МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра игровой индустрии (ИИ)

КУРСОВАЯ РАБОТА

по дисциплине: Инструментальные средства разработки программного обеспечения с открытым исходным кодом
по профилю: Проектирование и разработка сред и приложений дополненной и виртуальной реальностей
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Разработка инструмента для процедурной генерации 2D-тайлсетов.

Студент: Солдатова Полина Сергеевна
Группа: ИКБО-30-24
Работа представлена к защите_____(дата)_____/Солдатова П.С./
(подпись и ф.и.о. студента)

Руководитель: старший преподаватель, Коваленко Михаил Андреевич

Работа допущена к защите_____(дата)_____/Коваленко М.А./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2025

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра игровой индустрии (ИИ)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Инструментальные средства разработки программного обеспечения с открытым исходным кодом

Студент: Солдатова Полина Сергеевна
Группа: ИКБО-30-24
Срок представления к защите: 01.12.2025
Руководитель: Коваленко Михаил Андреевич, старший преподаватель

Тема: Разработка инструмента для процедурной генерации 2D-тайлсетов

Исходные данные: Git, нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18, ГОСТ 7.32-2017.

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области разрабатываемого веб-приложения. 2. Обосновать выбор лицензии и технологий разработки программного обеспечения с открытым исходным кодом. 3. Разработать приложения на основе выбранного стека и лицензии. 4. Протестировать созданное приложение. 5. Загрузить разработанное приложение в открытый репозиторий с readme описанием проекта на удобный веб-сервис с системой контроля версий. 6. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

И.О. Зав. каф. ИИ:	_____	/ П.В. Беляев /,	«19» сентября 2025 г.
Задание на КР выдал:	_____	/ М.А.Коваленко /,	«19» сентября 2025 г.
Задание на КР получил:	_____	/ П.С. Солдатова /,	«19» сентября 2025 г.

Оглавление

АНОТАЦИЯ

Цель работы — разработать приложение, позволяющее генерировать уникальные тайлсеты на основе заданных параметров: размера, цветовой палитры, уровня детализации. Приложение поддерживает генерацию как единичного тайла, так и сетки тайлов, а также экспорт результатов в нескольких графических форматах.

Основные результаты:

1. Проведён анализ существующих аналогов. Выявлены их сильные и слабые стороны.
2. Сформированы функциональные и нефункциональные требования, включая требования к производительности, удобству интерфейса и кроссплатформенности.
3. Обоснован выбор стека технологий: язык Python, библиотеки Pillow для работы с изображениями и Tkinter для создания GUI
4. Определена лицензия MIT как оптимальная для проекта
5. Разработана архитектура приложения, состоящая из трёх основных модулей: app.py (интерфейс), generator.py (генерация тайлов), saved.py (сохранение результатов).
6. Реализован алгоритм генерации тайлов на основе обработки спрайт-листа с применением параметров пользователя: выбор палитры, размера тайлов, уровня детализации.
7. Проведено функциональное тестирование, подтвердившее корректность работы приложения и обработку ошибочных сценариев ввода.

Объём работы: курсовая работа содержит 40 страниц, включает 52 рисунка и 15 таблиц.

ВВЕДЕНИЕ

Актуальность

В современной игровой индустрии стоит проблема трудоёмкости создания текстур и окружения, так как их разработка требует большого количества времени, средств и человеческого ресурса. Особенно эта проблема остра для инди разработчиков и малых студий, которые не имеют возможности вкладывать столько ресурсов. Данный проект же направлен на создание доступного, open-source инструмента для процедурной генерации тайлсетов, который может быть использован для игровой разработки и облегчить процесс создания игровых продуктов.

Цель работы

Целью работы является разработка приложения для процедурной генерации 2D тайлсетов с графическим интерфейсом. Приложение должно создавать уникальные текстуры на основе заданных пользователем параметров(размер, палитра и детализация), иметь возможность создания единичного тайла или сетки тайлсетов, экспортировать созданные тайлсеты в нескольких форматах на выбор пользователя.

Объект исследования

Объектом исследования являются алгоритмы процедурной генерации изображений, а также методы создания пользовательского интерфейса.

Используемые методы

По большей части в данной работе используется сравнительный анализ, дополненный собственными решениями из литературы и документации библиотек

Глава 1

1.1 Обзор аналогов и анализ их сильных и слабых сторон

Таблица 1 – обзор Procedural Tileset Generator (https://donitz.itch.io/procedural-tileset-generator)

S Бесплатность Доступность из браузера Есть возможность выбора палитры, размера тайлов и уровня детализации Простой интерфейс	W Ограниченный функционал Требуется подключение к интернету
O Популярность среди небольших студий и инди-разраотчиков	T Появление более мощных бесплатных приложений

Таблица 2 – обзор Tilesetter (https://led.itch.io/tilesetter)

S Широкий функционал Возможность создания полноценных игровых карт прямо в приложении	W Высокая стоимость Работа в программе требует времени на изучение функций Закрытый исходный код
O Готовность профессиональных студий платить за качественный софт	T Пиратское распространение Конкуренция со стороны бесплатных приложений

1.2.1. Функциональные требования

- Пользователь должен иметь возможность настройки цветовой палитры.
- Пользователь должен иметь возможность сохранения созданного тайлсета.
- Пользователь должен иметь возможность выбора размера тайлсетов.
- Пользователь должен иметь возможность просматривать предварительный результат.
- Пользователь должен иметь возможность выбора детализации тайлов.
- Пользователь должен иметь возможность сбрасывать настройки к настройкам по умолчанию.
- Программа должна иметь возможность генерации как единичного тайла, так и сетки тайлов.

1.2.2. Нефункциональные требования:

- Генерация должна занимать не более 5 секунд.
- Интерфейс должен быть интуитивно понятен.
- Приложение должно иметь открытый код.
- Приложение должно корректно обрабатывать ошибки.
- Приложение должно работать на Linux и Windows 10/11

1.3. Ограничения

- Используются только библиотеки с открытым исходным кодом.
- Размер генерируемого тайла ограничен 32x32
- Нет возможности работы на мобильных устройствах

1.4. Выбор стека технологий

Таблица 3 – выбор языка разработки

Критерий	Вес	Python	C++
Скорость	1,5	2	3
Количество библиотек	2,5	3	2
Приспособленность для изображений	3	3	2
Простота создания UI	2	3	1
Кросплатформенность	1,5	3	2
Личное предпочтение	1	3	2
		33	22,5

1.4.1. Описание критериев и обоснование их веса

Скорость (1,5) - в требованиях прописана скорость генерации тайлсетов не более 5 секунд, так что скорость в данном контексте не так критична

Количество библиотек (2,5) - для работы с цветом изображением наличие большого количества библиотек сильно сократит время разработки

Приспособленность для изображения (3) - главная функция приложения - создание и работа с различного вида изображениями, поэтому максимальный приоритет именно на этот критерий

Простота создания UI (2) - интерфейс должен быть интуитивно понятен и лёгок в создании, так как именно через него происходит взаимодействие пользователя и самой программы

Кроссплатформенность (1,5) - поддержка должна быть только на Linux и Windows, поэтому большая кроссплатформенность не требуется

Личное предпочтение (1) - субъективный фактор для личного комфорта)

1.4.2. Обоснование баллов по языкам

Таблица 4 - скорость

Python (2)	C++ (3)
Интерпретируемый язык, в целом медленнее компилируемых. Но для алгоритмов процедурной генерации с скорость будет достаточна.	Компилируемый язык, дающий максимальную производительность. Позволит гарантировать выполнение требования "не более 5

	секунд" даже для сложных вычислений.

Таблица 5 - количество библиотек:

Python (3)	C++ (2)
Огромный выбор: Pillow для изображений и PyQt/PySide для GUI. Также эти библиотеки простые в использовании.	Библиотеки есть (OpenCV для изображений, SFML/Qt для GUI), но их интеграция сложнее. Также их документация может быть менее дружелюбной.

Таблица 6 - приспособленность для изображения:

Python (3)	C++ (2)
Библиотека Pillow - стандарт для простой работы с изображениями. Позволяет в несколько строк кода загружать, создавать, модифицировать и сохранять изображения в десятках форматов. Идеально подходит для задачи.	OpenCV — мощнейший фреймворк, но его избыточность для простой генерации тайлов для данного проекта является недостатком, занимая лишнюю память.

Таблица 7 - постота создания UI:

Python (3)	C++ (1)
PyQt/PySide позволяют быстро создать интуитивный интерфейс. Tkinter идет в комплекте с Python и	Qt для C++ — это стандарт, но порог входа и время на сборку простого интерфейса значительно выше.

это идеально соответствует требованию "только библиотеки с открытым исходным кодом".	

Таблица 8 - кроссплатформенность:

Python (3)	C++ (2)
Язык и его библиотеки по умолчанию кроссплатформенны.	Qt обеспечивает кроссплатформенность, но требует настройки системы сборки и компиляции под каждую платформу

Таблица 9 - личное предпочтение:

Python	C++
В целом более долгий срок изучения этого языка. Чувствую себя при написании на нём более комфортно	Умею писать, но синтаксис менее приятен

Итог: Python является более предпочтительным выбором для данного конкретного проекта. Он набирает больше баллов в ключевых для проекта категориях

1.5. Выбор лицензии

Таблица 10 – выбор лицензии

Критерий	Вес	MIT	BSD	GPL v3
Образовательная доступность	3	3	3	1,5
Простота соблюдения	2,5	3	2,5	1
Коммерческая совместимость	2	3	3	1
Защита авторства	2	2	2,5	3
Экосистемная совместимость	1,5	3	3	1
		31	30,75	16,5

1.5.1. Описание критериев и обоснование их веса

Образовательная доступность (3) - проект создаётся в учебном заведении и должен быть максимально доступен студентам, преподавателям, другим ВУЗам

Простота соблюдения (2,5) - лицензия должна быть понятной без специальных знаний

Коммерческая совместимость (2) - проект может использоваться инди-разработчиками и малыми студиями для реальных проектов без ограничений

Защита авторства (2) – защита работы важна, но не является основной целью образовательного проекта

Экосистемная совместимость (1,5) - лицензия должна хорошо сочетаться с другими open-source библиотеками, используемыми в проекте

1.5.2. Обоснование баллов по языкам

Таблица 11 – образовательная доступность

MIT (3)	BSD (3)	GPL v3 (1,5)
Позволяет любое учебное использование: копирование для курсовых, модификация, включение в учебные материалы. Нет ограничений для	Идентична MIT для образования. Разрешает все формы учебного использования	Ограничивает использование: требует открытия исходного кода производных работ, что может мешать в учебных проектах с элементами закрытой разработки

студентов и преподавателей		

Таблица 12 – простота соблюдения

MIT (3)	BSD (2,5)	GPL v3 (1)
21 строка текста, единственное требование, это сохранить копирайт. Понятна без юридических знаний	Немного длиннее MIT, добавлен пункт о запрете использования имён авторов, что усложняет понимание	55+ страниц сложного юридического текста. Требует понимания терминов

Таблица 13 – коммерческая совместимость

MIT (3)	BSD (3)	GPL v3 (1)
Полностью разрешено коммерческое использование, включение в проприетарное ПО без ограничений	Аналогично MIT полная свобода коммерческого использования	Запрещает включение в проприетарное ПО. Все производные работы должны быть открыты под GPL v3

Таблица 14 – защита авторства

MIT (2)	BSD (2,5)	GPL v3 (3)
Только базовая защита	Дополнительная защита. Запрет на использование имён авторов для рекламы без разрешения	Максимальная защита. Требует открытия исходного кода всех производных работ

Таблица 15 – экосистемная совместимость

MIT (3)	BSD (3)	GPL v3 (1)
Высокая совместимость. Можно комбинировать с кодом под любой лицензией, включая проприетарные.	Аналогично MIT. Совместима практически со всеми лицензиями	Низкая совместимость. Можно комбинировать только с GPL-совместимыми лицензиями

Итог: MIT является оптимальным выбором для данного проекта

1.6. Пользовательский интерфейс

Главное окно (рисунок 1)

Рисунок 1

Главное окно с развёрнутым списком палитр (рисунок 2)

Рисунок 2

Окно сохранения тайлсета (рисунок 3)

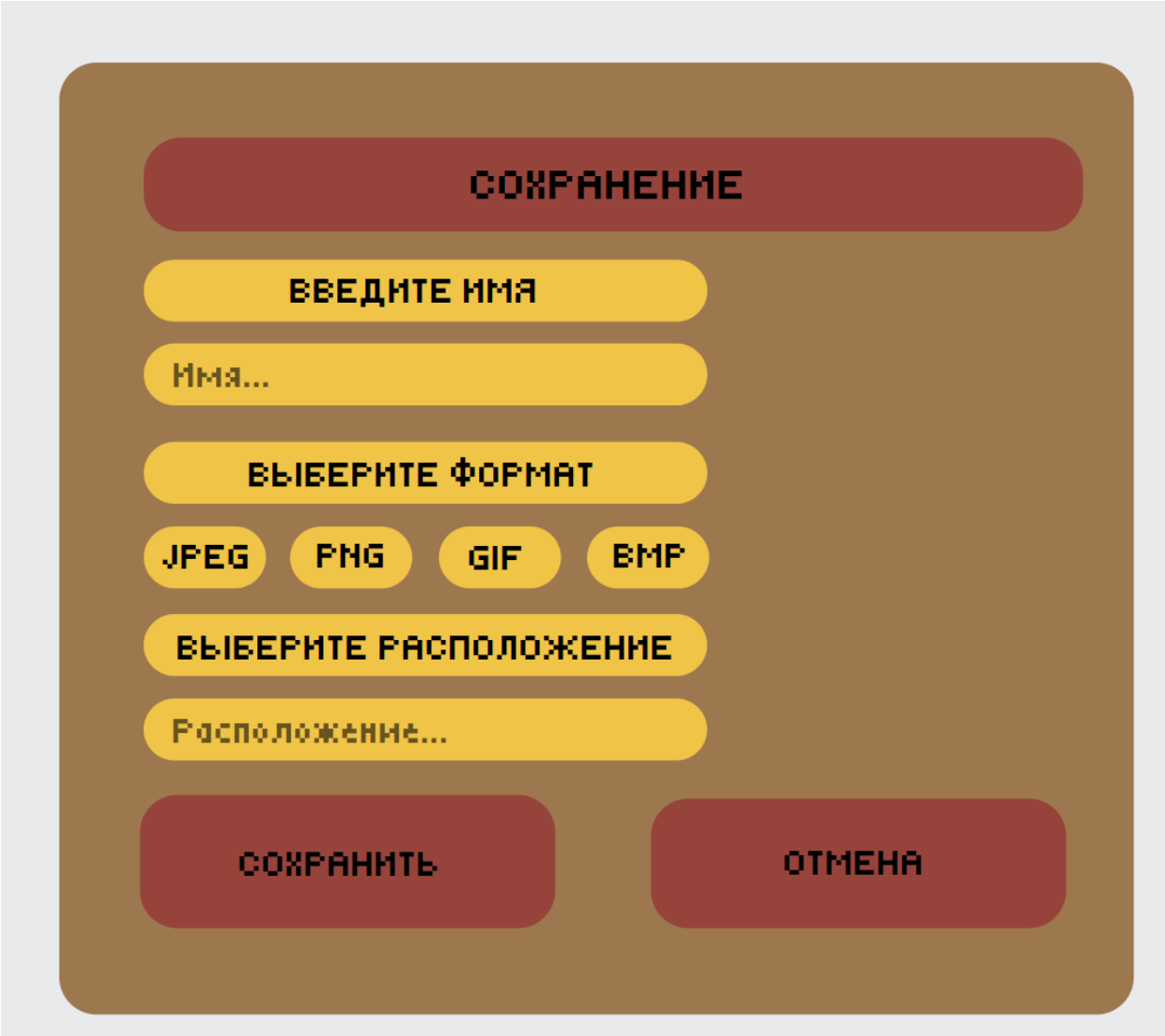

Рисунок 3

Глава 2

2.1. Описание принципа работы

Главным блоком ПО является сам генератор. Рассмотрим пошагово принцип его работы.

Загрузка и подготовка исходных данных

Имеется лист спрайтов размером 1724x1411 пикселей, содержащий 11x9 спрайтов. Узоры спрайтов используются для создания тайлов. Из спрайти листа берутся отдельные спрайты размером 156x156 пикселей. Для получения тайлов меньшего размера используется зум и вырезается центральная часть спрайта.

Параметризация процесса

Используются параметры, введённые пользователем

Алгоритм преобразования спрайта

Происходит попиксельное считывание данных картинки в формате RGB. Оценивается его яркость и зависимости от того неё на новом холсте новым цветом, выбранным из палитры, ставится новый пиксель. Уровень детализации определяет размер кисти (1, 2 или 4 пикселя).

Формирование тайлсета

Генерируется сетка тайлов указанного размера. Это и есть готовый тайлсет, который выводится пользователю.

2.2. Структура ПО

Приложение состоит из 3 блоков и png файла (рисунок 4)

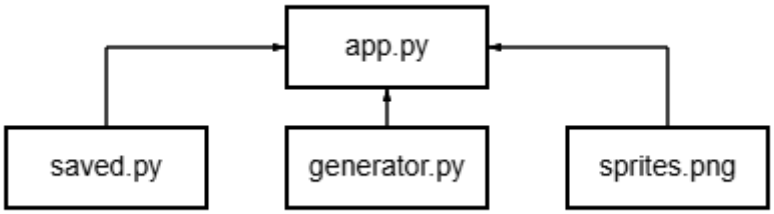

Рисунок 4

app.py – основной блок. Содержит вызов функций генерации и сохранения, графический пользовательский интерфейс с классами кнопок, раскрывающегося списка и полей ввода.

generator.app – блок генерации тайлсета. Содержит функции работы со спрайтами и генерации единичного тайла, а так же сетки тайлов с опорой на спрайт

saved.py – блок сохранения готового тайлсета. Содержит функцию перевода в нужный формат и самого сохранения

sprites.png – файл спрайтов, используемых для создания узора тайлов

Глава 3

3.1. Описание классов и методов

3.1.1. app.py

Функции класса SelectableButton: - класс кнопок главного окна

Конструктор класса (рисунок 5)

```
def __init__(self, parent, text, x, y, group=None, width=250, height=60):
    self.selected = False
    self.group = group
    self.text = text
    self.button = tk.Label(
        parent,
        text=text,
        bg='#EFC141',
        fg='black',
        font=('Arial', 16),
        relief='flat',
    )
    self.button.place(x=x, y=y, width=width, height=height)
    self.button.bind('<Button-1>', self.select)
    self.button.bind('<Enter>', self.on_hover)
    self.button.bind('<Leave>', self.on_leave)

    if group == 1:
        all_buttons_group1.append(self)
    elif group == 2:
        all_buttons_group2.append(self)
    elif group == 3:
        all_buttons_group3.append(self)
```

Рисунок 5

Функция выбора кнопки (рисунок 6)

```
def select(self, event=None):
    if self.group == 1:
        for btn in all_buttons_group1:
            btn.deselect()
    elif self.group == 2:
        for btn in all_buttons_group2:
            btn.deselect()
    elif self.group == 3:
        for btn in all_buttons_group3:
            btn.deselect()

    self.selected = True
    self.button.config(
        relief='flat',
        bg='#CCA243',
        highlightbackground='#5E0C32',
        highlightthickness=2
    )
```

Рисунок 6

Функция снятия выбора с кнопки (рисунок 7)

```
def deselect(self):
    self.selected = False
    self.button.config(
        relief='flat',
        bg='#EFC141',
        highlightthickness=0
    )
```

Рисунок 7

Функции обработки наведения и ухода курсора (рисунок 8)

```
def on_hover(self, event):
    if not self.selected:
        self.button.config(bg='#CCA243')

def on_leave(self, event):
    if not self.selected:
        self.button.config(bg='#EFC141')
```

Рисунок 8

Функция получения значения кнопки (рисунок 9)

```
def get_value(self):
    return self.text
```

Рисунок 9

Функции класса DropdownMenu: - класс выпадающего меню

Конструктор класса (рисунок 10)

```
def __init__(self, parent, x, y, width=790, height=60):
    self.parent = parent
    self.is_open = False
    self.width = width
    self.height = height
    self.selected_palette = None
    self.selected = False

    self.button = tk.Label(
        parent,
        text="посмотреть ещё...",
        bg="#EFC141",
        fg="black",
        font=("Arial", 16),
        relief="flat"
    )
    self.button.place(x=x, y=y, width=width, height=height)
    self.button.bind("<Button-1>", self.toggle)
    self.button.bind('<Enter>', self.on_hover)
    self.button.bind('<Leave>', self.on_leave)

    all_buttons_group2.append(self)

    self.frame = tk.Frame(parent, bg="#CCA243", relief="flat", pady=20)
    self.items = []

    extra_names = [
        "город", "пляж", "ледник",
        "горы", "пещера", "болото",
        "закат","неон","радуга",
        "космос","вулкан",
    ]

    self.build_items(extra_names)
```

Рисунок 10

Функция динамического построения меню (рисунок 11)

```
def build_items(self, names):
    for it in self.items:
        try:
            it.button.destroy()
        except:
            pass
    self.items.clear()
    row = 0
    col = 0
    max_cols = 3
    for name in names:
        btn = SelectableButton(
            self.frame,
            name,
            x=col * 270,
            y=row * 80,
            group=2,
            width=250,
            height=60
        )
        btn.button.bind("<Button-1>", lambda e, n=name: self.select_from_dropdown(n))
        self.items.append(btn)
        col += 1
        if col >= max_cols:
            col = 0
            row += 1
    total_rows = (len(names) + max_cols - 1) // max_cols
    self.frame_width = (max_cols * 270) - 20
    self.frame_height = total_rows * 80 + 20
```

Рисунок 11

Функция выбора палитры из меню (рисунок 12)

```
def select_from_dropdown(self, palette_name):
    for btn in all_buttons_group2:
        btn.deselect()
    self.selected_palette = palette_name
    self.button.config(
        text=palette_name,
        bg='#CCA243',
        highlightbackground='#5E0C32',
        highlightthickness=2
    )
    self.close()
    self.selected = True
```

Рисунок 12

Функции выбора и сброса меню (рисунок 13)

```
def select(self, event=None):
    for btn in all_buttons_group2:
        if btn != self:
            btn.deselect()

    self.selected = True
    self.button.config(
        bg='#CCA243',
        highlightbackground='#5E0C32',
        highlightthickness=2
    )

def deselect(self):
    self.selected = False
    self.selected_palette = None
    self.button.config(
        bg='#EFC141',
        highlightthickness=0,
        text="посмотреть ещё..."
    )
    self.close()
```

Рисунок 13

Функции обработки наведения и ухода курсора (рисунок 14)

```
def on_hover(self, event):
    if not self.selected:
        self.button.config(bg='#CCA243')

def on_leave(self, event):
    if not self.selected:
        self.button.config(bg='#EFC141')
```

Рисунок 14

Функция переключения состояния меню (рисунок 15)

```
def toggle(self, event=None):
    if self.is_open:
        self.close()
    else:
        self.open()
```

Рисунок 15

Функции открытия и закрытия выпадающего меню (рисунок 16)

```
def open(self):
    self.is_open = True
    btn_x = self.button.winfo_x()
    btn_y = self.button.winfo_y()
    self.frame.place(
        x=btn_x,
        y=btn_y + self.height,
        width=self.frame_width,
        height=self.frame_height
    )
    self.frame.lift()

def close(self):
    self.is_open = False
    self.frame.place_forget()
```

Рисунок 16

Функция получения текущего значения меню (рисунок 17)

```
def get_value(self):
    if self.selected_palette:
        return self.selected_palette
    elif self.button.cget('text') != "посмотреть ещё...":
        return self.button.cget('text')
    return None
```

Рисунок 17

Функция вывода тайлсета (рисунок 18)

```
def display_tileset(image): # Функция вывода тайлсета
    try:
        tileset_canvas.delete("all")
        display_width = 400
        display_height = 400
        # Получаем размер оригинального изображения
        img_width, img_height = image.size
        # Сохраняем пропорции
        scale_x = display_width / img_width
        scale_y = display_height / img_height
        scale = min(scale_x, scale_y)
        new_width = int(img_width * scale)
        new_height = int(img_height * scale)
        # Масштабируем с NEAREST для пиксельной графики
        image_resized = image.resize((new_width, new_height), Image.Resampling.NEAREST)
        # Конвертируем в PhotoImage
        photo = ImageTk.PhotoImage(image_resized)
        # Центрируем изображение
        x_offset = (380 - new_width) // 2
        y_offset = (380 - new_height) // 2
        # Отображаем на canvas
        tileset_canvas.create_image(
            x_offset + new_width // 2,
            y_offset + new_height // 2,
            image=photo,
            anchor='center'
        )
        # Сохраняем ссылку
        tileset_canvas.image = photo
        return True

    except Exception as e:
        print(f"Ошибка при отображении тайлсета: {e}")
        # Показываем ошибку
        tileset_canvas.create_text(
            190, 190,
            text="Ошибка отображения",
            fill='red',
            font=('Arial', 12)
        )
        return False
```

Рисунок 18

Функция сброса всех настроек интерфейса к исходному состоянию (рисунок 19)

```
def reset_settings(): # Функция сброса кнопок
    for btn in all_buttons_group1:
        btn.deselect()
    for btn in all_buttons_group2:
        btn.deselect()
    for btn in all_buttons_group3:
        btn.deselect()
    # Сброс меню
    more_palettes_btn.deselect()
    # Очищаем canvas
    tileset_canvas.delete("all")
    tileset_canvas.create_text(
        190, 190,
        text="Тайлсет не сгенерирован",
        fill='gray',
        font=('Arial', 14)
    )
    # Очищаем глобальную переменную
    global current_tileset
    current_tileset = None
    # Сбрасываем поля ввода
    rows_entry.delete(0, tk.END)
    cols_entry.delete(0, tk.END)
    rows_entry.insert(0, "3")
    cols_entry.insert(0, "3")
```

Рисунок 19

Функция сбора всех выбранных пользователем параметров из интерфейса (рисунок 20)

```
def get_selected_params():
    # Получаем выбранный размер
    selected_size = None
    size_map = {"8x8": 8, "16x16": 16, "32x32": 32}
    for btn in all_buttons_group1:
        if btn.selected:
            selected_size = size_map.get(btn.get_value())
            break
    # Получаем выбранную палитру
    selected_palette = None
    for btn in all_buttons_group2:
        if btn.selected:
            if hasattr(btn, 'get_value'):
                selected_palette = btn.get_value()
            else:
                selected_palette = btn.button.cget('text')
            break
    if not selected_palette:
        selected_palette = more_palettes_btn.get_value()
    # Получаем выбранный уровень детализации
    selected_detail = None
    for btn in all_buttons_group3:
        if btn.selected:
            selected_detail = btn.get_value()
            break
    # Проверяем, все ли выбрано
    if not selected_size:
        messagebox.showerror("Ошибка", "Выберите размер тайлов!")
        return None, None, None
    if not selected_palette:
        messagebox.showerror("Ошибка", "Выберите палитру!")
        return None, None, None
    if not selected_detail:
        messagebox.showerror("Ошибка", "Выберите уровень детализации!")
        return None, None, None
    return selected_size, selected_palette, selected_detail
```

Рисунок 20

Функция генерации тайлсета (рисунок 21)

```
def create_tileset():# Основная функция
    global current_tileset
    # Проверяем ввод размера сетки
    rows, cols = validate_input(rows_entry, cols_entry)
    if rows is None or cols is None:
        return
    # Получаем параметры
    params = get_selected_params()
    if params[0] is None:
        return
    selected_size, selected_palette, selected_detail = params
    try:
        # Генерируем тайлсет
        tileset = generator.generate_tileset(
            rows=rows,
            cols=cols,
            tile_size=selected_size,
            palette_name=selected_palette,
            detail_level=selected_detail
        )
        # Сохраняем в глобальную переменную
        current_tileset = tileset
        # Отображаем тайлсет
        display_tileset(tileset)
    except Exception as e:
        messagebox.showerror("Ошибка", f"Ошибка при генерации:\n{str(e)}")
```

Рисунок 21

Функция генерации тайлсета (рисунок 22)

```
def create_single_tile(): # Функция создания одного тайла
    global current_tileset
    # Получаем параметры
    params = get_selected_params()
    if params[0] is None:
        return
    selected_size, selected_palette, selected_detail = params
    try:
        # Генерируем один тайл
        tile = generator.generate_tileset(
            rows=1,
            cols=1,
            tile_size=selected_size,
            palette_name=selected_palette,
            detail_level=selected_detail
        )
        # Сохраняем в глобальную переменную
        current_tileset = tile
        # Отображаем тайл
        display_tileset(tile)
    except Exception as e:
        messagebox.showerror("Ошибка", f"Ошибка при генерации тайла:\n{str(e)}")
```

Рисунок 22

Функция сохранения тайлсета (рисунок 23)

```
def save_tileset_dialog(): # Сохранение тайлсета
    global current_tileset
    if current_tileset is None:
        messagebox.showerror("Ошибка", "Сначала создайте тайлсет!")
        return
    saved.save_image_dialog(root, current_tileset)
```

Рисунок 23

3.1.2. saved.py

Функции класса FormatSelectButton: - класс кнопок окна сохранения

Конструктор класса (рисунок 24)

```
def __init__(self, parent, text, x, y, width=120, height=50):
    self.selected = False
    self.text = text
    self.button = tk.Label(
        parent,
        text=text,
        bg='#EFC141',
        fg='black',
        font=('Arial', 14),
        relief='flat',
    )
    self.button.place(x=x, y=y, width=width, height=height)
    self.button.bind('<Button-1>', self.select)
    self.button.bind('<Enter>', self.on_hover)
    self.button.bind('<Leave>', self.on_leave)
```

Рисунок 24

Функция выбора кнопки (рисунок 25)

```
def select(self, event=None):
    for btn in format_buttons:
        btn.deselect()
    self.selected = True
    self.button.config(
        relief='flat',
        bg='#CCA243',
        highlightbackground='#5E0C32',
        highlightthickness=2
    )
```

Рисунок 25

Функция снятия выбора с кнопки (рисунок 26)

```
def deselect(self):
    self.selected = False
    self.button.config(
        relief='flat',
        bg='#EFC141',
        highlightthickness=0
    )
```

Рисунок 26

Функции обработки наведения и ухода курсора (рисунок 27)

```
def on_hover(self, event):
    if not self.selected:
        self.button.config(bg='#CCA243')

def on_leave(self, event):
    if not self.selected:
        self.button.config(bg='#EFC141')
```

Рисунок 27

Функция получения значения кнопки (рисунок 28)

```
def get_value(self):
    return self.text
```

Рисунок 28

Функция отображения диалогового окна сохранения (рисунок 29)

```
def save_image_dialog(parent, image, callback=None):
    dialog = SaveDialog(parent, image, callback)
    return dialog
```

Рисунок 29

Функция сохранения изображения (рисунок 30)

```
def save_image(image, filepath, format="PNG"):
    try:
        # Конвертируем для JPEG, если нужно
        if format.upper() == "JPEG" and image.mode != "RGB":
            image = image.convert("RGB")
        # Сохраняем изображение
        image.save(filepath, format=format.upper())
        return True
    except Exception as e:
        raise Exception(f"Ошибка при сохранении файла {filepath}: {str(e)}")
```

Рисунок 30

Класс SaveDialog: - класс диалогового окна сохранения

Конструктор класса (рисунок 31)

```
def __init__(self, parent, image, callback=None):
    self.parent = parent
    self.image = image
    self.callback = callback
    self.create_window()
    self.create_widgets()
```

Рисунок 31

Функция открытия и закрытия окна (рисунок 32)

```
def create_window(self): # Создание окна диалога
    self.window = tk.Toplevel(self.parent)
    self.window.title("Сохранение тайлсета")
    self.window.geometry("800x600")
    self.window.configure(bg='#987247')
    self.window.resizable(False, False)
    self.window.grab_set()
    self.window.focus_set()
    # Обработка закрытия окна
    self.window.protocol("WM_DELETE_WINDOW", self.on_cancel)
```

Рисунок 32

Функция создания виджетов окна (рисунки 35-36)

```
def create_widgets(self): # Создание виджетов в окне
    # Заголовок
    title_label = tk.Label(
        self.window,
        text="СОХРАНЕНИЕ",
        bg='#914638',
        fg='black',
        font=('Arial', 24, 'bold'),
        relief='flat'
    )
    title_label.place(x=0, y=0, width=800, height=80)

    # Имя файла
    name_label = tk.Label(
        self.window,
        text="ВВЕДИТЕ ИМЯ",
        bg='#987247',
        fg='black',
        font=('Arial', 16, 'bold')
    )
    name_label.place(x=50, y=100)

    self.filename_var = tk.StringVar(value="tileset")
    self.filename_entry = tk.Entry(
        self.window,
        textvariable=self.filename_var,
        font=('Arial', 16),
        bg='#EFC141',
        fg='black',
        relief='flat',
        justify='center'
    )
    self.filename_entry.place(x=50, y=140, width=670, height=50)

    # Метка расширения
    self.extension_label = tk.Label(
        self.window,
        text=".png",
        bg='#987247',
        fg='black',
        font=('Arial', 16)
    )
    self.extension_label.place(x=700, y=140, height=50)
```

Рисунок 33

```
# Формат файла
format_label = tk.Label(
    self.window,
    text="ВЫБЕРИТЕ ФОРМАТ",
    bg='#987247',
    justify='center',
    fg='black',
    font=('Arial', 16, 'bold')
)
format_label.place(x=50, y=220)

# Кнопки форматов
format_frame = tk.Frame(self.window, bg='#987247')
format_frame.place(x=50, y=260, width=700, height=60)

self.format_buttons = []
formats = ["JPEG", "PNG", "GIF", "BMP"]
format_width = 160

for i, fmt in enumerate(formats):
    x_pos = i * (format_width + 10)
    btn = FormatSelectButton(format_frame, fmt, x_pos, 0,
                             width=format_width, height=50)
    btn.button.bind('<Button-1>',
                    lambda e, b=btn: (self.select_format(b),
                                      self.update_extension()))
    self.format_buttons.append(btn)

# Выбираем PNG по умолчанию
for btn in self.format_buttons:
    if btn.text == "PNG":
        self.select_format(btn)
        break

# Выбор папки
location_label = tk.Label(
    self.window,
    text="ВЫБЕРИТЕ РАСПОЛОЖЕНИЕ",
    bg='#987247',
    fg='black',
    font=('Arial', 16, 'bold')
)
location_label.place(x=50, y=340)
```

Рисунок 34

```
self.folder_path_var = tk.StringVar(value="Расположение...")
location_button = tk.Label(
    self.window,
    textvariable=self.folder_path_var,
    bg='#EFC141',
    fg='black',
    font=('Arial', 16),
    relief='flat',
)
location_button.place(x=50, y=380, width=670, height=50)
location_button.bind('<Button-1>', lambda e: self.select_folder())
location_button.bind('<Enter>',
                     lambda e: location_button.config(bg='#CCA243'))
location_button.bind('<Leave>',
                     lambda e: location_button.config(bg='#EFC141'))

# Кнопки действий
buttons_frame = tk.Frame(self.window, bg='#987247')
buttons_frame.place(x=50, y=460, width=700, height=80)

# Кнопка Сохранить
save_btn = tk.Label(
    buttons_frame,
    text="СОХРАНИТЬ",
    bg='#913D36',
    fg='black',
    font=('Arial', 20, 'bold'),
    relief='flat',
)
save_btn.place(x=0, y=0, width=340, height=60)
save_btn.bind('<Button-1>', lambda e: self.perform_save())
save_btn.bind('<Enter>', lambda e: save_btn.config(bg='#A8453E'))
save_btn.bind('<Leave>', lambda e: save_btn.config(bg='#913D36'))

# Кнопка Отмена
cancel_btn = tk.Label(
    buttons_frame,
    text="ОТМЕНА",
    bg='#914638',
    fg='black',
    font=('Arial', 20, 'bold'),
    relief='flat',
)
cancel_btn.place(x=360, y=0, width=340, height=60)
```

Рисунок 35

```
# Кнопка Отмена
cancel_btn = tk.Label(
    buttons_frame,
    text="ОТМЕНА",
    bg='#914638',
    fg='black',
    font=('Arial', 20, 'bold'),
    relief='flat',
)
cancel_btn.place(x=360, y=0, width=340, height=60)
cancel_btn.bind('<Button-1>', lambda e: self.on_cancel())
cancel_btn.bind('<Enter>', lambda e: cancel_btn.config(bg='#A8453E'))
cancel_btn.bind('<Leave>', lambda e: cancel_btn.config(bg='#914638'))
```

Рисунок 36

Функция выбора формата файла (рисунок 37)

```
def select_format(self, selected_button):# Выбор формата файла
    for btn in self.format_buttons:
        btn.deselect()
    selected_button.select()
```

Рисунок 37

Функция обновления расширения (рисунок 38)

```
def update_extension(self): # Обновление расширения файла
    selected_format = None
    for btn in self.format_buttons:
        if btn.selected:
            selected_format = btn.get_value().lower()
            break

    if selected_format:
        if selected_format == "jpeg":
            self.extension_label.config(text=".jpg")
        else:
            self.extension_label.config(text=f".{selected_format}")
```

Рисунок 38

Функция выбора папки сохранения (рисунок 39)

```
def select_folder(self): # Выбор папки для сохранения
    folder = fd.askdirectory(title="Выберите папку для сохранения")
    if folder:
        self.folder_path_var.set(folder)
```

Рисунок 39

Функция выполнения сохранения (рисунок 40)

```
def perform_save(self):# Выполнение сохранения"""
    selected_format = None
    for btn in self.format_buttons:
        if btn.selected:
            selected_format = btn.get_value()
            break
    if not selected_format:
        messagebox.showerror("Ошибка", "Выберите формат файла!")
        return

    # Получаем имя файла
    filename = self.filename_var.get().strip()
    if not filename:
        messagebox.showerror("Ошибка", "Введите имя файла!")
        return

    # Получаем путь к папке
    folder_path = self.folder_path_var.get()
    if folder_path == "Расположение...":
        messagebox.showerror("Ошибка", "Выберите папку для сохранения!")
        return

    # Формируем полный путь
    if selected_format.lower() == "jpeg":
        extension = ".jpg"
    else:
        extension = f".{selected_format.lower()}"

    # Добавляем расширение
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp')):
        filename += extension

    full_path = os.path.join(folder_path, filename)

    try:
        # Сохраняем файл
        save_image(self.image, full_path, format=selected_format)
        messagebox.showinfo("Успех", f"Тайлсет сохранен:\n{full_path}")

        # Вызываем callback, если он есть
        if self.callback:
            self.callback(full_path)

        self.window.destroy()
```

Рисунок 40

Функция отмены сохранения (рисунок 41)

```
def on_cancel(self): # Обработка отмены
    self.window.destroy()
```

Рисунок 41

3.1.3. generator.py

Функция загрузки и подготовки спрайтов со спрайт листа (рисунок 42)

```python
def load_sprites_from_sheet_with_zoom(sheet_path, tile_size, zoom=0.5):
    sheet = Image.open(sheet_path).convert("RGBA")
    sheet_width, sheet_height = sheet.size  # 1724x1411

    # Примерный размер одного спрайта на листе
    sprite_on_sheet_width = sheet_width / 11  # 156 пикселей
    sprite_on_sheet_height = sheet_height / 9  # 156 пикселей

    sprites = []

    for row in range(9):
        for col in range(11):
            # Координаты центра спрайта
            center_x = col * sprite_on_sheet_width + sprite_on_sheet_width / 2
            center_y = row * sprite_on_sheet_height + sprite_on_sheet_height / 2

            # Размер области, которую мы вырезаем
            extract_size = int(sprite_on_sheet_width * zoom)

            # Если нужный tile_size меньше extract_size, то масштабируем потом
            left = int(center_x - extract_size / 2)
            top = int(center_y - extract_size / 2)
            right = int(center_x + extract_size / 2)
            bottom = int(center_y + extract_size / 2)

            # Проверяем границы
            left = max(0, left)
            top = max(0, top)
            right = min(sheet_width, right)
            bottom = min(sheet_height, bottom)

            box = (left, top, right, bottom)
            sprite = sheet.crop(box)

            # Масштабируем до нужного размера
            sprite = sprite.resize((tile_size, tile_size), Image.Resampling.NEAREST)
            sprites.append(sprite)
    return sprites
```

Рисунок 42

Функция создания текстуры на основе исходного спрайта (рисунок 43)

```
def apply_material_effects(sprite, palette, detail_level):
    size = sprite.size[0]
    result = Image.new("RGB", (size, size))
    base_color = random.choice(palette)
    pattern_color = random.choice([c for c in palette if c != base_color])
    element_size = max(1, {
        "низкий": size // 4,
        "средний": size // 8,
        "высокий": size // 16
    }[detail_level])

    for x in range(0, size, element_size):
        for y in range(0, size, element_size):
            sprite_x = min(x, sprite.width - 1)
            sprite_y = min(y, sprite.height - 1)
            pixel = sprite.getpixel((sprite_x, sprite_y))
            brightness = sum(pixel[:3]) / 3
            if brightness > 160:
                color = pattern_color
            elif brightness < 96:
                color = base_color
            else:
                mix = (brightness - 96) / (160 - 96)
                color = tuple(
                    int(base_color[i] * (1 - mix) + pattern_color[i] * mix)
                    for i in range(3)
                )
            for dx in range(element_size):
                for dy in range(element_size):
                    pos_x, pos_y = x + dx, y + dy
                    if pos_x < size and pos_y < size:
                        result.putpixel((pos_x, pos_y), color)
    return result
```

Рисунок 43

Функция генерации полного тайлсета (рисунок 44)

```
def generate_tileset(rows, cols, tile_size, palette_name, detail_level):
    sprites = load_sprites_from_sheet_with_zoom("sprites3.png", tile_size, zoom=0.3)
    palette = PALETTES.get(palette_name, PALETTES["океан"])
    tileset_img = Image.new("RGB", (cols * tile_size, rows * tile_size))
    for r in range(rows):
        for c in range(cols):
            base_sprite = random.choice(sprites)
            tile = apply_material_effects(base_sprite, palette, detail_level)
            tileset_img.paste(tile, (c * tile_size, r * tile_size))

    return tileset_img
```

Рисунок 44

3.2. Пользовательский интерфейс

Главный экран с созданным тайлсетом 3x3 (рисунок 45)

Рисунок 45

Окно сохранения (рисунок 46)

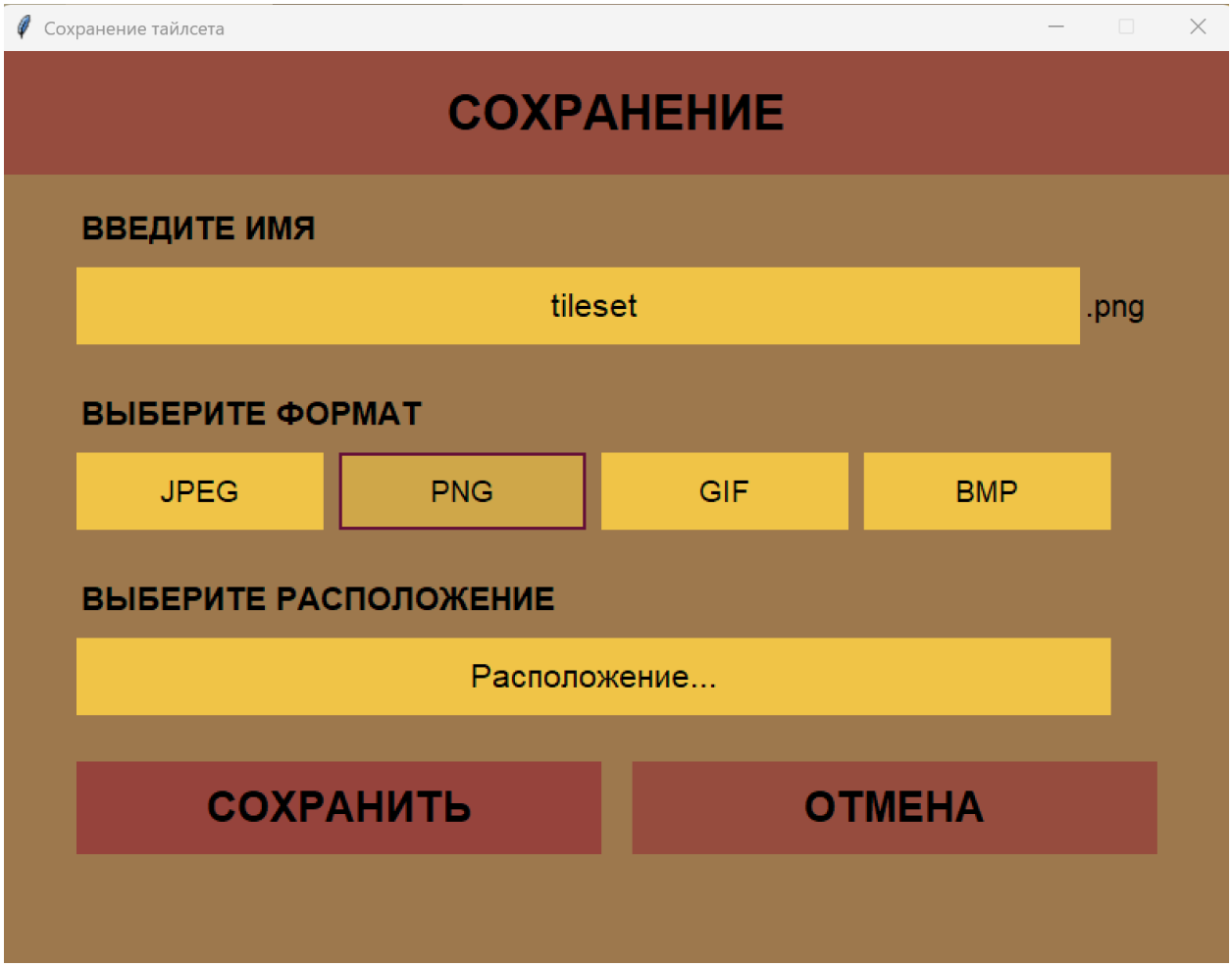

Рисунок 46

3.3. Тестирование

Тест 1. Успешный ввод (рисунок 47)

```
def test_1():
    rows_entry = MockEntry("3")
    cols_entry = MockEntry("3")
    result = app.validate_input(rows_entry, cols_entry)
    assert result is not None
    rows, cols = result
    class MockButton:
        def __init__(self, selected=False, text=""):
            self.selected = selected
            self.text = text
        def get_value(self):
            return self.text
    app.all_buttons_group1 = [MockButton(True, "16x16")]
    app.all_buttons_group2 = [MockButton(True, "океан")]
    app.all_buttons_group3 = [MockButton(True, "средний")]
    size, palette, detail = app.get_selected_params()
    if os.path.exists("sprites3.png"):
        img = generator.generate_tileset(rows, cols, size, palette, detail)
        assert img is not None
    test_img = Image.new('RGB', (50, 50), (255, 0, 0))
    with tempfile.NamedTemporaryFile(suffix='.png', delete=False) as tmp:
        filepath = tmp.name
    saved.save_tileset(test_img, filepath, format="PNG")
    assert os.path.exists(filepath)
    os.unlink(filepath)
```

Рисунок 47

Тест 2. Количество строк и столбцов не введено. Ожидает ошибку (рисунок 48)

```
def test_2():
    rows_entry = MockEntry("")
    cols_entry = MockEntry("")
    result = app.validate_input(rows_entry, cols_entry)
    assert result == (None, None)
```

Рисунок 48

Тест 3. Введены не числа. Ожидает ошибку (рисунок 49)

```
def test_3():
    rows_entry = MockEntry("abc")
    cols_entry = MockEntry("xyz")
    result = app.validate_input(rows_entry, cols_entry)
    assert result == (None, None)
```

Рисунок 49

Тест 4. Введены числа > 20. Ожидает ошибку (рисунок 50)

```
def test_4():
    rows_entry = MockEntry("25")
    cols_entry = MockEntry("25")
    result = app.validate_input(rows_entry, cols_entry)
    assert result == (None, None)
```

Рисунок 50

Тест 5. Не выбрана палитра. Ожидает ошибку (рисунок 51)

```
def test_5():
    class MockButton:
        def __init__(self, selected=False, text=""):
            self.selected = selected
            self.text = text
        def get_value(self):
            return self.text
    app.all_buttons_group1 = [MockButton(True, "16x16")]
    app.all_buttons_group2 = []
    app.all_buttons_group3 = [MockButton(True, "средний")]
    app.more_palettes_btn.selected_palette = None
    app.more_palettes_btn.button.cget.return_value = "посмотреть ещё..."

    result = app.get_selected_params()
    assert result == (None, None, None)
```

Рисунок 51

Результаты тестирования (рисунок 52)

```
Успешный сценарий: Тест пройден
Пустые поля: Тест пройден
Нечисловые значения: Тест пройден
Значения > 20: Тест пройден
Не выбрана палитра: Тест пройден
Все тесты пройдены
```

Рисунок 52

ЗАКЛЮЧЕНИЕ

Цель работы достигнута. Разработано приложение для процедурной генерации 2D-тайлсетов с графическим пользовательским интерфейсом. Приложение позволяет создавать уникальные текстуры на основе заданных пользователем параметров (размер, палитра, детализация), поддерживает генерацию как единичного тайла, так и сетки тайлов, а также экспорт результатов в нескольких графических форматах.

Поставленные, решены со следующими результатами:

1. Обзор и анализ существующих аналогов проведён, выявлены их преимущества и недостатки. Это позволило сформулировать требования к разрабатываемому инструменту.
2. Формулирование функциональных и нефункциональных требований выполнено, требования учтены в процессе разработки.
3. Выбор стека технологий и лицензии обоснован и реализован: выбран Python с библиотеками Pillow и Tkinter, лицензия MIT.
4. Проектирование архитектуры приложения выполнено, система разделена на логические модули (app.py, generator.py, saved.py).
5. Реализован интуитивно понятный интерфейс, хотя его визуальное исполнение не в полной мере соответствует первоначальному дизайн-макету, но остаётся функциональным и удобным для использования.
6. Реализация алгоритма генерации тайлов выполнена на основе обработки спрайт-листа с применением пользовательских параметров.
7. Тестирование приложения проведено, подтверждена работоспособность основных функций и обработка ошибочных сценариев.

Ссылка на открытый Git репозиторий

https://github.com/cabachokok/WinMax2025_Soldatova_v2?tab=readme-ov-file