A board can grow as needed, but should never be smaller than the initial board.

Sample Input 0: oscillator with borders

```
3
010
010
010
```

Sample Output 0

```
000
111
000

010
010
010

000
111
000
```

Sample Input 1: oscillator without borders

```
3
1
1
1
```

Sample Output 1

```
000
111
000

010
010
010

000
111
000
```

Sample Input 2: glider up left

```
4
111
100
010
```

Sample Output 2

```
010
110
101
000

110
101
100
000

0110
1100
0010
0000

1110
1000
0100
0000
```

Sample Input 3: vanishes

```
1
```

101
000
101

Sample Output 3
000
000
000

Sample Input 4: block
2
11
11

Sample Output 4
11
11

11
11

Sample Input 5: fade right without borders
3
110
001
110

Sample Output 5
010
001
010

000
011
000

000
000
000

Sample Input 6: fade left with borders
3
000000
001100
010000
001100
000000