Class Diagram for the Chess Game Learn to create a class diagram for the chess game using the bottom-up approach. We'll cover the following Components of Chess Box Chessboard

 Piece Move Account, Player, and Admin Chess move controller Chess game view Chess game • Enumerations and custom data types

We'll create the class diagram for the chess game. In the class diagram, we will first

design the classes and then identify the relationship between classes according to the

Box

Chessboard

+ getPieces(): Piece {list} + resetBoard(): void + updateBoard(): void

The class diagram of the Chessboard class

<<Abstract>>

killed: bool white: bool

+ isWhite(): bool + isKilled(): bool canMove(): bool

extends

The class diagram of the Piece and its derived classes

்டு R4: Chess Game

Move

+ isCastlingMove(): bool

The class diagram of Move class

்டு. R5 and R6: Chess Game

Account

- id: int

Player

person: Person - totalGamesPlayed: int

whiteSide: bool

+ isWhiteSide(): bool + isChecked(): bool

password: string - status: AccountStatus

extends

+ resetPassword(): bool

The class diagram of Account and its derived classes

∵穴 R1: Chess Game

R1: The purpose of this system is to enable multiplayer in a game of chess via an

ChessMoveController

The class diagram of the ChessMoveController class

்டு: Chess Game

ChessGameView

The class diagram of the ChessGameView class

ChessGame

The ChessGame class

GameStatus: This enumeration keeps track of the active status of the player and

AccountStatus: We need to create an enumeration to keep track of the account

Active Closed

None

Enums in the chess game

Person: This is used to store information related to a person like a name, street

Person

The class diagram of the Person class

Now, we'll discuss the relationships between the classes we have defined above for

The Player class has a one-way association with ChessMoveController.

The ChessGame class has a one-way association with ChessGameView.

The ChessGameView class has a one-way association with Player.

The ChessMoveController class has a one-way association with ChessGame.

Player

ChessGameView

The association relationship between classes

The aggregation relationship between classes

Box

Move

The composition relationship between classes

The King, Queen, Knight, Bishop, Rook, and Pawn classes extend the Piece class.

Note: We have already discussed the inheritance relationship between classes

ChessBoard

+ getPieces(): Piece {list}

ChessGame

movesPlayed: Move {list}

boxes: Box {matrix}

creationDate: date

+ resetBoard(): void

+ updateBoard(): void

players: Player {list}

- board: Chessboard

currentTurn: Player

- status GameStatus

+ playerMove(): bool

makeMove(): bool

Player

Admin

+ isOver(): bool

person: Person totalGamesPlayed: int

whiteSide: bool

+ isWhiteSide(): bool + isChecked(): bool

+ blockUser(): bool

manipulates

uses

extends

The class diagram of the chess game

of the chessboard at a given moment due to the shared nature of the chessboard

Command design pattern: This pattern is used to encapsulate the move logic

command, which allows it to move according to the rules defined for it. For

example, the knight moves in an L-shape pattern, or the rook can move only

allowing the pieces to behave in a uniform manner where the user does not need

to know the specifications or underlying logic behind the moves of the pieces.

piece, since all the chess pieces have their own respective implementations of

the chessboard is the subject. As soon as the state of the board changes, the

checkmate states which makes them behave differently from each other

horizontally or vertically on any number of boxes.

depending on the situation.

for each chess piece. Each chess piece has its own implementation of the move

as a resource. Multiple instances can cause the game state to become

piece: Piece

x: int

y: int

ChessMoveController

validateMove(): bool

Account

resetPassword(): bool

id: int password: string status: AccountStatus

Piece

killed: bool

white: bool

+ isWhite(): bool

+ isKilled(): bool + canMove(): bool Move

startBox: Box

endBox: Box

· player: Player

pieceKilled: Piece

pieceMoved: Piece

castlingMove: bool

ChessGameView

+ playMove(): void

sees

updates

+ isCastlingMove(): bool

The following classes demonstrate an inheritance relationship:

Both, Admin and Player extend the Account class.

in the component section above one by one.

ChessGame

ChessMoveController

ChessGame

ChessBoard

ChessGame

- name : string

- city : string - state : string zipcode : int country : string

Relationship between the classes

The class diagram has the following association relationships:

The Move class has a one-way association with Player.

The class diagram has the following aggregation relationships:

Player

• The ChessGame class contains the Player.

The Box class is composed of Piece.

Piece

Inheritance

The ChessBoard class is composed of Box.

The ChessGame class is composed of Move.

The ChessGame class is composed of ChessBoard.

- streetAddress : string

Canceled

Blacklisted

<<enumeration>>

AccountStatus

the game, i.e., who wins and whether or not the game is a draw.

status, whether it is active, canceled, closed, blocked, or none.

<<enumeration>>

GameStatus

Active

BlackWin

WhiteWin

Forfeit Stalemate Resignation

address, country, etc.

our chess game design.

Move

Aggregation

Association

- players: Player {list} - board: Chessboard - currentTurn: Plaver - status: GameStatus - movesPlayed: Move {list}

+ isOver(): bool + playerMove(): bool + makeMove(): bool

+ playMove(): void

+ validateMove(): bool

Admin

+ blockUser(): bool

- startBox: Box - endBox: Box pieceKilled: Piece pieceMoved: Piece - player: Player castlingMove: bool

+ canMove(): bool

King

castlingDone: bool

+ canMove(): bool

Bishop

canMove(): bool

Knight

+ canMove(): bool

boxes: Box {matrix} - creationDate: date

- piece: Piece - x: int y: int

 Relationship between the classes Association Aggregation Composition Inheritance • Class diagram for the Chess game

 Design pattern Al-powered trainer

requirements for the chess game problem. **Components of Chess** As mentioned earlier, we'll follow the bottom-up approach to designing a class diagram for the chess game.

Box A Box is a specific position/block on the 8x8 chessboard which is defined by row x and column y, respectively.

The class diagram of the Box class

Chessboard

A Chessboard is the 8x8 board that stores all the current game's active pieces. It is identified by the date of its creation and can be updated or reset.

Piece A chess Piece can only be black or white in color. It might be alive or killed depending

on the moves made by the opposition. There can be six types of pieces (Rook, Pawn, King, Queen, Knight, and Bishop derived from Piece) that have their respective moves based on the rules of the game which decides whether they are eligible to

move or not.

Pawn

canMove(): bool

+ canMove(): bool R4: At the start of the game, each player will have eight pawns, two rooks, two bishops, two knights, one queen, and one king on the board.

Move A Move is the displacement of a Piece from one Box to another on the chessboard. It may or may not kill a piece of the opposing player.

R5: The player with the white pieces will make the first move. **R6:** It is not possible for a player to retract or undo their move once it has been made. Account, Player, and Admin The Account class is a parent class that has two types: Player and Admin. These

classes are derived from the Account class. This class stores the user ID, password, and account status. Player: This derived class represents the players of the game and all records of the games played. It also keeps track of whether or not the player's chosen color is white. Admin: This derived class decides whether or not a user account is blocked.

online platform.

Chess move controller The ChessMoveController class validates the moves made by a player and responds accordingly.

chess game.

R2: The game will be played according to the official rules of an international Chess game view The ChessGameView class represents the game view. The ChessGame class updates the ChessGameView class. Chess game The ChessGame represents the gameplay of chess. It keeps track of the moves played by both the players, the turns, and the game status.

Enumerations and custom data types The enumerations and custom data types required in the chess game design problem are listed below:

Composition The class diagram has the following composition relationships.

Class diagram for the Chess game Here's the complete class diagram for our chess game:

Bishop

canMove(): bool

Knight

+ canMove(): bool

King

castlingDone: bool

canMove(): bool

Queen

+ canMove(): bool

Pawn

canMove(): bool

Rook

+ canMove(): bool

inconsistent.

Design pattern The following design patterns have been used in the class diagram: • Singleton design pattern: This pattern ensures the existence of a single instance The following design patterns can also be used to design chess:

pieces are notified to adapt to the changes accordingly. This decouples the pieces from the chessboard. Al-powered trainer At this stage, everything should be clear. If you encounter any confusion or ambiguity,

feel free to utilize the interactive AI-enabled widget below to seek clarification. This tool is designed to assist you in strengthening your understanding of the concepts.

• The Iterator design pattern would enable the game to move sequentially by • The State design pattern ensures the encapsulation of the state logic of each • The Observer design pattern enables the chess pieces to act as observers where