Performance Measurement & Optimization

For

Advanced Tic Tac Toe

Prepared by TicTacTactics

Submitted to: Dr. Omar Nasr

**Table of Contents**

[1. Introduction 1](#_Toc170375569)

[2. Performance Metrics 1](#_Toc170375570)

[3. Monitoring Tools 1](#_Toc170375571)

[4. Optimization Techniques 2](#_Toc170375572)

[4.1 Code Optimization 2](#_Toc170375573)

[4.2 Algorithm Improvement 3](#_Toc170375574)

[5. Benchmark Results 3](#_Toc170375575)

[5.1 Response Time 3](#_Toc170375576)

[5.2 CPU Usage 3](#_Toc170375577)

[5.3 Memory Usage 3](#_Toc170375578)

[6. Conclusion 3](#_Toc170375579)

# Introduction

This report focuses on the performance measurement and optimization of a C++ Tic Tac Toe game. The primary goal is to ensure that the game runs efficiently, using minimal resources while providing a responsive user experience.

# Performance Metrics

*To evaluate the performance of the Tic Tac Toe game, the following metrics were considered:*

* ***CPU Usage****: The percentage of CPU resources utilized by the game.*
* ***Memory Usage****: The amount of RAM consumed during the game's execution.*
* ***Response Time****: The time taken for the game to respond to user inputs.*

# Monitoring Tools

* ***Windows and psapi Libraries for C++****: For measuring the average CPU usage & Memory usage.*
* ***Chrono Library for C++****: For calculating response times.*

# Optimization Techniques

*Based on the benchmark results, several optimization techniques were applied to enhance the game's performance:*

## Code Optimization

* ***Description****: Refactored code to reduce redundant operations and improve efficiency.*

*We used a setupConnection function to make a template for all 9 buttons of the tic tac toe board, as all buttons do the same functions when they clicked.*

void pvai::setupConnections()

{

connect(ui->one, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->one); });

connect(ui->two, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->two); });

connect(ui->three, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->three); });

connect(ui->four, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->four); });

connect(ui->five, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->five); });

connect(ui->six, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->six); });

connect(ui->seven, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->seven); });

connect(ui->eight, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->eight); });

connect(ui->nine, &QPushButton::clicked, this, [this]() { handleButtonClick(ui->nine); });

}

## Algorithm Improvement

* ***Description****: Optimized the decision-making algorithm (minimax Algorithm) to reduce computational complexity and response time, specifically at the first move.*

*We found two implementations of the minimax algorithm, one with pairs, and the other with vectors. We used that one using vectors as it was faster, but there was a problem in the first move* ***only****, the first move took about 700ms, which is very long time.*

*So, we had to make the first move by using if conditions, as we found that the minimax algorithm makes a specific move for the first move depending on the player’s first move.*

|  |  |
| --- | --- |
| *Player’s first move* | *Corresponding move from Ai* |
| *(1, 1) , (0, 1) , (1, 0)* | ***(0, 0)*** |
| *(0, 0) , (0, 2) , (2, 0) , (2, 2)* | ***(1, 1)*** |
| *(1, 2)* | ***(0, 2)*** |
| *(2, 1)* | ***(0, 1)*** |

*This optimization made the game in the Ai hard mode using minimax algorithm much faster.*

# Benchmark Results

## Response Time

* *Response Time for easy ai (random) and player move is between 0.1ms and 0.3ms.*
* *Response Time for medium ai move is between 0.8ms and 3ms.*
* *Response Time for medium ai move is between 10ms and 95ms.*

## CPU Usage

* *The CPU Usage is between 0.1% and 0.5%*

## Memory Usage

* *The memory Usage is between 40 MB and 45 MB*

# Conclusion

*The performance measurement and optimization of the C++ Tic Tac Toe game ensured it runs smoothly and quickly. By focusing on important performance metrics, benchmarking, and applying optimization techniques, the game was improved to provide a great user experience.*