Tópicos de Análise

one-day meeting, 26 Junho 2021

one-day meeting II, 5 Julho 2021

Eds Carlos Ramos, Luís Bandeira

Universidade de Évora
Setembro de 2022

UNIVERSIDADE DE ÉVORA
ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE MATEMÁTICA

Preâmbulo

No âmbito da unidade curricular análise matemática IV, da Universidade de Évora são propostos, aos alunos, trabalhos de iniciação à investigação incidindo sobre tópicos relacionados com o programa da unidade curricular referida. Estes trabalhos servem como complemento de avaliação, com apresentação pública num workshop preparado para o efeito. Os trabalhos escritos que revelem qualidade são propostos para edição de actas do encontro. No ano lectivo 2020/2021 o encontro realizou-se em dois dias 26/6/2021 e 5/7/2021, com os seguintes programas:
26/6/2021

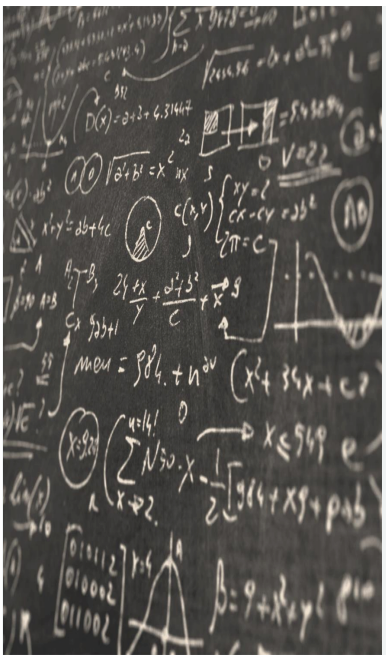
- Blandina Dobrescu, O problema do caixeiro viajante.
- Madalena Ascensão, Funções geradoras: A função zeta de Riemann.
- Gheorghe Ciolpan, Autómatos celulares.
- Bernardo Stoffel, Transformações de Mobius.
- Irina Gonçalves, Funções geradoras.
- Luís Romão, Funções geradoras de Dirichlet.
- Gonçalo Pereira, Isometrias do plano hiperbólico.

5/7/2021

- Mariana Sousa, Oscilador harmónico quântico.
- Beatriz Neves, Aplicação logística, conjunto de Mandelbrot e a constante de Feigenbaum.
- Pedro Spínola, A matemática e o Blockchain.
- Carolina Confraria, Modelo Black-Scholes.
- Francisco Pires, Estratégia de corrida de um carro de fórmula 1
- João Nogueira, Análise de redes complexas.
- Alexandre Pereira, Equação de Schrodinger.
- Maria Caldeira, Equação de Laplace.

Do processo de seleção e edição, no ano lectivo 2020/2021, resultou um trabalho realizado por Gheorghe Ciolpan, onde se abordam os autómatos celulares. Os autómatos celulares podem ser vistos como equações às diferenças em corpos finitos e podendo ser lineares ou não lineares oferecem possibilidades de modelação sofisticadas, além de relacionarem problemas teóricos relevantes e de grande actualidade, na área da análise, combinatória e sistemas dinâmicos. Neste caso o trabalho, autómatos celulares, expõe o assunto de modo introdutório e apresenta uma aplicação à propagação dos incêndios, incluindo a implementação em Python de um modelo.

Os editores, Carlos Ramos e Luís Bandeira
Setembro 2022, Universidade de Évora



Tópicos de análise matemática

One day meeting

Sala 124 clav

10h

Blandina Dobrescu, *O problema do caixeiro viajante*.
(apresentação convidada)

Madalena Ascensão, *Funções geradoras: A função zeta de Riemann*.
(apresentação contribuida)

Gheorghe Ciolpan, *Autómatos celulares*.
(apresentação contribuida)

Pausa

11h

Bernardo Stoffel, *Transformações de Mobius*.
(apresentação contribuida)

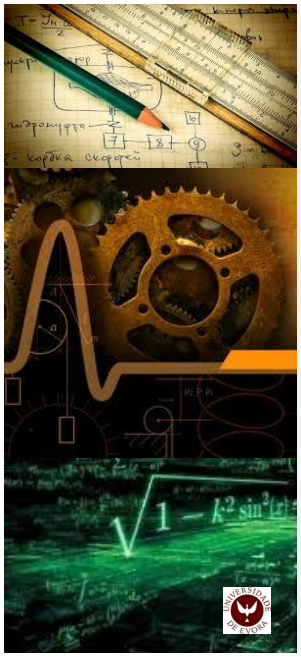
Irina Gonçalves, *Funções geradoras*.
(apresentação contribuida)

Luís Romão, *Funções geradoras de Dirichlet*.
(apresentação contribuida)

Gonçalo Pereira, *Isometrias do plano hiperbólico*.
(apresentação contribuida)

Encerramento

Departamento de Matemática, Escola de Ciências e Tecnologia, Universidade de Évora



Tópicos de análise matemática

One day meeting II

5/7/2021 Sala 128 clav

10h

Mariana Sousa, *Oscilador harmónico quântico*.

Beatriz Neves, *Aplicação logística, conjunto de Mandelbrot e a constante de Feigenbaum*.

Pedro Spínola, *A matemática e o Blockchain*.

Carolina Confraria, *Modelo Black-Scholes*.

15h

Francisco Pires, *Estratégia de corrida de um carro de fórmula 1*

João Nogueira, *Análise de redes complexas*.

Alexandre Pereira, *Equação de Schrodinger*.

Maria Caldeira, *Equação de Laplace*.

Encerramento

Departamento de Matemática, Escola de Ciências e Tecnologia, Universidade de Évora

Conteúdo

Preâmbulo iii

Trabalhos

Gheorghe Ciolpan

Autómatos Celulares, Tópicos de Análise – one-day meeting. 5, Julho 2021, actas (Eds Carlos Ramos, Luís Bandeira), Universidade de Évora, Setembro 2022 . . 1

Índice de autores 15

Trabalhos

Autómatos Celulares

Gheorghe Ciolpan[1]

[1]Aluno da licenciatura em Matemática Aplicada à Economia e à Gestão, Universidade de Évora, Portugal

Corresponding/Presenting author: l46281@alunos.uevora.pt

Resumo

Os autómatos celulares são modelos dinâmicos de evolução temporal, cujas variáveis discretas espaço/tempo permitem modelar vários sistemas complexos, com um grande número de componentes em interação ao nível matemático, físico, biológico e computacional. Por sua vez, estes sistemas são representados numa grelha infinita de células d-dimensionais, onde cada célula pode estar num de k estados, que variam consoante as regras determinísticas e as condições iniciais estabelecidas. O objetivo do presente trabalho é a abordagem de autómatos celulares determinísticos, unidimensionais e bidimensionais, mais concretamente as regras elementares e o famoso *Jogo da Vida*, com intuito de elaborar uma simulação da propagação dos fogos, utilizando o *software Python*.

Palavras-chave: Sistemas dinâmicos, autómatos celulares, simulações da propagação de fogos.

1 Introdução

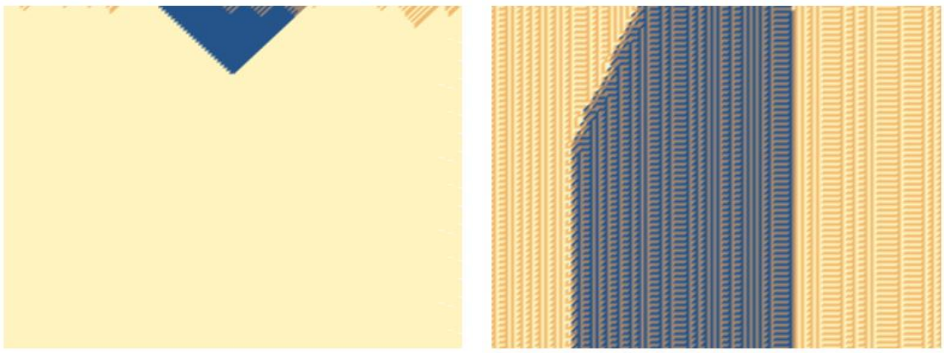
Os autómatos celulares podem ser determinísticos ou probabilísticos. No que diz respeito ao primeiro, o estado de cada elemento no instante t é uma função do estado no tempo $t-1$ de um número finito de células na sua vizinhança. Já relativamente ao modelo probabilístico, além da dependência dos estados no instante anterior, a transição de um estado depende também de variáveis aleatórias, ver por exemplo [1]. Contudo, este estudo centra-se apenas nos modelos determinísticos. Em geral, os autómatos celulares possuem um número finito de configurações possíveis. Por exemplo, para um autómato com $k = 2$ estados (0 ou 1) e $n = 9$ células existirá um total de $k^n = 2^9 = 512$ configurações possíveis.
Este conceito foi introduzido por John von Neumann em 1940, ver [3], que

sob ajuda de estudos de Stanislaw Ulam, deu origem ao primeiro autómato celular autorreplicante cujo nome foi *Universal Constructor*, baseado numa grelha com duas dimensões onde cada célula podia estar num de 29 estados possíveis, em cada instante.
Mais tarde, na década de 70, John Conway, introduziu um autómato celular bidimensional que teve grande sucesso, particularmente para a comunidade informática [2]. O autómato celular bidimensional referido é o chamado *Jogo da Vida*, com potencialidade para simular grande variedade de comportamentos dinâmicos.
Em 1983, Stephan Wolfram [4] analisou autómatos celulares unidimensionais (a que chamou de autómatos celulares elementares), classificando-os de forma intuitiva ao nível do seu comportamento dinâmico, em quatro classes. A primeira classe especifica que a evolução temporal leva o autómato celular a um estado homogéneo no qual todos os sítios atingem o mesmo valor (pontos fixos). Relativamente à segunda, esta estabelece que existem alguns estados finais diferentes, todos periódicos. Por sua vez, a terceira classe indica que o comportamento é para todos os efeitos pseudo-aleatório, apesar de algumas pequenas estruturas locais se poderem repetir. Por fim, a quarta classe refere que o autómato celular gera estruturas complexas com evolução imprevisível, que se podem propagar, criar ou aniquilar outras estruturas (é comum afirmar ser esta a classe na fronteira do caos). Ver exemplos na Figura 1.
No trabalho, para além de serem abordados os autómatos celulares, é feita uma simulação da propagação do fogo com base nos mesmos, sendo este o principal objetivo.

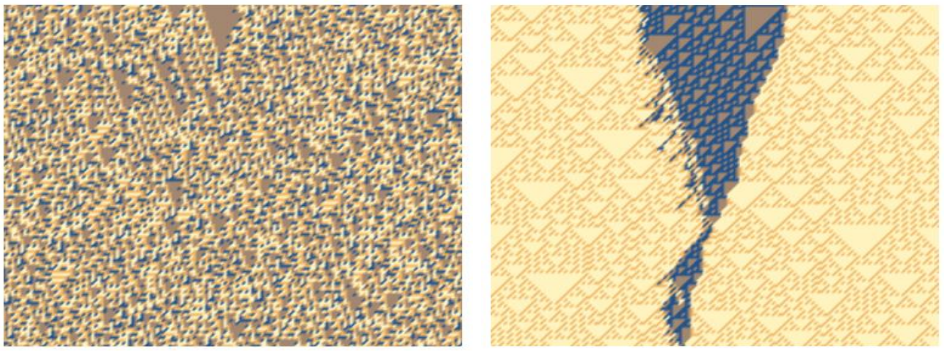
2 Autómatos celulares unidimensionais - regras elementares

O autómato celular não-trivial mais simples é unidimensional, com dois estados possíveis para cada célula, onde os vizinhos de cada uma destas células são a célula imediatamente à sua esquerda e à sua direita. Cada célula e os seus vizinhos formam então uma vizinhança de três células. Uma regra consiste então em decidir, para cada uma das configurações possíveis destas vizinhanças, qual o estado da célula do meio na próxima geração. Consequentemente, existem oito (2^3) configurações possíveis para as transições. Para estes modelos existem então 256 regras distintas.
Como exemplo concreto considere-se a regra 30, cujo o conjunto de estados é $Z_2 = \{0, 1\}$, onde o 30 é a designação em base decimal da sequência 00011110 em binário. Esta sequência codifica a regra 30 e denomina-se código da regra do autómato.

$$30 = \mathbf{0} \times 2^0 + \mathbf{1} \times 2^1 + \mathbf{1} \times 2^2 + \mathbf{1} \times 2^3 + \mathbf{1} \times 2^4 + \mathbf{0} \times 2^5 + \mathbf{0} \times 2^6 + \mathbf{0} \times 2^7$$



Classe 1. Classe 2.



Classe 3. Classe 4.

Figura1. Exemplos de evolução temporal em cada uma das diferentes classes de Wolfram.

Com base nas regras de transição, cada célula vai-se atualizando do instante $t-1$ para o instante t. Este conjunto de regras chama-se função de transição. Por outras palavras, a função de transição associa o valor de cada célula a um novo valor, após cada iteração, sendo esse novo valor definido com base nos valores de uma vizinhança pré-determinada e no código de cada regra, como é possível observar na Tabela 1 (ver também Figuras 2 e 3).

Configurações	000	001	010	011	100	101	110	111
	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
Código	0	1	1	1	1	0	0	0

Tabela1. Código da Regra 30.

Vejamos o seguinte exemplo da evolução temporal desta regra, com a condição inicial

01110010...00101

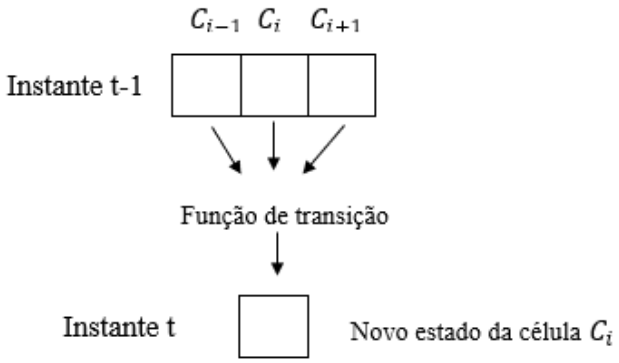


Figura2. Funcionamento da transição de estados.

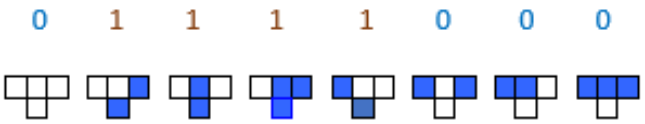


Figura3. Regras da função de transição para a regra 30.

0	1	1	1	0	0	1	0	...	0	0	1	0	1	**t=0**
1	1	0	0	1	1	1	1	...	0	1	1	0	1	**t=1**
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	0	1	0	1	0	0	1	...	1	0	0	0	1	**t=6**
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	0	1	0	1	1	0	0	...	1	1	0	1	1	**t=9**
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	0	1	0	1	0	1	0	...	0	0	0	1	1	**t=19**

Tabela2. Evolução temporal simbólica da Regra 30.

Apresentamos em seguida exemplos da evolução temporal de algumas outras regras, ver Figuras 5, 6 e 7, abaixo.

3 Autómatos celulares bidimensionais - jogo da vida

Tal como os autómatos a uma dimensão, a evolução de cada autómato celular a duas dimensões é determinada pelo seu estado inicial. O jogo da vida é o autómato celular mais conhecido e permite simular a vida e a morte

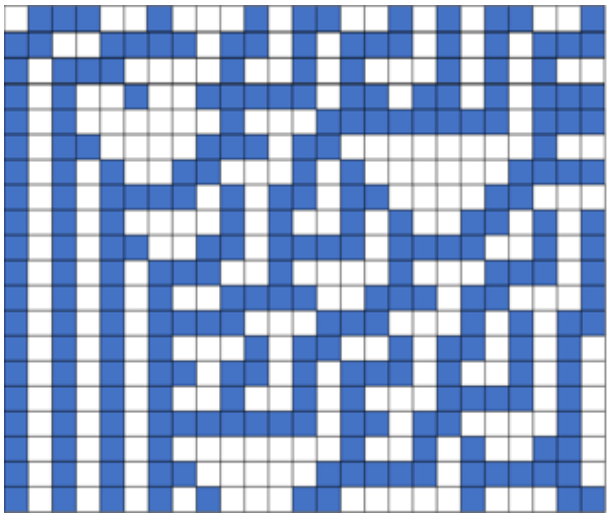


Figura4. Evolução temporal gráfica da regra 30.

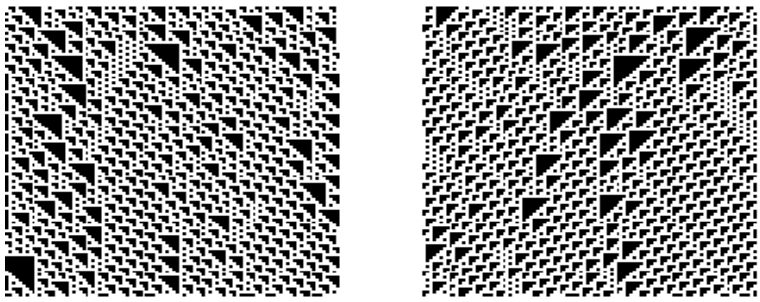


Figura5. Evolução temporal a regra 193 e a sua equivalente 137.
Código:11000001 e 10001001.

utilizando regras básicas de sobrevivência. A ideia principal é que um ser vivo necessita de outros seres vivos para sobreviver e procriar, tal que um excesso de densidade populacional provoca a morte do ser vivo devido à escassez de comida.
As leis genéricas de Conway [2] a serem tomadas em conta neste modelo são as seguintes (ver Figura 8):

1. Qualquer célula viva com menos de dois vizinhos vivos morre de solidão;
2. Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação;

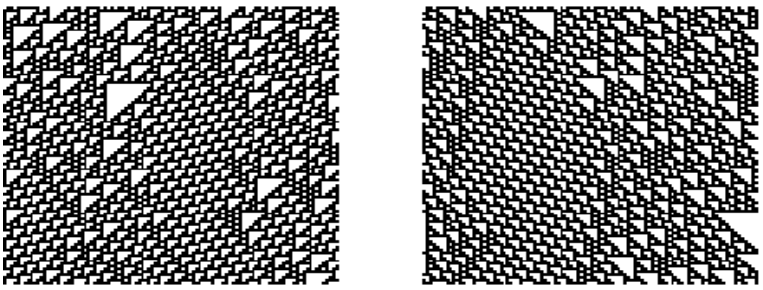


Figura6. Evolução temporal a regra 110 e a sua equivalente 124.
Código: 01101110 e 01111100.

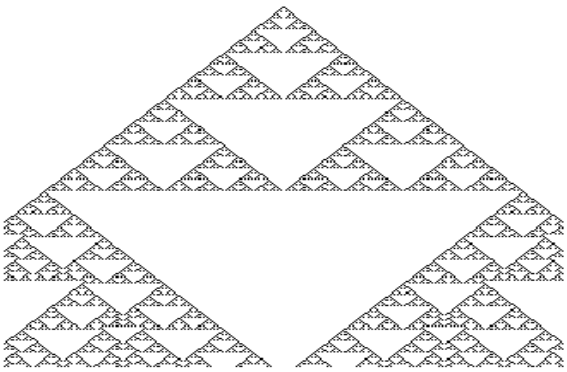


Figura7. Representação gráfica da evolução do autómato com a regra 18. O padrão inicial começa com apenas uma célula ativa.
Código: 00010010.

3. Qualquer célula morta com exatamente três vizinhos vivos torna-se uma célula viva;
4. Qualquer célula viva com dois ou três vizinhos vivos continua no mesmo estado para a próxima geração.

4 Modelo de propagação do fogo

De modo a dar resposta ao principal objetivo do presente trabalho, foi desenvolvido um código relativamente à propagação do fogo com auxílio ao *software Python*. O código esta disponível na seção final "Anexos". Este modelo é semelhante ao jogo da vida, no entanto apenas duas regras estão implícitas.

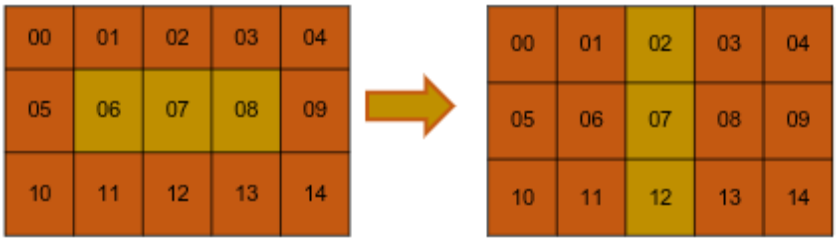


Figura8. Exemplo de aplicação das regras do jogo da vida.

Não existem fatores externos, como por exemplo o fogo ou a humidade, ou seja, o modelo é totalmente determinístico.
As regras subjacentes da formulação referida são as seguintes (ver exemplo na Figura 9):

1. Qualquer célula que estiver a arder pega fogo às restantes 8 vizinhas;
2. Qualquer célula que tenha pegado fogo fica queimada.

Deste modo, a propagação terá o espaço de estados $S = \{0, 1, 2\}$, onde 0 representa as células por arder, o 1 as células a arder e o 2 as células queimadas.
Para a simulação construímos um tabuleiro de 70 por 70 células. Para dar

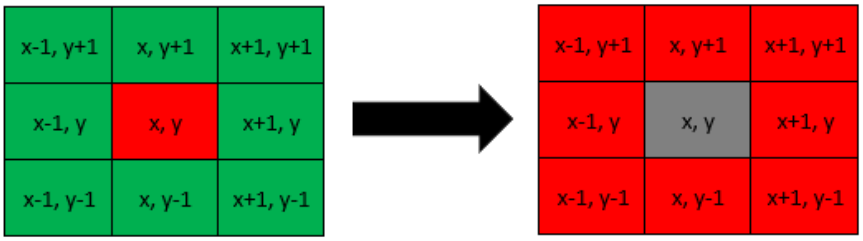


Figura9. Matriz das casa vizinhas (verde se o estado é 0, vermelho se o estado é 1 e cinzento se o estado é 2).

inicio à referida simulação é necessário colocar algumas células a arder, tendo duas formas distintas de o fazer: colocar o fogo de forma intencional em cada célula ou de forma aleatória, considerando 6 pontos de incêndio iniciais. Posteriormente, foi necessário a construção de mais três funções para auxiliar no funcionamento do algoritmo:

1. A função *calcula*, responsável por aplicar as regras em cada localização, constituída por dois ciclos, responsáveis por executar e guardar as transições de cada célula para o respetivo novo estado;
2. A função *vizinhos_queimados*, que indica se uma determinada célula tem vizinhos a queimar ou não;
3. A função *pinta*, é responsável por pintar o tabuleiro com as respetivas cores consoante o estado de cada célula.

Por fim, surge a função *executa* com o propósito de executar a simulação do fogo. Esta função, contém uma função interna ao módulo da interface gráfica, *tkinter*, que permite acelerar/desacelerar o processo de queima. Apesar do fogo não depender de fatores externos, pode ser interessante observar este aumento ou redução da velocidade de queima devido ao relevo do terreno, por exemplo.
Uma simulação do modelo proposto pode ser observado na Figura 10.

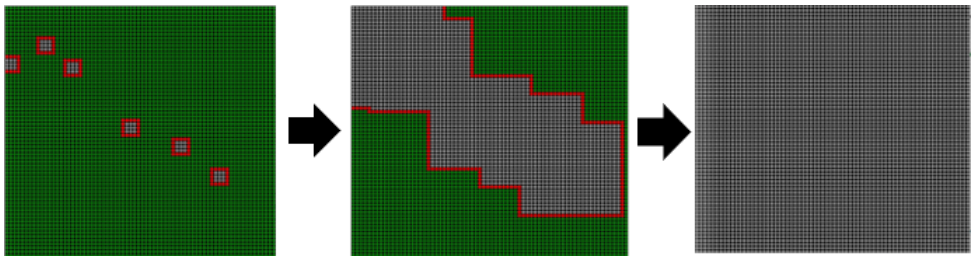


Figura10. Evolução da Propagação do fogo.

Referências

[1]Bouziani N., Ramos C., Tlemaçani M., Fernandes S., Characterization of Marble Blocks, 25-26 (preprint, 2021)
[2]Izhikevich, E. M., Conway, J. H., and Seth, A. (2015). Game of life. Scholarpedia, 10(6), 1816. von Neumann, J., The theory of self-reproducing automata, Rev. Mod. Phys.
[3]von Neumann, J., The theory of self-reproducing automata, Rev. Mod. Phys.(3) 55 (1983) 601-644.
[4]Wolfram, S., Statistical mechanics of cellular automata. Rev. Modern Phys. 55(1983), no. 3, 601–644.

5 Anexos

Sem zona segura

[1]:
```
import tkinter
import random

#Criação do tabuleiro com as casas todas, que é independente␣
 ↪dos valores inteiros de queimado (2), por queimar (0), a␣
 ↪queimar (1)
tabuleiro = [[0 for row in range(-1,71)] for column in␣
 ↪range(-1,71)]
#Criação do array com os valores inteiros que estarão em cada␣
 ↪casa (0,1,2), pela descrição mencionada anteriormente
executar = [[0 for row in range(-1,71)] for column in␣
 ↪range(-1,71)]
#Criação de um array Aque guarda os valores inteiros para␣
 ↪colocar no array executar, e os substituir depois de cada␣
 ↪iteração
copia = [[0 for row in range(-1,71)] for column in␣
 ↪range(-1,71)]

#Função para executar tudo
def executa():
    calcula()
    pinta()
    #Função interna ao tkinter que faz o numero de iterações␣
 ↪da app (chamando recursivamente)
    root.after(10, executa)

#Função para carregar o tabuleiro e os outros arrays
def carrega():
    for y in range(-1,71):
        for x in range(-1,71):
            executar[x][y] = 0
            copia[x][y] = 0
            tabuleiro[x][y] = canvas.create_oval((x*10, y*10,␣
 ↪x*10+10, y*10+10),outline="gray",fill="black")

    #Assumindo que só existem 6 localizações incendiadas
    '''
    for z in range(6):
```

```
        executar[random.randint(0,70)][random.randint(0,70)]␣
↪= 1

    '''
    #fogo prepositadamente
    executar[55][45] = 1
    executar[10][10] = 1
    executar[45][37] = 1
    executar[32][32] = 1
    executar[1][15] = 1
    executar[17][16] = 1



#Função para aplicar as regras para cada localização
def calcula():
    for y in range(0,70):
        for x in range(0,70):
            queimados = vizinhos_queimados(x,y)
            if executar[x][y] == 1 : copia[x][y] = 2
            if executar[x][y] == 0 and queimados: copia[x][y]␣
↪= 1
    for y in range(0,70):
        for x in range(0,70):
            executar[x][y] = copia[x][y]

#Função que da uma localização que diz se tem vizinhos a␣
↪queimar ou não
def vizinhos_queimados(x,y):
    queimados = False
    if executar[x-1][y+1] == 1:
        queimados = True
    if executar[x][y+1] == 1:
        queimados = True
    if executar[x+1][y+1] == 1:
        queimados = True
    if executar[x-1][y] == 1:
        queimados = True
    if executar[x+1][y] == 1:
        queimados = True
    if executar[x-1][y-1] == 1:
        queimados = True
    if executar[x][y-1] == 1:
```

```
        queimados = True
    if executar[x+1][y-1] == 1:
        queimados = True
    return queimados

#Função que dependendo do array executar pinta o tabuleiro␣
↪com as cores respetivas
def pinta():
    for y in range(70):
        for x in range(70):
            if executar[x][y]==0:
                canvas.itemconfig(tabuleiro[x][y],␣
↪fill="green", outline="black")
            if executar[x][y]==1:
                canvas.itemconfig(tabuleiro[x][y], fill="red")
            if executar[x][y]==2:
                canvas.itemconfig(tabuleiro[x][y],␣
↪fill="grey", outline="black")



root = tkinter.Tk()
root.title("Simulação de propagação de fogo")
canvas = tkinter.Canvas(root, width=700, height=700,␣
↪highlightthickness=0, bd=0.05, bg='black')
canvas.pack()
carrega()
executa()
root.mainloop()
```

Com zona segura

[6]:
```
import tkinter
import random

#Criação do tabuleiro com as casas todas, que é independente␣
↪dos valores inteiros de queimado (2), por queimar (0), a␣
↪queimar (1)
tabuleiro = [[0 for row in range(-1,71)] for column in␣
↪range(-1,71)]
#Criação do array com os valores inteiros que estarão em cada␣
↪casa (0,1,2), pela descrição mencionada anteriormente
```

```
executar = [[0 for row in range(-1,71)] for column in␣
↪range(-1,71)]
#Criação de um array que guarda os valores inteiros para␣
↪colocar no array executar, e os substituir depois de cada␣
↪iteração
copia = [[0 for row in range(-1,71)] for column in␣
↪range(-1,71)]

#Função para executar tudo
def executa():
    calcula()
    pinta()
    #Função interna ao tkinter que faz o numero de iterações␣
↪da app (chamando recursivamente)
    root.after(5, executa)

#Função para carregar o tabuleiro e os outros arrays
def carrega():
    for y in range(-1,71):
        for x in range(-1,71):
            executar[x][y] = 0
            copia[x][y] = 0
            tabuleiro[x][y] = canvas.create_oval((x*10, y*10,␣
↪x*10+10, y*10+10),outline="gray",fill="black")

    #Assumindo que só existem 5 localizações incencidiadas
    for z in range(5):
        executar[random.randint(0,70)][random.randint(0,70)]␣
↪= 1

    for z in range(40):
        executar[random.randint(0,70)][random.randint(0,70)]␣
↪= 3

    """
    executar[1][1] = 1
    executar[10][10] = 1
    executar[68][68] = 1
    executar[32][32] = 1
    executar[1][15] = 1
    executar[17][16] = 1
    """
```

```
#Função para aplicar as regras para cada localização
def calcula():
    for y in range(0,70):
        for x in range(0,70):
            queimados = vizinhos_queimados(x,y)
            if executar[x][y] == 1 : copia[x][y] = 2
            if executar[x][y] == 0 and queimados: copia[x][y]
 ↪= 1
            if executar[x][y] == 3 : copia[x][y] = 3
    for y in range(0,70):
        for x in range(0,70):
            executar[x][y] = copia[x][y]

#Função que dada uma localização se tem vizinhos queimados ou
 ↪nao
def vizinhos_queimados(x,y):
    queimados = False
    if executar[x-1][y+1] == 1:
        queimados = True
    if executar[x][y+1] == 1:
        queimados = True
    if executar[x+1][y+1] == 1:
        queimados = True
    if executar[x-1][y] == 1:
        queimados = True
    if executar[x+1][y] == 1:
        queimados = True
    if executar[x-1][y-1] == 1:
        queimados = True
    if executar[x][y-1] == 1:
        queimados = True
    if executar[x+1][y-1] == 1:
        queimados = True
    return queimados

#Função que dependendo do array executar pinta o tabuleiro
 ↪com as cores respetivas
def pinta():
    for y in range(70):
        for x in range(70):
            if executar[x][y]==0:
```

```
                canvas.itemconfig(tabuleiro[x][y],␣
 ↪fill="OliveDrab1", outline="black")
            if executar[x][y]==3:
                canvas.itemconfig(tabuleiro[x][y],␣
 ↪fill="green", outline="black")
            if executar[x][y]==1:
                canvas.itemconfig(tabuleiro[x][y], fill="red")
            if executar[x][y]==2:
                canvas.itemconfig(tabuleiro[x][y],␣
 ↪fill="grey", outline="black")



root = tkinter.Tk()
root.title("Simulação de propagação de fogo")
canvas = tkinter.Canvas(root, width=700, height=700,␣
 ↪highlightthickness=0, bd=0.05, bg='black')
canvas.pack()
carrega()
executa()
root.mainloop()
```

Índice

Ciolpan, Gheorghe, *1*