ComputerPlayer
- colour: PieceColour
- Level: int
- Seed: int

+ getMove(): pair<Position, Position>

MoveStrategy
+ virtual makeMove(): pair<Position, Position>

Level1
+ makeMove(): pair<Position, Position>

Level2
+ makeMove(): pair<Position, Position>

Level3
+ makeMove(): pair<Position, Position>

Level4
+ makeMove(): pair<Position, Position>

ChessController handles the game input loop. It can be replaced by main function. It interprets input from standard input and simulates the chess game. Player1 is white. It is included

<<<<enumeration>>>>
PlayerType

Human
ComputerLevel1
ComputerLevel2
ComputerLevel3
ComputerLevel4

ChessController
- showGraphics: bool
- player1Type: PlayerType
- player2Type: playerType
- Seed: int
- player1Point: int
- player2Point: int

+ simulateGame()

TextDisplay
+ update(BoardState)

GraphicsDisplay
+ update(BoardState)

<<<<enumeration>>>>
PieceType

King
Queen
Bishop
Rook
Knight
Pawn
None

<<<<enumeration>>>>
PieceColour

Black
White

<<struct>>
PieceInfo
+ type: PieceType
+ colour: PieceColour

Observer
+ *update(BoardState)*

Subject
+ attach(Observer)
+ detach(Observer)
# notifyObservers()

<<struct>>
Move
+ pieceInformation: PieceInfo
+ initialPos: Position
+ finalPos: Position
+ attackedPiece: PieceInfo

<<<<enumeration>>>>
GameState

Active
BlackWins
WhiteWins
BlackInCheck
WhiteInCheck
StaleMate

Position
- rank; int
- file: int

Piece
- pieceInformation: PieceInfo
- alive: bool
- moved: bool
- position: Position

+ getValidMoves(BoardState): vector<Position>

BoardModel
- Pieces: map<Position, Piece>
- state: BoardState
- previousMoves: vector<Move>
- gameState: GameState

+ MovePiece(Position, Postion)
+ init(BoardState)
+ getState(): BoardState
+ Undo()

<<struct>>
BoardState
+ BoardState: vector<vector<PieceInfo>>

InvalidMove
+ getMessage(): string

InvalidBoardSetup
+ getMessage(): string

Exception
- message: string

+ *getMessage(): string*

Rook
+ getValidMoves(BoardState): vector<Position>

Queen
+ getValidMoves(BoardState): vector<Position>

Bishop
+ getValidMoves(BoardState): vector<Position>

Knight
+ getValidMoves(BoardState): vector<Position>

Pawn
+ getValidMoves(BoardState): vector<Position>

King
+ getValidMoves(BoardState): vector<Position>

PPM

- width: int
- height: int
- pixels: vector<Pixel>

+ getWidth() : int &
+ getHeight() : int &
+ getPiexls() : vector<Pixel> &

Pixel

+ r: unsigned int
+ g: unsigned int
+ b: unsigned int

*

Image

+ render(PPM&) = 0

BasicImage

- source: string

+ BasicImage(string)
+ render(PPM&)

Decorator

component: Image

+ render(PPM&) = 0

PPMInputException

- s: string

+ getErrorMessage(): string

Flip

+ render(PPM&)

Rotate

+ render(PPM&)

Sepia

+ render(PPM&)

By: vrajang Parikh