# A. ТИТУЛЬНАЯ СТРАНИЦА

ФИО: Зиновьев Михаил Михайлович
Номер билета: 245121
Предмет: практикум по программированию
Лаба: 3
Группа: ИД24-2
Дата: 25.11.2025
Задача: Фрактал Омара Пола

---

# B. ОПИСАНИЕ ЗАДАНИЯ

Реализовать визуализацию фрактала Toothpick Sequence:

- Нарисовать узор из зубочисток рекурсивно
- Реализовать автоматический зум
- Применить ООП и конфигурацию

---

# C. ДОСТИГНУТАЯ СЛОЖНОСТЬ

Основные требования:
- ООП архитектура (классы Toothpick и FractalRenderer)
- Конфигурационный файл JSON
- Качественный документированный код
- Автоматический зум с плавной интерполяцией
- Правильная генерация фрактала

Дополнительно:
- Оптимизированный алгоритм с множеством для отслеживания точек
- Проверка дублирования через __hash__ и __eq__
- Управление в реальном времени (SPACE, R, ESC)

---

# D. ПРОЕКТИРОВАНИЕ

Класс Toothpick:

- Инкапсулирует (x, y, length, direction)
- Методы: get_endpoints(), draw(), eq(), hash()

Класс FractalRenderer:

- Управляет списком зубочисток
- generate_next_generation() - добавляет перпендикулярные линии к свободным концам
- calculate_zoom() - вычисляет масштаб для центрирования
- draw() - плавная интерполяция зума

Config.json (13 параметров):

json

```
{
  "window_width": 1000, "window_height": 1000, "fps": 30,
  "background_color": [0, 0, 0], "toothpick_color": [255, 255, 255],
  "toothpick_length": 20, "toothpick_thickness": 2,
  "max_generations": 15, "generation_delay": 30,
  "auto_zoom_enabled": true, "zoom_padding": 50, "zoom_speed": 0.05
}
```

---

# E. ОСНОВНАЯ ЧАСТЬ

## E.1. Автоматический зум

Требование: Камера отъезжает, чтобы видна была вся фигура.

Решение:

python

```
def calculate_zoom(self, bounds):
    """Масштаб подстраивается под размер фрактала с отступом"""
    min_x, max_x, min_y, max_y = bounds
    scale_x = (width - 2*padding) / max(width, 1)
    scale_y = (height - 2*padding) / max(height, 1)
    return (center_x, center_y, min(scale_x, scale_y))


# В draw() - плавная интерполяция
self.current_scale += (target_scale - self.current_scale) * zoom_speed
```

## E.2. Генерация фрактала

Требование: Алгоритм роста согласно правилам Toothpick Sequence.

Решение:

python

```
def generate_next_generation(self):
    """Для каждого свободного конца добавляем перпендикулярную линию"""
    for toothpick in self.toothpicks:
        for endpoint in toothpick.get_endpoints():
```

```
        if endpoint in self.used_endpoints:
            continue
        # Если конец касается ровно 1 линии - он свободен
        touching_count = sum(1 for t in self.toothpicks
                             if endpoint in t.get_endpoints())
        if touching_count == 1:
            new_direction = 'H' if toothpick.direction == 'V' else 'V'
            new = Toothpick(endpoint[0], endpoint[1],
                            self.config['toothpick_length'], new_direction)
            if new not in self.toothpicks:
                new_toothpicks.append(new)
                self.used_endpoints.add(endpoint)
```

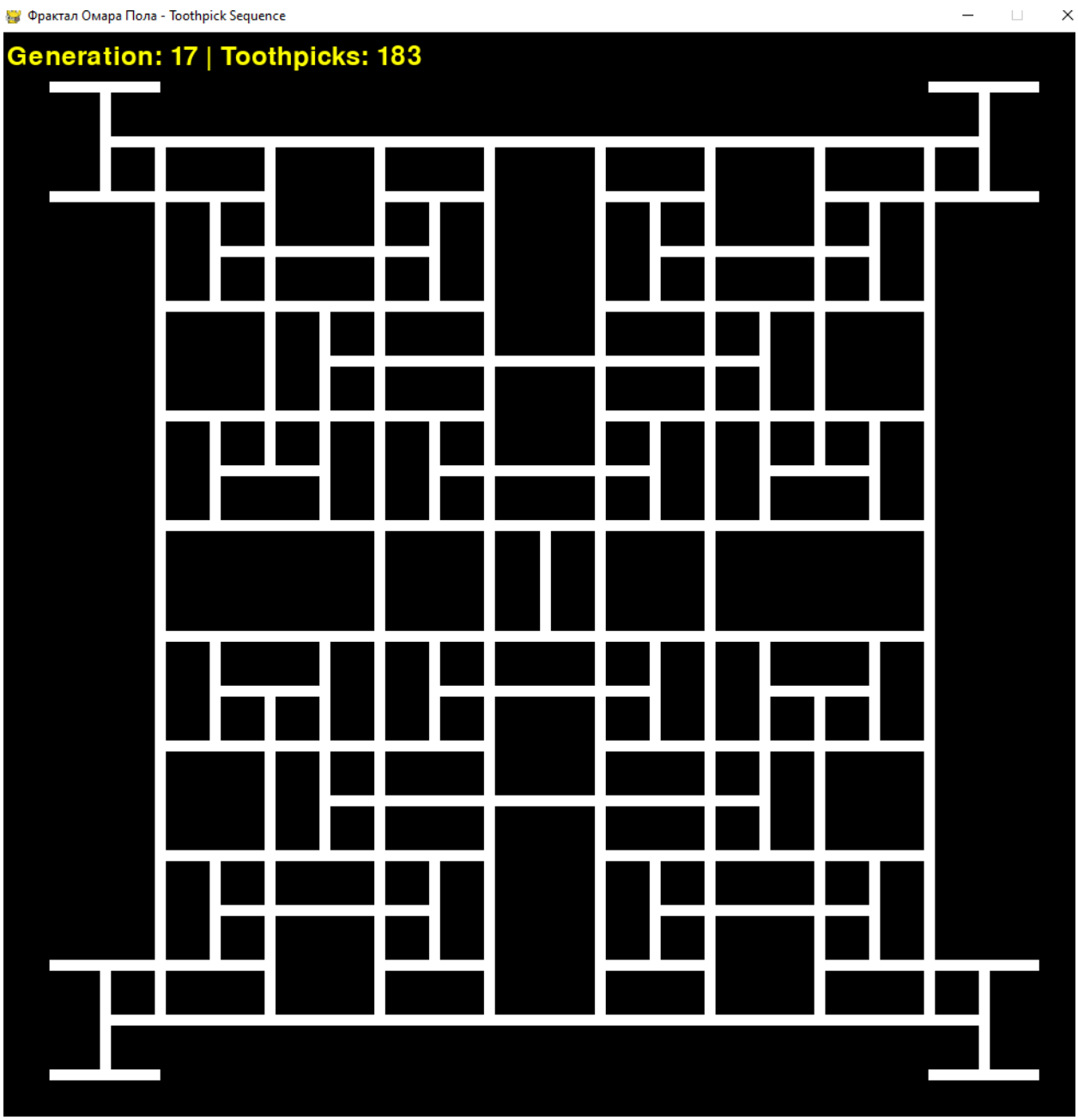
## E.3. ООП реализация

Требование: Классы с четкой ответственностью.

Решение:

- Toothpick - описывает одну линию (данные + методы рисования)
- FractalRenderer - управляет всей системой (логика + отрисовка)
- Разделение на файлы: toothpick.py, fractal_renderer.py, main.py

## E.4. скриншот работы кода




---

## F. ИСХОДНЫЙ КОД

**main.py**

python

```
from fractal_renderer import FractalRenderer


if __name__ == '__main__':
    FractalRenderer('config.json').run()
```

**toothpick.py**

python

```
class Toothpick:
    def __init__(self, x, y, length, direction):
        self.x, self.y, self.length, self.direction = x, y, length, direction

    def get_endpoints(self):
        half = self.length / 2
        if self.direction == 'H':
            return ((self.x - half, self.y), (self.x + half, self.y))
        return ((self.x, self.y - half), (self.x, self.y + half))

    def draw(self, surface, color, thickness, offset_x, offset_y, scale):
        import pygame
        p1, p2 = self.get_endpoints()
        p1 = ((p1[0] - offset_x) * scale, (p1[1] - offset_y) * scale)
        p2 = ((p2[0] - offset_x) * scale, (p2[1] - offset_y) * scale)
        pygame.draw.line(surface, color, p1, p2, max(1, int(thickness * scale)))

    def __eq__(self, other):
        return (self.x == other.x and self.y == other.y and
                self.direction == other.direction) if isinstance(other, Toothpick) else False

    def __hash__(self):
        return hash((self.x, self.y, self.direction))
```

**fractal_renderer.py**

```
python
import pygame, json
from toothpick import Toothpick

class FractalRenderer:
    def __init__(self, config_path='config.json'):
        with open(config_path, 'r', encoding='utf-8') as f:
            self.config = json.load(f)
```

```
        pygame.init()
        self.screen = pygame.display.set_mode(
            (self.config['window_width'], self.config['window_height']))
        pygame.display.set_caption('Toothpick Fractal')
        self.clock = pygame.time.Clock()
        self.font = pygame.font.Font(None, 36)
        self.reset_fractal()

    def reset_fractal(self):
        self.toothpicks = [Toothpick(0, 0, self.config['toothpick_length'], 'V')]
        self.used_endpoints = set()
        self.current_generation = 0
        self.frame_counter = 0
        self.running = True

    def get_bounds(self):
        if not self.toothpicks:
            return (-100, 100, -100, 100)
        coords = [c for t in self.toothpicks for c in t.get_endpoints()]
        return (min(c[0] for c in coords), max(c[0] for c in coords),
                min(c[1] for c in coords), max(c[1] for c in coords))

    def calculate_zoom(self, bounds):
        min_x, max_x, min_y, max_y = bounds
        pad = self.config['zoom_padding']
        scale = min((self.config['window_width'] - 2*pad) / max(max_x - min_x, 1),
                    (self.config['window_height'] - 2*pad) / max(max_y - min_y, 1))
        return ((min_x + max_x) / 2, (min_y + max_y) / 2, scale)

    def generate_next_generation(self):
        new = []
        for t in self.toothpicks:
```

```
            for ep in t.get_endpoints():
                if ep in self.used_endpoints:
                    continue
                if sum(1 for o in self.toothpicks if ep in o.get_endpoints()) == 1:
                    new_t = Toothpick(ep[0], ep[1],
                                      self.config['toothpick_length'],
                                      'H' if t.direction == 'V' else 'V')
                    if new_t not in self.toothpicks and new_t not in new:
                        new.append(new_t)
                        self.used_endpoints.add(ep)
        self.toothpicks.extend(new)
        self.current_generation += 1

    def draw(self):
        self.screen.fill(tuple(self.config['background_color']))
        if self.config['auto_zoom_enabled']:
            ox, oy, scale = self.calculate_zoom(self.get_bounds())
            if not hasattr(self, 'current_offset_x'):
                self.current_offset_x, self.current_offset_y, self.current_scale = ox, oy, scale
            else:
                zs = self.config['zoom_speed']
                self.current_offset_x += (ox - self.current_offset_x) * zs
                self.current_offset_y += (oy - self.current_offset_y) * zs
                self.current_scale += (scale - self.current_scale) * zs
            ox = self.current_offset_x - self.config['window_width'] / (2 * self.current_scale)
            oy = self.current_offset_y - self.config['window_height'] / (2 * self.current_scale)
            scale = self.current_scale
        else:
            ox, oy, scale = 0, 0, 1.0

        for t in self.toothpicks:
            t.draw(self.screen, tuple(self.config['toothpick_color']),
```

```
                self.config['toothpick_thickness'], ox, oy, scale)

        txt = self.font.render(f'Gen: {self.current_generation} | Count: {len(self.toothpicks)}',
                               True, (255, 255, 0))
        self.screen.blit(txt, (10, 10))
        pygame.display.flip()

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE and self.current_generation < self.config['max_generations']:
                    self.generate_next_generation()
                elif event.key == pygame.K_r:
                    self.reset_fractal()
                elif event.key == pygame.K_ESCAPE:
                    self.running = False

    def run(self):
        while self.running:
            self.handle_events()
            if self.current_generation < self.config['max_generations']:
                self.frame_counter += 1
                if self.frame_counter >= self.config['generation_delay']:
                    self.generate_next_generation()
                    self.frame_counter = 0
            self.draw()
            self.clock.tick(self.config['fps'])
        pygame.quit()
```

**config.json**

```
json
```

```
{
  "window_width": 1000, "window_height": 1000, "fps": 30,
  "background_color": [0, 0, 0], "toothpick_color": [255, 255, 255],
  "toothpick_length": 20, "toothpick_thickness": 2,
  "max_generations": 15, "generation_delay": 30,
  "auto_zoom_enabled": true, "zoom_padding": 50, "zoom_speed": 0.05
}
```