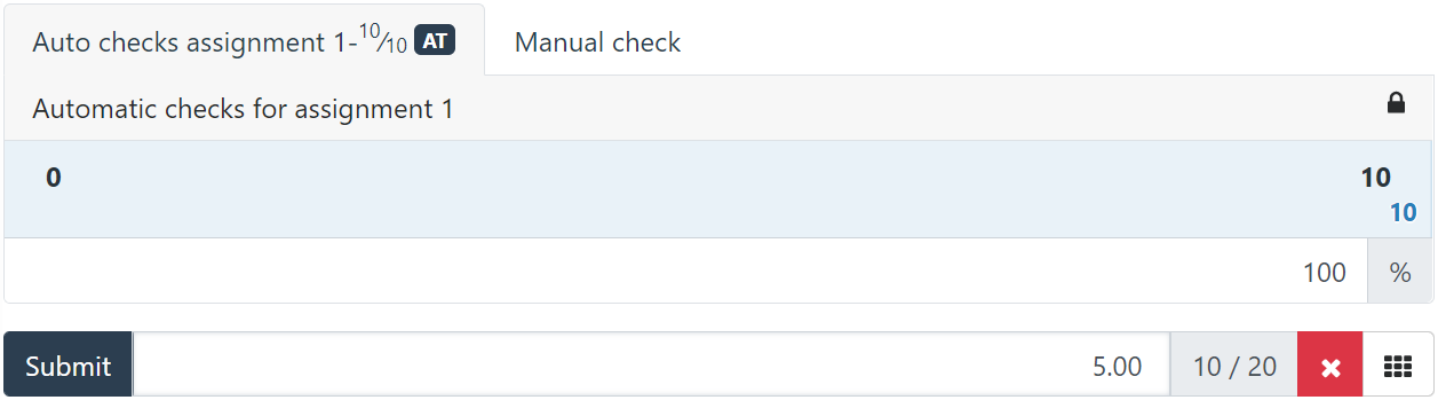
Week 6 assignments

When creating the program code, you must apply the following basic principles:

- create a separate project for each assignment;
- use name ‘assignment1’, ‘assignment2’, etcetera for the projects;
- create one solution for each week containing the projects for that week;
- make sure the output of your programs are the same as the given screenshots;

CodeGrade auto checks

Make sure all CodeGrade auto checks pass (10/10) for your assignments. The manual check will be done by the practical teacher.



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, ...).

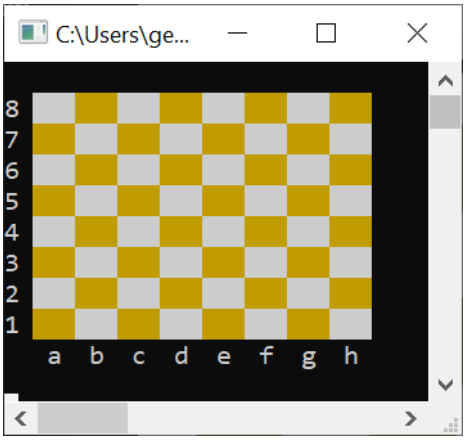
a) To create a chess game we need of course chess pieces. A chess piece has a color (Black or White) and a type (Pawn, Rook, Knight, ...). Because there are limited options for the color (Black or White), and limited options for type (Pawn, Rook, Knight, Bishop, King and Queen), we will create 2 enumerations: `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 light background color (e.g. Gray), otherwise use a dark background color (e.g. DarkYellow).
For now, print 3 spaces for each cell.
→ Call this method `DisplayChessboard` from the Start method.



Chess game - part II

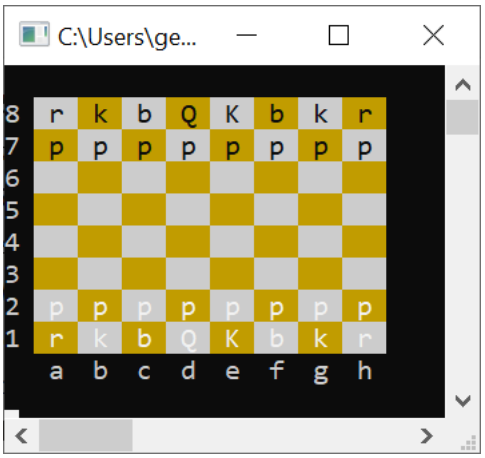
In this 2nd part of the assignment, we're going to put chesspieces on the chessboard, at their start positions.

a) 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 (see screenshot below). For these last 2 pieces, you can use a helper-array in order to get the right chess piece for each column:

```
ChessPieceType[] order = { CPT.Rook, CPT.Knight, CPT.Bishop, CPT.Queen,
                           CPT.King, 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)*.

b) 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, Q, or K; surround this letter with a space. You can get the type-string with: `chessPiece.type.ToString()`.
→ Call this method `DisplayChessPiece` from method `DisplayChessboard`, 8x8 times (*instead of printing 3 spaces there*), each time passing the chessboard content: `DisplayChessPiece(chessboard[row, col]`.



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

You can upload your assignment, to check your program. The first CodeGrade test ("Check part I and II") should be ok.

Chess game - part III

What is a chess game without user interaction? We will ask the user in this 3rd part of the assignment what move we have to make (a from-position and a to-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 (see screenshot): a7 → column 0 and row 1, g3 → column 6 and row 5. This means that we have to convert the entered position to the corresponding row and column of the chessboard.

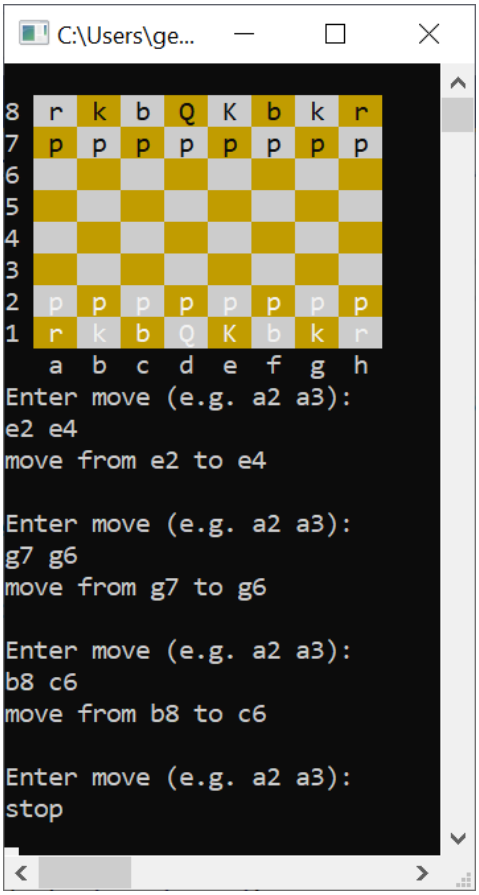
a) Create a `class Position` with (`int`) field `row` and (`int`) field `column`. Store this class in a separate file 'Position.cs'.

b) Create a method with signature `Position String2Position(string pos)`.
To convert the given position (e.g. "f3") into a row-index and a column-index you can use the following code:

```
int column = pos[0] - 'a';
int row = 8 - int.Parse(pos[1].ToString());
```

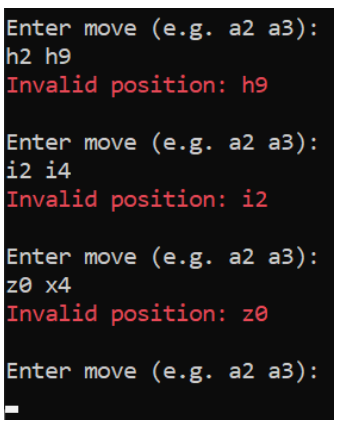
If the given position (parameter pos) is not a valid chessboard position, then throw an exception with the message "invalid position <pos>".
If the given position is a valid chessboard position, then return a `Position`, containing the corresponding row- and colum-index.

c) Create a method with signature `void PlayChess(ChessPiece[,] chessboard)`.
In an endless loop:
- read the move (from-position and to-position), for example: a2 a3
- if the move is "stop", then break out of the loop;
- split the entered move (using input.Split(' '), and get the from-position and to-position using (twice) method `String2Position`;
- write "move from <from> to <to>", see screenshot to the right;
- do the move, with method `DoMove` *(see later);*
- display the chessboard *(see later);*



Inside this endless loop all Exceptions must be caught, and the exception message must be displayed. The `String2Position` can throw an Exception, but also method `DoMove` (see later).

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



You can upload your assignment, to check your program. The 2nd CodeGrade test ("Check part III") should be ok. ✔

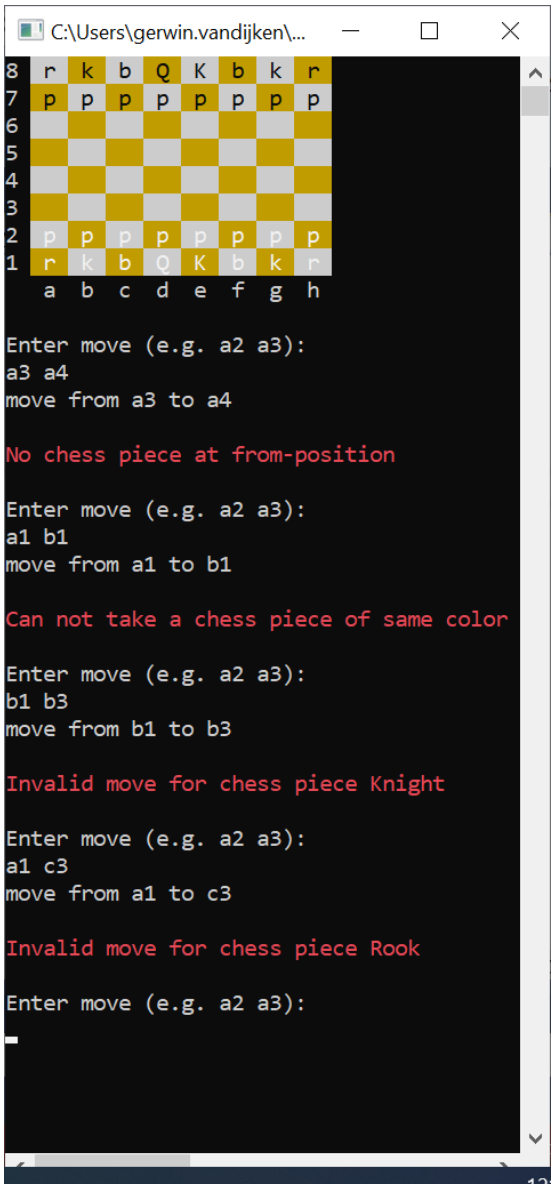
Chess game - part IV

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 (e.g. a rook can only move horizontally/vertically).

We will implement the last 2 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` and displays the chessboard again..)*.

b) Create a method `void CheckMove(ChessPiece[,] chessboard, Position from, Position to)`. We will do the following checks in this method:
- the from-position contains a chess piece (`!= null`);
- if the to-position contains a chess piece (`!= null`) then it must be chess piece of the opponent;
If one of these check fails, an Exception is thrown containing a meaningful message ("No chess piece at from-position" or "Can not take a chess piece of same color").
→ Call method `CheckMove` from method `DoMove`, before doing the move.

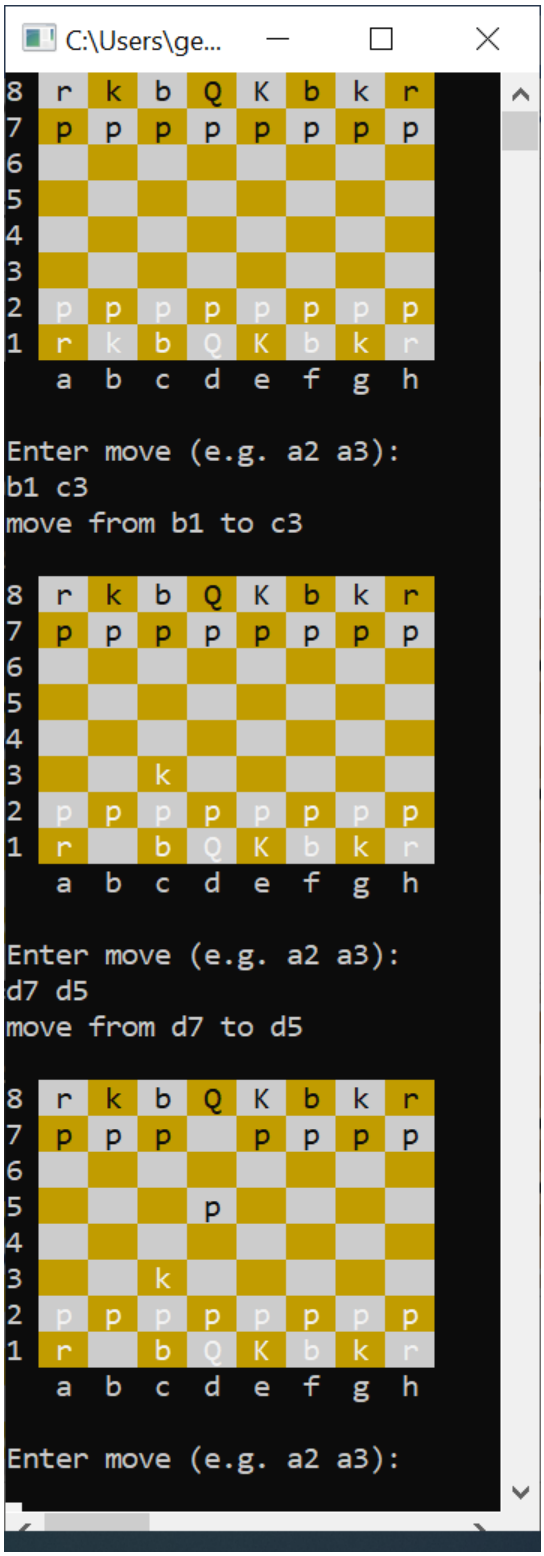


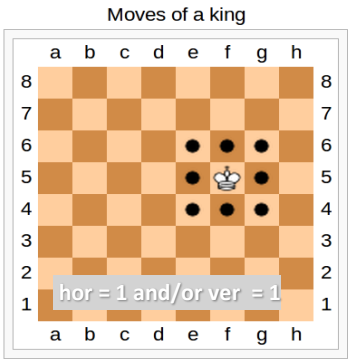
c) We now will add some more checks to method `CheckMove`: check if the move is valid for the given chess piece. First calculate the positive horizontal difference (hor) and the positive vertical difference (ver). To get the positive value (>=0) use `Math.Abs(...)`. In general, if both hor and ver are 0, then it's not a valid move (throw exception with message "No movement"). 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)*:

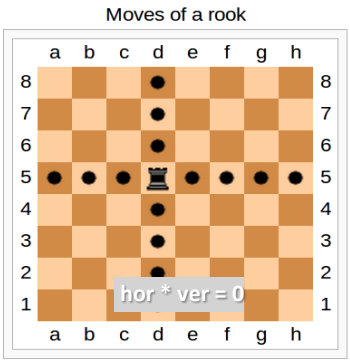
- `rook` `hor * ver = 0`
- `knight` `hor * ver = 2`
- `bishop` `hor = ver`
- `king` `hor = 1 and/or ver = 1`
- `queen` `hor * ver = 0 or hor = ver`
- `pawn` `hor = 0 and ver = 1`

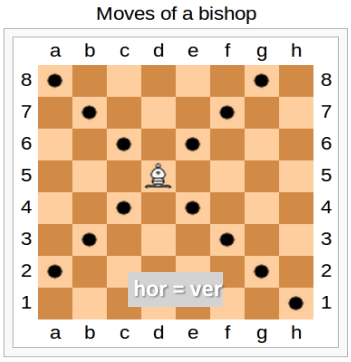
If this check fails, throw an Exception with message "Invalid move for chess piece <chess piece>".

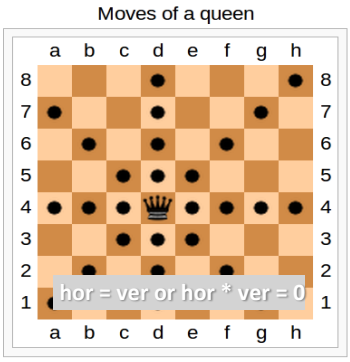
You can upload your assignment, to check your program. The last 2 CodeGrade tests ("Check part IV errors" and "Check part IV movements") should be ok.

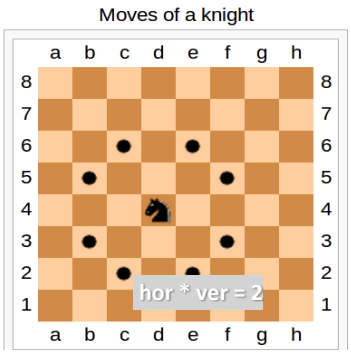

C:\Users\ge...
Enter move (e.g. a2 a3):
b1 c3
move from b1 to c3
Enter move (e.g. a2 a3):
d7 d5
move from d7 to d5
Enter move (e.g. a2 a3):

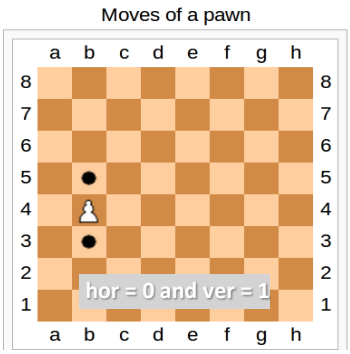

Moves of a king
hor = 1 and/or ver = 1


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

ChessGame – part V *(not mandatory)*

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!). 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 the move (method `CheckMove`);
- performing the move (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!