Design TicTacToe

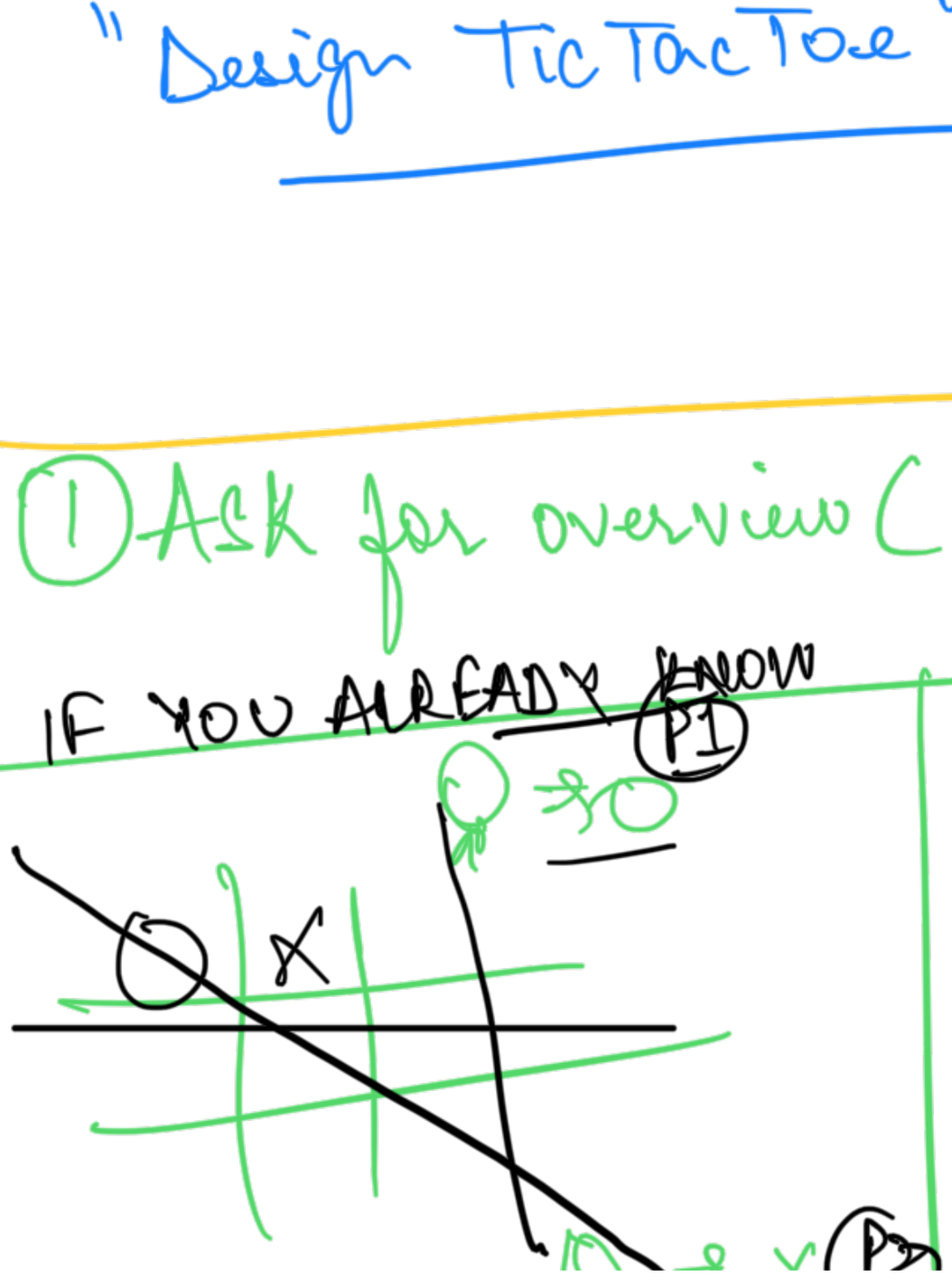
"Design TicTacToe"

Entity
→ Pen

Games
→ TicTacToe
→ Snake and Ladd

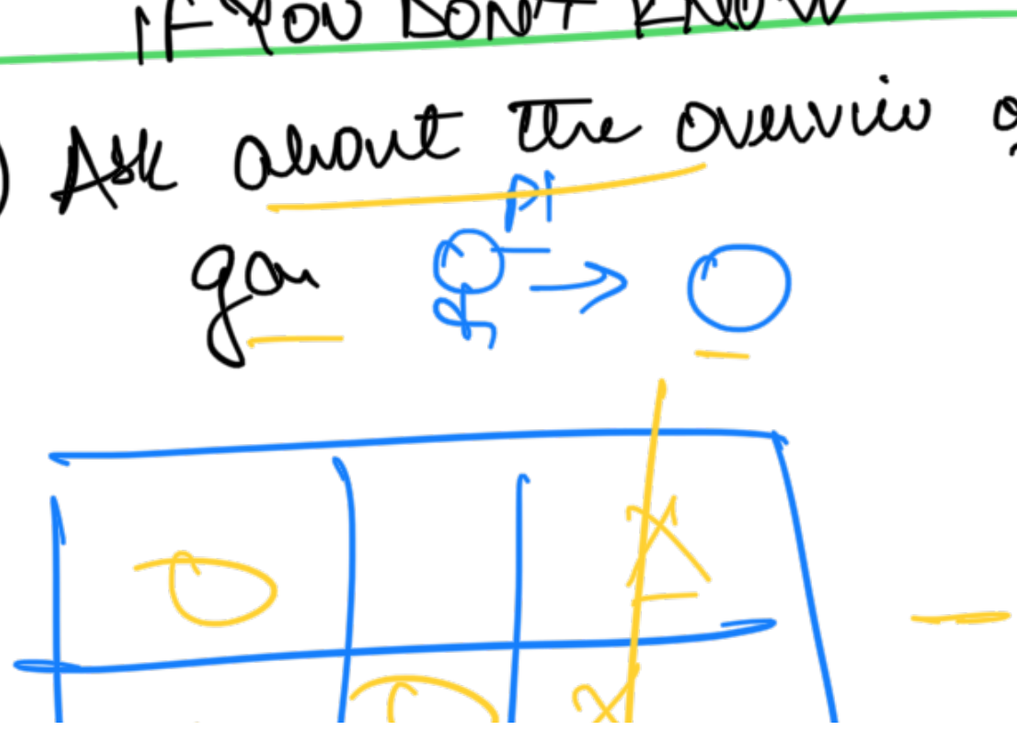
① Ask for overview (context)

IF YOU ALREADY KNOW



IF YOU DONT KNOW

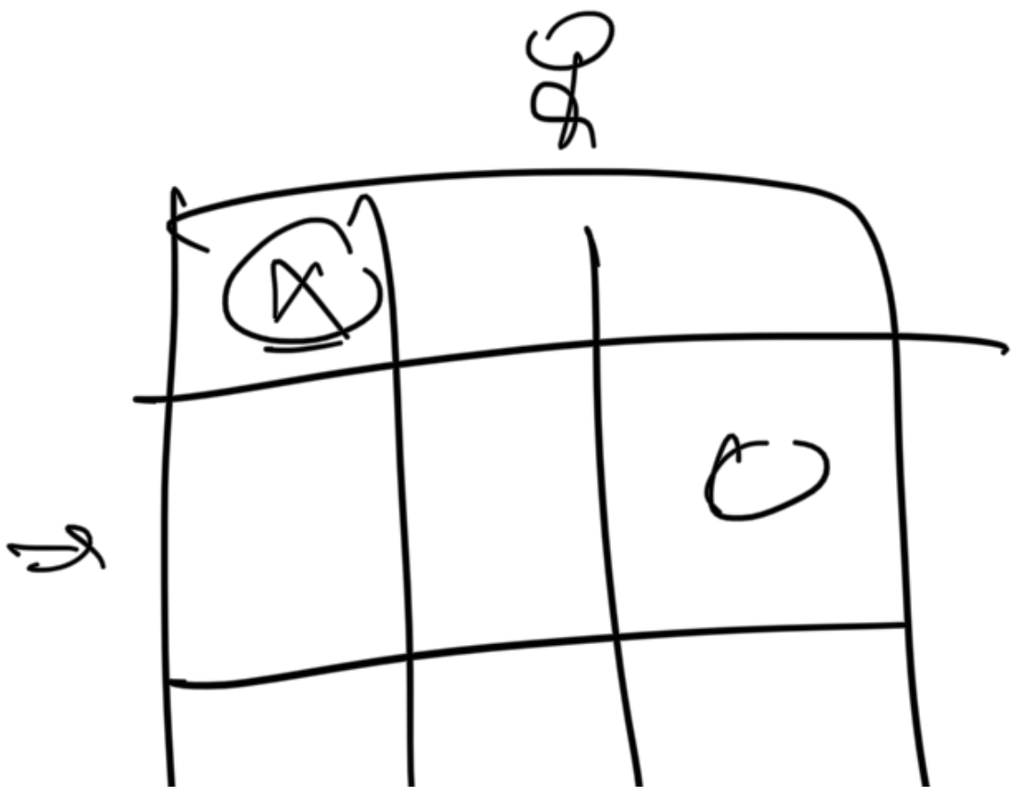
① Ask about the overview of game



→ what I know about the game.

→ ask about the contact

→ Ask about the contact



Strategies, Service

① Entity ← Controller, Board, Game, Symbol

② Desktop App^{n} ⇒ Running on a machine

(Run Through)

Controller

Strategy

Dynamic Cycle

③ Web App^{n} (Persisting Data)

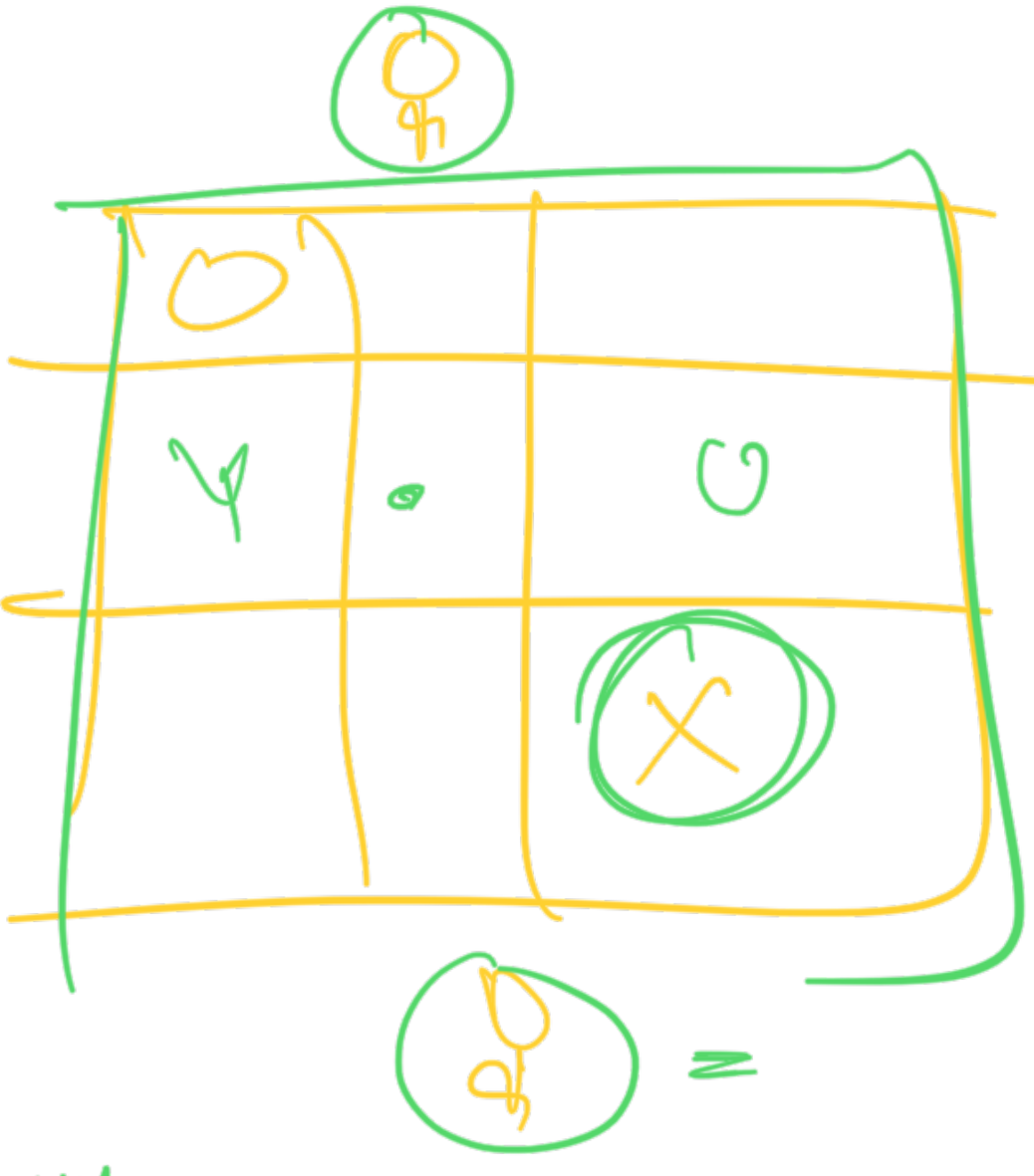
→ Database / Repo

Gathering Requirements

(a) Make Visualization

(b) Suggest Ideas

(c) Ask about diff feature



(1) There can be multiple players

② Board size can be any n x n

③ Each player can choose their own symbol.
Every player should have a unique symbol

④ There can be bots (Computer Players)

One game can have atmost one bot

⑤ Bots can have diff difficulty levels -

⑤ Any random player can be starting player

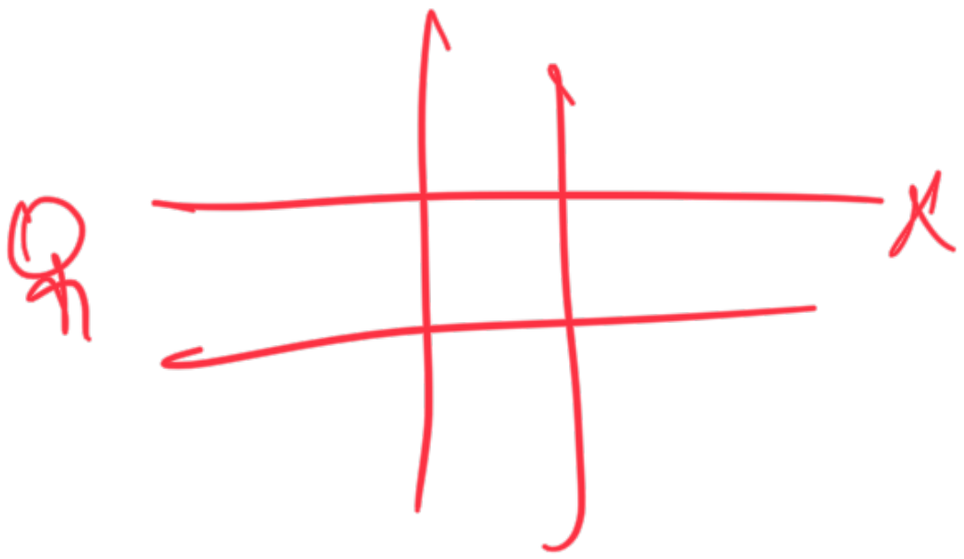
⑦ Undo feature

4 players

How to Start

→ Who will take the first turn

→ what will be the order of turns



How to Start

How to End

How to End

→ When will game end when someone has one OR DRAW

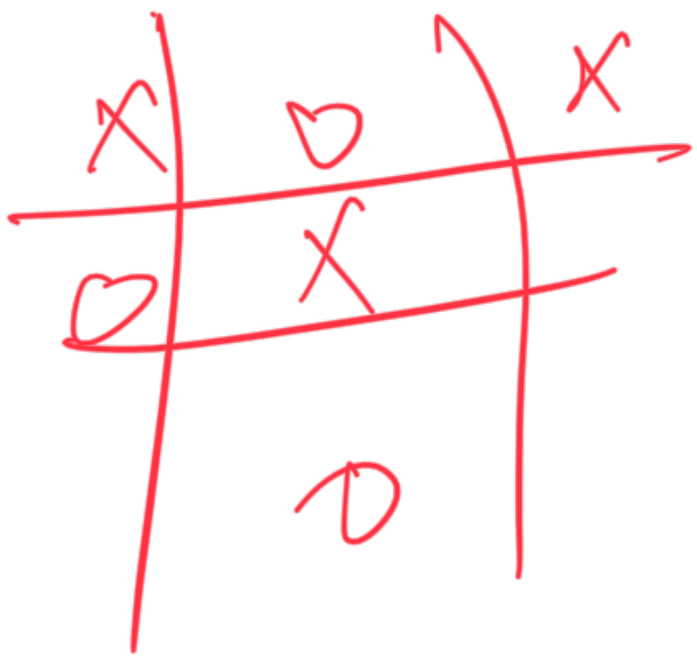
Or until game end when there is only a single player left

How do we decide victory → Columns / Row / Diagonal

can there be multiple diff ways to win in one

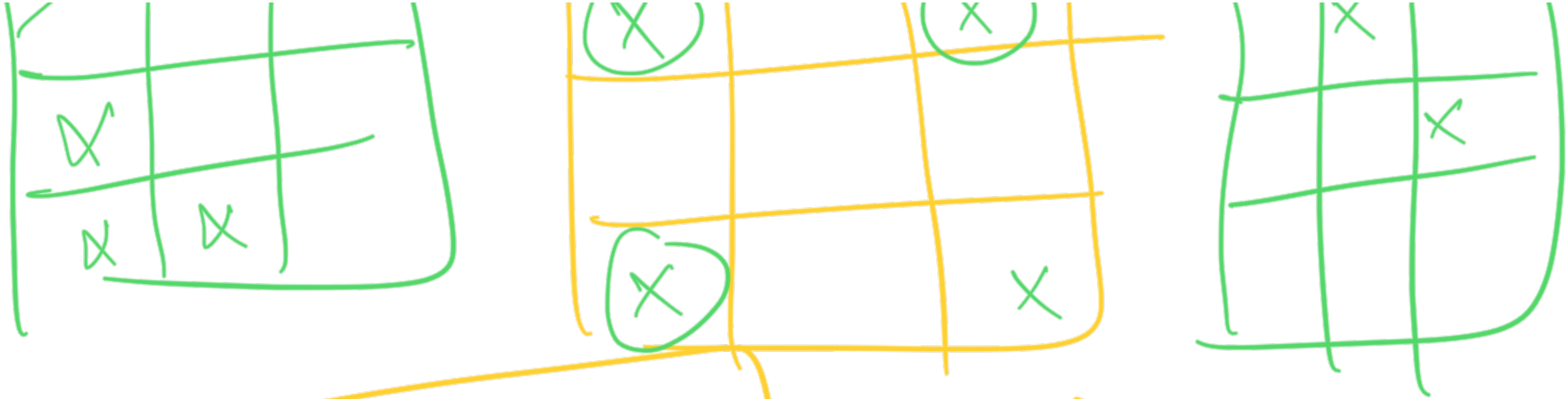
→ AI/ML

Easy → Random All



Medium →

Column / Row with least empty cell



Many LLD Problems have a hidden DSA Problem

→ Splitwise

→ TicTacToe

PayTM

Chess

Golf

FEATURES:

1.) Computer App's
2.) Any # of players
3.) Bot Players (AtMAX 1)

4.) Every player their own symbol [X, O, ...]

5.) n x n board

6.) Diff. ways to win

7.) Support undo operation (UNDO any # of moves)

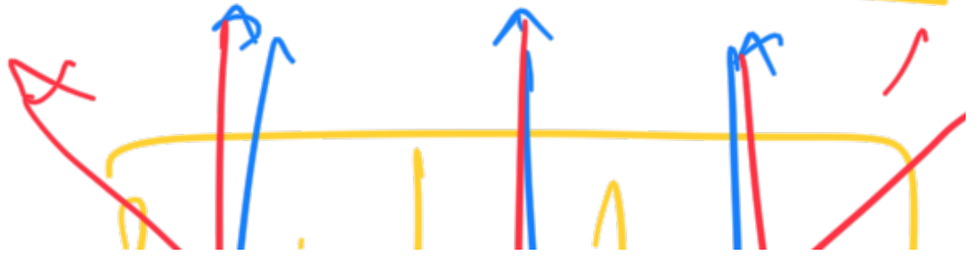
8.) Bot can have diff. difficulty

9.) Game ends when any 1 wins OR Draw

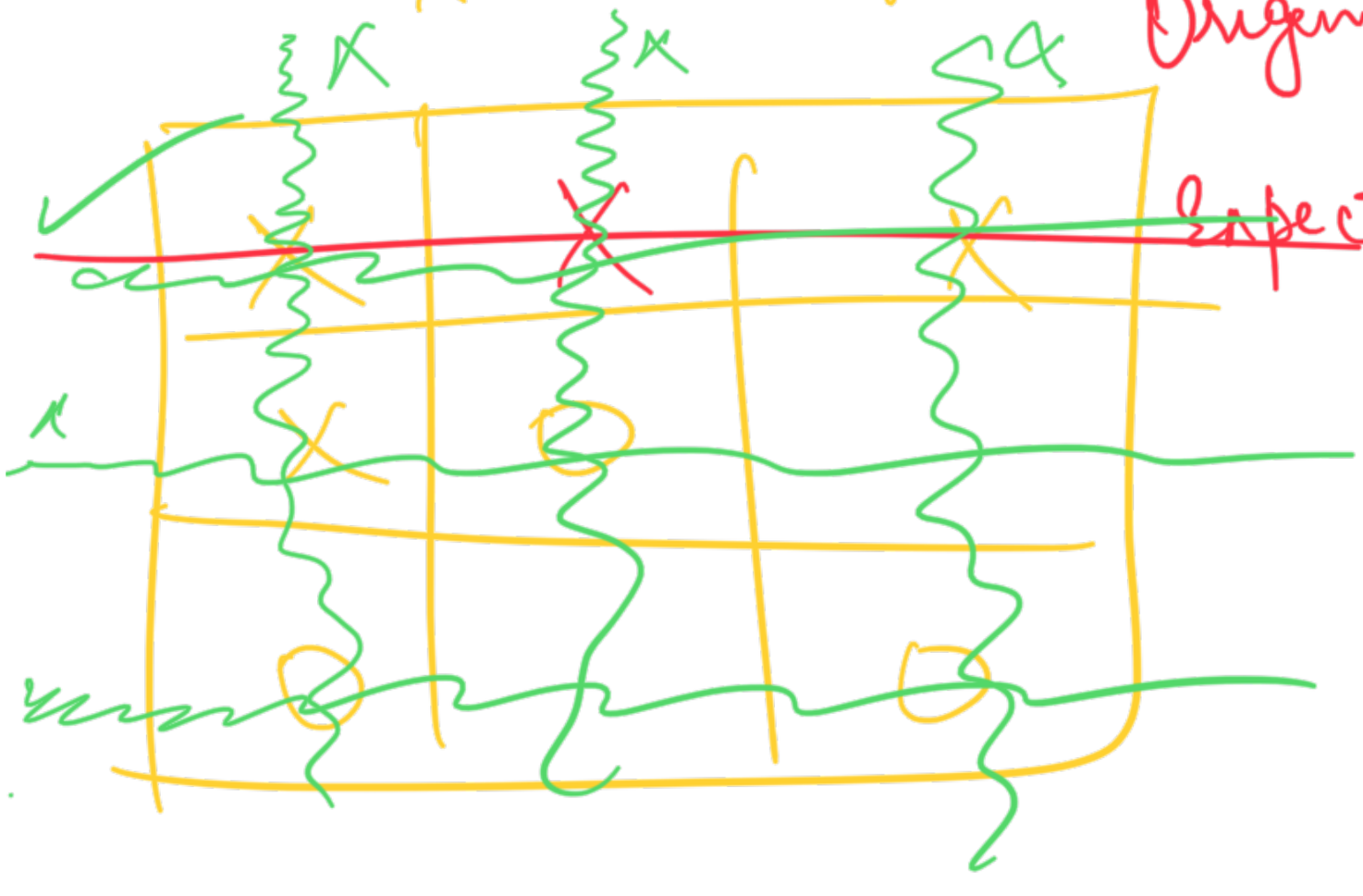
10.) Players make move in a random order } ?

11.) Implement the check that someone has won in O(1)

Tomorrow



X had the first Move

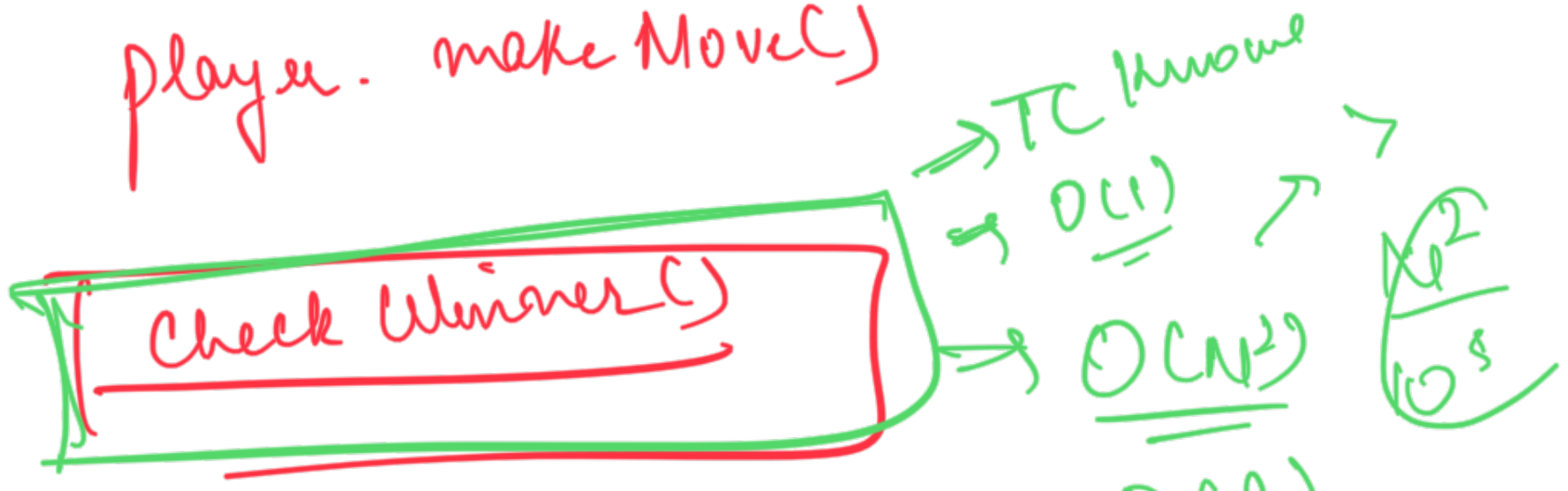


Original ⇒ O(N²)

Expected ⇒ O(1)

makeMove() {

player.makeMove()



}

Class Diagram

2 ways

① Visualization

Game

② Num

n

Player

row

n

X

Symbol

Cell

Board

X

Bot

Classes

But: mutate? validation

Game

Board

List<Player>

List<WinningStrategy>

int lastPlayerMovedIndex; // useful

GameState

Winner: Player

Order of player in list

0, 1, 2, 0, 1,

0 1 2

0 0 0

0

Board

List<List<Cell>>

dimension: int

Cell[][]

GameState

WON,

Cell

Symbol

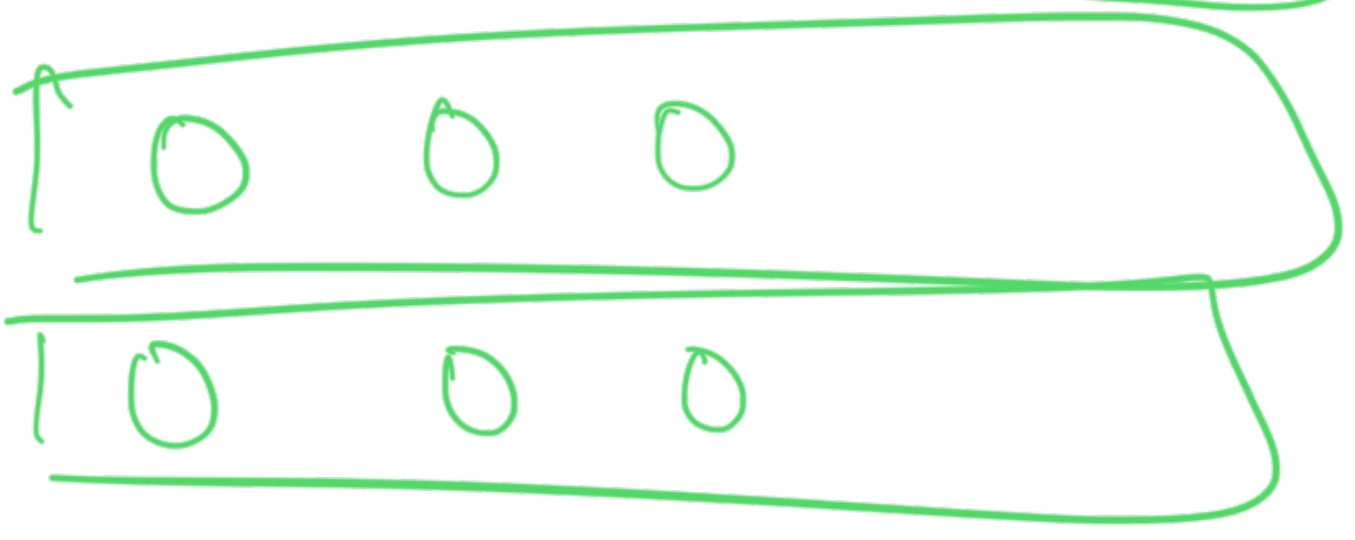
char a;char j;

DRAW()
RUNNII
NOT-START -

row
col

Symbol ::-

int Storage →
String Stud-Name →

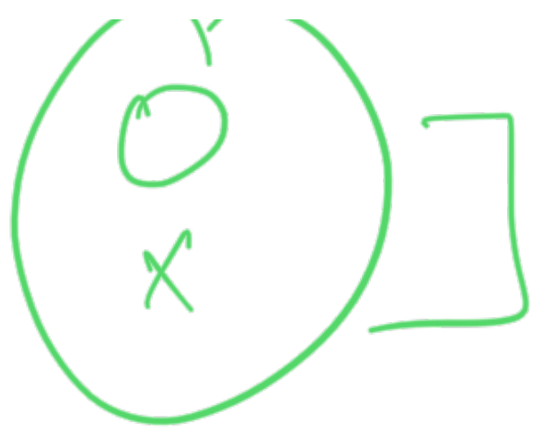


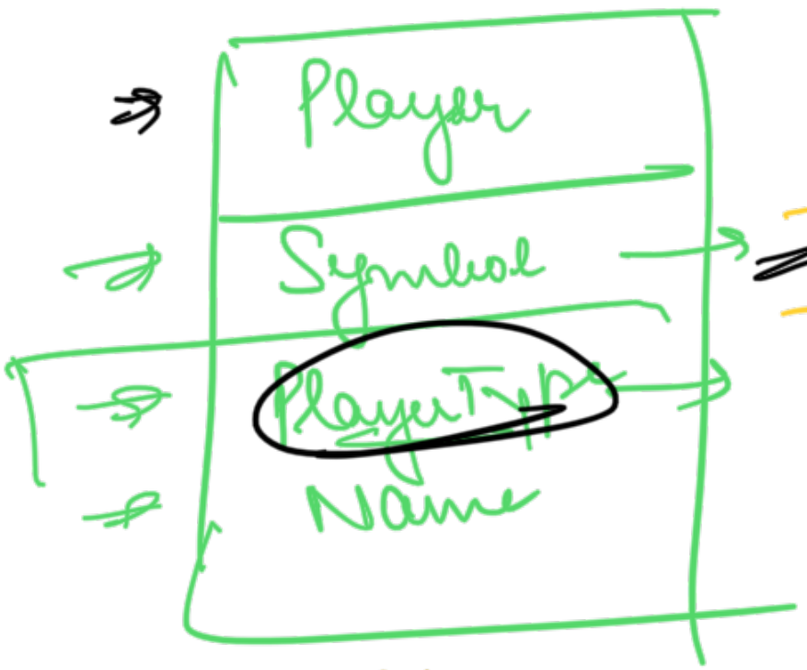
List (Student)

↳ age

↳ name

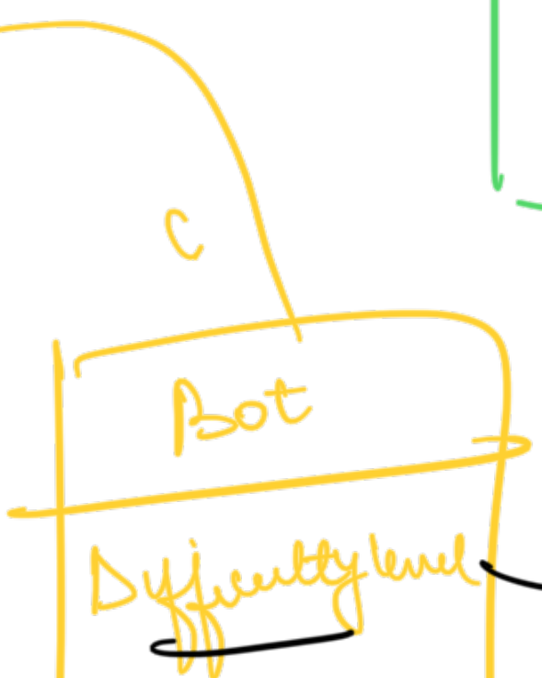
instance of .type





X, O, 1

Symbol a = UNDEFINED)



Level

UNDEFINED,

EASY,

MEDIUM,

HARD

N player

0, 1 Bot

N, N-1 Human Player

M

<< MakeMove (Strategy) >>

void

Cell makeMove(Board, Symbol)

Make Easy Move Strategy

Make Medium Move Strategy

Make Hard Move Strategy

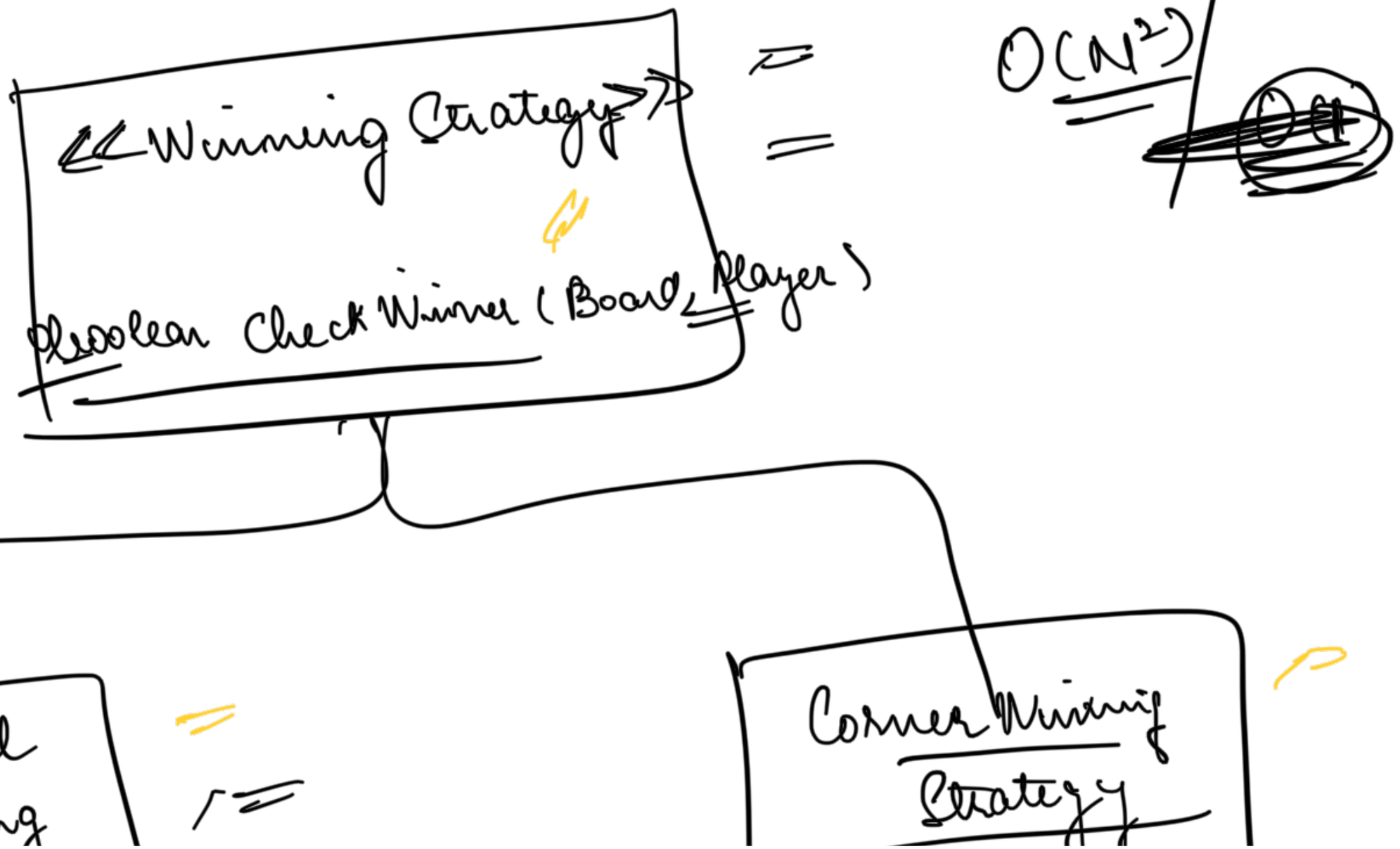
Random

Algo

AI

Make Move Strategy factory

get Make Move Strategy for Difficulty (Difficulty level)


"Winning Strategy"
Boolean CheckWinner (Board, Player)
O(N²)
Normal Winning
Corner Winning Strategy

Strategy

// row

// col

// diag

// Comm

Check Winner() {

⇒ for (WS ws : WStrategies)

if (ws.checkWinner(Board, Player)

return Player has win

}

Out Strategies

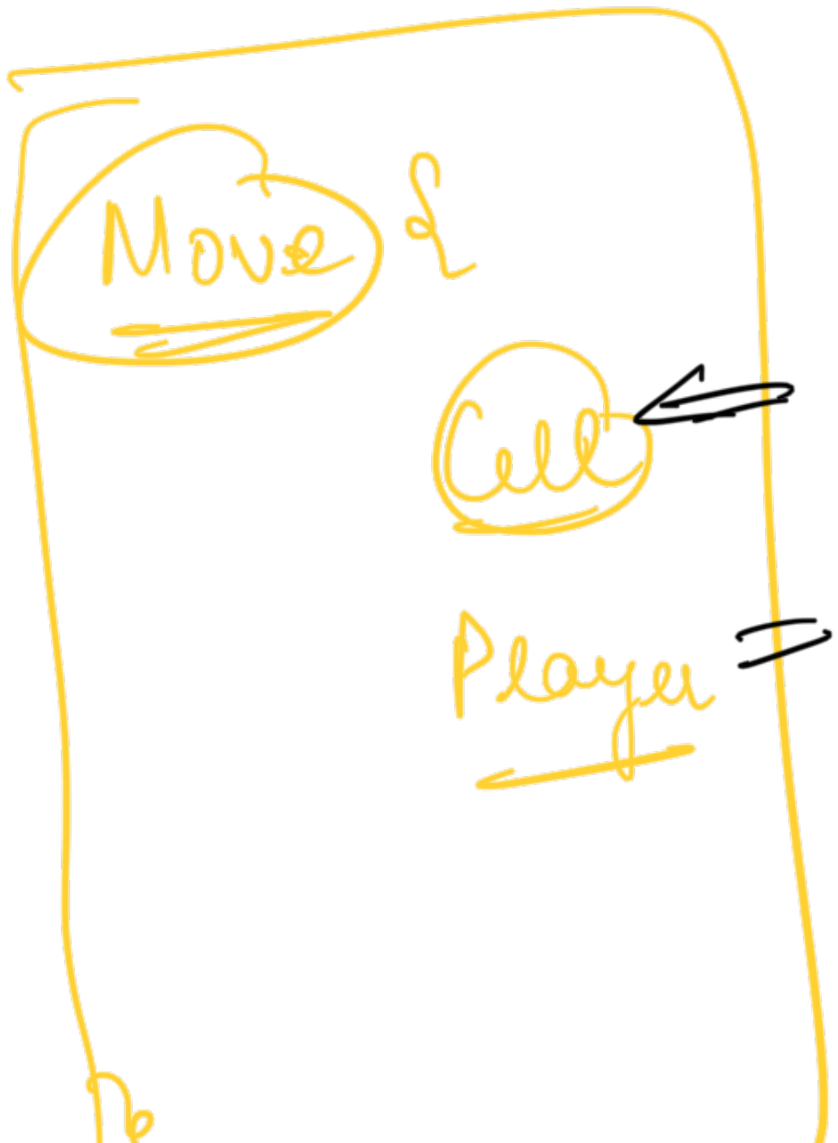
checkOut() {

Gully Cricket

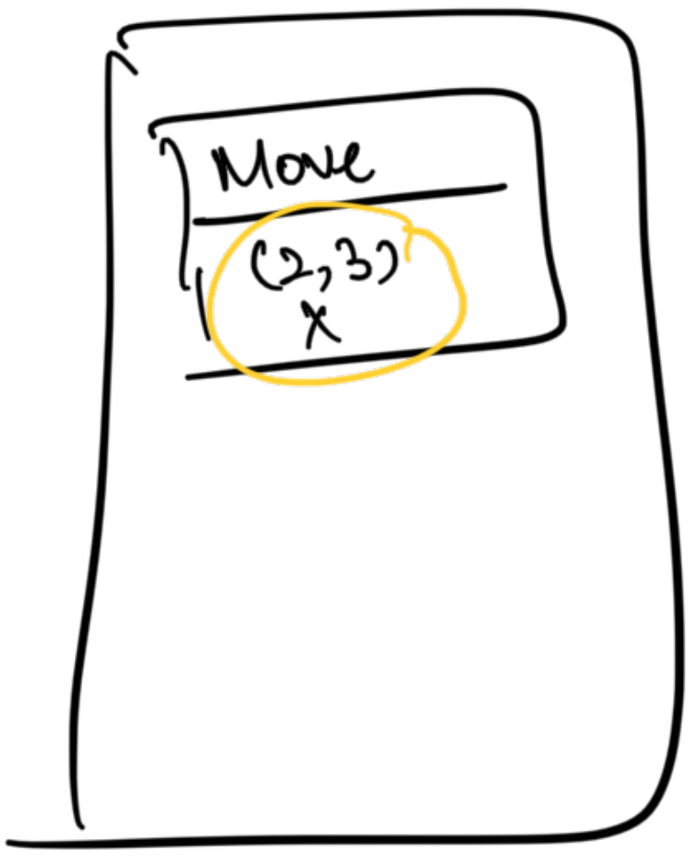
Undo Operation How To Implement

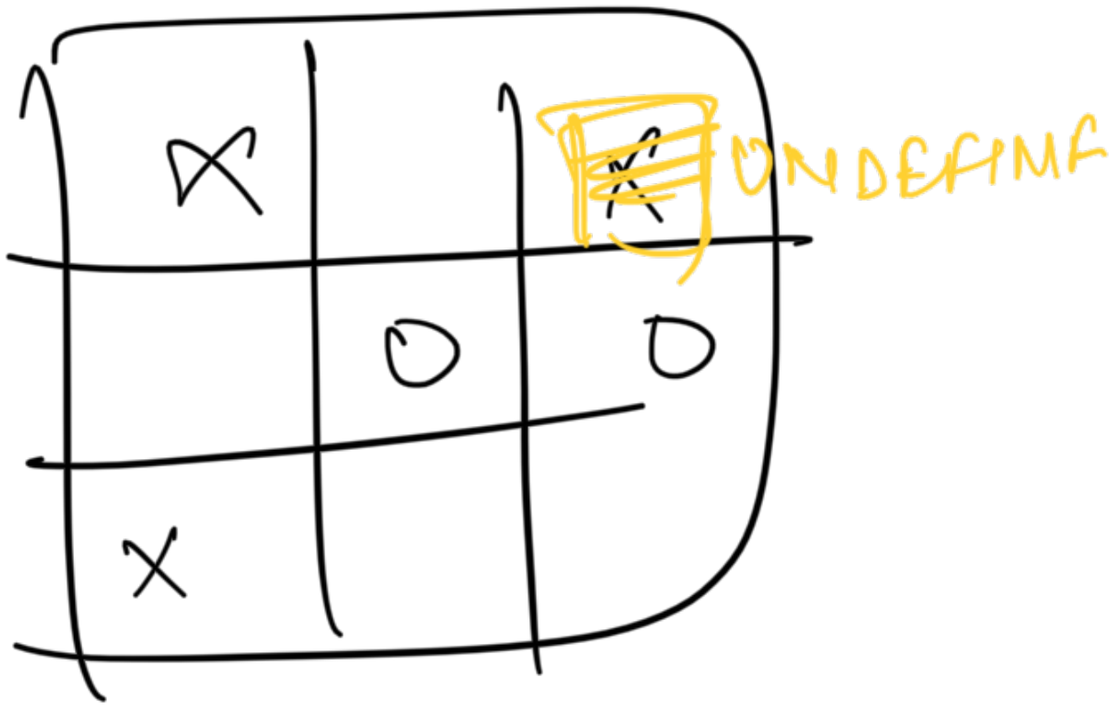
2 ways to implement Undo

Both of these ways need us to store Past Moves



1) Just get the latest Move from list and undo it





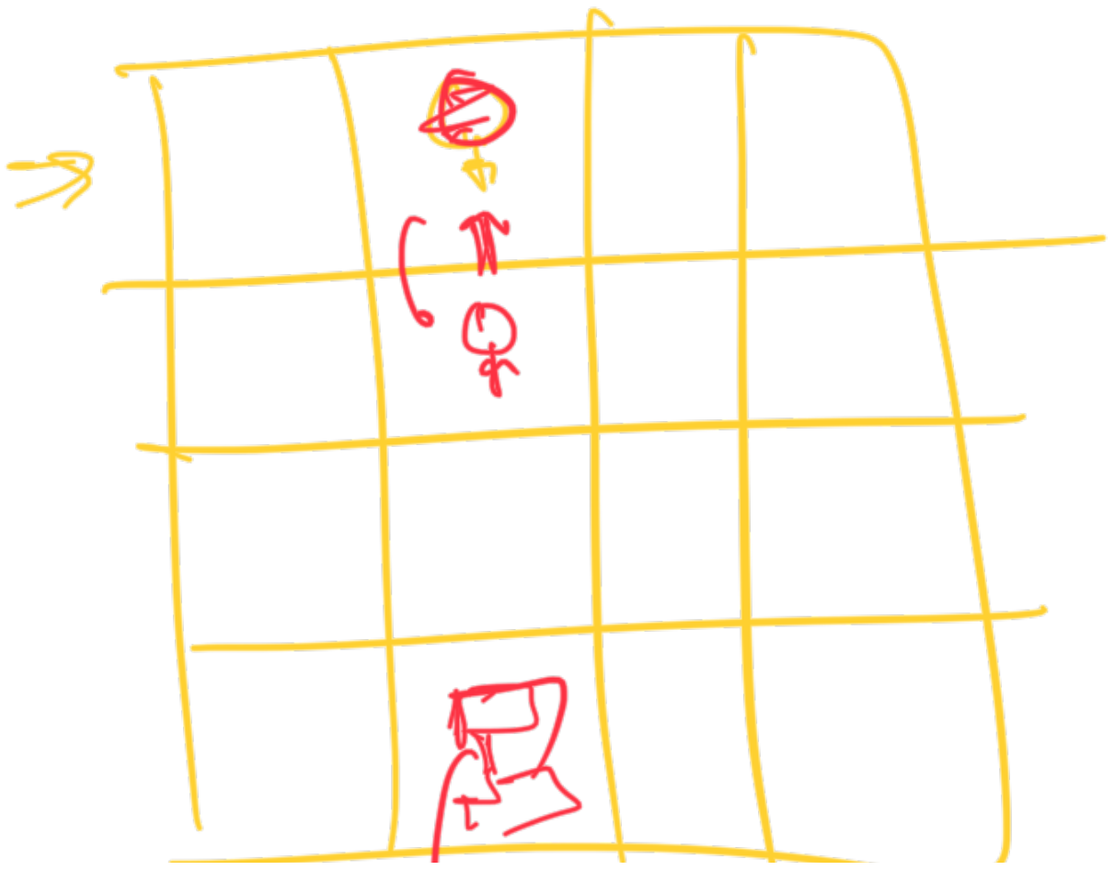
undo() {

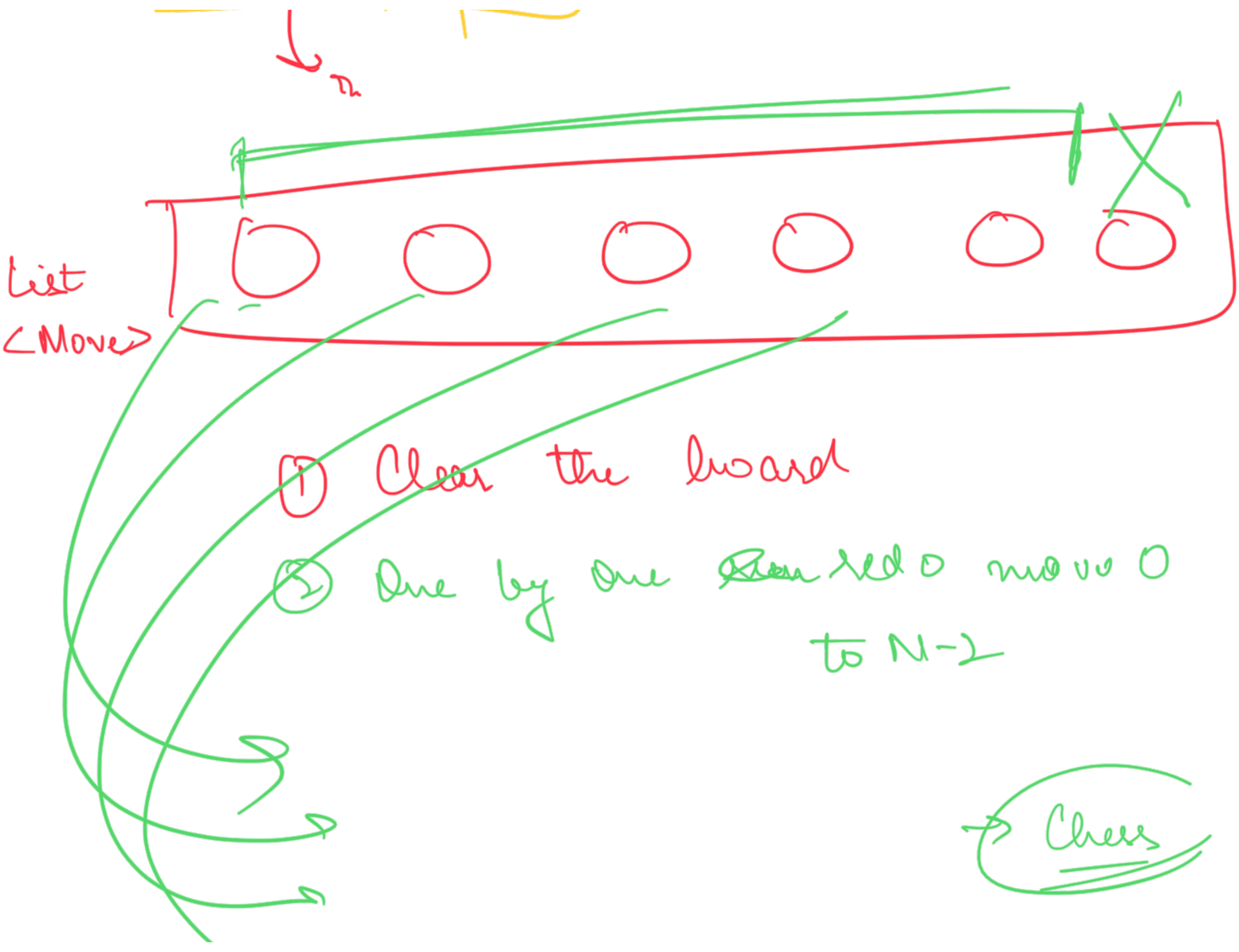
Current Player -= 1

1 Get Coll. Clear

2

(2) Redo all the other Movies.





① Clear the board

② One by one add N moves to N-1

Chess

③ Store Snapshot of state at every move

① Undo

Something went wro

② Redo

Undo	Redo	Snapshot
Less Time	More Time	Less Time
Less Storage	Less Storage	Heavy Storage

Snapshot of the CD on Draw.Do

Send a

HW

1. Implement all the above classes, models

IDE
Java
Project

models /

Board
Cell
Player

Strategies

makeMove Strategy

Winning Strategy

(2) Try to implement the code for NormalWinning Strategy // HasSomeone Won

makeMove(
input 3,3
)

redo

$O(N^2 \times M)$