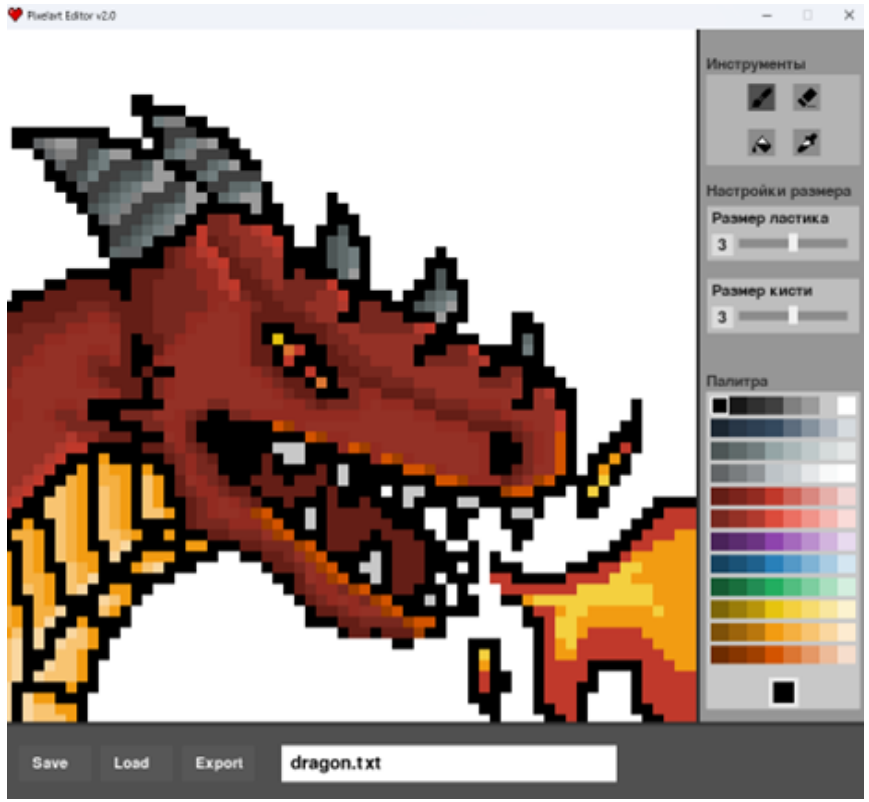
Рефакторинг процедурного подхода программирования к ООП с целью повышения производительности

Курсовой проект по ПМ11: Проверка работоспособности рефакторинга программного кода

Выполнил: Чернов Е.И.
Группа: 4ПОВТ-19Б

Преподаватель: Миллер Владимир Владимирович



ЧУ «Колледж предпринимательства КИнЭУ»

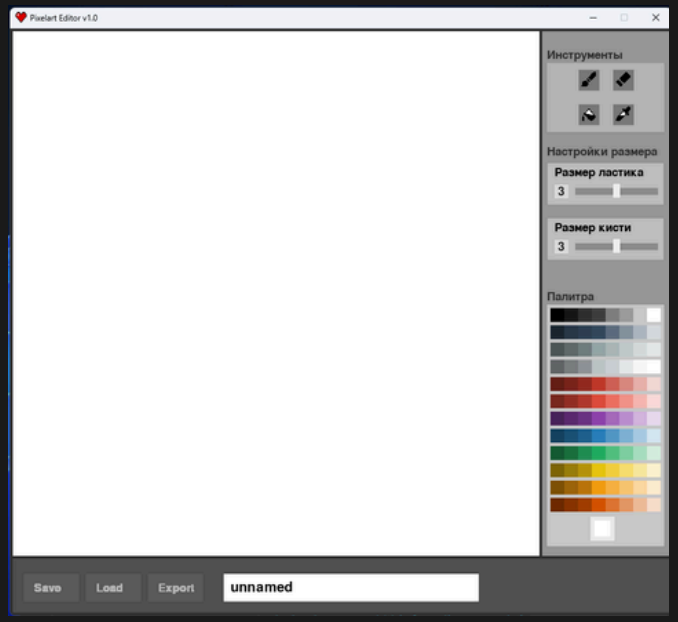
Проблемы кода

- **Монолитность**: Вся логика в одном файле main.py (609 строк)
- **Глобальное состояние:** Более 30 глобальных переменных
- **Дублирование:** До 25% кода повторяется
- **Длинные функции:** Главный цикл > 100 строк
- **Низкая модульность:** Сложность расширения и тестирования
- **Высокая связанность:** Компоненты зависят друг от друга

Цель работы

Комплексный рефакторинг графического редактора Pixelart Editor с переходом к объектно-ориентированному подходу для повышения:

- ✓ Читаемости кода
- ✓ Масштабируемости системы
- ✓ Производительности приложения



Теоретическая основа

Принципы ООП и паттерны проектирования

Четыре принципа ООП

- **Инкапсуляция**: Сокрытие внутренней реализации объекта и контролируемый доступ к данным
- **Наследование**: Создание иерархий классов для повторного использования кода
- **Полиморфизм**: Работа с объектами разных типов через единый интерфейс
- **Абстракция**: Выделение существенных характеристик объекта, игнорируя детали

Используемые паттерны

- **MVC**: Разделение ответственности на Модель, Представление, Контроллер
- **Strategy**: Инструменты рисования как взаимозаменяемые стратегии
- **Command**: Инкапсуляция операций для Undo/Redo функциональности
- **Observer**: Автоматическое уведомление об изменении состояния
- **Singleton**: Единственный экземпляр координирующего компонента

Результат проектирования

Модульная объектно ориентированная структура

Декомпозиция и модули

Монолитный файл main.py (609 строк) декомпозирован на 32 специализированных модуля организованных в 6 логических групп

core
Ядроприложения и конфигурация

models
Моделиданных (Grid, Palette)

tools
Инструменты рисования

controllers
Обработкаввода и координация

ui
Компоненты интерфейса

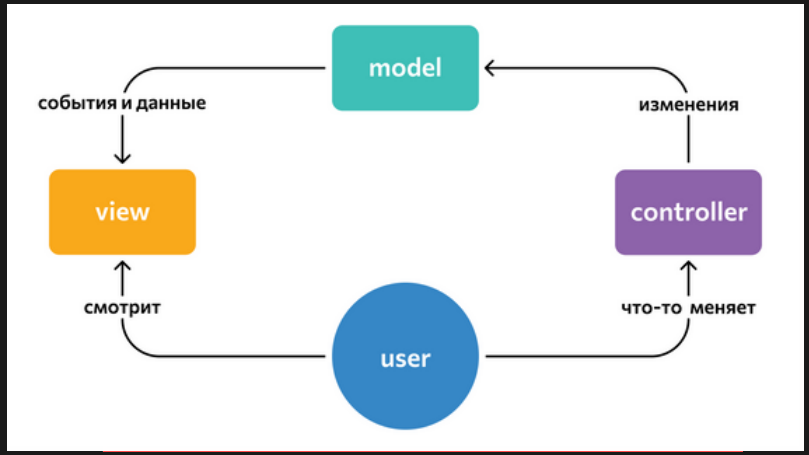
utils
Утилиты и история изменений

Ключевые компоненты

Grid: Инкапсулирует двумерный массив пикселей и методы манипуляции холстом

Controller: Обрабатывает пользовательский ввод и координирует Модель и Представление

View: Отвечает за отрисовку интерфейса и холста на экран



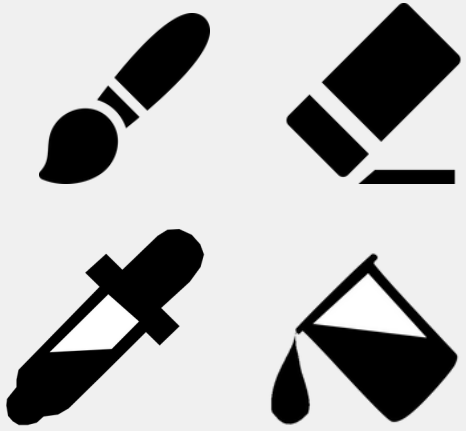
Реализация паттернов

Решения архитектурных проблем через паттерны проектирования

Strategy

Инструменты рисования (Кисть, Ластик, Линия) реализованы как отдельные стратегии, реализующие единый интерфейс.

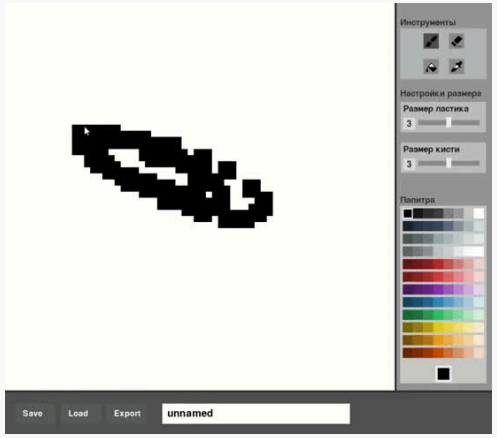
- ✓ Легко добавлять новые инструменты
- ✓ Изменение поведения во время выполнения
- ✓ Избегаем условной логики
- ✓ Повышаем переиспользуемость кода



Command

Каждое действие инкапсулируется как объект-команда, что позволяет реализовать функциональность Undo/Redo.

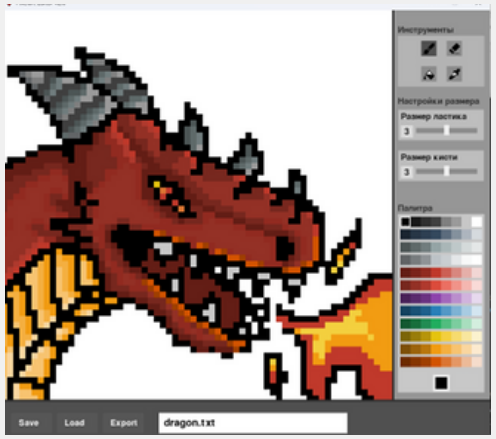
- ✓ Отмена и повтор операций
- ✓ История изменений
- ✓ Отложенное выполнение
- ✓ Очередь команд



Observer

Обеспечивает слабую связанность между Моделью и Представлением через механизм подписки на события.

- ✓ Автоматическое обновление UI
- ✓ Слабая связанность компонентов
- ✓ Реактивная архитектура
- ✓ Легче тестировать



Эффективность рефакторинга

Повышение качества и производительности

Качественные результаты

- ✓ **Читаемость**: Улучшена за счет модульности и инкапсуляции
- ✓ **Масштабируемость**: Упрощено добавление новой функциональности
- ✓ **Тестируемость**: Возможность изолированного модульного тестирования
- ✓ **Поддерживаемость**: Снижен технический долг

Количественные результаты

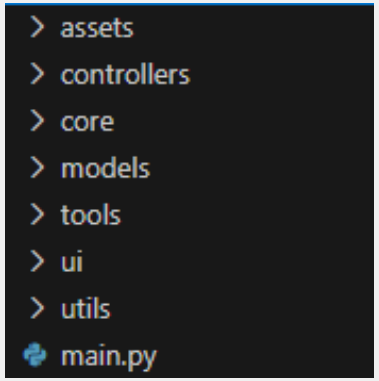
Модули : 1 файл → 32 специализированных модуля

Глобальные переменные: 30+ → 0

Дублирование кода: 25% → < 5%

Соответствие SOLID

- ✓ S: Single Responsibility — каждый класс имеет одну ответственность
- ✓ O: Open/Closed — открыто для расширения, закрыто для модификации
- ✓ L: Liskov Substitution — правильная иерархия наследования
- ✓ I: Interface Segregation — узкие, специализированные интерфейсы
- ✓ D: Dependency Inversion — зависимость от абстракций



Переход к ООП создал гибкую, поддерживаемую и производительную архитектуру