Design Documentation

1. Class Structure

●GeneralBoard

An interface that contains abstract methods for displaying the board, moving pieces, and determining whether the piece's position is valid.

●TicTacToeBoard

Implements the GeneralBoard interface. Initialize the board layout, display the board, record the piece placement, and check for win or draw conditions.

●SuperBoard

Inherits from GeneralBoard, the initialization and display methods have been overridden.

●QuoridorBoard

Implements the GeneralBoard interface. In addition to implementing the abstract methods in the interface, it also initializes the board, checks whether walls can be placed, and uses BFS to determine if a player is completely trapped. In the board display, we use dots to represent positions where pieces can move, while walls are placed in the squares between the dots, and can be located using coordinates. A wall spans two points, indicating that neither of these points can be traversed.

●GeneralGame

Abstract general game class. Set up methods to handle draws, methods to print the summary table, and methods to reset team statistics.

●TicTacToeGame

Inherits from the GeneralGame class. Show Tic-Tac-Toe game logic, including features such as replaying the game, inputting or changing the board size, entering the current player's number, allowing players to choose their piece type, updating and displaying the board, determining and recording the outcome, and switching players.

●OrderAndChaosGame

Inherits from the TicTacToeGame class. Show Order and Chaos game logic, which is similar to Tic-Tac-Toe, but the `inputPlayer` and `choosePiece` methods have been overridden. This is because the team names are fixed as 'ORDER' and 'CHAOS', and players can choose either 'X' or 'O' in each round.

●SuperTicTacToeGame

Inherits from the TicTacToeGame class. Show Super Tic-Tac-Toe game logic. I have divided the super board into 9 mini boards, first checking the victory conditions for the mini boards before placing them into the super board for further checks.

●QuoridorGame

Inherits from the TicTacToeGame. Show quoridor game logic, including choosing player numbers(2 or 4), initializing players, choosing which team to start first, setting walls, making movements, checking wining situations, switching players, printing summary table and continuing the game. The reason why extending the TicTacToeGame because they have the similar game flow.

●Player

Record the relevant members required for the player.

●Team

Record the relevant members required for the team.

●Piece

Record the piece type.

●Tile

Record the piece status on the board cell.

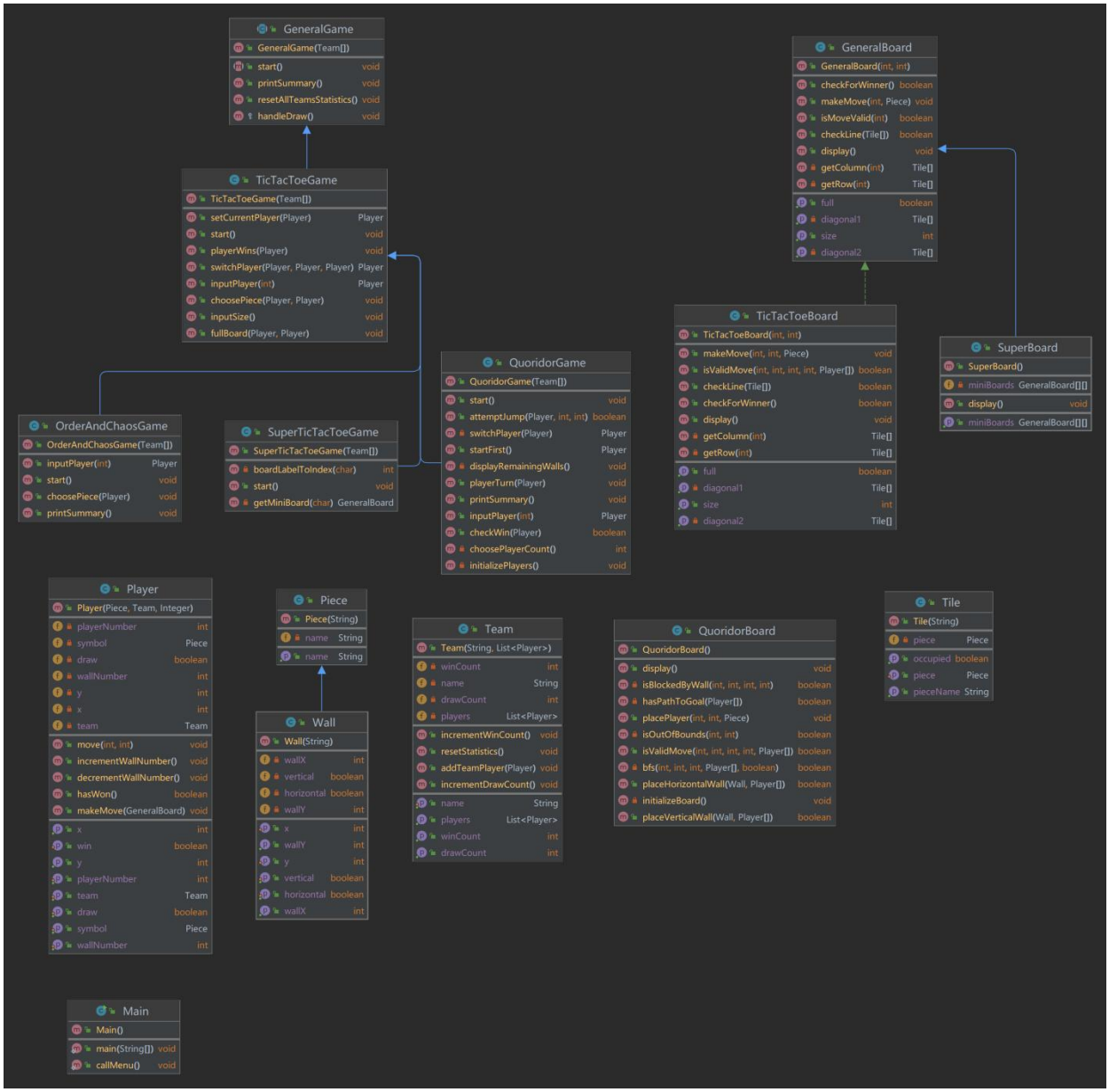
●Wall

Inherits from Piece where the status of the walls are stored.

●Main

Initialize and start the game.

2. UML graph



3. Evaluation

- This design document describes a scalable and extensible turn-based game framework. The framework supports various turn-based game variants such as tic-tac-toe, super tic-tac-toe, Chaos and Order, and Quoridor. By employing object-oriented design principles, we have created a flexible system that allows for the easy addition of new game variants without major changes to the existing code.

- **Scalability and Extendibility**:

By utilizing the abstract class GeneralGame and the general interface GeneralBoard, we have created a flexible foundation that can be easily extended to support new game variants. Each specific game class inherits from GeneralGame and implements its unique game logic. This design allows us to add new games without modifying the existing code. The QuoridorBoard class, along with other games, implement the GeneralBoard and adds logic specific to the Quoridor game. By extending it, we can reuse the general board logic from GeneralBoard while incorporating features unique to Quoridor.

Our design allows for easy expansion of existing games. The QuoridorGame class inherits from TicTacToeGame and adds game logic specific to Quoridor. By using inheritance, we can reuse the general game logic from TicTacToeGame while incorporating features unique to Quoridor.

- **Changes**:

1. A new Wall class has been added that inherits from the Piece class to handle wall-related implementations. Functions related to walls have been added to the player.
2. The logic of QuoridorGame is based on TicTacToe. Besides different winning conditions and movement methods, QuoridorGame is also a turn-based game that allows for player switching operations, printing summary table and continuing the game as set in TicTacToe. However, in this game, it seems easier to represent player positions as points rather than squares. Therefore, our code uses points to indicate reachable positions, allowing players to place walls between points. A key point is that when players meet, they will jump over their opponent. If the landing position has a wall blocking it, the system will prompt the player to choose whether to jump left or right. As for the 4-player situation, we have also implemented by iterating the players and comparing the coordinates that when multiple players meet, it's forbidden to jump more than 1 player. Additionally, this game does not allow for draws; after one player wins, the menu displays the winning player and their total victories, and asks whether to continue the game.
3. QuoridorBoard implements the GeneralBoard interface, and it also implements its own specific features based on the characteristics of the game. Since the board needs to display both pieces and walls, the actual board size is 17x17. A Tile array is responsible for recording the display status of various pieces on the board, while a Wall array is used to track the placement of walls, which is applied to BFS and other correctness checks. A wall placed cannot extend beyond the boundaries, cannot overlap with a previous wall, cannot have more than the number of walls each player has, and cannot completely trap an opponent. Determining whether a move is valid mainly involves checking if the piece moves out of bounds and whether the next action will hit a wall. BFS has been used to ensure that every player has a path to the target point.
4. Adding a GeneralBoard interface which allows for other board to implement, which makes the program much easier to extend.

- When selecting the initial code architecture, we conducted a detailed comparison of our respective codes. Yijia's code structure showcases a more complete game flow and an excellent object-oriented design implementation for multiple board games (Tic Tac Toe, Order and Chaos, Super Tic Tac Toe). Through inheritance and polymorphism, each specific game class derives from the GeneralGame base class, possessing common methods while implementing its own game logic according to different scenarios.

She also wrote the GeneralBoard interface, which makes it easier to extend QuoridorBoard, and programming to interfaces aligns better with the Open/Closed Principle. Additionally, a wall is also a type of piece, so using Wall to inherit from Piece and apply it to various operations on the Board makes sense.