ex2TP2

November 14, 2022

1 Exercício 2 (Conway's Game of Life) - Trabalho Prático 2

Grupo 4: Carlos Costa-A94543Ruben Silva-A9463

2 Problema:

2. O Conway's Game of Life é um exemplo bastante conhecido de um autómato celular . Neste problema vamos modificar as regras do autómato da seguinte forma
    1. O espaço de estados é finito definido por uma grelha de células booleanas (morta=0/viva=1) de dimensão $N \times N$ (com $N > 3$) identificadas por índices $(i, j) \in \{1..N\}$. Estas N^2 células são aqui referidas como "normais". No estado inicial todas as células normais estão mortas excepto um quadrado 3×3, designado por "centro", aleatoriamente posicionado formado apenas por células vivas.
    2. Adicionalmente existem $2\,N+1$ "células da borda" que correspondem a um dos índices, i ou j, ser zero. As células da borda têm valores constantes que, no estado inicial, são gerados aleatoriamente com uma probabilidade ρ de estarem vivas.
    3. As células normais o autómato modificam o estado de acordo com a regra "B3/S23": i.e. a célula nasce (passa de 0 a 1) se tem exatamente 3 vizinhos vivos e sobrevive (mantém-se viva) se o número de vizinhos vivos é 2 ou 3, caso contrário morre ou continua morta. A célula (i_0, j_0) e (i_1, j_1) são vizinhas $\Leftrightarrow$ $(i_0 - i_1 = \pm 1) \vee (j_0 - j_1 = \pm 1)$

Pretende-se: 1. Construir uma máquina de estados finita que represente este autómato; são parâmetros do problema os parâmetros N, ρ e a posição do "centro". 2. Verificar se se conseguem provar as seguintes propriedades: 1. Todos os estados acessíveis contém pelo menos uma célula viva. 2. Toda a célula normal está viva pelo menos uma vez em algum estado acessível.

3 Conway's Game of Life Rules:

(Excerto extraido da wikipédia)

The universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live or dead (or populated and unpopulated, respectively). Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

These rules, which compare the behavior of the automaton to real life, can be condensed into the following: 1. Any live cell with two or three live neighbours survives. 2. Any dead cell with three live neighbours becomes a live cell. 3. All other live cells die in the next generation. Similarly, all other dead cells stay dead. The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed, live or dead; births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick.[nb 1] Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations.

4 Análise do Problema

Este é um problema sobre um jogo. Para resolver este problema será necessário uma matriz N x N e cada membro dessa matriz será boleana, sendo 0 morto e 1 viva. A variável principal para usar neste exercício será "e_{ij}" sendo i as linhas e j as colunas. * $e_{ij} \in \mathbb{N}_{\mathbb{0}} * \mathbb{N}_{\mathbb{0}}; \rightarrow$ Posição do mapa do jogo (frame/estado atual) * $i \in \mathbb{N}_{\mathbb{0}}; \rightarrow$ Linha do mapa * $j \in \mathbb{N}_{\mathbb{0}}; \rightarrow$ Coluna do mapa * $vizinhos \in \mathbb{N}_{\mathbb{0}}; \rightarrow$ Função para encontrar vizinhos * $e'_{ij} \in \mathbb{N}_{\mathbb{0}} * \mathbb{N}_{\mathbb{0}}; \rightarrow$ Posição do mapa do jogo (frame/estado seguinte)

5 Limitações e obrigações

1. **Regra de permanecer vivo (2 vizinhos):**

$$e_{ij} = 1 \wedge \sum vizinhos = 2 \wedge e'_{ij} = 1$$

2. **Regra de permanecer vivo (3 vizinhos):**

$$e_{ij} = 1 \wedge \sum vizinhos = 3 \wedge e'_{ij} = 1$$

3. **Regra de morrer por sobrepopulação:**

$$e_{ij} = 1 \wedge \sum vizinhos >= 4 \wedge e'_{ij} = 0$$

4. **Regra de morrer por subpopulação:**

$$e_{ij} = 1 \wedge \sum vizinhos <= 1 \wedge e'_{ij} = 0$$

5. **Regra de nascer:**

$$e_{ij} = 0 \wedge \sum vizinhos = 3 \wedge e'_{ij} = 1$$

6. **Regra de permanecer morto:**

$$e_{ij} = 0 \wedge \sum vizinhos! = 3 \wedge e'_{ij} = 0$$

6 Implementaçao do Problema

Importar o solver 1. Instalar o z3-solver a partir da libraria do PyPi (pip) 2. Importar o z3-solver 3. Importar o random

[ ]:
```
from z3 import *
import random
%pip install z3-solver
```

```
Requirement already satisfied: z3-solver in
c:\users\ruben\appdata\local\programs\python\python310\lib\site-packages
(4.11.2.0)Note: you may need to restart the kernel to use updated packages.
```

7 Resolver o codigo

Função "declare" Esta função é responsável pela declaração de todas as variáveis que serão utilizadas no solver. 1. Parâmetros: 1. i -> um inteiro que será responsável por dar o nr às variaveis 2. N -> tamanho 2. Função: 1. Inicialmente criamos um dicionário para colocar todas as variáveis necessárias. 2. Criamos N x N variáveis 3. Return do novo dicionário com as variáveis

[ ]:
```
def declare(i, N):
    state = {}
    for n1 in range(N):
        for n2 in range(N):
            state["e"+str(n1)+str(n2)] = Int("e" +
                                             str(n1)+str(n2)+"frame"+str(i))
    return state
```

Função "init" Esta função é responsável pela inicialização do primeiro node do traço (Primeiro membro do dicionário principal da função) e algumas condições lógicas necessárias 1. Parâmetros: 1. *state* -> Primeiro frame da função 2. *n*-> Tamanho 3. *rho* -> Probabilidade 4. *center* -> Coordenada central do bloco 2. Função: 1. Criamos o Bloco 3x3 Inicial (case_alive_Initial_Block) 2. Criamos o resto da mapa (case_dead_without_border) 3. Criamos a bordar adicional com $2N+1$ elementos (case_border) 3. Return de um "And" com as condições (case_dead_without_border case_alive_Initial_Block case_border)

[ ]:
```
def init(state, n, rho, center):

    # Criar o Bloco 3x3 inicial
    spawn_block = []
    used_block = []
    vizinhos = [(1, 0), (0, 1), (1, 1), (-1, 0),
                (0, -1), (-1, -1), (-1, 1), (1, -1), (0,0)]
    #Percorrer todos os vizinhos
    for j1 in vizinhos:
        spawnLocationX = center[0] + j1[0]
        spawnLocationY = center[1] + j1[1]
        str_aux = "e" + str(spawnLocationX)+str(spawnLocationY)
        spawn_block.append(state[str_aux] == 1)
        used_block.append(str_aux)

    matrix_without_border = []
```

```
    # Inicializar tudo a 0 menos o bloco inicial e MENOS A BORDA
    for n1 in range(1, n):
        for n2 in range(1, n):
            str_aux = "e"+str(n1)+str(n2)
            if str_aux not in used_block:
                matrix_without_border.append(state[str_aux] == 0)

    # Probabilidade de a celula estar viva
    prob_int = int(100*rho)
    prob = [p for p in range(prob_int)]

    matrix_border = []
    for i1 in range(n):
        if (random.randint(0, 100) in prob):
            # Vivo
            matrix_border.append(state["e"+str(i1)+str(0)] == 1)
        else:
            pass
            # Morto
            matrix_border.append(state["e"+str(i1)+str(0)] == 0)

    # range(1,n) pois se nao, iamos tar a dar 2 estados ao e00 e é problematico
    for i2 in range(1, n):
        if (random.randint(0, 100) in prob):
            # Vivo
            matrix_border.append(state["e"+str(0)+str(i2)] == 1)
        else:
            pass
            # Morto
            matrix_border.append(state["e"+str(0)+str(i2)] == 0)

    case_alive_Initial_Block = And(spawn_block)
    case_dead_without_border = And(matrix_without_border)
    case_border = And(matrix_border)
    return And(case_dead_without_border, case_alive_Initial_Block, case_border)
```

Função "deteta_vizinhos" Esta função calcula números de vizinhos vivos 1. Parâmetros: 1. *state* -> Frame atual da função 2. *pos* -> Posição da variável que estamos a usar e_{ij} 2. Função: 1. Uma verificação se existe alguma membro em $(i+1$ ou $i-1)$ ou $(j+1$ ou $j-1)$ com um if para evitar que ele procure em valores fora da matriz 3. Return do somatório dos vizinhos todos com o valor de 1 (vivos)

[ ]:
```
def deteta_vizinhos(state, pos):
    x = pos[1]
    y = pos[2]
    vizinhos_L = [(1, 0), (0, 1), (1, 1), (-1, 0),
                  (0, -1), (-1, -1), (-1, 1), (1, -1)]
```

```
    vizinhos = []
    for j1 in vizinhos_L:
        LocationX = int(x) + int(j1[0])
        LocationY = int(y) + int(j1[1])
        str_aux = "e" + str(LocationX)+str(LocationY)

        #0 if tem como função evitar que seja escrito fora da matriz
        if(str_aux in state):
            vizinhos.append(state[str_aux])

    return sum(vizinhos)
```

Função "trans" Esta função é responsável pela criação das conexões lógicas necessárias para o FOTS fazer sentido e ser o pretendido 1. Parâmetros: 1. *curr* -> Frame atual da função 2. *prox* -> Frame seguinte da função 3. *pos* -> Posição da variável que estamos a usar e_{ij} 2. Função: 1. Criamos as condições lógicas chamadas transita: 1. transita01: $(eij = 1 \wedge \sum vizinhos = 2 \wedge eij' = 1)$ 2. transita02: $(eij = 1 \wedge \sum vizinhos = 3 \wedge eij' = 1)$ 3. transita03: $(eij = 1 \wedge \sum vizinhos >= 4 \wedge eij' = 0)$ 4. transita04: $(eij = 1 \wedge \sum vizinhos <= 1 \wedge eij' = 0)$ 5. transita05: $(eij = 0 \wedge \sum vizinhos = 3 \wedge eij' = 1)$ 6. transita06: $(eij = 0 \wedge \sum vizinhos! = 2 \wedge eij' = 0)$ 3. Return de um "And" com a seguinte condição lógica: ($ transita01 transita02 transita03 transita04 transita05 transita06$)

[ ]:
```
def trans(curr, prox, pos):

    # Regra de permanecer vivo (2 vizinhos)
    # (eij=1   SUM_vizinhos=2   eij=1)
    transita01 = And(curr[pos] == 1, deteta_vizinhos(
        curr, pos) == 2, prox[pos] == 1)

    # Regra de permanecer vivo (3 vizinhos)
    # (eij=1   SUM_vizinhos=3   eij=1)
    transita02 = And(curr[pos] == 1, deteta_vizinhos(
        curr, pos) == 3, prox[pos] == 1)

    # Regra de morrer por sobrepopulação
    # (eij=1   SUM_vizinhos>=4   eij=0)
    transita03 = And(curr[pos] == 1, deteta_vizinhos(
        curr, pos) >= 4, prox[pos] == 0)

    # Regra de morrer por subpopulação
    # (eij=1   SUM_vizinhos<=1   eij=0)
    transita04 = And(curr[pos] == 1, deteta_vizinhos(
        curr, pos) <= 1, prox[pos] == 0)

    # Regra de nascer
    # (eij=0   SUM_vizinhos=3   eij=1)
    transita05 = And(curr[pos] == 0, deteta_vizinhos(
```

```
        curr, pos) == 3, prox[pos] == 1)

    # Regra de permanecer morto
    # (eij=0   SUM_vizinhos!=3   eij=0)
    transita06 = And(curr[pos] == 0, deteta_vizinhos(
        curr, pos) != 3, prox[pos] == 0)

    return Or(transita01, transita02, transita03, transita04, transita05,
↪transita06)
```

Função "conways_game_of_life" Esta é a função principal e é a que irar juntar as funções todas e gerar o traço pretendido e com ele tabelar o output 1. Parâmetros: 1. declare -> Função declare 2. init -> Função init 3. trans -> Função trans 4. N -> Tamanho do mapa (input do utilizador) 5. k -> Quantidade de frames pretendidos (input do utilizador) 6. *rho* -> Probabilidade (input do utilizador) 7. *center* -> Coordenada central do bloco 2. Função: 1. Iniciamos o Solver 2. Criar as variáveis todas que serão usadas no solver 3. Inicializar as variáveis 4. Criar a conexão lógica entre os nodes do traço todos e adicionar ao solver 5. Correr o Solver, Tabelar o resultado e obter as seguintes informações: 1. Todos os estados acessíveis contém pelo menos uma célula viva? 2. Toda a célula normal está viva pelo menos uma vez em algum estado acessível?

[ ]:
```
def conways_game_of_life(declare, init, trans, N, k, rho, center):
    if (N > 3):
        # Pois assim criamos a "borda adicional"
        N = N+1
        s = Solver()

        # Matriz de N x N
        # Declarar todas as variaveis que serão usadas no solver
        trace = [declare(i, N) for i in range(k)]

        # inicializar as variáveis iniciais
        s.add(init(trace[0], N, rho,center))

        # Criar a conexão lógica entre os nodes do traço todos e adicionar ao
↪solver
        for i in range(k-1):
            for j1 in range(N):
                for j2 in range(N):
                    str_aux = "e"+str(j1)+str(j2)
                    s.add(trans(trace[i], trace[i+1], str_aux))

        # Correr o Solver e Tabelar o resultado
        # Obter as seguintes verificações tambem:
        # 1. Todos os estados acessíveis contém pelo menos uma célula viva.
        # 2. Toda a célula normal está viva pelo menos uma vez em algum estado
↪acessível.
```

```
        # Desenhar a tabela
        if s.check() == sat:
            m = s.model()

            # Percorrer cada estado/Frame e tabelar
            one_time_alive = []
            flag_alive = 0
            for i in range(k):
                print("\n\nFRAME -> " + str(i+1))
                aux_str = "e "
                for j1 in range(N):
                    aux_str += "| "+str(j1)+" "
                print(aux_str)

                # Percorrer cada variavel desse Frame/Estado
                counter = 0
                row = 1
                aux_str2 = str(0)+" "
                for v in trace[i]:
                    if (i == k-1 and m[trace[i][v]] == 1):
                        flag_alive = 1

                    # Adicionar à lista das celulas que pelo menos tiveram␣
↪vivas uma vez
                    if (v not in one_time_alive and m[trace[i][v]] == 1):
                        one_time_alive.append(v)

                    # Daqui para baixo é para desenhar a tabela
                    if counter == N:
                        print(aux_str2)
                        aux_str2 = str(row)+" "
                        row += 1
                        counter = 0

                    circle = "O"
                    if (m[trace[i][v]] == 0):
                        circle = "-"
                    aux_str2 += "| "+circle+" "
                    counter += 1
                print(aux_str2)

            # Validar a verificação se cada celula teve pelo menos uma vez viva
            str_validacao1 = "\n\nTodas as celulas tiveram pelo menos uma vez␣
↪vivas!"
            for (key, value) in trace[0].items():
                if(key not in one_time_alive):
```

```
                    str_validacao1 = "\n\nNem Todas as celulas ficaram pelo uma␣
↪vez vivas!!"
                    break
            print(str_validacao1)

            # .Todos os estados acessiveis teem pelos menos uma celula viva
            # Para isto basta verificar o ultimo estado/frame
            str_validacao2 = "\nExiste estados sem celulas vivas!"
            if (flag_alive):
                str_validacao2 = "\nTodos os estados tem celulas vivas!"
            print(str_validacao2)

        else:
            print("No Solution")
    else:
        print("Tamanho de N menor a 3!")
```

Exemplo 1 (Impossível) N < 3

[ ]:
```
N = 2  # Tamanho da matriz
k = 5  # Quantidade de frames/estados
rho = 0.5  # Probabilidade
center = (3, 4)  # Tuplo com a localização do centro

conways_game_of_life(declare, init, trans, N, k, rho, center)
```

```
Tamanho de N menor a 3!
```

Exemplo 1 (Possível) Exemplo possível e com poucos estados

[ ]:
```
N = 4 # Tamanho da matriz
k = 4  # Quantidade de frames/estados
rho = 0.5  # Probabilidade
center = (3, 3)  # Tuplo com a localização do centro

conways_game_of_life(declare, init, trans, N, k, rho,center)
```

```
FRAME -> 1
e | 0 | 1 | 2 | 3 | 4
0 | 0 | - | 0 | - | 0
1 | 0 | - | - | - | -
2 | - | - | 0 | 0 | 0
3 | - | - | 0 | 0 | 0
4 | 0 | - | 0 | 0 | 0


FRAME -> 2
```

```
e | 0 | 1 | 2 | 3 | 4
0 | - | O | - | - | -
1 | - | - | O | - | O
2 | - | O | O | - | O
3 | - | - | - | - | -
4 | - | O | O | - | O


FRAME -> 3
e | 0 | 1 | 2 | 3 | 4
0 | - | - | - | - | -
1 | - | - | O | - | -
2 | - | O | O | - | -
3 | - | - | - | - | -
4 | - | - | - | - | -


FRAME -> 4
e | 0 | 1 | 2 | 3 | 4
0 | - | - | - | - | -
1 | - | O | O | - | -
2 | - | O | O | - | -
3 | - | - | - | - | -
4 | - | - | - | - | -


Nem Todas as celulas ficaram pelo uma vez vivas!!

Todos os estados tem celulas vivas!
```

Exemplo 1 (Possível) Exemplo possível e com muitos estados

[ ]:
```
N =9  # Tamanho da matriz
k = 15  # Quantidade de frames/estados
rho = 0.8  # Probabilidade
center = (6, 5)  # Tuplo com a localização do centro

conways_game_of_life(declare, init, trans, N, k, rho,center)
```

```
FRAME -> 1
e | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
0 | O | - | O | O | O | O | O | O | O | O
1 | O | - | - | - | - | - | - | - | - | -
2 | O | - | - | - | - | - | - | - | - | -
3 | O | - | - | - | - | - | - | - | - | -
4 | O | - | - | - | - | - | - | - | - | -
5 | O | - | - | - | O | O | O | - | - | -
```

```
6 | O | - | - | - | O | O | O | - | - | -
7 | O | - | - | - | O | O | O | - | - | -
8 | - | - | - | - | - | - | - | - | - | -
9 | O | - | - | - | - | - | - | - | - | -
```

```
FRAME -> 2
e | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
0 | - | O | - | O | O | O | O | O | O | -
1 | O | - | - | O | O | O | O | O | O | -
2 | O | O | - | - | - | - | - | - | - | -
3 | O | O | - | - | - | - | - | - | - | -
4 | O | O | - | - | - | O | - | - | - | -
5 | O | O | - | - | O | - | O | - | - | -
6 | O | O | - | O | - | - | - | O | - | -
7 | - | - | - | - | O | - | O | - | - | -
8 | - | - | - | - | - | O | - | - | - | -
9 | - | - | - | - | - | - | - | - | - | -
```

```
FRAME -> 3
e | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
0 | - | - | O | O | - | - | - | - | O | -
1 | O | - | - | O | - | - | - | - | O | -
2 | - | - | O | - | O | O | O | O | - | -
3 | - | - | O | - | - | - | - | - | - | -
4 | - | - | O | - | - | O | - | - | - | -
5 | - | - | - | - | O | O | O | - | - | -
6 | O | O | O | O | O | - | O | O | - | -
7 | - | - | - | - | O | O | O | - | - | -
8 | - | - | - | - | - | O | - | - | - | -
9 | - | - | - | - | - | - | - | - | - | -
```

```
FRAME -> 4
e | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
0 | - | - | O | O | - | - | - | - | - | -
1 | - | O | - | - | - | O | O | - | O | -
2 | - | O | O | - | O | O | O | O | - | -
3 | - | O | O | - | O | - | - | - | - | -
4 | - | - | - | O | O | O | O | - | - | -
5 | - | - | - | - | - | - | - | O | - | -
6 | - | O | O | - | - | - | - | O | - | -
7 | - | O | O | - | - | - | - | O | - | -
8 | - | - | - | - | O | O | O | - | - | -
9 | - | - | - | - | - | - | - | - | - | -
```

FRAME -> 5

e	0	1	2	3	4	5	6	7	8	9
0	-	-	O	-	-	-	-	-	-	-
1	-	O	-	-	-	-	-	-	-	-
2	O	-	-	-	O	-	-	O	-	-
3	-	O	-	-	-	-	-	O	-	-
4	-	-	O	O	O	O	O	-	-	-
5	-	-	O	O	O	O	-	O	-	-
6	-	O	O	-	-	-	O	O	O	-
7	-	O	O	O	-	O	-	O	-	-
8	-	-	-	-	-	O	O	-	-	-
9	-	-	-	-	-	O	-	-	-	-

FRAME -> 6

e	0	1	2	3	4	5	6	7	8	9
0	-	-	-	-	-	-	-	-	-	-
1	-	O	-	-	-	-	-	-	-	-
2	O	O	-	-	-	-	-	-	-	-
3	-	O	O	-	-	-	-	O	-	-
4	-	O	-	-	-	-	-	O	-	-
5	-	-	-	-	-	-	-	-	O	-
6	-	-	-	-	-	-	-	-	O	-
7	-	O	-	O	O	O	-	-	O	-
8	-	-	O	-	-	O	-	-	-	-
9	-	-	-	-	-	O	O	-	-	-

FRAME -> 7

e	0	1	2	3	4	5	6	7	8	9
0	-	-	-	-	-	-	-	-	-	-
1	O	O	-	-	-	-	-	-	-	-
2	O	-	-	-	-	-	-	-	-	-
3	-	-	O	-	-	-	-	-	-	-
4	-	O	O	-	-	-	-	O	O	-
5	-	-	-	-	-	-	-	O	O	-
6	-	-	-	-	O	-	-	O	O	O
7	-	-	O	O	O	O	-	-	-	-
8	-	-	O	O	-	-	-	-	-	-
9	-	-	-	-	-	O	O	-	-	-

FRAME -> 8

e	0	1	2	3	4	5	6	7	8	9
0	-	-	-	-	-	-	-	-	-	-
1	O	O	-	-	-	-	-	-	-	-
2	O	-	-	-	-	-	-	-	-	-
3	-	-	O	-	-	-	-	-	-	-

```
4 | - | O | O | - | - | - | - | O | O | -
5 | - | - | - | - | - | - | O | - | - | -
6 | - | - | - | - | O | O | O | O | - | O
7 | - | - | O | - | - | O | - | - | O | -
8 | - | - | O | - | - | - | O | - | - | -
9 | - | - | - | - | - | - | - | - | - | -
```

```
FRAME -> 9
e | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
0 | - | - | - | - | - | - | - | - | - | -
1 | O | O | - | - | - | - | - | - | - | -
2 | O | - | - | - | - | - | - | - | - | -
3 | - | - | O | - | - | - | - | - | - | -
4 | - | O | O | - | - | - | - | O | - | -
5 | - | - | - | - | - | - | - | - | - | -
6 | - | - | - | - | O | - | - | O | O | -
7 | - | - | - | O | O | - | - | - | O | -
8 | - | - | - | - | - | - | - | - | - | -
9 | - | - | - | - | - | - | - | - | - | -
```

```
FRAME -> 10
e | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
0 | - | - | - | - | - | - | - | - | - | -
1 | O | O | - | - | - | - | - | - | - | -
2 | O | - | - | - | - | - | - | - | - | -
3 | - | - | O | - | - | - | - | - | - | -
4 | - | O | O | - | - | - | - | - | - | -
5 | - | - | - | - | - | - | - | O | O | -
6 | - | - | - | O | O | - | - | O | O | -
7 | - | - | - | O | O | - | - | O | O | -
8 | - | - | - | - | - | - | - | - | - | -
9 | - | - | - | - | - | - | - | - | - | -
```

```
FRAME -> 11
e | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
0 | - | - | - | - | - | - | - | - | - | -
1 | O | O | - | - | - | - | - | - | - | -
2 | O | - | - | - | - | - | - | - | - | -
3 | - | - | O | - | - | - | - | - | - | -
4 | - | O | O | - | - | - | - | - | - | -
5 | - | - | O | O | - | - | - | O | O | -
6 | - | - | - | O | O | - | O | - | - | O
7 | - | - | - | O | O | - | - | O | O | -
8 | - | - | - | - | - | - | - | - | - | -
9 | - | - | - | - | - | - | - | - | - | -
```

FRAME -> 12

e	0	1	2	3	4	5	6	7	8	9
0	-	-	-	-	-	-	-	-	-	-
1	O	O	-	-	-	-	-	-	-	-
2	O	-	-	-	-	-	-	-	-	-
3	-	-	O	-	-	-	-	-	-	-
4	-	O	-	-	-	-	-	-	-	-
5	-	O	-	-	O	-	-	O	O	-
6	-	-	-	-	-	O	O	-	-	O
7	-	-	-	O	O	O	-	O	O	-
8	-	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-

FRAME -> 13

e	0	1	2	3	4	5	6	7	8	9
0	-	-	-	-	-	-	-	-	-	-
1	O	O	-	-	-	-	-	-	-	-
2	O	-	-	-	-	-	-	-	-	-
3	-	O	-	-	-	-	-	-	-	-
4	-	O	O	-	-	-	-	-	-	-
5	-	-	-	-	-	O	O	O	O	-
6	-	-	-	O	-	-	-	-	-	O
7	-	-	-	-	O	O	-	O	O	-
8	-	-	-	-	O	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-

FRAME -> 14

e	0	1	2	3	4	5	6	7	8	9
0	-	-	-	-	-	-	-	-	-	-
1	O	O	-	-	-	-	-	-	-	-
2	O	-	-	-	-	-	-	-	-	-
3	O	O	O	-	-	-	-	-	-	-
4	-	O	O	-	-	-	O	O	-	-
5	-	-	O	-	-	-	O	O	O	-
6	-	-	-	-	-	-	-	-	-	O
7	-	-	-	O	O	O	-	-	O	-
8	-	-	-	-	O	O	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-

FRAME -> 15

e	0	1	2	3	4	5	6	7	8	9
0	-	-	-	-	-	-	-	-	-	-
1	O	O	-	-	-	-	-	-	-	-

```
2 | - | - | O | - | - | - | - | - | - | -
3 | O | - | O | - | - | - | - | - | - | -
4 | O | - | - | O | - | - | O | - | O | -
5 | - | O | O | - | - | - | O | - | O | -
6 | - | - | - | O | O | O | O | - | - | O
7 | - | - | - | O | - | O | - | - | - | -
8 | - | - | - | O | - | O | - | - | - | -
9 | - | - | - | - | - | - | - | - | - | -
```

Nem Todas as celulas ficaram pelo uma vez vivas!!

Todos os estados tem celulas vivas!