Features:

1. Tic Tac Toe Game
    1.1 Undo option.
    1.2 Multiple players can Play.
    1.3 Grid Size can vary.
    1.4 Any symbol more than (X,O) can be used.
    1.5 Rules can be modified.
    1.6 Announce Winning Player.
    1.7 You can also play against Machine.

2. Enhanced TicTacToe
    2.1 More Complex winnig system.
    2.2 All the features of TicTacToe is provided.

3. Hex Shaped TicTacToe
    3.1 Hexagon Shaped Grid.
    3.2 Rest features are same.

Feature Implementation:

Undo - By Cloning the Grid. (Completed)
Multiplayer - Doubly_Ended_Queue. (Completed)
Multiple Game Types - Interfaces. (Completed)
Machine Move - Optimal move algo function. (Completed)

Design Decision:

* Doubly Ended Queue is used for carrying multiple players because it is more easy to iterate in that rather than ArrayList.
* Cloning of grid is used for Undo Operations although Stack would also work.
* 2-D Array is uesd for Grid because its easy to pass and can access any value easily.

GameDesign v2.0 - Requirement I

<Completed> - 1. Tic-Tac-Toe consists of 3x3 Square Cells
    Grid Size is fixed so used a 2-D Array.
<Completed> - 2. Game Between Two Humans
    Used Dequeue to carry all players.
<Completed> - 3. Game Between Human and Machine
    For Machine I have made a function which will make a optimal move.
<Completed> - 4. Winning Criteria - 3 Cells in Row/Column/Diagonal are in Same State.
    Just Simple Implementation.
<Completed> - 5. Announce Winning Player
    Just Simple Implementation.

GameDesign v2.0 - Requirement II

<Completed> - 6. Enhanced Tic-Tac-Toe Game Consist of 9x9 Squares...

Made a function which can be extended in future.

<Completed> - 7. Enhanced Tic-Tac-Toe will continue to expand in depth levels...

Made a function which can be extended in future.

<Completed> - 8. Extend Game to 4x4 Board

Already Implemented Variable size Grid.

<Completed> - 9. Human Player is Biased...

By Cloning the current Grid.

<Not Completed> - 10. Storing and Retrieving Game State

Stack Implementation

<Not Completed> - 11. Store Players Game Statistics: Leaderboard

Class Leaderboard will be sufficient.

GameDesign v3.0 - Requirement III

<Completed> - 12. Super Tic-Tac-Toe Game Extends Enhanced Tic-Tac-Toe Game...

Every new Board_type Classes Should implement a common Abstract Class.

<Completed> - 13. Design Winning and Losing Criterias On All Edges...

Just Implementation in Rule Classes.

<Not Completed> - 14. Incorporate Irregular shaped Hexagonal Boards

Move Decision of Machine was a trick part.

GameDesign v4.0 - Requirement IV

<Not Completed> - 15. Incorporate Biased Game Board

Feature Specific Design Decision?

<Not Completed> - 16.Incorporate Connect Four Game In Design

Feature Specific Design Decision?

<Not Completed> - 17. Discover Newer Abstract Types

Feature Specific Design Decision?

<Not Completed> - 18. Refactor and Reuse Code In Both Games

Feature Specific Design Decision?

SECTION III: How to Run/Test Your Code?

Q.Describe How To Run Your Code

1. Compile and Run the command by simple commands.
2. It will ask for number of players --> Give an integer N input as no. of players (eg. 2)
3. It will ask for information of players -> In next N lines mention (Name + Type + Symbol)
    - 3.1 Type can be only Human/Machine
    - 3.2 Symbol should be a single Char.
4. In next line it will ask for playing Super tic toe --> Answer yes or no in one line

if "yes"

5. Next line it will ask for the depth of the SuperTicToe. --> integer input (eg. 3)
6. Game Starts and you move according to the coordinates --> integer pair input (eg. 4 5)

if "no"

5. It will ask for the size of the grid --> integer pair input (eg. 9 9)
6. Game Starts and you move according to the coordinates --> integer pair input (eg. 4 5)
7. It will print who wins after a dead state will occur.

Q.Can I Run Test.java to test your whole source code?

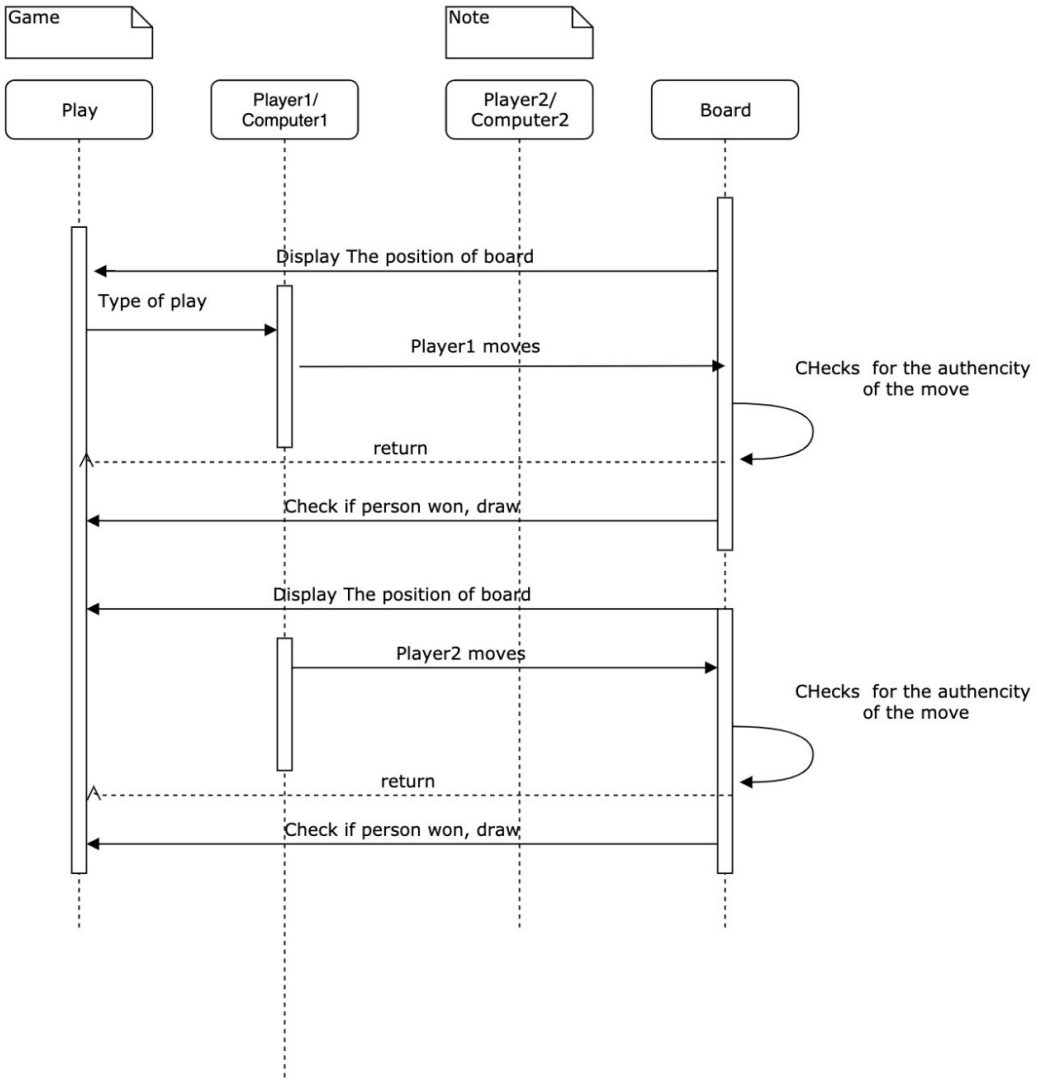
Yes.

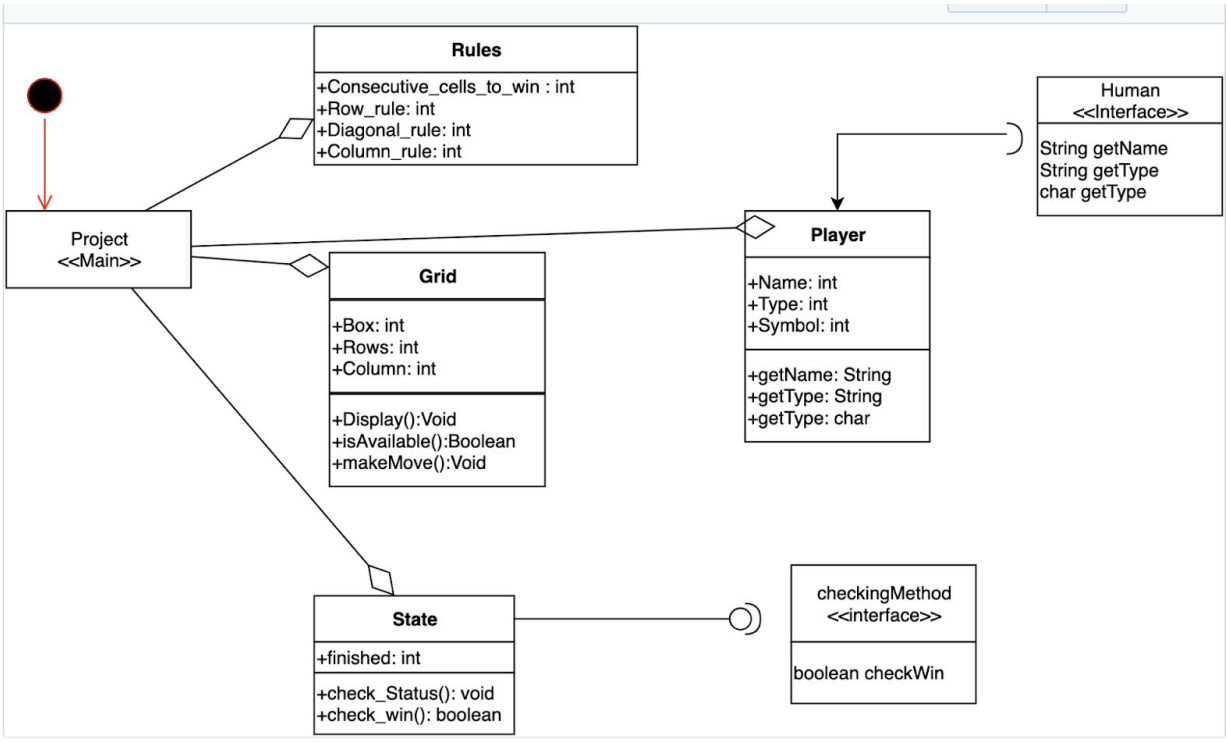
Classes and Interfaces.

1. Rules (This class contains information about the rules how the game has to be played)
2. Board_type (interface)
    2.1. Grid (This class contains board of grid shaped and has the information of board)
    2.2. Hexagonal (This class contains board of hex shaped and has the information of board)
3. Blocked_state (This class knows about the cell information where you cant move)
4. Human (interface)
    4.1. Player (This class contains information about the players like how many games he won or his symbol or his type)
5. Checking_Method (interface)
    5.1. State (This Class calculate the result of a Move.)
6. Project (contain main Function)

Some Imp Functions Used.

* optimal_move() : Calculate the optimal move for machine after analysing the situation.
* Calculate_level() : Calculate the grid level that helps further in implementation.
* Start_Game() : Start the Game after taking some basic inputs the user and initialize the enviornment.
* Check_status() : Returns the current state from Board information.
* Compute_for_grid_board() : Function specifically simulating result for Grid Shaped.
* Compute_for_Hex_board() : Function specifically simulating result for Hex Shaped.
* checkDiagonalHorizontalVertical() : Returns some result after analysing the situation using help of rules.


Game
Note
Play
Player1/
Computer1
Player2/
Computer2
Board
Display The position of board
Type of play
Player1 moves
CHecks for the authencity
of the move
return
Check if person won, draw
Display The position of board
Player2 moves
CHecks for the authencity
of the move
return
Check if person won, draw


Rules
+Consecutive_cells_to_win : int
+Row_rule: int
+Diagonal_rule: int
+Column_rule: int
Human
<<Interface>>
String getName
String getType
char getType
Project
<<Main>>
Player
+Name: int
+Type: int
+Symbol: int
+getName: String
+getType: String
+getType: char
Grid
+Box: int
+Rows: int
+Column: int
+Display():Void
+isAvailable():Boolean
+makeMove():Void
State
+finished: int
+check_Status(): void
+check_win(): boolean
checkingMethod
<<interface>>
boolean checkWin