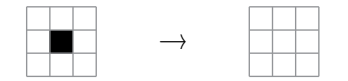
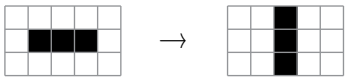
http://www.libe.nara-k.ac.jp/~yano/biseki1_2018/life_game/20180626.html にて、Python 版および JavaScript 版のライフゲームを置いています。JavaScript 版は Python 環境がなくても、ブラウザ上で動作するので、ぜひ試してみてください。参考書: “ライフゲイムの宇宙”、ウィリアム・パウンドストーン著 (有澤誠 訳)

(小課題第 7 回裏面) 離散的な時間の流れの中の生命モデルとして、イギリスの数学者 J. H. Conway が編み出したのがライフゲーム **(Conway's Game of Life)** である。マスで区切られたライフゲームの世界では、各マス目 (セルという) の状態は生と死のいずれかで、生ならば黒いセル、死ならば白いセルで表現される。そして、以下のルールに従って次のステップ (時刻) のセルの生き死にが決まる。

rule 1. 注目している黒いセルの周り (全部で 8 個のセルがある) にちょうど 2 個または 3 個の黒いセルがあれば、そのセルは黒のままで、そうでなければ白くなる。

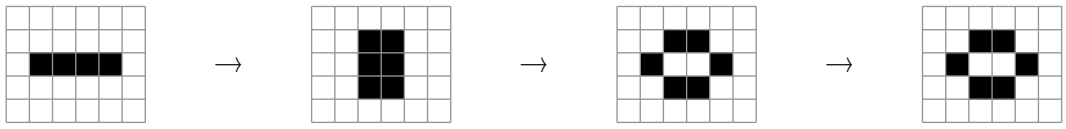


rule 2. 注目している白いセルの周りにちょうど 3 個の黒いセルがあれば、そのセルは黒になり、そうでなければ白のままになる。

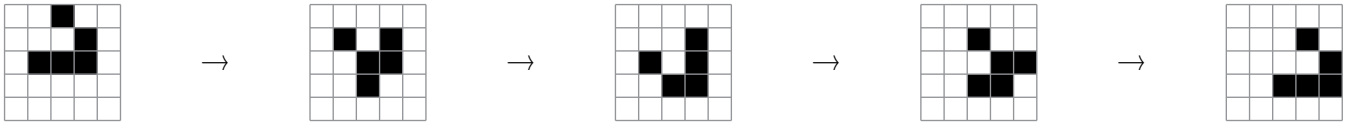


(このルールは、適度な環境では生命が生まれ、過密や過疎では、生命が死んでしまう様子を表している。)

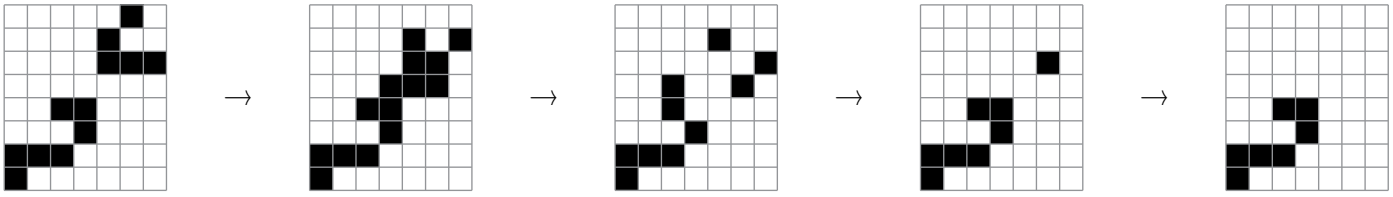
(1) 次の生命の時刻毎の変化を調べよ。



glider



glider と eater



(2) 各セルの生き死にを判定する関数のプログラムを設計せよ。→ **life 関数**

例えば、生命 $G = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ が与えられたとする。各 (m, n)-セルに対し、その周りのセルの値を集めた配列 **nbd** を考え、1 が何個あるか数える。G の端っこで **nbd** の処理を気にする必要があるが、この行列の周りを 0 で埋めた行列 $\mathrm{ext}G = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ に対して **nbd** を考えればよい。

(3) 与えられた生命の時刻毎の変化をシミュレートし、ディスプレイするプログラムを書いてみよ。

```
#!/usr/bin/python3
# coding: UTF-8

import matplotlib.pyplot as plt
import numpy as np
import life_game_display as lgdisplay


class LifeGame:

    def __init__(self, matrix, universe_level=2):
        self.lives = matrix # 生命モデル(行列として扱う)
        self.universe_level = universe_level # 生命モデルの世界の大きさのレベル(>0)
        self.univrow, self.univcol = self.set_universe_size()
        self.universe = self.put_in_universe() # 生命モデルのいる世界(行列)
        self.ext = self.extension() # 生命モデルの拡大

    def set_universe_size(self):
        row = 10 * 2**(self.universe_level - 1)
        col = 2 * row
        # print('univrow, univcol: ', row, col)
        return row, col

    def put_in_universe(self):
```

```
        """
        生命モデル(lives)を中央に入れた世界(行列)を返す
        """
        row, col = self.lives.shape
        # print('row, col: ', row, col)
        universe = np.zeros([self.univrow, self.univcol], dtype='int8')
        pos_col = int((self.univcol - col) / 2)
        pos_row = int((self.univrow - row) / 2)
        # print('pos_row, pos_col: ', pos_row, pos_col)
        for i in range(row):
            for j in range(col):
                universe[i + pos_row, j + pos_col] = self.lives[i, j]
        return universe

    def extension(self):
        """
        世界(universe)の行列の外側を0で埋めた、拡大された行列を返す。
        この行列で各セルの生き死にを判定しやすくする。
        """
        ext = np.zeros([self.univrow + 2, self.univcol + 2], dtype='int8')
        for i in range(self.univrow):
            for j in range(self.univcol):
                ext[i + 1, j + 1] = self.universe[i, j]
        return ext

    def life(self, m, n):
        """
        (m,n)の位置のセルの生き死にを判定する。生 -> 1 死 -> 0
        """

        # (m,n)のセルの周りの値を調べる
        nbd = [
            self.ext[m + i + 1, n + j + 1] for i in [-1, 0, 1]
            for j in [-1, 0, 1]
        ]
        num = nbd.count(1) - self.universe[m, n] # 周りの生存マスの数
        if self.universe[m, n] == 0:
            if num == 3:
                return 1
            else:
                return 0
        else:
            if num == 2 or num == 3:
                return 1
            else:
                return 0

    def next(self): # 次の世代の世界を返す
        generation = np.zeros([self.univrow, self.univcol], dtype='int8')
        for i in range(self.univrow):
            for j in range(self.univcol):
                generation[i, j] = self.life(i, j)
        return generation

    def display(self):
        lgdisplay.display(self.universe, self.universe_level)


class IteratorLifeGame(LifeGame):

    def __init__(self, matrix, universe_level=2, max=10):
        super().__init__(matrix, universe_level)
        self.count = 1
        self.max_count = max

    def __iter__(self):
        for i in range(self.max_count):
            tmpLG = LifeGame(self.universe, self.universe_level)
            yield tmpLG
            # yield self.universe

            self.universe = self.next()
            self.ext = self.extension()
            self.count += 1


if __name__ == '__main__':

    G = np.array([[0, 0, 0], [1, 1, 1], [0, 0, 0]])
    LG = LifeGame(G, 1)
    print(LG.universe)
    print(LG.ext)
    print(LG.next())

    print('--------next-------')
    G = np.zeros([5, 6], dtype='int8')
    for i in range(1, 5):
        G[2, i] = 1
    # G=[[0 0 0 0 0 0]
    # [0 0 0 0 0 0]
    # [0 1 1 1 1 0]
    # [0 0 0 0 0 0]
    # [0 0 0 0 0 0]]

    LG = LifeGame(G, 1)
    print(LG.universe)
    print(LG.ext)
    print(LG.next())

    iterLG = IteratorLifeGame(G, 1, 4)
    for i, LG in enumerate(iterLG):
        print("generation␣=␣", i + 1, "␣:")
        print(LG.universe)
        LG.display()
```