

Automaty Komórkowe

Wykład 9

<https://github.com/houp/ca-class>

Witold Bołt, 08.05.2024

Poprzednio omówiliśmy

- **Wykład 1:** Sprawy organizacyjne, motywację do zajmowania się CA, podstawowe pojęcia / definicje / intuicje.
- **Wykład 2:** Definicja (formalna) i podstawowe fakty o ECA. Reprezentacja Wolframa.
- **Wykład 3:** Symetrie w zbiorze ECA, relacje do ogólnej teorii układów dynamicznych, własności CA/ECA.
- **Wykład 4:** Alternatywne reprezentacje reguły lokalnej (wielomiany, wyrażenia logiczne), problem klasyfikacji gęstości (DCP).
- **Dwa tygodnie przerwy** 😞
- **Wykład 5 (zdalny):** Algorytmy ewolucyjne - poszukiwanie automatów komórkowych o określonych własnościach
- **Wykład 6:** Stochastyczne automaty komórkowe - SCAs, pLUT, α -ACAs, Diploid CAs, stochastic mixture, dekompozycja pLUT
- **Wykład 7:** Afiniczne Ciągłe Automaty Komórkowe - wielomiany, cLUT, relaxed DCP + bonus - praca w IT w Trójmieście (i nie tylko)
- **Wykład 8:** Identyfikacja Deterministycznych Automatów Komórkowych
- **Wykład 9:** Identyfikacja Stochastycznych Automatów Komórkowych

Co będzie dalej*

- (15.05) **Wykład 10:** Dwu-wymiarowe Automaty Komórkowe / Reguła Life i Life-like / Totalistyczne i Zewnętrzne-Totalistyczne Automaty Komórkowe (totalistic & outer-totalistic CAs)
- (29.05) **Wykład 11:** Automaty Komórkowe zachowujące gęstość
- (5.06) **Wykład 12:** Nie-jednородne (non-uniform) Automaty Komórkowe; Zastosowania w modelowaniu ruchu ulicznego
- (12.06) **Wykład 13:** Modele pożaru lasu, rozprzestrzeniania się epidemii, Greenberg–Hastings i podobne modele
- (termin **dodatkowy**) **Wykład 14:** Neural CAs (Neuronowe Automaty Komórkowe)

Totalistyczne Automaty Komórkowe

Definicja

- Niech $f: \{0,1\}^{2r+1} \rightarrow \{0,1\}$ będzie regułą lokalną pewnego automatu komórkowego. Będziemy mówili, że reguła ta definiuje automat komórkowy, który jest:

- totalistyczny** (ang. *totalistic*), jeśli istnieje funkcja $g: \{0,1,\dots,2r+1\} \rightarrow \{0,1\}$ taka, że zachodzi:

$$f(x_{-r}, x_{-r+1}, \dots, x_0, \dots, x_{r-1}, x_r) = g\left(\sum_{i=-r}^r x_i\right),$$

- zewnątrzny-totalistyczny** (ang. *outer-totalistic*), jeśli istnieje funkcja $h: \{0,1\} \times \{0,1,\dots,2r\} \rightarrow \{0,1\}$ taka, że zachodzi:

$$f(x_{-r}, x_{-r+1}, \dots, x_0, \dots, x_{r-1}, x_r) = h\left(x_0, \sum_{i=-r}^{-1} x_i + \sum_{i=1}^r x_i\right)$$

- Naturalnie można z łatwością rozszerzyć tę definicję na większe zbiory stanów i większą liczbę wymiarów.
- Mówiąc po ludzku automat totalistycznych wylicza stan komórki w oparciu o **sumę stanów sąsiedztwa**. Natomiast automatu zewnętrze-totalistyczny bierze pod uwagę sumę stanów sąsiedztwa bez komórki centralnej i osobno stan komórki centralnej.

Co wynika z definicji?

- **Fakt.** Jeśli reguła lokalna f definiuje automat komórkowy totalistyczny to definiuje również automat zewnętrzny-totalistyczny.

Dowód. Skoro f jest totalistyczna, to istnieje g taka, że $f(x_{-r}, \dots, x_r) = g(\sum x_i)$.

Niech $h(x, y) = g(x + y)$. Wtedy oczywiście

$$h(x_0, \sum_{i=-r}^{-1} x_i + \sum_{i=1}^r x_i) = g(\sum_{i=-r}^r x_i) = f(x_{-r}, \dots, x_r).$$

- **Fakt.** Niech $r > 0$ będzie promieniem sąsiedztwa. Istnieje 2^{2r+2} dwu-stanowych, 1-wymiarowych reguł totalistycznych, oraz $2^{2(2r+1)}$ reguł zewnętrznych-totalistycznych.
- **Obserwacja.** Wszystkich reguł jest $2^{2^{2r+1}}$, czyli znacznie więcej!

Co wynika z definicji?

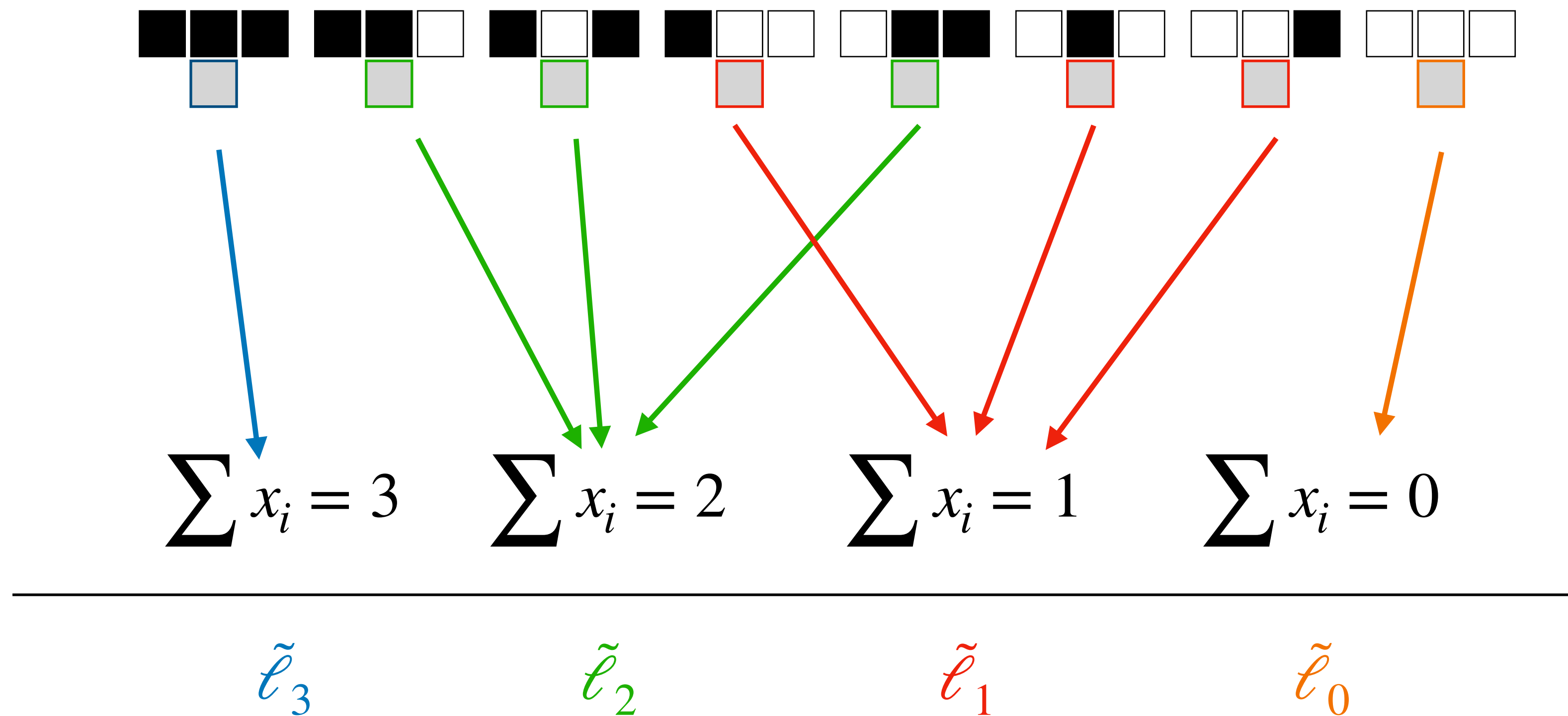
- Reguły totalistic (oraz outer-totalistic) możemy zapisać w formie specjalnej tablicy podobnej do LUT.
- Przykład dla $r = 1$ i reguł totalistic:

$$\begin{array}{cccc} \sum x_i = 3 & \sum x_i = 2 & \sum x_i = 1 & \sum x_i = 0 \\ \hline \tilde{\ell}_3 & \tilde{\ell}_2 & \tilde{\ell}_1 & \tilde{\ell}_0 \end{array}$$

- Podobnie jak w przypadku zwykłego LUT, możemy użyć drugiej wiersz takiej tabelki do ponumerowania reguł.

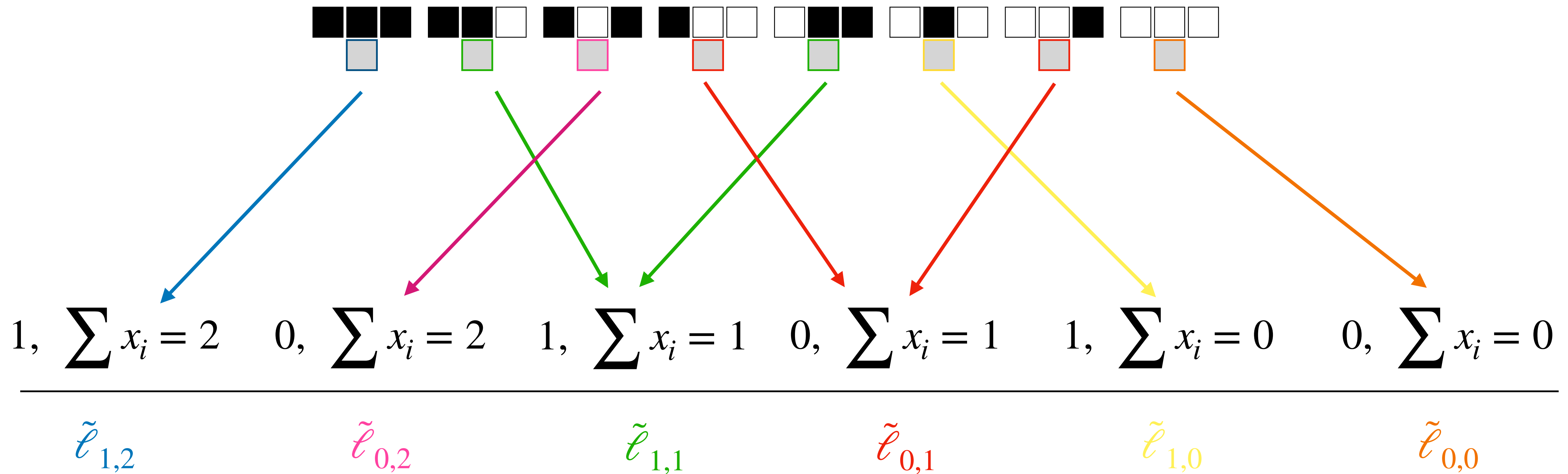
Co wynika z definicji?

- Relacja między “zwykłym” LUT a “LUT totalistycznym”:



Co wynika z definicji?


- Relacja między “zwykłym” LUT a “LUT zewnętrznym-totalistycznym”:




Ćwiczenia dla czytelnika

- Wypisz wszystkie ECA, które są totalistic i outer-totalistic.
- Wypisz wszystkie ECA, które nie są outer-totalistic.
- Sprawdź czy własności bycia totalistic / outer-totalistic wpływają na “complexity” wyrażone przez klasyfikację Wolframa, lub podobne klasyfikacje. To znaczy, czy reguły outer-totalistic są mniej / bardziej złożone od pozostałych? Czy ta własność raczej mówi o tym, że te reguły są “prostsze” czy przeciwnie bardziej złożone?

Polecam!






Physica D: Nonlinear Phenomena
Volume 432, April 2022, 133074








Review



Progress, gaps and obstacles in the classification of cellular automata

[Milan Vispoel](#)  , [Aisling J. Daly](#), [Jan M. Baetens](#)

[Show more](#) 

 Add to Mendeley  Share  Cite

<https://doi.org/10.1016/j.physd.2021.133074>  [Get rights and content](#) 

Under a Creative Commons [license](#)   open access

<https://doi.org/10.1016/j.physd.2021.133074>

Uwagi

- SCA oraz ACCA też mogą być totalistic / outer-totalistic. Definicja jest właściwie identyczna / analogiczna (jedynie dziedzina funkcji g oraz h musi być odpowiednio dobrana w przypadku ACCA).
- Automaty totalistic i outer-totalistic mają istotne zastosowania, bo własność ta dobrze odpowiada własnościom znanym z **fizyki** i **chemii**.
- Dzięki temu, że tych automatów jest znacznie mniej (niż wszystkich) i dzięki temu, że można je sensownie numerować - sensowne jest badanie całej klasy np. automatów totalistycznych o zadanym promieniu.
- Można napisać GA, który poszukuje automaty totalistyczne o zadanych własnościach.

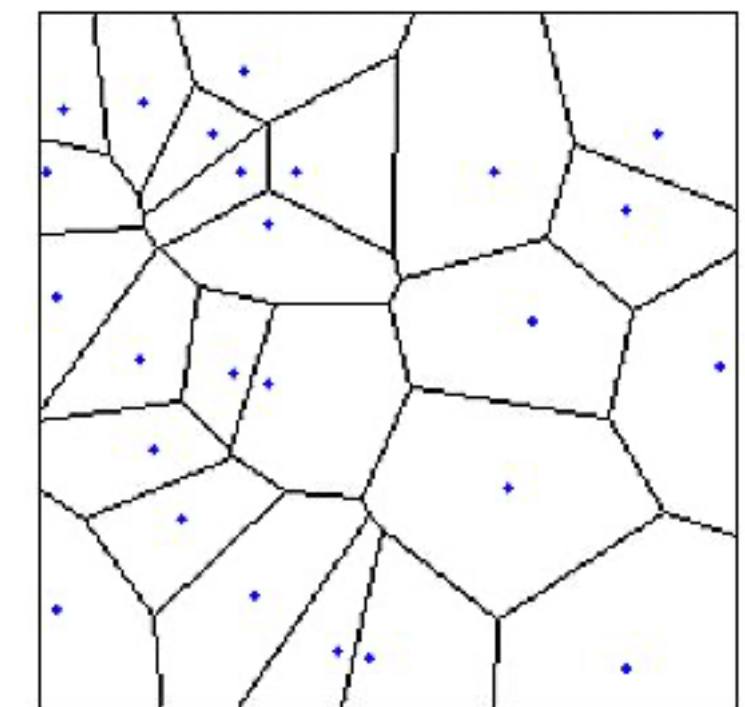
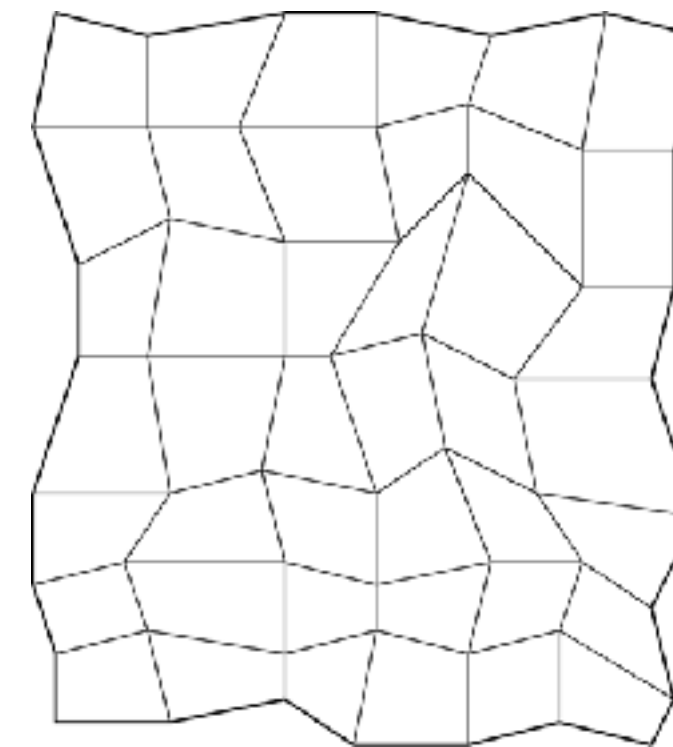
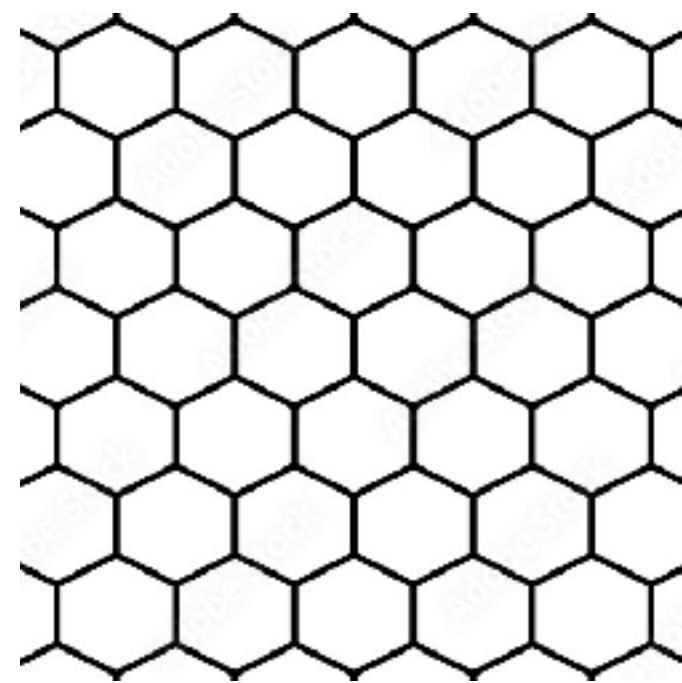
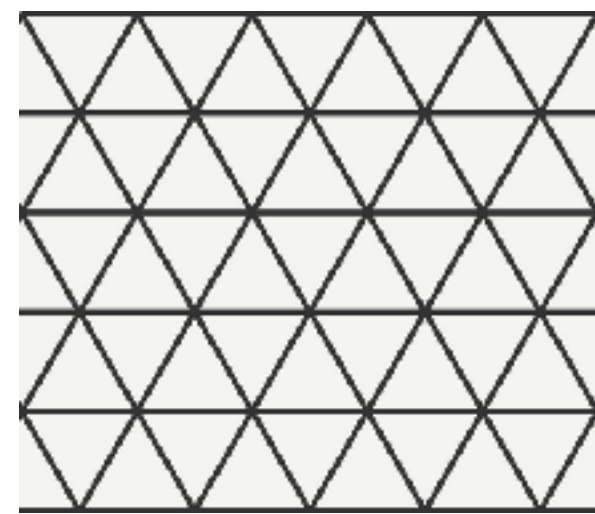
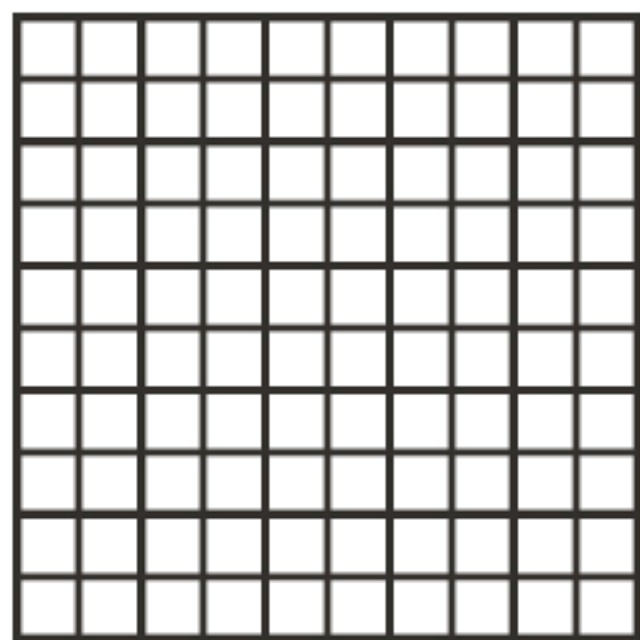
Dwuwymiarowe Automaty Komórkowe

A co jeśli chcemy mieć 2D CA?

- Póki co rozpatrywaliśmy automaty komórkowe 1-wymiarowe. To miłe i fajne, ale mało praktyczne, bo wszystko wskazuje na to, że nasz świat nie jest 1-wymiarowy.
- Co trzeba zrobić aby CA miał więcej wymiarów?
 - Stany
 - Komórki
 - Sąsiedztwo
 - Warunki brzegowe
 - Reguła lokalna i globalna, LUT
 - Space-time diagram

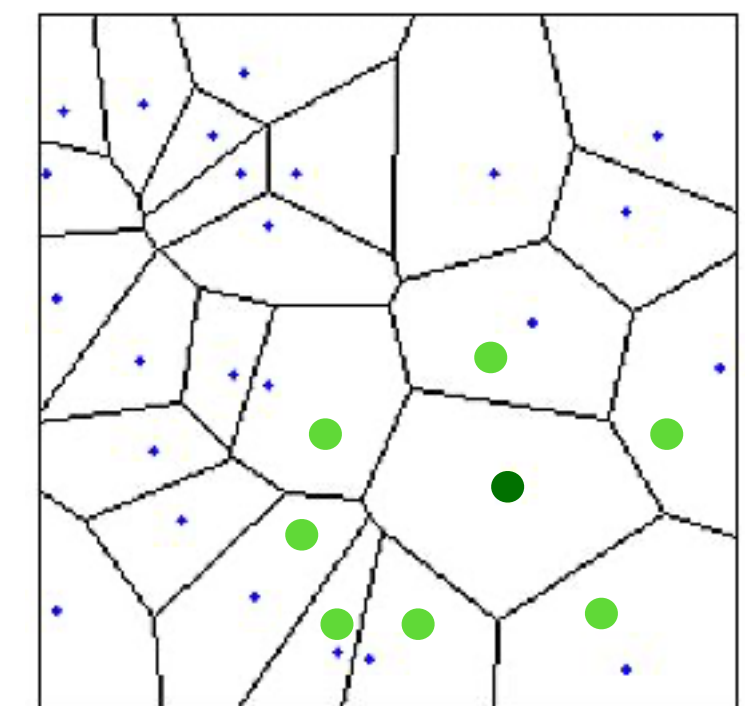
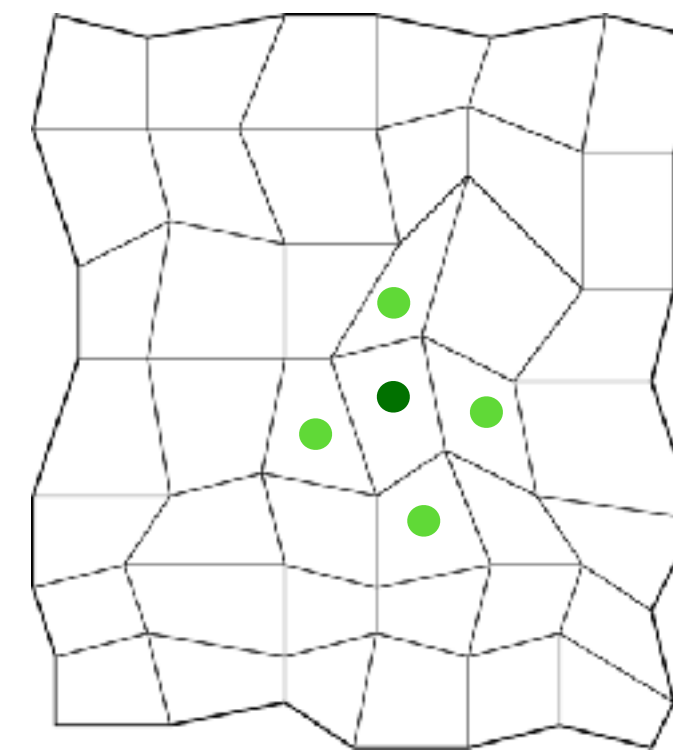
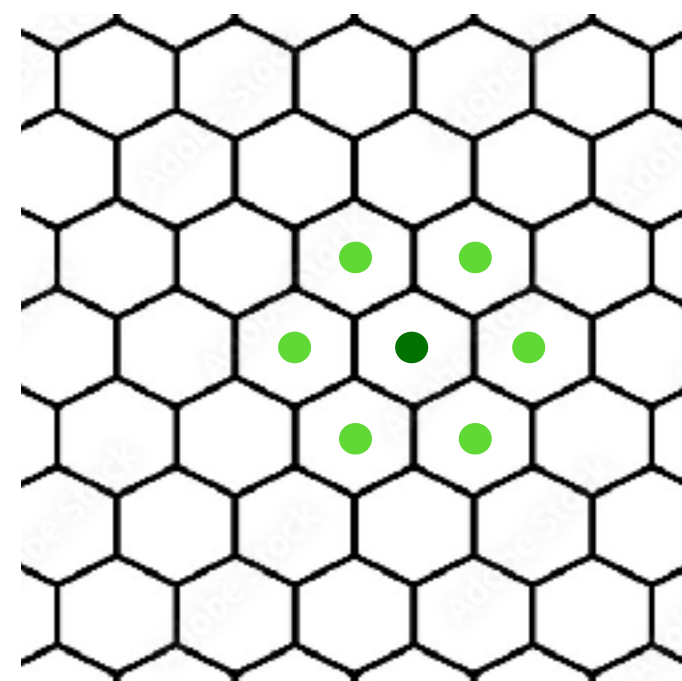
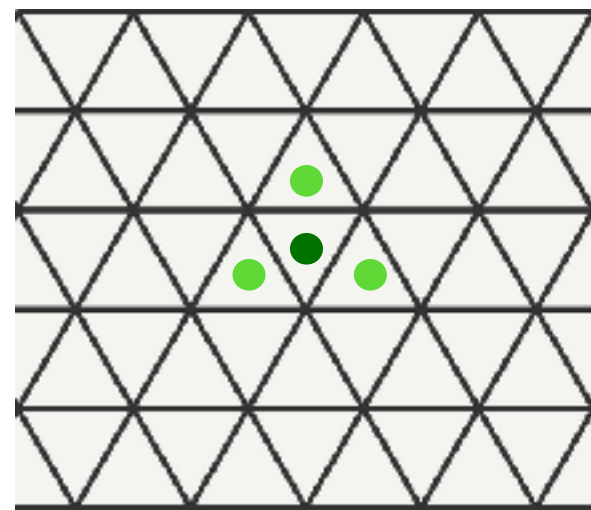
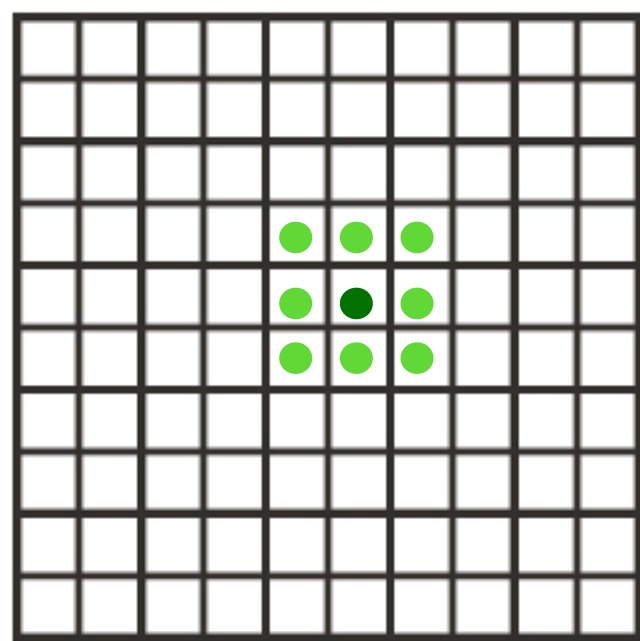
A co jeśli chcemy mieć 2D CA?

- **Stany** - bez zmian, każda z komórek może mieć przypisany któryś ze (skończenie wielu) stanów, ze zbioru stanów.
- **Komórki** - w przypadku 1D dzieliliśmy odcinek / prostą na niepodzielne kawałki. Rysowaliśmy je co prawda jako kwadraciki, ale ich rozmiar nie miał żadnego znaczenia - liczyło się tylko ułożenie obok siebie.
- W przypadku 2D jest podobnie, ale inaczej. Mamy bowiem **wiele sposobów** podziału **płaszczyzny** na małe, niepodzielne kawałki - różnego rodzaju “kraty” i inne podziały.



A co jeśli chcemy mieć 2D CA?

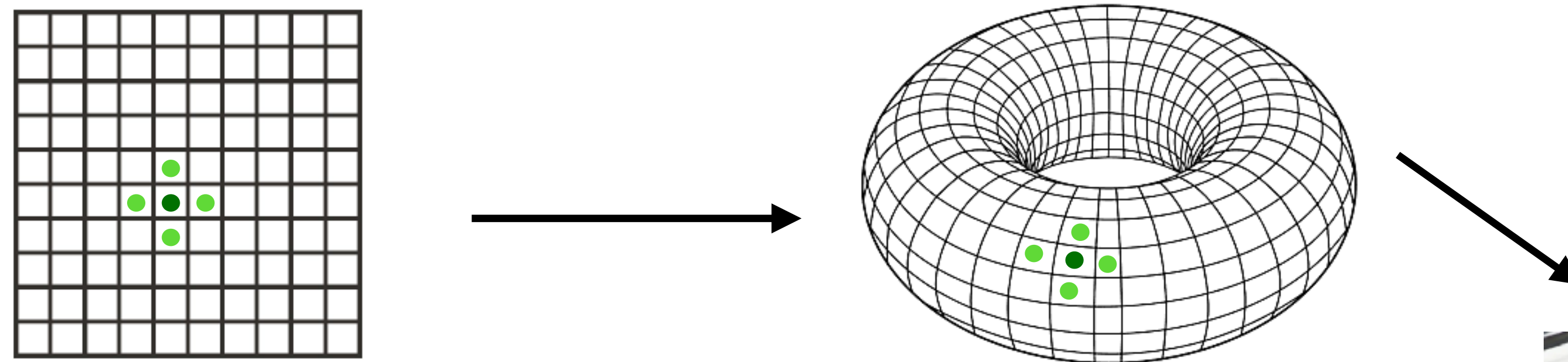
- **Sąsiedztwo** - w przypadku 1D musieliśmy ustalić jedynie promień, czyli liczbę komórek “w prawo” / “w lewo” i właściwie nie było więcej opcji.
- Na płaszczyźnie, zależnie od tego jakie mamy komórki, możemy tworzyć bardzo różne sąsiedztwa. Przykłady (można wymyślać **więcej!**):



- komórka “centralna”
- komórka “sąsiednie” względem “centralnej”

A co jeśli chcemy mieć 2D CA?

- **Warunki brzegowe** - w przypadku 1D, jeśli mieliśmy skończenie wiele komórek, musieliśmy ustalić co się dzieje przed pierwszą i po ostatniej komórce. Popularne wyjścia to “periodic” (sklejamy początek z końcem), “null” (same zera), ewentualnie “reflective” (odbicie lustrzane).
- W przypadku 2D robimy podobnie, ale mamy więcej możliwości.

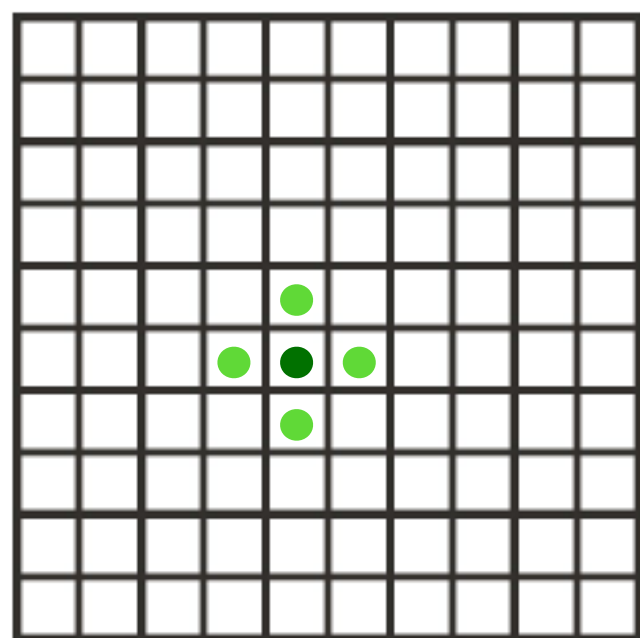


Przykład: periodyczne warunki brzegowe w 2D.



A co jeśli chcemy mieć 2D CA?

- **Reguła lokalna i globalna** - w przypadku 1D to były po prostu funkcje. Reguła lokalna zależna była od tego ile komórek znajduje się w sąsiedztwie. Reguła globalna przypisywała nowe stany wszystkim komórkom “na raz” zgodnie z regułą lokalną.
- W przypadku 2D robimy dokładnie tak samo, ale... z reguły nawet najprostsze przypadki będą miały więcej komórek w sąsiedztwie niż ECA, a zatem reguł lokalnych będzie **dużo**.
- Oczywiście LUT działa analogicznie jak w 1D.



$$f \begin{pmatrix} & x_u & \\ x_l & x_c & x_r \\ & x_d & \end{pmatrix}$$

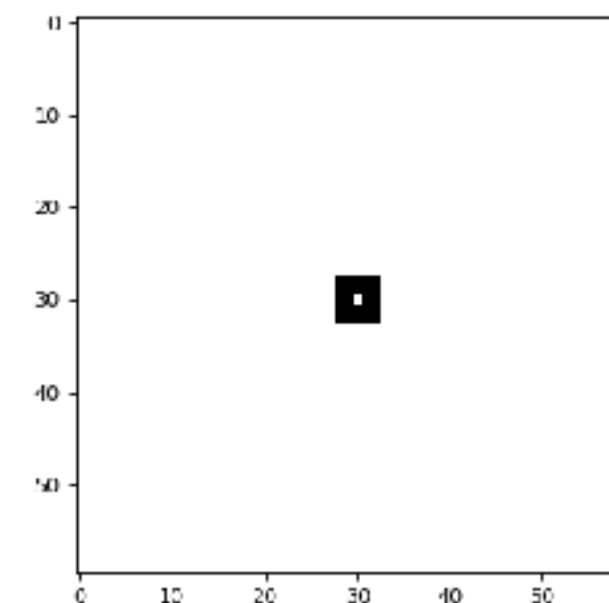
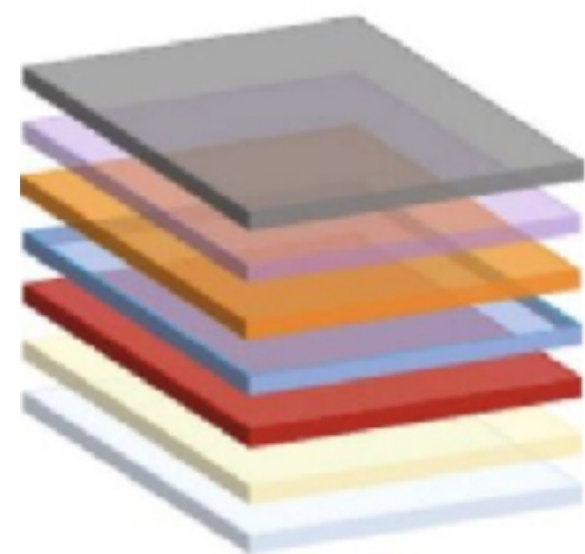


$$f(x_u, x_l, x_c, x_r, x_d)$$

To jest zwykła funkcja 5 zmiennych, ale argumenty są napisane tak, żeby odpowiadały “kształtowi” sąsiedztwa.

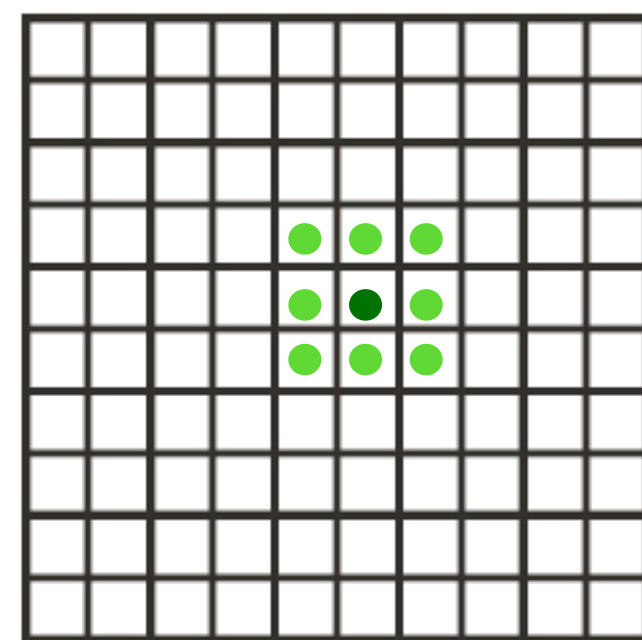
A co jeśli chcemy mieć 2D CA?

- **Space-time diagram** - w przypadku 1D CA diagram był dwu-wymiarowy. Każdy wiersz diagramu odpowiadał pełnemu krokowi czasowemu pokazując stany wszystkich komórek. Innymi słowy space-time diagram pomagał zwizualizować czas.
- W przypadku 2D CA można by rozważać space-time diagram skonstruowany analogicznie jednak jest to bardzo niepraktyczne, bo w tym przypadku ... space-time diagram to macierz trójwymiarowa! Trudno taki obiekt zwizualizować w sposób, który dawałby nam sensowne informacje.
- Dlatego dużo częściej wizualizujemy 2D CA w postaci animacji!

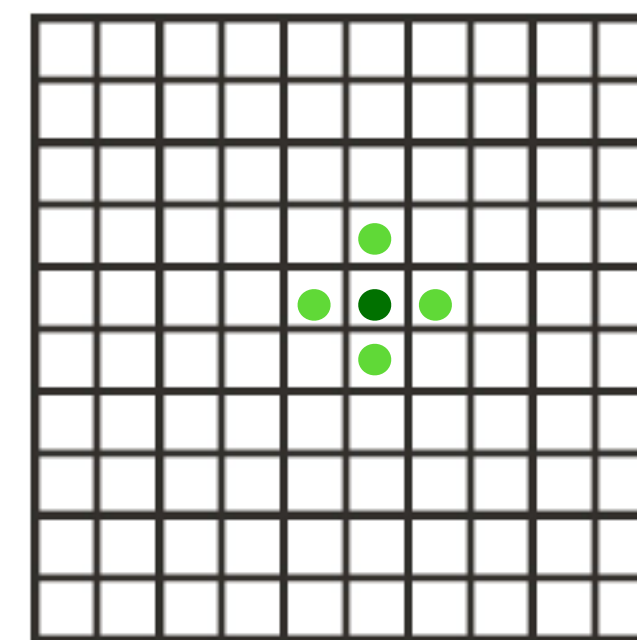


Typowy przypadek

- Będziemy rozważali jedynie przypadek 2D, w którym płaszczyzna podzielona jest na kwadratowe komórki o równym rozmiarze (ang. square grid).
- Komórek będzie skończenie wiele.
- Obowiązywać będą (najczęściej) okresowe warunki brzegowe.
- Rozwierać będziemy dwa rodzaje sąsiedztwa:



sąsiedztwo Moore'a



sąsiedztwo von Neumanna

Conway's Game of Life



John Horton Conway (1937 - 2020)

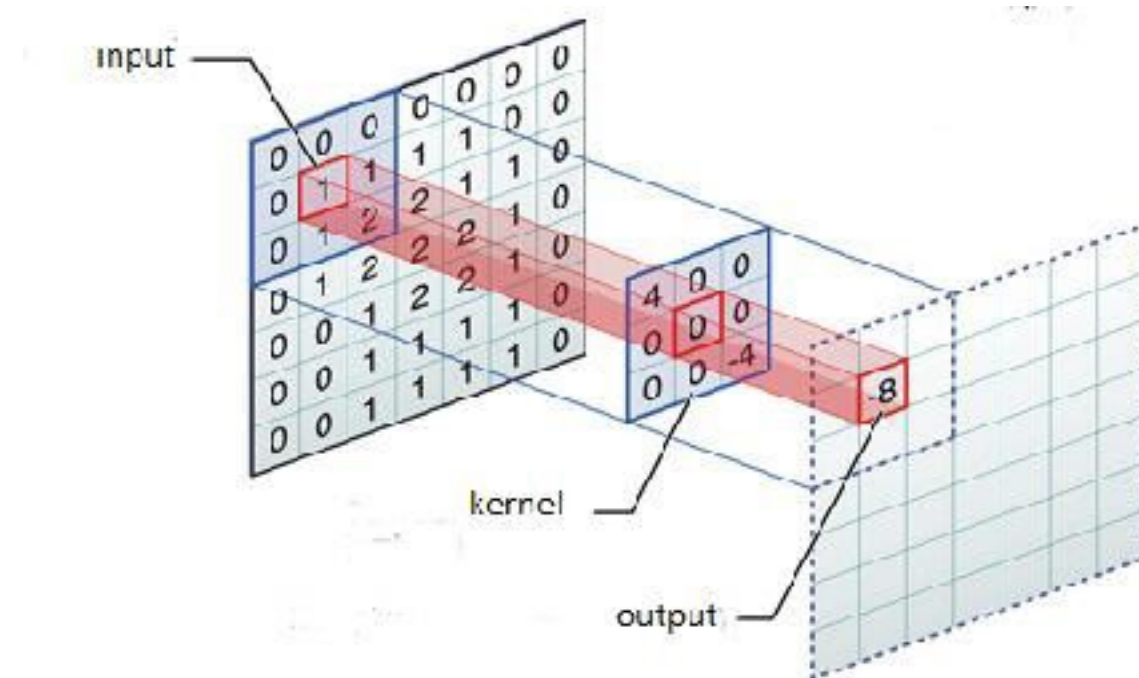
<https://www.youtube.com/watch?v=R9Plq-D1gEk>

Game of Life

- Automat Komórkowy, dwu-stanowy, dwu-wymiarowy, zadany na sąsiedztwie Moore'a, **outer-totalistic**! Czyli prawie wszystko już wiemy!
- Stany: 0 - komórka martwa, 1 - komórka żywa
- Reguła lokalna:
 - martwa komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa w następnej jednostce czasu (rodzi się);
 - żywa komórka z 2 albo 3 żywymi sąsiadami pozostaje nadal żywa; przy innej liczbie sąsiadów umiera (z „samotności” albo „zatłoczenia”).

Implementacja Game of Life

- Operacja splotu macierzy / tensora (ang. *convolution*) w szczególności **conv2d**.
- Popularna dzięki sieciom CNN (Convolution Neural Network).



$$\begin{aligned} &4*0 + 0*0 + 0*0 \\ &+ 0*0 + 0*1 + 0*1 \\ &+ 0*0 + 0*1 + (-4)*2 = -8 \end{aligned}$$

- Kernel to “mała” macierz, która “wędruje” po dużej macierzy i zawiera wagi, przez które mnożone są odpowiednie wartości, a wynik w obrębie mnożenia jest sumowany, zgodnie z rysunkiem powyżej.
- Jak zadziała conv2d z jądrem $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$?
- Jaki to ma związek z Game of Life i regułami outer-totalistic na sąsiedztwie Moore’a?



```
import numpy as np
from scipy.signal import convolve2d

kernel = np.array([
    [1, 1, 1],
    [1, 0, 1],
    [1, 1, 1]
])

game_of_life_rule = lambda n, c: (n == 3) | (c & (n == 2))

def update_grid(grid, rule):
    neighbors = convolve2d(grid, kernel, mode='same', boundary='wrap')
    new_grid = np.zeros_like(grid)
    for i in range(grid.shape[0]):
        for j in range(grid.shape[1]):
            new_grid[i, j] = rule(neighbors[i, j], grid[i, j])
    return new_grid
```

[1]



9.3s

Python

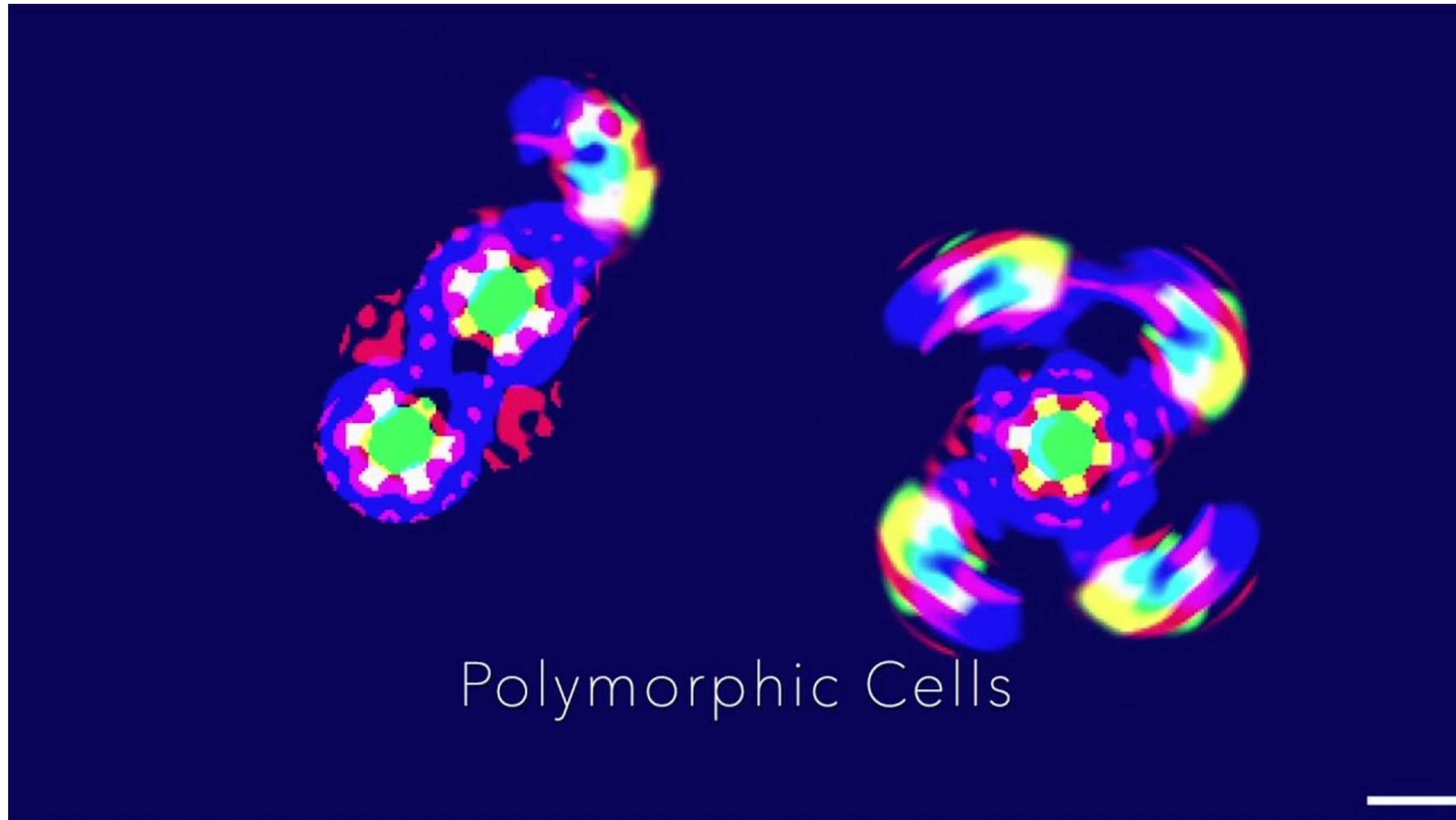
Game of Life

- O regule GoL bardzo wiele wiadomo - istnieje obszerna literatura i dokumentacja dostępna online (<https://conwaylife.com/wiki/>), a także liczne implementacje online (<https://playgameoflife.com/>)
- Ważne konfiguracje:
 - **Glider**: <https://playgameoflife.com/lexicon/glider>
 - **Glider gun**: https://playgameoflife.com/lexicon/Gosper_glider_gun
 - **Glider eater**: <https://playgameoflife.com/lexicon/eater>
 - **Spaceship LWSS**: <https://playgameoflife.com/lexicon/LWSS> (1)
- Te i podobne konstrukcje są podstawą do stwierdzenia, że GoL ma własność *universal computation* - czyli potrafi wykonać dowolnie skomplikowane obliczenie, które umie zrobić komputer. Warto o tym poczytać, a także o teorii Wolframa “zasada obliczeniowej równoważności” (**Principle of Computational Equivalence**).

Reguły Life-like

- Na przestrzeni lat i na fali olbrzymiej popularności GoL powstało wiele podobnych do GoL reguł, które często nazywamy “Life-like”.
- Trudno sensownie sformułować definicję reguł “life-like”, dlatego to co się spotyka w literaturze, to z reguły bardzo ogólne stwierdzenie, że są podobne do GoL lub że są to **dwu-stanowe, dwu-wymiarowe** automaty komórkowe zdefiniowane na **sąsiedztwie Moore’a** spełniające własność **outer-totalistic**. (por. [https://en.wikipedia.org/wiki/Life-like cellular automaton](https://en.wikipedia.org/wiki/Life-like_cellular_automaton))
- Poza regułami Life-like rozważa się też liczne uogólnienia, które dotyczą szerszy klas automatów komórkowych. Szczególnie bardzo mocno :) polecam zapoznanie się z automatami **Lenia** (<https://en.wikipedia.org/wiki/Lenia> oraz <https://chakazul.github.io/lenia.html>)

Lenia



<https://www.youtube.com/watch?v=HT49wpyux-k>

pygame

Pygame is a set of *Python* modules designed for writing video games. Pygame adds functionality on top of the excellent *SDL* library. This allows you to create fully featured games and multimedia programs in the python language.

<https://www.pygame.org/wiki/about>

Podstawy pygame

- **Nie** uruchamiamy w Jupyter (Google Collab nie zadziała)
- Instalacja przez pip (**pip install pygame**), conda lub podobne
- Podstawowe konstrukcje:
 - Inicjalizacja
 - Main game loop
 - Obsługa zdarzeń (eventów)
 - Klawiatura, mysz, resize okna, wyjście z programu
 - Rysowanie w 2D - podstawy
- Nic więcej nie potrzebujemy
 - ... no ewentualnie wyświetlanie napisów :)

Pygame mini demo

```
1  import pygame
2
3  pygame.init()
4
5  screen = pygame.display.set_mode((500,500))
6  pygame.display.set_caption("Pygame demo")
7  clock = pygame.time.Clock()
8
9  running = True
10 while running:
11     for event in pygame.event.get():
12         if event.type == pygame.QUIT:
13             running = False
14         elif event.type == pygame.KEYDOWN:
15             if event.key == pygame.K_ESCAPE:
16                 running = False
17
18     screen.fill((0, 0, 0))
19
20     # drawing goes here
21
22     pygame.display.flip()
23     clock.tick(10)
24
25 pygame.quit()
```

To musi być na początku

Ustawiamy rozmiar okna

Tytuł okna (można zmieniać dynamicznie)

To jest ważne - dużo będzie zależało od czasu

Główna pętla naszej "gry"

Obsługa zdarzeń (ang. event) - jeśli będzie więcej, warto mieć osobną funkcję!

Wypełniamy cały ekran na czarno: (0,0,0) - punkt w przestrzeni kolorów RGB

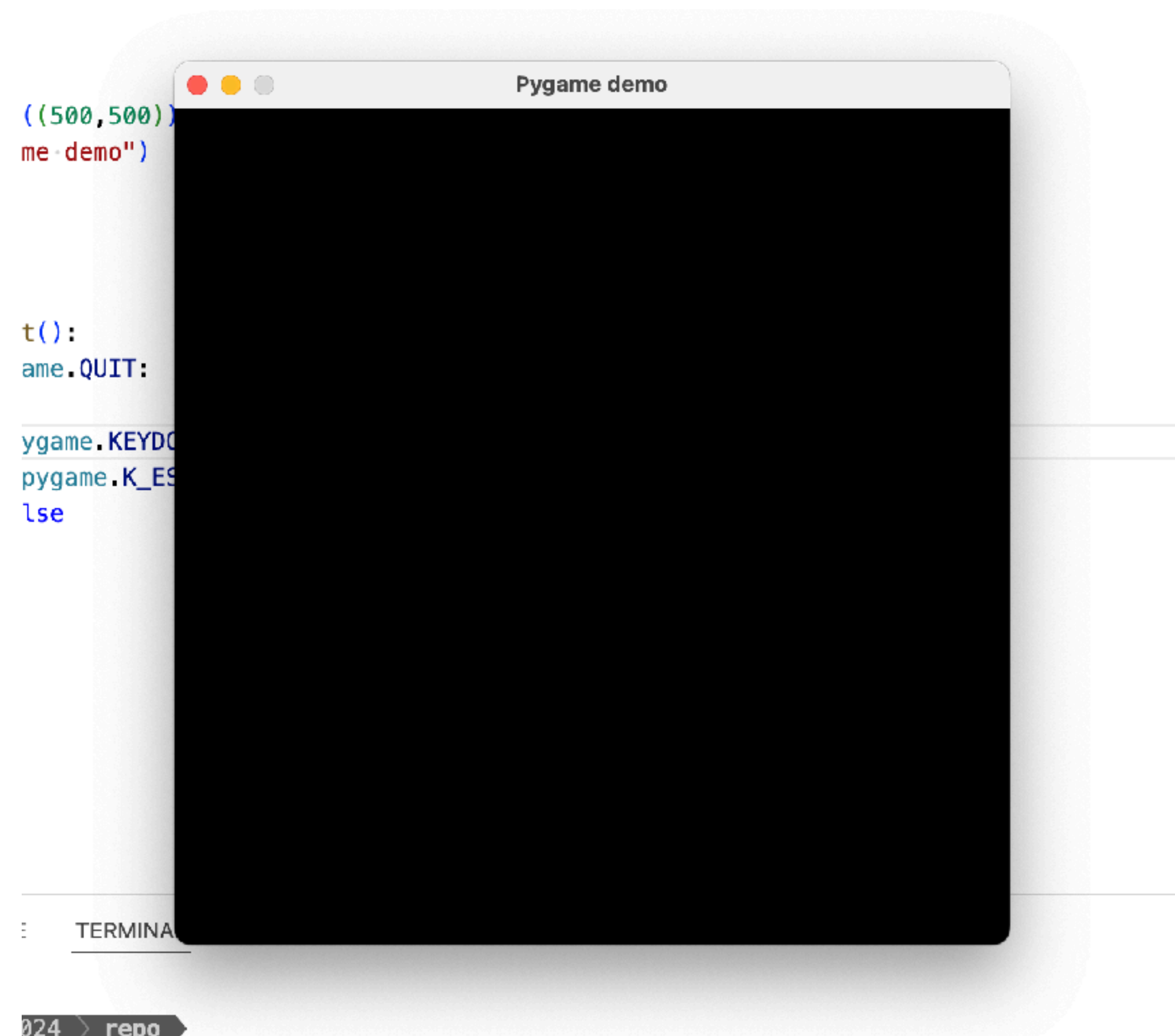
Tu zapewne będzie nasz kod, który "coś" robi.

Double buffering - dzięki temu animacje są płynne.

Pozwala kontrolować tempo naszych animacji (których póki co nie ma).

To musi być na końcu ;)

Pygame mini demo



pygame_starter.py na GitHub

pygame_shapes.py

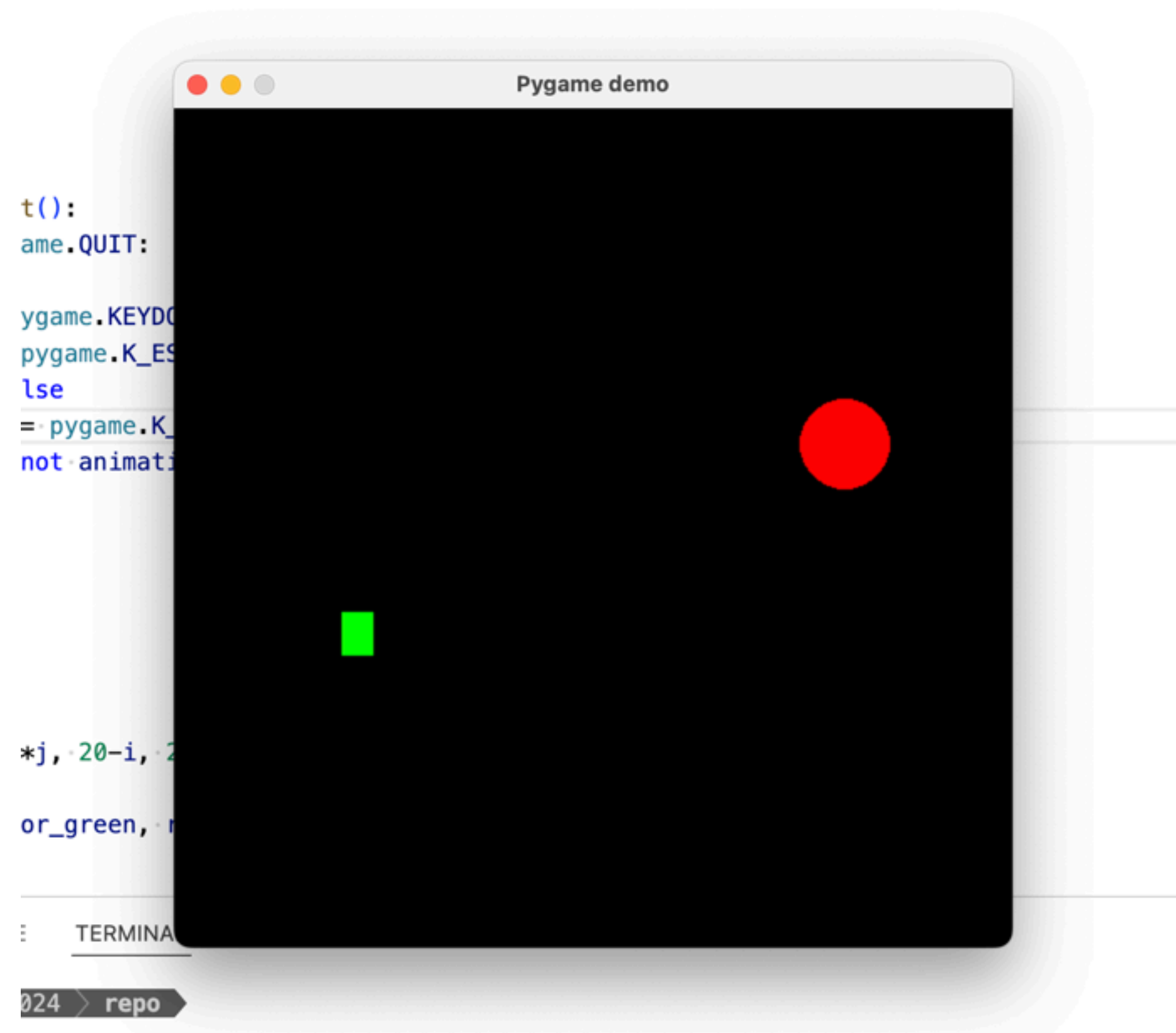
```
1 import pygame
2
3 pygame.init()
4 screen = pygame.display.set_mode((500,500))
5 pygame.display.set_caption("Pygame demo")
6 clock = pygame.time.Clock()
7
8 i, j = 0, 0
9 color_green = (0, 255, 0)
10 color_red = (255, 0, 0)
11
12 running = True
13 animating = False
14
15 while running:
16     for event in pygame.event.get():
17         if event.type == pygame.QUIT:
18             running = False
19         elif event.type == pygame.KEYDOWN:
20             if event.key == pygame.K_ESCAPE:
21                 running = False
22             elif event.key == pygame.K_SPACE:
23                 animating = not animating
24
25     screen.fill((0, 0, 0))
26
27     if animating:
28         i, j = (i+1) % 5, (j+1) % 10
29
30     rect = pygame.Rect(100*i, 50*j, 20-i, 20+j)
31     pygame.draw.rect(screen, color_green, rect)
32
33     pygame.draw.circle(screen, color_red, (500-100*i, 500-50*j), 20+i+j)
34
35     pygame.display.flip()
36     clock.tick(10)
37 pygame.quit()
```

pygame.Rect to obiekt, który definiuje położenie i rozmiar prostokąta.

pygame.draw.rect rysuje prostokąt o zadanych rozmiarach, w zadanym kolorze.

pygame.draw.circle rysuje koło o zadanych środku i zadanym promieniu.

Pygame mini demo 2



pygame_shapes.py na GitHub

Zadanie laboratoryjne

Zadanie 17. Zaimplementuj wizualizację automatu komórkowego Gra w Życie z wykorzystaniem pygame. Zakładamy periodyczne warunki brzegowe. Liczba komórek, rozmiar okna, rozmiar komórki na ekranie mogą być zaszyte w kodzie na sztywno. Program ma umożliwiać:

- **Wstrzymanie / wznowienie symulacji** (pauza - np. przez wciśnięcie spacji na klawiaturze)
- **Wylosowanie warunku początkowego** (np. przy każdym naciśnięciu przycisku enter losujemy nowy warunek)
- **Zwiększenie / zmniejszenie prędkości symulacji** (np. przez wciśnięcie strzałki w górę / w dół na klawiaturze)

Zadanie laboratoryjne (c.d.)

Dodatkowo program powinien umożliwiać conajmniej jedną z następujących operacji:

- Ustalanie konfiguracji początkowej przez klikanie myszką w poszczególne komórki (kliknięcie zmieni stan na przeciwny).
- Poza losowymi warunkami początkowymi, udostępnienie użytkownikowi kilku zapisanych konfiguracji - np. pokazujących glider, glider gun, space ship.
- Możliwość zmiany reguły z tradycyjnej Game of Life na inne reguły “life-like” np. HighLife, Life without Death, 34 Life, Seeds ...
- Wizualizację zmieniających się stanów - odróżnienie w wizualizacji “nowej” od “starej”
1. Czyli np. jeśli dana komórka dopiero co przeszła z 0 na 1, to przez chwilę będzie rysowana jaśniejszym kolorem, niż komórka, która była w stanie 1 już wcześniej. (Podobnie można zrobić dla 0, czyli świeżo “umarłe” komórki zaznaczać na przykład na szaro.) W ten sposób choć automat nadal będzie 2-stanowy, to pokażemy niejako więcej stanów ale jedynie na poziomie **wizualizacji** a nie reguły CA.

Dziękuję bardzo
Witold.Bolt@ug.edu.pl

