Отчет по лабораторной работе «Игра жизнь»

Комаров Иван, Журавлев Федор

242061

Инженерия данных, ИД24-2

Лаба 3

ИД24-2

25.11.2025

Реализация игры «Жизнь» с тороидальной геометрией и альтернативными правилами

Описание задания

Реализовать известный клеточный автомат — игру "Жизнь" с использованием тороидальной геометрии, начиная с небольшого количества живых клеток. На том же движке реализовать игру по альтернативным правилам. Обеспечить возможность добавления известных фигур на поле путем перетаскивания.

Достигнутая сложность и реализация

Основные требования выполнены:

- Игра реализована с классическими и альтернативными правилами
- Сетка с тороидальной геометрией (границы завёрнуты)
- Начальное заполнение поля с несколькими живыми клетками
- Добавление фигур (glider, small_exploder) перетаскиванием мыши

Дополнительная функциональность:

- Использование конфигурационного JSON-файла для параметров окна, цветов, скорости
- Автоматическое обновление поколения с помощью таймера
- Поддержка двух вариантов правил
- Переключение правил с клавиатуры

Объяснение проектирования программы

ООП-структура:

- GameOfLife — хранит состояние клеток, обновляет сетку, загружает фигуры
- GameWindow — отвечает за графический интерфейс, обработку событий, отрисовку

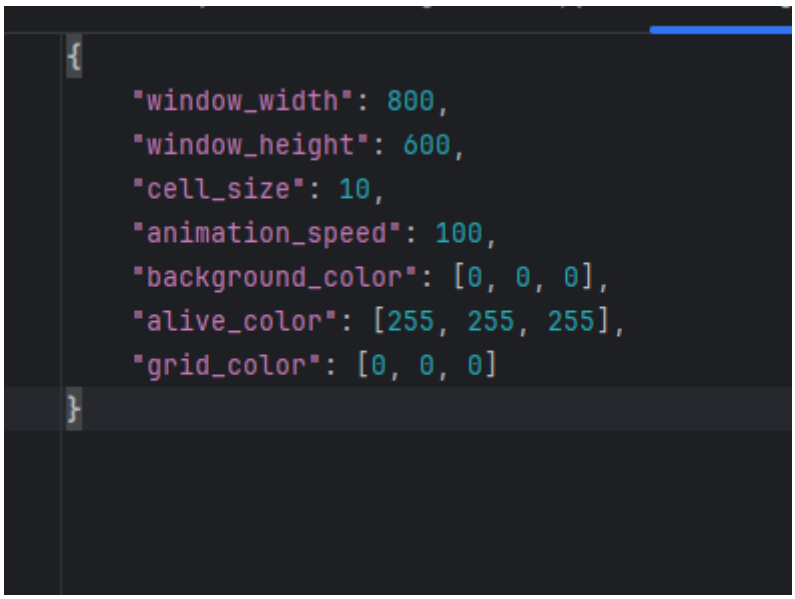
Организация кода:

- Четкое разделение логики и интерфейс
- Конфигурация параметров в отдельном JSON
- Использование PyQt5 для визуализации и взаимодействия

Математическое исследование:

- Классические: клетка живет при 2 или 3 соседях, рождается при 3
- Альтернативные: дополнительно рождается при 6 соседях
- Соседи считаются с учетом завёрнутых границ (тороид)

Конфигурационный файл задает: размер окна, размер клеток, цвета фона, клеток, сетки, скорость анимации.



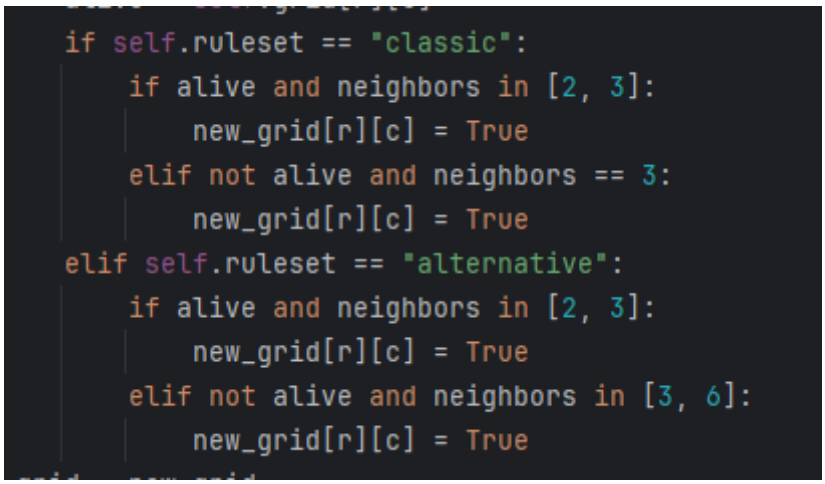
```
{
    "window_width": 800,
    "window_height": 600,
    "cell_size": 10,
    "animation_speed": 100,
    "background_color": [0, 0, 0],
    "alive_color": [255, 255, 255],
    "grid_color": [0, 0, 0]
}
```

```
# Конфигурация из json
with open("config.json", "r", encoding="utf-8") as file:
    config = json.load(file)
```

Основная часть (по требованиям)

Требование: реализовать классические правила игры "Жизнь"

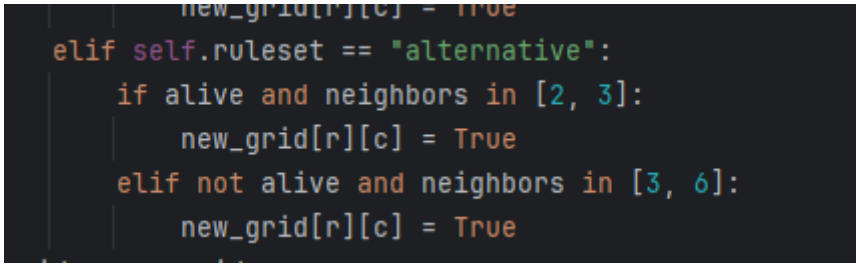
Решение: метод update класса GameOfLife проверяет соседей и обновляет состояние



```
if self.ruleset == "classic":
    if alive and neighbors in [2, 3]:
        new_grid[r][c] = True
    elif not alive and neighbors == 3:
        new_grid[r][c] = True
elif self.ruleset == "alternative":
    if alive and neighbors in [2, 3]:
        new_grid[r][c] = True
    elif not alive and neighbors in [3, 6]:
        new_grid[r][c] = True
```

Требование: реализовать альтернативные правила

Решение: в методе update добавлено условие для рождения при 6 соседях



```
elif self.ruleset == "alternative":
    if alive and neighbors in [2, 3]:
        new_grid[r][c] = True
    elif not alive and neighbors in [3, 6]:
        new_grid[r][c] = True
```

Требование: добавить фигуры перетаскиванием

Решение: обработчики событий мыши в GameWindow позволяют перетягивать и размещать шаблоны

```python
# Отрисовка перетаскиваемой фигуры
if self.dragging_shape and self.mouse_pos:
    shape_coords = self.game.shapes[self.dragging_shape]
    painter.setBrush(QColor(200, 200, 200, 180))
    for dr, dc in shape_coords:
        x = self.mouse_pos.x() + dc * self.cell_size
        y = self.mouse_pos.y() + dr * self.cell_size
        rect = QRect(x, y, self.cell_size, self.cell_size)
        painter.drawRect(rect)
```

Скриншоты показывают интерфейс и игровой процесс:

Полный исходный код и файл конфигурации JSON приложены к письму.