# Лабораторная работа №4: Игра "Жизнь"

## Цель работы

Научиться моделировать клеточный автомат, исследовать самоорганизацию и возникновение сложных структур, а также понять связь с ИИ и многоагентными системами.

## Вариант задания

Паттерн: Blinker  
Размер сети: 30x30  
Особенности: Осциллирующий паттерн на фоне случайной сетки.

## Постановка задачи

1. Создать случайную сетку клеточного автомата.  
2. Добавить фиксированный паттерн.  
3. Реализовать правила Конвея.  
4. Запустить симуляцию и визуализировать изменения.

## Теоретическая часть

Игра «Жизнь» — клеточный автомат, предложенный Джоном Конвеем. Каждая клетка может быть живой или мёртвой, переход в следующее поколение происходит по правилам:  
- Живая клетка с <2 живыми соседями умирает.  
- Живая клетка с 2–3 соседями остаётся живой.  
- Живая клетка с >3 соседями умирает.  
- Мёртвая клетка с 3 соседями оживает.

## Описание паттерна Blinker

Blinker — простой осциллирующий паттерн с периодом 2. Состоит из 3 клеток в ряд и меняет положение с вертикального на горизонтальное.

## Код программы

## Пояснения к коду (блоками)

### Инициализация и параметры

Класс GameOfLife принимает ширину/высоту сетки, вероятность рождения живых клеток (alive\_probability) и seed для воспроизводимости. В конструкторе настраивается семя для numpy и random, создаётся случайная сетка и сразу добавляется паттерн Blinker.

### Создание случайной сетки

Метод create\_random\_grid генерирует матрицу из 0 и 1 с заданной вероятностью живых клеток. Это гарантирует случайное, но контролируемое стартовое распределение (20–50% по условию работы).

### Добавление паттерна Blinker

Метод add\_blinker\_pattern строит вертикальный Blinker (3 клетки столбцом), сканирует сетку (без крайних границ), ищет область с минимальным количеством живых клеток вокруг, чтобы снизить конфликт при размещении. Затем накладывает паттерн поверх случайной сетки.

### Подсчёт соседей (тороидальная топология)

Метод count\_neighbors обходит 8 соседей клетки, используя взятие по модулю по обеим осям, что соответствует замыканию поля на тор (левый край соединён с правым, верхний — с нижним).

### Обновление по правилам Конвея

Метод update\_grid создает новую сетку и для каждой клетки применяет правила: меньше 2 соседей — смерть от одиночества; 2–3 — выживание; больше 3 — смерть от перенаселения; ровно 3 у мёртвой — рождение.

### Статистика и сброс

Метод get\_stats считает число живых клеток, плотность и номер поколения. reset возвращает систему к начальному состоянию, сохранённому после генерации и вставки паттерна.

### Запуск симуляции (текст/анимация)

run\_simulation может работать в двух режимах: текстовый (печатает статистику по шагам) или анимированный (animate\_simulation). В текстовом режиме предусмотрена ранняя остановка при вымирании.

### Анимация Matplotlib

animate\_simulation строит поле, настраивает сетку координат, монокромную палитру, текст со статистикой и обновляет кадры через FuncAnimation. Заголовок и блок статистики обновляются на каждом кадре без blit, чтобы корректно перерисовывать текст.

### Сохранение/загрузка паттерна

save\_pattern и load\_pattern сохраняют/загружают текущее состояние матрицы в текстовый файл. После загрузки корректируются высота/ширина и номер поколения.

### Точка входа (\_\_main\_\_)

Создаётся игра с параметрами варианта (30×30, alive\_probability около 0.35, фиксированный seed), печатается краткая стартовая статистика и первые 10 строк поля в псевдографике, затем запускается анимация. Если анимация недоступна — запускается текстовый режим.

## Полный исходный код

import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.animation as animation  
from matplotlib.colors import ListedColormap  
import random  
  
class GameOfLife:  
 def \_\_init\_\_(self, width=30, height=30, alive\_probability=0.3, seed=None):  
 """  
 Инициализация игры Жизнь  
   
 Args:  
 width: ширина сетки  
 height: высота сетки  
 alive\_probability: вероятность живой клетки при случайной генерации  
 seed: семя для генератора случайных чисел (для воспроизводимости)  
 """  
 self.width = width  
 self.height = height  
   
 # Установка семени для уникальности каждого студента  
 if seed is None:  
 seed = random.randint(1, 10000)  
 np.random.seed(seed)  
 random.seed(seed)  
   
 print(f"Семя генератора: {seed} (сохраните для воспроизводимости)")  
   
 # Создание случайной сетки  
 self.grid = self.create\_random\_grid(alive\_probability)  
   
 # Добавление паттерна Blinker  
 self.add\_blinker\_pattern()  
   
 # Сохранение начального состояния  
 self.initial\_grid = self.grid.copy()  
   
 self.generation = 0  
   
 def create\_random\_grid(self, alive\_probability):  
 """Создание случайной сетки с заданной вероятностью живых клеток"""  
 return np.random.choice([0, 1], size=(self.height, self.width),   
 p=[1-alive\_probability, alive\_probability])  
   
 def add\_blinker\_pattern(self):  
 """  
 Добавление паттерна Blinker на сетку в свободной зоне  
 Blinker - это осциллирующий паттерн из 3 клеток в ряд  
 """  
 # Паттерн Blinker (вертикальный)  
 blinker = np.array([[0, 1, 0],  
 [0, 1, 0],  
 [0, 1, 0]])  
   
 pattern\_height, pattern\_width = blinker.shape  
   
 # Поиск свободной зоны для размещения паттерна  
 best\_pos = None  
 min\_conflicts = float('inf')  
   
 # Проверяем различные позиции, избегая границ  
 for start\_row in range(2, self.height - pattern\_height - 2):  
 for start\_col in range(2, self.width - pattern\_width - 2):  
 # Проверяем область размещения и окрестности  
 end\_row = start\_row + pattern\_height  
 end\_col = start\_col + pattern\_width  
   
 # Расширенная область для проверки (включая границу в 1 клетку)  
 check\_area = self.grid[start\_row-1:end\_row+1, start\_col-1:end\_col+1]  
 conflicts = np.sum(check\_area)  
   
 # Ищем место с минимальным количеством живых клеток  
 if conflicts < min\_conflicts:  
 min\_conflicts = conflicts  
 best\_pos = (start\_row, start\_col)  
   
 # Если нашли полностью свободную область - используем её  
 if conflicts == 0:  
 break  
   
 if min\_conflicts == 0:  
 break  
   
 # Если не нашли идеальное место, используем лучшее из найденных  
 if best\_pos is None:  
 # Fallback к центру если ничего не найдено  
 best\_pos = (self.height // 2 - 1, self.width // 2 - 1)  
   
 start\_row, start\_col = best\_pos  
 end\_row = start\_row + pattern\_height  
 end\_col = start\_col + pattern\_width  
   
 # Размещение паттерна  
 self.grid[start\_row:end\_row, start\_col:end\_col] = blinker  
   
 print(f"Паттерн Blinker добавлен в позицию ({start\_row}, {start\_col})")  
 print(f"Конфликтов с существующими клетками: {min\_conflicts}")  
   
 def count\_neighbors(self, row, col):  
 """Подсчет живых соседей для клетки"""  
 count = 0  
 for i in range(-1, 2):  
 for j in range(-1, 2):  
 if i == 0 and j == 0:  
 continue  
   
 neighbor\_row = (row + i) % self.height  
 neighbor\_col = (col + j) % self.width  
 count += self.grid[neighbor\_row, neighbor\_col]  
   
 return count  
   
 def update\_grid(self):  
 """Обновление сетки согласно правилам игры Конвея"""  
 new\_grid = np.zeros((self.height, self.width), dtype=int)  
   
 for row in range(self.height):  
 for col in range(self.width):  
 neighbors = self.count\_neighbors(row, col)  
 current\_cell = self.grid[row, col]  
   
 # Применение правил игры Жизнь  
 if current\_cell == 1: # Живая клетка  
 if neighbors < 2: # Смерть от одиночества  
 new\_grid[row, col] = 0  
 elif neighbors == 2 or neighbors == 3: # Выживание  
 new\_grid[row, col] = 1  
 else: # neighbors > 3, смерть от перенаселения  
 new\_grid[row, col] = 0  
 else: # Мертвая клетка  
 if neighbors == 3: # Рождение  
 new\_grid[row, col] = 1  
   
 self.grid = new\_grid  
 self.generation += 1  
   
 def get\_stats(self):  
 """Получение статистики текущего поколения"""  
 alive\_cells = np.sum(self.grid)  
 total\_cells = self.width \* self.height  
 density = alive\_cells / total\_cells  
   
 return {  
 'generation': self.generation,  
 'alive\_cells': alive\_cells,  
 'total\_cells': total\_cells,  
 'density': density  
 }  
   
 def reset(self):  
 """Сброс к начальному состоянию"""  
 self.grid = self.initial\_grid.copy()  
 self.generation = 0  
   
 def run\_simulation(self, steps=100, show\_animation=True):  
 """  
 Запуск симуляции  
   
 Args:  
 steps: количество шагов симуляции  
 show\_animation: показывать ли анимацию  
 """  
 if show\_animation:  
 self.animate\_simulation(steps)  
 else:  
 # Текстовый режим  
 for step in range(steps):  
 stats = self.get\_stats()  
 print(f"Поколение {stats['generation']}: "  
 f"Живых клеток: {stats['alive\_cells']}, "  
 f"Плотность: {stats['density']:.3f}")  
   
 self.update\_grid()  
   
 # Остановка при вымирании всех клеток  
 if np.sum(self.grid) == 0:  
 print("Все клетки погибли!")  
 break  
   
 def animate\_simulation(self, steps=100):  
 """Анимированная визуализация симуляции"""  
 fig, ax = plt.subplots(figsize=(12, 10))  
   
 # Создание цветовой карты  
 colors = ['white', 'black']  
 cmap = ListedColormap(colors)  
   
 # Начальное отображение с правильной статистикой  
 im = ax.imshow(self.grid, cmap=cmap, animated=True)  
   
 # Получаем начальную статистику  
 initial\_stats = self.get\_stats()  
 ax.set\_title(f'Игра "Жизнь" с паттерном Blinker - Поколение: {initial\_stats["generation"]}', fontsize=14)  
 ax.set\_xlabel('X')  
 ax.set\_ylabel('Y')  
   
 # Добавление сетки  
 ax.set\_xticks(np.arange(-0.5, self.width, 1), minor=True)  
 ax.set\_yticks(np.arange(-0.5, self.height, 1), minor=True)  
 ax.grid(which="minor", color="gray", linestyle='-', linewidth=0.5, alpha=0.3)  
   
 # Текст для отображения статистики с начальными значениями  
 initial\_stats\_str = (f'Поколение: {initial\_stats["generation"]}\n'  
 f'Живых клеток: {initial\_stats["alive\_cells"]}\n'  
 f'Плотность: {initial\_stats["density"]:.3f}\n'  
 f'Размер сетки: {self.width}×{self.height}')  
   
 stats\_text = ax.text(0.02, 0.98, initial\_stats\_str, transform=ax.transAxes,   
 verticalalignment='top', fontfamily='monospace', fontsize=10,  
 bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.9))  
   
 def animate(frame):  
 # Для первого кадра показываем начальное состояние (поколение 0)  
 # Для последующих кадров сначала обновляем, потом отображаем  
 if frame > 0:  
 self.update\_grid()  
   
 # Обновляем изображение  
 im.set\_array(self.grid)  
   
 # Получаем актуальную статистику  
 stats = self.get\_stats()  
   
 # Обновляем заголовок  
 title = f'Игра "Жизнь" с паттерном Blinker - Поколение: {stats["generation"]}'  
 ax.set\_title(title, fontsize=14)  
   
 # Обновляем текст статистики  
 stats\_str = (f'Поколение: {stats["generation"]}\n'  
 f'Живых клеток: {stats["alive\_cells"]}\n'  
 f'Плотность: {stats["density"]:.3f}\n'  
 f'Размер сетки: {self.width}×{self.height}')  
 stats\_text.set\_text(stats\_str)  
   
 # Возвращаем только изображение  
 return [im]  
   
 # Создание анимации (отключаем blit для корректного отображения текста)  
 ani = animation.FuncAnimation(fig, animate, frames=steps,   
 interval=300, blit=False, repeat=False)  
   
 plt.tight\_layout()  
 plt.show()  
   
 return ani  
  
 def save\_pattern(self, filename):  
 """Сохранение текущего паттерна в файл"""  
 np.savetxt(filename, self.grid, fmt='%d')  
 print(f"Паттерн сохранен в файл: {filename}")  
   
 def load\_pattern(self, filename):  
 """Загрузка паттерна из файла"""  
 self.grid = np.loadtxt(filename, dtype=int)  
 self.height, self.width = self.grid.shape  
 self.generation = 0  
 print(f"Паттерн загружен из файла: {filename}")  
  
# Пример использования  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("=== Лабораторная работа №4: Игра Жизнь с паттерном Blinker ===\n")  
   
 # Создание игры с параметрами из задания  
 game = GameOfLife(width=30, height=30, alive\_probability=0.35, seed=12345)  
   
 print(f"Размер сетки: {game.width}×{game.height}")  
 print(f"Начальная статистика:")  
 initial\_stats = game.get\_stats()  
 print(f" Живых клеток: {initial\_stats['alive\_cells']}")  
 print(f" Плотность: {initial\_stats['density']:.3f}")  
 print()  
   
 # Отображение начального состояния  
 print("Начальное состояние (первые 10 строк):")  
 for i in range(min(10, game.height)):  
 row\_str = ''.join(['█' if cell else '·' for cell in game.grid[i]])  
 print(f"{i:2d}: {row\_str}")  
 print()  
   
 # Запуск симуляции  
 print("Запуск анимированной симуляции...")  
 print("Закройте окно с анимацией для продолжения.\n")  
   
 try:  
 # Анимированная симуляция  
 game.run\_simulation(steps=200, show\_animation=True)  
 except Exception as e:  
 print(f"Ошибка при показе анимации: {e}")  
 print("Запуск в текстовом режиме...\n")  
   
 # Сброс и запуск в текстовом режиме  
 game.reset()  
 game.run\_simulation(steps=50, show\_animation=False)  
   
 print("\n=== Анализ результатов ===")  
 print("В данной реализации:")  
 print("1. Создана случайная сетка 30×30 с уникальным семенем")  
 print("2. Добавлен осциллирующий паттерн Blinker в свободной зоне")  
 print("3. Реализованы все правила игры Конвея")  
 print("4. Показано взаимодействие структур с случайной динамикой")  
 print("5. Визуализирована эволюция системы")  
 print("6. Паттерн размещается с минимальным перекрытием случайных клеток")  
   
 # Дополнительная информация о паттерне Blinker  
 print("\nИнформация о паттерне Blinker:")  
 print("- Период осцилляции: 2 поколения")  
 print("- Чередует вертикальное и горизонтальное положение")  
 print("- Один из простейших осцилляторов в игре Жизнь")

## Результаты и наблюдения

При запуске с параметрами 30×30 и alive\_probability=0.35 Blinker осциллирует с периодом 2, взаимодействуя с ближайшими кластерами случайно рождённых клеток. На отдельных шагах наблюдаются локальные стабилизирующиеся структуры и затухающие волны рождаемости/смертей. При высокой плотности начального поля видны участки перенаселения, ведущие к уменьшению плотности в последующих поколениях.