Design TicTacToe (Zero kaanta)

Agenda

TicTacToe

1. Overview ←
2. Req gathering and Clarification ←
3. Class diagram ←
4. ~~Schema Design~~
5. ~~Code~~

How to approach LLD interviews

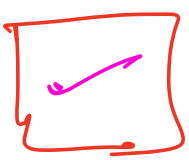
1. How to do Schema Design
2. Best practices for code.

TIC TAC TOE

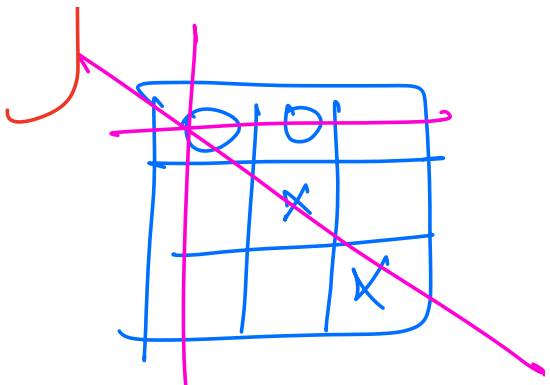
1. Overview

- Do you know
  - 0. What we know
  - 1. what kind of system
    - entity/ web/ desktop
  - 2. persisting data →

Do you not know



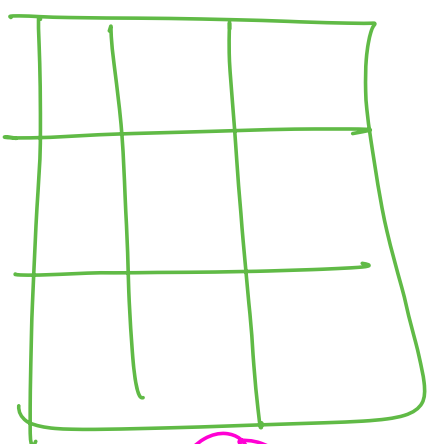
③ taking input



② Req Gathering → Suggest ideas with whys

(Most of those features are also going to be valid feature for almost every other game)

→

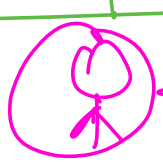


① Size of board can be any → n x n

② # of players = n-1 ?

③ We want to support bots

④ Bots can be of diff difficulty level.



⑤ Every player has a diff symbol. Symbol → Char

⑥ At max one bot per game. → Validate distinct symbol

~~⑧ Supporting Tournaments~~

How many is not a good ques^n to ask
- ⇒ it doesn't affect design
- ⇒ (0) or more than 1 is okay

⇒ ~~##~~ ✓

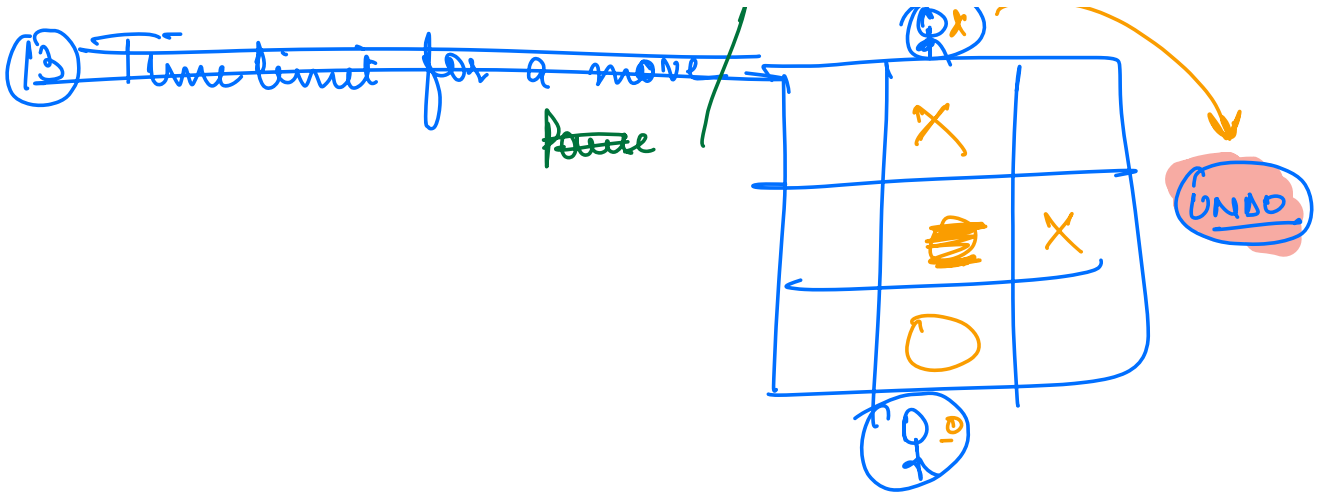
~~⑨ Supporting leaderboard / Score of players~~ ✗

~~⑩ Team~~

⑪ Undo

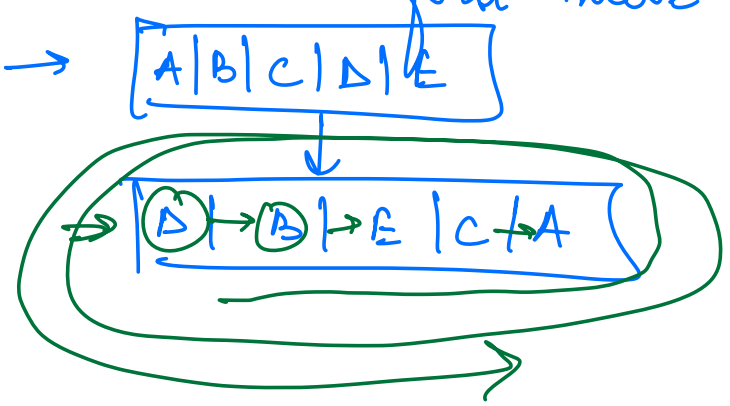
- → global undo
- → doesn't depend on who clicked it
- → move also reverts back

~~⑫ Redo~~



14 When / How will a game start
When / How will a game end.

1.) Who will make the first move

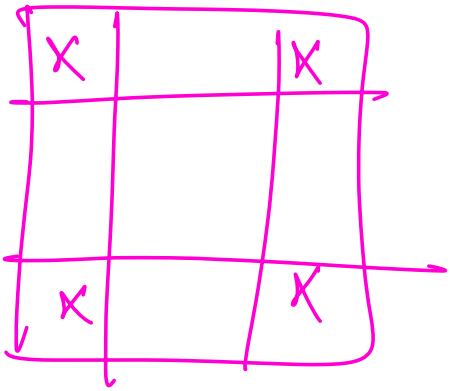


2.) How will a game end

a.) when someone has won / draw

b.) ~~when all but one have lost~~

3.) What decides a victory..



→ Else we should allow to add new ways in which someone can win

<<Winning strategy>>

→ when we start a game, we configure ways in which you can win it.

(15) What to do when someone exits:

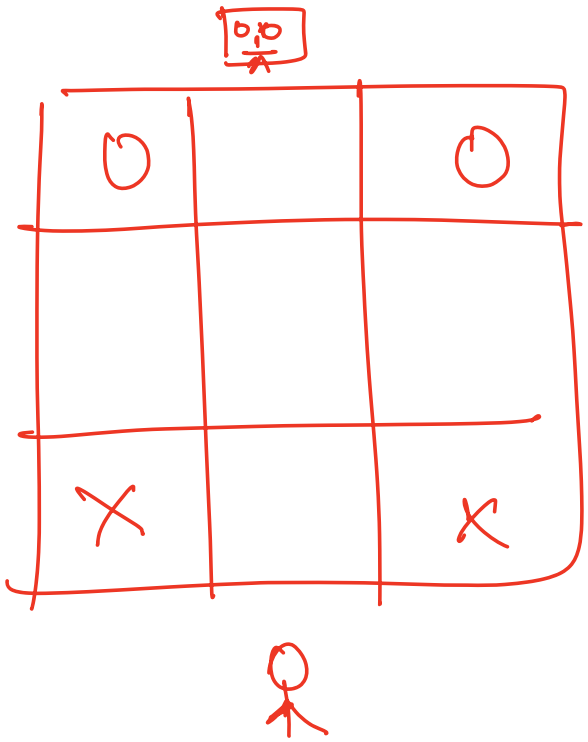
a.) Remove their symbols

b.) End the game.

c.) Replace By Bot

d.) NOT SUPPORTED ✓

(16) Replay the game.



→ Don't spend more than 5 mins in Req gathering

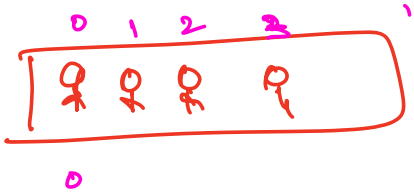
→ 5-6 core features =

Break till 10:12 PM

→ Class Diagram of TTT

(Visualize → Out → Dry)

→ How to implement Undo

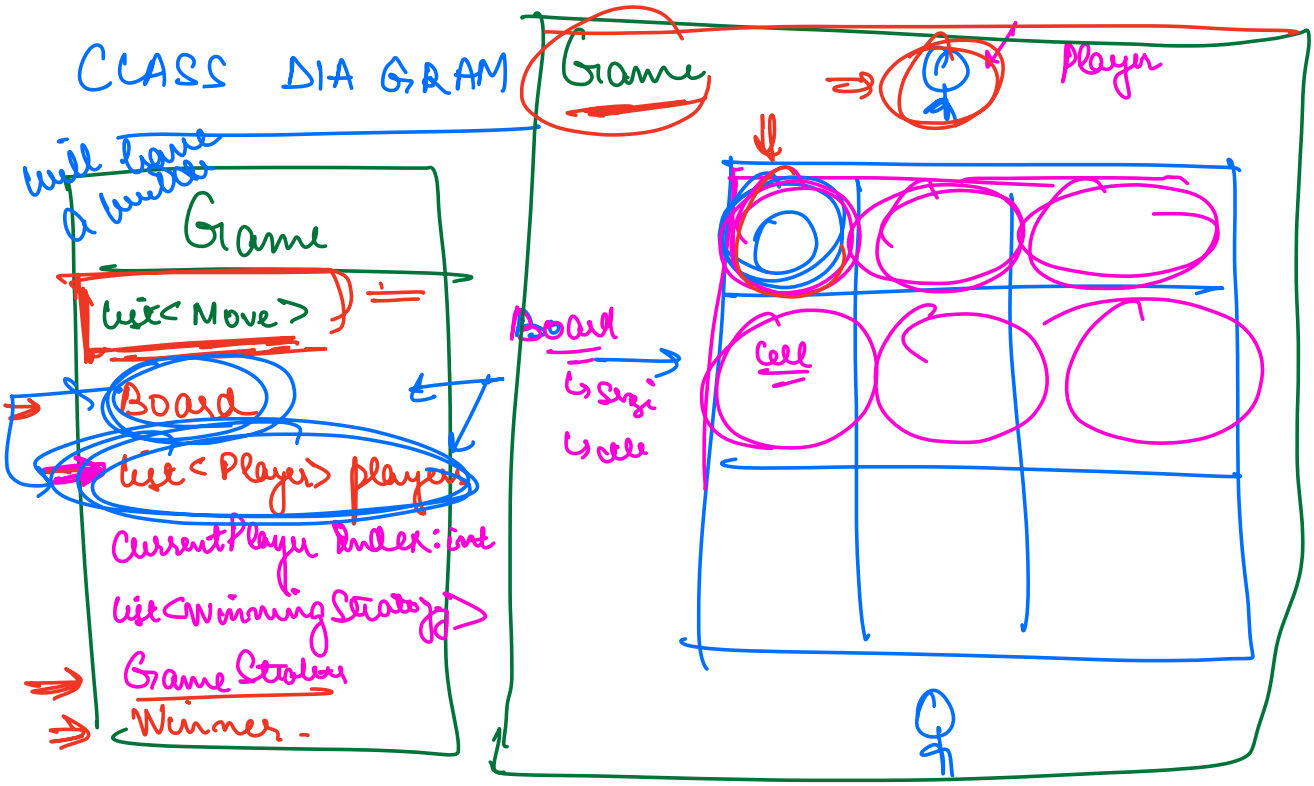


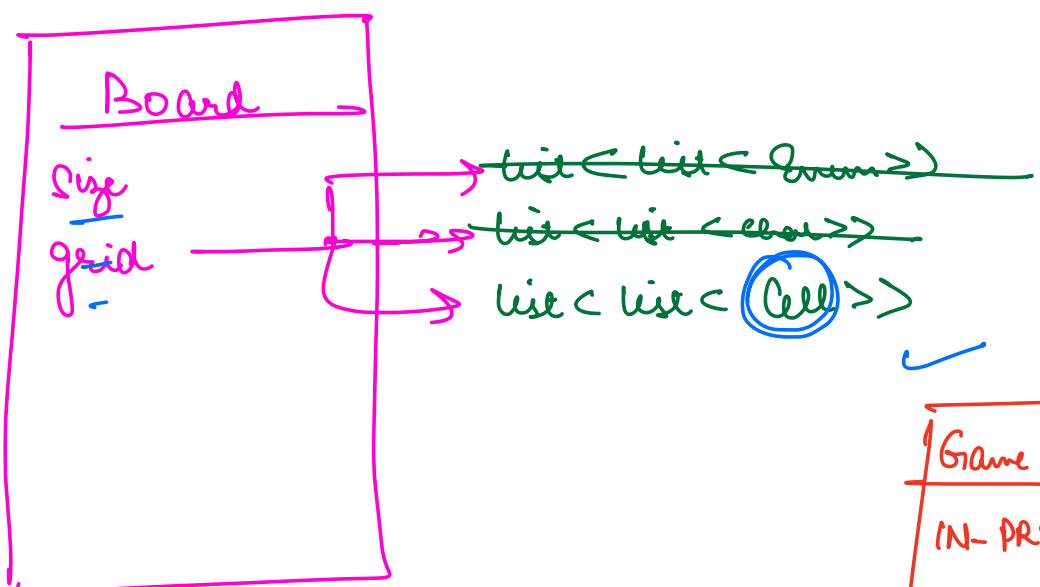
→ Kuch Kuch Hota Hai

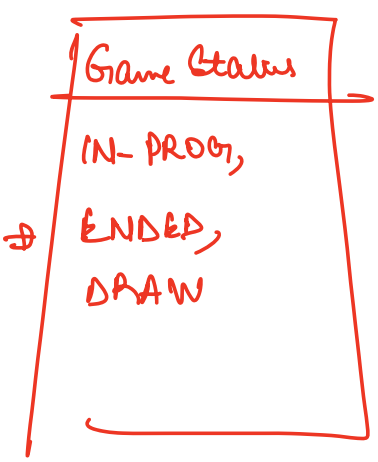
→ One Chance One

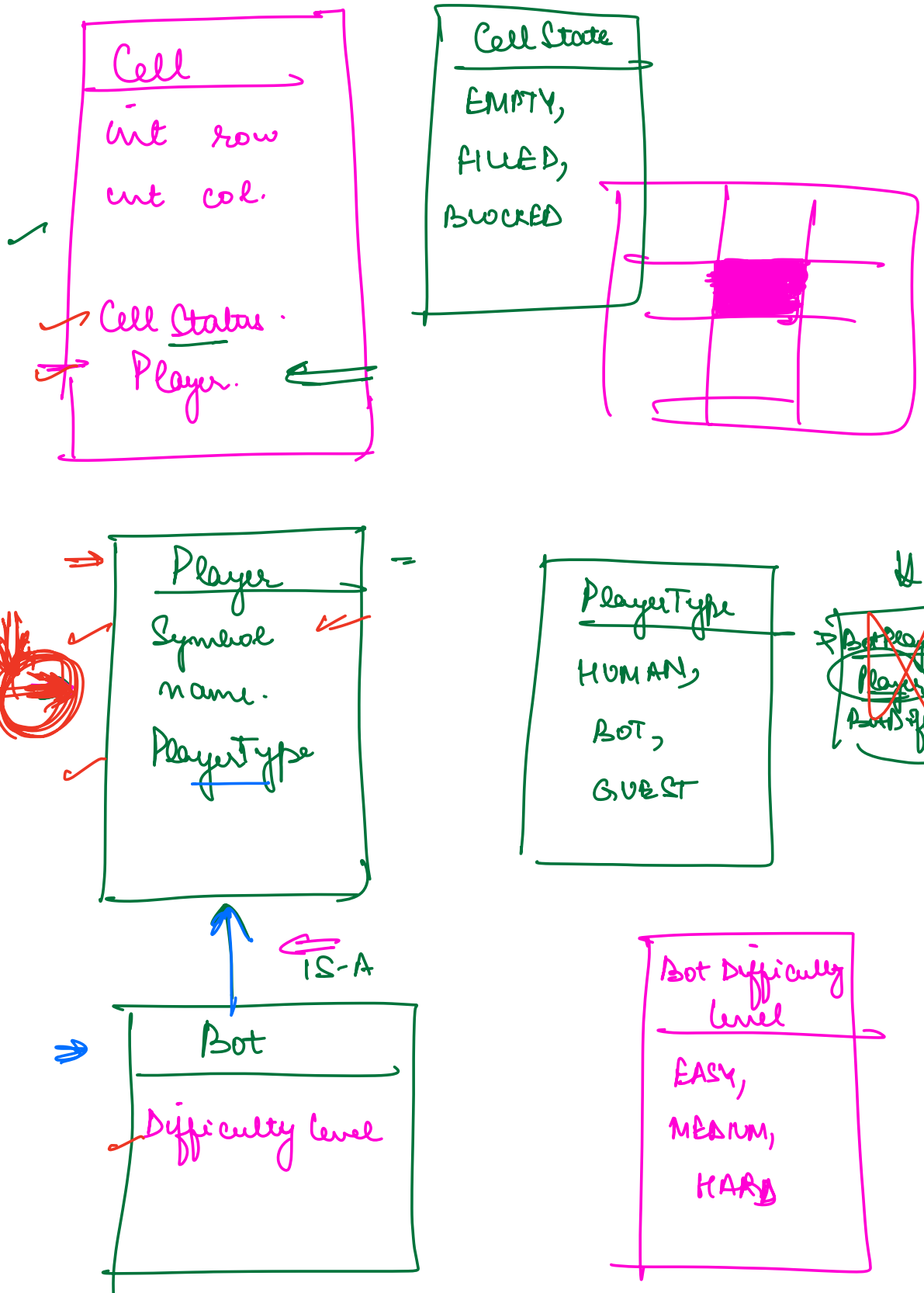
↳ Doraemon

CLASS DIAGRAM

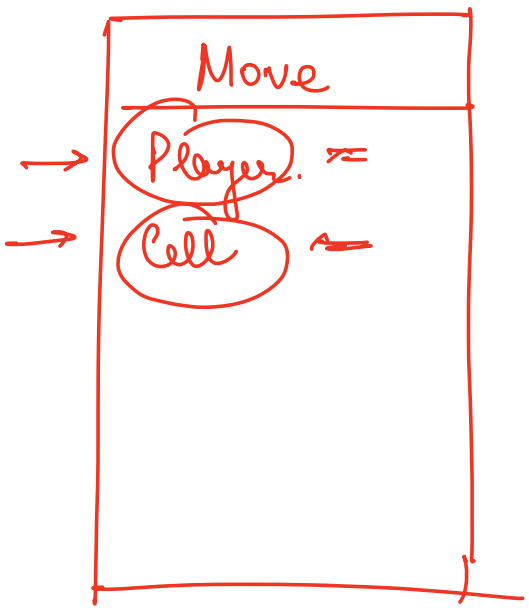


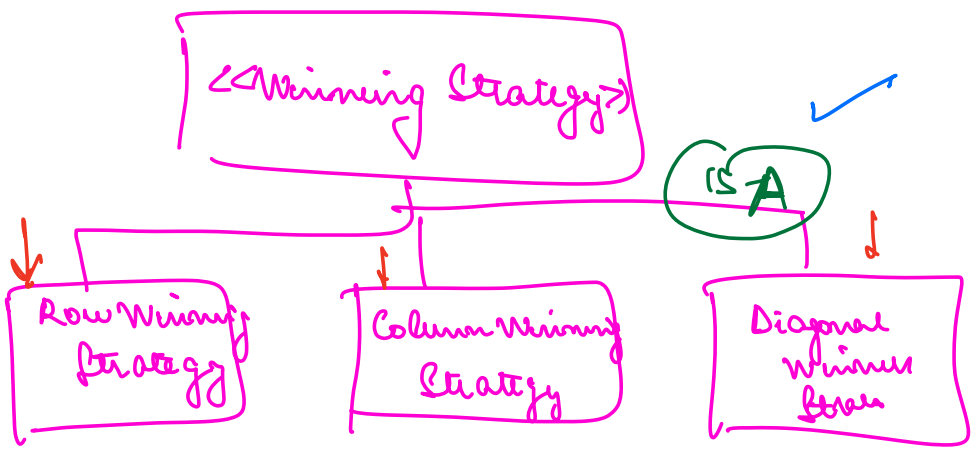


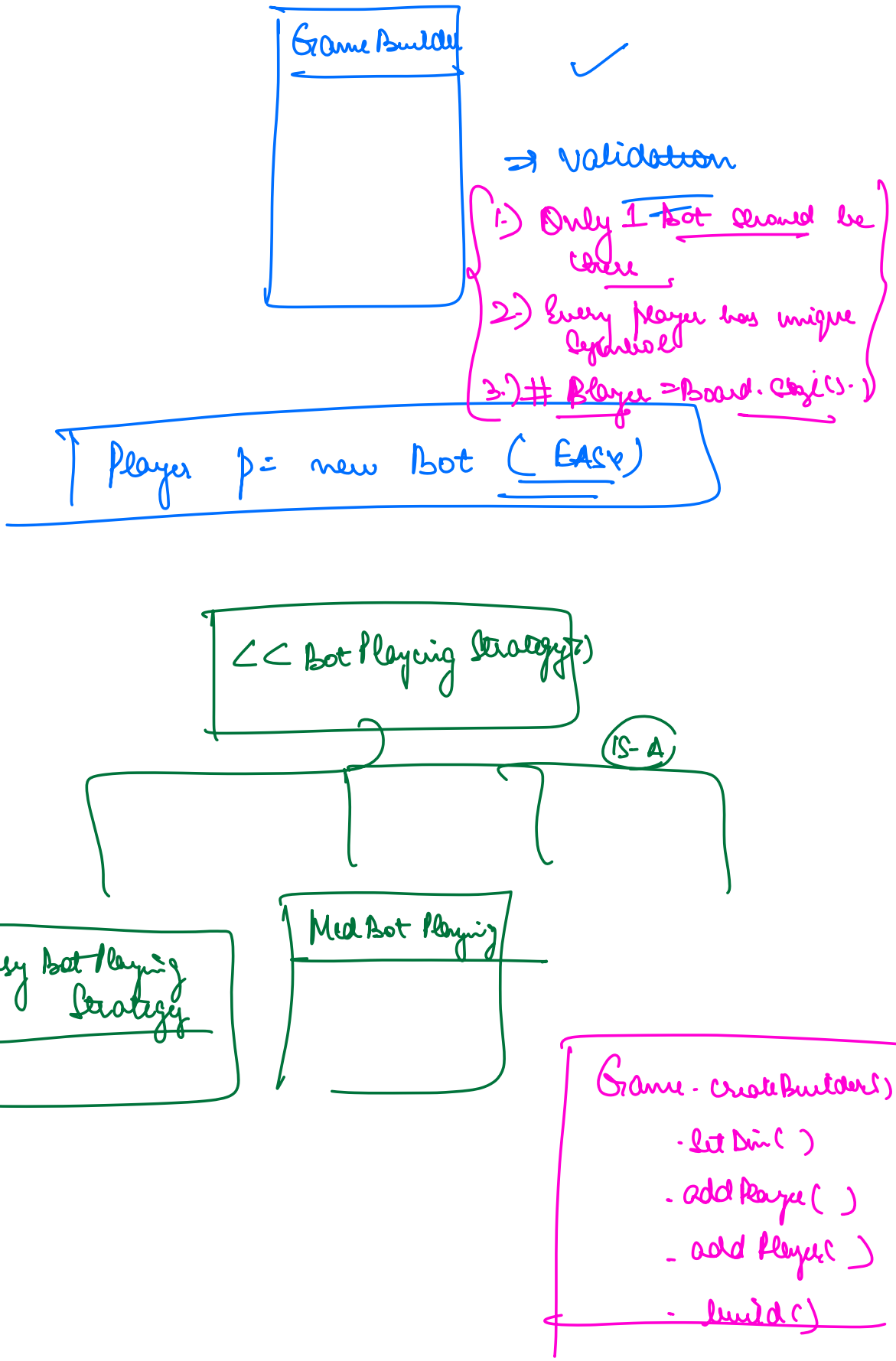


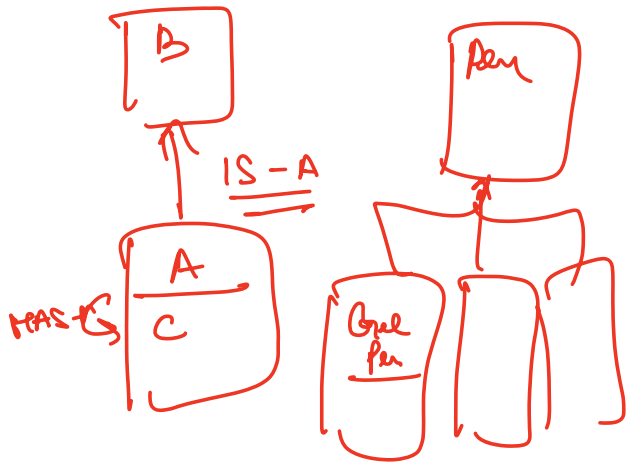

Cell
int row
int col.
Cell Status.
Player.
Cell State
EMPTY,
FILLED,
BLOCKED
Player
Symbol
name.
PlayerType
PlayerType
HUMAN,
BOT,
GUEST
IS-A
Bot
Difficulty level
Bot Difficulty level
EASY,
MEDIUM,
HARD

→ Don't make something unnecessarily Singleton






GameBuilder
⇒ validation
1.) Only 1 Bot should be there
2.) Every player has unique symbol
3.) # Player = Board.dim() - 1
Player p = new Bot (EASY)
<< Bot Playing Strategy >>
IS-A
Easy Bot Playing Strategy
Med Bot Playing
Game.createBuilder()
.setDim()
.addPlayer()
.addPlayers()
.build()



Draw.io

Google Docs

↳ Google Drawing

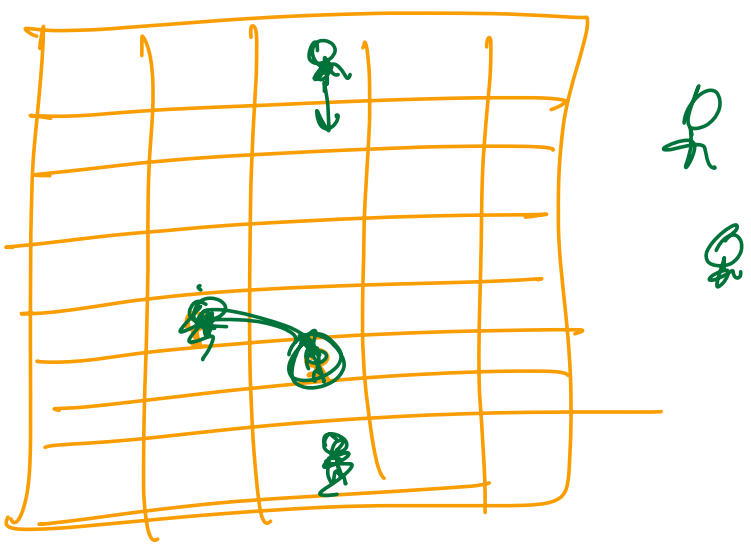
How to implement Undo

Easy

in TTT, when undo happens:

1. Clear cell
2. remove move from list
3. move turn to prev player.

But let's think of chess



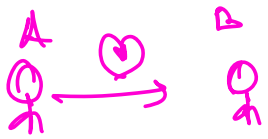
Across all types of games, there are 3 popular ways to do undo. Use which one is easiest.

not always possible → a.) Kuch Kuch Hota Hai

b.) Om Shanti Om

c.) Dokkemon

	TC	SC
	O(1)	O(1)
	O(N)	O(1)
	O(1)	O(N)



married someone else

A is single.

how can A get back to B ?

undo what the problem says and that's it

→ undo the affect of the last move

⇒ reverse what the last move did you and done.

Eg in TTT:

Undo:

→ removed from list<move>

just reverse all you did in make Move

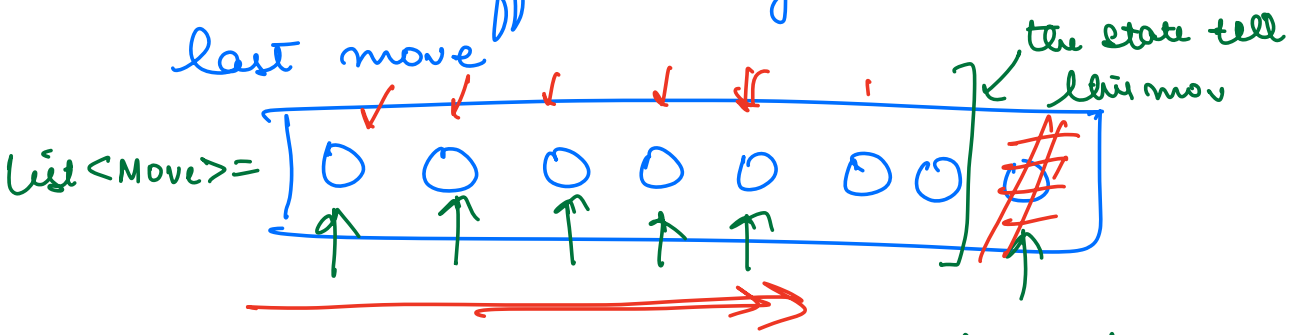
→ Remove Player, Symbol from that cell.

→ CurrentPlayerInd -= 1

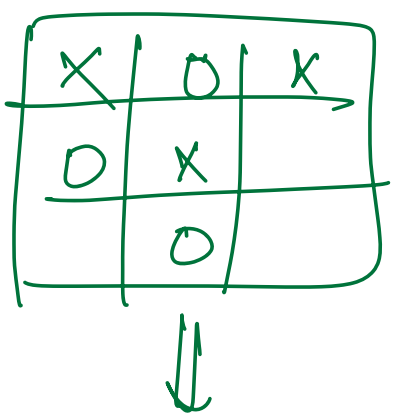
⇒ But for games like chess not easy to do.

Om Shanti Om

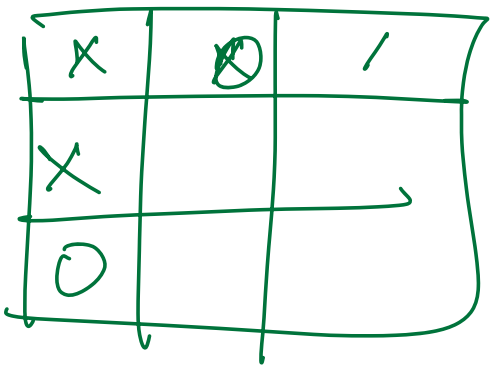
When it is difficult to just revert the last move



1. Clear the board.
2. Remove last move from list<move>
3. redo all moves



→ Easier to implement
→ it is slow

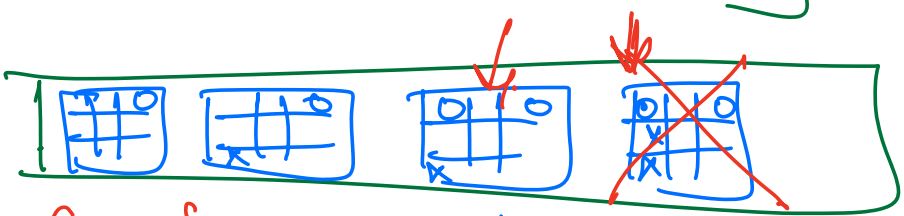


Doraemon

Store the state of game as well after every move.

List <Move>

List <Board>



Game {
board = []
board = state.[-1]

undo:

1) remove last state
2) the last state in list will now become your current state

⇒ 2/3 more classes where we will code

TTT