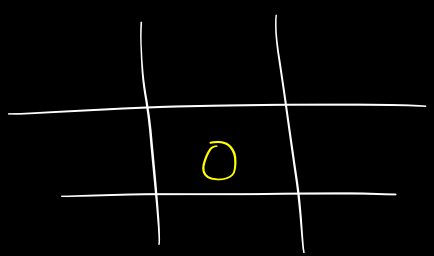
Today's Agenda :-

1) How to implement undo feature
2) Class Diagram of Tic Tac Toe
3) checkWinner() algo.

*Pointers:-

1) Design discussions are subjective

Global undo : → undo | remove the last move that was played on the board.



Move → which player made a turn at which location on the board.

↳ f(player, location)

⇓

(r, c)

undo

P_1 P_2 P_3 P_1 P_2 P_3 . . . .

move will be dependant on (player, x,y)

we are storing moves so if players wants more than 1 undo

m

I^st way

① List < Move > :- [{ P1, 0, 0 }, { P2, 1, 2 }, { P1, 1, 1 }, ...]

x

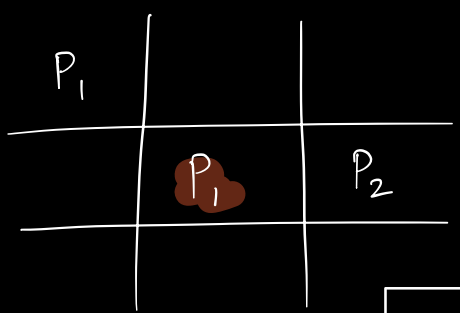
O(1)

Stack

→ Store moves in a list

→ Remove the last move from the list

→ Make that cell empty in the board as well.



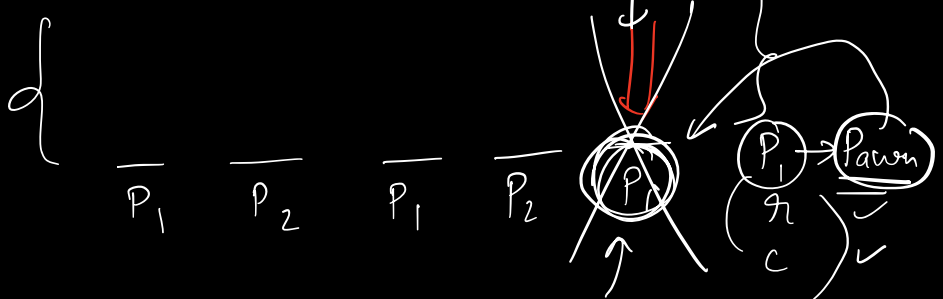
✓

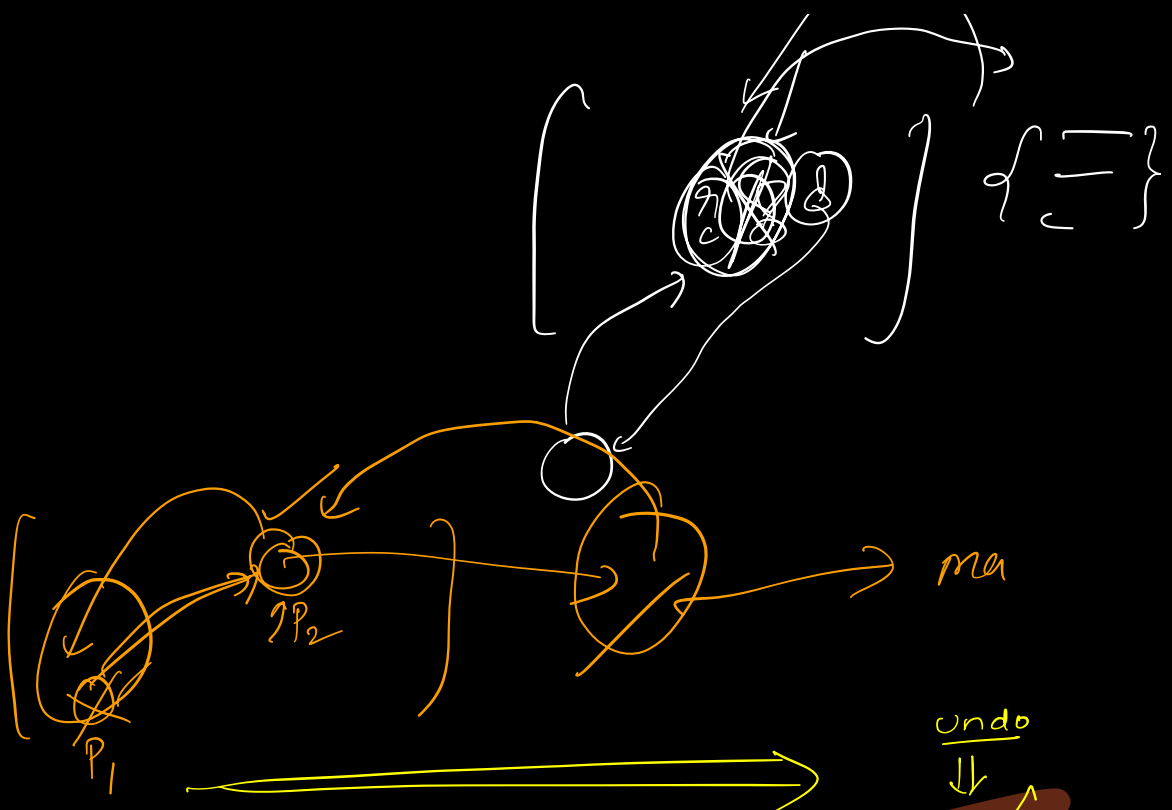
nextPlayer will update to that player.

II Chess

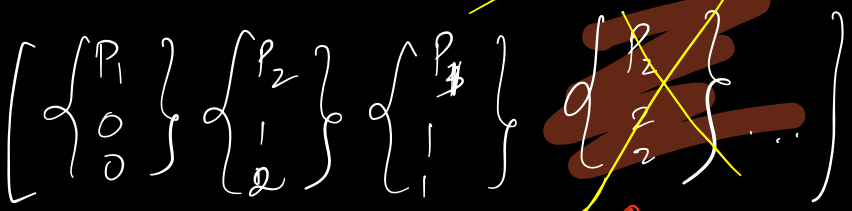
↳ In games like Chess, it's not easy

Chess





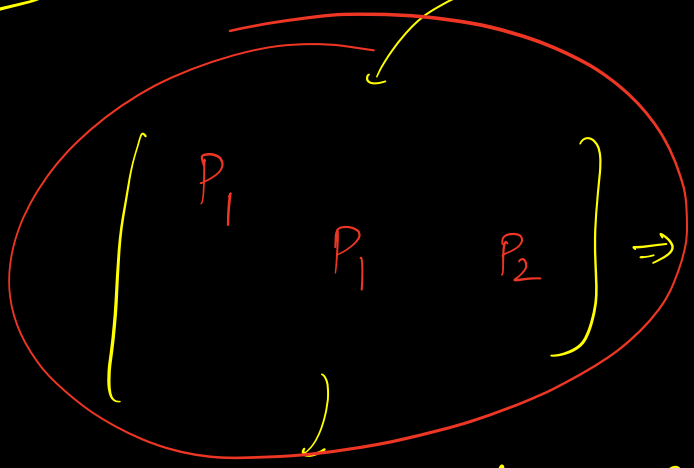
Π hist <More> :



TC-higher

This move was never played ??

initial



fresh board

exactly at our required slot

① Remove last move from list

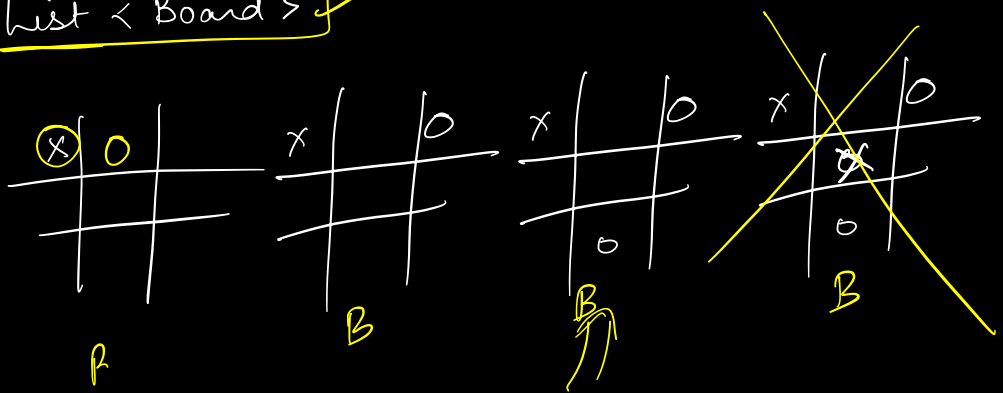
② Clear the board

③ Execute all (N-1) moves on empty board.

III Store the snapshot of the game after every move.

Snapshots

List < Board >



new state, we are done.

Easy to implement

Good TC

Bad SC ⟹ Space is v.vv.cheap.

prefer

Time >> Space

Maintainable code

Bad SC

Good enough TC

V.Opt TC

VS V.Op SC

Complex

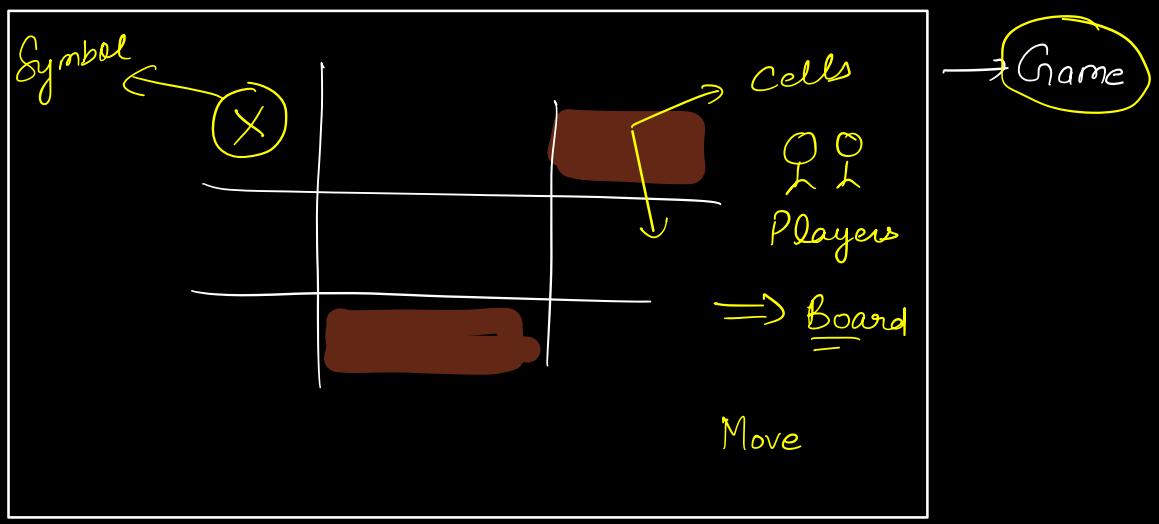
Class Diagram :-

How to identify entities

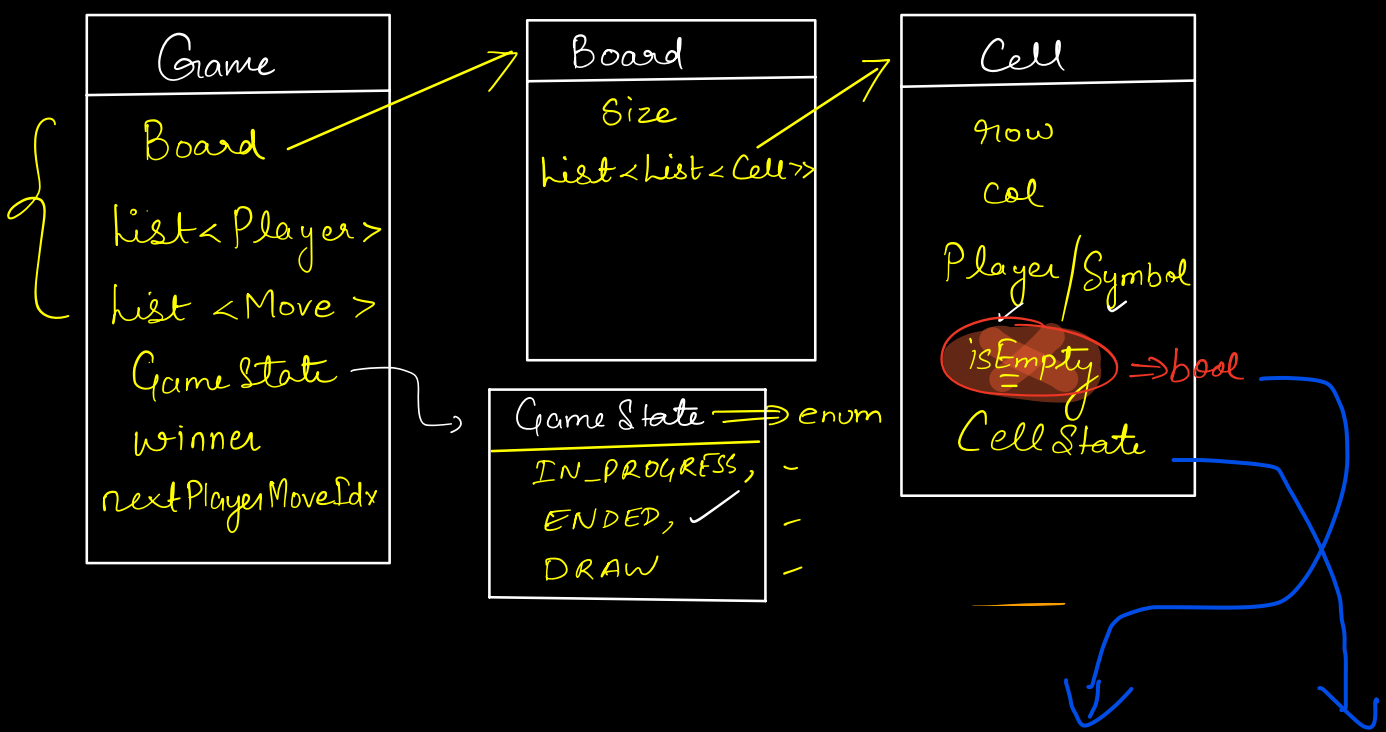
identify Nouns from Req Gath

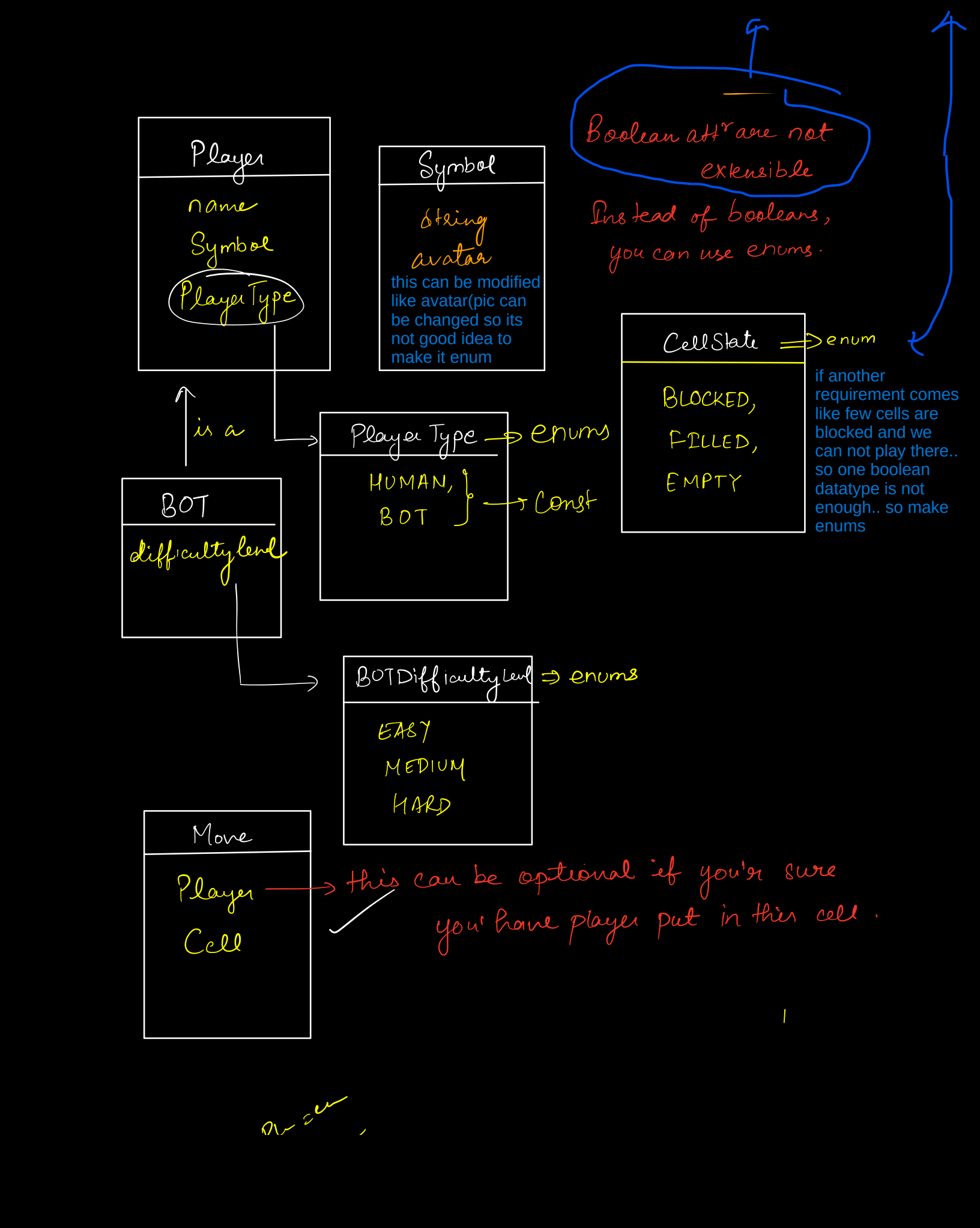
Visualization

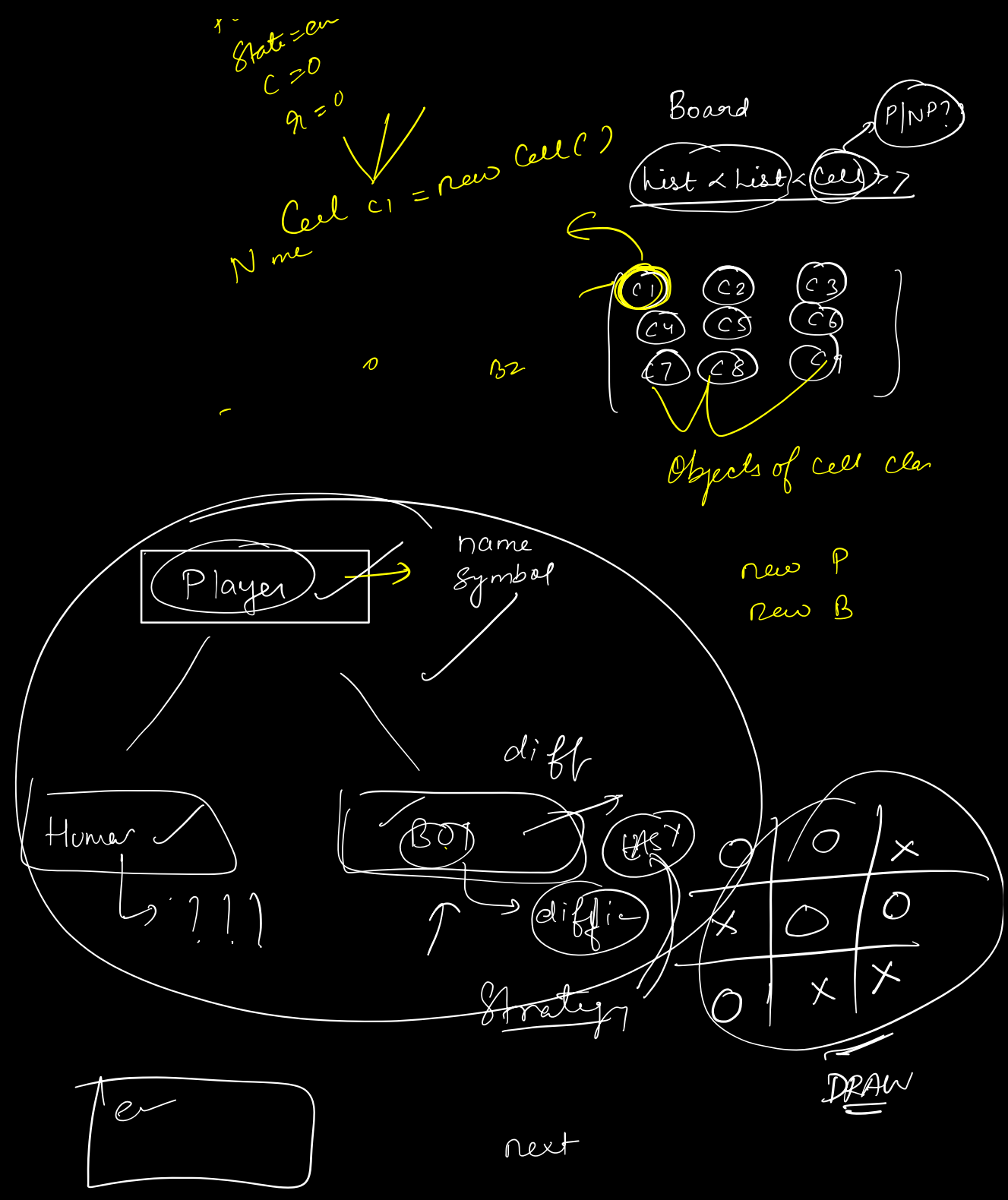
1) Game 2) Board 3) Cells



whenever you feel if any data type is not primitive or non-primitive and new class is required then immediately create another class




Boolean attr are not extensible
Instead of booleans, you can use enums.
Player
name
Symbol
PlayerType
Symbol
String
avatar
this can be modified like avatar(pic can be changed so its not good idea to make it enum
CellState ⇒ enum
BLOCKED,
FILLED,
EMPTY
if another requirement comes like few cells are blocked and we can not play there.. so one boolean datatype is not enough.. so make enums
is a
PlayerType ⇒ enums
HUMAN,
BOT
⇒ Const
BOT
difficultyLevel
BOTDifficultyLevel ⇒ enums
EASY
MEDIUM
HARD
Move
Player
Cell
this can be optional if you're sure you have player put in this cell.


State = en
C = 0
R = 0
Cell c1 = new Cell()
Name
Board
P/NP?
list < List < Cell > >
C1
C2
C3
C4
C5
C6
C7
C8
C9
Objects of cell cla
new P
new B
Player
name
Symbol
diff
Human
BOT
diffic
Strategy
DRAW
next

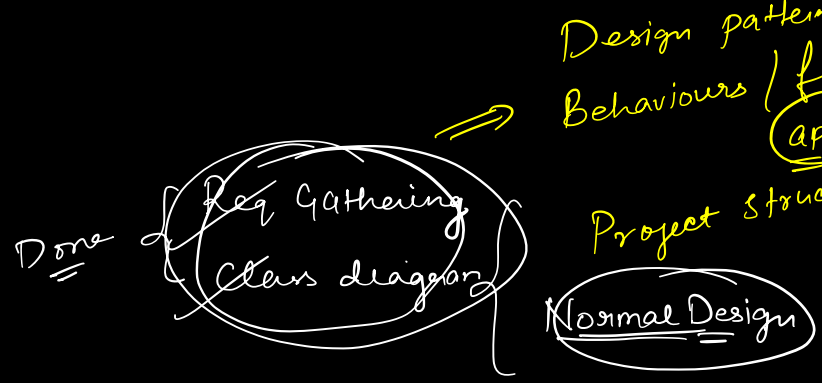
createGame — makeMove()

|

initialize the game

Game Builder ⇒ perform all validations in the game.

Strategy:



Dwp

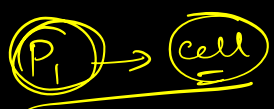
Check if someone won the game ? ⇒ algo

20 min

Assume somebody has the won the game.

Game should ideally stop as soon as somebody wins.

After any move, the game can end. Should ideally check winner ?

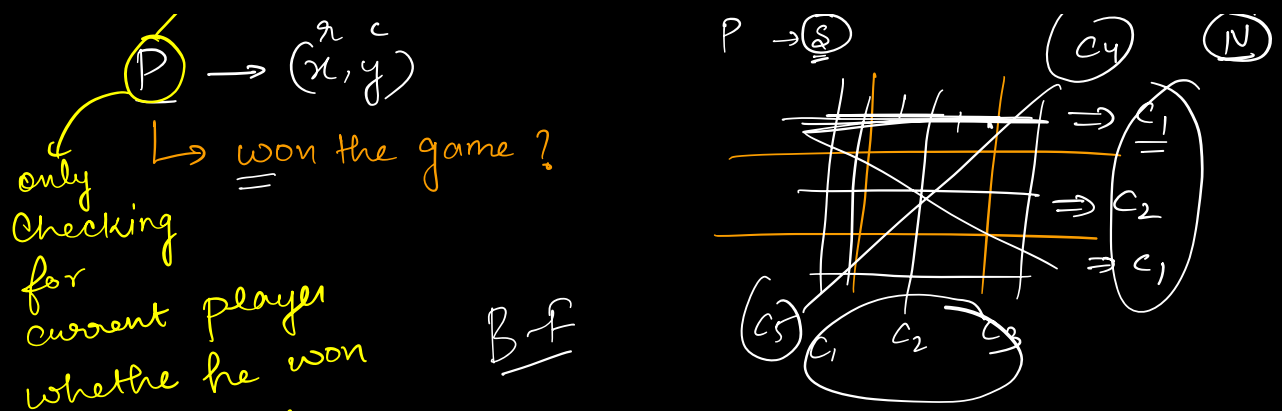


P2

P3

After every move, check if that player has won the game. If yes, Stop the game, else continue

Symbol




```
checkWinner(Board, Player){
    for(i=0; i<N; i++)
        int cnt=0
        for(int j=0; j<N; j++){
            if(board[i][j]==Player.symbol){
                cnt++
            }
        }
        if(cnt==N){
            return true;
        }
    }

    for(int j=0; j<N; j++)
        int cnt=0
        for(int i=0; i<N; i++){
            if(board[i][j]==Player.symbol){
                cnt++
            }
        }
```

Rows N^2

Col^n N^2

```
    }
    if(cnt == N){
        return true ;
    }
}
int cnt = 0
for(int i=0; i<N; i++){
    if(Board[i][i] == Player.symbol){
        cnt++
    }
}
if(cnt == N) return true
```

L-R Diag

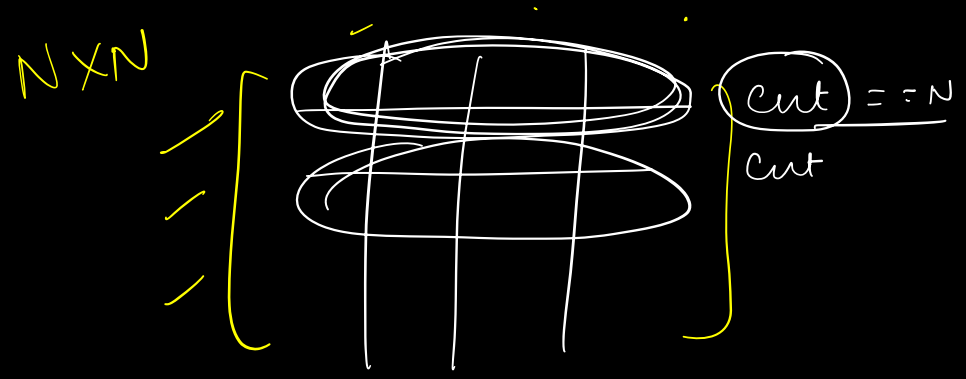
N

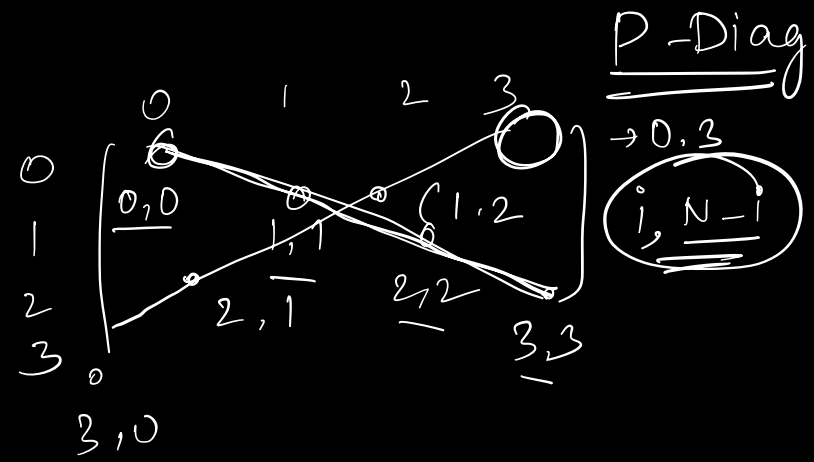
```
int cnt = 0
for(int i=0; i<N; i++){
    if(Board[i][N-i] == Player.symbol){
        cnt++
    }
}
if(cnt == N) return true

return false
}
```

R-L Diag

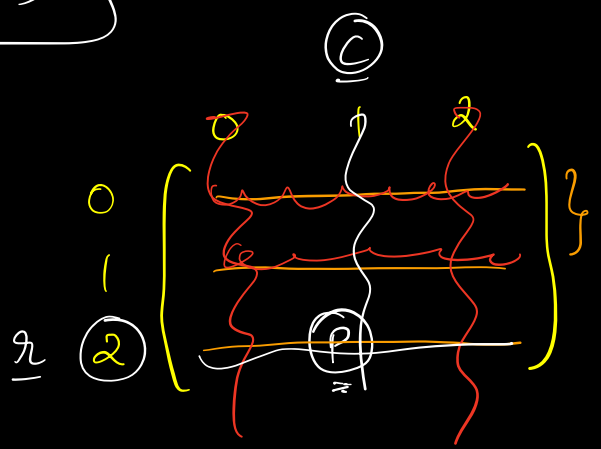
N


N X N
Cut
= = N
Cut


P - Diag
0 1 2 3
0
1
2
3
0,0
1,1
2,1
2,2
3,3
3,0
1,2
→ 0,3
i, N-1

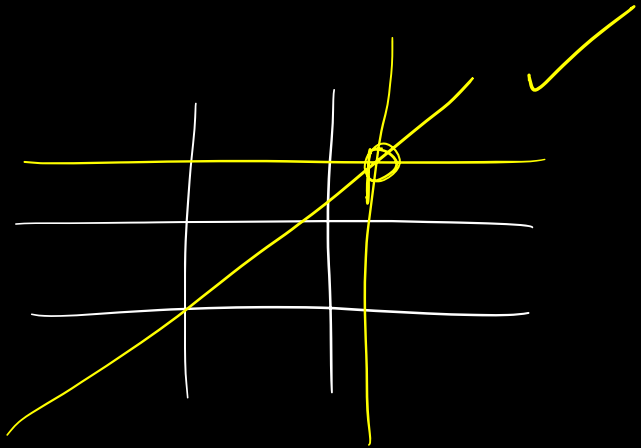
$N^2 + N^2 + N + N$

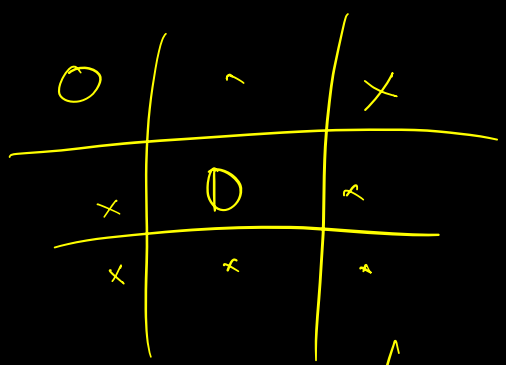
$O(N^2)$

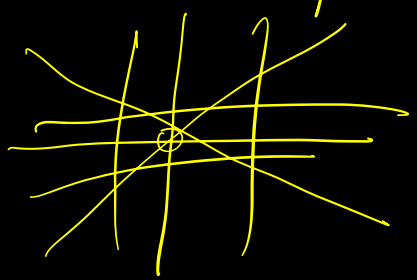

C
0 1 2
0
1
2
P

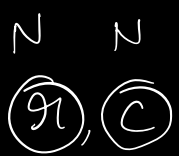
Obs

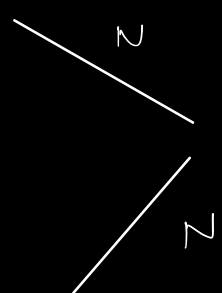
If Player is making a move on (r, c) index then winning can only happen either on r^{th} row or c^{th} col or diag if that (r, c) is part of that diagonal.











```
int cnt = 0
for(int j = 0; j<N; j++){
    if(board[x][j] == Player.symbol){
        cnt++
    }
}
if(cnt == N){
    return true;
}
```

N

Check x^{th} row (counting no of player's symbol in x^{th} row)

```
int cnt = 0
for(int i = 0; i<N; i++){
    if(board[i][c] == Player.symbol){
        cnt++
    }
}
if(cnt == N){
    return true;
}
```

N

Check c^{th} col

```
int cnt = 0
for(int i = 0; i<N; i++){
    if(board[i][i] == Player.symbol
        cnt++
    }
}
```

N

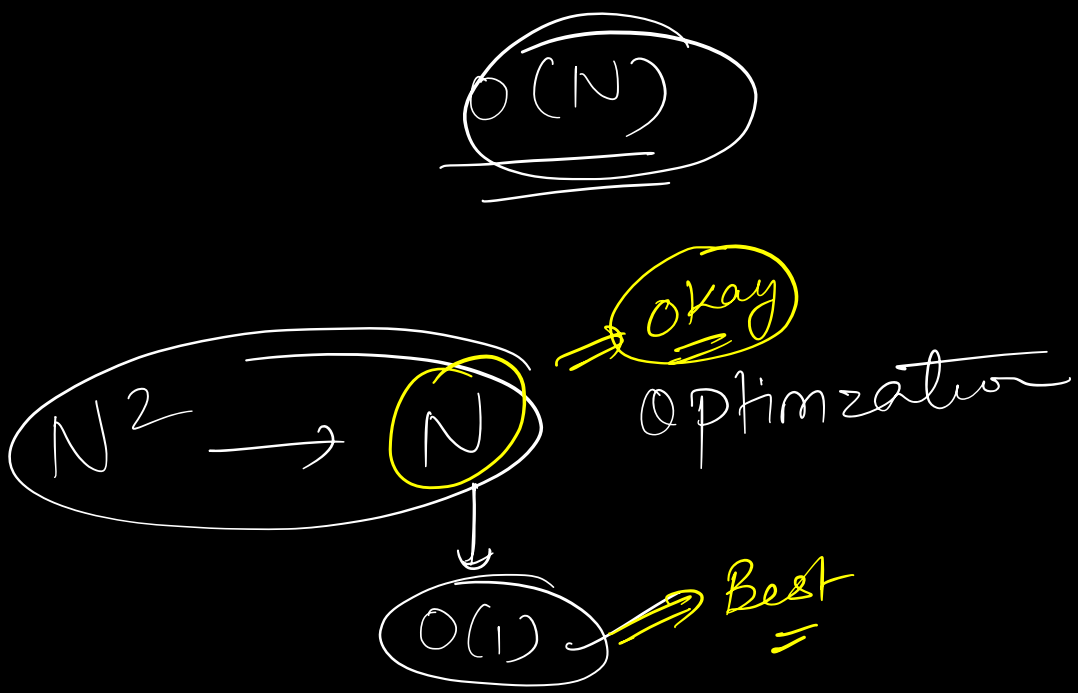
```
if (cnt == N) return true

int cnt = 0
for(int i=0; i<N; i++) {
    if (Board[i][N-i] == Player
        cnt++
    }
}
if (cnt == N) return true

return false
```

N

Total TC - N * 4



P (r, c)

At the end of the day,

aren't we counting no of Player's symbol in rth row & cth col & P & S diag

↓↓

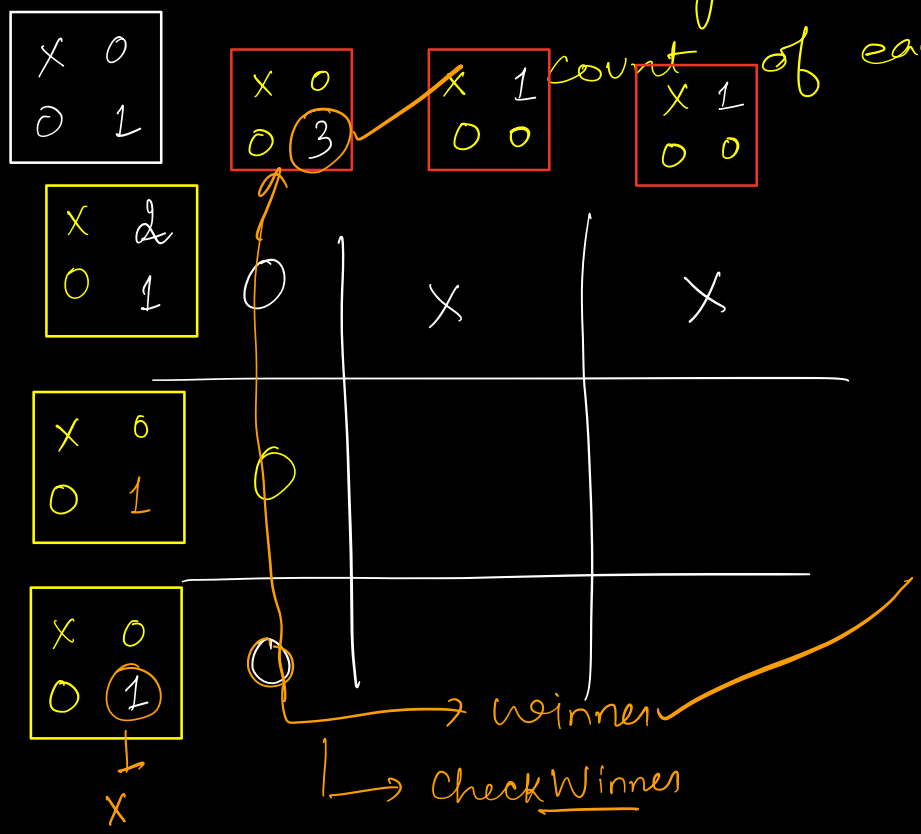
What if we maintain this info on the fly!?

Hashmap ⇐

- P, (r) ⇒ cnt
- P, (c) ⇒ cnt
- P, (L-R) ⇒ cnt
- P, (R-L) ⇒ cnt

O(1)

III Maintain hashmaps for each row, each column & each diagonal to store count of each symbol.



extra space to optimize Check win

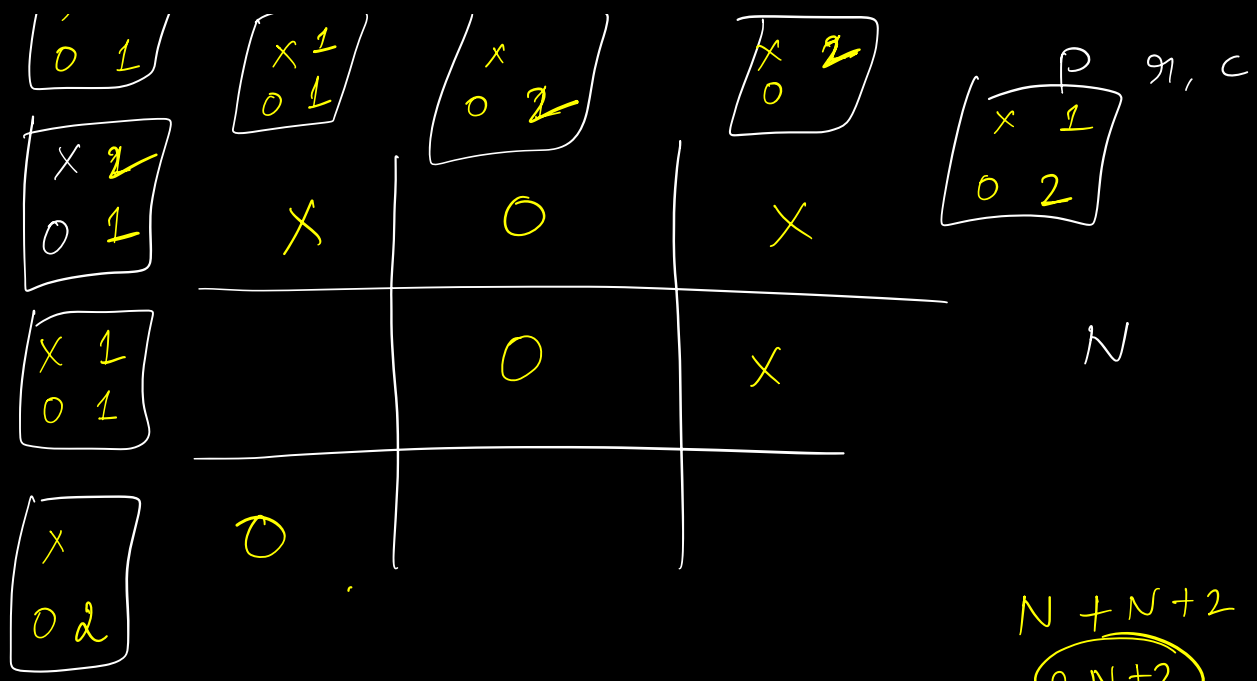
For each row, column, diagonals separate hashmap those have key as players(X,O) and value will be count of the cells in which that player symbol is present.

SC → N^2

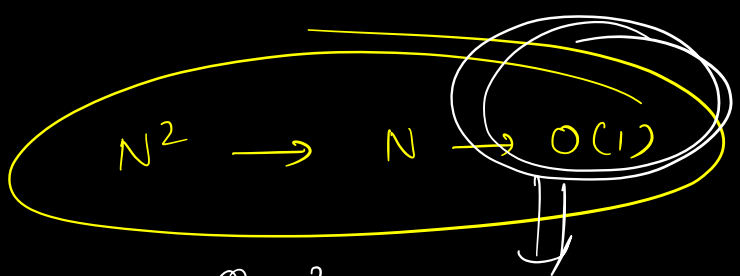
N, P, C ⇒ # row, # col, # 2

$N * (N + N + 2)$

X 1



O(1)

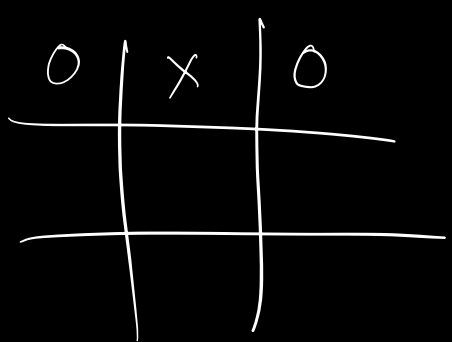


0 2

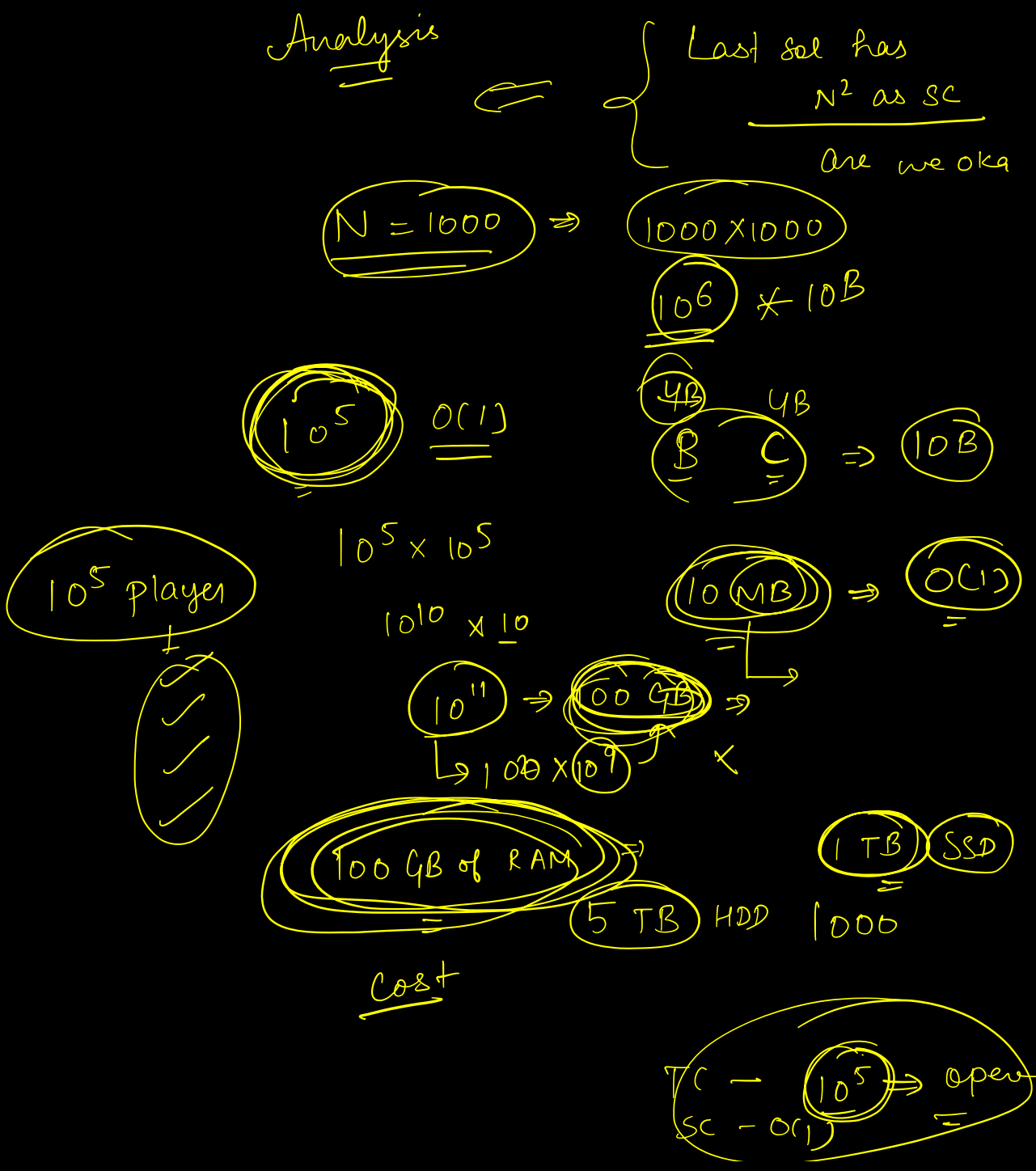
X

0, 1

Implement this in our code



Impl will start from next class



10^5

Board

$10^5 \times 10^5$

Good idea