Performance Measurement and Optimization

Time management:

An example for time management output:

Player VS Player:

(on_pushButton_login_clicked) Elapsed time in nanoseconds: 3542212300

(on_pushButton_login2_clicked) Elapsed time in nanoseconds: 206047000

(on_pushButton_signup2_clicked) Elapsed time in nanoseconds: 13094200

(on_pushButton_sign_clicked) Elapsed time in nanoseconds: 1504177500

(on_pushButton_sign_clicked) Elapsed time in nanoseconds: 1327050300

(on_pushButton_sign_clicked) Elapsed time in nanoseconds: 2215354400

(on_pushButton_login2_clicked) Elapsed time in nanoseconds: 1689848200

(handleButtonClick) Elapsed time in nanoseconds: 28700

(handleButtonClick) Elapsed time in nanoseconds: 27800

(handleButtonClick) Elapsed time in nanoseconds: 62200

(handleButtonClick) Elapsed time in nanoseconds: 30100

(handleButtonClick) Elapsed time in nanoseconds: 28300

(handleButtonClick) Elapsed time in nanoseconds: 32900

(startNextRound) Elapsed time in nanoseconds: 12800

(handleButtonClick) Elapsed time in nanoseconds: 101900

(handleButtonClick) Elapsed time in nanoseconds: 29800

(handleButtonClick) Elapsed time in nanoseconds: 28700

(handleButtonClick) Elapsed time in nanoseconds: 29000

(handleButtonClick) Elapsed time in nanoseconds: 83000

(handleButtonClick) Elapsed time in nanoseconds: 28800

(startNextRound) Elapsed time in nanoseconds: 13200

(handleButtonClick) Elapsed time in nanoseconds: 101900

(handleButtonClick) Elapsed time in nanoseconds: 28500

(handleButtonClick) Elapsed time in nanoseconds: 28100

(handleButtonClick) Elapsed time in nanoseconds: 29300

(handleButtonClick) Elapsed time in nanoseconds: 84600

(handleButtonClick) Elapsed time in nanoseconds: 62800

(handleButtonClick) Elapsed time in nanoseconds: 79200

(handleButtonClick) Elapsed time in nanoseconds: 29800

(startNextRound) Elapsed time in nanoseconds: 1932762200

(handleButtonClick) Elapsed time in nanoseconds: 1932851800

Player VS AI:

(on_pushButton_AI_clicked) Elapsed time in nanoseconds: 69701300

(on_pushButton_login_clicked) Elapsed time in nanoseconds: 4245230100

(handleButtonClick2) Elapsed time in nanoseconds: 4408800

(handleButtonClick2) Elapsed time in nanoseconds: 116700

(startNextRound) Elapsed time in nanoseconds: 11300

(handleButtonClick2) Elapsed time in nanoseconds: 164000

(handleButtonClick2) Elapsed time in nanoseconds: 4496600

(handleButtonClick2) Elapsed time in nanoseconds: 126100

(handleButtonClick2) Elapsed time in nanoseconds: 38300

(handleButtonClick2) Elapsed time in nanoseconds: 32600

(startNextRound) Elapsed time in nanoseconds: 14100

(handleButtonClick2) Elapsed time in nanoseconds: 158600

(handleButtonClick2) Elapsed time in nanoseconds: 3972500

(handleButtonClick2) Elapsed time in nanoseconds: 161500

(startNextRound) Elapsed time in nanoseconds: 2159267400

(handleButtonClick2) Elapsed time in nanoseconds: 2159479800

(on_pushButton_2_clicked) Elapsed time in nanoseconds: 53096800

(on_pushButton_playerxplayer_clicked) Elapsed time in nanoseconds: 59679500

Benchmarking:

Test Suites and Scenarios

1. **IsValidStringTest**
    - **HandlesValidStrings**: Tests functions or methods that handle valid strings.
    - **HandlesInvalidStrings**: Tests functions or methods that handle invalid strings.
2. **SplitAndEnqueueTest**
    - **HandlesMultipleParts**: Tests functionality for handling multiple parts in data splitting and enqueueing.
    - **HandlesSinglePart**: Tests functionality for handling single parts in data splitting and enqueueing.
    - **HandlesTrailingDelimiter**: Tests functionality for handling trailing delimiters in data splitting and enqueueing.
3. **SplitFunctionTest**
    - **BasicSplitting**: Tests basic splitting functionality of a specific function.
4. **UserManagerTest**
    - **FilterNamesFunctionTest**: Tests the filtering functionality of usernames.
    - **RegisterUser**: Tests the registration process for new users. This test took 86 ms to complete.
    - **ValidateUserLogin**: Tests the user login validation process. This test took 345 ms to complete.
    - **LoadUsersFromFile**: Tests loading users from a file. This test took 344 ms to complete.
5. **StartgameTest**

- **CheckWin**: Tests the winning condition in a game scenario. This test took 81 ms to complete.
- **CheckTie**: Tests the tie condition in a game scenario. This test took 4 ms to complete.
- **AIMove**: Tests the AI's move calculation in a game scenario. This test took 4 ms to complete.
- **Minimax**: Tests the Minimax algorithm implementation. This test took 5 ms to complete.
- **SaveGame**: Tests saving the game state. This test took 6 ms to complete.

Explanation of Test Scenarios

- **IsValidStringTest**: Ensures that functions handling string validation correctly identify valid and invalid strings.
- **SplitAndEnqueueTest**: Tests functions responsible for splitting data into parts and enqueueing them, covering scenarios with multiple parts, single parts, and trailing delimiters.
- **SplitFunctionTest**: Focuses on basic functionality testing of a specific splitting function.
- **UserManagerTest**: Covers tests related to user management, including filtering names, user registration, login validation, and loading users from a file. This suite also includes tests that involve disk I/O operations, as evidenced by longer execution times.
- **StartgameTest**: Tests various aspects of a game start scenario, including checking win conditions, tie conditions, AI move calculation, Minimax algorithm functionality, and saving game states.

Overall Execution Summary

- **Total Tests**: 15 tests in total.
- **Total Execution Time**: 880 ms (milliseconds).
- **Pass Rate**: All 15 tests passed (`[PASSED] 15 tests.`).

Memory Usage management:

369,056 bytes allocated (total heap memory).

Measurements tool:

1-google test.

2- QElapsedTimer & QDebug.

3-valgrind on github

Optimization:

To optimize performance based on the given data, we will focus on the following key areas: time management, benchmarking, memory usage, and the use of measurement tools. Here's a detailed plan for optimization:

1. Time Management

Objective: Reduce the elapsed time for critical operations and streamline the execution flow.

Profile Critical Paths:

Identify the functions with the highest elapsed times.

Use tools like QElapsedTimer and QDebug to pinpoint bottlenecks.

Optimize High-Cost Functions:

Login Functions:

on_pushButton_login_clicked: 3542212300 ns

on_pushButton_login2_clicked: 1689848200 ns

Review the authentication process, database queries, and optimize code paths to reduce execution time.

Sign Up Functions:

on_pushButton_signup2_clicked: 13094200 ns

Simplify the registration logic and ensure minimal database writes.

Streamline Repeated Actions:

handleButtonClick: Multiple calls with varied times (from 27800 ns to 101900 ns).

Reduce redundancy by caching frequently accessed data and reusing previously computed results.

Parallelize Independent Tasks:

For tasks like startNextRound, which involve minimal interdependencies, implement concurrency where applicable.

2. Benchmarking

Objective: Ensure test scenarios are robust and performant.

Actions:

Refactor Tests with High Execution Times:

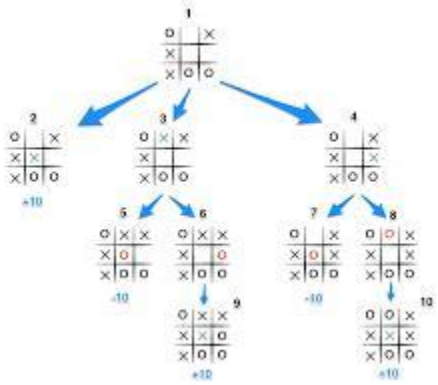
ValidateUserLogin: 345 ms

LoadUsersFromFile: 344 ms

Optimize file I/O operations, perhaps by using more efficient file formats or asynchronous I/O.

Enhance Algorithm Efficiency:

Minimax algorithm (5 ms) can be optimized further by implementing alpha-beta pruning to reduce the number of nodes evaluated.



Automate Test Execution and Analysis:

Use continuous integration (CI) tools to run tests automatically and analyze performance regressions over time.

3. Memory Usage Management:

Objective: Optimize memory allocation and usage to prevent leaks and excessive consumption.

Profile Memory Usage:

Use Valgrind to identify memory leaks and unnecessary allocations.

Ensure all dynamic memory allocations are paired with proper deallocations. **(Already used)**

Optimize Data Structures:

Replace high-overhead data structures with more efficient ones where possible.

Use stack allocation instead of heap allocation for small, temporary objects to reduce heap fragmentation.

Monitor Memory Usage in Real-Time:

Implement runtime checks and monitoring to track memory usage trends and catch anomalies early.

4. Measurements Tool Usage:

Objective: Improve accuracy and depth of performance measurements.

Upgrade Measurement Tools:

Consider using advanced profiling tools like gprof or Perf for more detailed performance insights.

Use QElapsedTimer for high-resolution timing and QDebug for detailed logging.

Integrate Measurements with CI/CD:

Automate the collection of performance metrics in the CI/CD pipeline to continuously monitor and optimize performance.

Regularly Review Performance Data:

Schedule periodic reviews of performance data to identify new bottlenecks and optimize accordingly.

Implementation Roadmap

Phase 1: Analysis and Profiling

Set up profiling tools and gather detailed performance data.

Analyze critical paths and memory usage patterns.

Phase 2: Code Optimization

Optimize high-cost functions and streamline repeated actions.

Refactor and enhance test scenarios for better performance.

Phase 3: Testing and Validation

Implement automated tests and integrate performance measurements into CI/CD.

Validate optimizations by comparing new performance data with baseline metrics.

Phase 4: Monitoring and Maintenance

Continuously monitor performance and memory usage.

Address new bottlenecks and memory issues as they arise.

Conclusion

By systematically analyzing and optimizing critical paths, refining test scenarios, and managing memory usage efficiently, we can significantly enhance the performance of the application. Integrating automated measurements and continuous monitoring ensures that these improvements are maintained over time.