

**FINAL PROJECT OBJECT**

**SUBJECT:** OBJECT-ORIENTED PROGRAMING  
**PROJECT TITLE**: LINEGRINDER

**GROUP MEMBER**Phạm Thái Quốc (Leader) ITITWE21118: 20%Bùi Thế Tùng ITITIU22243: 20%  
Võ Nguyễn Thanh Liêm ITITIU21069: 20%  
Phạm Nguyễn Công Danh ITDSIU23035: 20%  
Nguyễn Trầm Gia Hưng ITITIU23007: 20%

Instructor : Mr. Nguyễn Trung Nghĩa

TABLE OF CONTENTS

Contents

[**Chapter 1: Introduction** 2](#_Toc185802182)

[**1.1.** **Gaming in the Field** 2](#_Toc185802183)

[**1.2.** **About the Game Project** 3](#_Toc185802184)

[**1.3.** **Our LineGrinder Game** 3](#_Toc185802185)

[**1.4.** **References** 4](#_Toc185802186)

[**Chapter 2: Software Requirements** 4](#_Toc185802187)

[**2.1.** **What we have** 4](#_Toc185802188)

[**2.2.** **What We Want** 5](#_Toc185802189)

[**2.3.** **Working Tools and Platform** 5](#_Toc185802190)

[**2.4.** **Use Case Scenario** 5](#_Toc185802191)

[**2.5.** **Class Diagram** 7](#_Toc185802192)

[**Chapter 3: Design and Implementation** 12](#_Toc185802193)

[**Chapter 4: Final App Game** 15](#_Toc185802194)

[**Chapter 5: Experience** 23](#_Toc185802195)

[**5.1.** **Lessons Learned** 23](#_Toc185802196)

[**5.2.** **Challenges** 23](#_Toc185802197)

[**5.3.** **Future Features** 24](#_Toc185802198)

[**5.4.** **Reflections** 25](#_Toc185802199)

[**Chapter 6: Conclusion** 25](#_Toc185802200)

[**Chapter 7: Reference** 26](#_Toc185802201)

# **Chapter 1: Introduction**

## **1.1.** **Gaming in the Field**

In the rapidly growing field of software engineering and development, video game creation stands out as both unique and similar to other software endeavors. It is unique in that it combines the work of teams from multiple disciplines (art, music, acting, programming, etc.) and seeks engaging gameplay through prototypes and iterative development.

Our project focuses on creating the game LineGrinder as part of the Object-Oriented Programming course. This work allows us to refine our development cycle, improve Java programming skills, and apply theoretical knowledge gained in class, such as object-oriented programming principles, design patterns, and data structures. Through this project, we aim to blend creativity with technical proficiency.

## **1.2.** **About the Game Project**

While there are existing strategy games like Gomoku available on various platforms, many lack customization and additional features that maintain long-term player engagement. Recognizing this, our team designed LineGrinder to modernize Gomoku with a fresh, up-to-date interface and new features.

Key features include customizable board sizes, local multiplayer options, and enhanced gameplay mechanics. The goal is to provide a more immersive experience that caters to diverse player preferences while ensuring replayability and user satisfaction. By creating LineGrinder, we aim to produce a modernized strategy game that stands out in the competitive gaming landscape.

## **1.3.** **Our LineGrinder Game**

LineGrinder builds upon Gomoku's classic gameplay with innovative enhancements designed to keep users engaged. The basic rule is to align five consecutive pieces (vertically, horizontally, or diagonally) on a customizable grid. In addition to this, we have added the following features to enrich the game experience:

* Customizable Board Sizes: Players can choose grid dimensions for varied gameplay challenges.
* Local Multiplayer: Enables two players to compete on the same device.
* Timer Options: Adds intensity and strategy with countdown timers.
* Enhanced Interface: Includes animations and sound effects for an engaging experience.
* Login and Profiles: Players can choose their name and avatars

These features are implemented using advanced programming techniques, including object-oriented design principles, modular architecture, and efficient algorithms for win detection.

## **1.4.** **References**

The development process relied on various resources and tools, including:

* Gameplay Rules and Strategies: Research on Gomoku and related strategy games.
* Java and Swing Development: Tutorials and documentation on GUI development.
* Collaborative Platforms: GitHub for version control and teamwork.
* Educational Resources:
  + Online Java courses
  + Stack Overflow for problem-solving and community support
  + Software development tutorials

# **Chapter 2: Software Requirements**

## **2.1.** **What we have**

* A user-friendly, efficient, and visually appealing system, designed to cater to diverse user needs.
* Minimal maintenance costs, with a focus on optimizing graphics for performance and resource management.
* Compatibility with standard PC and mobile configurations, ensuring accessibility for a wide audience.
* An intuitive interface that simplifies operations and enhances the user experience.
* A codebase crafted with professional standards, emphasizing maintainability, scalability, and robust functionality.

## **2.2.** **What We Want**

Develop LineGrinder, a strategic Gomoku-inspired game, featuring:

* Interactive GUI with customizable board sizes to suit user preferences.
* Single and Local Multiplayer Modes to enhance user engagement through shared gameplay experiences.
* Advanced Game Logic that includes efficient win detection and strict rules enforcement to ensure fair play.
* Enhanced Visuals and Animations to provide an immersive gaming experience.
* Ease of Updates and Maintenance for adding future features and ensuring long-term usability.

## **2.3.** **Working Tools and Platform**

* Programming Language: Java, ensuring robust and scalable application development.
* Framework: Swing for building an interactive and visually appealing GUI.
* Integrated Development Environment (IDE):Visual Studio Code (VS Code) for efficient coding, debugging, and extension support.
* Version Control System: GitHub for collaboration, version tracking, and repository management.

## **2.4.** **Use Case Scenario**

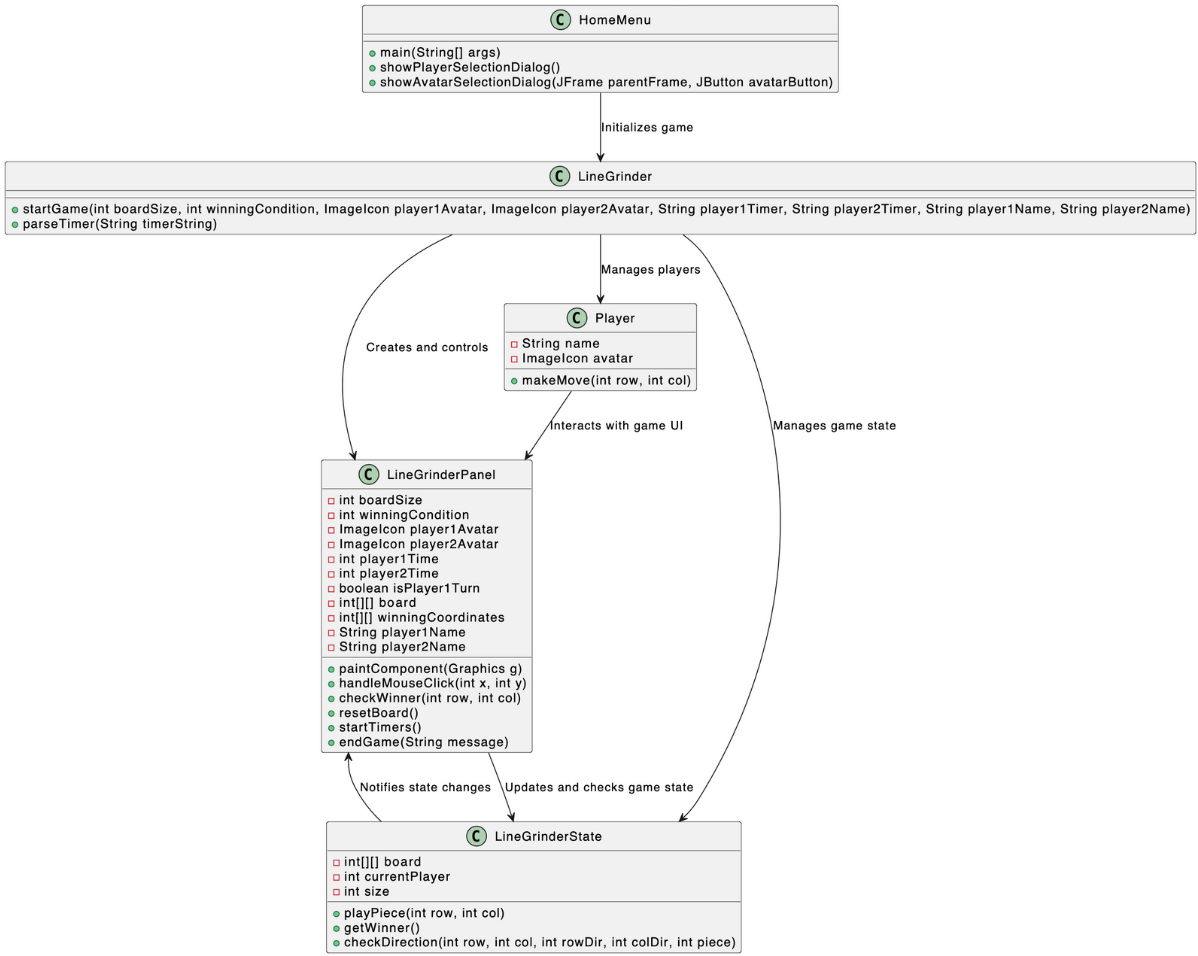
LineGrinder is a modernized Gomoku game designed to provide players with customizable settings, a user-friendly interface, and engaging gameplay experiences. Players can:

* Start a new game and select their preferred board size.
* Play against another player locally, fostering competitive and social interactions.
* Access game instructions to understand the rules and gameplay mechanics.
* Exit the game safely, preserving user settings and game state.

Use Case Workflow:

1. Players launch the game and are greeted by a main menu offering options to play, customize settings, or view instructions.
2. The game progresses turn-by-turn, with visual and auditory feedback to enhance engagement.
3. Advanced game logic checks for win conditions after every move, ensuring smooth and fair gameplay.

## **2.5.** **Class Diagram**

**Figure 1: UML diagram**

**How does it work:**

**Relationship between the Classes:**

**HomeMenu:**

* Role: Serves as the entry point of the program.
* Relation:
  + Allows users to select player details (like names and avatars).
  + Transfers control to the LineGrinder class to start the game.

**LineGrinder**

* Role: Acts as the primary controller of the game.
* Relation:
  + Manages coordination between the user interface (LineGrinderPanel) and the game logic (LineGrinderState).
  + Initializes the game with settings such as board size, timers, and player avatars.

**Player**

* Role: Represents individual players in the game.
* Relation:
  + Stores player-specific detail like name and avatar.
  + Interacts with LineGrinderPanel during gameplay.

**LineGrinderPanel**

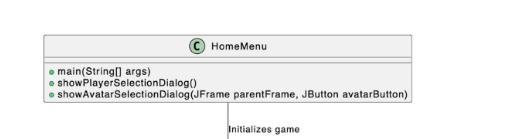
* Role: Handles user interface and interactions.
* Relation:
  + Visually updates the game board and processes user actions, such as clicks.
  + Communicates game events to LineGrinderState for state updates (e.g., checking for winners).
  + Uses player data from the Player class to display information like names and avatars.

**LineGrinderState**

* Role: Manages the state and rules of the game.
* Relation:
  + Track the game board, current player, and win conditions.
  + Notifies LineGrinderPanel about state changes to update the interface accordingly.

**Class Details:**

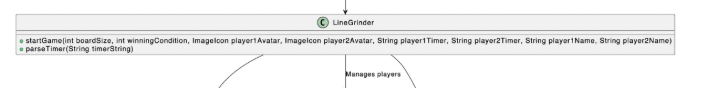
**HomeMenu.java: starting point of the game**

****

**Figure 2: HomeMenu class diagram**

* main(): Starts the whole game.
* showPlayerSelectionDialog(): For players to choose their avatars and name.
* showAvatarSelectionDiaglog(): Pop up showing all the avatars that are available.

**LineGrinder.java:**

****

**Figure 3: LineGrinder class diagram**

* startGame(): starting up with the board size, winning-drawing conditions, player avatars, timers, names.
* parseTimer(String timerString): used for timer countdown, into a usable format for the game.

**Player.java:**

* String name: Store player’s name.
* Imageicon avatar: Store the player’s chosen avatar.
* makeMove(int row, int col): Handles the player’s move by marking the chosen spot on the game board based on the row/columns of the board.

**LineGrinderPanel.java: Handles the game board and player interactions**

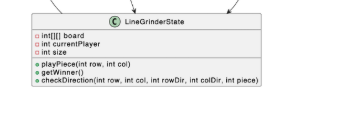
**A screenshot of a computer program

Description automatically generated**

**Figure 4: LineGrinderPanel class diagram**

* paintComponent(Graphics g):Draws the game board, grid, and player pieces. Highlights the winning line if a player wins.
* handleMouseClick(int x, int y):Processes a player’s click on the board. Updates the board state, checks for a win or tie, and switches turns.
* checkWinner(int row, int col): Checks if the current move causes a win by scanning in all possible directions (horizontal, vertical, and diagonal).
* resetBoard(): Resets the game board to its starting state, clearing all moves and restarting timers.
* startTimers(): Starts countdown timers for both players. If a player’s time runs out, the other player wins.
* endGame(String message): Stops the game, displays the result (winner or tie), and offers options to restart or exit the game.

**LineGrinderState.java: Updates and checks the game state**



**Figure 5: LineGrinderState class diagram**

* **int[][] board**:  
  Represents the game board as a 2D array, where each cell stores the state (empty, X, or O).
* **int currentPlayer**:  
  Keeps track of the player currently making a move (X or O).
* **int size**:  
  The size of the game board (e.g., 3x3, 5x5).
* **playPiece(int row, int col)**:  
  Places the current player’s piece on the board and switches to the other player if the cell is empty.
* **getWinner()**:  
  Checks the entire board to determine if there is a winner, a draw, or if the game is still ongoing.  
  Returns:
  + X (1) if Player X wins.
  + O (2) if Player O wins.
  + -1 for a draw.
  + NONE (0) if no winner yet.
* **checkDirection(int row, int col, int rowDir, int colDir, int piece)**:  
  Verifies if there are 5 consecutive pieces (horizontally, vertically, or diagonally) starting from the given cell in the specified direction.
* **isBoardFull()**:  
  Checks if every cell on the board is filled, indicating a draw if no winner exists.
* **getCurrentPlayer()**:  
  Returns the current player (X or O).

# **Chapter 3: Design and Implementation**

**Package Diagram**  
The project is modularized into the following packages:

* **Game**:  
  This package manages the game logic, rules, and board state.
  + Key Classes:
    - LineGrinderState: Handles the state of the game board, validates moves, and checks for a winner or tie.
    - LineGrinder: Manages the overall flow of the game, including initialization and coordination between components.
* **UI**:  
  This package handles the user interface and interactions.
  + Key Classes:
    - HomeMenu: Displays the main menu, allowing users to start the game, view instructions, or customize settings.
    - LineGrinderPanel: Draws the game board, processes player interactions, and displays player status and timers.
* **player**:  
  This package stores player-related information and behavior.
  + Key Classes:
    - Player: Represents a player, including their name and avatar. (This class might exist in your project based on game logic.)

UI Design  
The interface includes the following components:

1. Main Menu:
   * Allows players to start a new game, adjust game settings, or view instructions.
   * Managed by the HomeMenu class.
2. Game Screen:
   * Displays the game grid, players’ avatars, and timers.
   * Shows whose turn it is and includes a reset option.
   * Managed by the LineGrinderPanel class.
3. Settings Menu:
   * Allows customization of game settings, such as board size and timer duration.
   * Managed through logic connected to HomeMenu and possibly additional UI classes.

**Core Features**

1. **Customizable Board**: Players can choose board sizes (e.g., 10x10 or 15x15).
2. **Timer**: Chess-style countdown timers to add intensity.
3. **Win Detection**: Automatically checks for victory conditions after each move.
4. **Multiplayer**: Supports two players locally on the same machine.

**Key Classes and Methods**

**Game**

* **LineGrinderState**: Manages the game state, including the board and current player.

**getPiece(int row, int col)**: Returns the piece at the specified position.

**playPiece(int row, int col)**: Places a piece on the board and switches the current player.

**getWinner()**: Checks for a winner based on the current board state.

**checkDirection(int row, int col, int rowDir, int colDir, int piece)**: Checks for a winning line in a specified direction.

**UI**

* **LineGrinderPanel**: Handles the game interface, including the board and player information.

**createPlayerPanel(String playerName, ImageIcon playerAvatar, boolean isPlayer1)**: Creates a panel for player information.

**handleMouseClick(int x, int y)**: Processes mouse clicks on the board.

**startTimers()**: Starts the countdown timers for both players.

**endGame(String message)**: Ends the game and displays the result.

**resetBoard()**: Resets the board and timers for a new game.

**drawBoard(Graphics g):** Draws the game board and pieces.

* **LineGrinder**: Initializes and starts the game.

**startGame(int boardSize, int winningCondition, ImageIcon player1Avatar, ImageIcon player2Avatar, String player1Timer, String player2Timer, String player1Name, String player2Name)**: Sets up the game frame and panel.

**parseTimer(String timerString)**: Parses the timer string into seconds.

**Player**

* **Player**: Tracks player moves and stats.

**Player(String name, ImageIcon avatar)**: Constructor to initialize player with name and avatar.

**getName()**: Returns the player's name.

**getAvatar()**: Returns the player's avatar.

# **Chapter 4: Final App Game**

**Source Code**

The full source code is available at [GitHub - LineGrinder](https://github.com/tquoc0112/OOP-LineGrinder).

**Instructions**

1. Launch the program using the HomeMenu.java file.

2. Select board size and timer options from the settings menu after both sides selected their name and avatars.

3. Players take turns placing their pieces on the board.

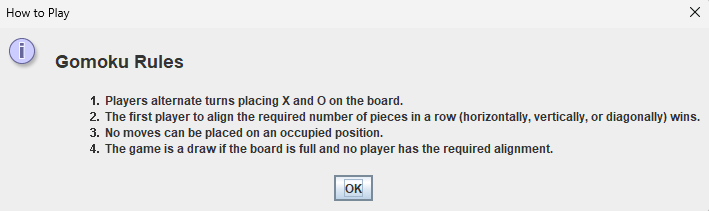
4. The game ends when a player aligns five pieces in a row (horizontal, vertical, or diagonal).

**GamePlay**

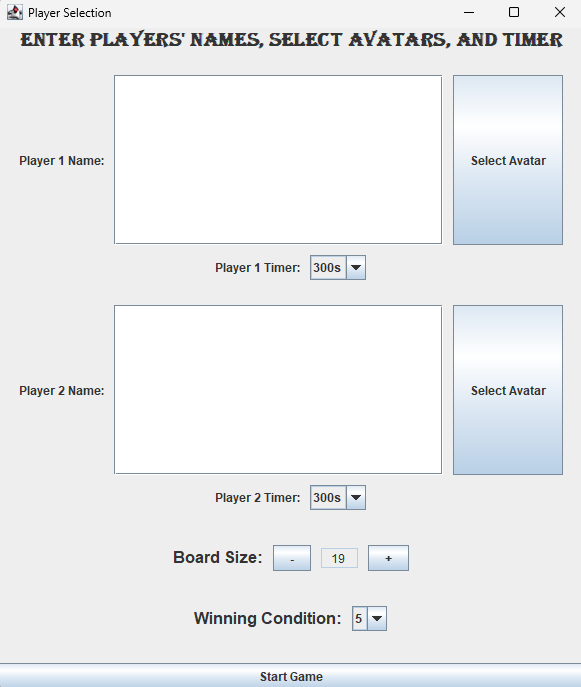
The home menu image appears when running the program:



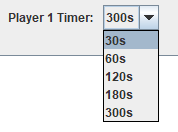
When clicking on “how to play”:



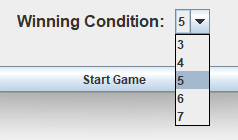
When clicking on “player vs player”, a settings panel will appear to enter the name and adjust the game settings:



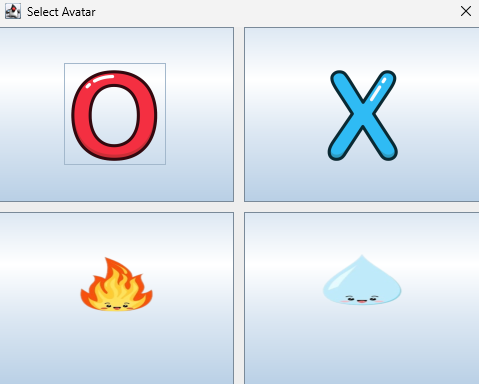
Adjust the thinking time of each player:



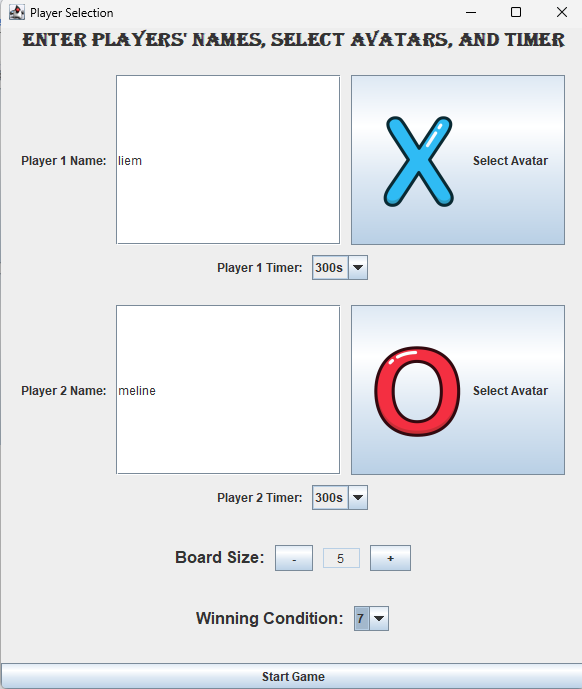
Adjust the game's victory conditions



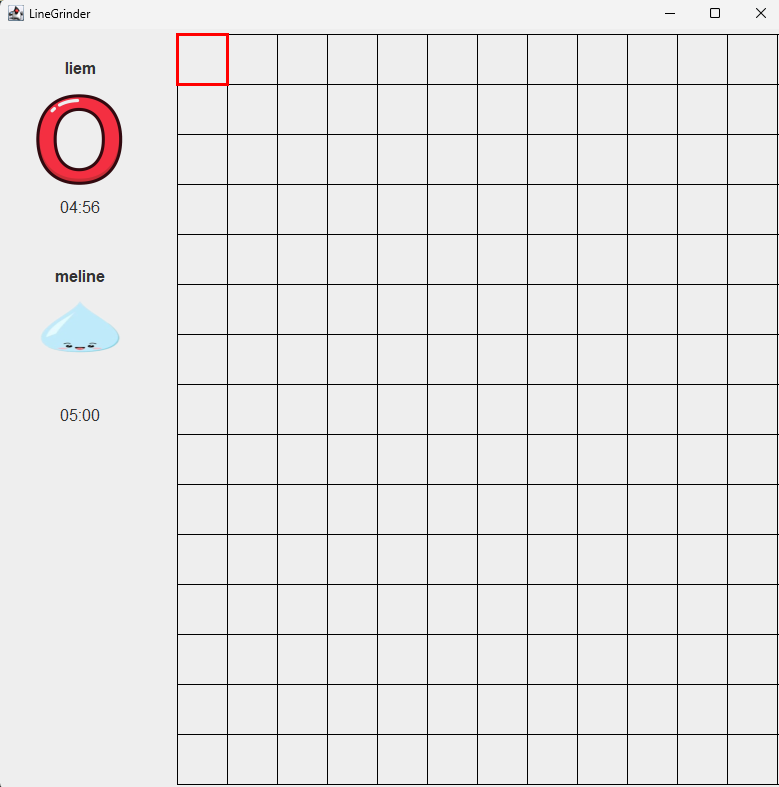
Adjust player avatar:



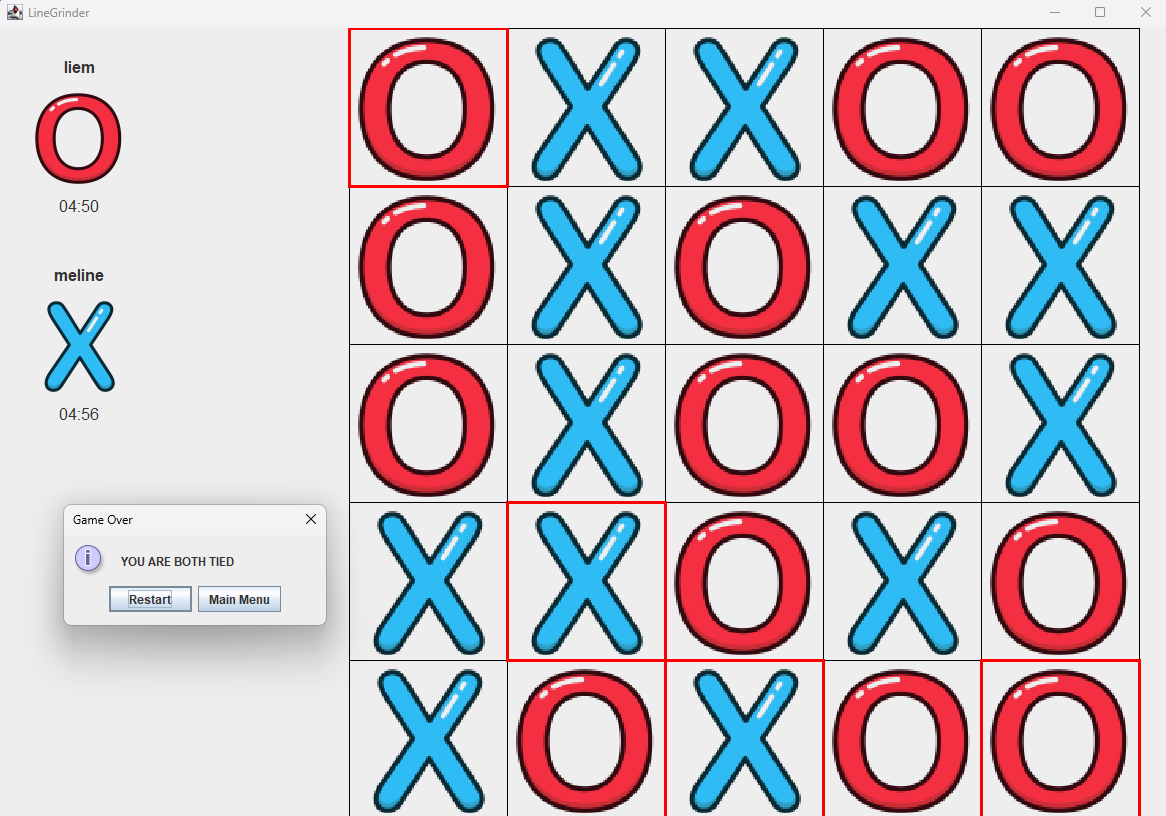
After completing the adjustment of settings:



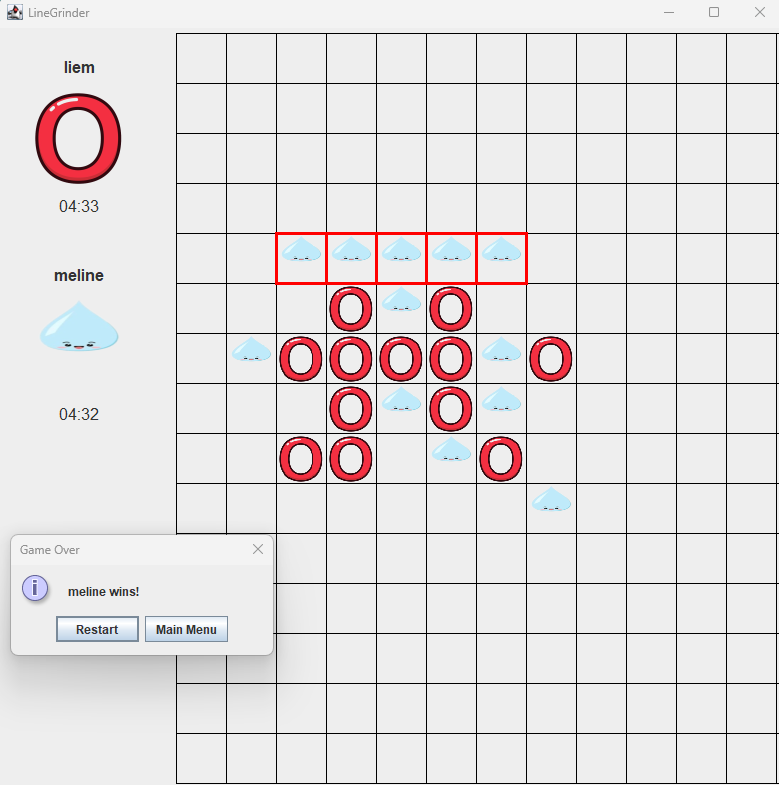
The chess board will appear after pressing “start game”:



The case when 2 players are tied:



The case when there is 1 winner:



In both cases, after clicking the “restart” button will return to the chessboard screen as before

**Demo Video**

[FINAL DEMO LINEGRINDER.mp4 - Google Drive](https://drive.google.com/file/d/1PhyoGKUXSQnK1ZYAaw8l2ESFUYNWOvNw/view?fbclid=IwY2xjawHU_r5leHRuA2FlbQIxMAABHV3g-q6K5thz-YiiDLhTeywyuBTeV0RKcNKFSIdd66L51b86FowlCVhnLQ_aem_mzLghzlhf7DUeW8ifvzmZg)

# **Chapter 5: Experience**

## **5.1.** **Lessons Learned**

Working on the LineGrinder project over the past two months has provided the team with invaluable lessons, not only in software development but also in teamwork and creative problem-solving. Key takeaways include:

* **Practical Application of Knowledge:**

The project allowed us to apply theoretical concepts learned in class, such as object-oriented programming. Implementing classic game logic using **Java**, combined with **GUI** design in Swing, strengthened our technical skills and confidence.

* **Collaboration and Version Control:**

Using **GitHub** for version control taught us the importance of effective collaboration and version management, helping us organize and integrate contributions from all team members.

* **Creative Thinking and Adaptability:**

As the game development process unfolded, we realized the importance of considering all aspects of a game, including its story, gameplay mechanics, artwork, and user experience. This holistic approach required creativity and adaptability to bring everything together seamlessly.

## **5.2.** **Challenges**

The journey to create LineGrinder was not without its difficulties. Some of the significant challenges we faced included:

* **Synchronizing GUI Updates with Game Logic:** Ensuring smooth and responsive interaction between the game logic and the GUI required meticulous debugging and optimization.
* **Optimizing Win Detection for Large Boards:** Developing efficient algorithms to handle large board sizes while maintaining quick response times tested our problem-solving skills and attention to detail.
* **Balancing Workloads and Schedules:** Coordinating the team's efforts and managing time effectively, especially with academic and personal commitments, proved to be a critical learning experience.

## **5.3.** **Future Features**

While the current version of LineGrinder represents a significant milestone, the project has room for further enhancement. Future updates and features we aim to implement include:

* **Online Multiplayer:** Introducing multiplayer functionality using socket programming will enable players to compete online, expanding the game's reach and appeal.
* **Enhanced AI:** Developing AI opponents with adjustable difficulty levels will cater to players of varying skill levels, making the game more engaging.
* **Improved Graphics and Animations:** Adding polished graphics and dynamic animations will elevate the user experience, making the game visually compelling and immersive.
* **Cloud Save and Leaderboards:** Implementing cloud saving and global leaderboards will allow players to store progress and compete on an international scale.

## **5.4.** **Reflections**

This project reinforced the notion that game development is about more than just programming. A successful game requires an engaging concept, seamless integration of design elements, and a compelling user experience.

Moreover, we learned that the field of Computer Science demands continuous self-study and exploration beyond the classroom. The IT world is vast and ever-evolving, and staying ahead requires proactive learning and adaptability.

As a team, we are proud of what we have accomplished with LineGrinder. It serves as a testament to our hard work, creativity, and determination. Moving forward, we are motivated to refine this game, publish it, and embark on new projects that challenge and inspire us.

# **Chapter 6: Conclusion**

Developing LineGrinder as part of our Object-Oriented Programming (OOP) course has been an invaluable learning experience. It allowed us to apply OOP principles, such as modular design and abstraction, to create a functional and engaging game. Through this project, we gained deeper insights into Java programming, GUI development with Swing, and efficient algorithm design for gameplay logic.

Beyond technical skills, we learned the importance of teamwork, effective communication, and version control to manage collaborative projects. The challenges we faced strengthened our problem-solving abilities and adaptability.

This project not only showcases what we have achieved in our OOP course but also inspires us to continue refining LineGrinder and tackling more ambitious projects in the future.

# **Chapter 7: Reference**

<https://github.com/gomoku/Gomoku-Java/blob/master/Gomoku.java>

<https://stackoverflow.com/questions/22087006/using-arrays-to-detect-a-win-in-a-gomoku-game>

<https://github.com/rchen8/connect-five>