TP2-Exercicio2

November 15, 2022

1 TP2

1.1 Grupo 15

Carlos Eduardo Da Silva Machado A96936

Gonçalo Manuel Maia de Sousa A97485

1.2 Problema 2

1.2.1 Descrição do 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".
    2. No estado inicial todas as células normais estão mortas excepto um quadrado 3×3, designado por "centro", aleatoriamente posicionado formado apenas por células vivas.
    3. 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.
    4. 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.

1.2.2 Abordagem do Problema

Para resolver este problema construimos uma máquina de estados finita que representa esse autómato, de modo que cada estado é representado unicamente por uma configuração específica de células vivas e mortas.

São parametros deste problema:
1. ρ a probabilidade de cada uma das células da borda estar viva.
2. N o tamaho do quadro.
3. k o número de estados inicialmente reservados para o autómato (caso seja necessário, este número é duplicado).
4. *bound* o número de estados até ao primeiro repetido.
5. *centro* a posição do centro aleatóriamente posicionado.

Para além do autómato é necessário verificar de as seguintes propriedades de aplicam:
1. Todos os estados acessíveis contém pelo menos uma célula viva (apenas são verificadas as células normais).
2. Toda a célula normal está viva pelo menos uma vez em algum estado acessível.

1.3 Código Python

Esta secção de codigo serve para importar as bibliotecas nesseçárias para a realização do trabalho.

[3]:
```
import matplotlib.pyplot as plt
from pysmt.shortcuts import *
from pysmt.typing import INT
import random as rn
from pprint import pprint
```

Aqui são apresentadas funções auxiliares, uma delas que gera aleatóriamente a borda, outro que gera aleatoriamente o centro e uma que calcula a vizinhança de um ponto.

[4]:
```
def gera_rand(p,size):
    return rn.choices([0,1],[1-p,p],k=size)

def gera_border(p,size):
    return (gera_rand(p,size),gera_rand(p,size-1))

def gera_centro(N):
    return (rn.randint(3,N-1),rn.randint(3,N-1))

def PontosProximos(pos,N):
    x,y = pos
    p = [(x+1,y),(x-1,y),(x,y+1),(x,y-1),(x-1,y-1),(x+1,y+1),(x-1,y+1),(x+1,␣
 ↪y-1)]
    return list(filter(lambda x: x[0] < N+1 and x[1] < N+1,  p))
```

2 Construção da máquina de estados finita

Função de declaração.

[5]:
```
def declare(c,N):
    state = dict()
    for i in range(1,N+1):
        for j in range(1,N+1):
            state[f'({i},{j})'] = Symbol(f'({i},{j})'+str(c),INT)
    return state
```

Função inicial. Nesta função, garantimos que a borda e o centro são iguais aos que foram gerados aleatoriamente e garantimos também que os pontos do centro estão vivos, enquanto que os restantes pontos normais estão mortos.

[6]:
```
def init(state,N,centro,border):

    l = list()

    for i,x in enumerate(border[0],start=1):
        l.append(Equals(state[f'({i},1)'], Int(x)))
    for j,x in enumerate(border[1],start=2):
        l.append(Equals(state[f'(1,{j})'], Int(x)))

    ic,jc = centro

    for i in range(2,N+1):
        for j in range(2,N+1):
            if i in [ic-1,ic, ic+1] and j in [jc-1, jc, jc+1]:
                l.append(Equals(state[f'({i},{j})'], Int(1)))
            else:
                l.append(Equals(state[f'({i},{j})'], Int(0)))

    #l.append(Equals(state['pc'], Int(0)))
    return And(l)
```

Função de transição. Garantimos que a borda se mantem constante e realiza uma de quatro ações para as células normais:
1. Se a célula estiver morta e possuir exatamente 3 vizinhos vivos, a mesma passa a viva no estado seguinte.
2. Se a célula estiver viva e possuir exatamente 2 ou 3 vizinhos vivos, mantém-se viva no estado seguinte.
3. Se a célula estiver viva e não possuir exatamente 2 ou 3 vizinhos vivos, então morre no próximo estado.
4. Se a célula estiver morta e não possuir exatamente 3 vizinhos vivos, então mantém-se morta no próximo estado.

$$\mathsf{trans}(x_{i,j}, x'_{i,j} : i,j \in N,) \equiv$$

$$\begin{cases} \forall_{i\in\{1\ldots N\}} & (x'_{i,1} = x_{i,1}) \\ \forall_{j\in\{1\ldots N\}} & (x'_{1,j} = x_{1,j}) \end{cases} \wedge \bigwedge \begin{cases} \forall_{i\in\{2\ldots N\}}\cdot\forall j \in \{2\ldots N\} & (x_{i,j} = 0) \wedge (neighbors(x_{i,j}) = 3) \wedge (x'_{i,j} = 1) \\ \forall_{i\in\{2\ldots N\}}\cdot\forall j \in \{2\ldots N\} & (x_{i,j} = 1) \wedge ((neighbors(x_{i,j}) = 2) \vee neighbors(x_{i,j} \\ \forall_{i\in\{2\ldots N\}}\cdot\forall j \in \{2\ldots N\} & (x_{i,j} = 1) \wedge ((neighbors(x_{i,j}) < 2) \vee neighbors(x_{i,j} \\ \forall_{i\in\{2\ldots N\}}\cdot\forall j \in \{2\ldots N\} & (x_{i,j} = 0) \wedge ((neighbors(x_{i,j}) \leq 2) \vee neighbors(x_{i,j} \end{cases}$$

[7]:
```
def trans(curr,prox,N):
    l = []
    for i in range(1,N+1):
        l.append(Equals(prox[f'({i},1)'], curr[f'({i},1)']))
    for j in range(2,N+1):
        l.append(Equals(prox[f'(1,{j})'], curr[f'(1,{j})']))
    for i in range(2,N+1):
        for j in range(2,N+1):
```

```
            valor = Plus([curr[f'({x},{y})'] for x,y in PontosProximos((i,j),N)])
            tBorn = And(Equals(curr[f'({i},{j})'], Int(0)), Equals(valor,␣
↪Int(3)), Equals(prox[f'({i},{j})'], Int(1)))
            tStayAlive = And(Equals(curr[f'({i},{j})'], Int(1)),␣
↪Or(Equals(valor, Int(2)),Equals(valor, Int(3))), Equals(prox[f'({i},{j})'],␣
↪Int(1)))
            tDie = And(Equals(curr[f'({i},{j})'], Int(1)), Or(valor <␣
↪Int(2),valor > Int(3)), Equals(prox[f'({i},{j})'], Int(0)))
            tStillDead = And(Equals(curr[f'({i},{j})'], Int(0)), Or(valor <=␣
↪Int(2),valor > Int(3)), Equals(prox[f'({i},{j})'],Int(0)))

            l.append(Or(tBorn, tStayAlive, tDie,tStillDead))
    return And(l)
```

Função que usa *SMT solver* para gerar um possível traço de execução do programa, e mostra o estado do autómato na forma de um quadrado $N \times N$ tal que as células mortas são apresentadas com cor escura e as vivas com cor clara.

[8]:
```
def resolve(N,k,centro,border):
    with Solver(name="z3") as s:

        flag = True

        while flag:

            bound = 0

            mat_list = list()

            trace = [declare(i,N) for i in range(k)]

            s.add_assertion(init(trace[0],N,centro,border))

            for i in range(k-1):
                s.add_assertion(trans(trace[i], trace[i+1],N))

            mat_set = set()

            if s.solve():
                for i in range(k):
                    matrix = []
                    for x in range(1,N+1):
                        #print([s.get_value(trace[i][f'({k},{x})']) for k in␣
↪range(1,N+1)])
                        matrix.append([(s.get_value(trace[i][f'({k},{x})']).
↪constant_value()) for k in range(1,N+1)])
```

```
                    #pprint(matrix)
                    matrix_str = str(matrix)
                    if matrix_str in mat_set:
                        flag = False
                        break
                    mat_set.add(matrix_str)
                    mat_list.append(matrix)
                    #pprint(matrix)
                    bound += 1
            else:
                print('Não foi possivel resolver')
            #print(trace)

            k = 2*k
            #print('cicle')

            s.reset_assertions()

        for i,l in enumerate(mat_list):
            plt.imshow(l)
            plt.axis('off')
            print(i)
            plt.show()
        return bound
```

Função para testar invariante para um traço limitado.

[9]:
```
def bmc_always(declare,init,trans,inv,K,N,centro,borda):
    for k in range(1,K+1):
        with Solver(name="z3") as s:
            trace = [declare(i,N) for i in range(k)]
            s.add_assertion(init(trace[0],N,centro,borda))
            for i in range(k-1):
                s.add_assertion(trans(trace[i], trace[i+1],N))
            s.add_assertion(Not(inv(trace[k-1],N)))
            if s.solve():
                print("A propriedade é inválida")
                return
    print("A propriedade é válida")
```

Função para testar a inevitabilidade de alguma propriedade.

[10]:
```
def bmc_eventually(declare,init,trans,prop,bound,N,centro,borda):
    for k in range(1,bound+1):
         with Solver(name="z3") as s:
            # completar
            trace = [declare(i,N) for i in range(k)]
```

```
            s.add_assertion(init(trace[0],N,centro,borda))
            for i in range(k-1):
                s.add_assertion(trans(trace[i], trace[i+1],N))

            for i in range(k):
                s.add_assertion(Not(prop(trace,k,N)))

            s.add_assertion(Or(trans(trace[k-1],trace[i],N) for i in range(k)))

            if s.solve():
                print("Propriedade é inválida")
                return

    print("A propriedade é válida")
```

Propriedade A: Todos os estados acessíveis contém pelo menos uma célula viva.

[11]:
```
def propA(state,N):
    return Plus([state[f'({i},{j})'] for i in range(1,N+1) for j in␣
↪range(1,N+1)]) >= 1
```

Propriedade A2: Todos os estados acessíveis contém pelo menos uma célula normal viva.

[12]:
```
def propA2(state,N):
    return Plus([state[f'({i},{j})'] for i in range(2,N+1) for j in␣
↪range(2,N+1)]) >= 1
```

Propriedade B: Toda a célula normal está viva pelo menos uma vez em algum estado acessível.

[13]:
```
def propB(trace,k,N):
    l = []
    for i in range(2,N+1):
        for j in range(2,N+1):
            l.append(Plus([trace[x][f'({i},{j})'] for x in range(k)]) >= 1)
    return And(l)
```

2.0.1 Exemplos e testes de aplicação.

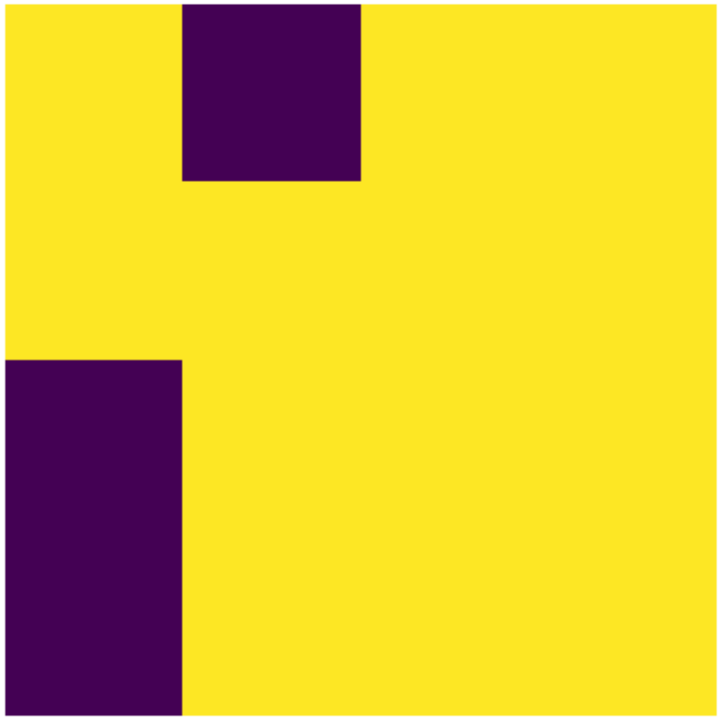
Exemplo 1 Neste exemplo procuramos mostrar que é encontrado um loop mesmo que k inicial não seja suficientemente grande.

[12]:
```
N = 4
p = 0.75
k = 2
centro = gera_centro(N)
border = gera_border(p,N)
print(centro)
print(border)
```

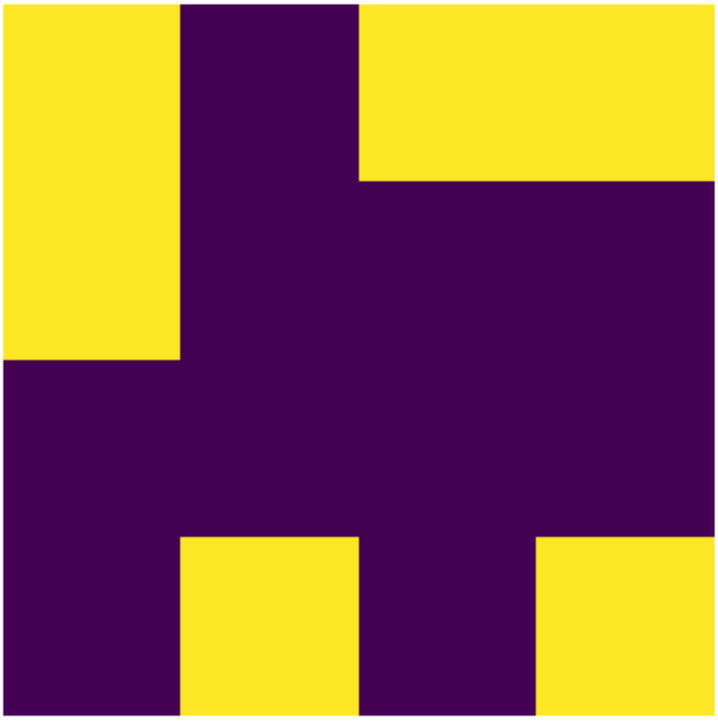
```
(3, 3)
([1, 0, 1, 1], [1, 0, 0])
```

[13]:
```
print(bound := resolve(N,k,centro,border))
```

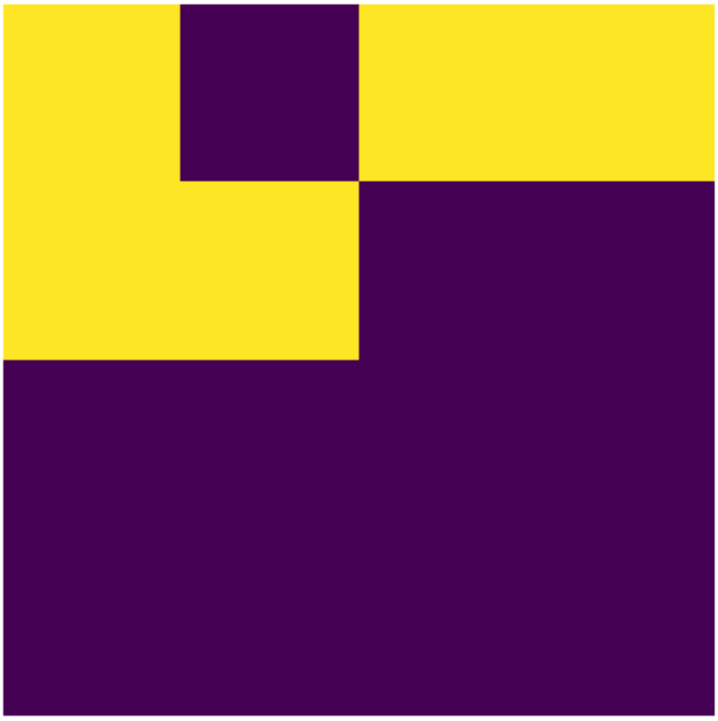
```
0
```



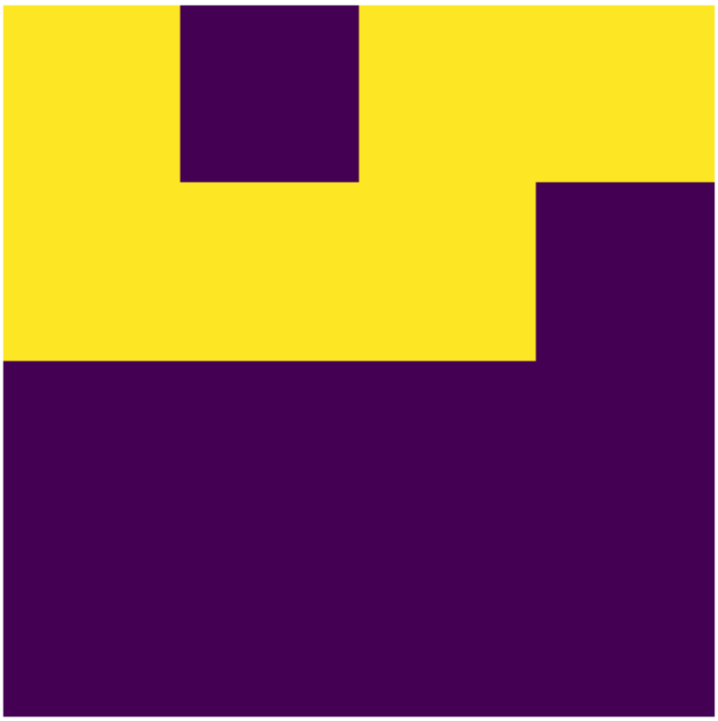
```
1
```



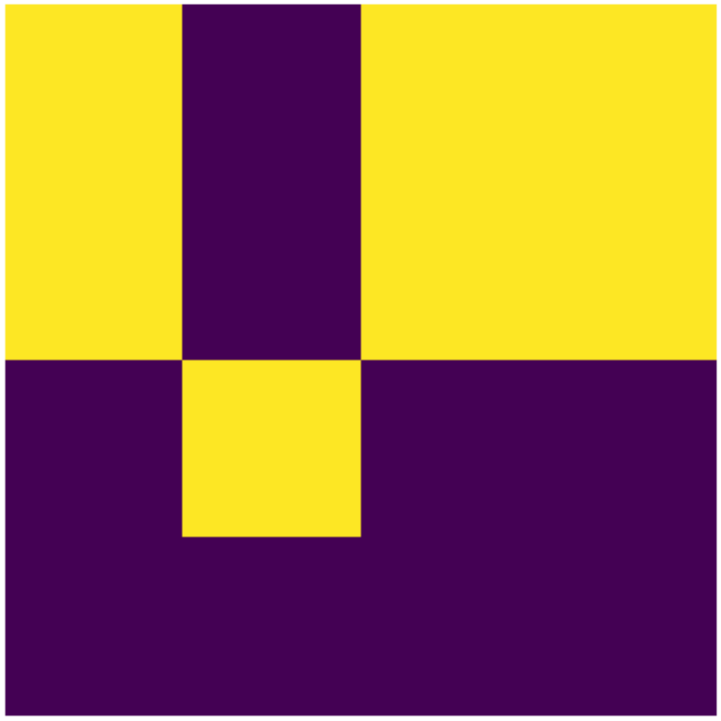
2



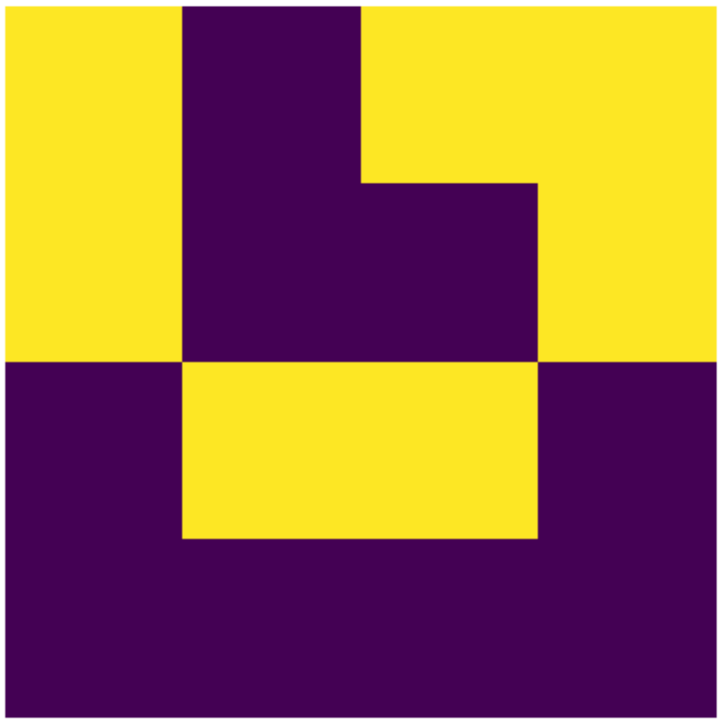
3



4



5



```
6
```

[14]:
```
print("Propriedade A:")
bmc_always(declare,init,trans,propA,bound,N,centro,border)
print("Propriedade A, mas para células normais apenas:")
bmc_always(declare,init,trans,propA2,bound,N,centro,border)
print("Propriedade B:")
bmc_eventually(declare,init,trans,propB,bound,N,centro,border)
```

```
Propriedade A:
A propriedade é válida
Propriedade A, mas para células normais apenas:
A propriedade é válida
Propriedade B:
A propriedade é válida
```

Exemplo 2 Este é um exemplo em a proposição A é verdadeira mas a B é falsa.

[15]:
```
N = 5
p = 1
k = 2
centro = gera_centro(N)
border = gera_border(p,N)
print(centro)
print(border)
```

```
(3, 4)
([1, 1, 1, 1, 1], [1, 1, 1, 1])
```

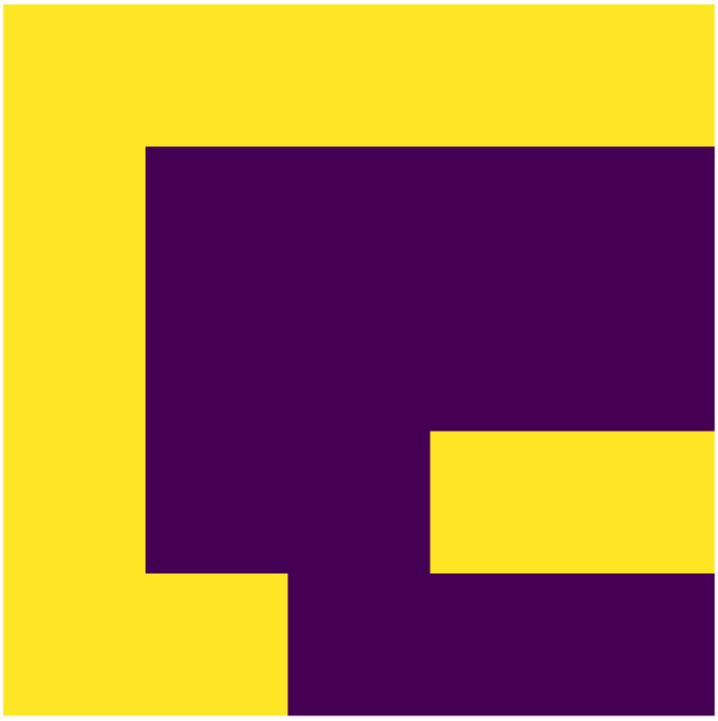
[16]:
```
print(bound := resolve(N,k,centro,border))
```

```
0
```

1

2

3



4

5

6

7

```
8
```

[17]:
```
print("Propriedade A:")
bmc_always(declare,init,trans,propA,bound,N,centro,border)
print("Propriedade A, mas para células normais apenas:")
bmc_always(declare,init,trans,propA2,bound,N,centro,border)
print("Propriedade B:")
bmc_eventually(declare,init,trans,propB,bound,N,centro,border)
```

```
Propriedade A:
A propriedade é válida
Propriedade A, mas para células normais apenas:
A propriedade é válida
Propriedade B:
Propriedade é inválida
```

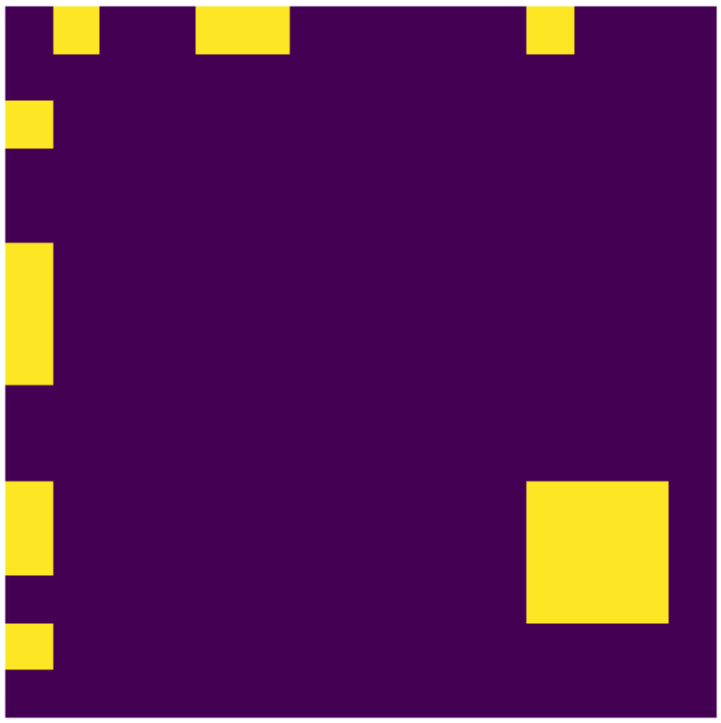
Exemplo 3 Este é um exemplo com um quadro maior.

[21]:
```
N = 15
p = 0.5
k = 50
centro = gera_centro(N)
border = gera_border(p,N)
print(centro)
print(border)
```

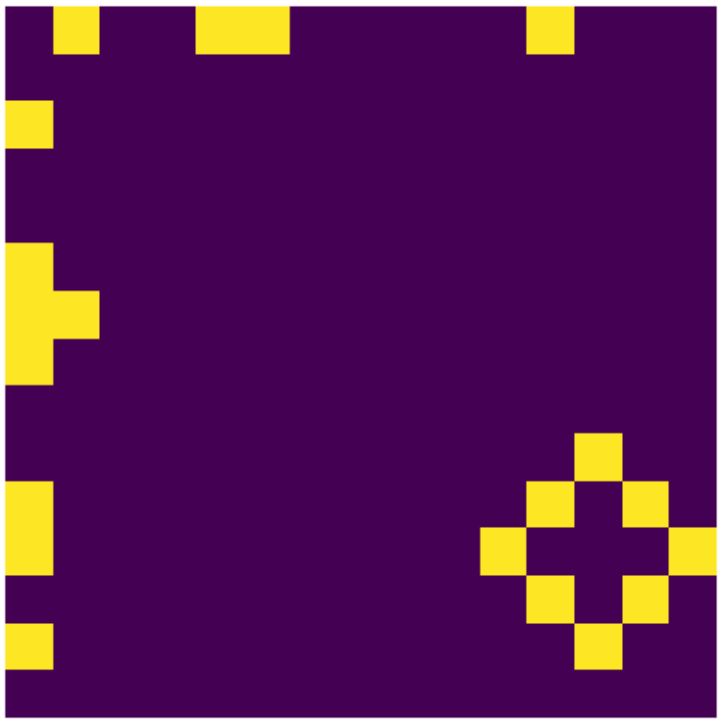
```
(13, 12)
([0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0], [0, 1, 0, 0, 1, 1, 1, 0, 0, 1,
1, 0, 1, 0])
```

[22]:
```
print(bound := resolve(N,k,centro,border))
```

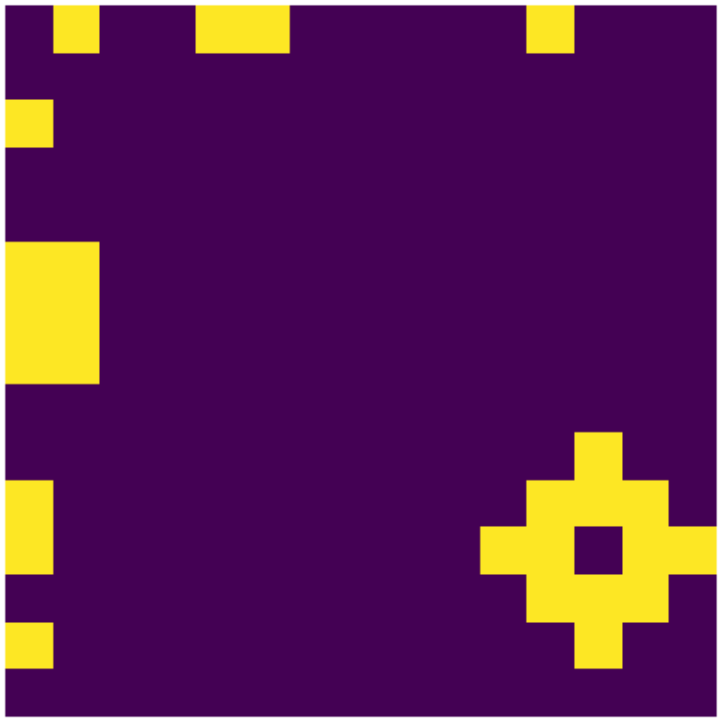
```
0
```



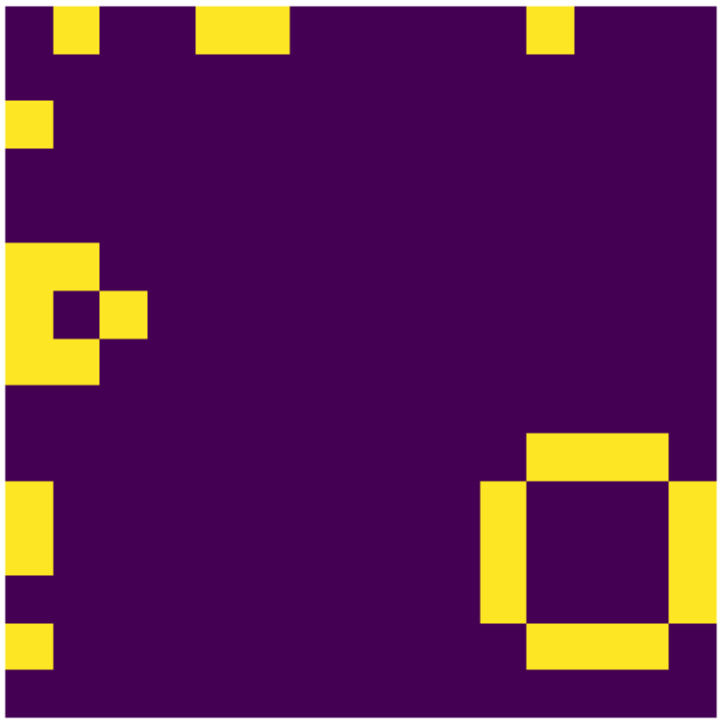
1



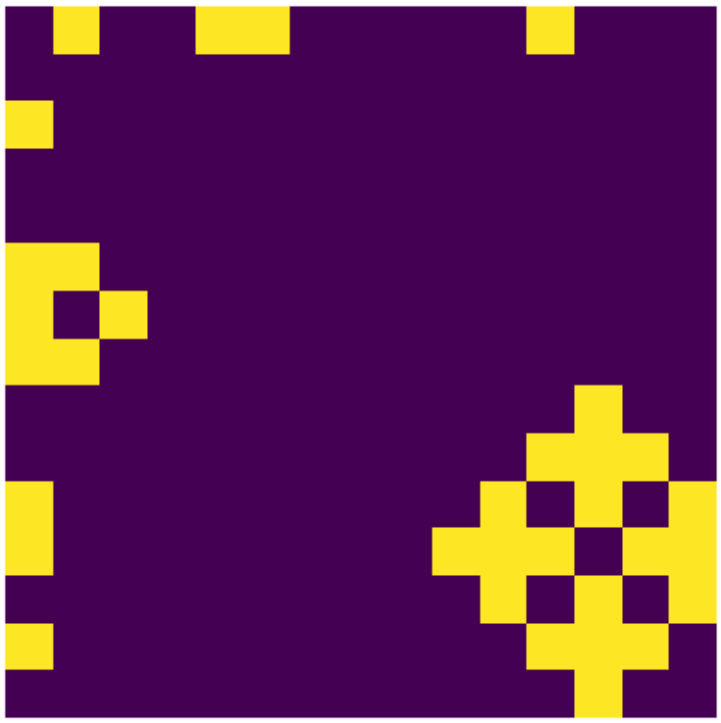
2



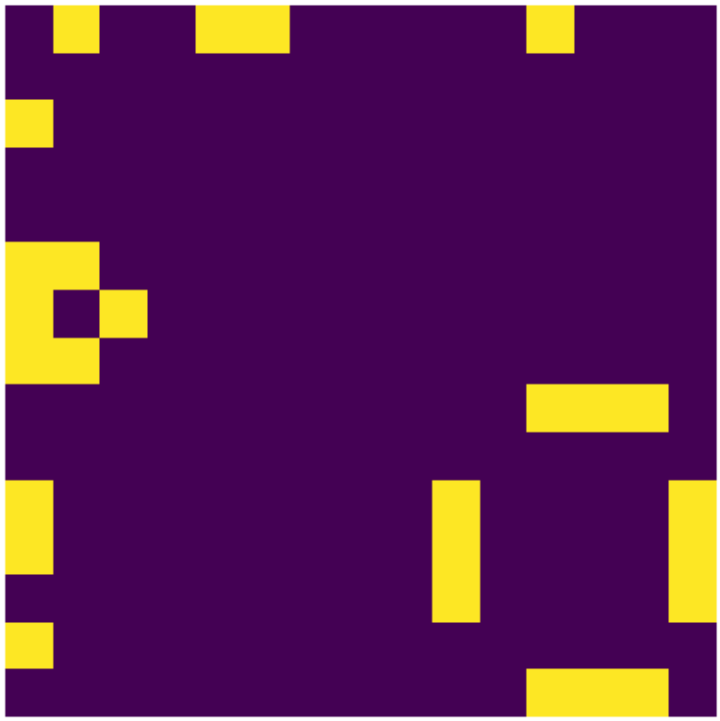
3



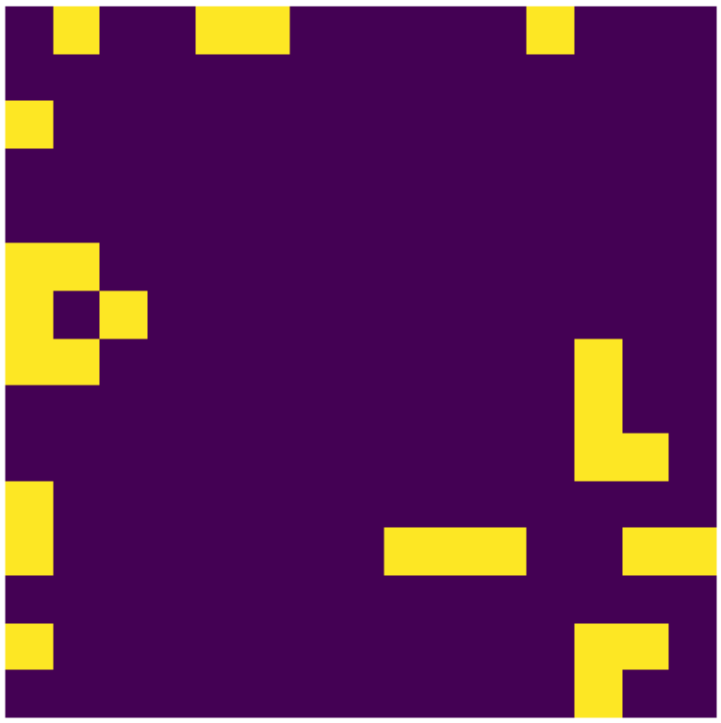
4



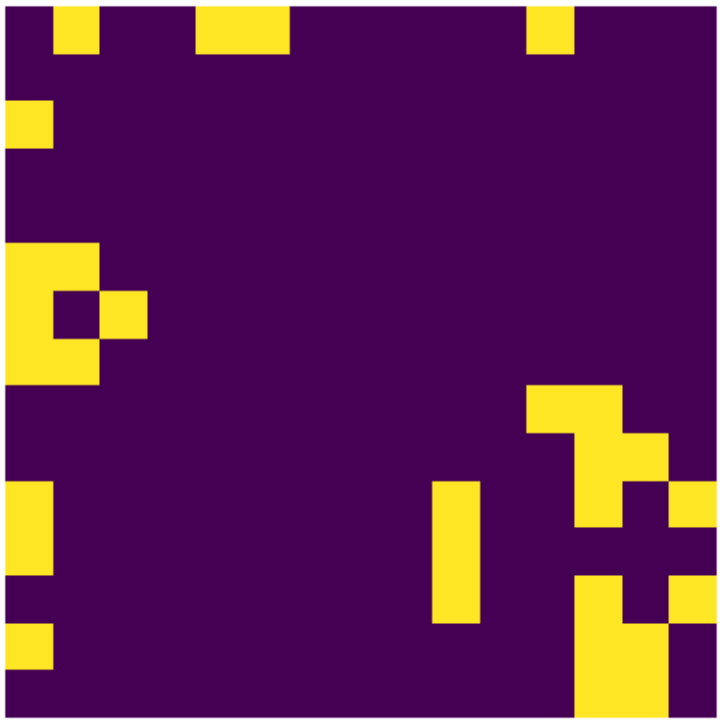
5



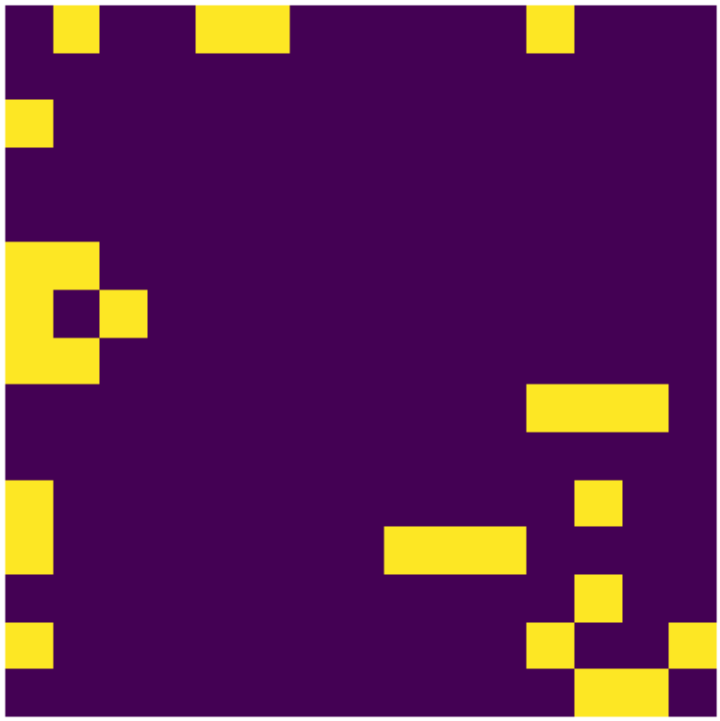
6



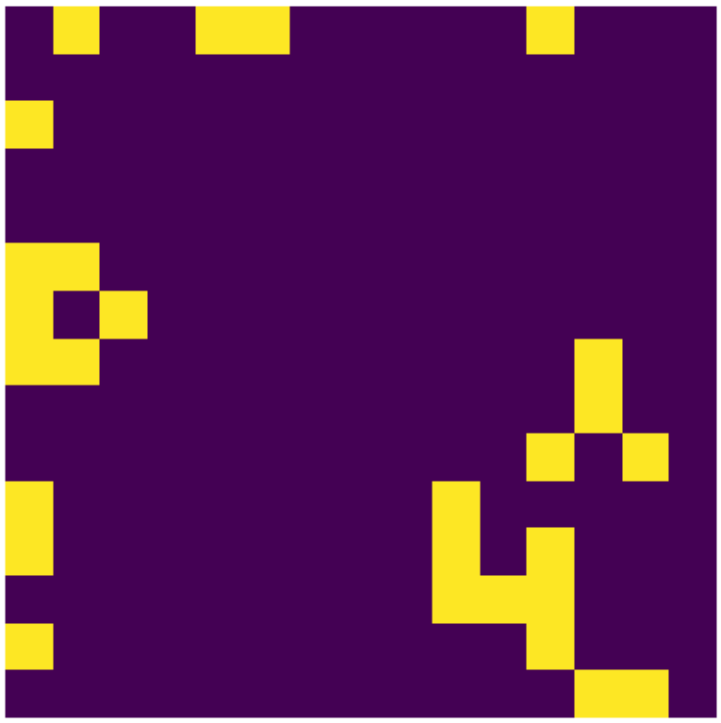
7



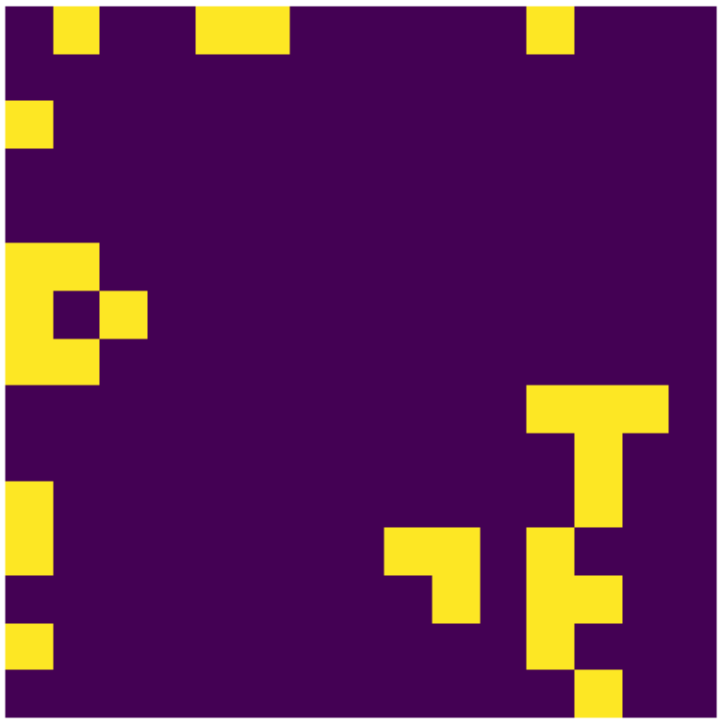
8



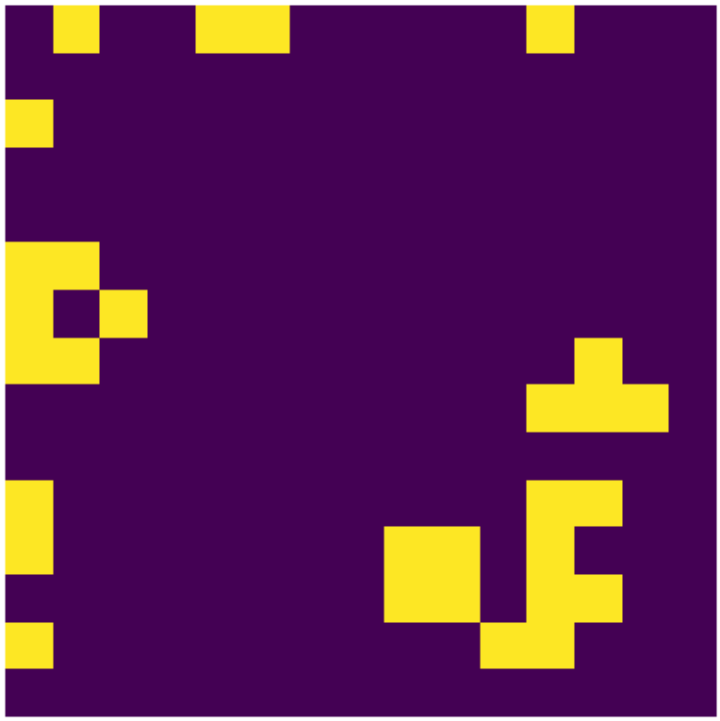
9



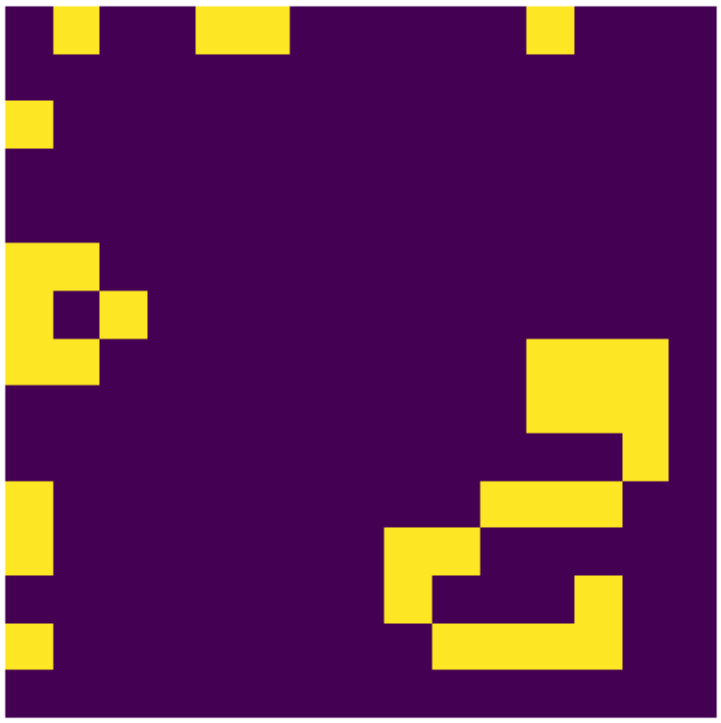
10



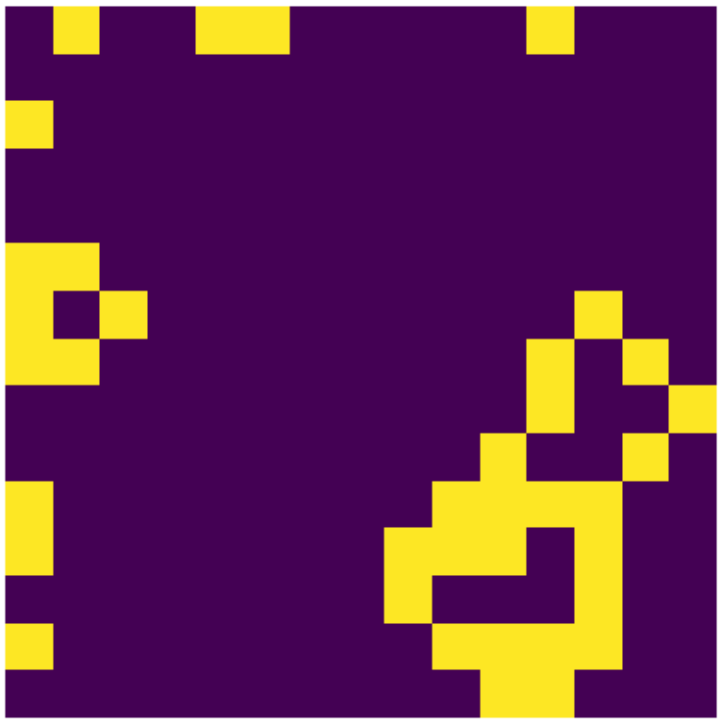
11



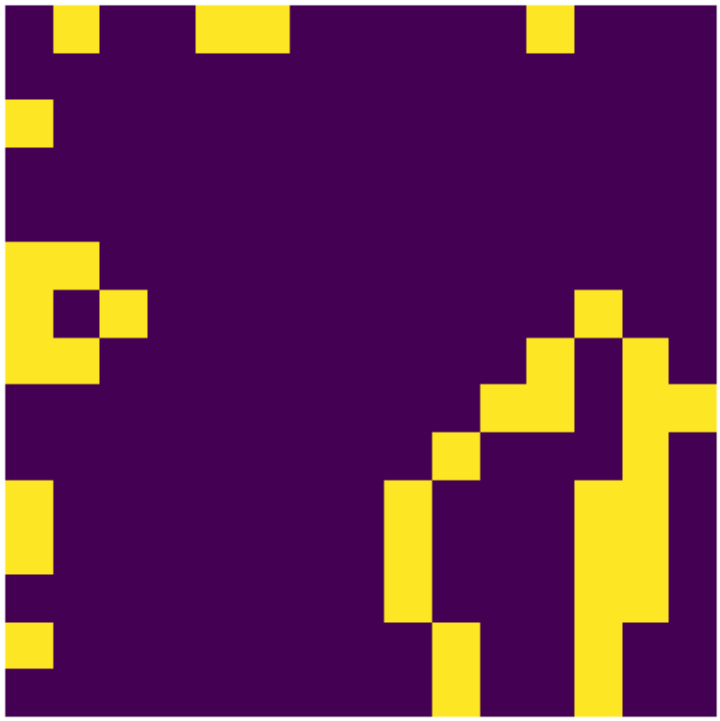
12



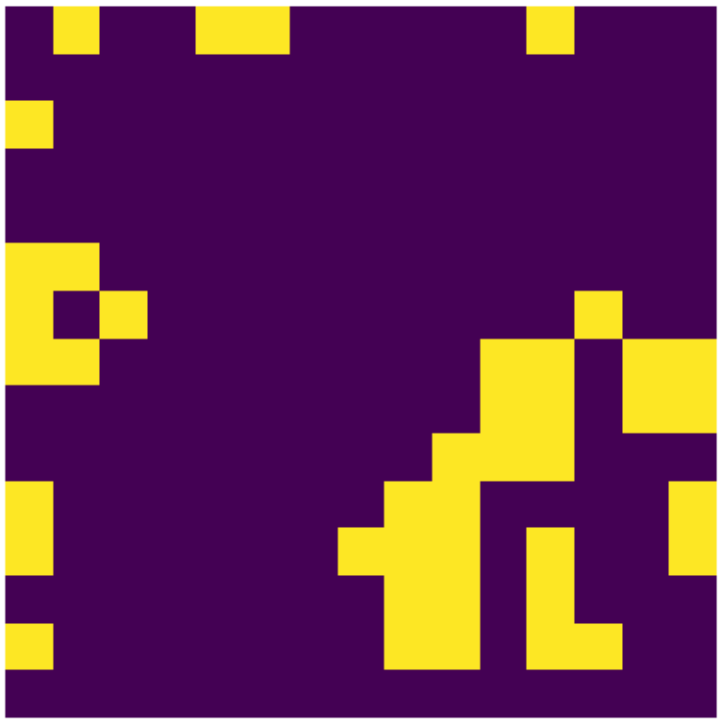
13



14

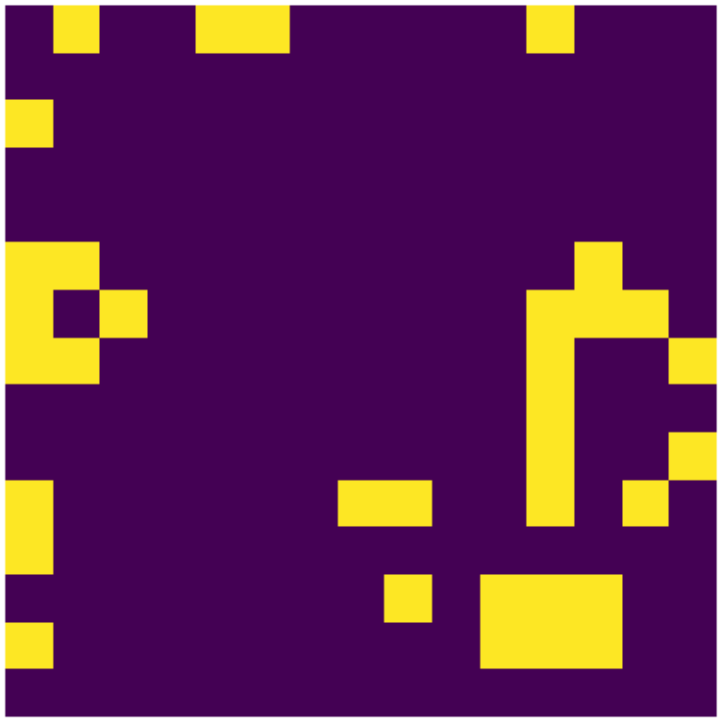


15



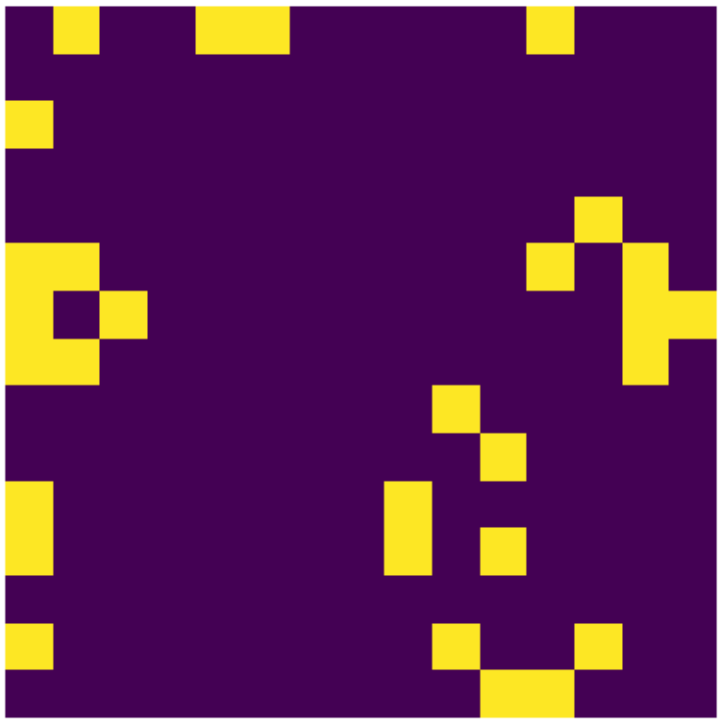
16

17



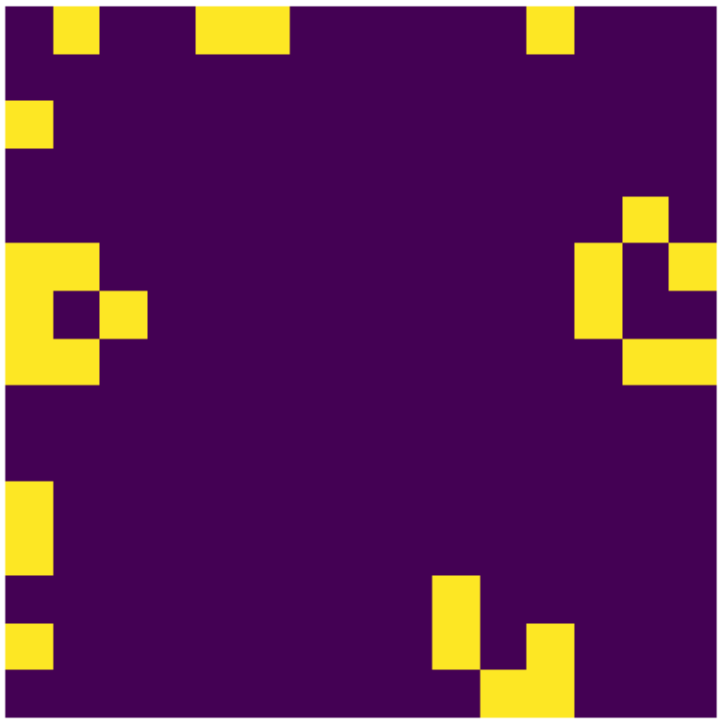
18

19

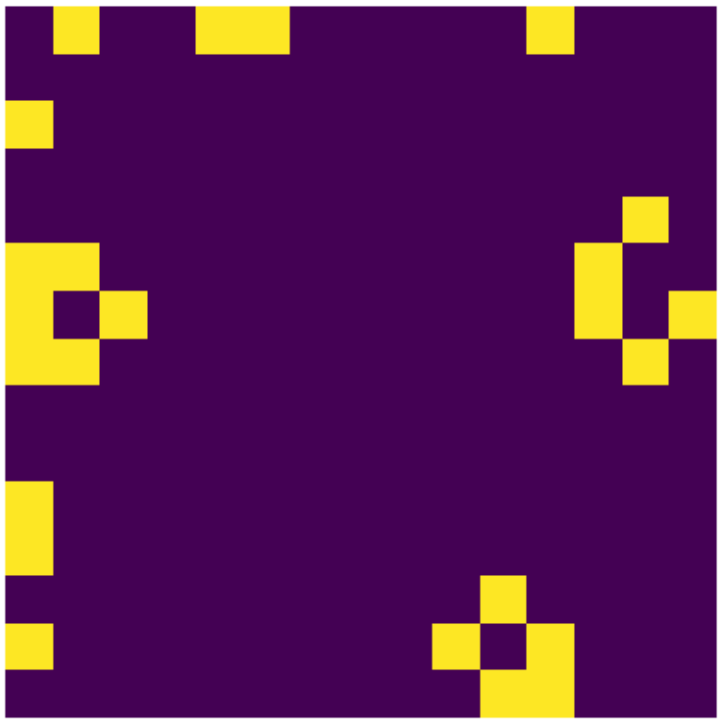


20

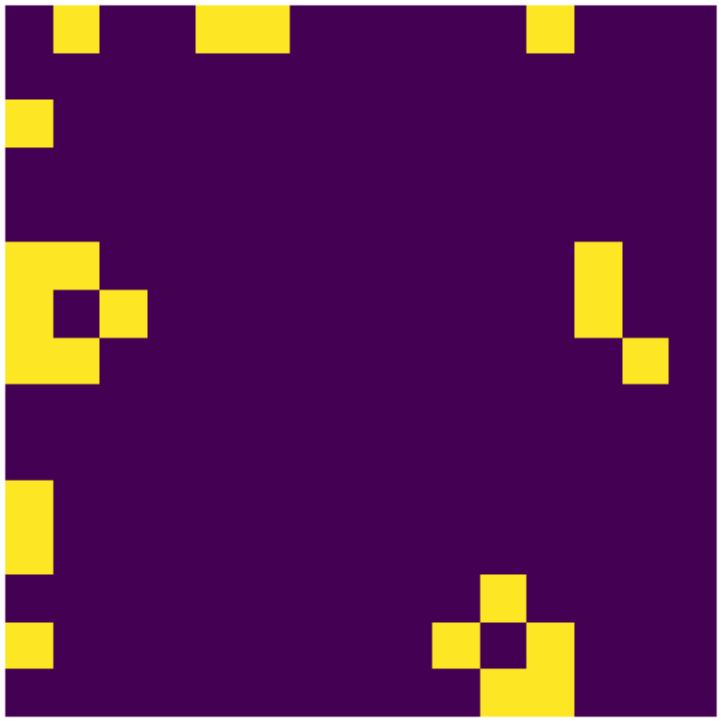
21



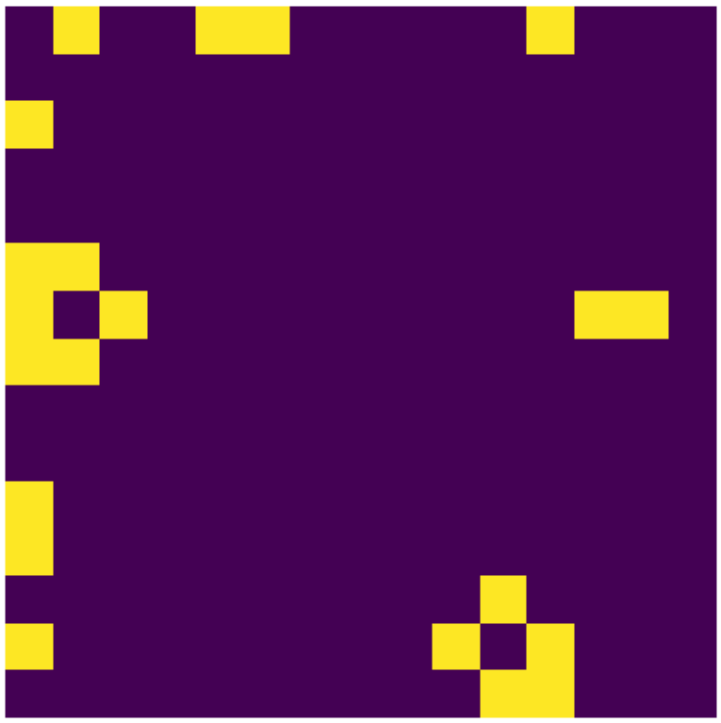
22



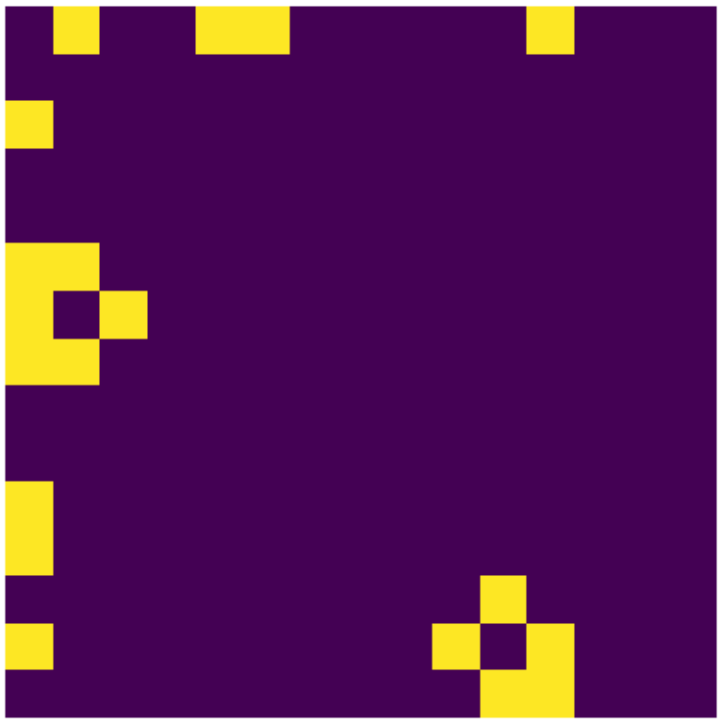
23



24



25



26

[23]:
```
print("Propriedade A:")
bmc_always(declare,init,trans,propA,bound,N,centro,border)
print("Propriedade A, mas para células normais apenas:")
bmc_always(declare,init,trans,propA2,bound,N,centro,border)
print("Propriedade B:")
bmc_eventually(declare,init,trans,propB,bound,N,centro,border)
```

```
Propriedade A:
A propriedade é válida
Propriedade A, mas para células normais apenas:
A propriedade é válida
Propriedade B:
Propriedade é inválida
```

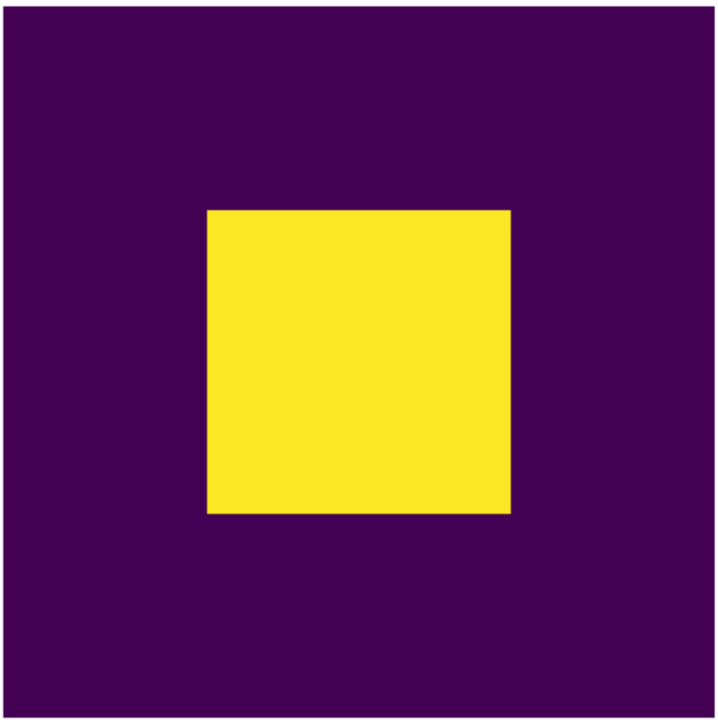
Exemplo 4 Este é um exemplo em que a propriedade A e B são falsas.

[24]:
```
N = 7
p = 0
k = 20
centro = gera_centro(N)
border = gera_border(p,N)
print(centro)
print(border)
```

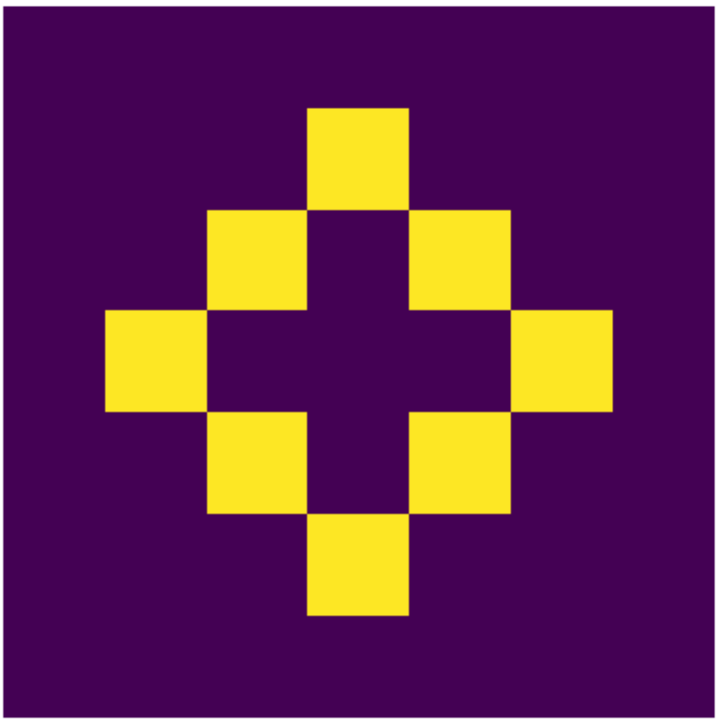
```
(4, 4)
([0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0])
```

[25]:
```
print(bound := resolve(N,k,centro,border))
```

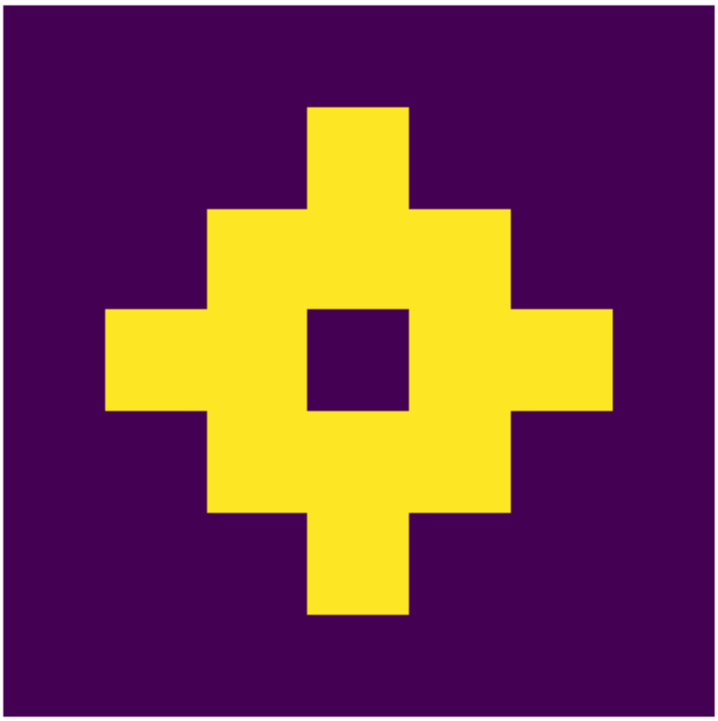
```
0
```



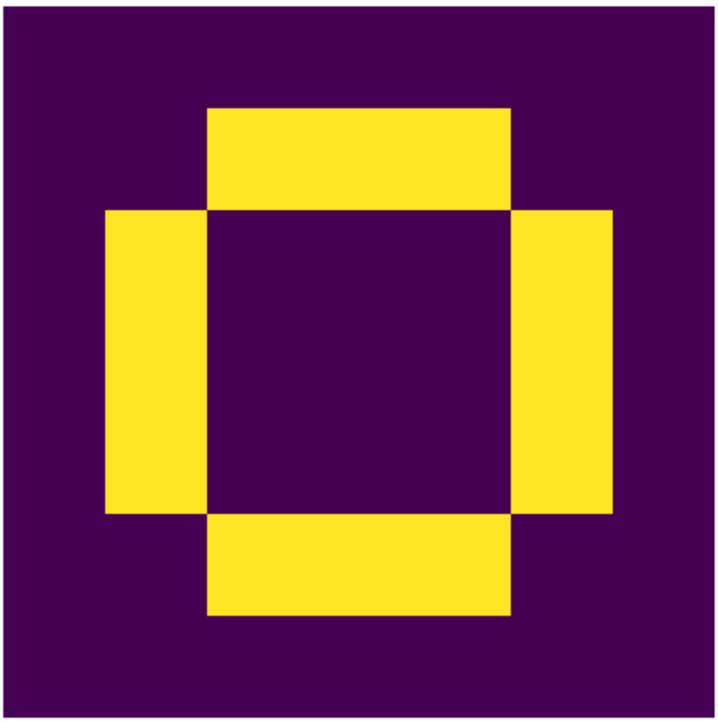
1



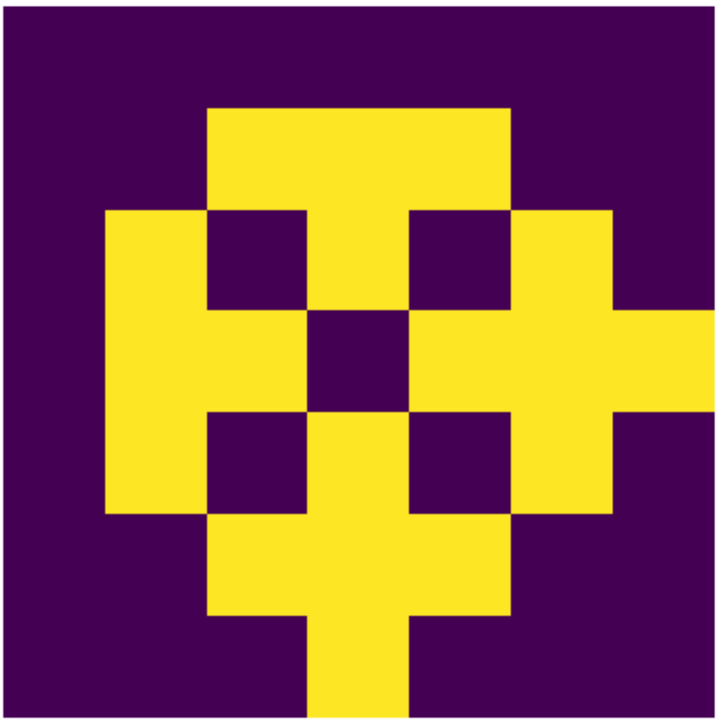
2



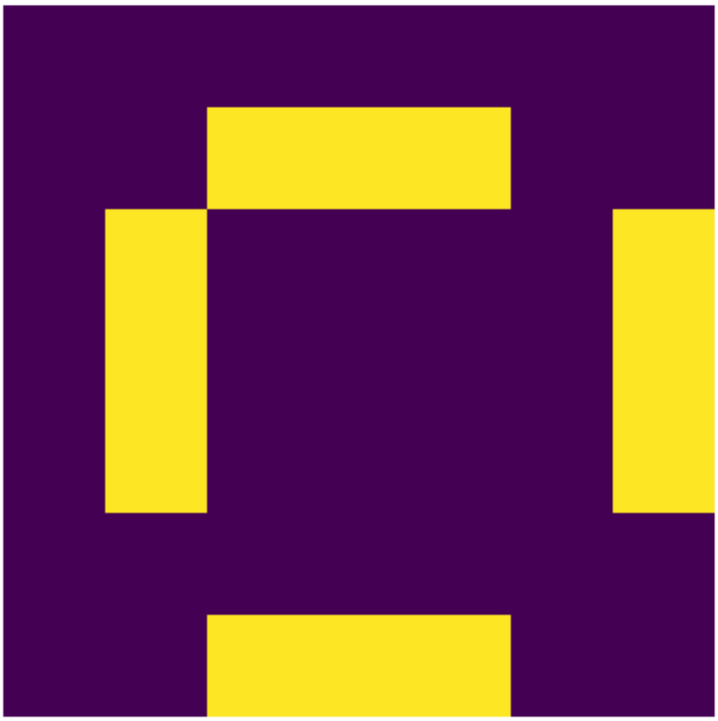
3



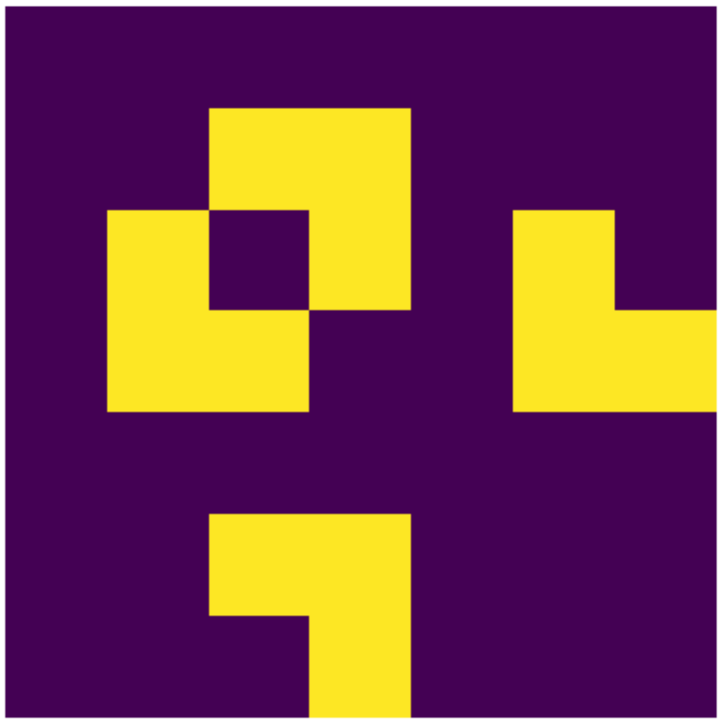
4



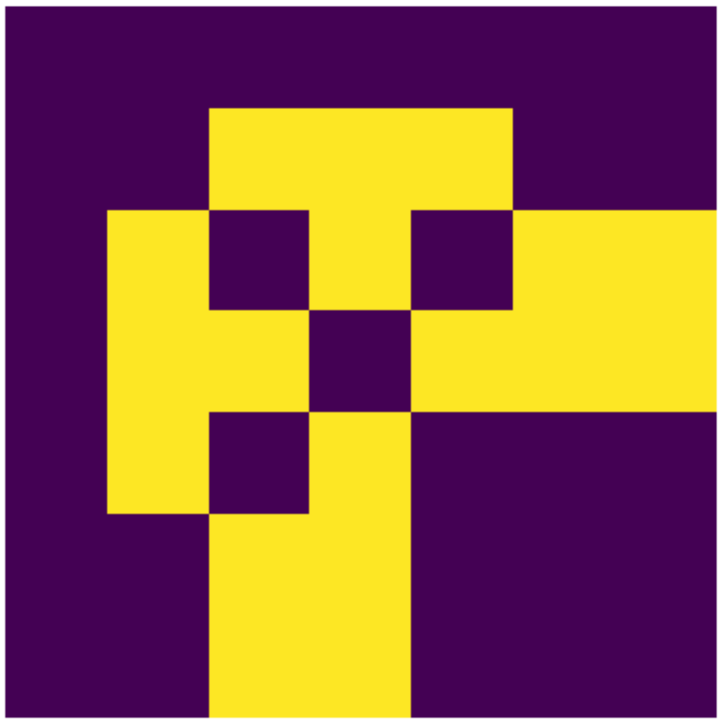
5



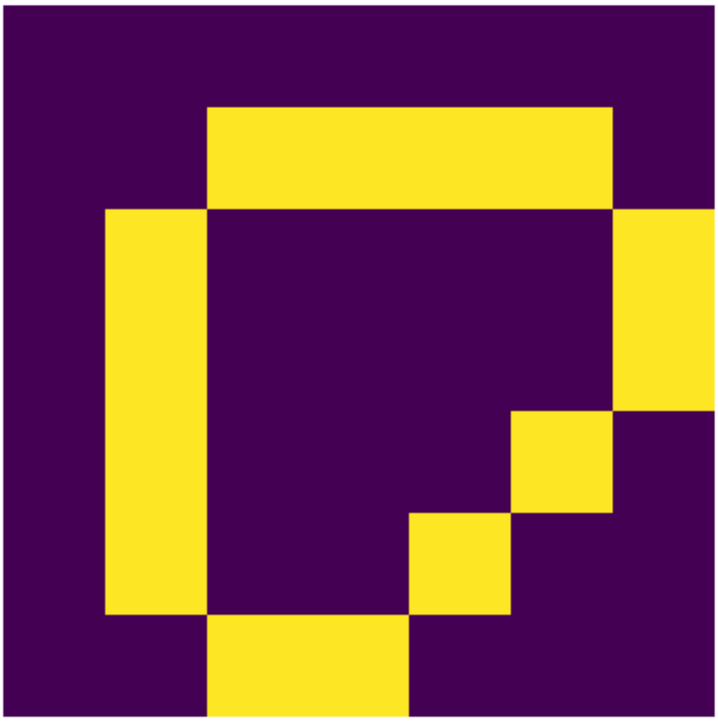
6



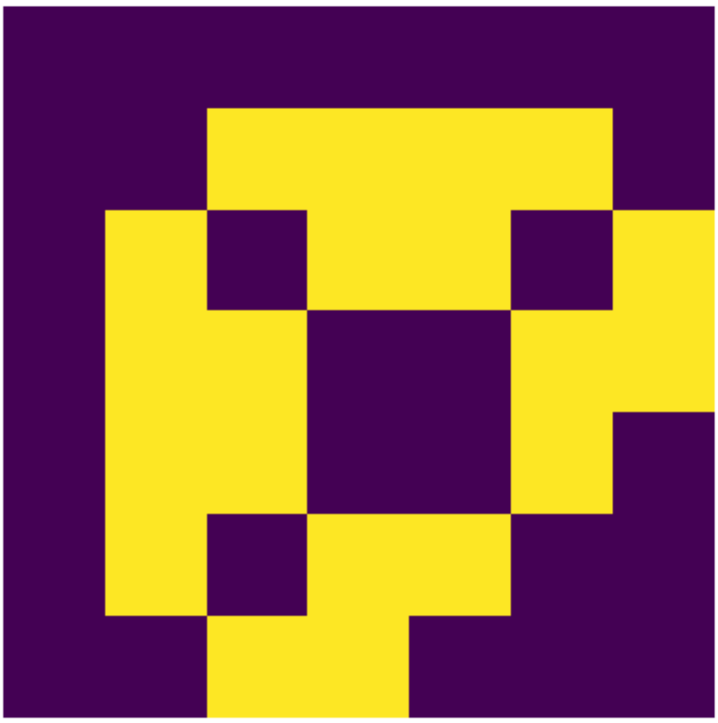
7



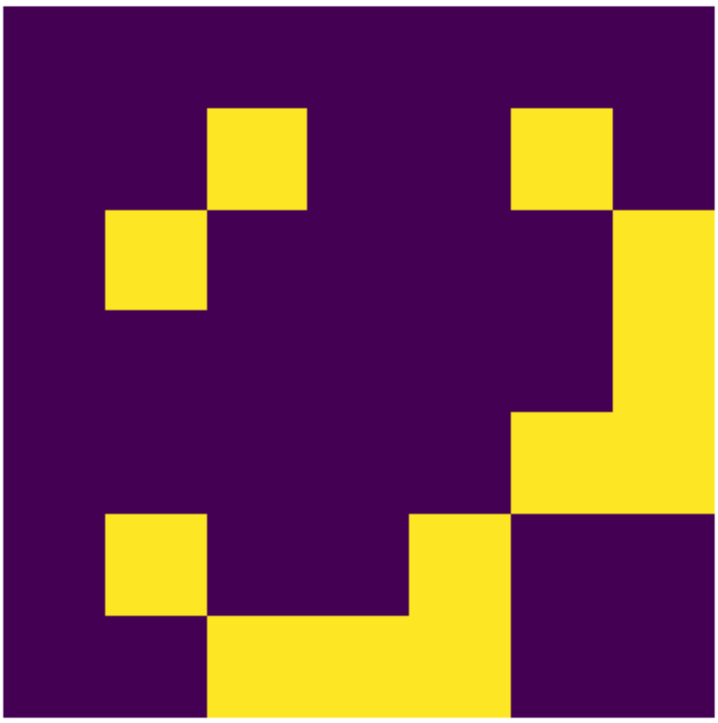
8



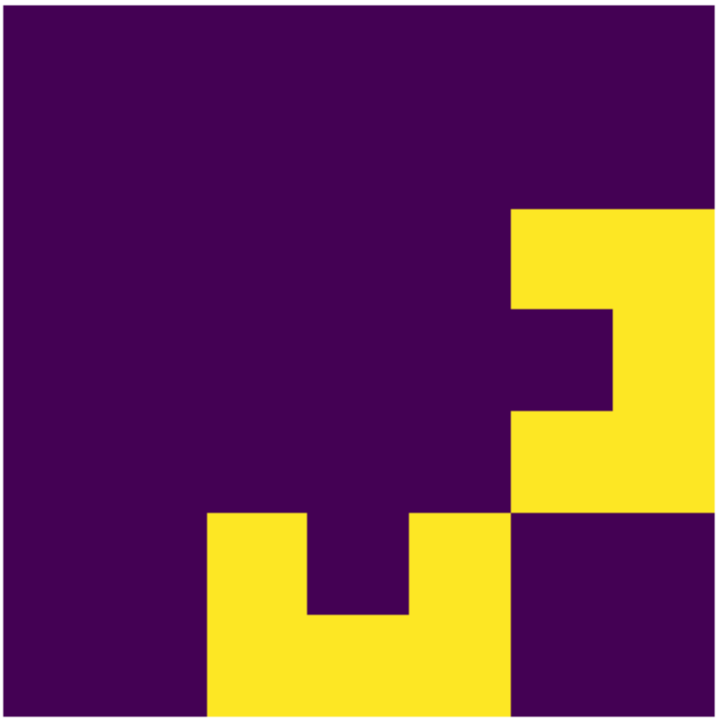
9



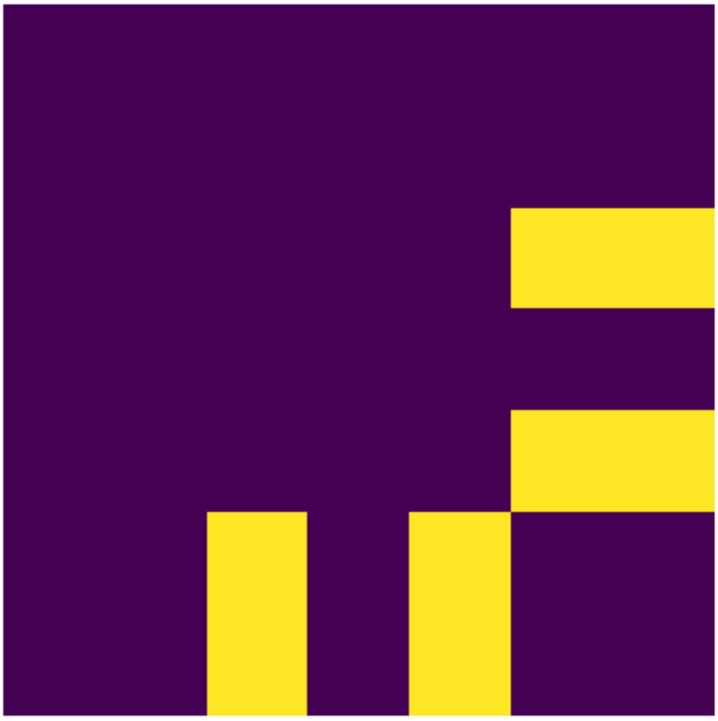
10



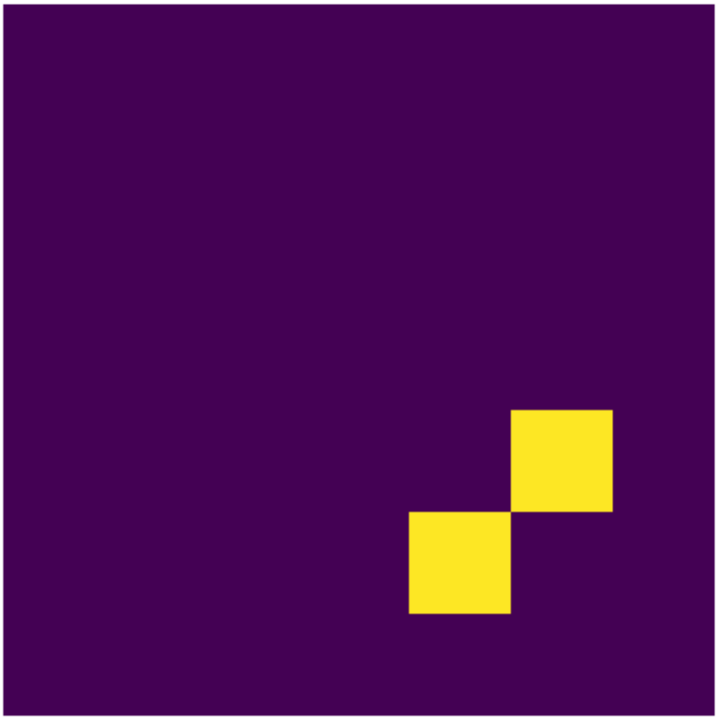
11



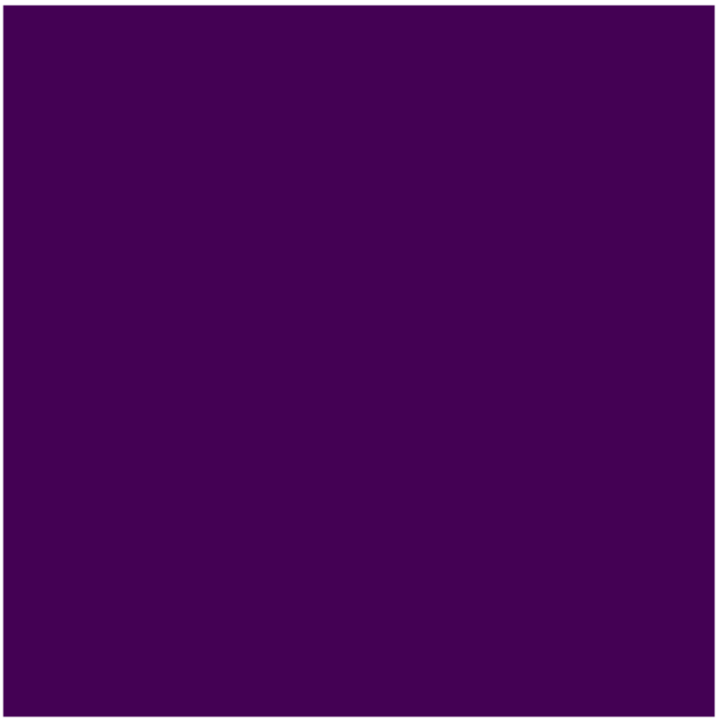
12



13



14



15

[26]:
```
print("Propriedade A:")
bmc_always(declare,init,trans,propA,bound,N,centro,border)
print("Propriedade A, mas para células normais apenas:")
bmc_always(declare,init,trans,propA2,bound,N,centro,border)
print("Propriedade B:")
bmc_eventually(declare,init,trans,propB,bound,N,centro,border)
```

```
Propriedade A:
A propriedade é inválida
Propriedade A, mas para células normais apenas:
A propriedade é inválida
Propriedade B:
Propriedade é inválida
```

Exemplo 5 Este é outro exemplo de uma execução com um quadro relativamente grande.

[14]:
```
N = 20
p = 0.40
k = 100
```

```
centro = gera_centro(N)
border = gera_border(p,N)
print(centro)
print(border)
```

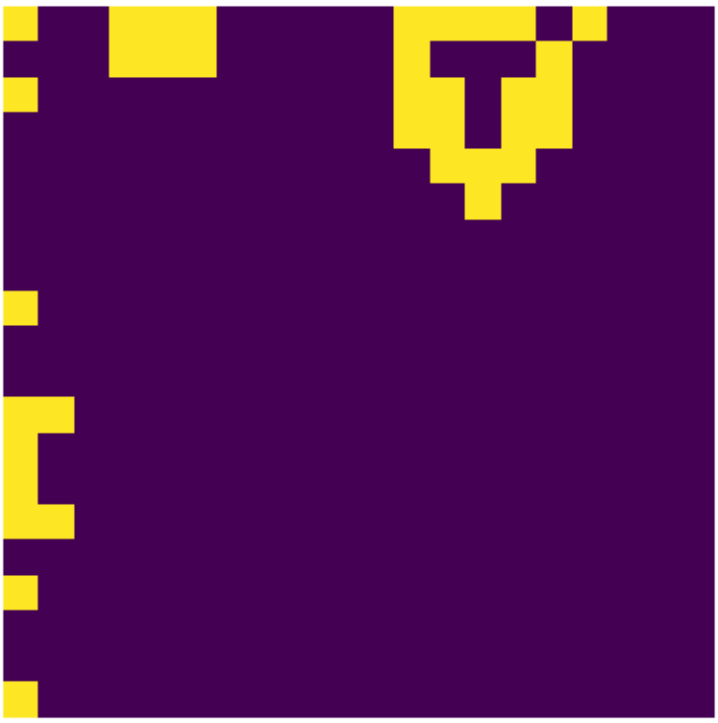
```
(14, 4)
([1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0], [0, 1, 0, 0, 0,
0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1])
```

[15]:
```
print(bound := resolve(N,k,centro,border))
```

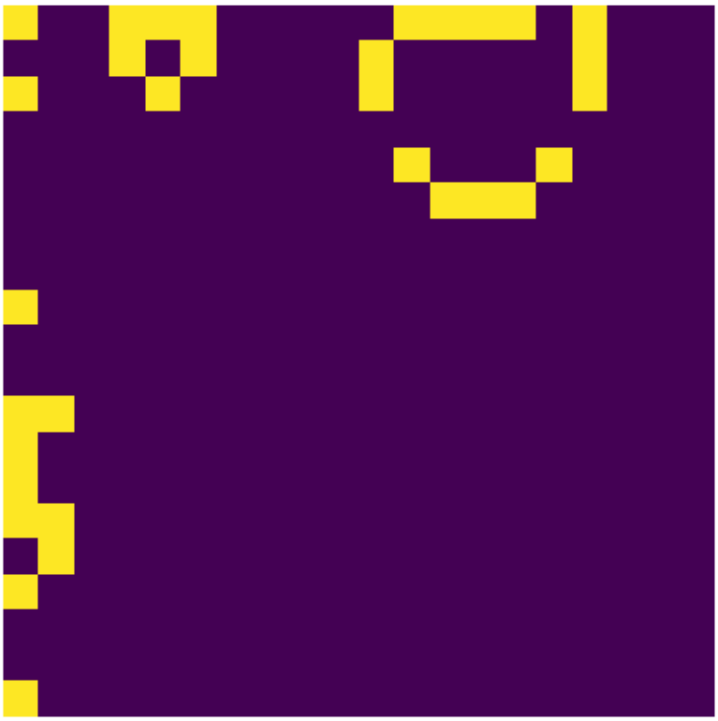
```
0
```

```
1
```

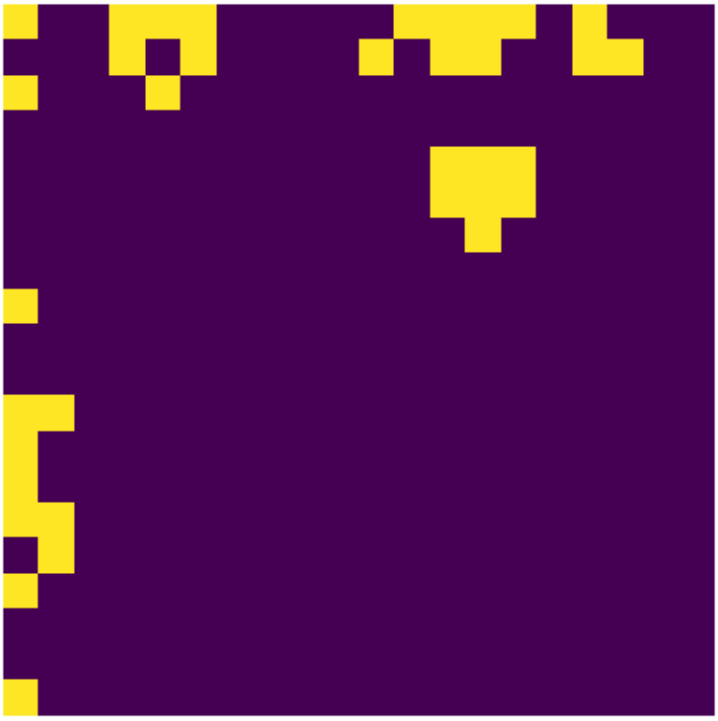
2



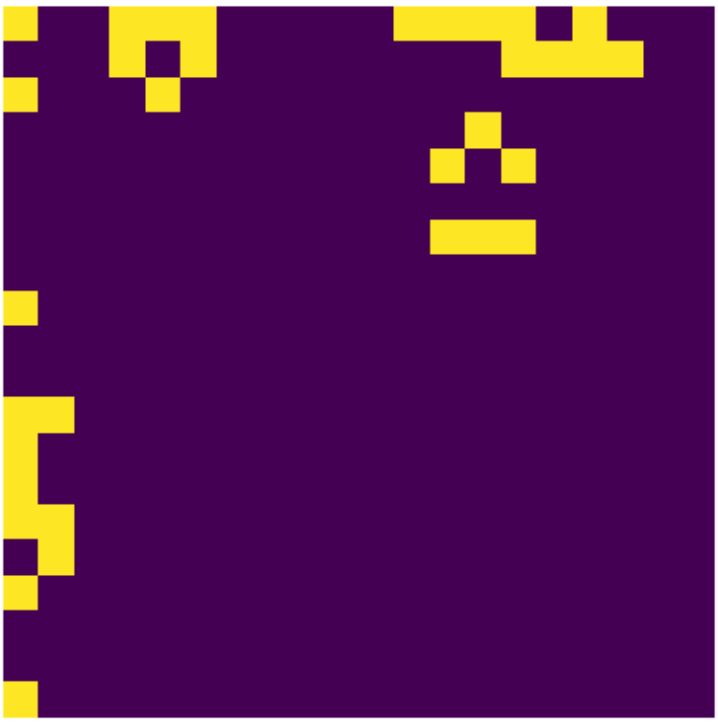
3



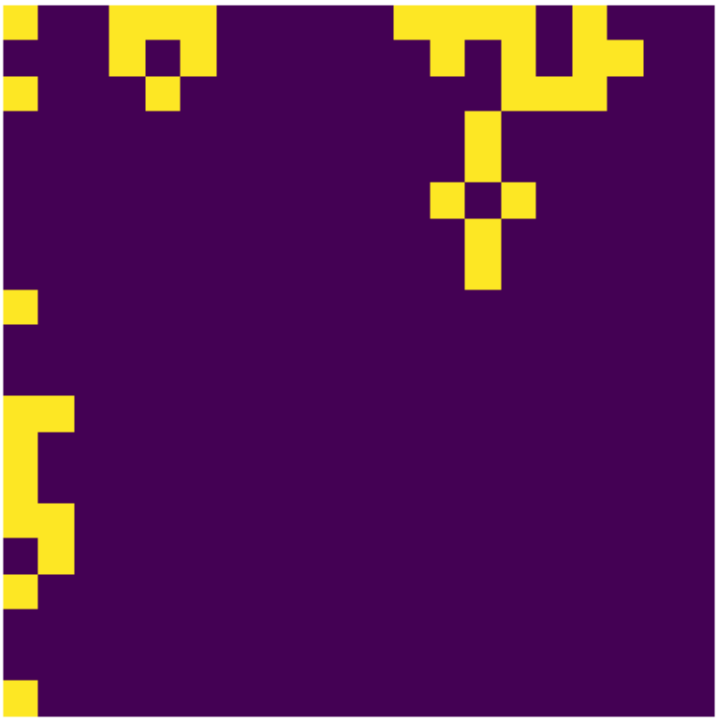
4



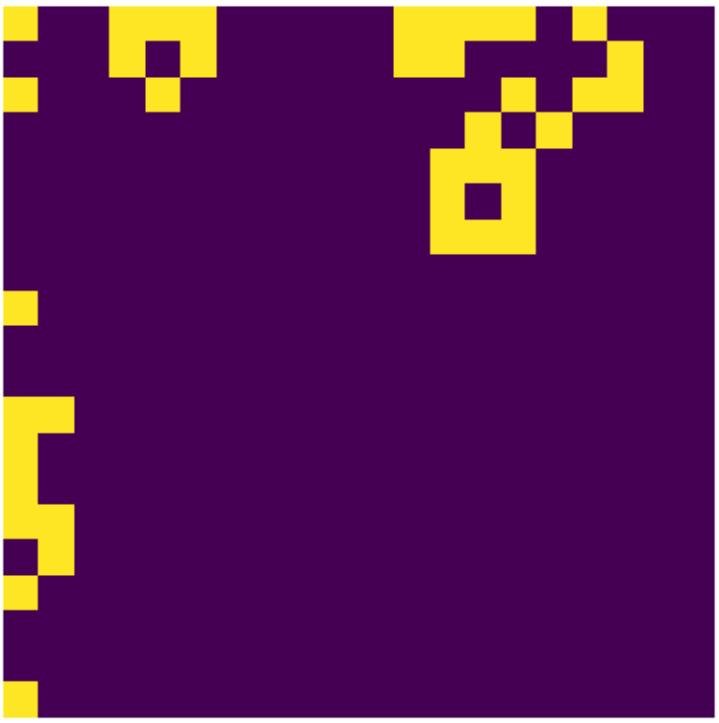
5



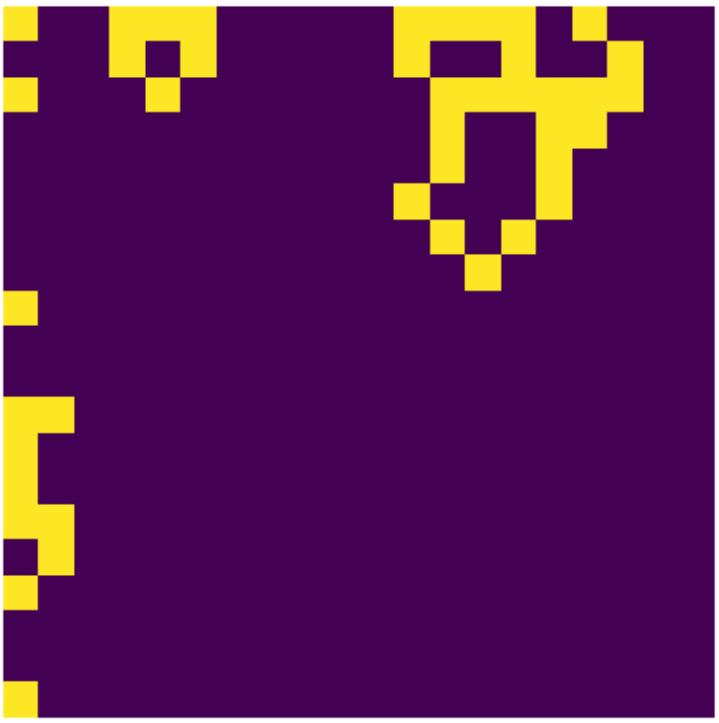
6



7

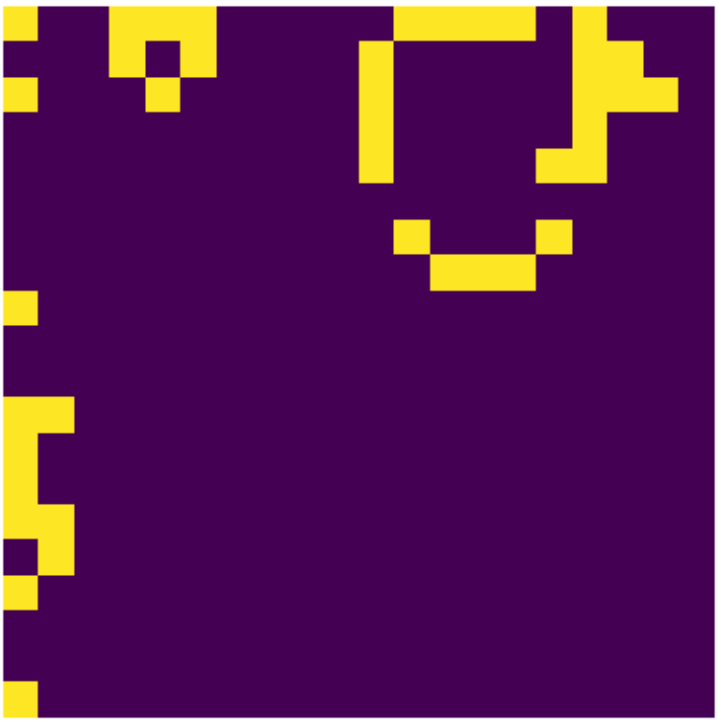


8

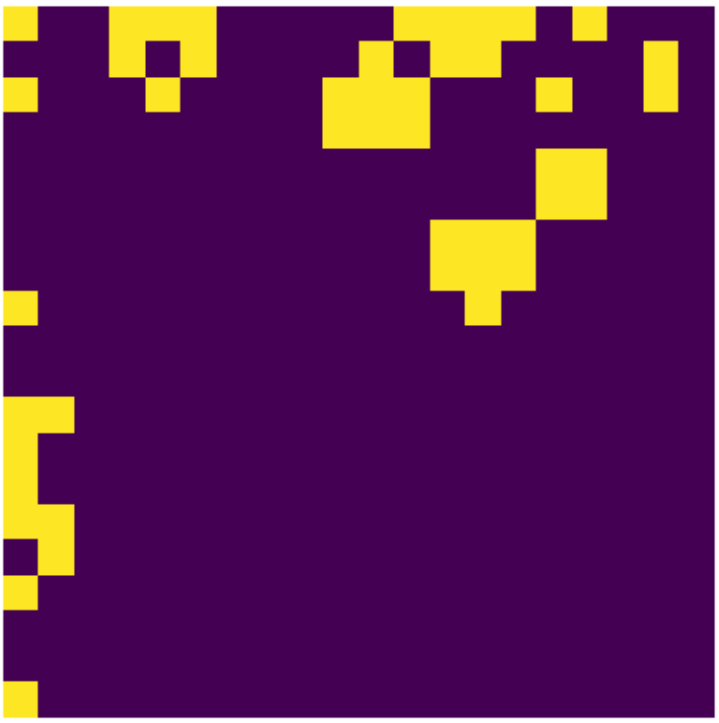


9

10

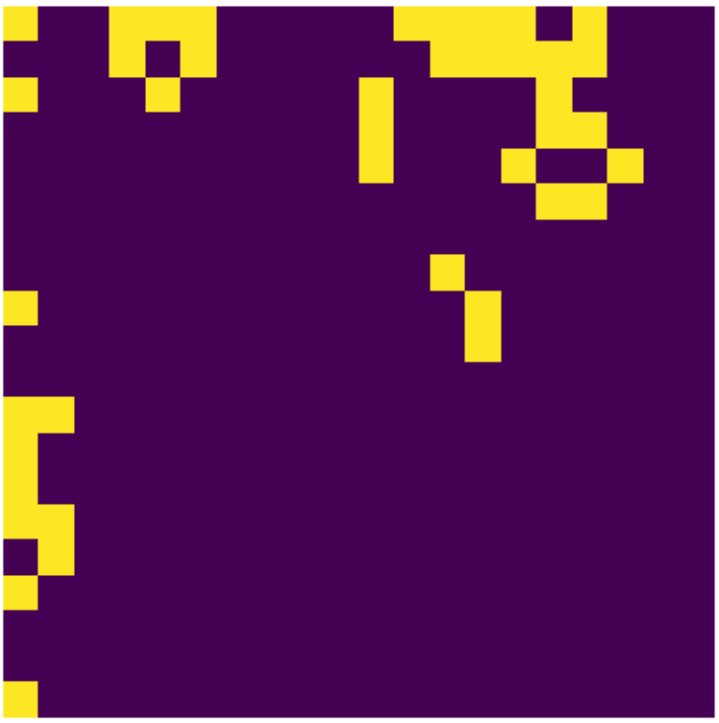


11

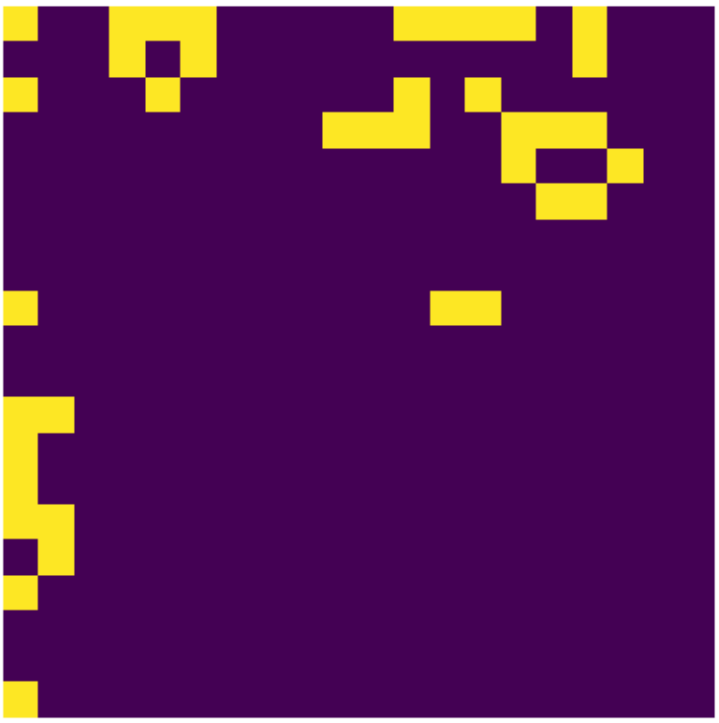


12

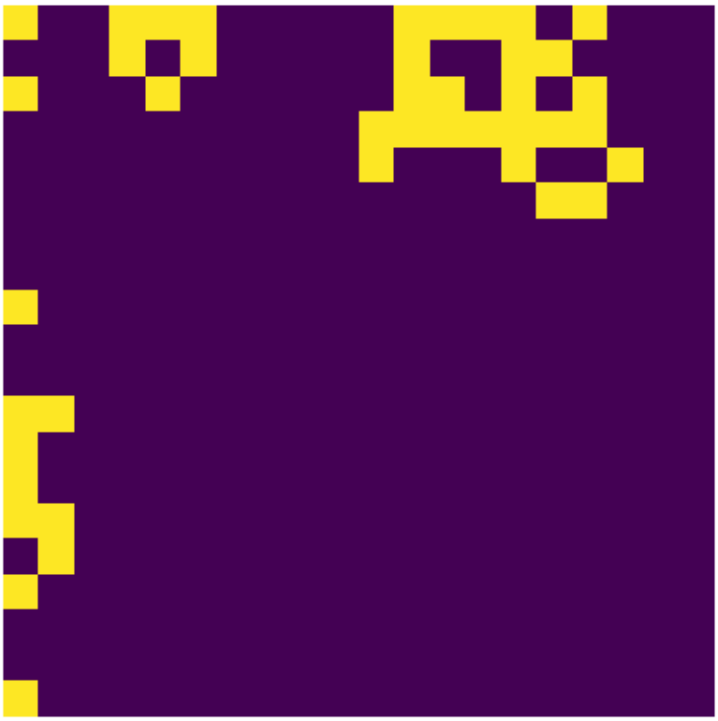
13



14

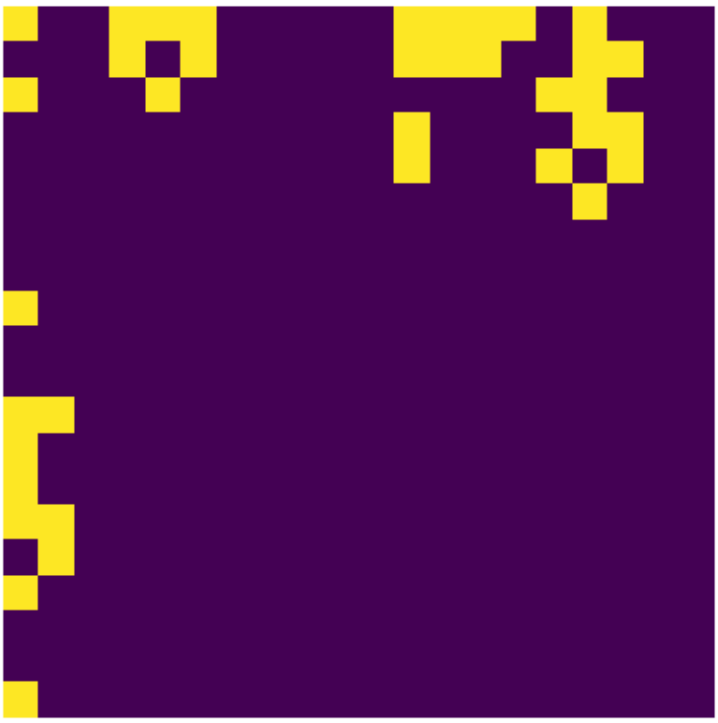


15



16

17

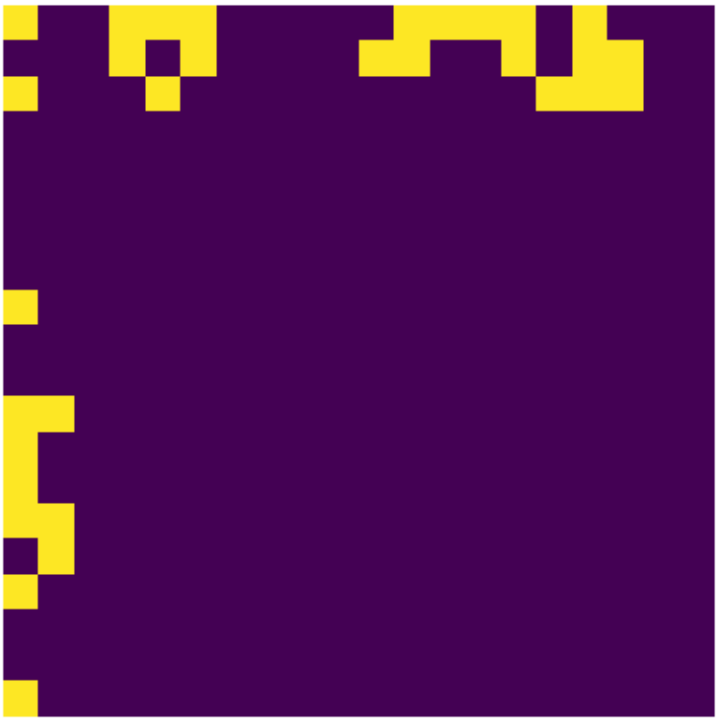


18

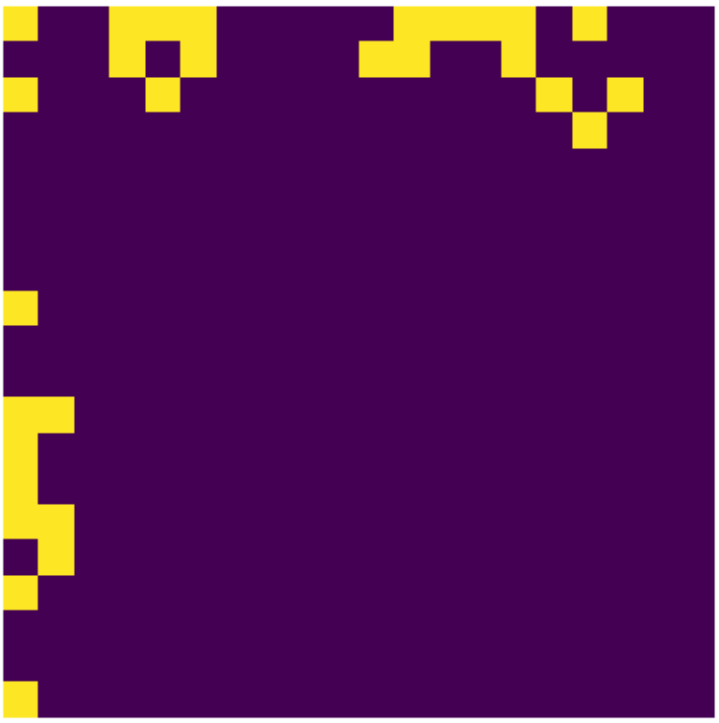
19

20

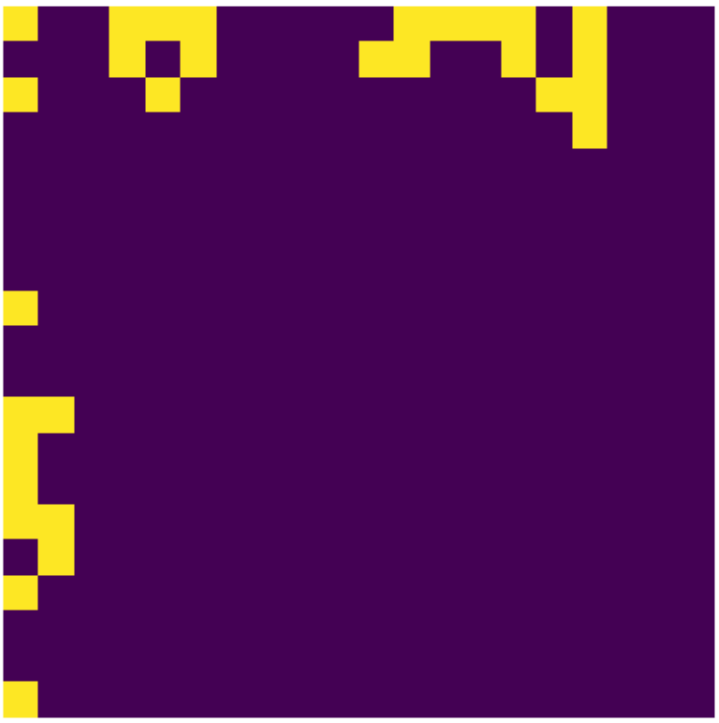
21



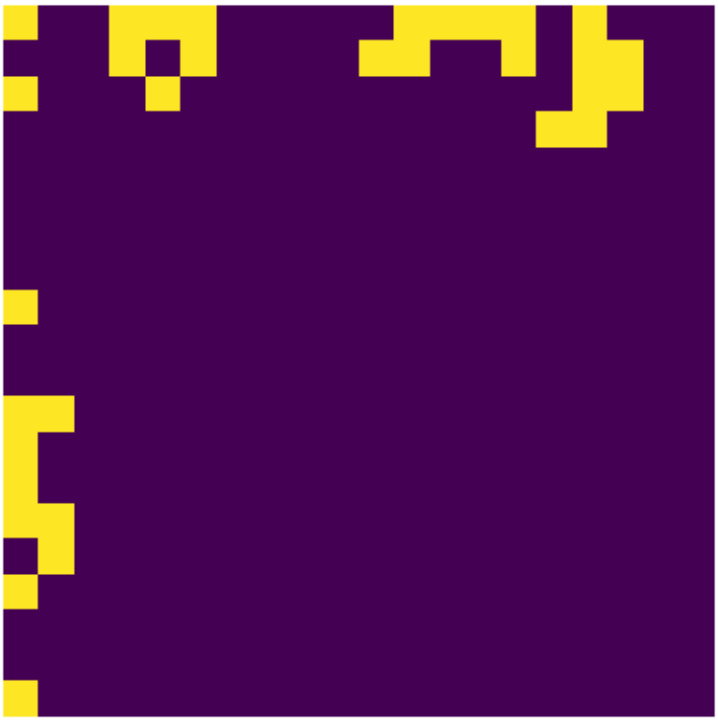
22



23

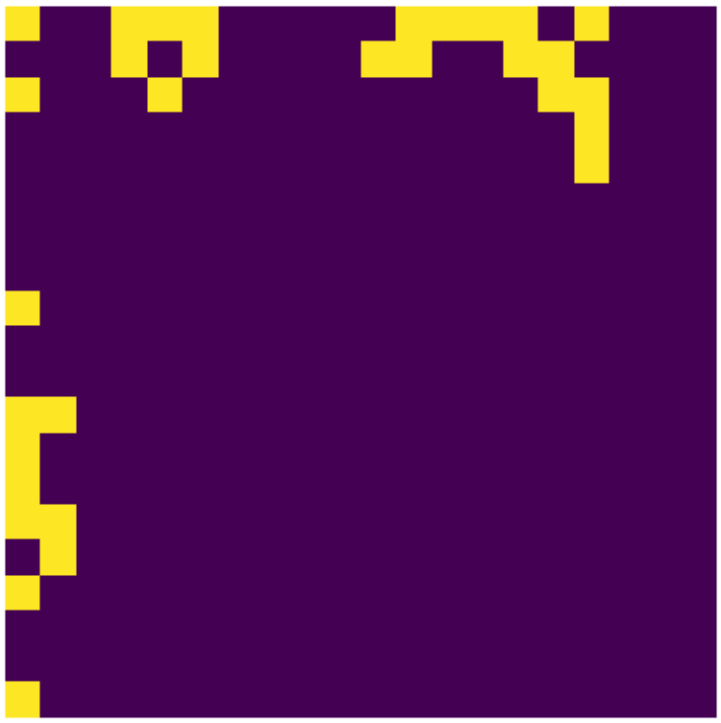


24

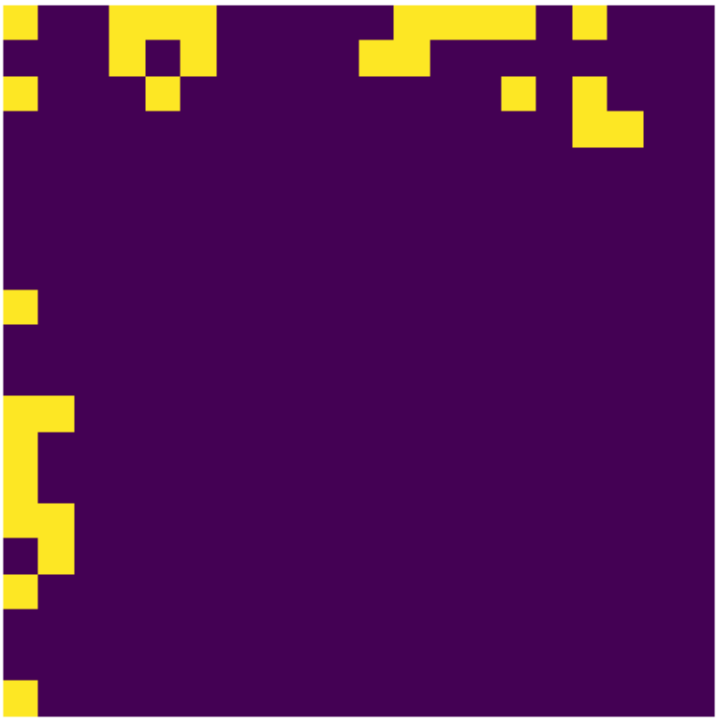


25

26

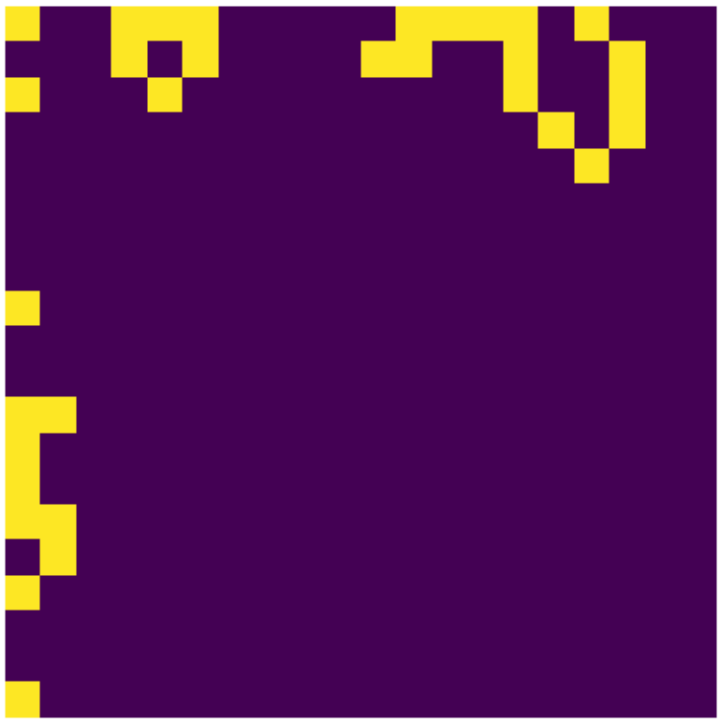


27

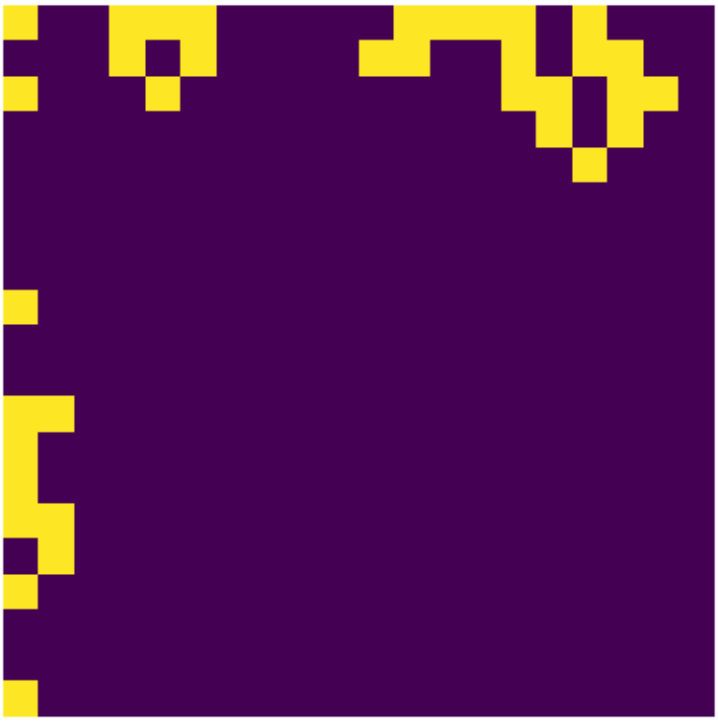


28

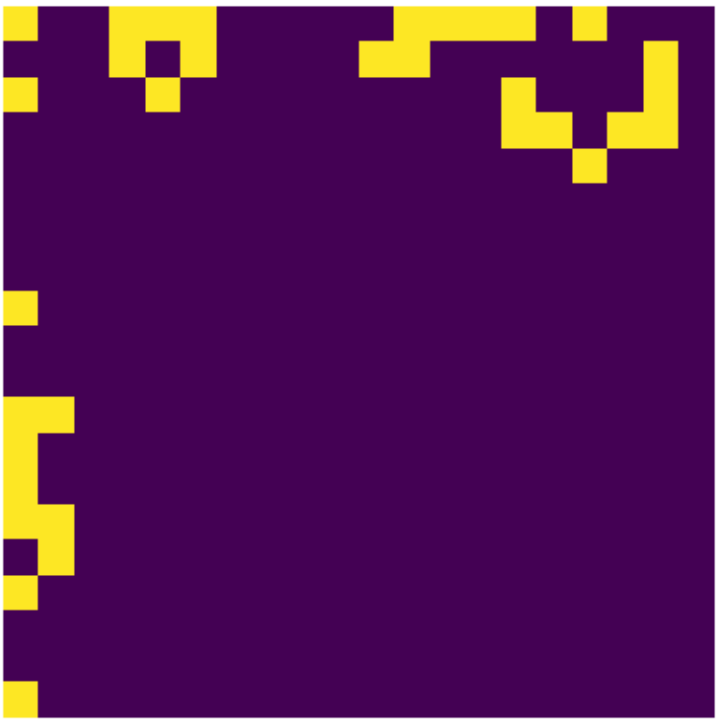
29



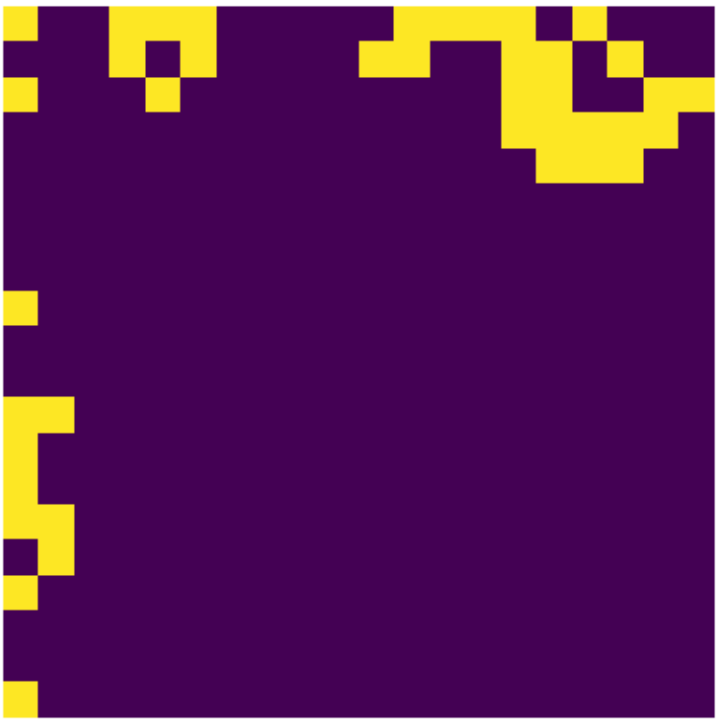
30



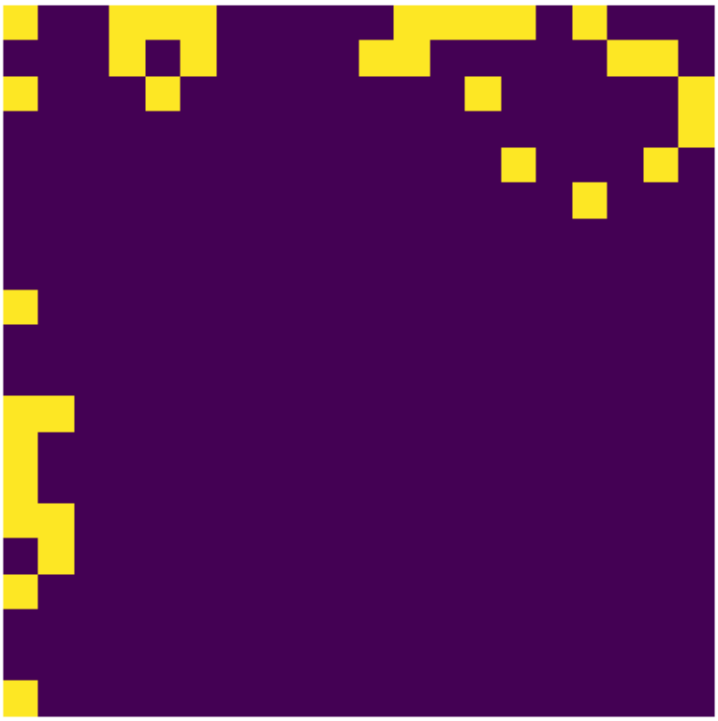
31



32

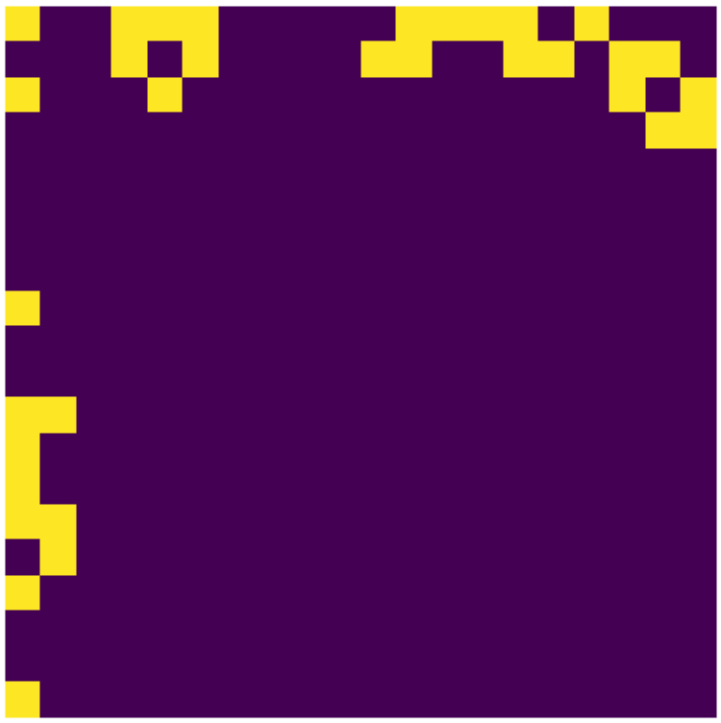


33



34

35

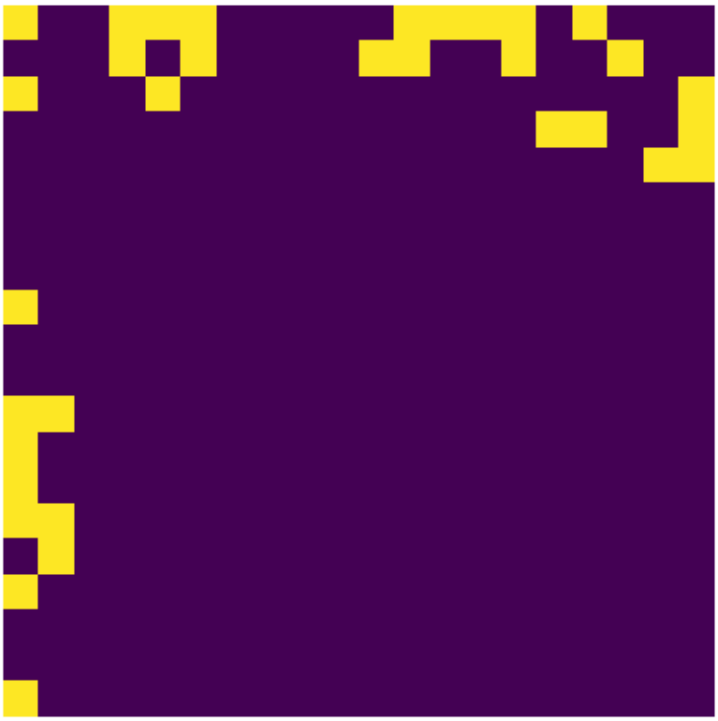


36

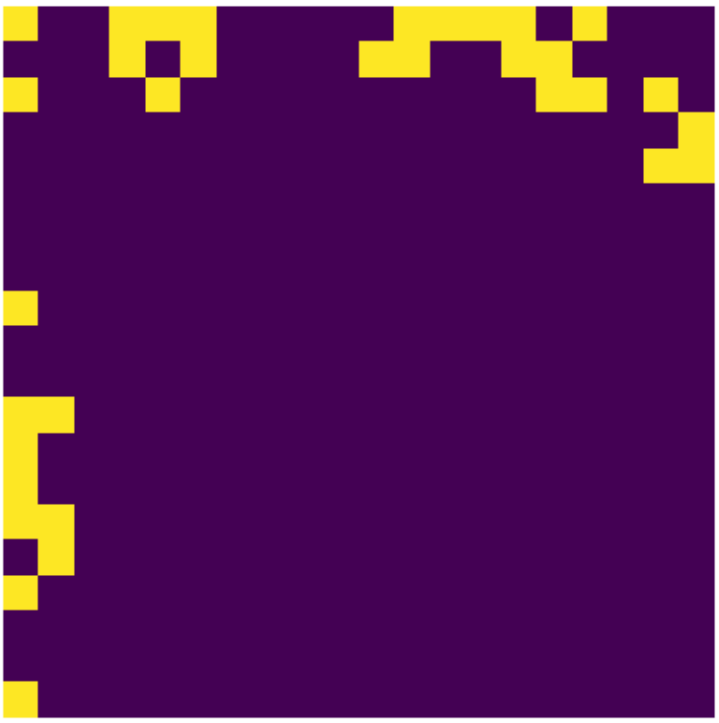
37

38

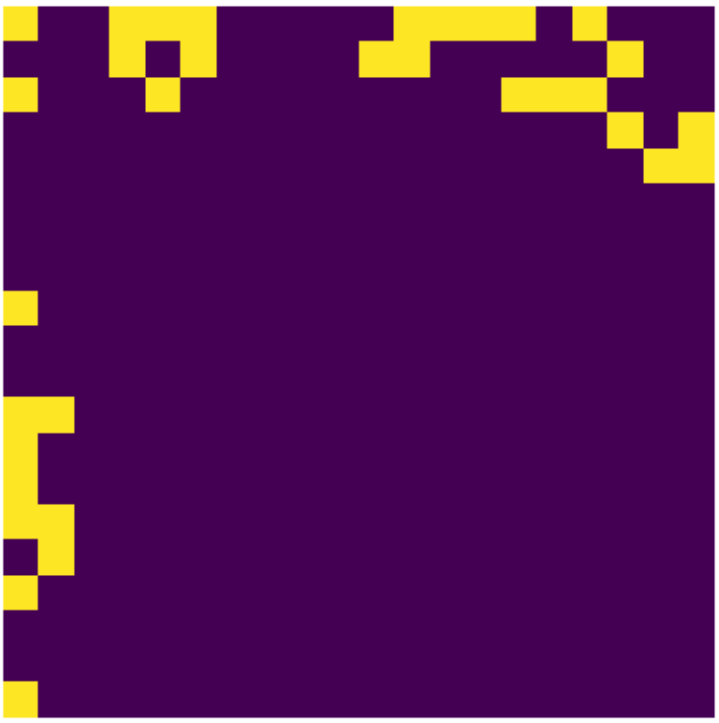
39



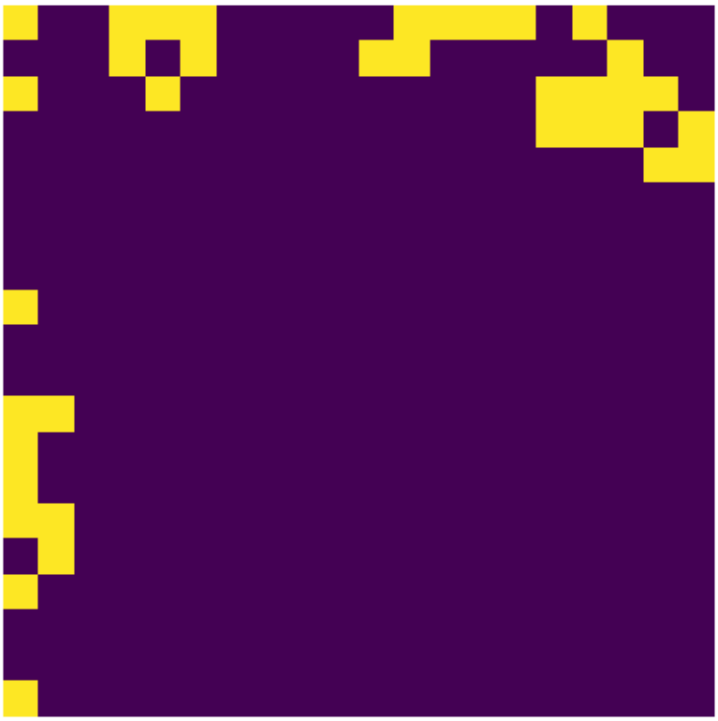
40



41

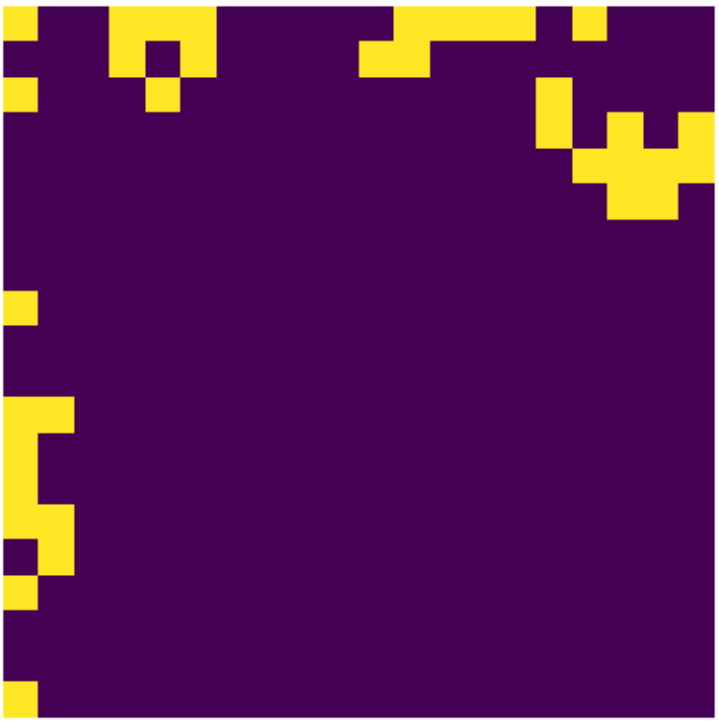


42

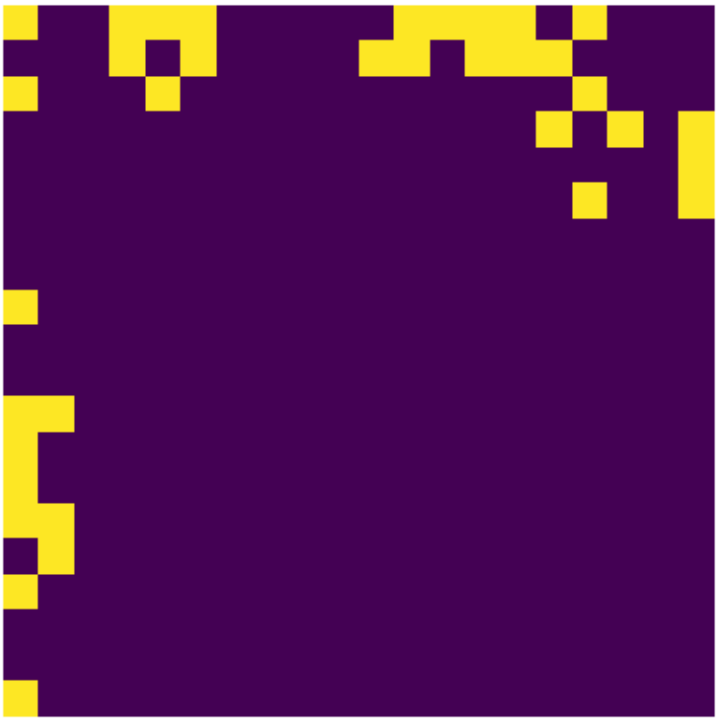


43

44

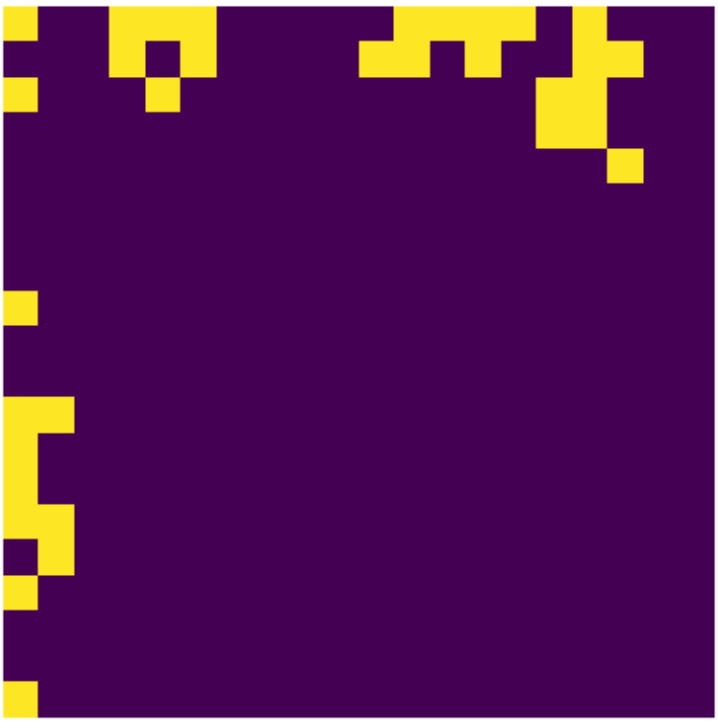


45

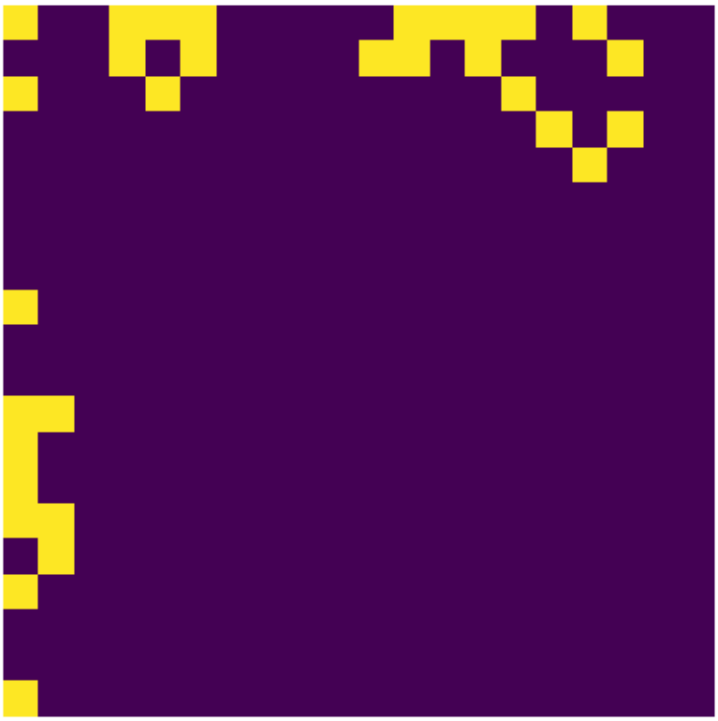


46

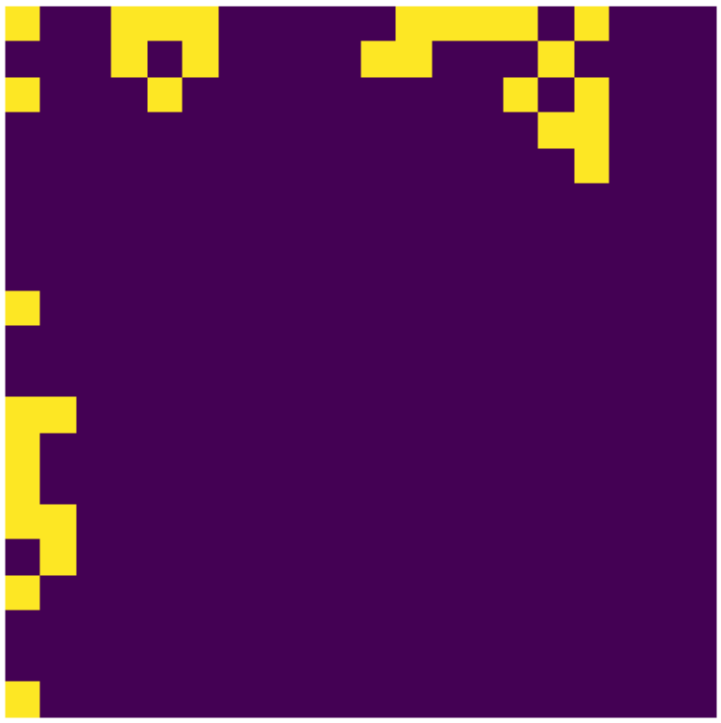
47



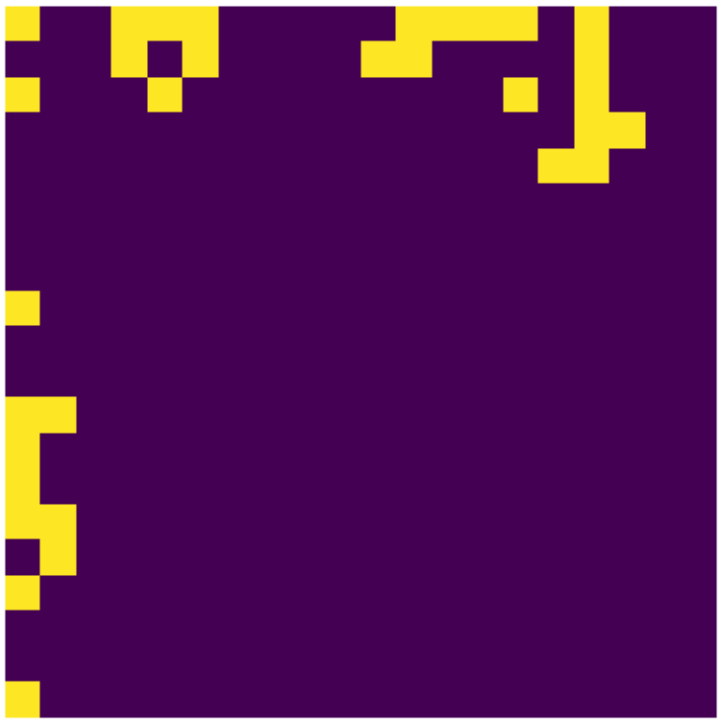
48



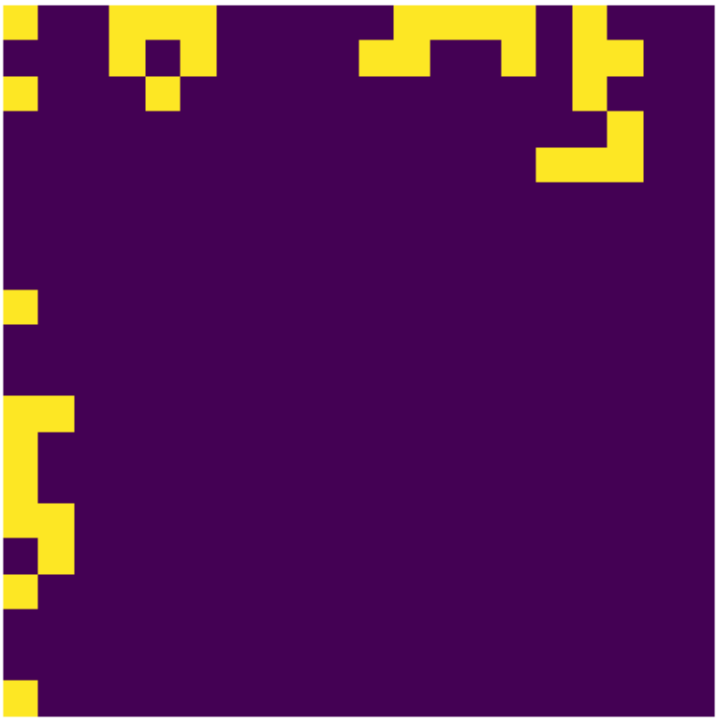
49



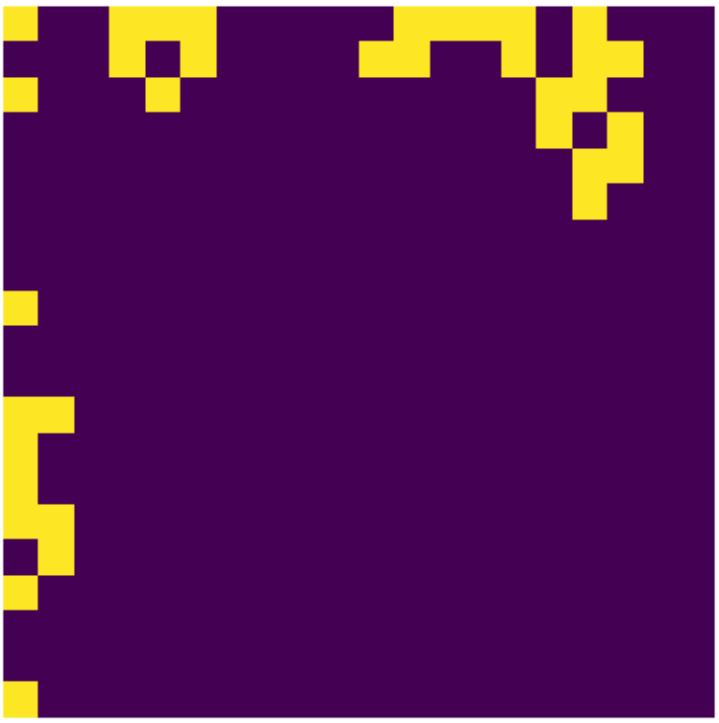
50



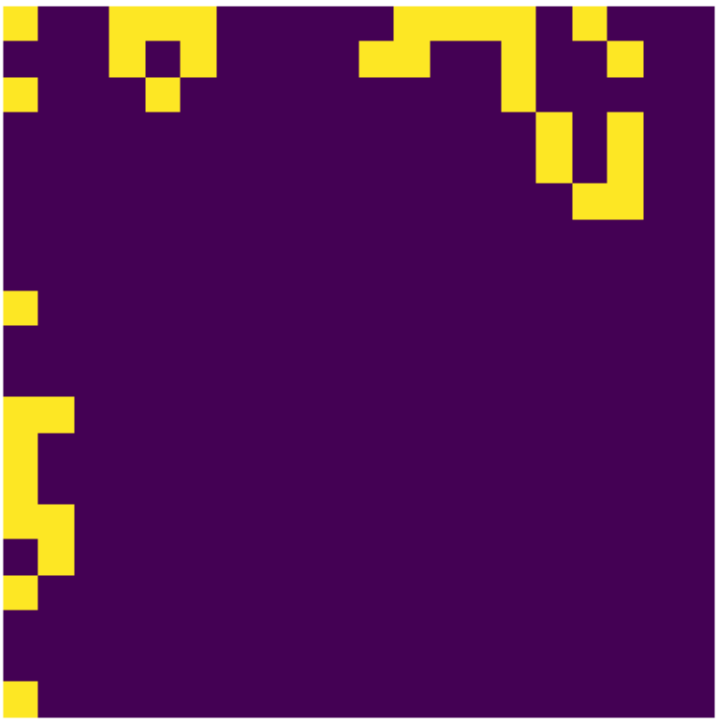
51



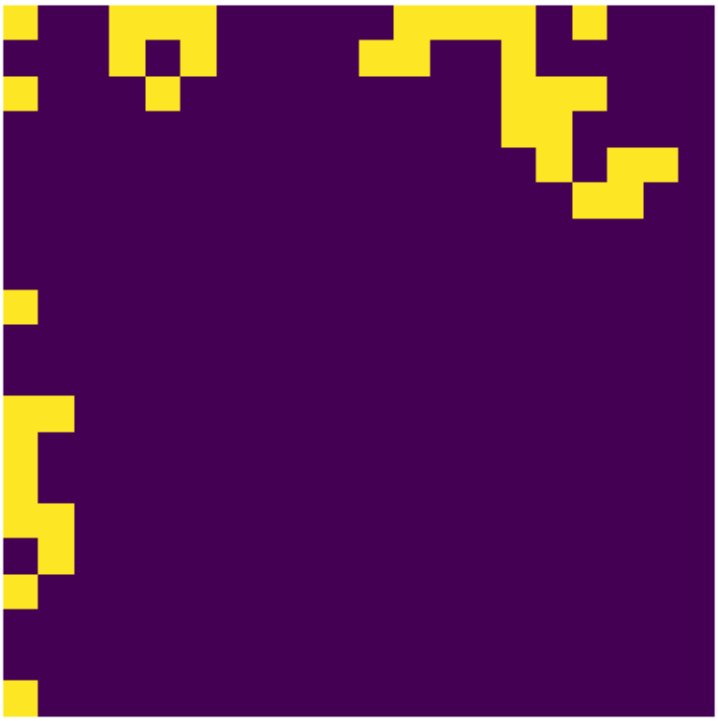
52



53

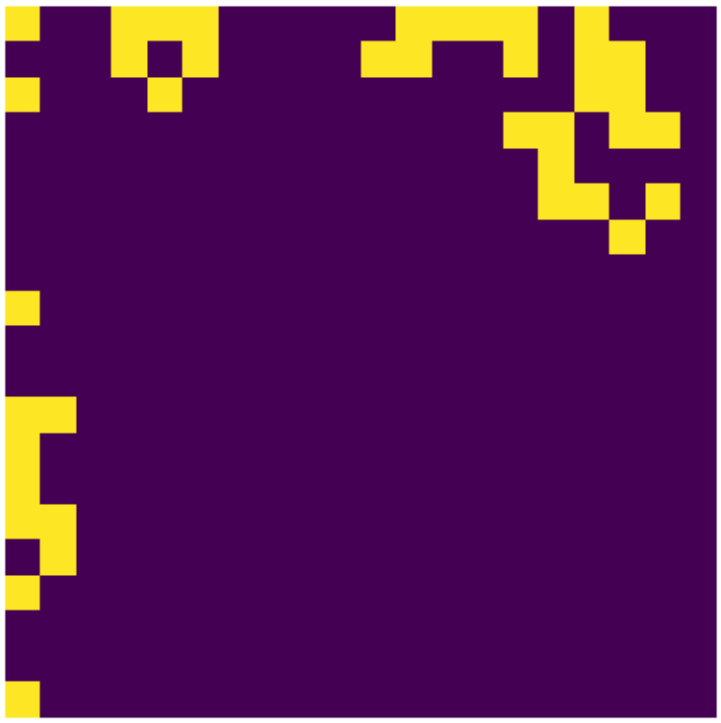


54

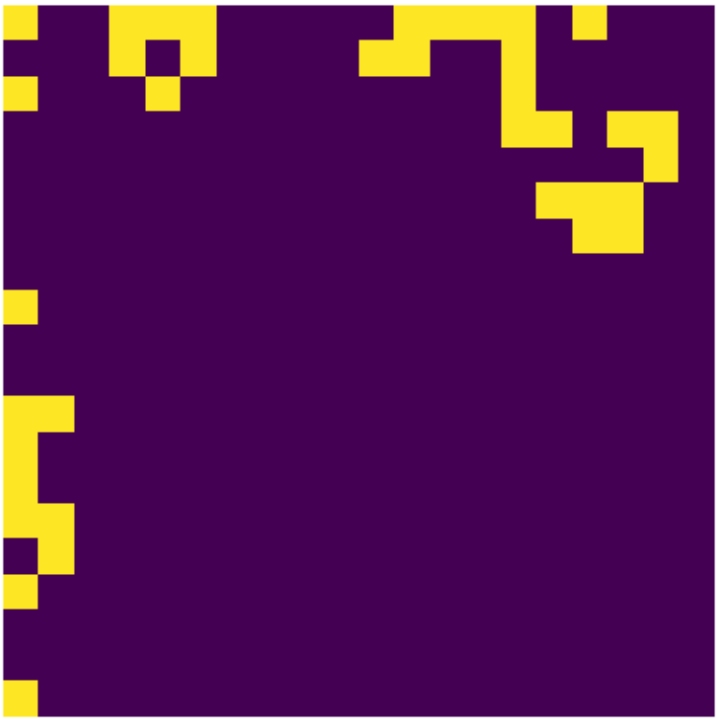


55

56

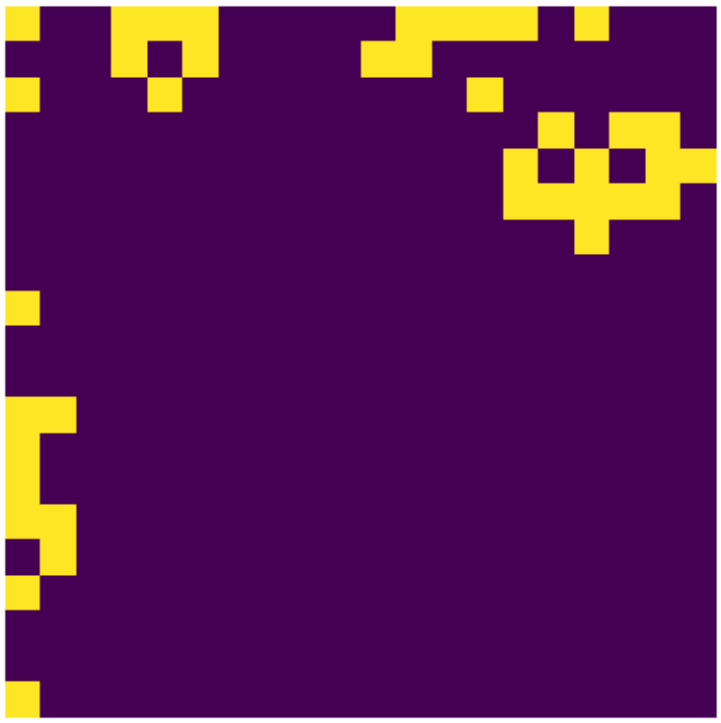


57



58

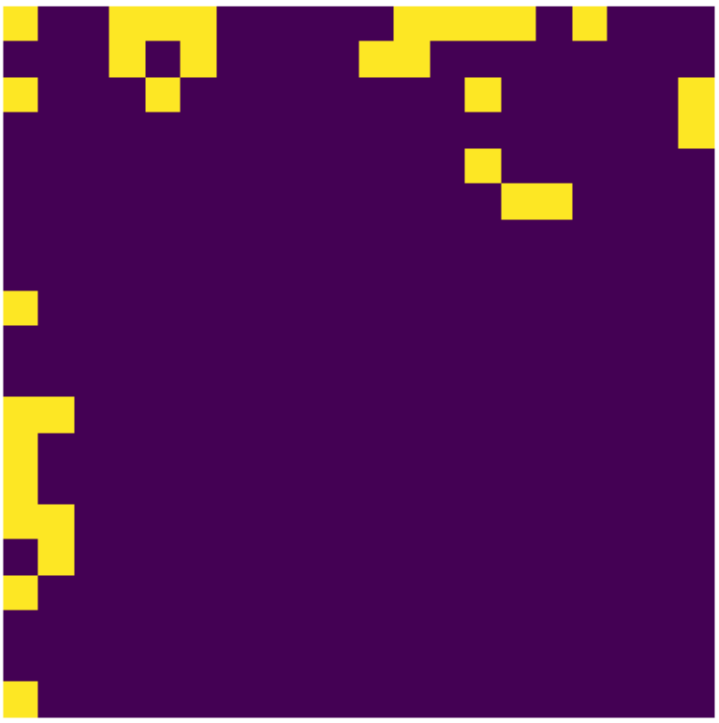
59



60

61

62



63

64

65

[16]:
```
print("Propriedade A:")
bmc_always(declare,init,trans,propA,bound,N,centro,border)
print("Propriedade A, mas para células normais apenas:")
bmc_always(declare,init,trans,propA2,bound,N,centro,border)
print("Propriedade B:")
bmc_eventually(declare,init,trans,propB,bound,N,centro,border)
```

```
Propriedade A:
A propriedade é válida
Propriedade A, mas para células normais apenas:
A propriedade é válida
Propriedade B:
Propriedade é inválida
```