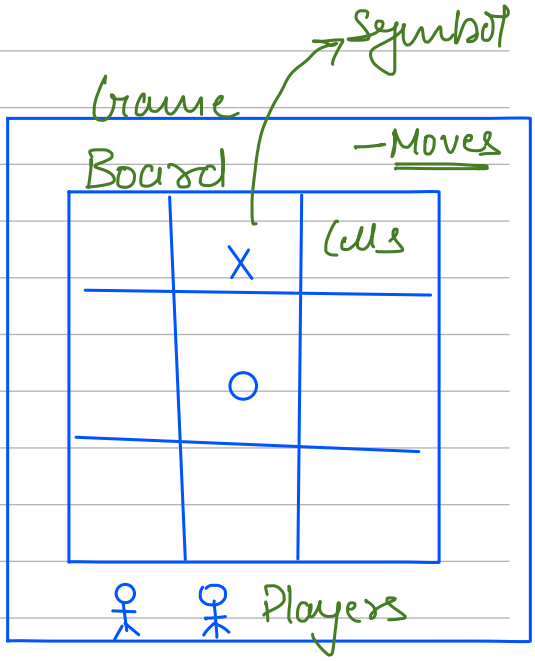
Agenda

Class diagram of TicTacToe

Class diagram
- Nouns
- Visualisations

- Entities (Class | Interface | Abs. Classes | Enums)
- Design Patterns



Game
- Board
- List<Player>
- List<Move>
- nextPlayer Index
- GameState → Start, InProgress, Ended
- ~~isEnded~~ → Start, End, Pause
- Player winner
- List<GameWinningStrategy>

MVC System
↓
GameService

GameState
InProgress,
Ended,
Draw,
Paused,

Board

- size
- List<List<Cells>>

① List<List<Cells>> ✓ Matrix

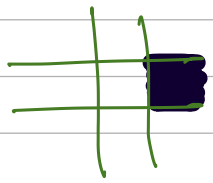
② List<Cells>

Cell

- row
- col
- isEmpty
- CellState → Occupied, Empty, Blocked
- Player

CellState

Occupied,
Empty,
Blocked



Player

- name
- symbol
- isBot
- PlayerType

PlayerType

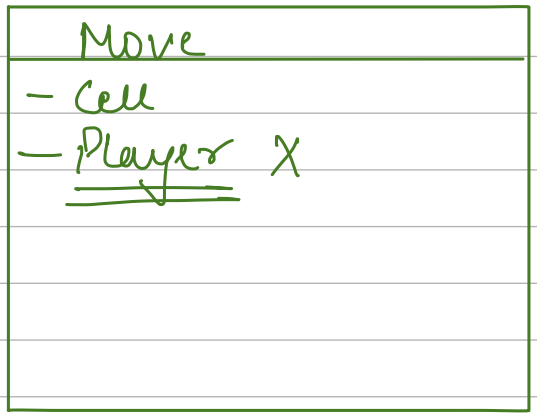
Human,
Bot,
Aliens,

Bot

- Bot difficulty level
- BotPlayingStrategy

BotdifficultyLevel

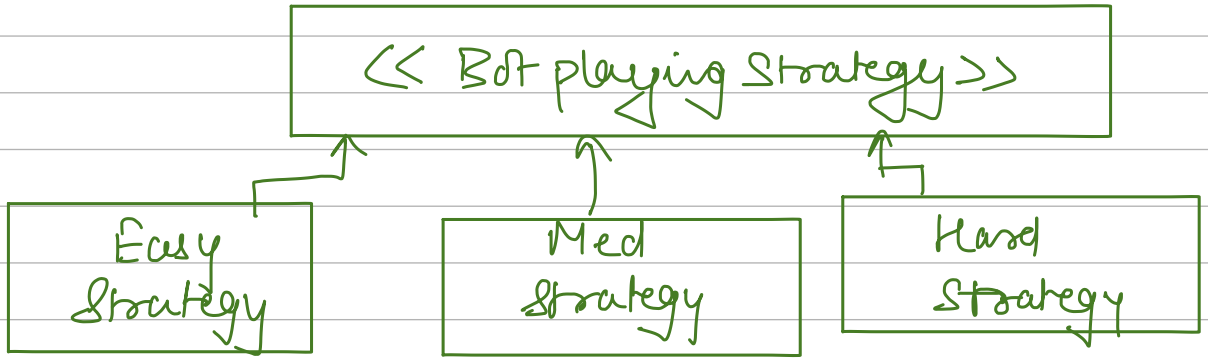
EASY,
MED,
HARD



⇒ Design Pattern

- Game
  - Lot of attributes + Validations

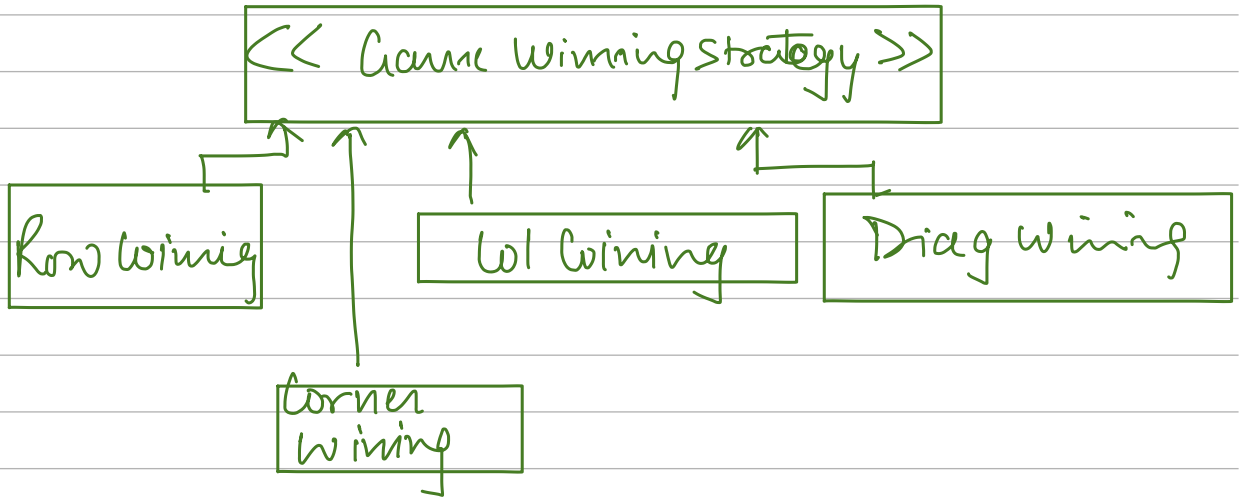
- Bot can play in multiple ways

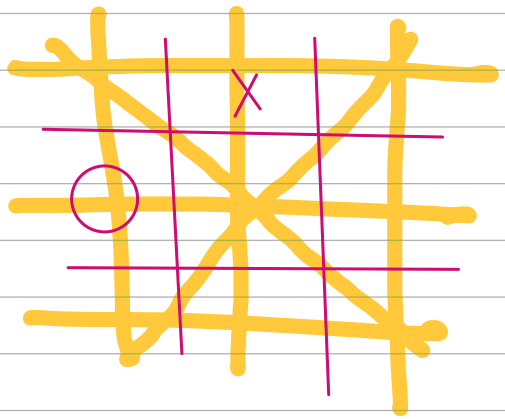


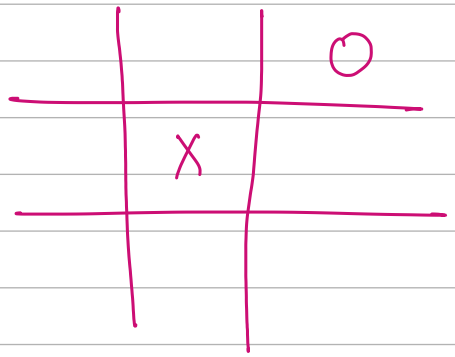
Bot playing Strategy Factory

BotplayingStrategy getBotPlayingStrategyforDifficultyLevel(BotDifficultyLevel)

- Game can be won in multiple ways







Approach 1 with every move, we should check for a winner.

```
for every player
{  for (i = 0 to N)
       for (i = 0 to N)
           if symbol matches in all rows
```

Rows $N \times (N^2)$

Cols

$N \times (N^2)$

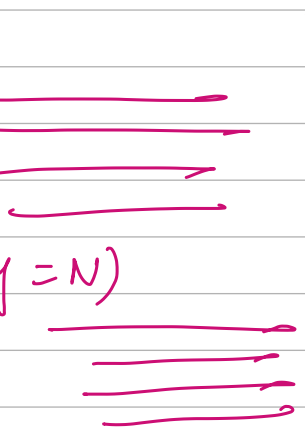
for every player in every col

Diag

if $(x=y)$

$O(N)$

$(2N) \times N$ if else $(x+y=N)$



for every player Diagonals Condition

Time Complexity $N(N^2+N^2+2N)$

$O(N^3)$

Approach 2 If we know the current player we can check only Row / Diag / Cols for that symbol

$(N^2+N^2+2N) \times 1$ $O(N^2)$

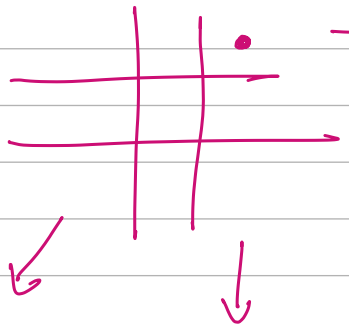
Approach 3

Corner NXY

$(N+N+2N) \times 1 = O(N)$

Approach 4

If we store count of every symbol for every row | col | diag



→ Map < symbol, count) row

Map <symbol, count> col 3

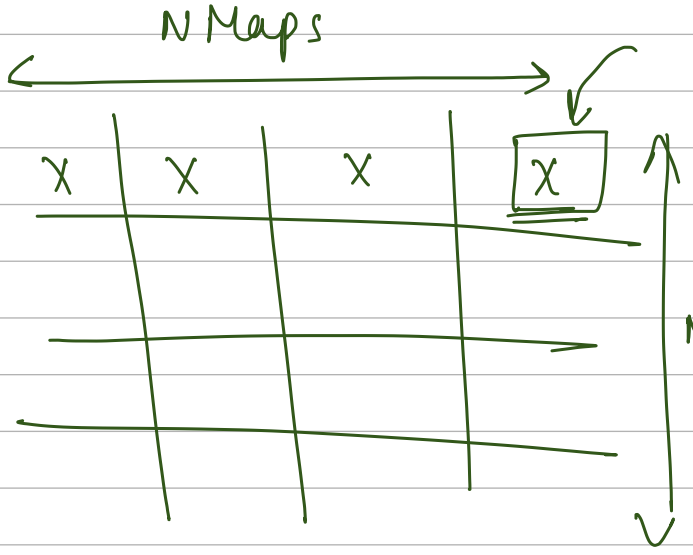
Map <symbol, count) diag

if (count == N) winner — O(1)

Space Complexity

O (2N)

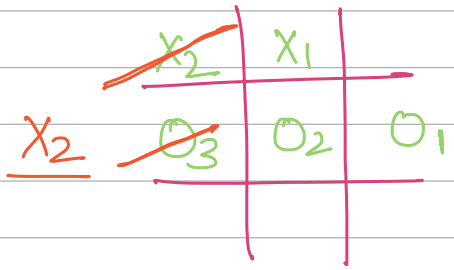
O(N³) → O(1)



row Map (x, 3)
Map (x, 4)
4 = N
O(1)

Map (x, 0)
(x, 1)

UNDO



○ Saket
X Prithvi

UNDO ⇒ Global Button
⇒ Any player can press any number of times

⇒ When a UNDO will be pressed it will remove the most recent move

List<Move> moves;

App1

List<Move>

1) Remove the latest move from the list
2) Remove the symbol from the cell
(UNDO the state of Board)

In a game like Chess, it is going to be very difficult.

App2 If we create a New game and Redo all Moves except the last one

List<Moves>

Time Complexity — O(List<Moves>)

App3

List<Moves>
List<Board>

Restore the previous Board state

Borad = Previous Boarsd

Space complexity

Size of Chess = 64B

64 x 1000 B
= 64 Kb

Summary

1) If reversing is easy then
   => Store list of Moves
   => Remove the last Move from the list
   => Remove the last Move from the board.

2) If Reversing is Not easy

II
1) Store the Moves
2) Create a new board
3) Redo all the Moves except the last Move

TC ↑

III
1) Store the Moves as well as Board for every Move
2) Remove the Latest Move & Board from the list

SC ↑

=> UNDO Strategy

Revise Venn Diagram