**1. Введение**

Этот документ описывает процесс тестирования приложения TaskMngr.exe — диспетчера задач с графическим интерфейсом, разработанного с использованием tkinter и psutil. Цель тестирования — убедиться, что приложение корректно работает, обрабатывает ошибки и не вызывает сбоев в системе.

**2. Цели тестирования**

Проверка корректности работы интерфейса (GUI.py).

Тестирование получения данных о процессах (main.py).

Проверка работы кнопок управления процессами (обновление, завершение).

Тестирование вкладки "Performance" (графики загрузки CPU и памяти).

Проверка функционала мониторинга GPU.

Тестирование обработки ошибок и исключений.

**3. Методология тестирования**

Для тестирования использовались:

Ручное тестирование — запуск программы и проверка корректности работы.

Автоматическое тестирование — использование unittest для проверки основных функций.

Тестирование проводилось в средах:

Windows 10/11

Python 3.8

**4. Тестируемые модули**

| Модуль | Описание | Проверяемые функции |
| --- | --- | --- |
| main.py | Управление процессами | get\_process\_info(), start\_monitoring() |
| GUI.py | Графический интерфейс | refresh\_process\_list(), kill\_selected\_process(), update\_performance\_charts() |

**5. План тестирования**

5.1 Тестирование интерфейса

| Тест | Ожидаемый результат | Статус |
| --- | --- | --- |
| Запуск GUI.py | Окно программы открывается | Успешно |
| Обновление списка процессов | Процессы обновляются без зависаний | Успешно |
| Завершение процесса | Выбранный процесс завершается, отображается сообщение | Успешно |
| Открытие вкладки "Performance" | Отображаются графики загрузки CPU и памяти | Успешно |

5.2 Тестирование управления процессами (main.py)

| Тест | Ожидаемый результат | Статус |
| --- | --- | --- |
| get\_process\_info() получает список процессов | Возвращает корректные данные | Успешно |
| start\_monitoring() выводит список процессов | Данные обновляются в реальном времени | Успешно |

5.3 Тестирование графиков производительности (GUI.py)

| Тест | Ожидаемый результат | Статус |
| --- | --- | --- |
| Обновление графика CPU | Данные обновляются каждую секунду | Успешно |
| Обновление графика памяти | Данные корректно отображаются | Успешно |
| Запуск GPU-мониторинга | Отображается окно с графиком загрузки GPU | Успешно |

5.4 Тестирование обработки ошибок

| Ошибка | Ожидаемое поведение | Статус |
| --- | --- | --- |
| psutil.AccessDenied при получении списка процессов | Пропустить процесс, программа продолжает работу | Успешно |
| Ошибка при завершении процесса | Выводится сообщение об ошибке, программа не зависает | Успешно |

**6. Автоматическое тестирование (unittest)**

Добавим тесты для проверки работы функций main.py и GUI.py:

**Файл test\_main.py**

import unittest

from main import get\_process\_info

class TestTaskManager(unittest.TestCase):

def test\_get\_process\_info(self):

processes = get\_process\_info()

self.assertIsInstance(processes, list)

if processes:

self.assertIsInstance(processes[0].pid, int)

self.assertIsInstance(processes[0].name, str)

self.assertIsInstance(processes[0].memory, float)

self.assertIsInstance(processes[0].cpu, float)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**Файл test\_gui.py**

import unittest

import tkinter as tk

from GUI import TaskManagerGUI

class TestGUI(unittest.TestCase):

def setUp(self):

self.root = tk.Tk()

self.app = TaskManagerGUI(self.root)

def test\_refresh\_process\_list(self):

self.app.refresh\_process\_list()

self.assertGreater(len(self.app.tree.get\_children()), 0)

def tearDown(self):

self.root.destroy()

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Запуск тестов:

python -m unittest test\_main.py

python -m unittest test\_gui.py