Steps for LLD -

Problem - Design Tic Tac Toe

Design and implement a **Tic Tac Toe** game where two players take turns marking spaces in a N**xN grid**. The game should handle invalid moves, determine the winner, and announce the game result appropriately.

Requirements

1. Game Setup

- Provide a way to launch the game
- The game is played between **two players**.
- Each player **enters their name** before the game starts.
- The first player chooses their **symbol (X or O)**, and the second player automatically gets the other symbol.
- The game board is a **nxn grid**, initially empty.
- Players take turns choosing an **empty cell** to place their mark in a round robin method

2. Valid Moves & Gameplay

- Players enter a move as **(row, column)** (0-based index).
- A move is **valid** if, the cell is **within bounds** (0 ≤ row < nand 0 ≤ column < n) and the cell is **empty** (not already marked X or O).
- If an **invalid move** is attempted, Display an **error message** explaining why the move is invalid.
- **Do not switch turns** until a valid move is made.

3. Winning & Draw Conditions

- A player **wins** if they form a row, column, or diagonal with **n of their marks (X or O)**. (equal to grid size).
- If the board is full and no player has won, the game ends in a **draw**.
- The game should **stop immediately** once a winner is found or the board is full.

4. Play Against an AI (Single Player Mode)

- Allow the user to choose **Human vs. Human** or **Human vs. AI** mode.
- Implement a **basic AI** (random move)
- In the future, an **advanced AI** with difficulty levels can be introduced, using **optimal move strategies** for better gameplay.

Step 1 - Identify the Entities (Noun)

Design and implement a **Tic Tac Toe** game where two players take turns marking spaces in a **NxN grid**. The game should handle invalid moves, determine the winner, and announce the game result appropriately.

Requirements

1. Game Setup

- Provide a way to launch the game
- The game is played between **two players**.
- Each player **enters their name** before the game starts.
- The first player chooses their **symbol (X or O)**, and the second player automatically gets the other symbol.
- The game board is a **nxn grid**, initially empty.
- Players take turns choosing an **empty cell** to place their mark in a round robin method

2. Valid Moves & Gameplay

- Players enter a move as **(row, column)** (0-based index).
- A move is **valid** if, the cell is **within bounds** (0 ≤ row < nand 0 ≤ column < n) and the cell is **empty** (not already marked X or O).
- If an **invalid move** is attempted, Display an **error message** explaining why the move is invalid.
- **Do not switch turns** until a valid move is made.

3. Winning & Draw Conditions

- A player **wins** if they form a row, column, or diagonal with **n of their marks (X or O)**. (equal to grid size).
- If the board is full and no player has won, the game ends in a **draw**.
- The game should **stop immediately** once a winner is found or the board is full.

4. Play Against an AI (Single Player Mode)

- Allow the user to choose **Human vs. Human** or **Human vs. AI** mode.
- Implement a **basic AI** (random move)
- In the future, an **advanced AI** with difficulty levels can be introduced, using **optimal move strategies** for better gameplay.

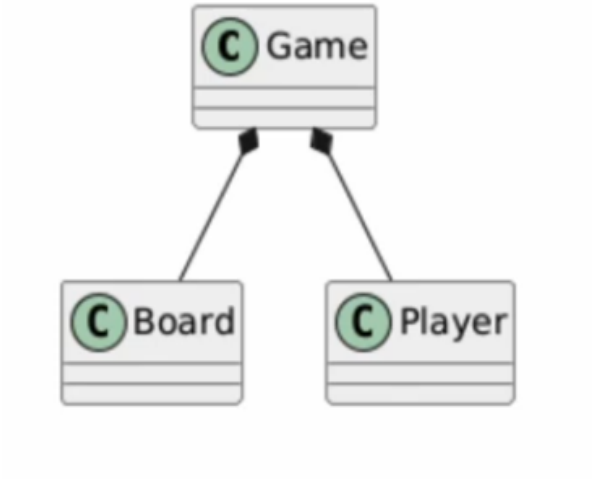
Step 2: Identify the relationship between the Entities

Try to develop relation between different entities, Convert entities into **is-a, has-a, and has-many** relationship.

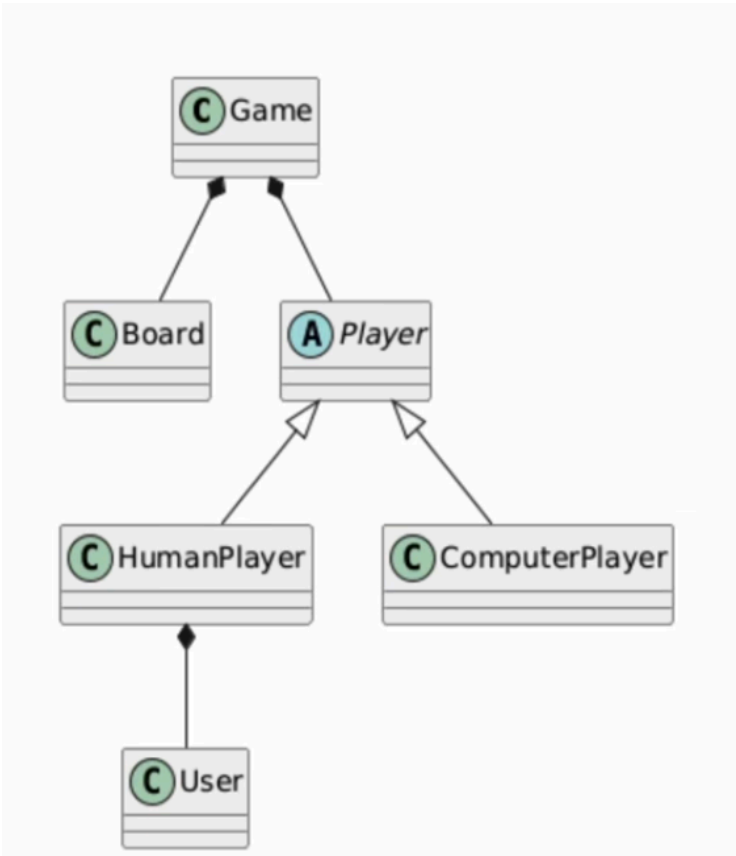
1. **Game has Board**
2. **Game has users XX - But user is not good attribute as we can play with Bot as well**
   Game has many player

3. **Human is a Player**
4. **Bot is a Player**
5. **Computer has many strategies**

Step 3: Convert to a class diagram for clarity



Game has Board and Player - But Player **is a not a concrete class**



Now Player is a abstract class and also we take humanplayer as User.

Step 4: Identity the Features of the System

Design and implement a **Tic Tac Toe** game where two players take turns marking spaces in a N**xN grid**. The game should handle invalid moves, determine the winner, and announce the game result appropriately.

Requirements

1. Game Setup

- Provide a way to launch the game
- The game is played between **two players**.
- Each player **enters their name** before the game starts.
- The first player chooses their **symbol (X or O)**, and the second player automatically gets the other symbol.
- The game board is a **nxn grid**, initially empty.
- Players take turns choosing an **empty cell** to place their mark in a round robin method

2. Valid Moves & Gameplay

- Players enter a move as **(row, column)** (0-based index).
- A move is **valid** if, the cell is **within bounds** (0 ≤ row < nand 0 ≤ column < n) and the cell is **empty** (not already marked X or O).
- If an **invalid move** is attempted, Display an **error message** explaining why the move is invalid.
- **Do not switch turns** until a valid move is made.

3. Winning & Draw Conditions

- A player **wins** if they form a row, column, or diagonal with **n of their marks (X or O)**. (equal to grid size).
- If the board is full and no player has won, the game ends in a **draw**.
- The game should **stop immediately** once a winner is found or the board is full.

4. Play Against an AI (Single Player Mode)

- Allow the user to choose **Human vs. Human** or **Human vs. AI** mode.
- Implement a **basic AI** (random move)
- In the future, an **advanced AI** with difficulty levels can be introduced, using **optimal move strategies** for better gameplay.

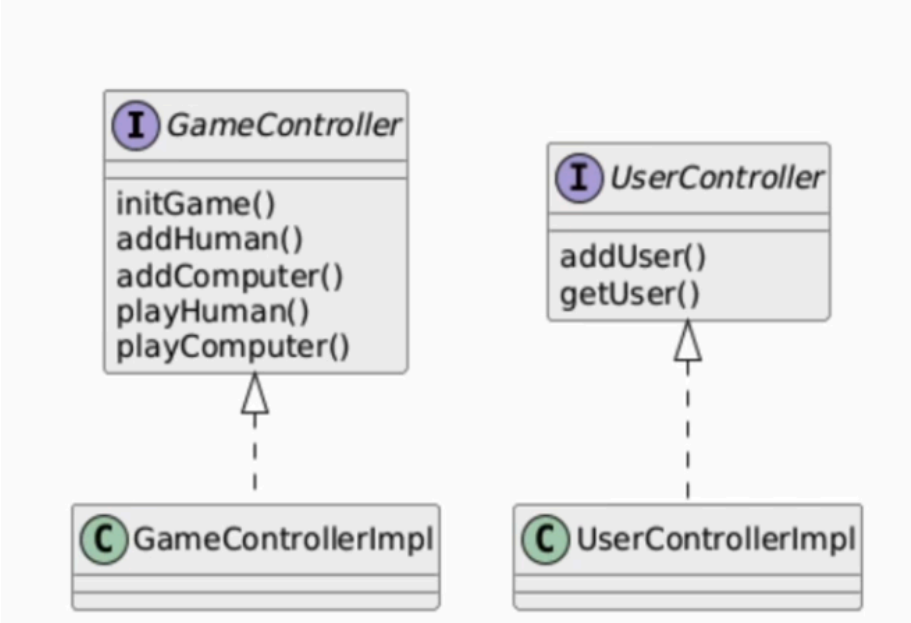
We can categorize into internal and external features

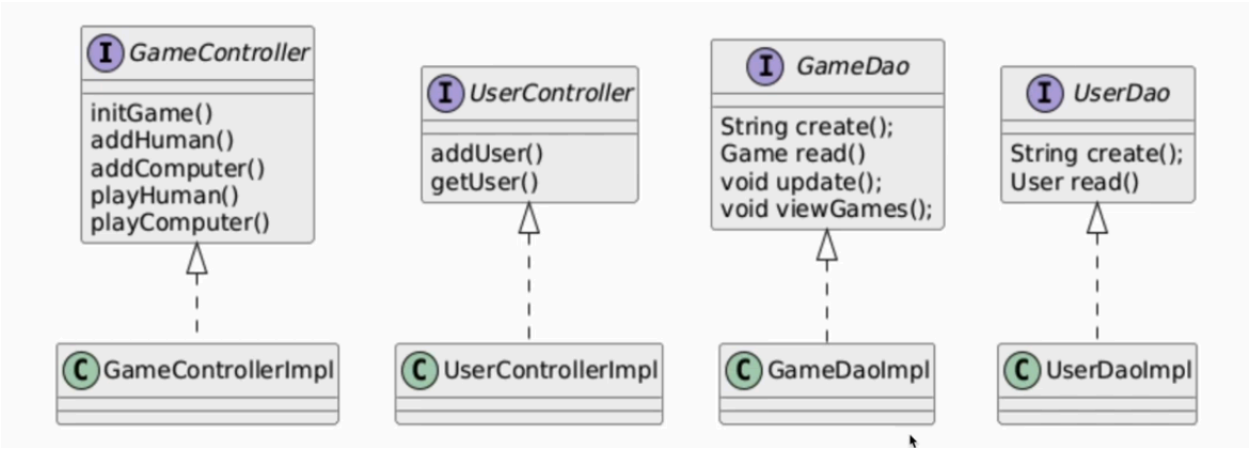
Internal features

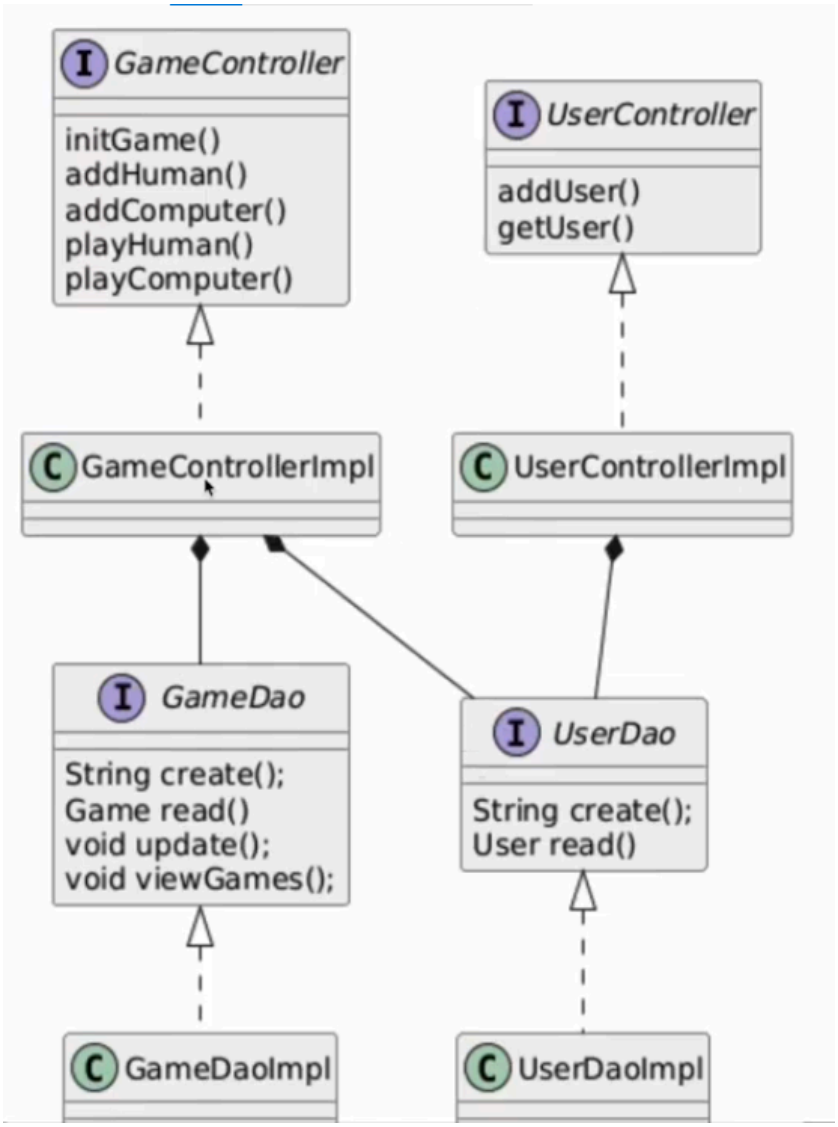
1. Determine valid and invalid moves
2. Check winning and draw conditions
3. Implement AI strategy

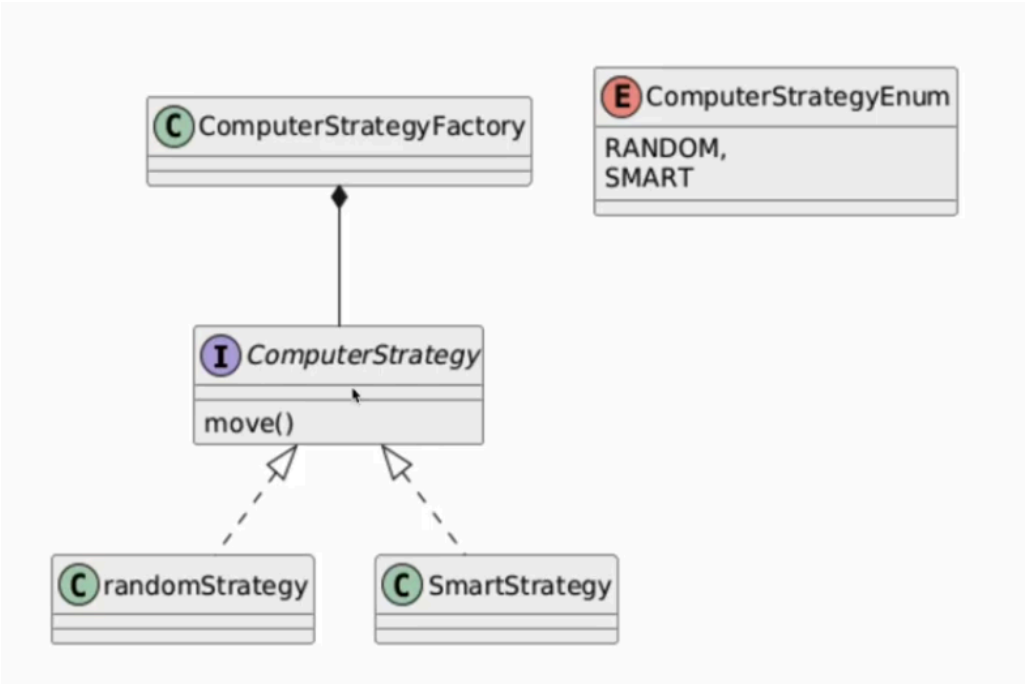
External features

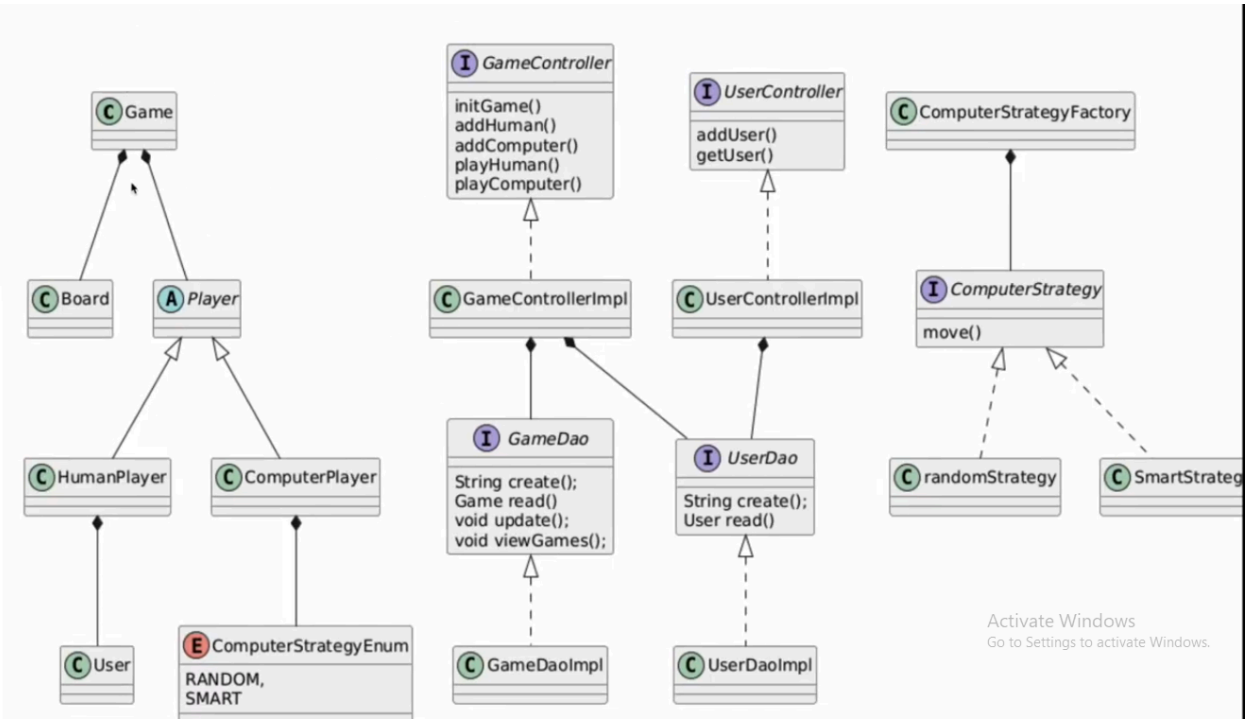
1. Launch the game
2. Add a player or user
3. Add a computer player
4. Execute moves for human and computer players




GameController
initGame()
addHuman()
addComputer()
playHuman()
playComputer()
UserController
addUser()
getUser()
GameDao
String create();
Game read()
void update();
void viewGames();
UserDao
String create();
User read()
GameControllerImpl
UserControllerImpl
GameDaoImpl
UserDaoImpl


GameController
initGame()
addHuman()
addComputer()
playHuman()
playComputer()
UserController
addUser()
getUser()
GameControllerImpl
UserControllerImpl
GameDao
String create();
Game read()
void update();
void viewGames();
UserDao
String create();
User read()
GameDaoImpl
UserDaoImpl


ComputerStrategyFactory
ComputerStrategyEnum
RANDOM,
SMART
ComputerStrategy
move()
randomStrategy
SmartStrategy


Game
GameController
initGame()
addHuman()
addComputer()
playHuman()
playComputer()
UserController
addUser()
getUser()
ComputerStrategyFactory
Board
Player
GameControllerImpl
UserControllerImpl
ComputerStrategy
move()
GameDao
String create();
Game read()
void update();
void viewGames();
UserDao
String create();
User read()
HumanPlayer
ComputerPlayer
randomStrategy
SmartStrateg
User
ComputerStrategyEnum
RANDOM,
SMART
GameDaoImpl
UserDaoImpl
Activate Windows
Go to Settings to activate Windows.