Basics of Programming II
BMEVIIIAA03

PONG

Program Documentation

FINAL PROJECT

MUHAMMAD IBRAHIM SHOEB

BSc in Computer Science Engineering, BME
OZLVV3

Lab Instructor: **Vaitkus Márton**
Group CS16D

TABLE OF CONTENT

INTRODUCTION **2**
PROGRAM INTERFACE **3**
How to Start the Program: 3
How to Terminate the Program: 4
PROGRAM EXECUTION **4**
User Interaction: 4
Output: 4
Error Handling: 4
INPUT AND OUTPUT **5**
Input: 5
Output: 5
Input and Output Formats: 5
PROGRAM STRUCTURES **5**
Overview: 5
Paddle Class: 6
Ball Class: 6
GameStats Class: 7
Main Function: 7
Program Flow: 7
EXAMPLES **8**
TESTING AND VERIFICATION **8**
IMPROVEMENTS AND EXTENSIONS **9**
DIFFICULTIES ENCOUNTERED **10**
CONCLUSION **11**
REFERENCES **13**
APPENDICES **14**

INTRODUCTION

Pong is a two-player arcade game where each player controls a paddle to hit a ball back and forth across the screen. The objective is to prevent the ball from passing your paddle while attempting to pass it to the opponent's side. Points are scored when a player fails to return the ball.

Solution Overview

The solution involves creating a C++ program that simulates the Pong game. This program utilizes classes for the paddles and the ball, enabling players to interact with the game environment. Additionally, functionality for managing game statistics, such as scores and high scores, is incorporated into the program.

Features

- Two-player gameplay.
- Paddles are controlled by arrow keys < > and A/D keys for player 1 and player 2, respectively.
- Ball bouncing mechanics.
- Scoring system.
- High score management.

Programming Environment

Developed using C++ programming language and SFML (Simple and Fast Multimedia Library).

Key Components:

- Paddle: Represents the paddles controlled by each player.
- Ball: Represents the ball object that moves across the screen.
- GameStats: Manages high scores and final score recording.

Technical Details:

- Object-oriented design.
- Dynamic memory management for file handling.
- Exception handling for error management.

- Utilization of SFML library for graphics and window management.

PROGRAM INTERFACE

How to Start the Program:

1. To start the program, execute the Pong.sln file.
2. Ensure that the necessary SFML library files are accessible.

How to Terminate the Program:

1. To terminate the program, press the escape key.
2. Alternatively, use the close button provided in the title bar of the application window.

PROGRAM EXECUTION

User Interaction:

1. The game begins with the ball placed at the center of the screen, and each player's paddle positioned at their respective sides.
2. Players control their paddles using the arrow keys (Player 1) and A/D keys (Player 2) for left and right movement.
3. The ball moves automatically across the screen, bouncing off the paddles and boundaries.
4. Players aim to hit the ball with their paddles to prevent it from passing their side of the screen.
5. Scoring occurs when a player fails to return the ball, with points awarded to the opposing player.
6. The game continues until one player terminate the program.

Output:

1. The game provides real-time visual feedback through the rendering of game objects (paddles, ball) on the screen.
2. Text elements display the current scores for both players and previous HS (High Scores)
3. Upon termination, the final scores are recorded and then displayed for reference.

Error Handling:

- The program includes error-handling mechanisms to address unexpected user inputs or system issues.
- Error messages or exceptions are displayed or logged as appropriate to alert users or developers to potential issues.

INPUT AND OUTPUT

Input:

1. **User Input:** The primary input for the program is through keyboard input.
2. **Controls:**
    - Player 1 (Bottom paddle): Arrow keys (Left and Right)
    - Player 2 (Top paddle): A and D keys (Left and Right)
3. **Command-Line Parameters:** There are no command-line parameters required for program execution.

Output:

1. **Visual Output:** The program provides visual output through rendering game objects (paddles, ball) on the screen.
2. **Textual Output:**
    - Current scores for both players are displayed on the screen.
    - High scores, if available, are also displayed.
3. **File Output:**
    - High scores will be recorded and stored in an external file ("GameStats.txt") for file management.

Input and Output Formats:

1. **Keyboard Input:** Direct keyboard input is processed to control paddle movements.
2. **Screen Output:** Graphics are rendered on the screen using SFML library functions.
3. **File Output Format:**
    - High scores are stored as integer values in a plain text file ("GameStats.txt").

    - Each player's high score is stored on a separate line.

PROGRAM STRUCTURES

Overview:

1. **Object-Oriented Design:** The program employs an object-oriented design paradigm, encapsulating related functionality within classes and promoting code reusability and maintainability.
2. **Main Components:** The program consists of the following main components:
    - **Paddle:** Represents the paddles controlled by players.
    - **Ball:** Represents the ball object that moves across the screen.
    - **GameStats:** Handles high score management and file I/O operations.
    - **Main Function:** Contains the game loop and orchestrates interactions between game objects.

Paddle Class:

1. **Purpose:** Represents a paddle in the Pong game.
2. **Attributes:**
    - Position (Vector2f): Stores the position of the paddle.
    - Shape (RectangleShape): Represents the graphical shape of the paddle.
    - Speed (float): Defines the speed of paddle movement.
    - MovingRight, MovingLeft (bool): Flags indicating paddle movement direction.
3. **Member Functions:**
    - Constructors: Initialize paddle attributes.
    - setColor(): Set the color of the paddle.
    - moveLeft(), moveRight(): Start paddle movement in the respective direction.
    - stopLeft(), stopRight(): Stop paddle movement in the respective direction.
    - update(): Update paddle position based on movement flags.

Ball Class:

1. **Purpose:** Represents the ball object in the game.
2. **Attributes:**
    - Position (Vector2f): Stores the position of the ball.

- Shape (RectangleShape): Represents the graphical shape of the ball.
- Speed (float): Defines the speed of ball movement.
- DirectionX, DirectionY (float): Define the direction of ball movement.

3. **Member Functions:**
    - Constructors: Initialize ball attributes.
    - setColor(): Set the color of the ball.
    - getXVelocity(): Get the horizontal velocity of the ball.
    - reboundSides(), reboundPaddleOrTop(), reboundBottom(), reboundPaddleMult(): Handle ball collision and rebound.
    - update(): Update ball position based on current velocity.

GameStats Class:

1. **Purpose:** Manages high scores and file I/O operations.
2. **Attributes:**
    - playerOneCurrentScore, playerTwoCurrentScore (int): Store current high scores.
3. **Member Functions:**
    - recordFinalScores(): Record final scores to a file.
    - getHighScores(): Retrieve high scores from a file.

Main Function:

1. **Purpose:** Orchestrates the game loop and manages interactions between game objects.
2. **Components:**
    - Initialization: Set up window, paddles, ball, and game objects.
    - Event Handling: Process user input and window events.
    - Game Logic: Update game state based on user input and collisions.
    - Rendering: Draw game objects and text elements on the screen.

Program Flow:

1. **Initialization:** Initialize window, game objects, and variables.
2. **Game Loop:** Continuously process events, update game state, and render frame until the game is terminated.
3. **Event Handling:** Process user input and window events to control paddle movement and quit the game.
4. **Game Logic:** Update positions of paddles and ball, handle collisions, and update

scores.

5. **Rendering:** Draw game objects, HUD elements, and text on the screen for each frame.

Documentation Structure:

- Comments and documentation within source code provide insights into class functionality and usage.

EXAMPLES

This section provides brief examples illustrating key aspects of the Classic Pong program's execution.

1. **Basic Gameplay Example**:
   - **Input**: Player controls paddles using arrow keys ('←' and '→' for Player 1, 'A' and 'D' for Player 2).
   - **Outcome**: Players attempt to hit the ball back and forth, scoring points when the opponent fails to return the ball.
2. **Score Update Example**:
   - **Input**: Successful ball return by Player 1 and same for player 2.
   - **Outcome**: Scoreboard updates to reflect the point scored by Player 1.
3. **Game Over Example**:
   - Game continues uninterrupted until a player closes the window by either escape key or close the window from title bar.

TESTING AND VERIFICATION

This section outlines the testing process and verification methods employed to ensure the functionality and reliability of the Classic Pong program.

1. **Test Plan Overview**:
   - **Objective**: Ensure the program operates as intended across various scenarios and inputs.
   - **Scope**: Test all core functionalities, including player control, ball movement, scoring, and game continuation.
2. **Testing Procedures**:
   - **Unit Testing**: Verify individual components (paddles, ball, game

    mechanics) for correct behavior.
    - **Integration Testing**: Assess interactions between components to ensure seamless functionality.
    - **Regression Testing**: Validate that previously implemented features remain unaffected by new changes.
    - **User Acceptance Testing (UAT)**: Engage end-users to evaluate program usability and identify potential issues.
3. **Testing Environment**:
    - Utilize diverse hardware and software configurations to simulate real-world usage scenarios.
    - Ensure compatibility across different operating systems and screen resolutions.
4. **Test Results**:
    - Document test outcomes, including successes, failures, and any unexpected behaviors encountered.
    - Identify and address bugs, glitches, or performance issues through debugging and optimization.
5. **Verification Process**:
    - Validate program functionality against predefined requirements and specifications.
6. **Conclusion**:
    - Summarize the effectiveness of testing procedures in ensuring program reliability and functionality.
    - Highlight any areas for improvement or future enhancements based on testing insights.

IMPROVEMENTS AND EXTENSIONS

Identified Improvements:

1. Enhance User Interface: The Pong game's user interface can be improved with better graphics, animations, and visual effects to enhance the gaming experience.
2. Add Sound Effects: Incorporating sound effects for paddle hits, ball movements, and score updates can make the gameplay more engaging and immersive.
3. Implement AI Opponent: Introduce an AI-controlled opponent for single-player mode, providing users with the option to play against the computer.
4. Multiplayer Mode: Extend the game to support multiplayer functionality, allowing

two players to compete against each other locally or over a network.

5. Customization Options: Add customizable settings for paddle size, ball speed, and other game parameters to accommodate different player preferences.

Potential Extensions:

1. Power-ups and Special Abilities: Introduce power-up items that players can collect during gameplay to gain temporary advantages, such as increased paddle size or faster ball speed.
2. Tournament Mode: Implement a tournament structure where players compete in a series of matches to determine the ultimate winner.
3. Online Leaderboards: Integrate online leaderboards to track high scores and achievements, enabling players to compare their performance with others globally.
4. Cross-Platform Support: Extend the game's compatibility to multiple platforms, including mobile devices, consoles, and web browsers, to reach a wider audience.
5. Custom Game Modes: Allow players to create custom game modes with unique rules and objectives, fostering creativity and replayability.

DIFFICULTIES ENCOUNTERED

Throughout the development process of the Pong game project, several challenges were encountered, hindering progress and requiring strategic solutions:

Programming Language Transition:

Transition to C++: Transitioning from C programming language to C++ posed initial difficulties, including understanding syntax nuances and memory management concepts.

Complexity of Game Logic:

Game Mechanics: Implementing the intricate game mechanics of Pong, such as paddle-ball collision detection and scoring algorithms, proved challenging due to the need for precise timing and accurate physics simulation.

Graphics and User Interface

Graphic Rendering: Achieving smooth and visually appealing graphics within the SFML framework required experimentation with rendering techniques and optimization strategies to prevent lag and frame rate drops.

User Interface Design: Designing an intuitive and user-friendly interface, including scoreboards, and previous scores, required iterative testing and feedback gathering to ensure optimal usability.

Testing and Debugging

Testing Coverage: Ensuring comprehensive test coverage for all game features and edge cases was challenging, requiring meticulous planning and execution of test cases to identify and resolve bugs effectively.

Debugging Complexity: Debugging complex interactions between game components, such as paddle-ball collisions and AI behavior, required systematic troubleshooting and diagnostic tools to isolate and address issues efficiently.

Resource Management:

Memory Leaks: Managing dynamic memory allocation and deallocation (new/delete) to prevent memory leaks and optimize resource utilization posed challenges, particularly in scenarios with frequent object instantiation and destruction.

Cross-Platform Compatibility:

Platform-Specific Issues: Ensuring compatibility and consistent performance across different operating systems and hardware configurations required platform-specific optimizations and testing procedures to address compatibility issues and performance discrepancies.

CONCLUSION

In conclusion, the development of the Pong game project was a rewarding individual endeavor that showcased proficiency in system and algorithm design, object-oriented programming, and testing methodologies.

Achievements:

1. Successful Implementation: The project successfully implemented core gameplay mechanics, including paddle movement, ball physics, scoring system, and user interface elements.
2. Object-Oriented Design: Demonstrated proficiency in object-oriented design principles through modular code organization and class-based implementation.
3. Dynamic Memory Management: Effectively utilized dynamic memory management techniques to prevent memory leaks and optimize resource usage.
4. Exception Handling: Incorporated exception handling mechanisms to gracefully handle runtime errors and exceptional conditions.
5. File Management: Utilized file management functionalities for storing and retrieving high scores, enhancing user experience and data persistence.

Lessons Learned:

1. Problem-Solving Skills: Developed problem-solving skills through navigating technical challenges and implementing creative solutions.
2. Documentation Importance: Recognized the importance of comprehensive documentation for code understanding, maintenance, and knowledge transfer.
3. Individual Development: Highlighted the value of individual project development in honing technical skills and fostering self-reliance.

Future Considerations:

1. Further Feature Enhancements: Consider future iterations for adding features such as multiplayer support and customizable settings to enhance player experience.
2. Cross-Platform Compatibility: Explore optimization for cross-platform compatibility to reach a wider audience across different devices and operating systems.
3. Community Engagement: Engage with the gaming community for feedback and

insights to refine and improve the game based on user preferences.

Acknowledgments: Special thanks to Vaitkus Márton for guidance as the lab instructor and Dr. Dmitriy Dunaev for academic support and mentorship throughout the project.

REFERENCES

[1] J. Horton, *Beginning C++ Game Programming Second Edition*, Second. Livery Place 35 Livery Street Birmingham B3 2PB, UK.: Packt Publishing Ltd., 2019.

[2] J. Horton, “Making games: Where do I start?,” Game Code School. Accessed: May 19, 2024. [Online]. Available: https://gamecodeschool.com/blog/making-games-where-do-i-start/

[3] Accessed: May 19, 2024. [Online]. Available: https://cplusplus.com/doc/tutorial/polymorphism/

[4] “Techno > Sci-fi fonts | dafont.com.” Accessed: May 19, 2024. [Online]. Available: https://www.dafont.com/theme.php?cat=303

APPENDICES

1. **Paddle.cpp**:

```cpp
// Paddle constructor
Paddle::Paddle(float startX, float startY) : Position(startX, startY)
{
    // Implementation code...
}
```

2. **Paddle.h**:

```cpp
// Class for representing a paddle in the Pong game
class Paddle
{
private:
    Vector2f Position;
    RectangleShape Shape;
    float Speed = 350.0f;
    bool MovingRight = false;
    bool MovingLeft = false;
    float thickness;
    sf::Color outlineColor;

public:
    // Constructor, member functions, and other declarations...
};
```

3. **Ball.cpp**:

```cpp
// Ball constructor
Ball::Ball(float startx, float starty) {
    // Implementation code...
}
```

4. **Ball.h**:

```
// Class for representing a ball in the game
class Ball
{
private:
    Vector2f m_Position;
    RectangleShape m_Shape;
    float m_Speed = 1000.0f;
    float m_DirectionX = .2f;
    float m_DirectionY = .2f;

public:
    // Constructor, member functions, and other declarations...
};
```

5. **GameStats.cpp**:

```
// Static variables must be initialized only once outside the class before
usage
int GameStats::playerOneCurrentScore;
int GameStats::playerTwoCurrentScore;

// Function definitions...
```

6. **GameStats.h**:

```
// Class for managing game statistics
class GameStats {
public:
    // Static variables, function declarations...
};
```