Project Analysis Report

Introduction

This report provides a detailed analysis of the MrSnowman project. Within the analysis, we used various code analysis and testing tools, including Address Sanitizer, Astyle, Clang Static Analyzer, Thread Sanitizer and unit tests.

Analysis Process

1. Address Sanitizer

Steps:

- Added Address Sanitizer flags in the `.pro` file.
- Rebuilt the project and ran the application.

Results:

- No detected errors or warnings.

Note:

- Observed positional window errors and a font issue with "Magnolia Sky". The obtained errors are not directly a result of Address Sanitizer's operation. They indicate potential code issues that require further investigation.

2. Astyle

Steps:

- Ran the Astyle script to format all `.cpp` and `.h` files.

Results:

- All specified files were successfully formatted.

3. Clang Static Analyzer

Steps:

- Added Clang Static Analyzer flags in the `.pro` file.
- Rebuilt the project and ran the analysis.
- Analysis was executed using `clang-17`.

Results:

- No errors or warnings found.

4. Thread Sanitizer

Steps:

- Added Thread Sanitizer flags in the `.pro` file.
- Rebuilt the project and ran the application.

Results:

- No detected errors or warnings.

Note:

- A malloc error, positional window errors, and a font issue with "Magnolia Sky" were noticed. The obtained errors are not directly a result of Thread Sanitizer's operation. They indicate potential code issues, such as memory management and window positioning, which require further investigation and resolution.

5. Unit Tests

Steps:

- Added additional unit tests and updated the `.pro` file with the appropriate flags.
- Rebuilt the project and ran the tests.

Results:

- Code coverage is 39.1% for lines and 41.3% for functions.

Conclusions

- **Address Sanitizer:** No detected errors or warnings.
- **Astyle:** Code formatting successfully applied to all files.
- **Clang Static Analyzer:** No detected errors or warnings.
- **Thread Sanitizer:** No detected errors or warnings.
- **Unit Tests:** There is room for improvement in code coverage, considering the current coverage percentage.

Detailed Code Coverage Analysis

Overview

Lines

- **Coverage Rate:** 39.1%
- **Total Lines:** 12319
- **Covered Lines:** 4813

Functions

- **Coverage Rate:** 41.3%
- **Total Functions:** 12384

- **Covered Functions:** 5120

Detailed Analysis

Coverage by Directories

/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/include

- **Line Coverage:** 72.7%
- **Function Coverage:** 80.0%

/Users/miloszivkovic/Desktop/VSProject/04-MrSnowman/MrSnowman/code

- **Line Coverage:** 74.8%
- **Function Coverage:** 77.6%

/Users/miloszivkovic/Desktop/VSProject/04-MrSnowman/MrSnowman_tests

- **Line Coverage:** 89.1%
- **Function Coverage:** 77.2%

/Users/miloszivkovic/Desktop/VSProject/04-MrSnowman/MrSnowman_tests/catch2

- **Line Coverage:** 29.2%
- **Function Coverage:** 41.1%

/opt/homebrew/lib/QtCore.framework/Headers

- **Line Coverage:** 66.8%
- **Function Coverage:** 62.7%

/opt/homebrew/lib/QtGui.framework/Headers

- **Line Coverage:** 81.0%
- **Function Coverage:** 85.7%

/opt/homebrew/lib/QtWidgets.framework/Headers

- **Line Coverage:** 78.9%
- **Function Coverage:** 72.7%

Conclusions and Recommendations

- **Prioritize Areas with Low Coverage:** Initiate testing in sections with insufficient coverage to identify potential issues.
- **Enhance Test Suite:** Expand the test suite in areas with moderate coverage to guarantee comprehensive functionality verification.

- **Monitor Coverage Reports:** Consistently review coverage reports to track advancements and pinpoint areas requiring focus.