Aru Pandey
U14204225

Design Documentation

1. Introduction:

The system is designed to support turn-based games, specifically focusing on variants like Tic Tac Toe, Order and Chaos, and Super Tic Tac Toe. The design emphasizes scalability and extendibility to accommodate different turn-based games.

2. Class Structure:

2.1 Board:
Represents the game board.
Has a 2D array of Cell objects.
Allows for dynamic sizing.
Provides methods for displaying the board, checking emptiness, getting the board, setting cell values, and getting the size.
2.2 Cell:
Represents a single cell on the game board.
Stores a value (X, O, or empty space).
Provides methods for getting, setting, and checking emptiness of the cell.
2.3 Player:
Represents a player in the game.
Stores the player's name and marker (X or O).
Uses Scanner to get user input for marker selection.
Ensures marker selection is valid and unique.
2.4 WinRules:
Contains static methods to check for a win or draw.
Works for both Tic Tac Toe and Order and Chaos games.
Allows for extension to other turn-based games by adding specific win conditions.
2.5 Menu:
Handles the game menu, user input validation, and game result display.
Provides a modular approach for different games to have their own menu and flow.
2.6 TicTacToe:
Implements the game logic for regular Tic Tac Toe.
Inherits from the BoardGame interface.
Uses a Board and two Player objects.
Overrides interface methods for playing, making a move, getting user input, and resetting the game.
2.7 OandC (Order and Chaos):

Implements the game logic for Order and Chaos.
Inherits from the BoardGame interface.
Uses a modified Board and two Player objects.
Overrides interface methods for playing, making a move, getting user input, and resetting the game.
2.8 SuperTicTacToe:
Implements the game logic for Super Tic Tac Toe.
Inherits from the BoardGame interface.
Uses a 2D array of TicTacToe boards and two Player objects.
Overrides interface methods for playing, making a move, getting user input, and resetting the game.
Introduces an additional layer of complexity by managing a matrix of smaller Tic Tac Toe boards.

3. Scalability and Extendibility:

Interface (BoardGame):

The introduction of an interface allows for the addition of more turn-based games without modifying existing classes.
Any new game needs to implement the BoardGame interface, ensuring a standardized set of methods.
Inheritance and Polymorphism:

The use of inheritance facilitates the reuse of common game logic and the addition of specific rules for each game.
The TicTacToe, OandC, and SuperTicTacToe classes share a common interface but provide game-specific implementations.
Dynamic Board Size:

The Board class supports dynamic sizing, making it adaptable to games with different board dimensions.
Modularity:

Each game (Tic Tac Toe, Order and Chaos, Super Tic Tac Toe) is encapsulated in its own class, promoting modularity.
New games can be added by creating a class that implements the BoardGame interface.
Player Marker Selection:

The Player class handles marker selection, ensuring the flexibility to add new markers for different games.

4. Changes for Extendibility:

Introduction of Interface (BoardGame):

The primary change is the introduction of the BoardGame interface, defining a common set of methods for any turn-based game.
Dynamic Board Size in TicTacToe:

The TicTacToe class supports dynamic board sizing based on user input.
Modified Board for OandC:

The OandC class uses a modified version of the Board class to accommodate the unique requirements of Order and Chaos.
Super Tic Tac Toe (SuperTicTacToe):

Introduces a more complex game structure with a matrix of smaller Tic Tac Toe boards.

5. Conclusion:

The design prioritizes scalability and extendibility, allowing for the seamless addition of new turn-based games. The use of interfaces, inheritance, and modularity ensures a flexible architecture that can adapt to different game variants while minimizing code duplication.