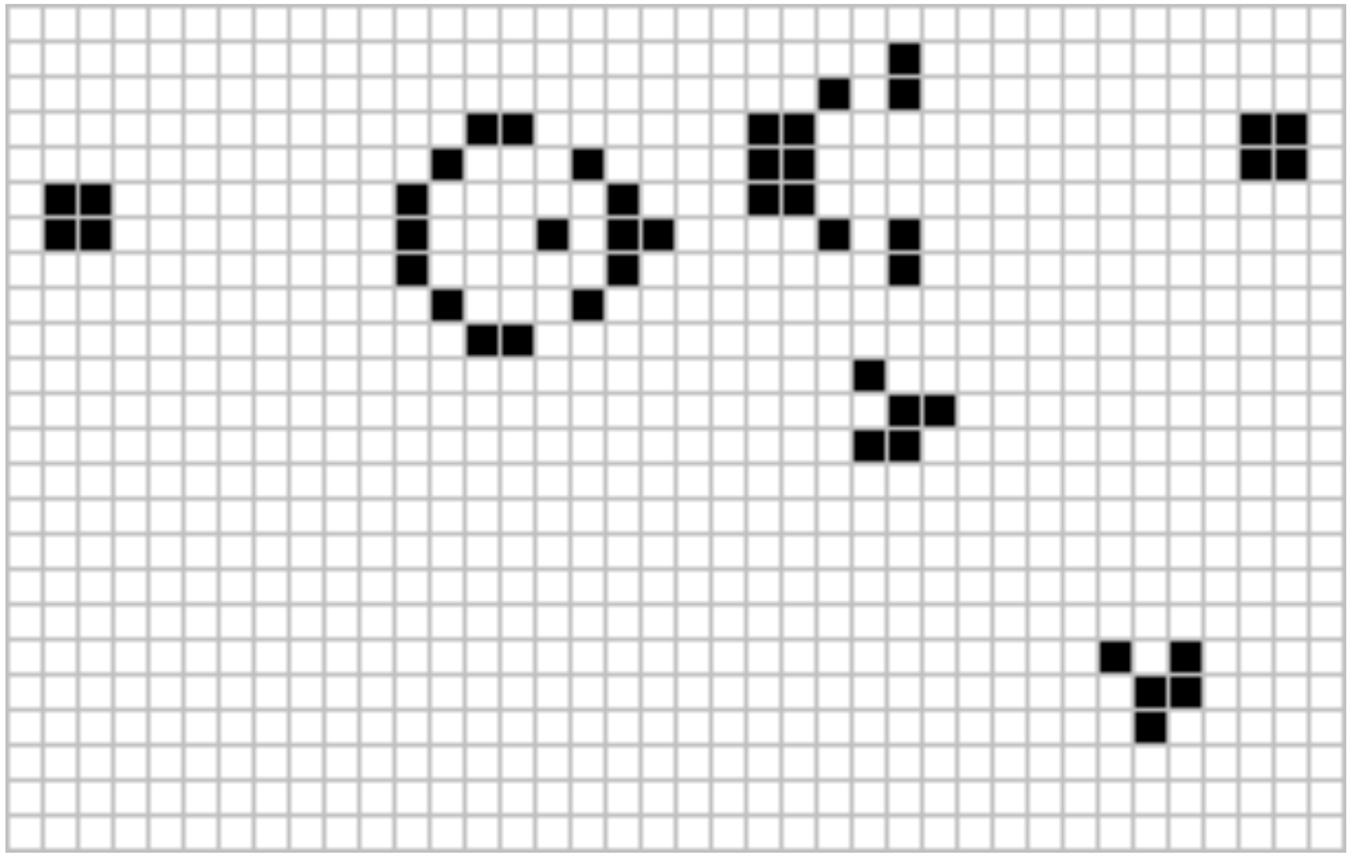
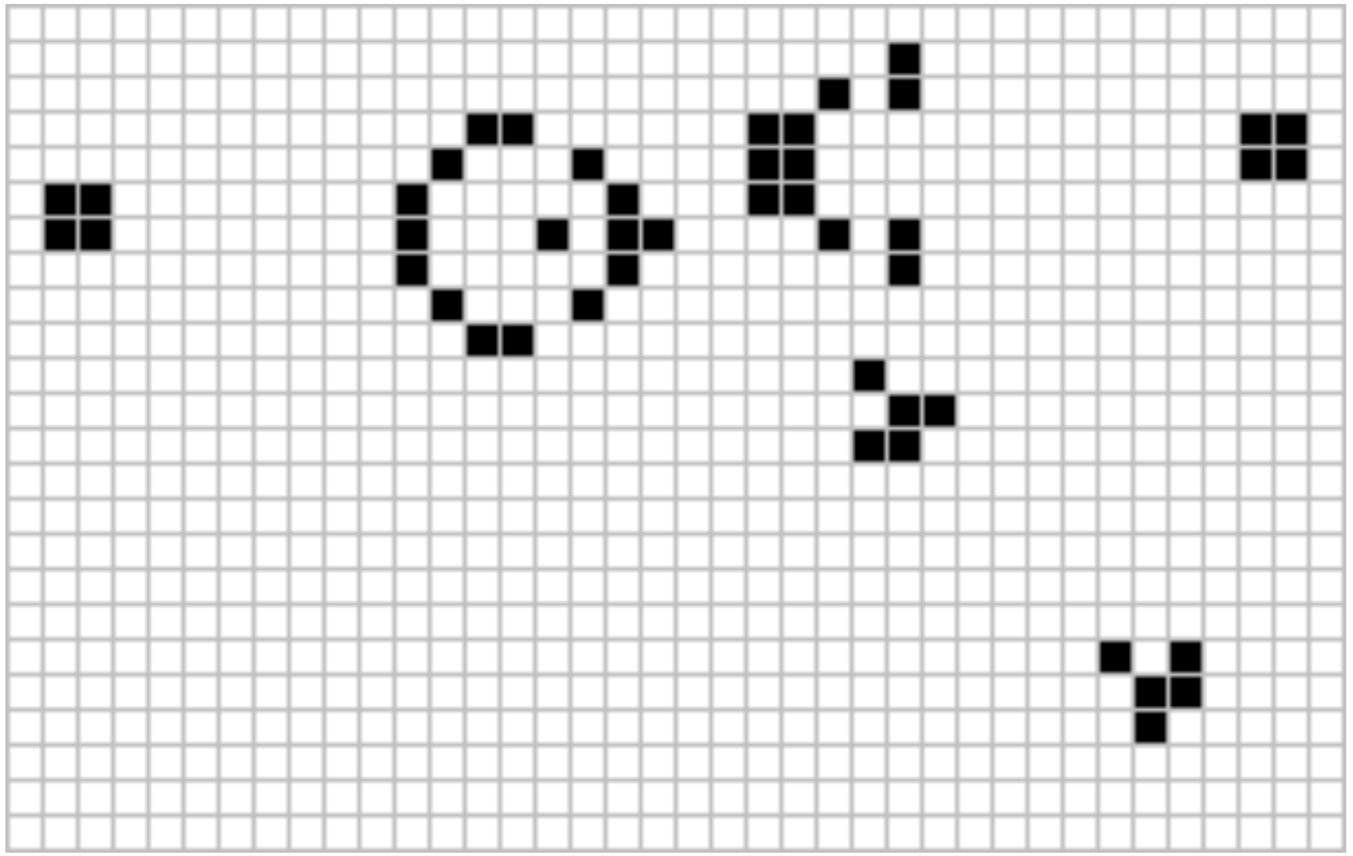
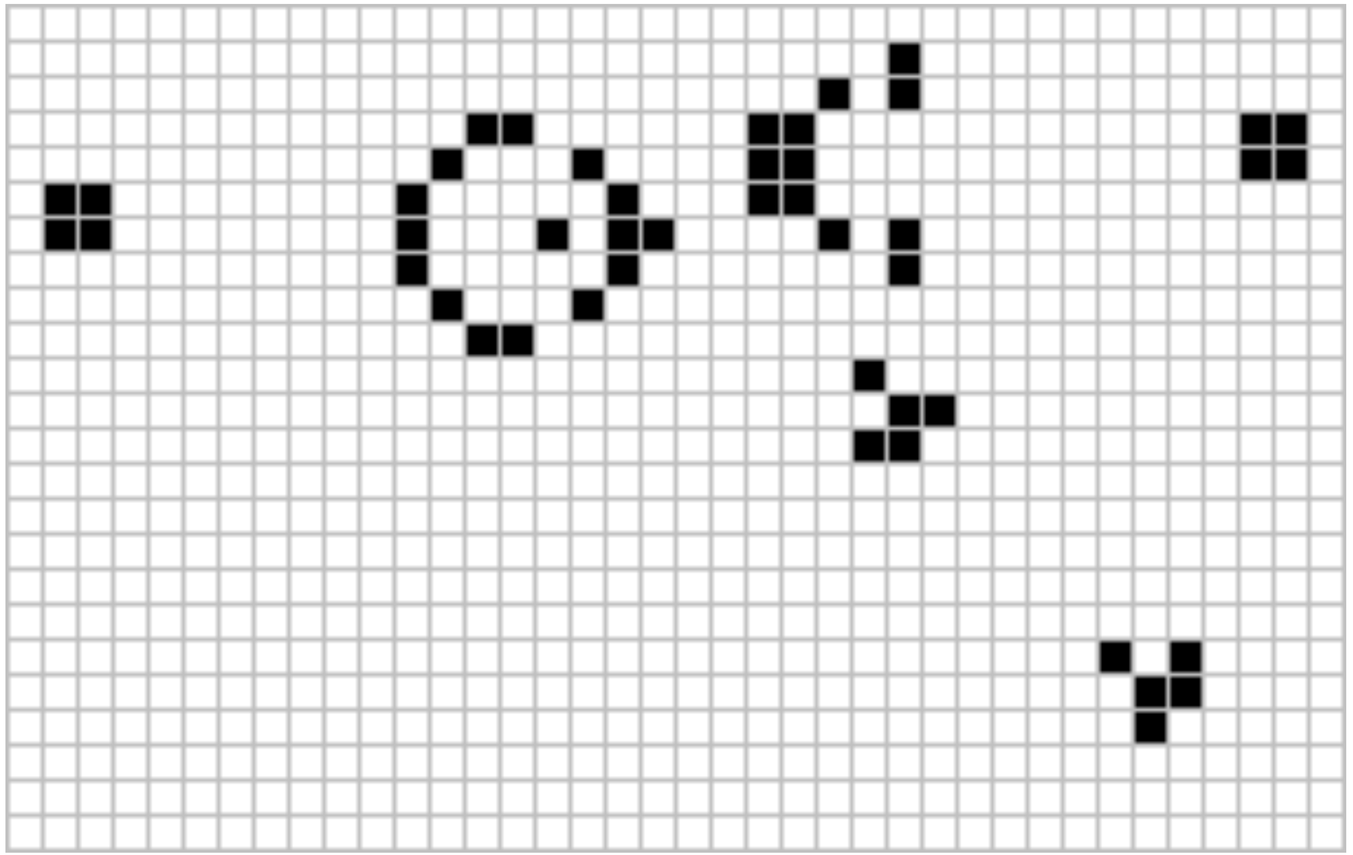
Do Regex dream of Turing Completeness?

Daniel Magliola

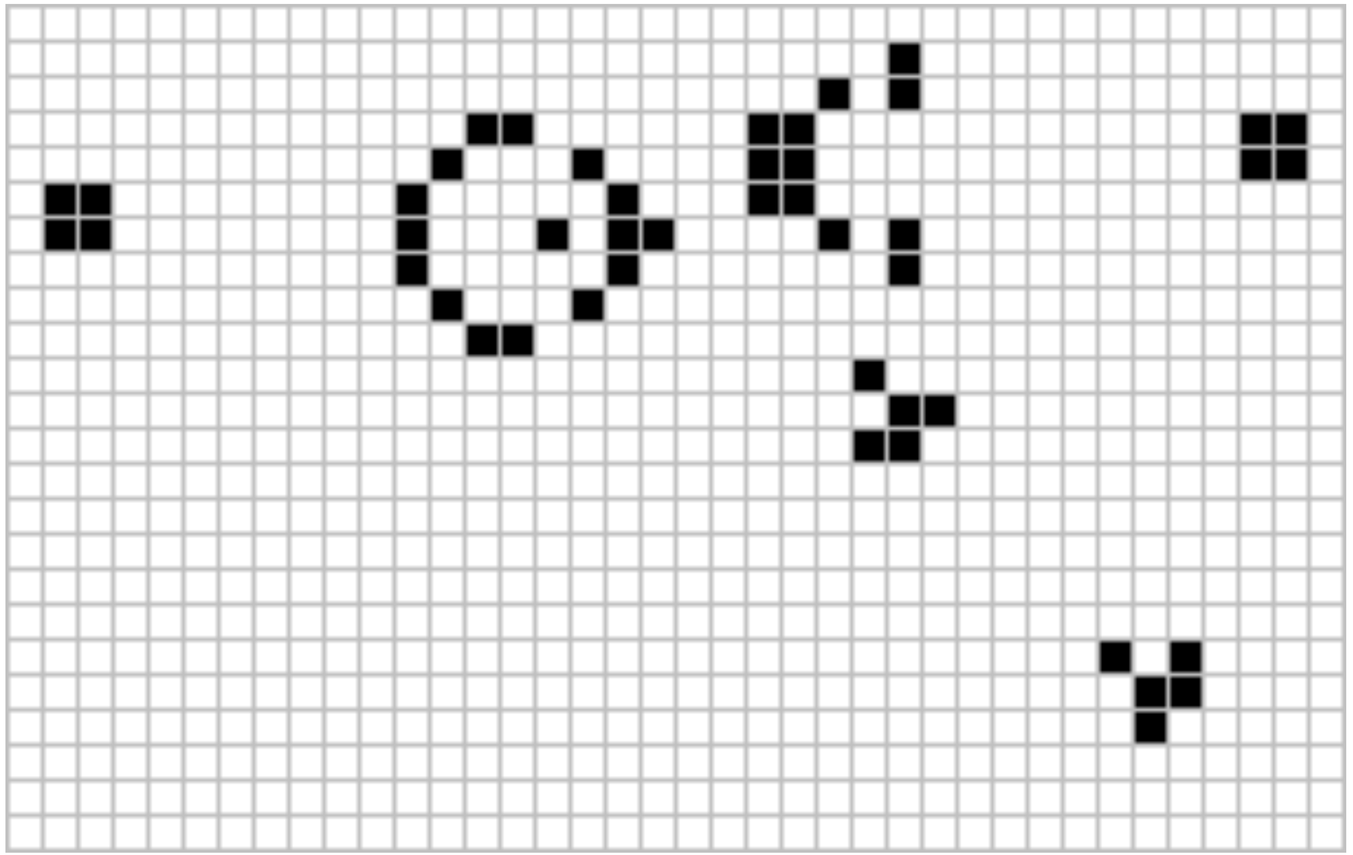
@dmagliola







```
while true
  new_board = (...)
  (1..100).each do |y|
    (1..100).each do |x|
      neighbour_count = (...)
      new_board[x][y] = (...)
    end
  end
  board = new_board
end
```



```
while true
  new_board = (...)
  (1..100).each do |y|
    (1..100).each do |x|
      neighbour_count = (...)
      new_board[x][y] = (...)
    end
  end
  board = new_board
end
```

```
while true
  board.gsub!(/SOME_REGEX/, "something")
end
```

The Game of Life, Regex'd

Can you implement the game of life, with just a Regex?

The Game of Life, Regex'd

Can you implement the game of life, with just a Regex?

Hi


indeed flex

The Game of Life, Regex'd

Can you implement the game of life, with just a Regex?

The Game of Life, Regex'd

Can you implement the game of life, with just a Regex?

@dmagliola

The Game of Life, Regex'd

Can you implement the game of life, with just a Regex?

Can you implement the game of life, with just a Regex?

WHY?

- A reasonable person

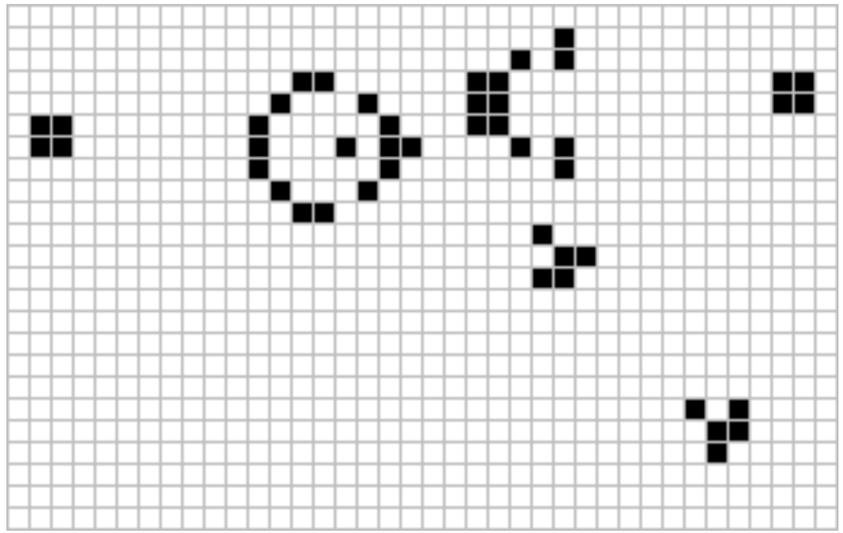
Can you implement the game of life, with just a Regex?

YES!

Can you implement the game of life, with just a Regex?

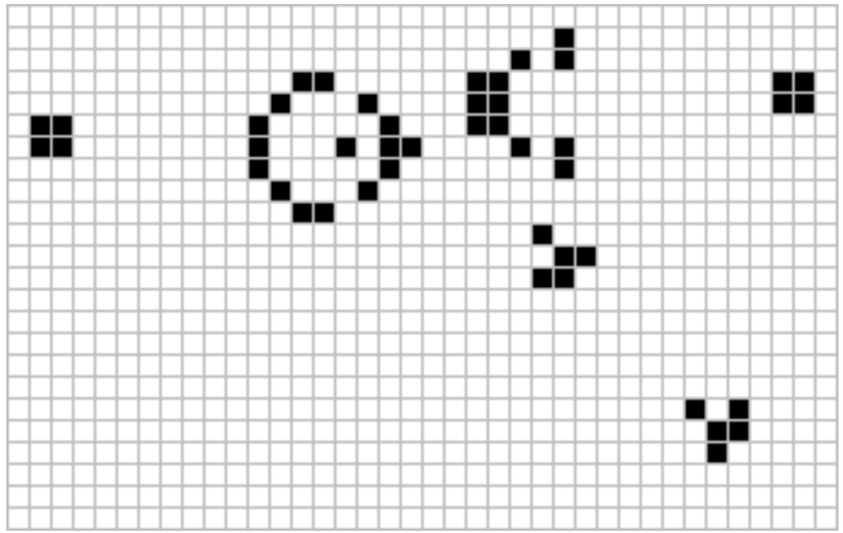
YES!, almost, kinda, it's complicated

The Rules



- "Infinite" 2D board of cells
- Each cell can be "alive" [1], or "dead" [0]
- Cells have 8 neighbours
- Live cells with fewer than 2 live neighbours die of loneliness 💔
- Live cells with more than 3 live neighbours die of overcrowding 👨‍👨‍👦‍👦
- Dead cells with exactly 3 live neighbours magically spawn into life 🐣

The Rules



- "Infinite" 2D board of cells
- Each cell can be "alive" [1], or "dead" [0]
- Cells have 8 neighbours
- Live cells with fewer than 2 live neighbours die of loneliness 💔
- Live cells with more than 3 live neighbours die of overcrowding 👨‍👨‍👦‍👦
- Dead cells with exactly 3 live neighbours magically spawn into life 🐣

The Rules

```
alive =
  neighbour_count == 3 ||
  neighbour_count == 2 && cell.alive?
```

The Rules

The Rules

0	0	0
0	1	0
0	1	0

The Rules

0	0	0
0	1	0
0	1	0

0	0	0
0	0	0
0	1	0

The Rules

0	0	0
0	1	0
0	1	0

0	0	0
0	0	0
0	1	0

0	0	0
0	1	0
1	0	1

The Rules

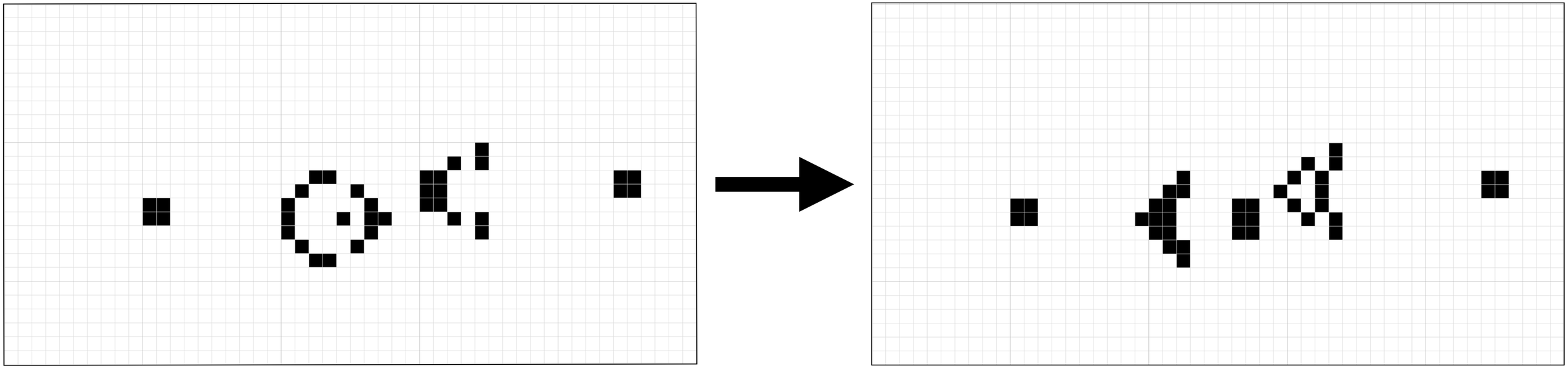
0	0	0
0	1	0
0	1	0

0	0	0
0	0	0
0	1	0

0	0	0
0	1	0
1	0	1

1	0	1
0	0	0
0	1	0

The General Approach



The General Approach

1) Find the cells that need to change (🪓 🐣)

The General Approach

1) Find the cells that need to change (🪓 🐣)

2) Change them into their new state (⚰️ 👍)

The General Approach

1) Find the cells that need to change (🪓 🐣)

2) Replace them into their new state (⚰️ 👍)

The General Approach

0) **Represent** this data conveniently for our cunning plan to work

1) Find the cells that need to change (🪓 🐣)

2) Replace them into their new state (⚰️ 👍)

The General Approach

🔢 Represent 🔍 Find 🖍️ Replace

0) **Represent** this data conveniently for our cunning plan to work

1) **Find** the cells that need to change (🪓 🐣)

2) **Replace** them into their new state (⚰️ 👍)

The General Approach

🔢 Represent | 🔍 Find | 🖍️ Replace

0	0	0
0	1	0
0	1	0

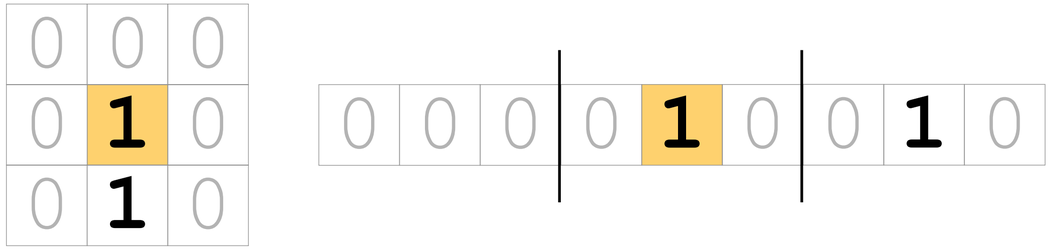
0	0	0
0	0	0
0	1	0

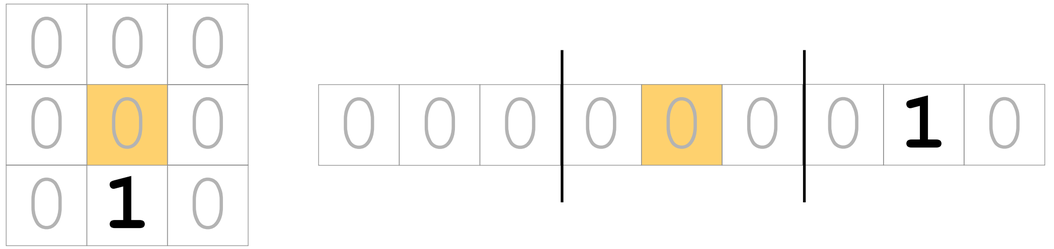
0	0	0
0	1	0
1	0	1

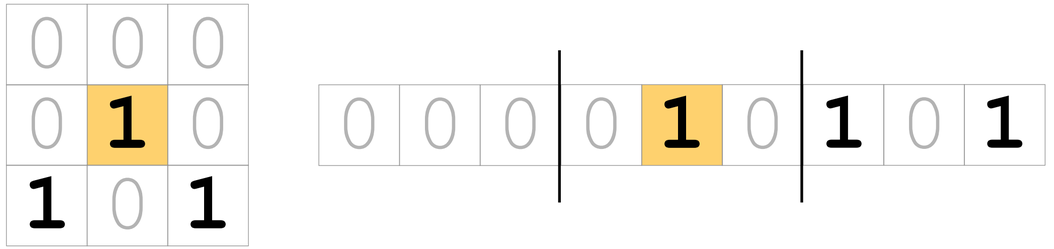
1	0	1
0	0	0
0	1	0

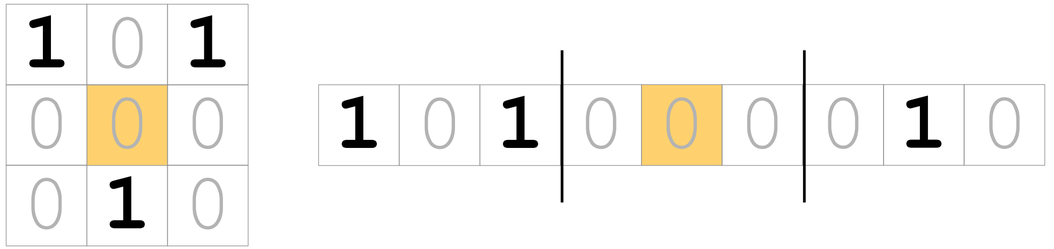
The General Approach

🔢 Represent 🔍 Find 🖍️ Replace



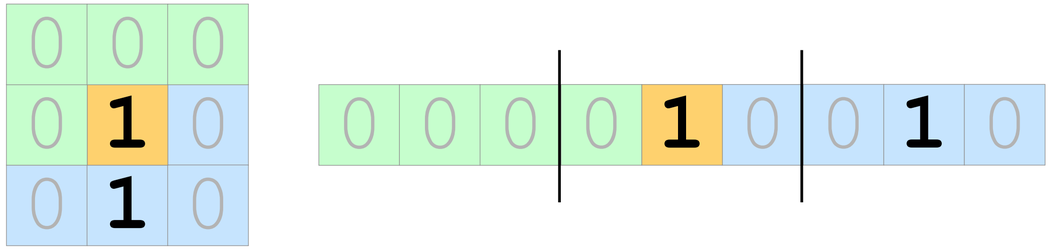


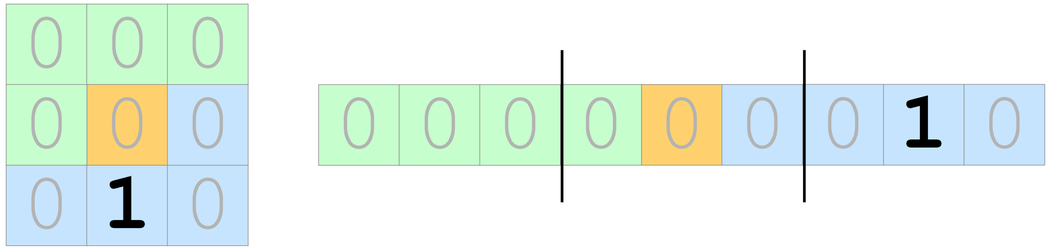


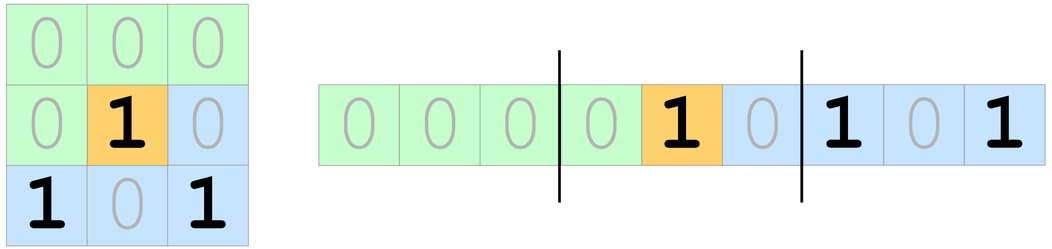


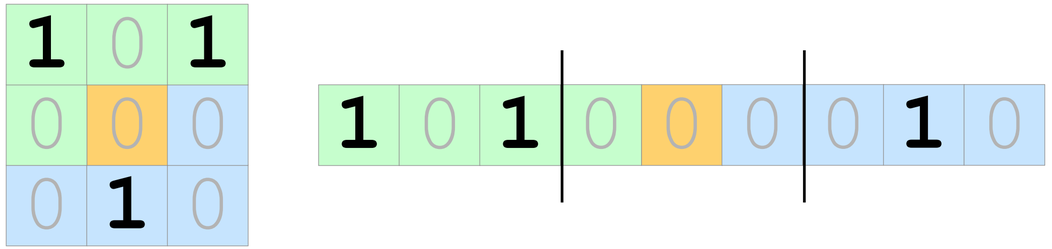
The General Approach

🔢 Represent 🔍 Find 🖍️ Replace



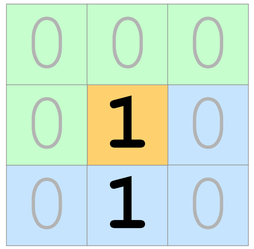


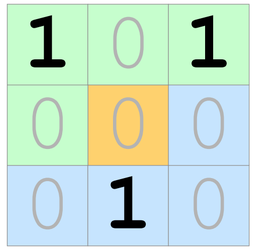




The General Approach

1234 Represent | 🔍 Find | 🖍 Replace





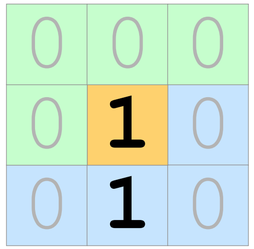
`(?<=matcher)`

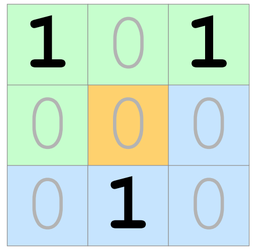
`(?=matcher)`

Lookbehind and Lookahead
Zero-Width Assertions

The General Approach

Represent | Find | Replace





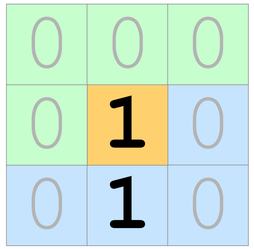
`(?<=matcher)`

`(?=matcher)`

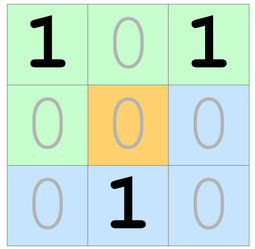
Lookbehind and Lookahead
Zero-Width Assertions

The General Approach

1234 Represent 🔍 Find ✏️ Replace



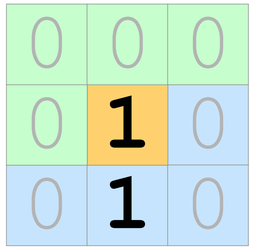

```
(?<=0000)1(?=0010)
```



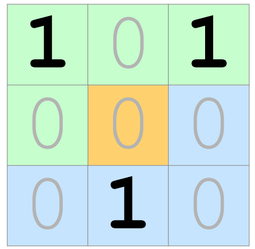

```
(?<=1010)0(?=0010)
```

The General Approach

1234 Represent 🔍 Find 🖍️ Replace



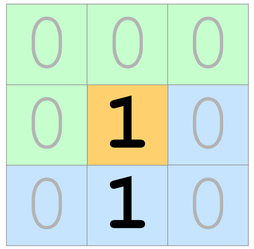
`(?<=0000)1(?=0010)` → `0` 🪓



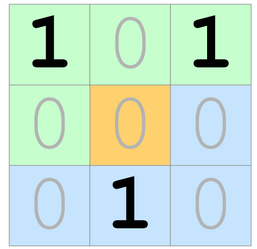
`(?<=1010)0(?=0010)` → `1` 🐣

The General Approach

1234 Represent Find Replace



`(?<=0000)1(?=0010)` → `0`

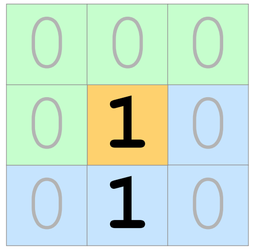


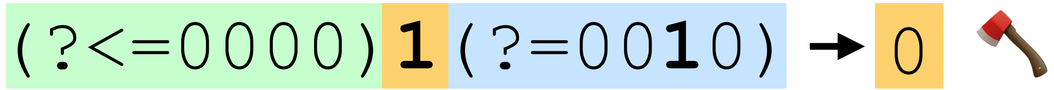
`(?<=1010)0(?=0010)` → `1`

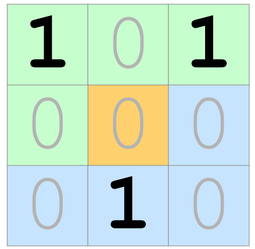
@dmagliola

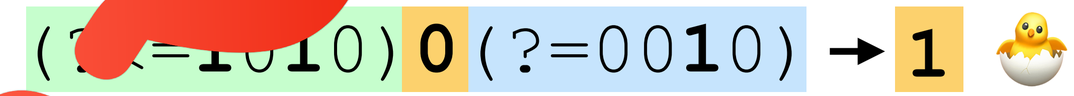
The General Approach

Represent Find Replace







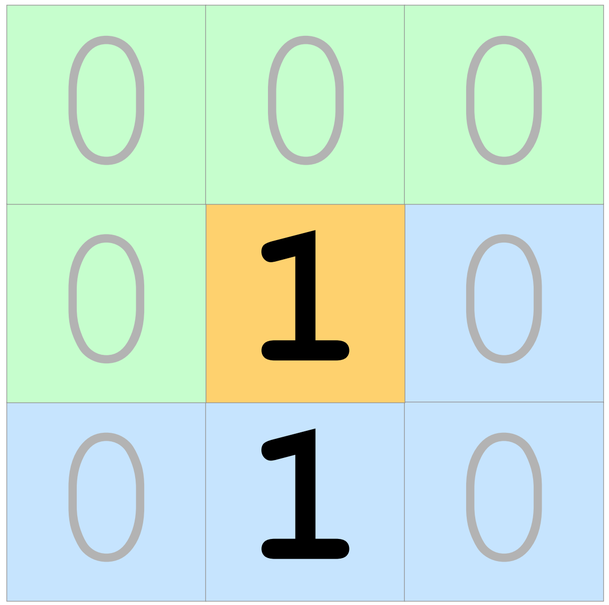


@dmagliola


DON'T CALL ME SHIRLEY
@dmagliola

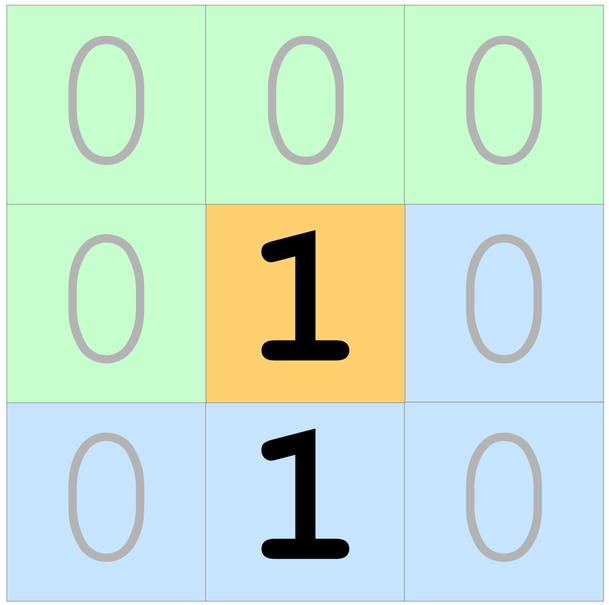
Matching changing cells

Represent | Find | Replace



Matching changing cells

Represent | Find | Replace



{2,3}

[1]{2,3}

Matching changing cells

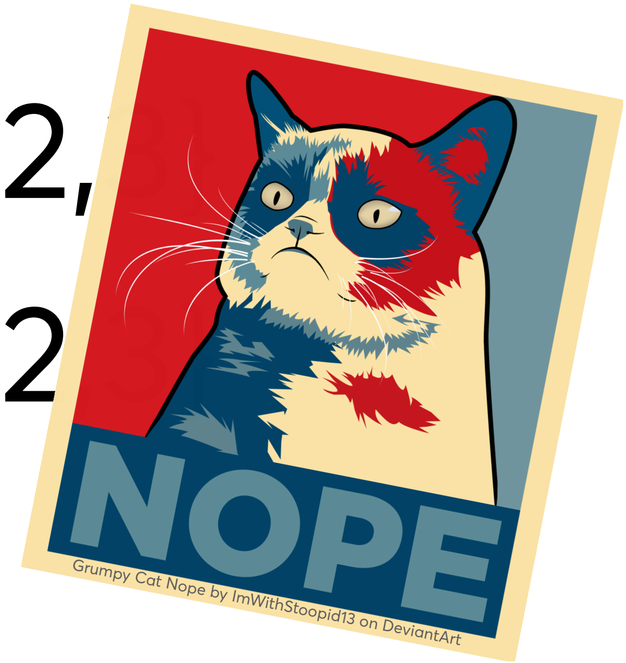
Represent | Find | Replace

0	0	0
0	1	0
0	1	0

0	0	0	0	1	0	0	1	0

{2,

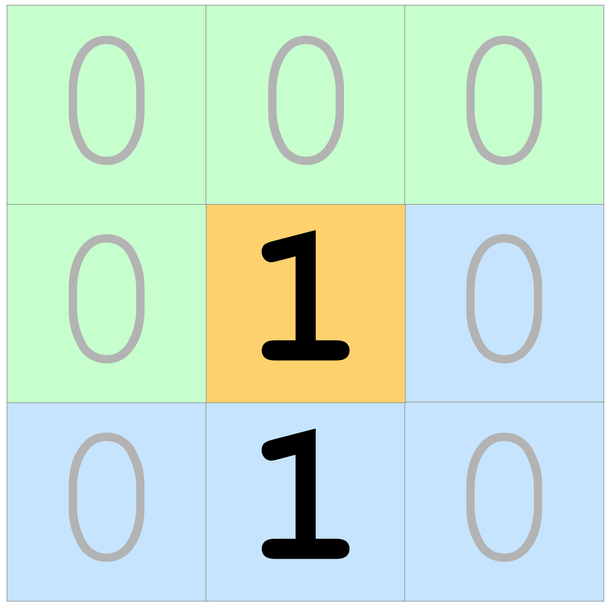
[1]{2




$\int a\,dx = c$
$\int a\,dx = a + c$
$\log_a x^r = r \log_a x$
$\log_a(x_1 x_2) = \log_a x_1 + \log_a x_2$
$P(A) = \frac{m(A)}{n}$
$P(A \cup B) = P(A) + P(B) - P(A \cap B)$
$[a_1, a_2] = a_1 + a_2 i$
$[a_1, a_2] = |a|(\cos\varphi + i\sin\varphi)$
$(af + bg)' = af' + bg'$
$\sqrt[n]{r}(\cos\varphi + i\sin\varphi) = \sqrt[n]{r}\left(\cos\frac{\varphi + 2k\pi}{n} + i\sin\frac{\varphi + 2k\pi}{n}\right)$
$(\operatorname{ctg} x)' = -\frac{1}{\sin^2 x}$
$6x - 4y = 0$
$a = 6$
$b = -4$
$6 = +15$
$D(3;15)$

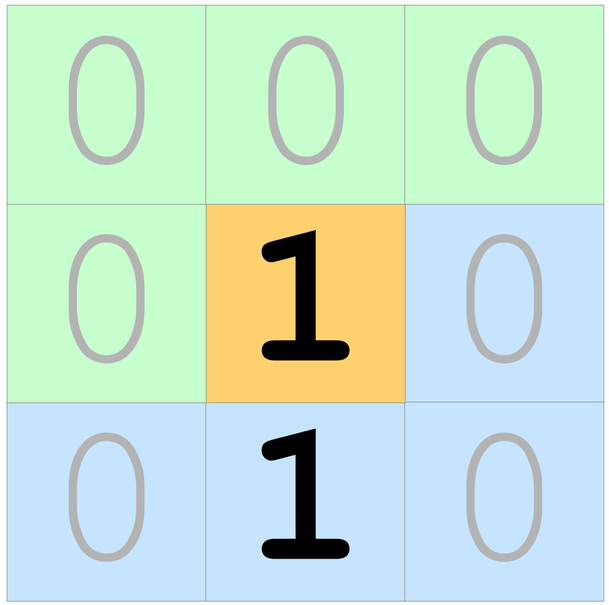
Matching changing cells

Represent | Find | Replace



Matching changing cells

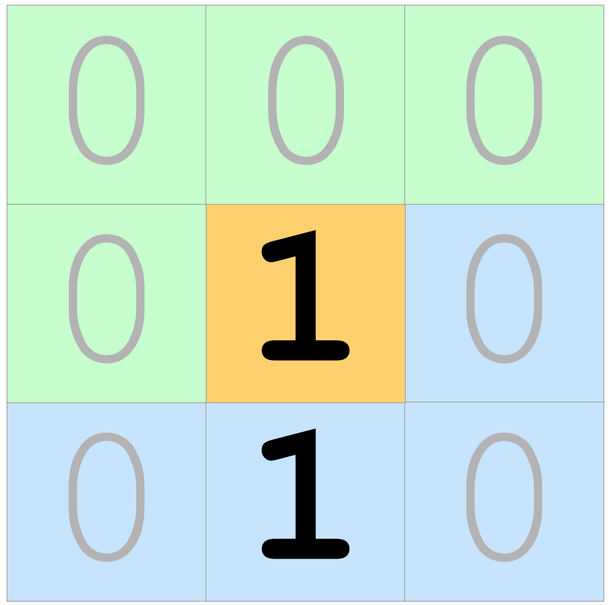
Represent | Find | Replace



$2^9 == 512$

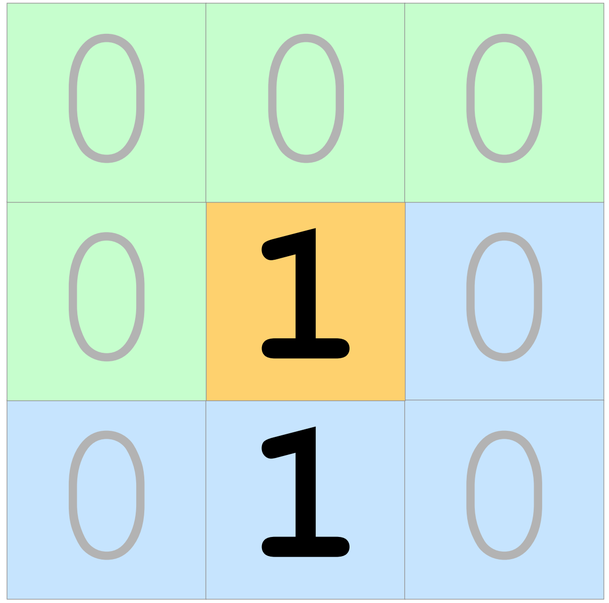
Matching changing cells

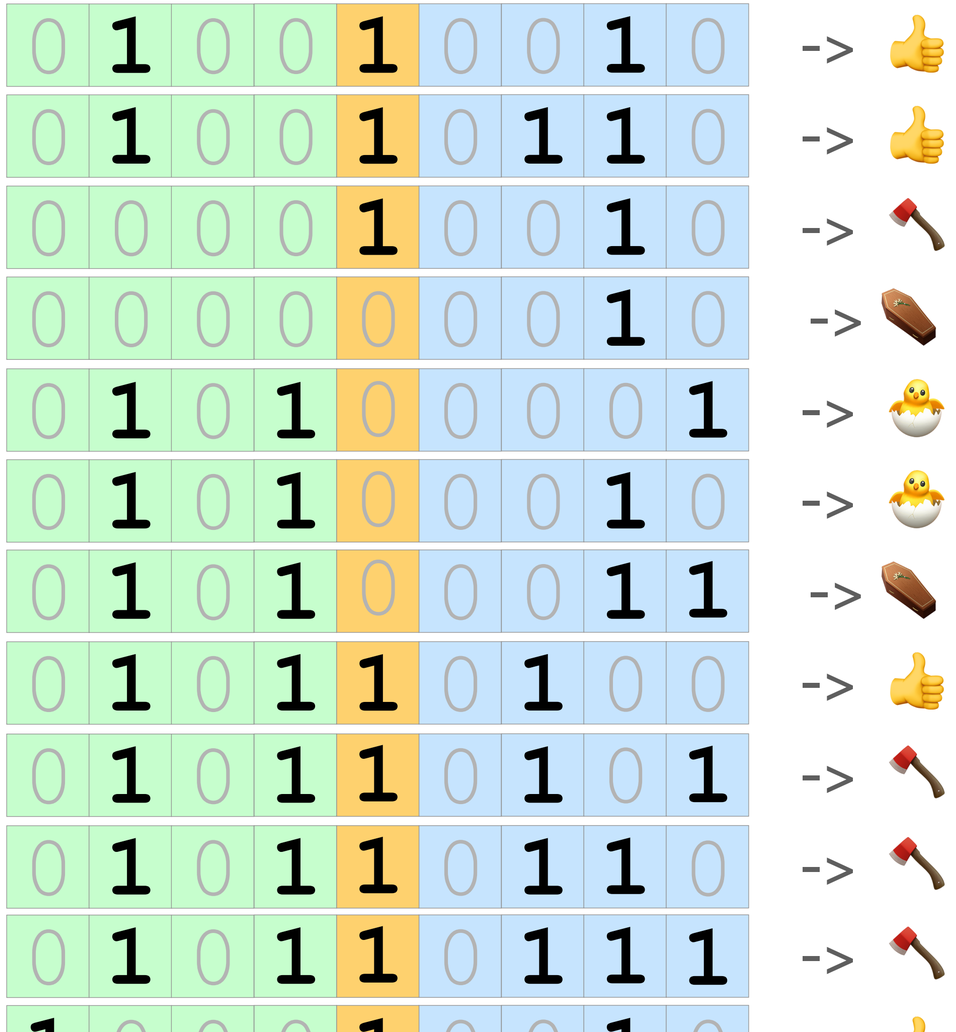
Represent | Find | Replace



Matching changing cells

Represent | Find | Replace

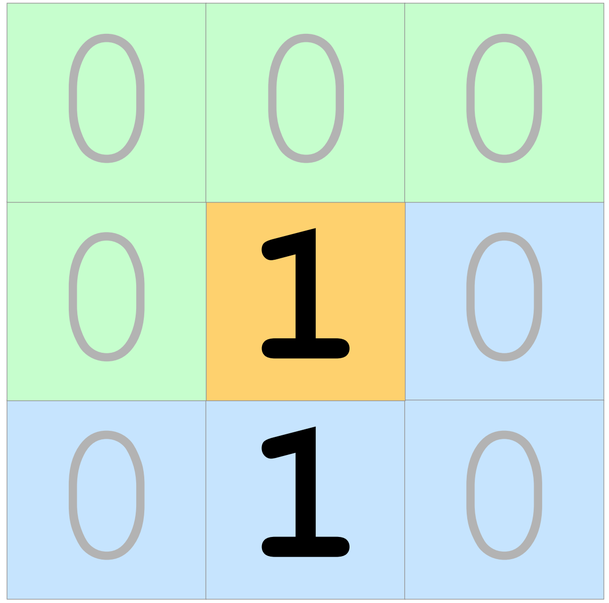


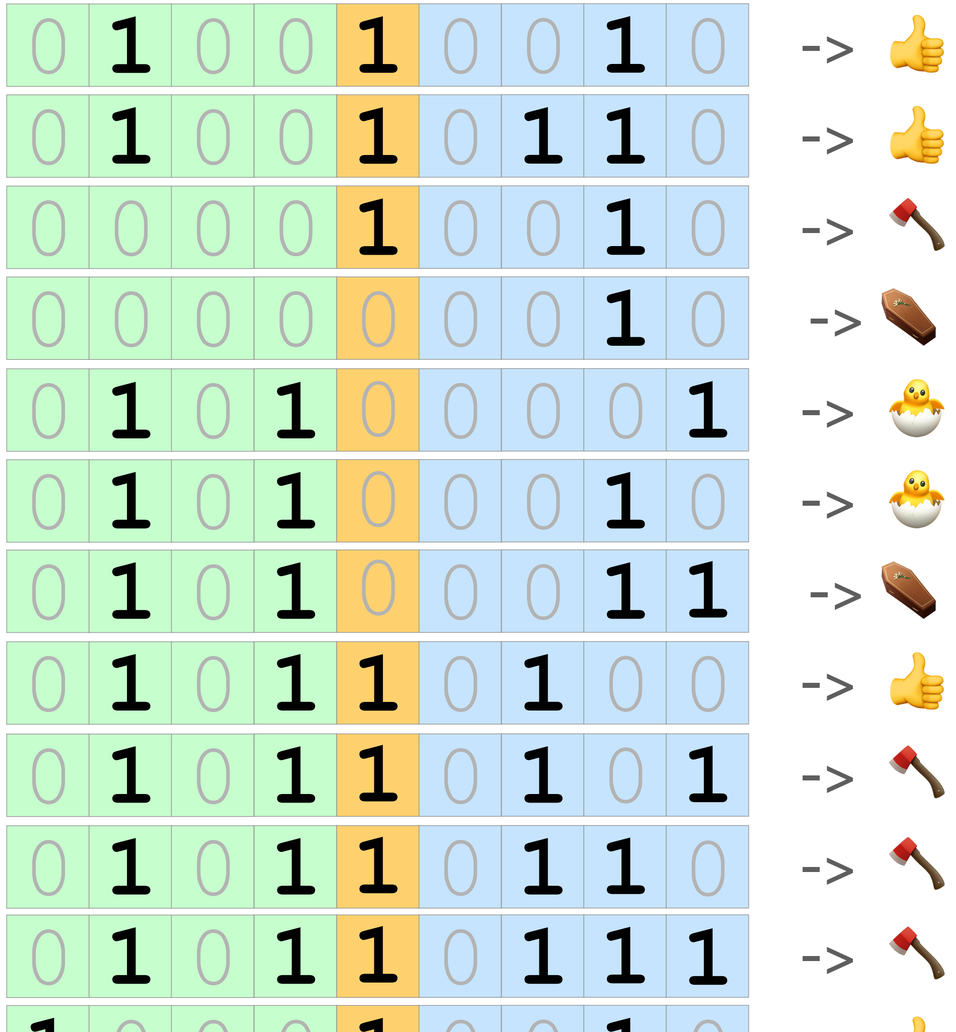


@dmagliola

Matching changing cells

Represent | Find | Replace

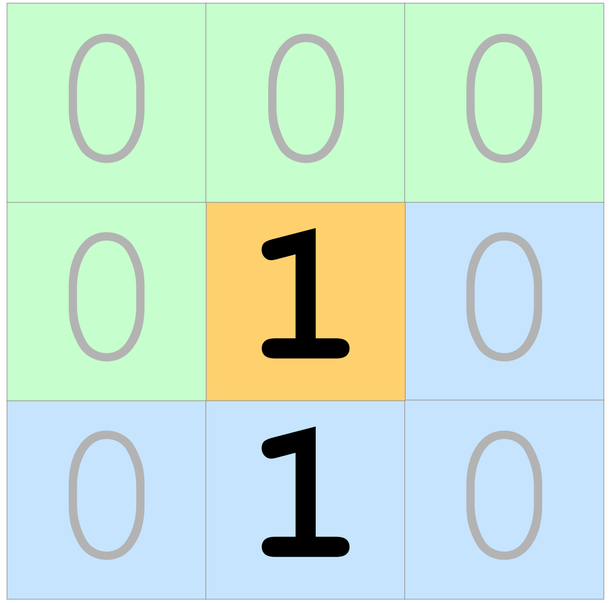


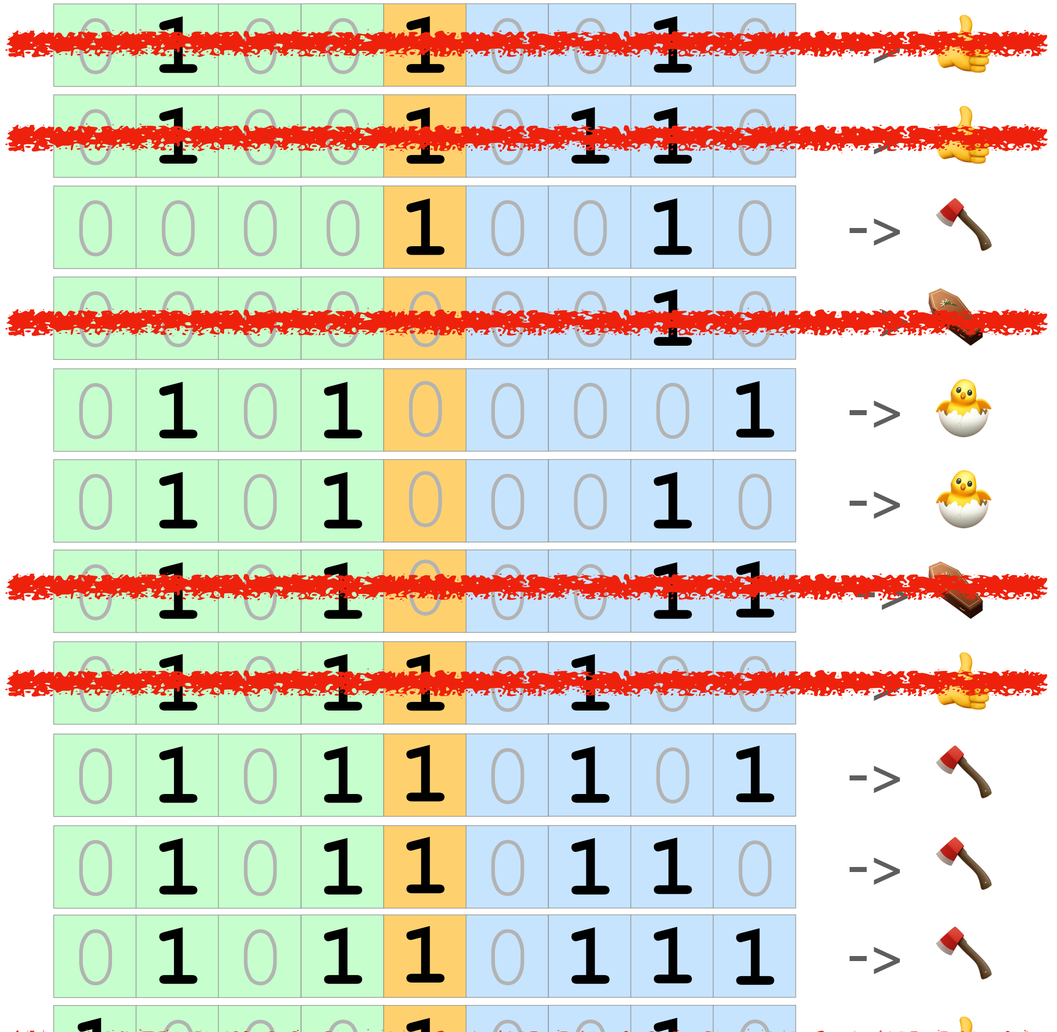


@dmagliola

Matching changing cells

Represent | Find | Replace





@dmagliola

Matching changing cells

Represent | Find | Replace

```
(0...512).each do |i|
  binary = i.to_s(2).rjust(9, "0")
  alive = binary[4].to_i # is the center cell alive

  neighbours = binary.chars.count{|d| d == "1" }
  neighbours -= 1 if alive == 1

  if (alive == 1 && neighbours != 2 && neighbours != 3) ||
      (alive == 0 && neighbours == 3)
    puts binary # cell should change
  end
end
```

Matching changing cells

Represent | Find | Replace

```
(0...512).each do |i|
  binary = i.to_s(2).rjust(9, "0")
  alive = binary[4].to_i # is the center cell alive

  neighbours = binary.chars.count{|d| d == "1" }
  neighbours -= 1 if alive == 1

  if (alive == 1 && neighbours != 2 && neighbours != 3) ||
      (alive == 0 && neighbours == 3)
    puts binary # cell should change
  end
end
```

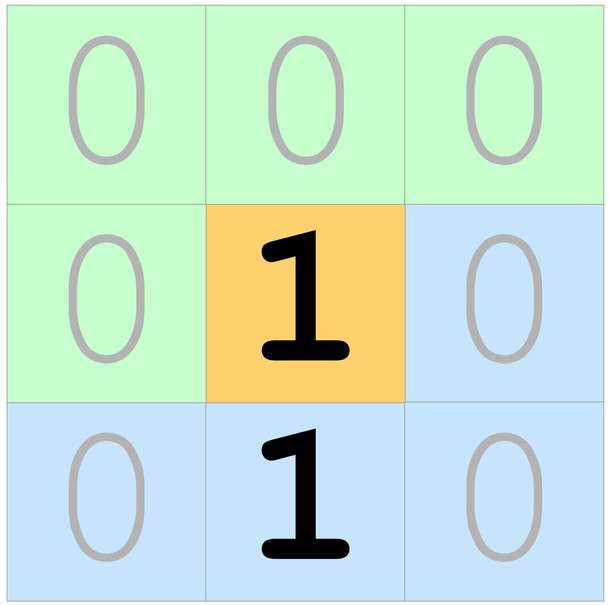
Matching changing cells

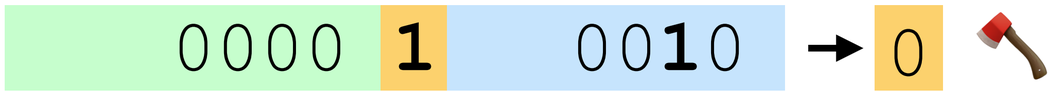
1 2 3 4 Represent 🔍 Find 🖍 Replace

```
(0...512).each do |i|
  binary = i.to_s(2).rjust(9, "0")
  alive = binary[4].to_i # is the center cell alive

  neighbours = binary.chars.count{|d| d == "1" }
  neighbours -= 1 if alive == 1

  if (alive == 1 && neighbours != 2 && neighbours != 3) ||
      (alive == 0 && neighbours == 3)
    puts binary # cell should change
  end
end
```

Matching changing cells

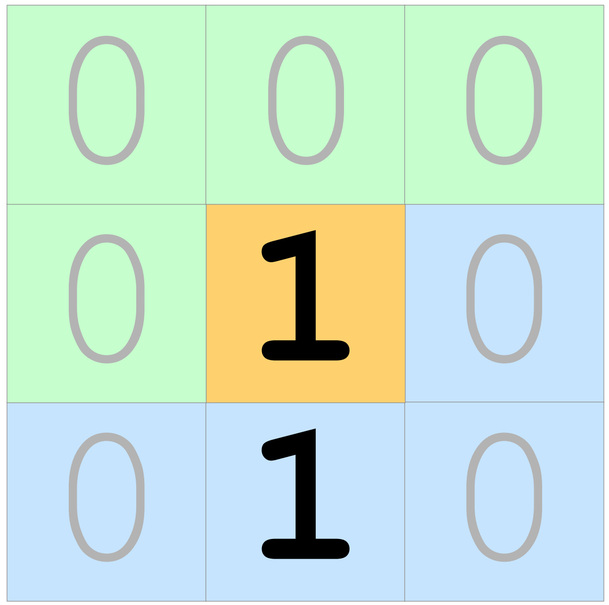
Represent | Find | Replace





Matching changing cells

Represent | Find | Replace

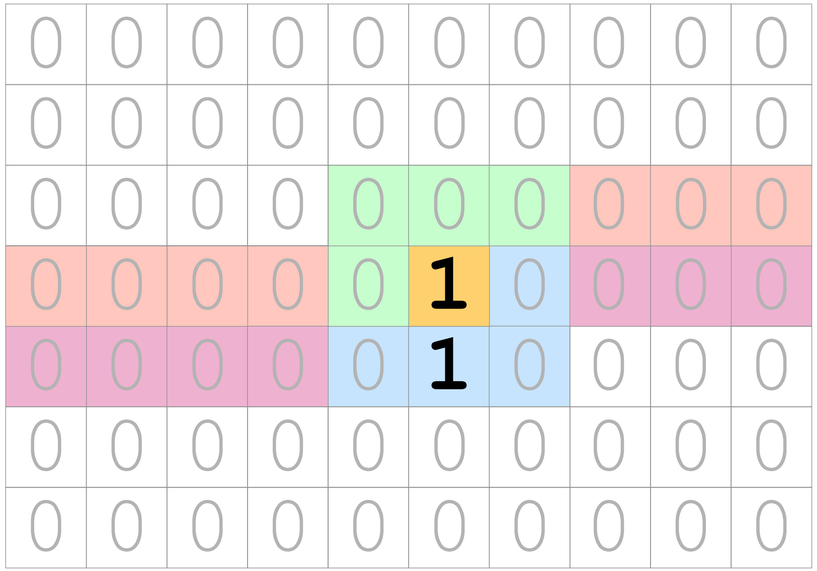


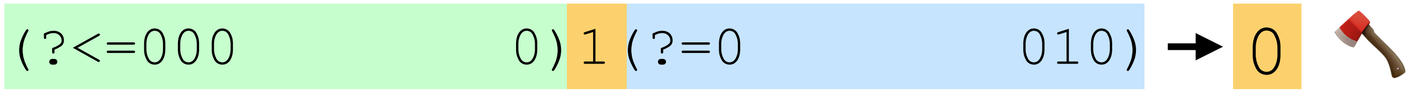
`0000 1 0010` → `0`

`(?<=0000)1(?=0010)` → `0`

Matching changing cells

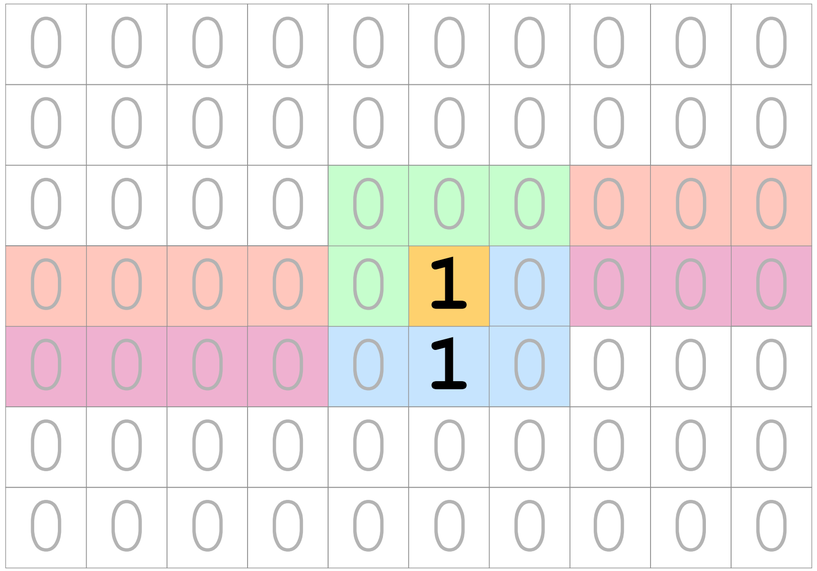
Represent | Find | Replace





Matching changing cells

Represent | Find | Replace




```
(?<=000          0)1(?=0          010) → 0
```

```
(?<=000(?:.{7})0)1(?=0(?:.{7})010) → 0
```

Non-capturing groups

Mega-Regex

1234 Represent 🔍 Find ✏️ Replace

Branch: master ▾ game_of_life / regex / **mega_regex.txt**

228 lines (228 sloc) | 11.8 KB

```
1   (?<=000(?:.{BOARD_SIZE})0)0(?=0(?:.{BOARD_SIZE})111)
2   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})011)
3   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})101)
4   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})110)
5   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})000)
6   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})001)
7   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})010)
8   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})100)
9   (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})000)
10  (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})111)
11  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})011)
12  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})101)
13  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})110)
14  (?<=000(?:.{BOARD_SIZE})1)0(?=1(?:.{BOARD_SIZE})001)
15  (?<=000(?:.{BOARD_SIZE})1)0(?=1(?:.{BOARD_SIZE})010)
16  (?<=000(?:.{BOARD_SIZE})1)0(?=1(?:.{BOARD_SIZE})100)
17  (?<=000(?:.{BOARD_SIZE})1)1(?=0(?:.{BOARD_SIZE})000)
```

@dmagliola

Mega-Regex

Represent 🔍 Find Replace

Branch: master ▾ game_of_life / regex / mega_regex.txt

228 lines (228 sloc) | 11.8 KB

```
1   (?<=000(?:.{BOARD_SIZE})0)0(?=0(?:.{BOARD_SIZE})111)
2   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})011)
3   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})101)
4   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})110)
5   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})000)
6   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})001)
7   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})010)
8   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})100)
9   (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})000)
10  (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})111)
11  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})011)
12  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})101)
13  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})110)
```

Top row Middle row Bottom row

Mega-Regex

Represent 🔍 Find Replace

```ruby
def load_mega_regex(board_size)
  lines = IO.readlines("mega_regex.txt")
  regex_string = lines.join("|")
  regex_string = regex_string.gsub(
      "BOARD_SIZE",
      (board_size - 3).to_s
  )
  Regexp.new(regex_string)
end
```

Branch: master — game_of_life / regex / mega_regex.txt

228 lines (228 sloc) | 11.8 KB

```
1   (?<=000(?:.{BOARD_SIZE})0)0(?=0(?:.{BOARD_SIZE})111)
2   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})011)
3   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})101)
4   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})110)
5   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})000)
6   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})001)
7   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})010)
8   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})100)
9   (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})000)
10  (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})111)
11  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})011)
12  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})101)
13  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})110)
```

Top row Middle row Bottom row

Mega-Regex

Represent 🔍 Find Replace

```ruby
def load_mega_regex(board_size)
  lines = IO.readlines("mega_regex.txt")
  regex_string = lines.join("|")
  regex_string = regex_string.gsub(
      "BOARD_SIZE",
      (board_size - 3).to_s
  )
  Regexp.new(regex_string)
end
```

Branch: master — game_of_life / regex / mega_regex.txt

228 lines (228 sloc) | 11.8 KB

```
1   (?<=000(?:.{BOARD_SIZE})0)0(?=0(?:.{BOARD_SIZE})111)
2   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})011)
3   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})101)
4   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})110)
5   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})000)
6   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})001)
7   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})010)
8   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})100)
9   (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})000)
10  (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})111)
11  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})011)
12  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})101)
13  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})110)
```

Top row Middle row Bottom row

Mega-Regex

Represent 🔍 Find Replace

```ruby
def load_mega_regex(board_size)
  lines = IO.readlines("mega_regex.txt")
  regex_string = lines.join("|")
  regex_string = regex_string.gsub(
      "BOARD_SIZE",
      (board_size - 3).to_s
  )
  Regexp.new(regex_string)
end
```

Branch: master — game_of_life / regex / mega_regex.txt

228 lines (228 sloc) | 11.8 KB

```
1   (?<=000(?:.{BOARD_SIZE})0)0(?=0(?:.{BOARD_SIZE})111)
2   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})011)
3   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})101)
4   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})110)
5   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})000)
6   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})001)
7   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})010)
8   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})100)
9   (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})000)
10  (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})111)
11  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})011)
12  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})101)
13  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})110)
```

Top row Middle row Bottom row

Mega-Regex

Represent 🔍 Find Replace

```ruby
def load_mega_regex(board_size)
  lines = IO.readlines("mega_regex.txt")
  regex_string = lines.join("|")
  regex_string = regex_string.gsub(
      "BOARD_SIZE",
      (board_size - 3).to_s
  )
  Regexp.new(regex_string)
end
```

Branch: master — game_of_life / regex / mega_regex.txt

228 lines (228 sloc) | 11.8 KB

```
1   (?<=000(?:.{BOARD_SIZE})0)0(?=0(?:.{BOARD_SIZE})111)
2   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})011)
3   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})101)
4   (?<=000(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})110)
5   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})000)
6   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})001)
7   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})010)
8   (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})100)
9   (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})000)
10  (?<=000(?:.{BOARD_SIZE})0)1(?=1(?:.{BOARD_SIZE})111)
11  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})011)
12  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})101)
13  (?<=000(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})110)
```

Top row Middle row Bottom row

Mega-Regex

🔢 Represent 🔍 Find ✏️ Replace

```
/(?<=000(?:.{67})0)0(?=0(?:.{67})111)|(?<=000(?:.{67})0)0(?=1(?:.
{67})011)|(?<=000(?:.{67})0)0(?=1(?:.{67})101)|(?<=000(?:.{67})0)0(?
=1(?:.{67})110)|(?<=000(?:.{67})0)1(?=0(?:.{67})000)|(?<=000(?:.
{67})0)1(?=0(?:.{67})001)|(?<=000(?:.{67})0)1(?=0(?:.{67})010)|(?
<=000(?:.{67})0)1(?=0(?:.{67})0)1(?=1(?:.{67})000)|
(?<=000(?:.{67})0)1(?=1(?:.{67})1)0(?=0(?:.
{67})011)|(?<=000(?:.{67})1)0(?=1(?:.{67})101)|(?<=000(?:.{67})1)0(?
=0(?:.{67})110)|(?<=000(?:.{67})1)0(?=0(?:.{67})001)|(?<=000(?:.
{67})1)0(?=1(?:.{67})010)|(?<=000(?:.{67})1)0(?=1(?:.{67})100)|(?
<=000(?:.{67})1)1(?=0(?:.{67})000(?:.{67})1)1(?=0(?:.{67})111)|
(?<=000(?:.{67})1)1(?=1(?:.{67})000(?<=000(?:.{67})1)1(?=1(?:.
{67})101)|(?<=000(?:.{67})1)1(?=1(?:.{67})110)|(?<=000(?:.{67})1)1(?
=1(?:.{67})111)|(?<=001(?:.{67})0)0(?=0(?:.{67})011)|(?<=001(?:.
{67})0)0(?=0(?:.{67})101)|(?<=001(?:.{67})0)0(?=0(?:.{67})110)|(?
<=001(?:.{67})0)0(?=1(?:.{67})001)|(?<=001(?:.{67})0)0(?=1(?:.{67})010)|
(?<=001(?:.{67})0)0(?=1(?:.{67})100)|(?<=001(?:.{67})0)1(?=0(?:.
{67})000)|(?<=001(?:.{67})0)1(?=0(?:.{67})111)|(?<=001(?:.{67})0)1(?
```

Mega-Regex

🔢 Represent 🔍 Find ✏️ Replace

```
{67})011)|(?<=111(?:.{67})0)1(?=0(?:.{67})100)|(?<=111(?:.{67})0)1(?
=0(?:.{67})101)|(?<=111(?:.{67})0)1(?=0(?:.{67})110)|(?<=111(?:.
{67})0)1(?=0(?:.{67})111)|(?<=111(?:.{67})0)1(?=1(?:.{67})000)|(?
<=111(?:.{67})0)1(?=1(?:.{67})001)|(?<=111(?:.{67})0)1(?=1(?:.{67})010)|
(?<=111(?:.{67})0)1(?=1(?:.{67})011)|(?<=111(?:.{67})0)1(?=1(?:.
{67})100)|(?<=111(?:.{67})0)1(?=1(?:.{67})101)|(?<=111(?:.{67})0)1(?
=1(?:.{67})110)|(?<=111(?:.{67})0)1(?=1(?:.{67})111)|(?<=111(?:.
{67})1)1(?=0(?:.{67})000)|(?<=111(?:.{67})1)1(?=0(?:.{67})001)|(?
<=111(?:.{67})1)1(?=0(?:.{67})011)|
(?<=111(?:.{67})1)1(?=0(?:.{67})101)|(?<=111(?:.{67})1)1(?=0(?:.
{67})110)|(?<=111(?:.{67})1)1(?=0(?:.{67})111)|(?<=111(?:.{67})1)1(?
=1(?:.{67})000)|(?<=111(?:.{67})1)1(?=1(?:.{67})001)|(?<=111(?:.
{67})1)1(?=1(?:.{67})010)|(?
<=111(?:.{67})1)1(?=1(?:.{67})011)|(?<=111(?:.{67})1)1(?=1(?:.{67})100)|
(?<=111(?:.{67})1)1(?=1(?:.{67})101)|(?<=111(?:.{67})1)1(?=1(?:.
{67})110)|(?<=111(?:.{67})1)1(?=1(?:.{67})111)/
```

8.2 kb of regex!!!

@dmagliola

Mega-Regex

🔢 Represent 🔍 Find 🖍️ Replace

```
board.gsub(mega_regex, '?????')
```

Mega-Regex

🔢 Represent 🔍 Find 🖍️ Replace

```
board.gsub(mega_regex, '?????')
```

Mega-Regex

1234 Represent 🔍 Find 🖍️ Replace

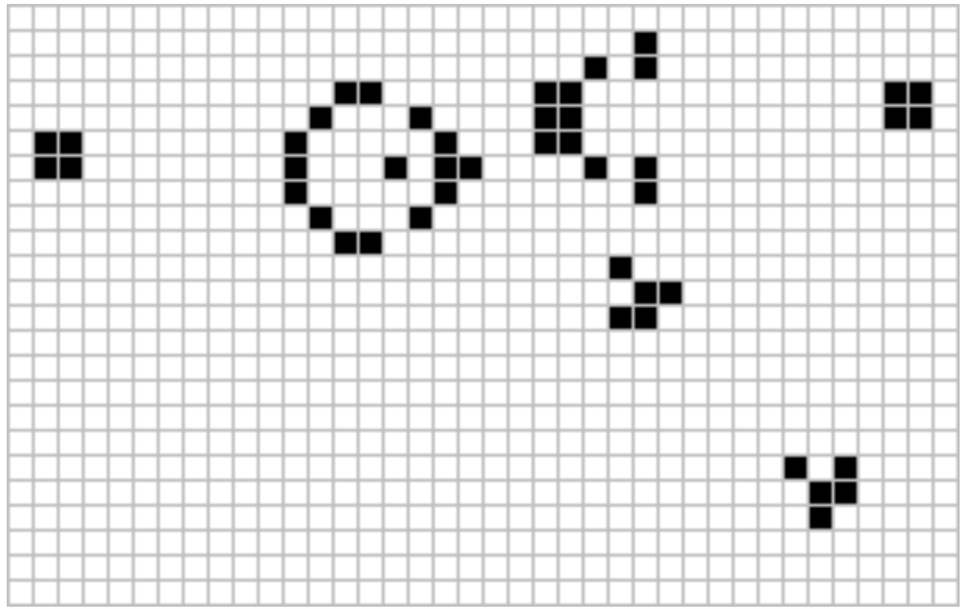
```
board.gsub(mega_regex, '?????')
```

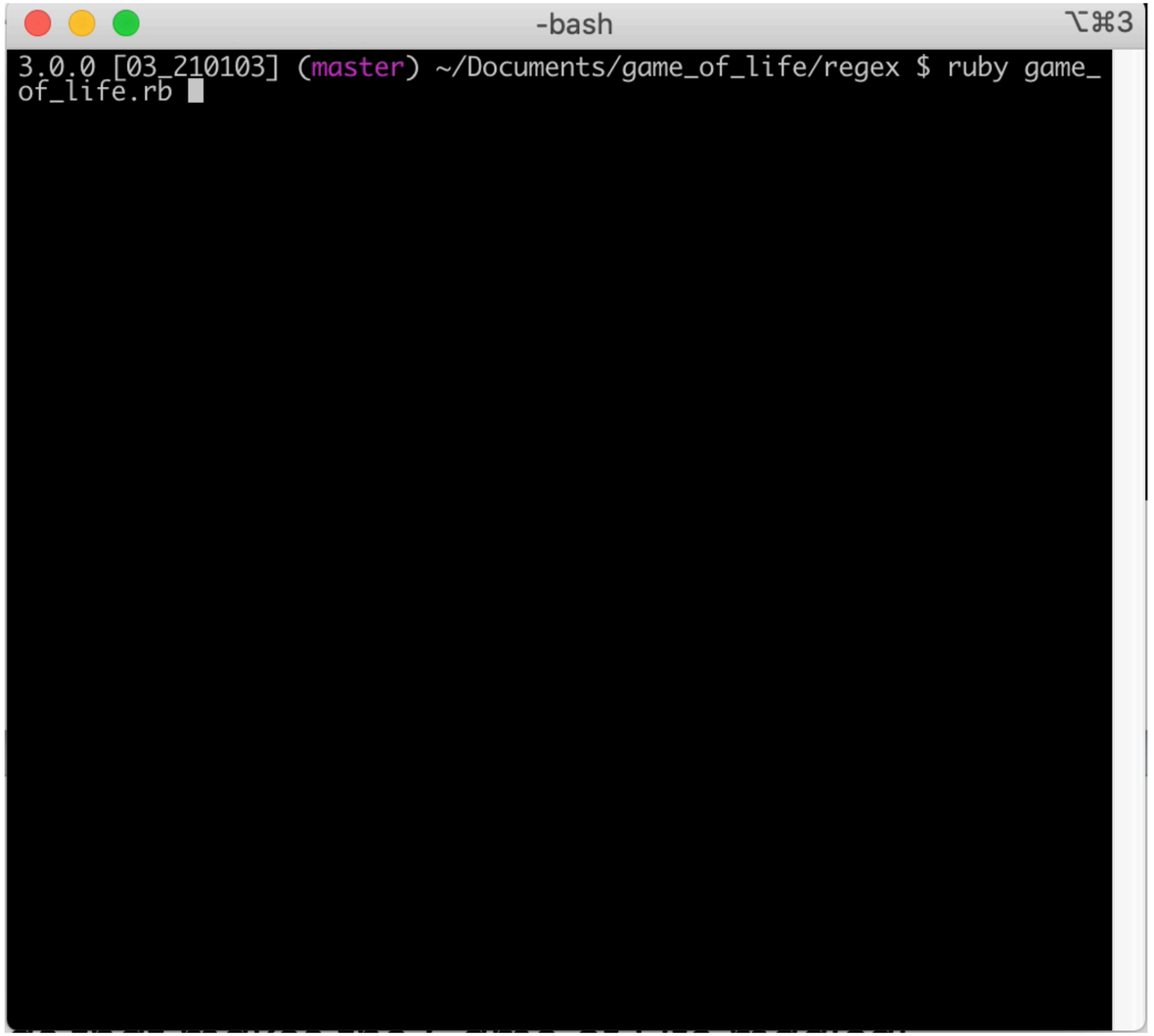
Mega-Regex

Represent Find Replace

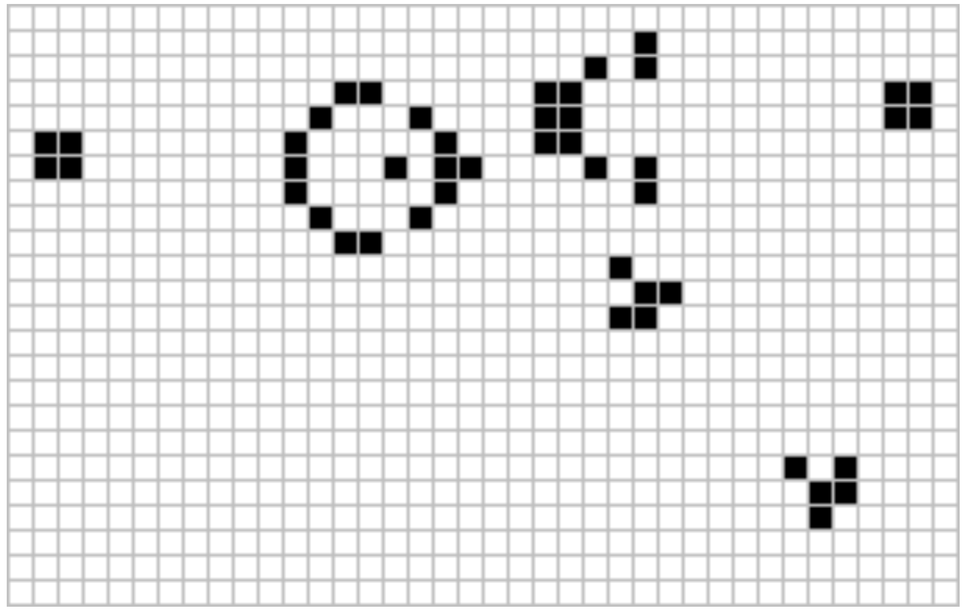
```
board.gsub(mega_regex, '??????')
board.gsub(mega_regex) { |m| m == "0" ? "1" : "0" }
```

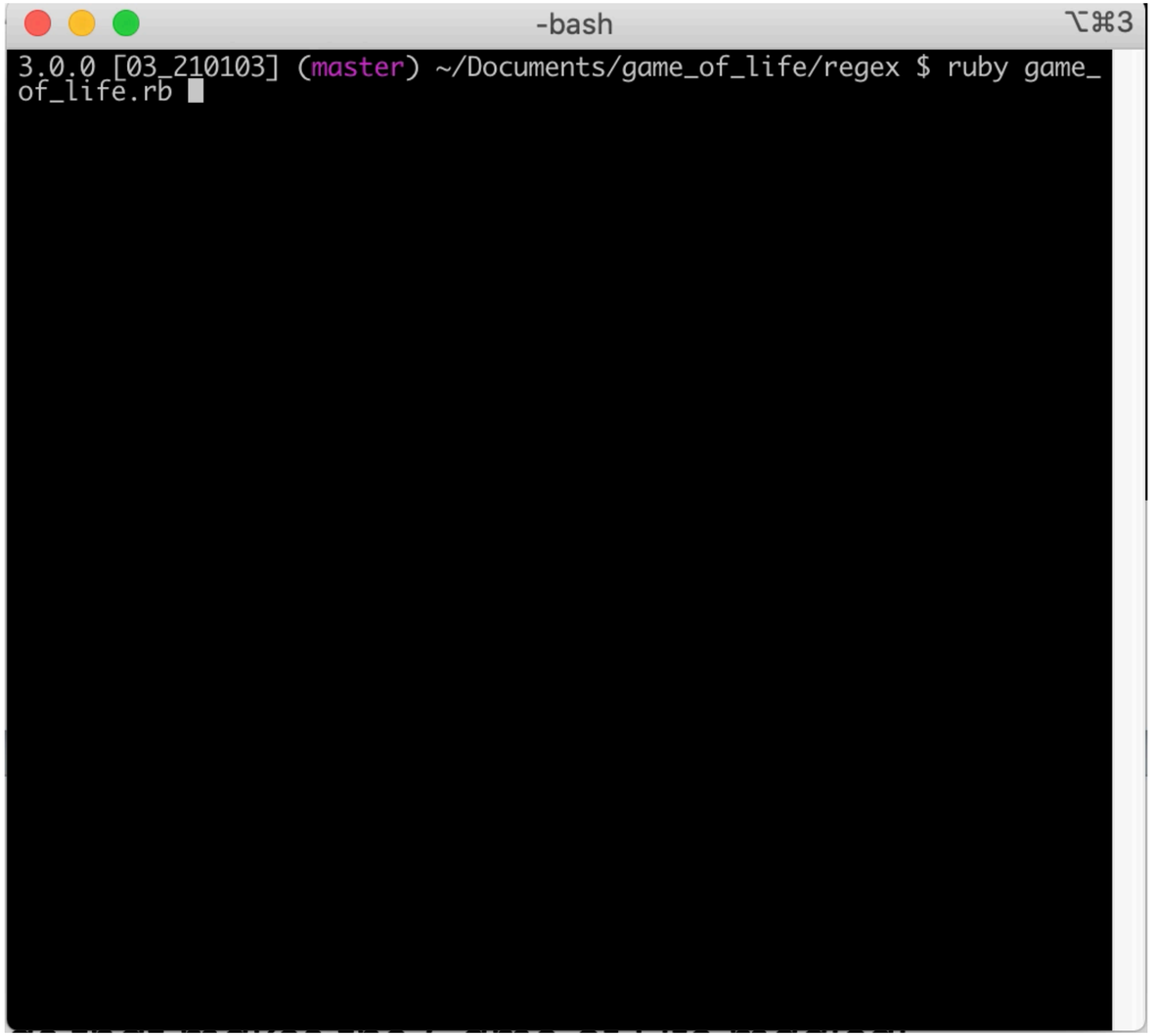
It's alive!




-bash
⌥⌘3
3.0.0 [03_210103] (master) ~/Documents/game_of_life/regex $ ruby game_
of_life.rb

It's alive!




-bash
⌥⌘3
3.0.0 [03_210103] (master) ~/Documents/game_of_life/regex $ ruby game_
of_life.rb

The cake is a lie

🔢 Represent 🔍 Find 🖍️ Replace

```
board.gsub(mega_regex) { |m| m == "0" ? "1" : "0" }
```

Kill first, spawn later?

Represent Find Replace

```
DEATH_REGEX = /(?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})000)|
              (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})001)|
              (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})001)|
              (?<=000(?:.{BOARD_SIZE})0)1(?=0(?:.{BOARD_SIZE})010)|
              (...)/

SPAWN_REGEX = /(?<=001(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})001)|
              (?<=001(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})010)|
              (?<=001(?:.{BOARD_SIZE})0)0(?=1(?:.{BOARD_SIZE})100)|
              (?<=001(?:.{BOARD_SIZE})1)0(?=0(?:.{BOARD_SIZE})001)|
              (...)/

              board = board.gsub(DEATH_REGEX, '0')
              board = board.gsub(SPAWN_REGEX, '1')
```

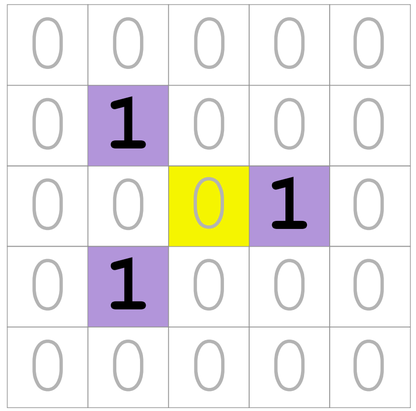
Kill first, spawn later?

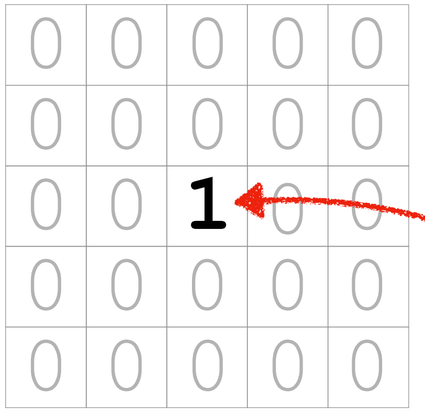
Represent Find Replace

```
while true
  new_board = (...)
  (1..100).each do |y|
    (1..100).each do |x|
      neighbour_count = (...)
      new_board[x][y] = (...)
    end
  end
  board = new_board
end
```

Kill first, spawn later?

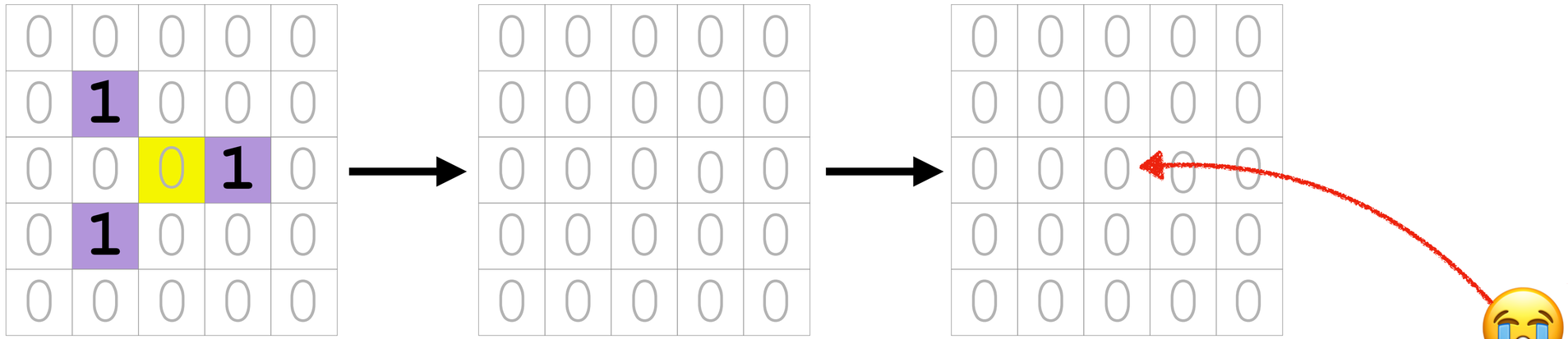
Represent Find Replace





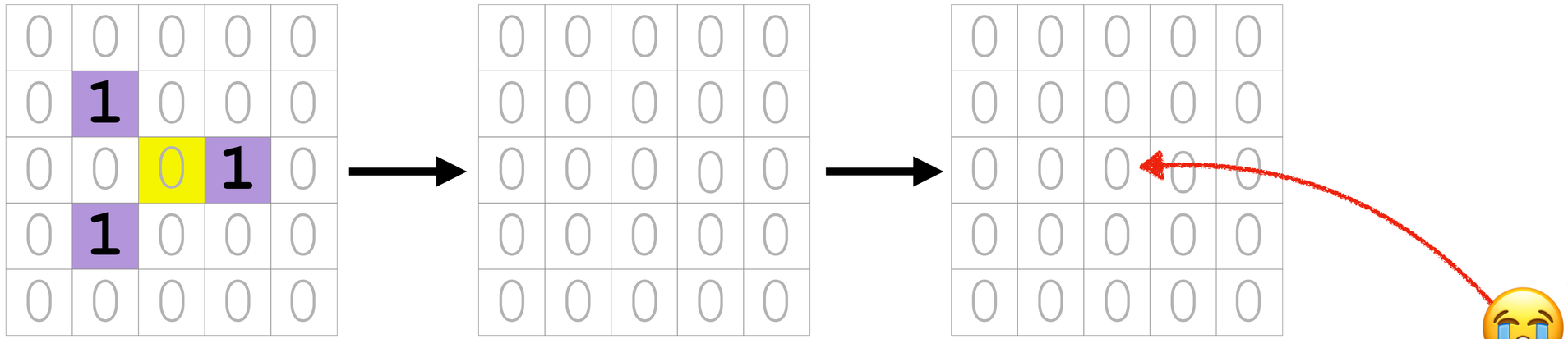
```
while true
  new_board = (...)
  (1..100).each do |y|
    (1..100).each do |x|
      neighbour_count = (...)
      new_board[x][y] = (...)
    end
  end
  board = new_board
end
```

Kill first, spawn later?




```
#Death
board = board.gsub(DEATH_REGEX, '0')

#Spawn
board = board.gsub(SPAWN_REGEX, '1')
```

Kill first, spawn later?



Death

Spawn

```
#Death
board = board.gsub(DEAT

#Spawn
board = board.gsub(SPAWN
```

Named Captures

🔢 Represent 🔍 Find 🖍️ Replace

```
6162345678 => (616) 234-5678
```

Named Captures


```
6162345678 => (616) 234-5678
```

```
/(\d{3})(\d{3})(\d{4})/
```

Named Captures

Represent Find Replace

```
6162345678 => (616) 234-5678


phone_number.gsub(
    /(\d{3})(\d{3})(\d{4})/,
    "(\1) \2-\3"
)
```

Named Captures

Represent Find Replace

```
6162345678 => (616) 234-5678

phone_number.gsub(
    /(\d{3})(\d{3})(\d{4})/,
    "(\1) \2-\3"
)

phone_number.gsub(
    /(?<area>\d{3})(?<middle>\d{3})(?<last>\d{4})/,
    "(\k<area>) \k<middle>-\k<last>"
)
```

Named Captures

Represent Find Replace

```
6162345678 => (616) 234-5678

phone_number.gsub(
    /(\d{3})(\d{3})(\d{4})/,
    "(\1) \2-\3"
)

phone_number.gsub(
    /(?<area>\d{3})(?<middle>\d{3})(?<last>\d{4})/,
    "(\k<area>) \k<middle>-\k<last>"
)
```

Named Captures


```
(?<=          0 00)1
```

Named Captures

🔢 Represent 🔍 Find 🖍️ Replace

```
(?<=(?<replace>0)00)1
```

Named Captures

🔢 Represent 🔍 Find 🖍️ Replace

```
(?<=(?<replace>0)00)1|(?<=(?<replace>1)01)0
```

The Dream!


```
(?<=(?<replace>0)00)1|(?<=(?<replace>1)01)0
```

This matches the final digit in: `0001` `1010`

And `\k<replace>` has value: `0` `1`

So now I can:

```
board.gsub(regex, '\k<replace>')
```

The Dream!

Represent Find Replace

Before:

```
(?<=001(?:.{BOARD_SIZE})1)1(?=0(?:.{BOARD_SIZE})011)
(?<=110(?:.{BOARD_SIZE})0)0(?=0(?:.{BOARD_SIZE})010)
```

After:

```
(?<=0(?<replace>0)1(?:.{BOARD_SIZE})1)1(?=0(?:.{BOARD_SIZE})011)
(?<=(?<replace>1)10(?:.{BOARD_SIZE})0)0(?=0(?:.{BOARD_SIZE})010)
```

The Dream!

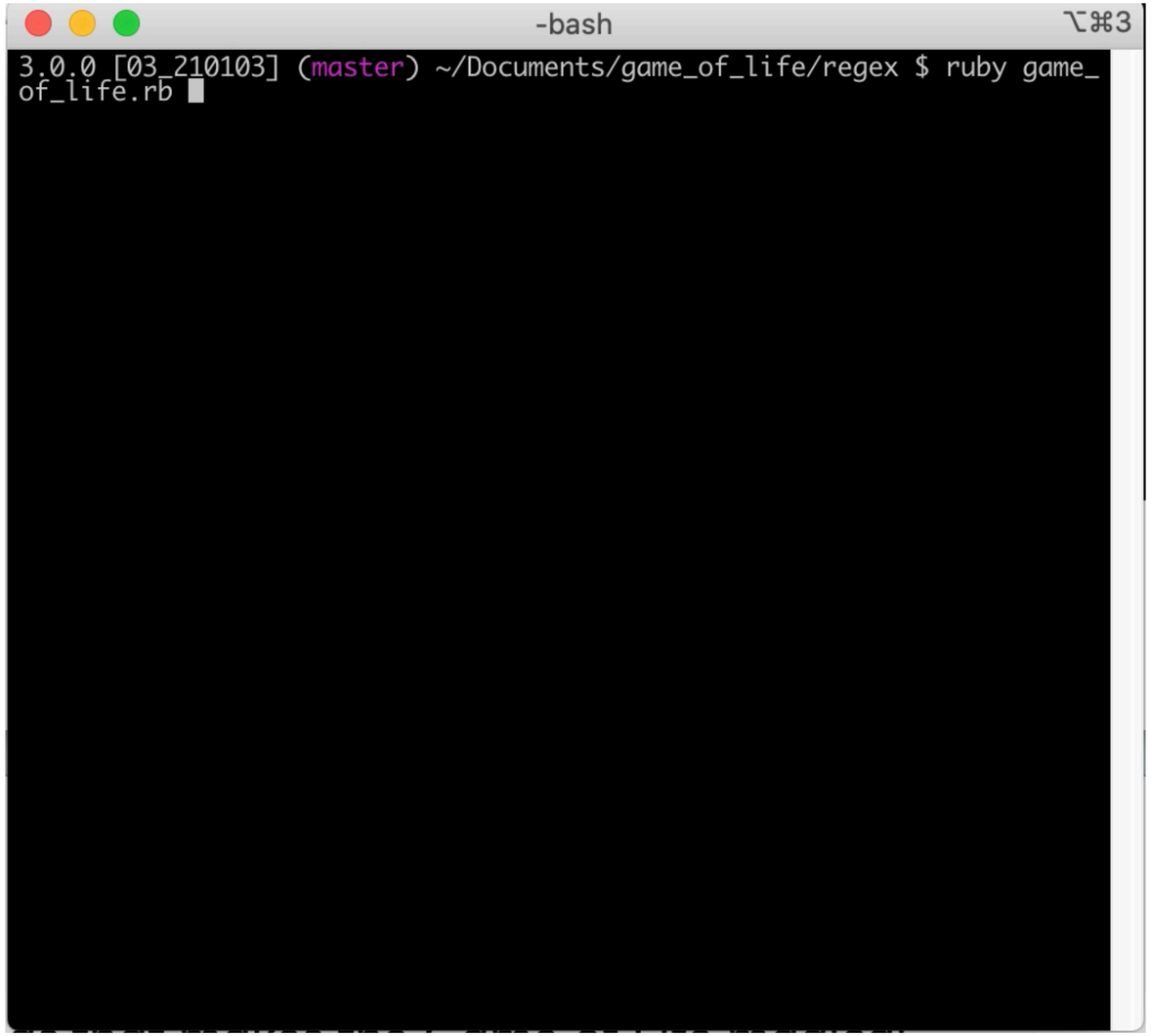
Represent Find Replace

```
while true
    board = board.gsub(regex) { |m| m == "0" ? "1" : "0" }
    board = board.gsub(regex, '\k<replace>')
end
```

Look ma, no Ruby!

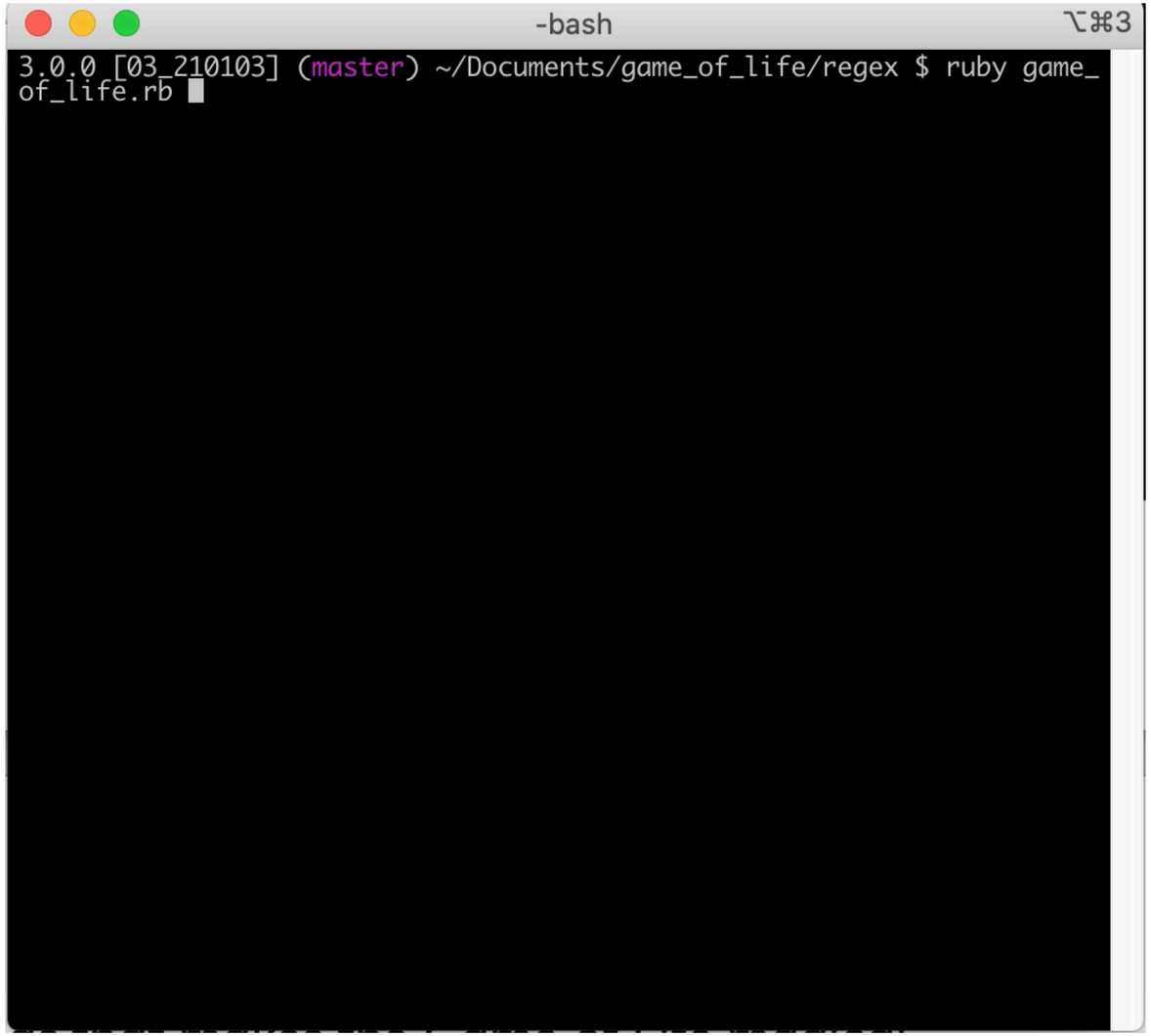
The Dream!

1 2 3 4 Represent Find Replace



The Dream!

1 2 3 4 Represent Find Replace



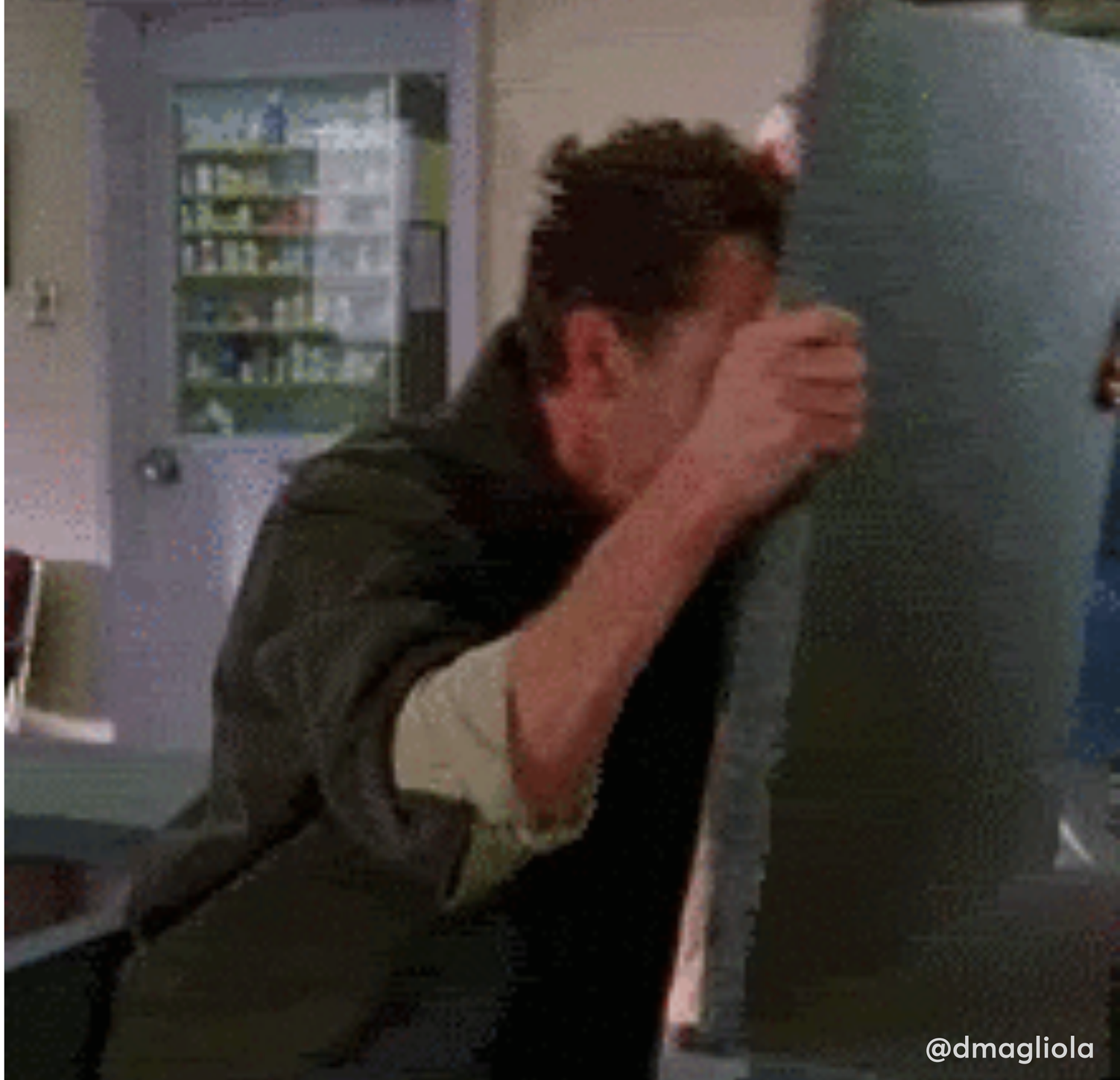

Except!
@dmagliola


Except!
@dmagliola

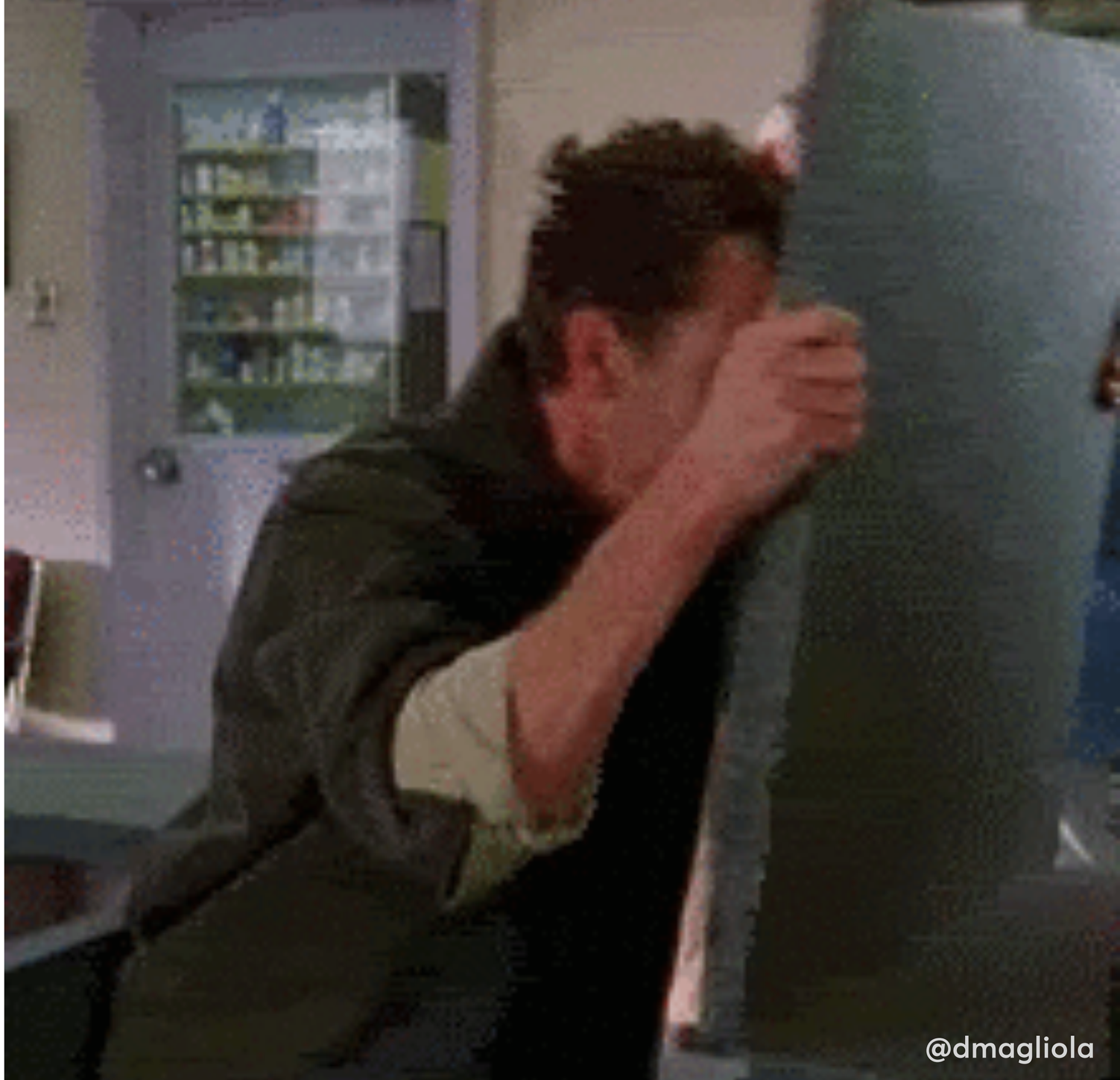
Except!

1	1	1
1	1	1
1	1	1

Except!



Except!



Except!

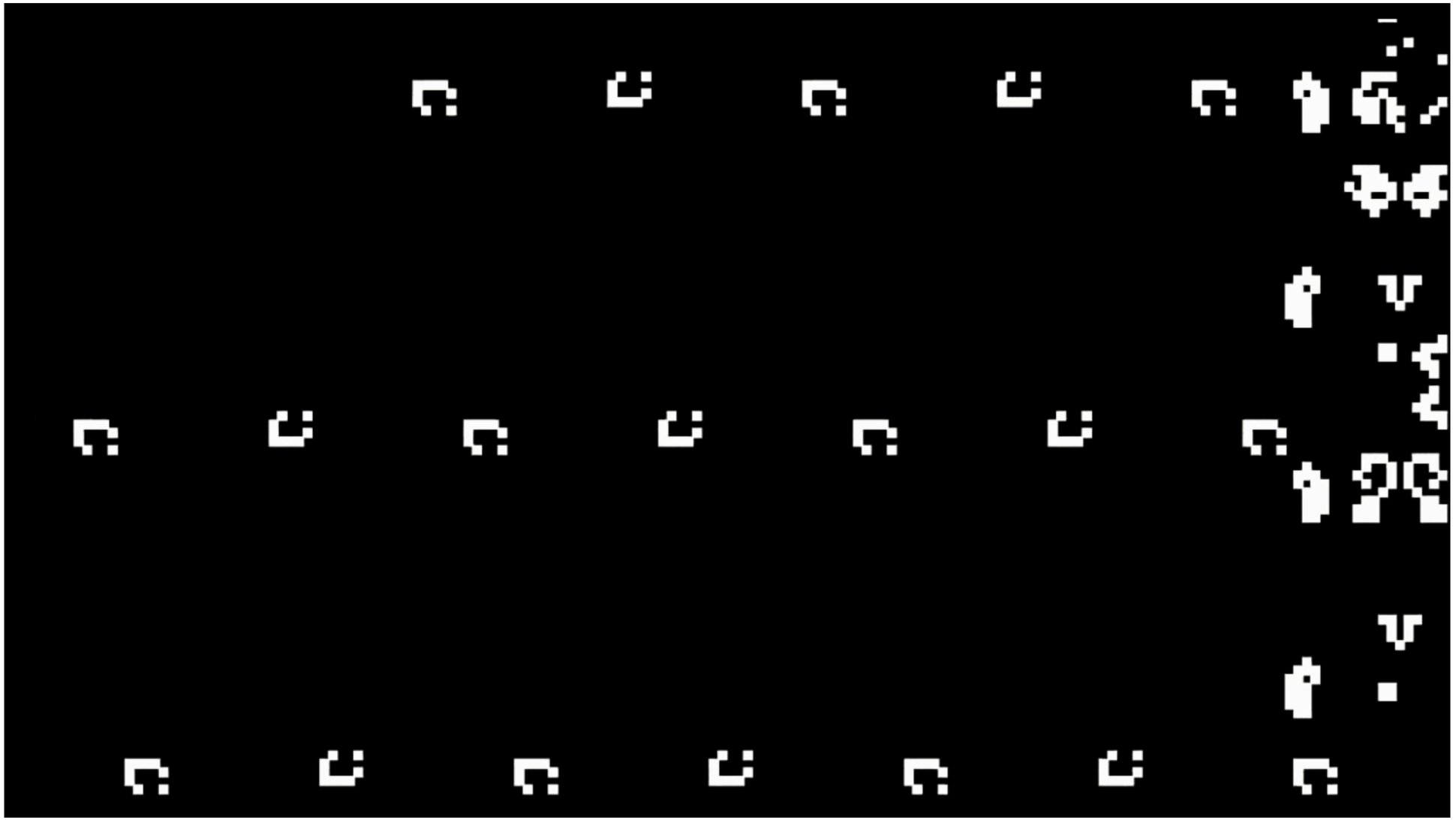
Can you implement the game of life, with just a Regex?

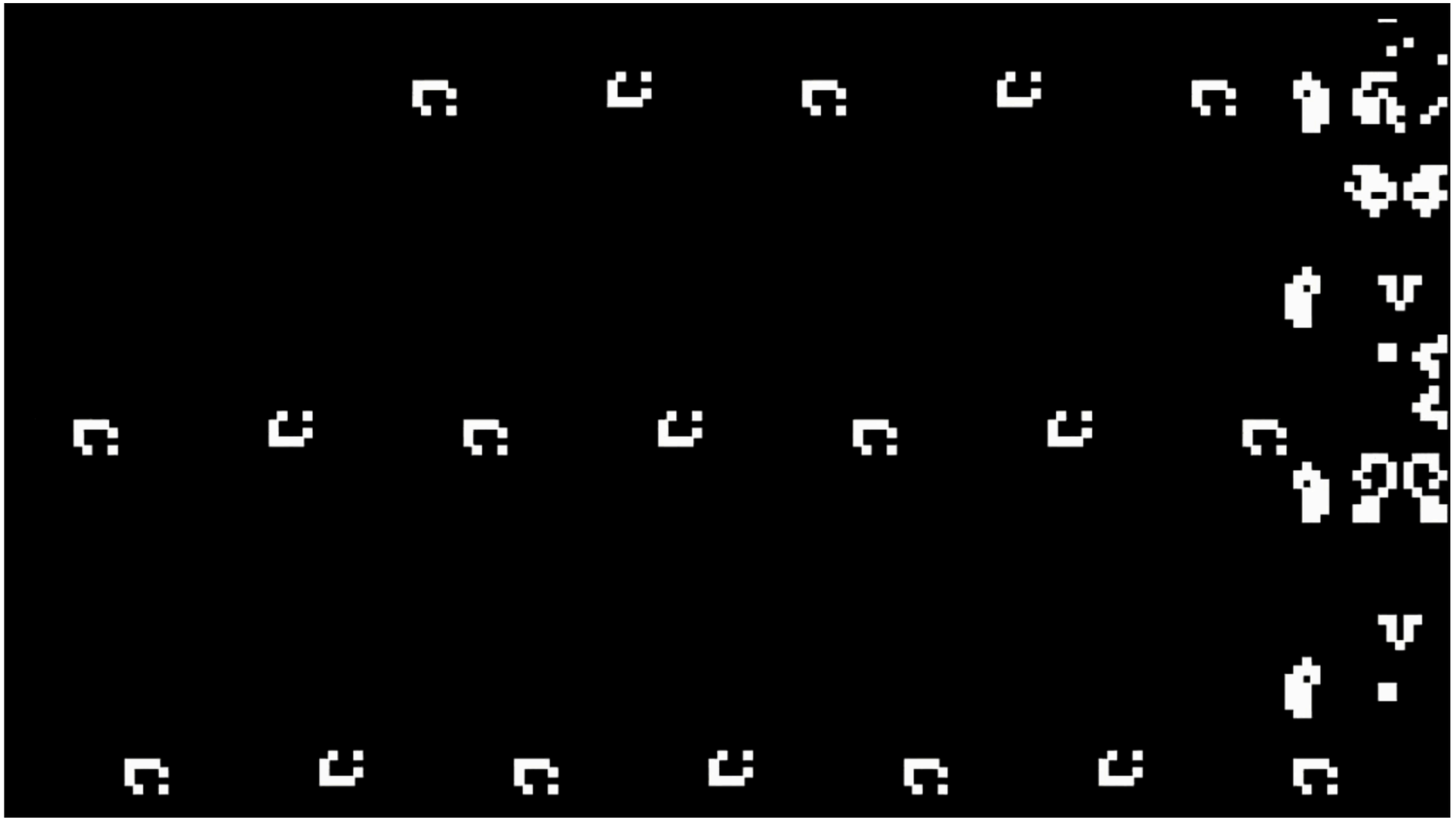
No, but seriously,

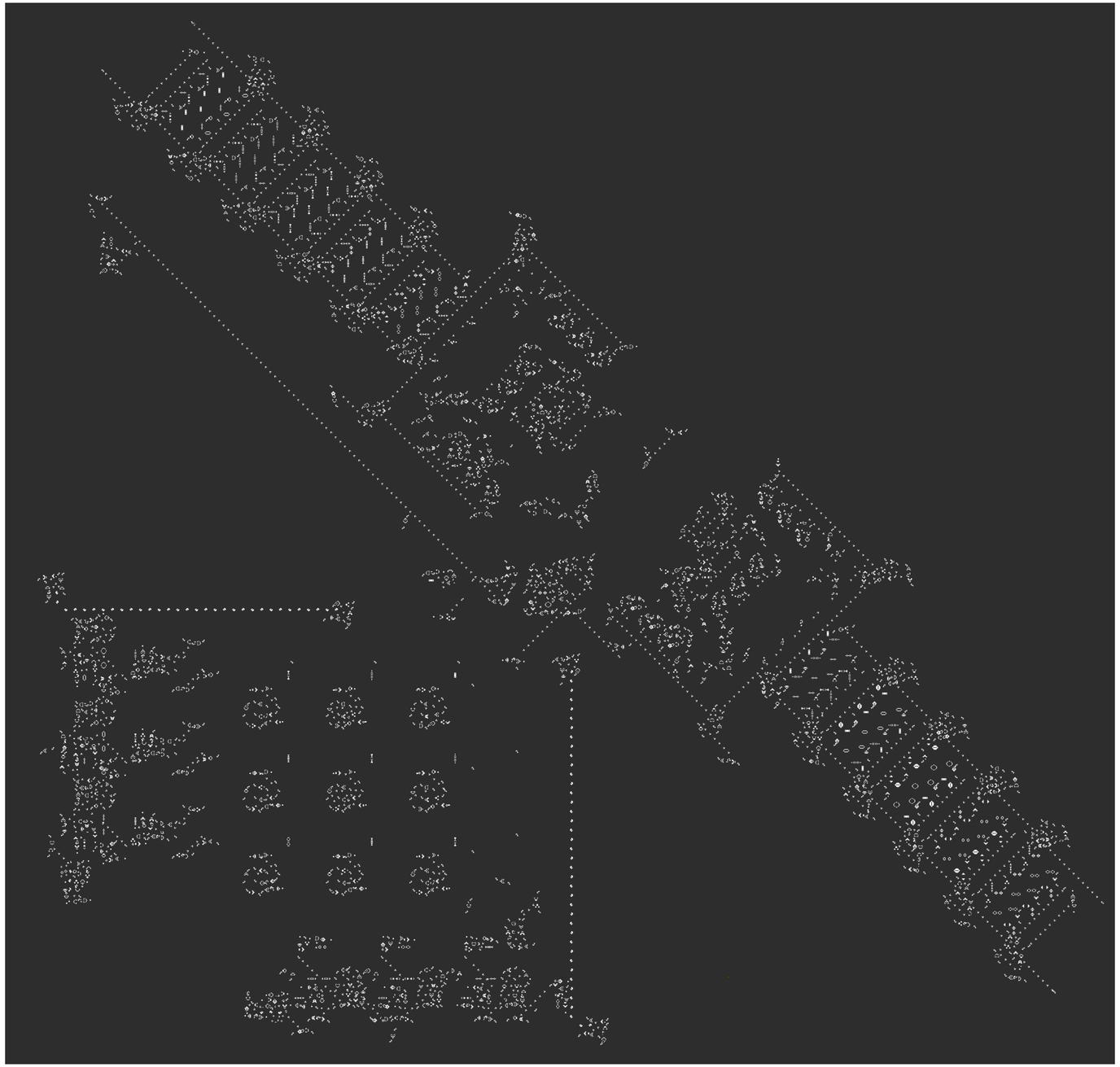
WHY?

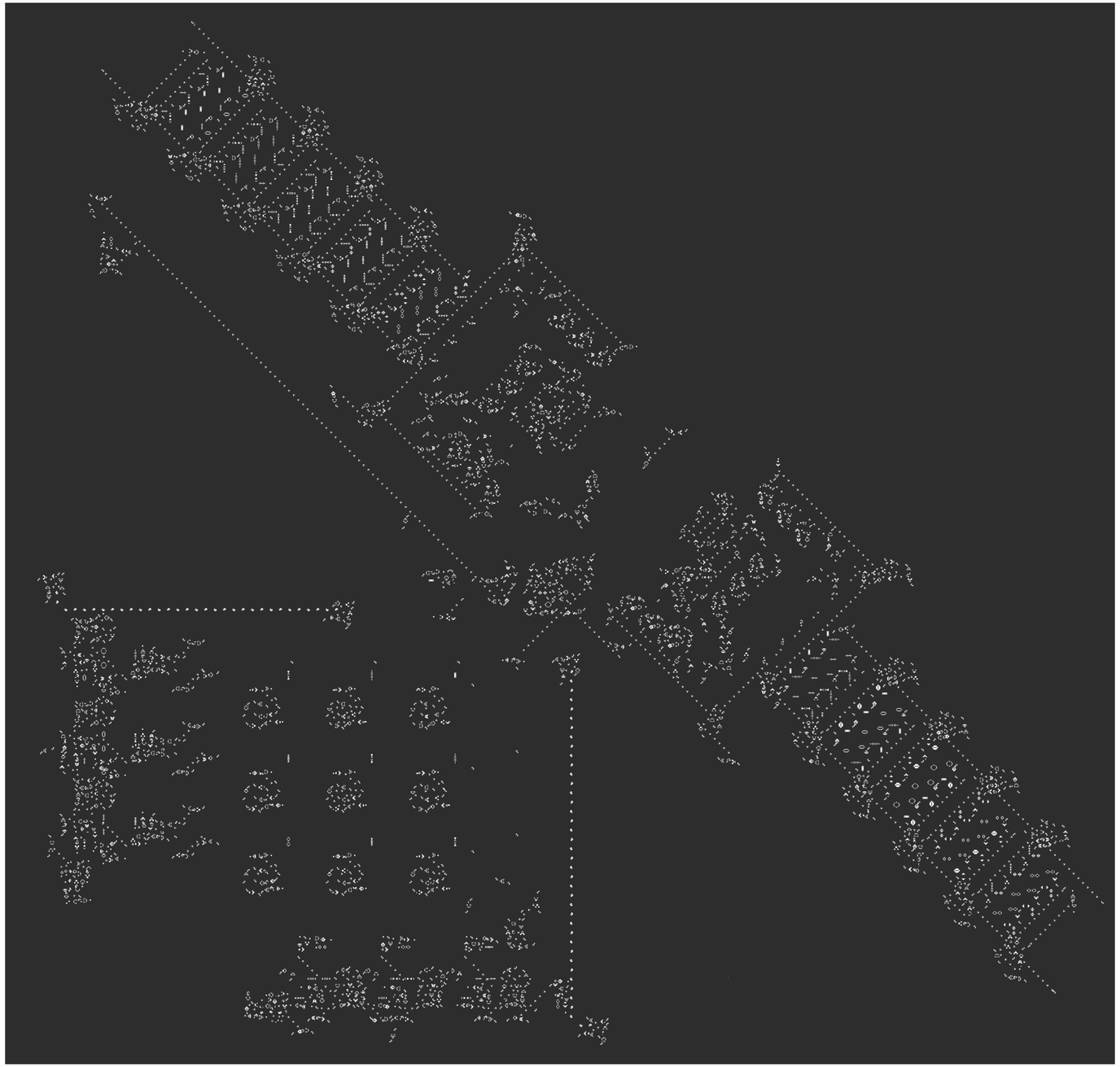
Can you implement the game of life, with just a Regex?

```
alive =
  neighbour_count == 3 ||
  neighbour_count == 2 && cell.alive?
```









Can you implement the Game of Life,

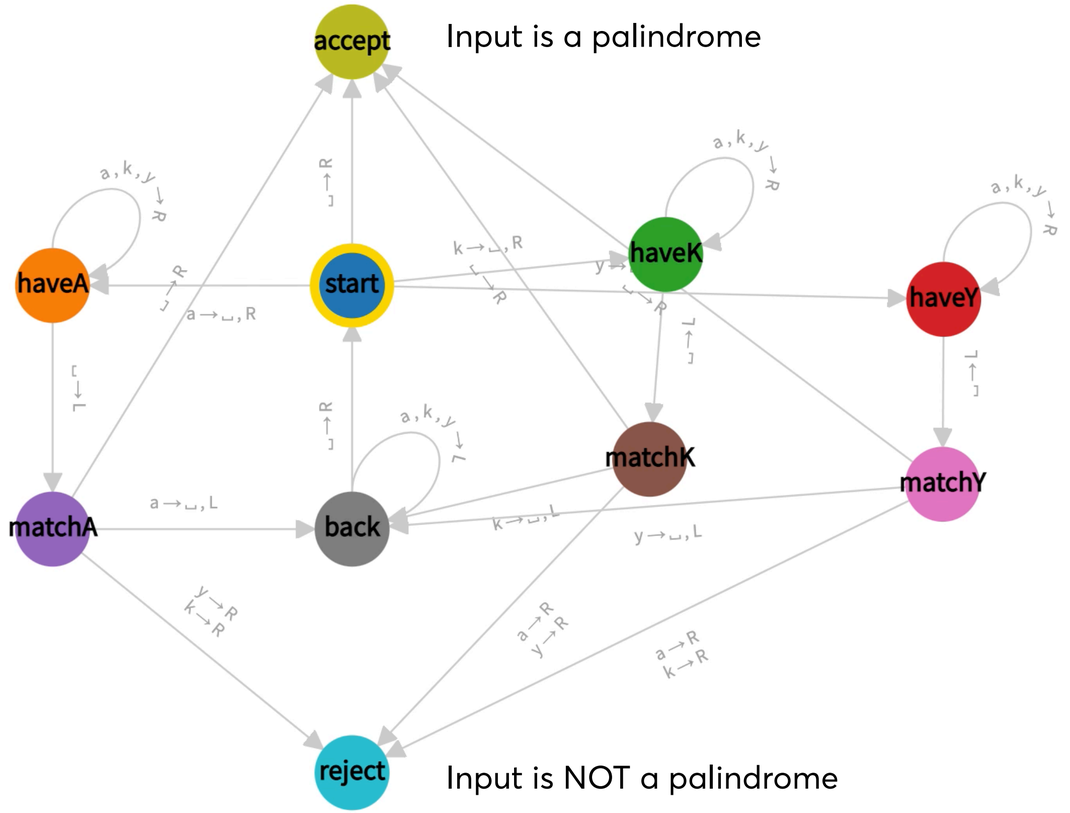
with just a Regex?

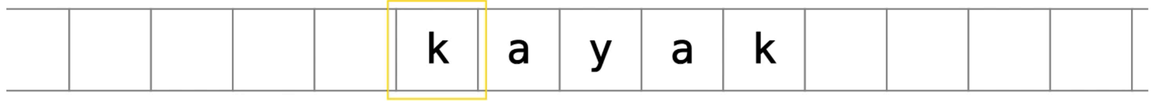
Can you implement ~~the Game of Life~~ a Turing machine

with just a Regex?

Turing Machine Refresher

Palindrome Detector

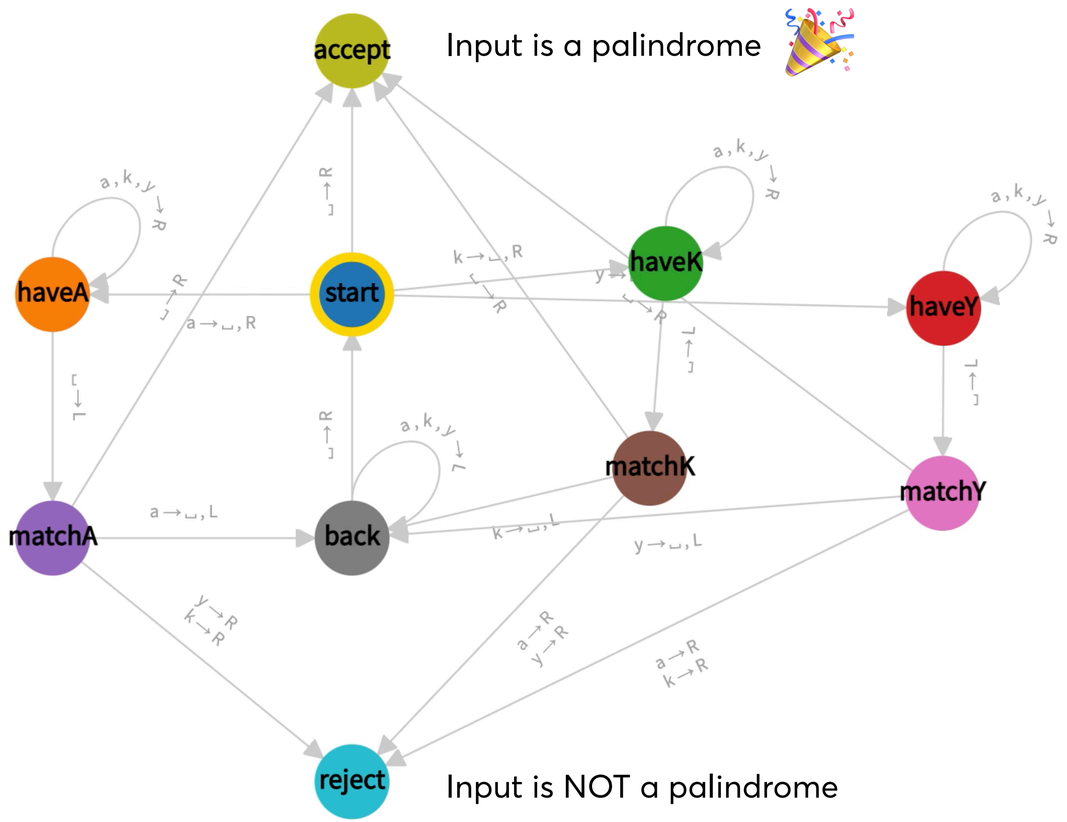




https://turingmachine.io/

Turing Machine Refresher

Palindrome Detector



https://turingmachine.io/

Regex Turing Machine

🔢 Represent 🔍 Find 🖍️ Replace

```
_____________________#Q0Q1Q2Q3Q10Q20Q30Q99:B01CXY_-Q0:B#0010110000011B_____________________
```

Regex Turing Machine

🔢 Represent 🔍 Find 🖍️ Replace

```
_____________________#Q0Q1Q2Q3Q10Q20Q30Q99:B01CXY_-Q0 B#0010110000011B_____________________
```

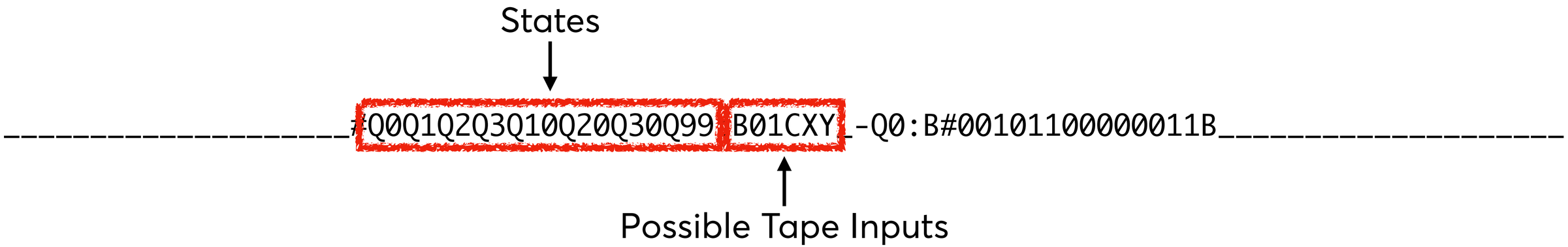
Regex Turing Machine

🔢 Represent 🔍 Find 🖍️ Replace

```
____________________#Q0Q1Q2Q3Q10Q20Q30Q99:B01CXY_-Q0:B#0010110000011B____________________
```

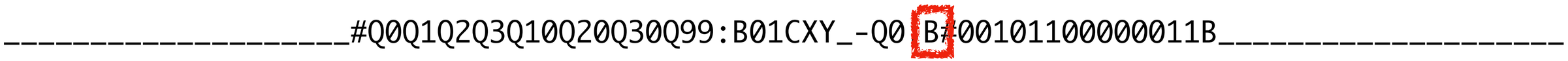
Regex Turing Machine

🔢 Represent 🔍 Find 🖍️ Replace



Regex Turing Machine

🔢 Represent 🔍 Find 🖍️ Replace



Regex Turing Machine

🔢 Represent 🔍 Find 🖍️ Replace

```
____________________#Q0Q1Q2Q3Q10Q20Q30Q99:B01CXY_-Q0 B#0010110000011B____________________
```

Regex Turing Machine

```
DUPLICATE_BINARY_STRING = [
    "Q0:B/B->R:Q1", # Q0: Skip the first B

    "Q1:0/0->R:Q1", # Q1: Find a B, replace with a C, move to Q2
    "Q1:1/1->R:Q1",
    "Q1:B/C->L:Q2",

    "Q2:0/0->L:Q2", # Q2: Rewind until find the initial B, an X or a Y
    "Q2:1/1->L:Q2",
    "Q2:C/C->L:Q2",
    "Q2:B/B->R:Q3",
    "Q2:X/X->R:Q3",
    "Q2:Y/Y->R:Q3",

    "Q3:0/X->R:Q10", # Replace a 0 with X, and move to Q3 (Q3 copies a 0 at the end)
    "Q3:1/Y->R:Q20", # Replace a 1 with Y, and move to Q?? (Q?? copies a 1 at the end)
    "Q3:C/B->L:Q30", # If we find a C, we're done copying, re-replace first number with 0's a
    "Q10:0/0->R:Q10", # Move right until end of tape
    "Q10:1/1->R:Q10",
    "Q10:C/C->R:Q10",
```

Regex Turing Machine

```
DUPLICATE_BINARY_STRING = [
    "Q0:B/B->R:Q1", # Q0: Skip the first B

    "Q1:0/0->R:Q1", # Q1: Find a B, replace with a C, move to Q2
    "Q1:1/1->R:Q1",
    "Q1:B/C->L:Q2",

    "Q2:0/0->L:Q2", # Q2: Rewind until find the initial B, an X or a Y
    "Q2:1/1->L:Q2",
    "Q2:C/C->L:Q2",
    "Q2:B/B->R:Q3",
    "Q2:X/X->R:Q3",
    "Q2:Y/Y->R:Q3",

    "Q3:0/X->R:Q10", # Replace a 0 with X, and move to Q3 (Q3 copies a 0 at the end)
    "Q3:1/Y->R:Q20", # Replace a 1 with Y, and move to Q?? (Q?? copies a 1 at the end)
    "Q3:C/B->L:Q30", # If we find a C, we're done copying, re-replace first number with 0's a
    "Q10:0/0->R:Q10", # Move right until end of tape
    "Q10:1/1->R:Q10",
    "Q10:C/C->R:Q10"
```

Regex Turing Machine

Match:

```
/(#Q0(?<new_state>Q1)Q2Q3Q10Q20Q30Q99:(?<new_left>B)01CXY_-Q0:B#(?<new_cursor>\w))|
(#Q0(?<new_state>Q1)Q2Q3Q10Q20Q30Q99:B(?<new_left>0)1CXY_-Q1:0#(?<new_cursor>\w))|
(#Q0(?<new_state>Q1)Q2Q3Q10Q20Q30Q99:B0(?<new_left>1)CXY_-Q1:1#(?<new_cursor>\w))|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B01(?<new_right>C)XY_-Q1:B#)|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B(?<new_right>0)1CXY_-Q2:0#)|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B0(?<new_right>1)CXY_-Q2:1#)|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B01(?<new_right>C)XY_-Q2:C#)|
(#Q0Q1Q2(?<new_state>Q3)Q10Q20Q30Q99:(?<new_left>B)01CXY_-Q2:B#(?<new_cursor>\w))|
(#Q0Q1Q2(?<new_state>Q3)Q10Q20Q30Q99:B01C(?<new_left>X)Y_-Q2:X#(?<new_cursor>\w))|
(#Q0Q1Q2(?<new_state>Q3)Q10Q20Q30Q99:B01CX(?<new_left>Y)_-Q2:Y#(?<new_cursor>\w))|
(#Q0Q1Q2Q3(?<new_state>Q10)Q20Q30Q99:B01C(?<new_left>X)Y_-Q3:0#(?<new_cursor>\w))|
(#Q0Q1Q2Q3Q10(?<new_state>Q20)Q30Q99:B01CX(?<new_left>Y)_-Q3:1#(?<new_cursor>\w))|
((?<new_cursor>\w)#Q0Q1Q2Q3Q10Q20(?<new_state>Q30)Q99:(?<new_right>B)01CXY_-Q3:C#)|
(#Q0Q1Q2Q3(?<new_state>Q10)Q20Q30Q99:B(?<new_left>0)1CXY_-Q10:0#(?<new_cursor>\w))|
(#Q0Q1Q2Q3(?<new_state>Q10)Q20Q30Q99:B0(?<new_left>1)CXY_-Q10:1#(?<new_cursor>\w))|
(#Q0Q1Q2Q3(?<new_state>Q10)Q20Q30Q99:B01(?<new_left>C)XY_-Q10:C#(?<new_cursor>\w))|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B(?<new_right>0)1CXY_-Q10:_#)|
(#Q0Q1Q2Q3Q10(?<new_state>Q20)Q30Q99:B(?<new_left>0)1CXY_-Q20:0#(?<new_cursor>\w))|
(#Q0Q1Q2Q3Q10(?<new_state>Q20)Q30Q99:B0(?<new_left>1)CXY_-Q20:1#(?<new_cursor>\w))|
(#Q0Q1Q2Q3Q10(?<new_state>Q20)Q30Q99:B01(?<new_left>C)XY_-Q20:C#(?<new_cursor>\w))|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B0(?<new_right>1)CXY_-Q20:_#)|
((?<new_cursor>\w)#Q0Q1Q2Q3Q10Q20(?<new_state>Q30)Q99:B(?<new_right>0)1CXY_-Q30:X#)|
((?<new_cursor>\w)#Q0Q1Q2Q3Q10Q20(?<new_state>Q30)Q99:B0(?<new_right>1)CXY_-Q30:Y#)|
((?<new_cursor>\w)#Q0Q1Q2Q3Q10Q20Q30(?<new_state>Q99):(?<new_right>B)01CXY_-Q30:B#)/
```

Replace with:

```
\k<new_left>
#Q0Q1Q2Q3Q10Q20Q30Q99:B01CXY_-
\k<new_state>:
\k<new_cursor>#\k<new_right>
```

Regex Turing Machine

Match:

```
/(#Q0(?<new_state>Q1)Q2Q3Q10Q20Q30Q99:(?<new_left>B)01CXY_-Q0:B#(?<new_cursor>\w))|
(#Q0(?<new_state>Q1)Q2Q3Q10Q20Q30Q99:B(?<new_left>0)1CXY_-Q1:0#(?<new_cursor>\w))|
(#Q0(?<new_state>Q1)Q2Q3Q10Q20Q30Q99:B0(?<new_left>1)CXY_-Q1:1#(?<new_cursor>\w))|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B01(?<new_right>C)XY_-Q1:B#)|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B(?<new_right>0)1CXY_-Q2:0#)|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B0(?<new_right>1)CXY_-Q2:1#)|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B01(?<new_right>C)XY_-Q2:C#)|
(#Q0Q1Q2(?<new_state>Q3)Q10Q20Q30Q99:(?<new_left>B)01CXY_-Q2:B#(?<new_cursor>\w))|
(#Q0Q1Q2(?<new_state>Q3)Q10Q20Q30Q99:B01C(?<new_left>X)Y_-Q2:X#(?<new_cursor>\w))|
(#Q0Q1Q2(?<new_state>Q3)Q10Q20Q30Q99:B01CX(?<new_left>Y)_-Q2:Y#(?<new_cursor>\w))|
(#Q0Q1Q2Q3(?<new_state>Q10)Q20Q30Q99:B01C(?<new_left>X)Y_-Q3:0#(?<new_cursor>\w))|
(#Q0Q1Q2Q3Q10(?<new_state>Q20)Q30Q99:B01CX(?<new_left>Y)_-Q3:1#(?<new_cursor>\w))|
((?<new_cursor>\w)#Q0Q1Q2Q3Q10Q20(?<new_state>Q30)Q99:(?<new_right>B)01CXY_-Q3:C#)|
(#Q0Q1Q2Q3(?<new_state>Q10)Q20Q30Q99:B(?<new_left>0)1CXY_-Q10:0#(?<new_cursor>\w))|
(#Q0Q1Q2Q3(?<new_state>Q10)Q20Q30Q99:B0(?<new_left>1)CXY_-Q10:1#(?<new_cursor>\w))|
(#Q0Q1Q2Q3(?<new_state>Q10)Q20Q30Q99:B01(?<new_left>C)XY_-Q10:C#(?<new_cursor>\w))|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B(?<new_right>0)1CXY_-Q10:_#)|
(#Q0Q1Q2Q3Q10(?<new_state>Q20)Q30Q99:B(?<new_left>0)1CXY_-Q20:0#(?<new_cursor>\w))|
(#Q0Q1Q2Q3Q10(?<new_state>Q20)Q30Q99:B0(?<new_left>1)CXY_-Q20:1#(?<new_cursor>\w))|
(#Q0Q1Q2Q3Q10(?<new_state>Q20)Q30Q99:B01(?<new_left>C)XY_-Q20:C#(?<new_cursor>\w))|
((?<new_cursor>\w)#Q0Q1(?<new_state>Q2)Q3Q10Q20Q30Q99:B0(?<new_right>1)CXY_-Q20:_#)|
((?<new_cursor>\w)#Q0Q1Q2Q3Q10Q20(?<new_state>Q30)Q99:B(?<new_right>0)1CXY_-Q30:X#)|
((?<new_cursor>\w)#Q0Q1Q2Q3Q10Q20(?<new_state>Q30)Q99:B0(?<new_right>1)CXY_-Q30:Y#)|
((?<new_cursor>\w)#Q0Q1Q2Q3Q10Q20Q30(?<new_state>Q99):(?<new_right>B)01CXY_-Q30:B#)/
```

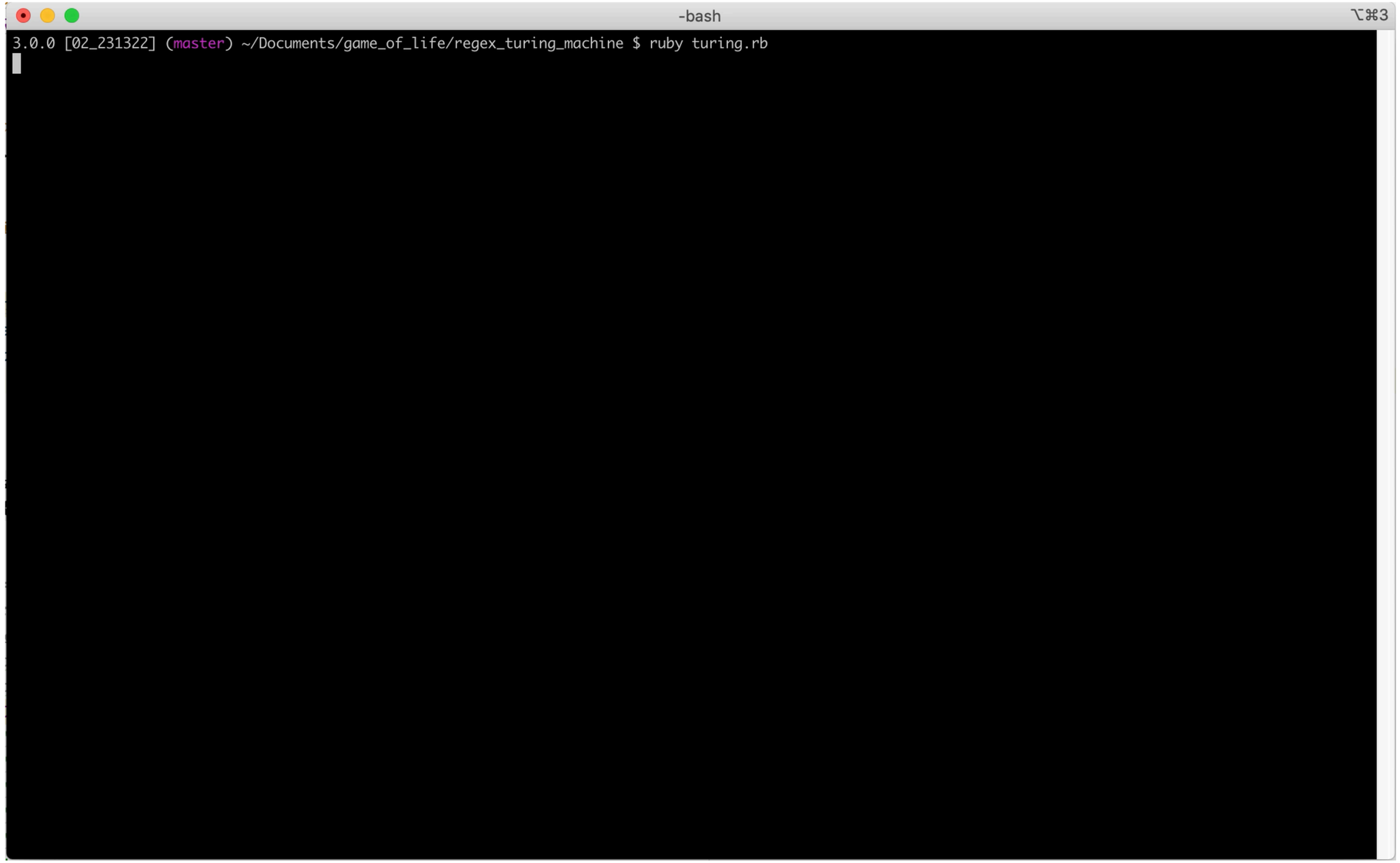
Replace with:

```
\k<new_left>
#Q0Q1Q2Q3Q10Q20Q30Q99:B01CXY_-
\k<new_state>:
\k<new_cursor>#\k<new_right>
```

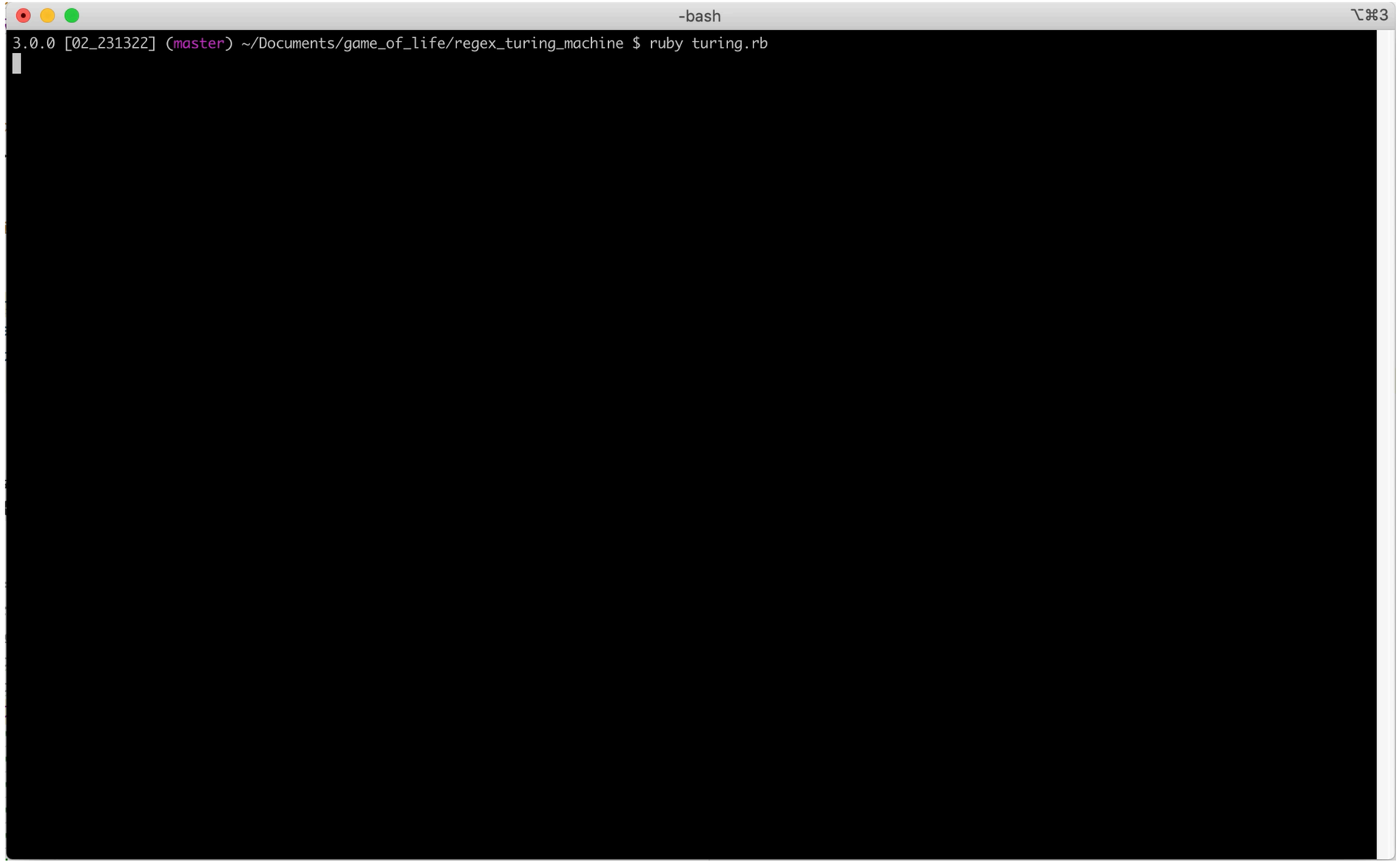
Regex Turing Machine

```
while true
  tape.gsub!(regex, replace_regex)
end
```

Regex Turing Machine



Regex Turing Machine



Are Regex Turing Complete?

Very, very

NO!

- They cannot loop, and you really need to loop
- PCRE* gives you much more than "a Regular Language" does

*PCRE: Perl Compatible Regular Expressions
(the regex you use every day)

Kinda

YES!*‡ , though?

* if you run them in a loop
‡ and kinda depends what exactly you mean by "a Regex"

Thank you

Do Regex dream of Turing Completeness?

Daniel Magliola

@dmagliola

Code, slides and more info at: https://bit.ly/regexgol