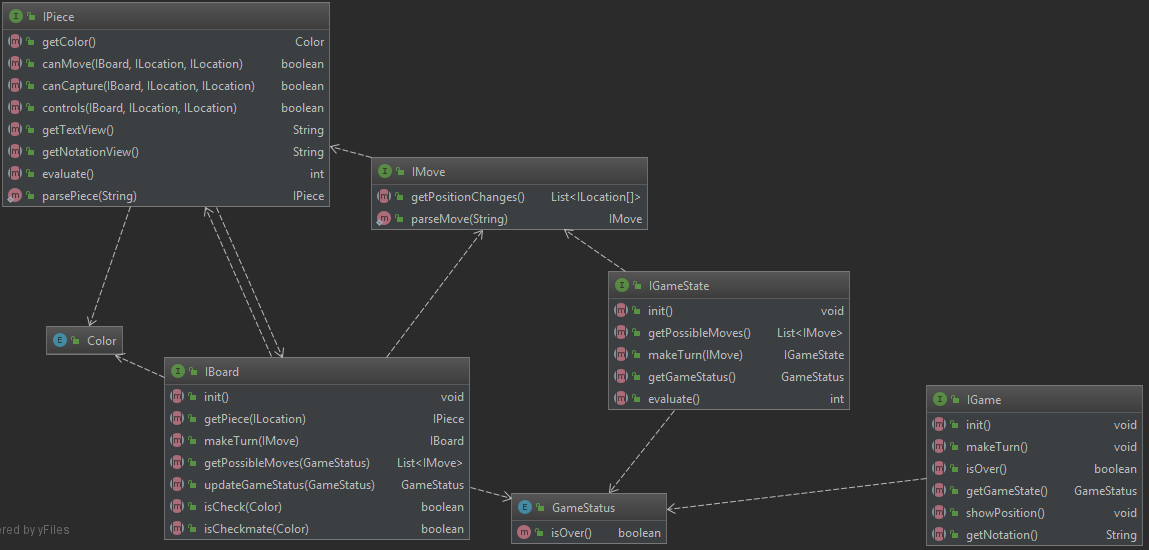
# Game mechanics

The following diagram contains all classes (and interfaces) which take part in game mechanics.



Now, we will take a look at each of these classes and interfaces and each of its methods

## IGame

The main chess game interface. Implemented by class ***Game.java***.

* ***init()*** Initiates chess game. Sets game state to start positions.
* ***makeTurn()*** Makes next turn of the game: asks current player for the move and applies it.
* ***isOver()*** Returns true or false, depending if game is over or not.
* ***getGameState()*** Return game status of the game. Game status is presented by enum **GameStatus.java**.
* ***showPosition()*** Shows current game state. In my implementation if shows board and a player to make move.
* ***getNotation()*** Returns string, containing all moves in chess notation format. It is output after game is over.

## IGameState

Interface representing current game state. Implemented by class **GameState.java**.

* ***init()*** Initiates games state. Sets game status to initial status (**GameStatus.WHITE\_TURN**) and board to initial position.
* ***getPossibleMoves()*** Returns collection of moves, which are possible in current game state.
* ***makeTurn(IMove)*** Returns new IGameState instance, which is obtained after given move is applied to current game.
* ***getGameStatus()*** Returns current game status of current game state.
* ***evaluate()*** Gets evaluation of current game state. It is used by MinimaxAIPlayer to evaluate current situation.

## GameStatus

Enum, representing game status. It contains 5 values: **WHITE\_TURN**, **BLACK\_TURN**, **WHITE\_WON**, **BLACK\_WON**, **DRAW**. Values has method ***isOver()***, which returns false for **WHITE\_TURN**, **BLACK\_TURN** and true for **WHITE\_WON**, **BLACK\_WON**, **DRAW**.

## Color

Enum, representing color (of player, of pieces, etc). It contains 2 values: **WHITE** and **BLACK**.

## IBoard

Interface, representing chess board. Implemented by class Board.java.

* ***init()*** Initates board: in particular puts chess pieces on their start positions.
* ***getPiece(ILocation)*** Returns piece on the given location of board.
* ***makeTurn(IMove)*** Returns new IBoard instance, which is obtained after given move is applied to current board.
* ***getPossibleMoves(GameStatus)*** Returns collection of moves, which are possible on the current board state with given game status.
* ***updateGameStatus(GameStatus)*** Returns new value of game status, obtained after given game status.
* ***isCheck(Color)*** Returns true if king of given color is checked, and false – otherwise.
* ***isCheckmate*** ***(Color)*** Returns true if king of given color is checkmated, and false – otherwise.

## IPiece

Interface, representing chess piece. Implemented by multiple classes: one class for each chess piece type.

* ***getColor()*** Returns color of piece.
* ***canMove(IBoard, ILocation, ILocation)*** Checks, if current piece can move to one location to another on the given board.
* ***canCapture(IBoard, ILocation, ILocation)*** Checks, if current piece can capture opponents’ piece on location from another on the given board.
* ***controls(IBoard, ILocation, ILocation)*** Checks, if current piece controls one location from another. Indeed this method is need to check if King can move to some location.
* ***getTextView()*** Returns string, representing current piece in the text view.
* ***getNotation()*** Returns string, representing current piece in chess notation.
* ***evaluate()*** Returns evaluation of current chess piece. It is used to calculate current board evaluation.
* ***parsePiece(String)*** Static method for parsing IPiece instance from the string.

## IMove

Interface, representing chess move. Implemented by **Move.java**, **Castling.java**, **Capture,java**.

* ***getPositionChanges()*** Returns collection of location pairs: each pair represents “from” and “to” location.
* ***parseMove()*** Static method for parsing IMove instance from string.