Assignment 1 – Chess game (part I)

In this ChessGame assignment several subjects of the previous lessons will be used (enums, classes, 2-dim arrays, errorhandling, ...).

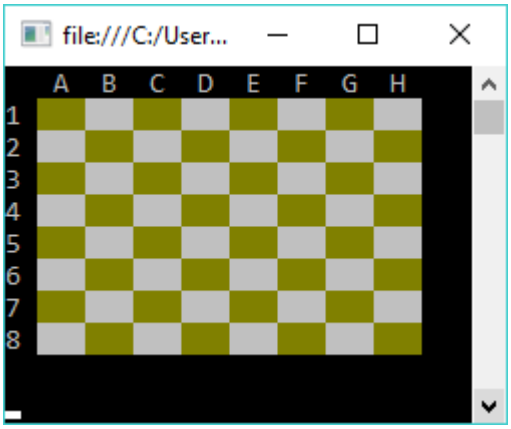
a) To create a chess game we need of course chess pieces. A chess piece has a color (Black or White) and a type (Pawn, Rook, Knight, ...). Because there are limited options for the color (Black or White), and limited options for type (Pawn, Rook, Knight, Bishop, King and Queen), we will create 2 enumerations: `ChessPieceColor` and `ChessPieceType`. Store these 2 enumerations in a separate file 'Enumerations.cs'.

b) Create a `class ChessPiece` that contains 2 fields: a color and a type; store this class in file 'ChessPiece.cs'.

c) Besides the chess pieces we also need a chess board, a 2-dimensional array with 8x8 fields. On this chessboard we will put chess pieces. Create in the Start method a 2-dim array (chessboard) of type `ChessPiece[,]` with 8 rows and 8 columns.

d) Create a method with signature `void InitChessboard(ChessPiece[,] chessboard)`.
This method fills the complete array with the value `null` (which means '*no object*'). Use a nested loop.
→ Call this method `InitChessboard` from the Start method.

e) Create a method with signature `void DisplayChessboard(ChessPiece[,] chessboard)`.
This method displays the chessboard, including the row-numbers and column-letters (see screenshot below).
Hint: if row + column is even, then use a dark background color (e.g. DarkYellow), otherwise use a light background color (e.g. Gray). For now, print 3 spaces for each cell.
→ Call this method `DisplayChessboard` from the Start method.

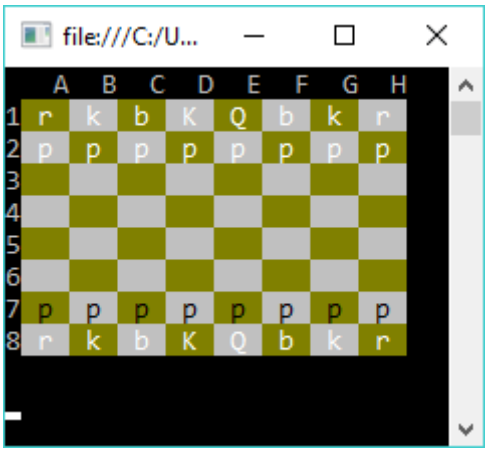


f) Create a method with signature `void PutChessPieces(ChessPiece[,] chessboard)`.
When using a loop that processes the 8 columns, we can quite easily put all chess pieces on the chessboard. In each column 4 chess pieces must be put on the board: a white pawn (at the 2nd row), a black pawn (at the 7th row), and 2 other pieces on the first and last row. For these last 2 pieces, we will use a helper-array in order to get the right chess piece for each column:

```
ChessPieceType[] order = { CPT.Rook, CPT.Knight, CPT.Bishop, CPT.King,
                           CPT.Queen, CPT.Bishop, CPT.Knight, CPT.Rook};
```

(in the order array, `CPT` is used instead of `ChessPieceType`, only to save space here...)
→ Call this method `PutChessPieces` from method `InitChessboard` *(as last statement)*.

g) Create a method with signature `void DisplayChessPiece(ChessPiece chessPiece)`.
This method displays the given chess piece. If there is no chess piece (`chessPiece == null`), then simply print 3 spaces and leave. Otherwise, set the `ForegroundColor` to Black or White (depending on the chess piece), and then write the first letter of the chesspiece type, lowercase if not King nor Queen: p, r, k, b, K, or Q; surround this letter with a space. You can get the type-string with: `chessPiece.type.ToString()`.
→ Call this method `DisplayChessPiece` from the method `DisplayChessboard`, 8x8 times (*instead of printing 3 spaces there*), each time passing the chessboard content.



p=pawn, r=rook, k=knight, b=bishop, K=king, Q=queen

Assignment 2 – Chess game (part II)

What is a chess game without user interaction? We will ask the user in this 2nd assignment which chess piece should be moved, to what position.

A position on the chessboard is indicated with a letter (A..H) and a number (1..8). In the chessboard array the rows and columns are indexed with integers 0..7. Some examples: A7 (column 0, row 6), G3 (column 6, row 2). This means that we have to convert the entered position to the corresponding row and column of the chessboard.

a) Create a `class Position` with (`int`) field `row` and (`int`) field `column`. Store this class in a separate file 'Position.cs'.

b) Create a method with signature `Position ReadPosition(string question)`.
This method prints the question and reads the position (e.g. "F3"). If the entered position is not a valid chessboard position, then an exception is thrown containing a meaningful message (e.g. "invalid position"). If the entered position is a valid chessboard position, then a `Position` is returned, containing the corresponding row- and colum-index.

To convert the entered position (e.g. "F3") into a row-index and a column-index you can use the following code:

```
int column = userPos[0] - 'A';
int row = int.Parse(userPos[1].ToString()) - 1;
```

c) Create a method with signature `void PlayChess(ChessPiece[,] chessboard)`.
In an endless loop:
- read the from-position (a position that contains a chess piece);
- read the to-position (a position the chess piece should be moved to);
- check if the move (from → to) is valid, with method `CheckMove` *(see later)*;
- do the move, with method `DoMove` *(see later)*;
- display the chessboard;

Inside this endless loop all Exceptions must be caught, and the exception message must be displayed. The `ReadPosition` can throw an Exception, but also method `CheckMove` (see later).

Call the `PlayChess` method from the Start method. *Verify that the Start method now only calls 3 methods: `InitChessboard`, `DisplayChessboard` and `PlayChess`.*

Assignment 3 – Chess game (part III)

In this last part we will actually move a chess piece, as the user has requested (from → to). This move should only be done if the requested move is valid (a rook can only move horizontally/vertically, a bishop can only move diagonally, ...).

The `PlayChess` method calls a few methods that are still undefined, `CheckMove` and `DoMove`. We will implement these methods now.

a) Create a method `void DoMove(ChessPiece[,] chessboard, Position from, Position to)`. This method moves the chess piece at the from-position to the to-position. This can be done in 2 steps: first assign to chessboard[to] the value of chessboard[from], and then assign to chessboard[from] the value `null`.
→ You can test if a chess piece really moves, simply by running the application again *(make sure that method `PlayChess` calls method `DoMove`...)*.

b) Create a methode `void CheckMove(ChessPiece[,] chessboard, Position from, Position to)`. This method checks if the specified move (from → to) is valid. If the move is not valid, an Exception is thrown containing a meaningful message (e.g. "no chess piece at from-position").
The following checks are done:
- the from-position contains a chess piece;
- the to-position does not contain a chess piece;
- the move is valid for the specific chess piece (at the from-position); to check this, call method `ValidMove`, see next;

c) Create a method `bool ValidMove(ChessPiece chessPiece, Position from, Position to)`. This method checks if the move is valid for the given chess piece. First calculate the positive horizontal difference (hor) and the positive vertical difference (ver). To get the positive value (>=0) use `Math.Abs(...)`. In general, if both hor and ver are 0, then it's not a valid move. Otherwise you can determine if the move is valid by using the following rules *(use a switch to test the type of the chess piece)*:

- `rook` `hor * ver = 0`
- `knight` `hor * ver = 2`
- `bishop` `hor = ver`
- `king` `hor = 1 and/or ver = 1`
- `queen` `hor * ver = 0 or hor = ver`
- `pawn` `hor = 0 and ver = 1`

Assignment 4 – Using a class ChessGame

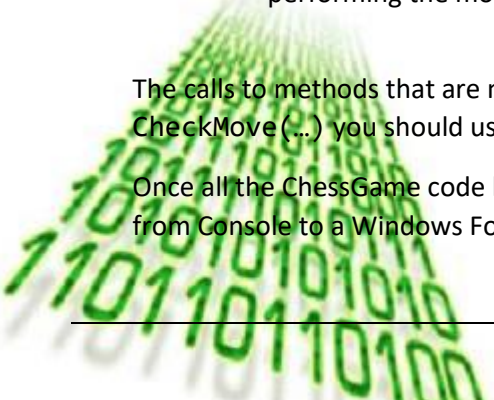
This last assignment is optional but strongly recommended!

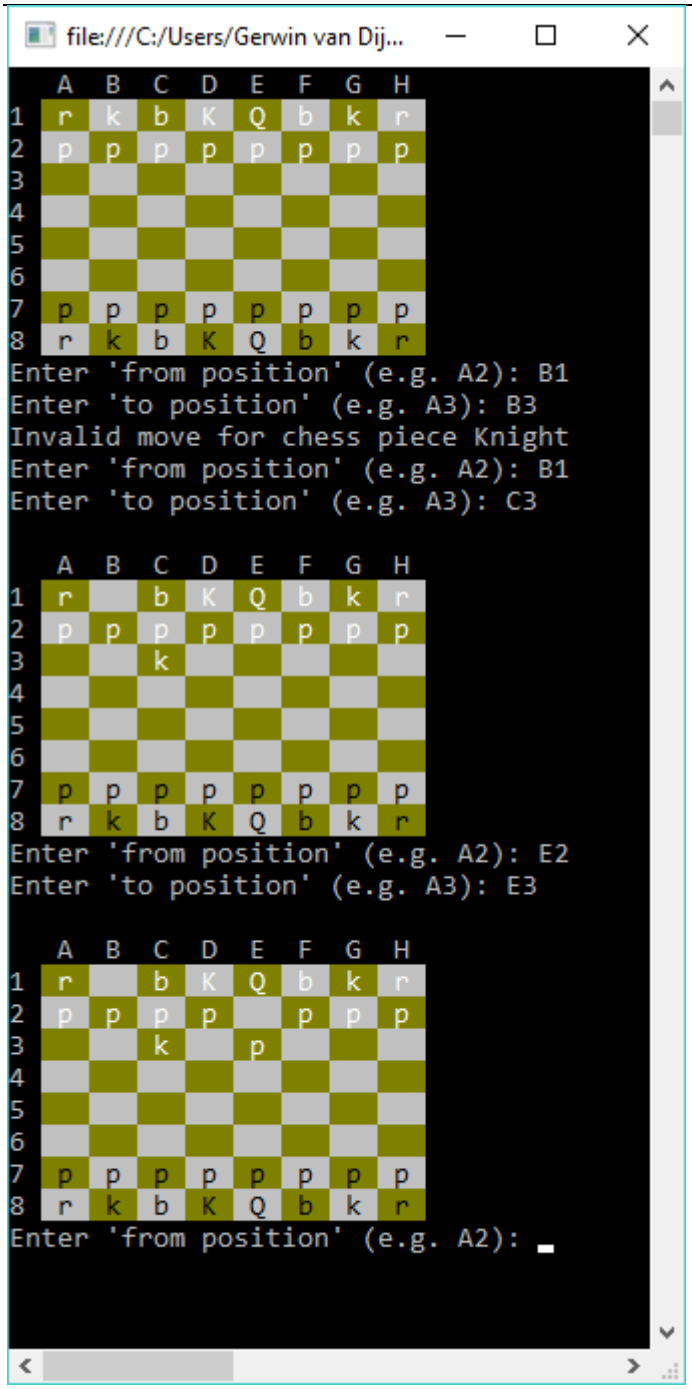
Most of the ChessGame code is stored in file 'Program.cs'. It is much better to split up the code (code for userinteraction code and code for the chessgame → Separation of Concerns!). Your task is to add a `class ChessGame` which contains the chessboard and is responsible for the following tasks:

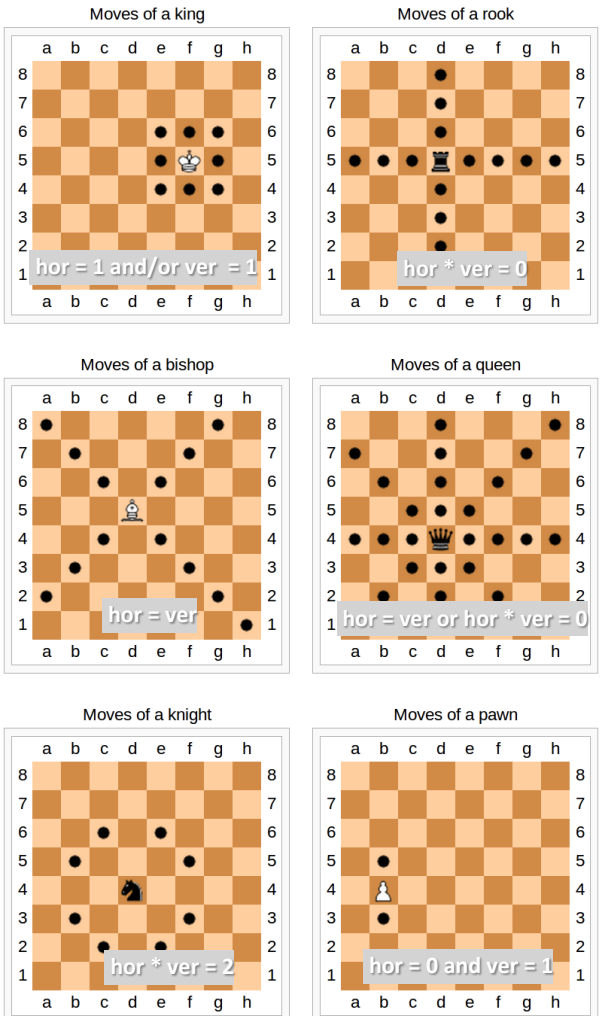
- creating the chessboard;
- initializing the chessboard (methods `InitChessboard` and `PutChessPieces`);
- checking moves (methods `CheckMove` and `ValidMove`);
- performing the moves (method `DoMove`);

The calls to methods that are moved to `class ChessGame` must be changed, e.g. instead of using `CheckMove(…)` you should use `chessGame.CheckMove(…)`.

Once all the ChessGame code has been moved to a separate class, it will be much easier to change the project type from Console to a Windows Forms (or WPF) application!




file:///C:/Users/Gerwin van Dij...
A B C D E F G H
1 r k b K Q b k r
2 p p p p p p p p
3
4
5
6
7 p p p p p p p p
8 r k b K Q b k r
Enter 'from position' (e.g. A2): B1
Enter 'to position' (e.g. A3): B3
Invalid move for chess piece Knight
Enter 'from position' (e.g. A2): B1
Enter 'to position' (e.g. A3): C3
A B C D E F G H
1 r b K Q b k r
2 p p p p p p p p
3 k
4
5
6
7 p p p p p p p p
8 r k b K Q b k r
Enter 'from position' (e.g. A2): E2
Enter 'to position' (e.g. A3): E3
A B C D E F G H
1 r b K Q b k r
2 p p p p p p p
3 k p
4
5
6
7 p p p p p p p p
8 r k b K Q b k r
Enter 'from position' (e.g. A2):


Moves of a king
hor = 1 and/or ver = 1
Moves of a rook
hor * ver = 0
Moves of a bishop
hor = ver
Moves of a queen
hor = ver or hor * ver = 0
Moves of a knight
hor * ver = 2
Moves of a pawn
hor = 0 and ver = 1