Intern Assignment: GUI-Based Test Framework Setup Report

GitHub Repository:
https://github.com/Ryanzychen/GUI-Based-Test-Framework

1. Objective & Motivation

The goal of this project was to develop a lightweight, script-driven GUI-based test automation framework using AutoIt for Windows desktop applications. The target application, the MGC-400 Configurator, is a legacy MFC-based tool that lacks modern accessibility support, making conventional GUI automation tools ineffective.

This framework was designed to:
- Automate GUI interaction through coordinate-based logic, avoiding dependency on accessibility elements.
- Provide a practical and reusable solution for automating MFC-style Windows applications, which are still widely used in enterprise environments.
- Demonstrate how OCR (Optical Character Recognition) can be used to validate visual UI content where direct control access is unavailable.

2. Tools, Architecture & Implementation

Selected GUI Automation Tool: AutoIt

AutoIt was chosen for several key reasons:

- Strong Windows GUI Automation Support:
  It supports low-level GUI actions such as mouse movement, keystrokes, and control manipulation, making it suitable for MFC applications.

- Simplicity and Lightweight Scripting:
  AutoIt's syntax is straightforward, enabling quick learning and script development. Compared to Pywinauto, WinAppDriver, or TestComplete, AutoIt requires minimal setup and integrates well with traditional desktop applications.

- No Installation Overhead:
  Scripts can be compiled into .exe files, runnable on any system without requiring AutoIt to be installed.

Tool Comparisons

- AutoIt vs. Pywinauto & WinAppDriver:
  These tools depend on accessibility properties like control names and automation IDs, which MFC applications often lack. AutoIt instead interacts via coordinates and window messages, making it more reliable in this context.

- AutoIt vs. TestComplete:
  While TestComplete offers enterprise-level features, it is overly complex and licensed, making AutoIt more suitable for quick, flexible testing in MFC environments.

Framework Structure

The framework is implemented in AutoIt and designed with modularity and extensibility in mind:

- Utility Library (*lib/utils.au3*):
  Includes reusable logic for launching the application, clicking via relative positions, logging, and handling pop-ups, such as preference dialogues.

- Test Cases (*tests/test_launch_about.au3, test_add_device.au3*, *test_toolsmenu_build.au3*):
  Test_launch_about opens the Configurator and verifies the Help → About dialogue.
  Test_add_device adds a new device, captures its type using a fixed coordinate region, and validates it through Tesseract OCR (e.g., checking if the type is 'Signal').
  Test_toolsmenu_build enhances robustness by checking whether the "Tools" menu is visible on the main window using OCR. If not visible, it automatically opens User Preferences to enable the "Show Tools Menu" checkbox before continuing to build the job. This dynamic UI state detection improves test reliability and adapts to different application states.

Note: OCR-based validation was added as part of a deeper exploration toward future possibilities of fully automated testing. As such, some OCR-related results may currently be less accurate or stable.

- Test Runner (*test_runner.au3*):
  Executes all test cases in sequence with logging and delay handling, providing a central script for streamlined testing.

- Logging Mechanism:
  All test outputs are logged via LogToFile, allowing easy debugging and result tracing.

3. Results, Limitations & Reflections

Results and Benefits

- A working test framework capable of automating real MFC GUI behavior using low-level interaction.

- Proof of concept showing OCR can be a viable workaround when direct control access is unavailable.
- The entire system is portable and self-contained, with no dependencies beyond AutoIt and Tesseract OCR.

Limitations & Challenges

- Coordinate-Based Clicking:
Scripts are resolution- and window-position-sensitive, which makes them fragile across different environments.

- Lack of Object Recognition:
AutoIt does not provide high-level access to control elements or their properties, unlike modern automation tools.

- OCR Accuracy & Visual Sensitivity:
The capture region is fixed. If a device is already listed or if UI layouts shift slightly, the OCR may return incorrect results. Changes in font rendering, resolution, or even minor alignment issues can degrade reliability.

- Limited Error Handling & Reporting:
AutoIt lacks rich exception handling. Although text-based logging is implemented, structured reports (e.g., HTML/XML) would require further custom work.

- Environment Notes:
The provided version of the MGC-400 Configurator used during development did not require a WibuKey hardware security dongle, and no output devices were listed in the UI. This simplified the development process compared to a production environment, where hardware authentication and additional device handling would be required.

Conclusion

This project successfully demonstrates that even for older MFC-based applications, test automation is possible using a minimal tool setting. Through careful use of AutoIt and OCR-based validation, we created a flexible and scalable foundation for further automated testing in non-accessible GUI environments.

Note: Although OCR-based validation was not part of the official assignment requirements, it was included as part of my deeper understanding and vision for fully automated testing of the application. As a result, OCR-related validations are experimental at this stage and may not always produce accurate results.