Agenda :-

→ How to approach schema design

→ How to code

→ Design of TTT

How to approach schema design :-

- 1 : 1 ⟶ put the id of one side to other side
- 1 : M / M : 1 ⟶ put id of 1 side to other side, in many
- M : M → mapping table

Step 1 ⟶ For all the classes, create a table

Step 2 ⟶ For each class, put its primitive attributes as columns in the table

↓

(int / double / boolean / string)

```
class Ticket {
    int id;
    Date dayOfBooking;
    List<Seat> seats;
    User bookedBy;
    int amount;
}
```

relationships

Ticket

id	Date	amount	userId

Step3 → For every relationship :

a) Find cardinality of rel^n

b) Based on cardinality, represent the rel^n.

⇒ How to code :- ⇒ Parsing US-

i) Project structure ⇒ Create your codebase structure (rough)

ii) Atleast some features should be running

- a) Write all the models
  ↓
  [classes from class diagram]
- b) Do requirement by req.

Code one feature/req/use case at a time.

- controller → (0)
- Service (6) → Business Logic
- repository / DAO (8)
- models (10)

{ MVC → model view controller

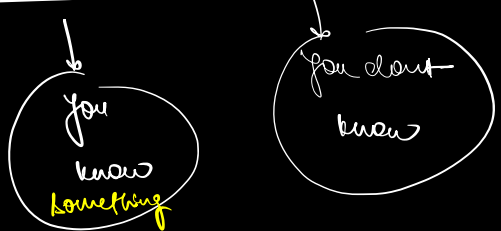
↓ DB ↓ UI ↓ Backend

- model ⟶ entities [classes]
- repository / dao ⟶ classes that talk to DB
- Service ⟶ does business logic
- Controller ⟶ accepts the request, validates & returns response

Call response

Design Tic Tac Toe

- Overview



we know about Tic Tac Toe

1) align
2) type of system
   - interactive → no DB, only features
   - persistent → features & DB

⇒ Gathering Requirement :- [Interactive]

- Size of the board n x n
- No. of players ⇒ (n-1), each player has their own symbols, minm 2 players (assuming bot is a player)
- Every player can choose, their own symbols, as any char at start of the game, no two people can have same character.
- At beginning, we will randomise the player list and move in that order.

⇒ player list ⇒ | A B C D | —random→ | C B A D |

- If any player achieves same marking, across a row, column or diagonal, player wins
- Draw, if matrix | board is full, and no winning player
- Will the game have bots ⇒ Yes
  ⇒ Only 1 bot per game
- Will bots have difficulty level ⇒ Yes
  EASY, MEDIUM, HARD.

- Timer between moves → ✗

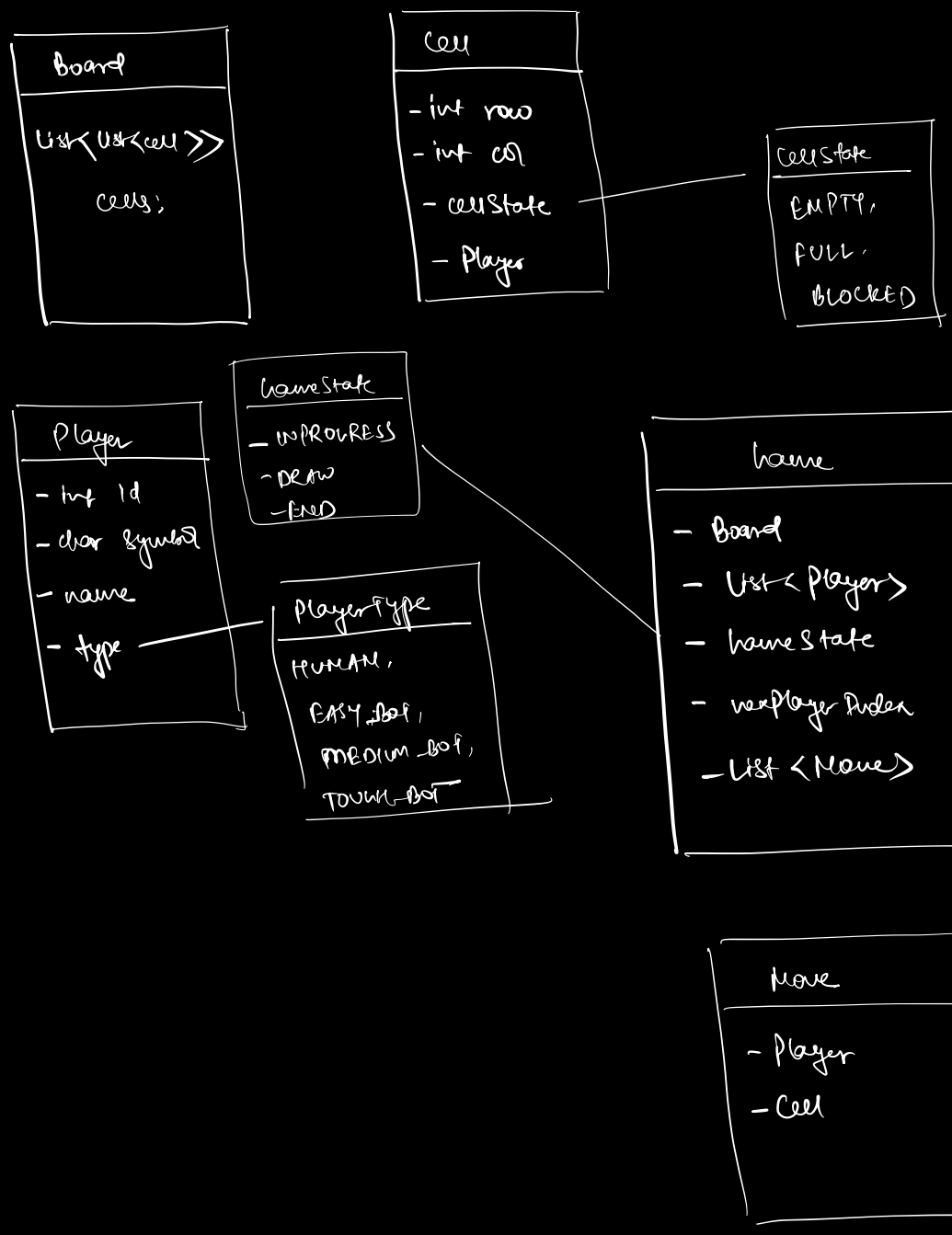
- Game ends when somebody won, or game is drawn.

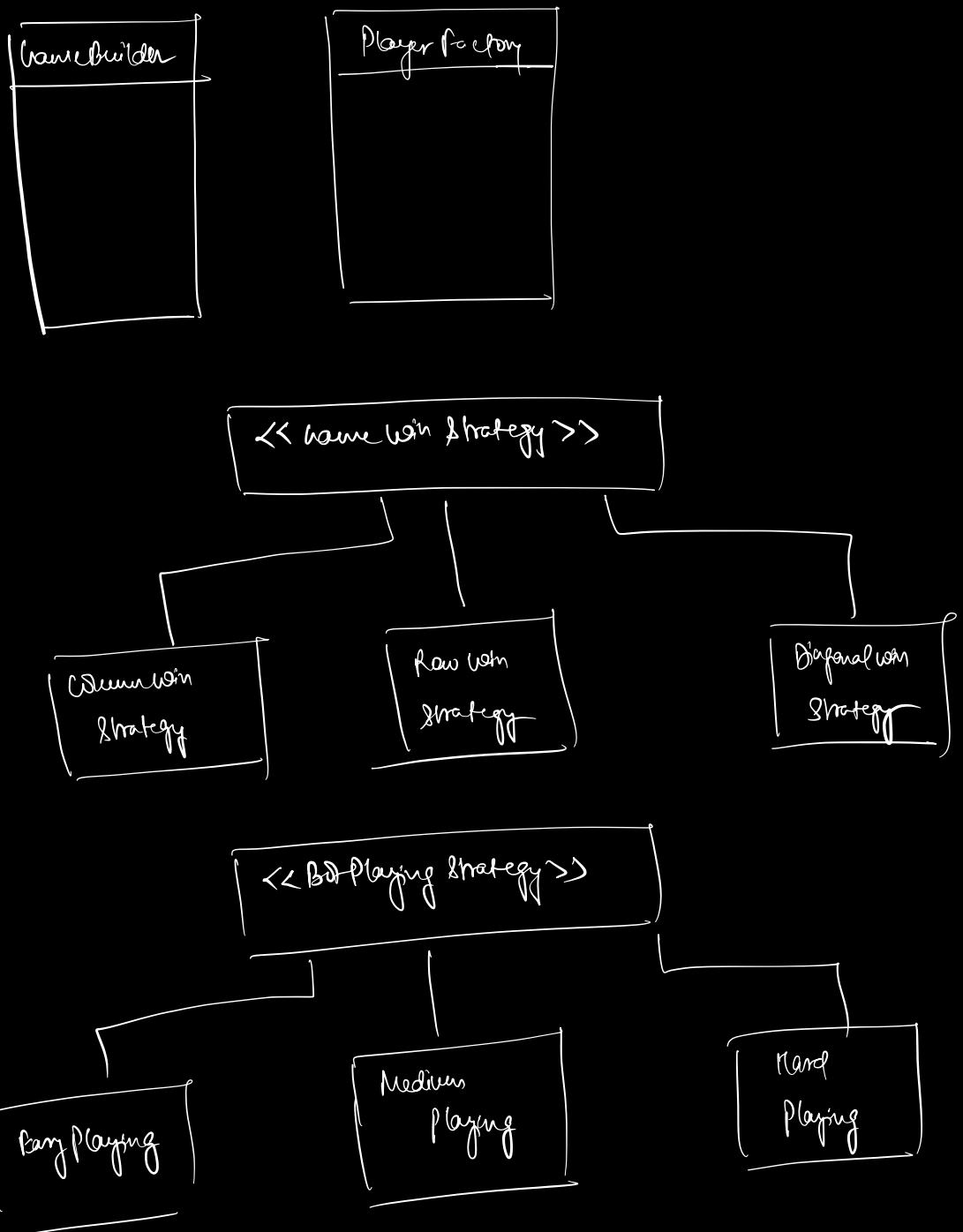
- Tournament
- Pause | resume | exit | forfeit ⇒ ✗
- Leaderboard

- Allow people to undo their moves
- Blocked cells
- Rematch of the game should be available.

⇒ Any game :-

→ Size / design of board
→ no. of players
→ Bots, with difficulty levels
→ Leaderboard
→ Undo
→ How many undoes are possible.
→ How game starts
→ How game ends
→ Rematch after a game ends
→ How will each player's turn will come up
→ winning, draw or lose cond


Board
List<List<Cell>>
cells;
Cell
- int row
- int col
- cellState
- Player
CellState
EMPTY,
FULL,
BLOCKED
Player
- int id
- char symbol
- name
- type
GameState
- INPROGRESS
- DRAW
- END
PlayerType
HUMAN,
EASY_BOT,
MEDIUM_BOT,
TOUGH_BOT
Game
- Board
- List<Player>
- gameState
- nextPlayerIndex
- List<Move>
Move
- Player
- Cell




```
BotPlayingStrategyFactory {

    getStrategyForBot ( Enum ____ ) {

    }

}
```

HW

How to implement UNDO in game?