Functional Programming
Project: Chess (part 1)

Assistant: Noah Van Es
noah.van.es@vub.be

The final grade of the course is determined by an oral exam and a project (each counts for 50% of the total grade for the course). The project consists out of two parts. A non-empty solution for both parts has to be submitted in order to get a grade. This document describes the first part of the project.

Project description

The goal of the first part of this project is to implement a terminal-based chess game. The game should be playable in two modes: a multiplayer mode that allows two humans to play against each other, as well as a singleplayer mode against an AI opponent (based on a simple minimax algorithm). The rules of chess are well-defined and widely available online (cf. e.g. `https://www.chess.com/learn-how-to-play-chess`).

Practical information

Submission The deadline for this assignment is **May 21, 2024 (23:59)**. You submit all your code and other materials in single ZIP-file through the Canvas platform.

Grading Note that your project will not only be graded on functionality, but also on the quality of your implementation (programming style, performance considerations, ...).

Individual work In case you have questions or need clarification to complete this project, feel free to contact the assistent by mail (`noah.van.es@vub.be`) or make an appointment to schedule a meeting (virtual or at my office). **This project is to be made strictly individually and on your own. This means that you must create your project on an independent basis and you must be able to explain your work, reimplement it under supervision, and defend your solution. Copying work (code, text, etc.) from or sharing it with third parties (e.g., fellow students, websites, GitHub, etc.) is not allowed. Electronic tools will be used to compare all submissions with online resources and with each other, even across academic years. Any suspicion of plagiarism will be reported to**

the Dean of Faculty without delay. Both user and provider of such code will be reported and will be dealt with according to the plagiarism rules of the examination regulations (cf. OER, Article 118). The dean may decide on (a combination of) the disciplinary sanctions (cf. OER, Article 118§5).

Project assignment

Your project should be written entirely in Haskell and compiled using the Glasgow Haskell Compiler[1]. A file `Main.hs` is made available on Canvas to help you with the interactive aspects of your application (that is, this file provides a game loop to read and process an input command from the terminal on each iteration). The remainder of this section describes the exact requirements that should be met for this assignment.

Game Interaction At each turn, the current state of the board should be displayed and the current user should be offered the opportunity to enter a move as a command. If the command is valid and the move is legal, the move is performed and the updated state of the board is displayed. This continues until an "end condition" (cf. inf.) is reached. The snippet below illustrates a possible interaction with the application: (i) the initial board is rendered, (ii) the white player performs the traditional king's pawn opening, (iii) the black player entered an illegal pawn move and is expected to enter another command.

```
> ./chess
8 | r n b q k b n r
7 | p p p p p p p p
6 | . . . . . . . .
5 | . . . . . . . .
4 | . . . . . . . .
3 | . . . . . . . .
2 | P P P P P P P P
1 | R N B Q K B N R
--+----------------
  | A B C D E F G H
d2d4
1 | R N B K Q B N R
2 | P P P P . P P P
3 | . . . . . . . .
4 | . . . . P . . .
5 | . . . . . . . .
6 | . . . . . . . .
7 | p p p p p p p p
8 | r n b k q b n r
--+----------------
  | H G F E D C B A
d7d4
Illegal pawn move
```

[1] `https://www.haskell.org/ghc/`

Board Rendering The precise rendering of the board is left to your own appreciation. However, it should adhere to the following rules: (i) each piece should be properly differentiated, (ii) file letters and rank digits should be displayed to help the player encode their desired move, (iii) the current player should always have their starting position at the bottom of the board – i.e., the board should be flipped at every turn.

Command Format Commands are encoded using four letters which respectively indicate the initial file (i.e., column), the initial rank (i.e., row), the destination file, and the destination rank. Hence, a valid command should satisfy the following regex: `^[a-h][1-8][a-h][1-8]/$`. For instance, the command `d2d4` encodes moving the piece residing at the square `d2` to the square `d4`.[2] If a command is invalid, an appropriate message should be displayed, prompting the current player to enter their move again.

Chess Rules Before moving to the next turn, the legality of the move entered by the player should be checked. The rules of chess are well-defined and widely available online (cf. e.g. `https://www.chess.com/learn-how-to-play-chess`). In particular, ensure that queenside and kingside castling, "en passant" pawn capture and pawn promotion[3] are also supported. Note that moves that leave the player in "check" are illegal.
When an endgame is reached an appropriate message should be displayed. It is required to detect the three following endgames: "checkmate", "draw by stalemate" and "draw by insufficient material". "Checkmate" happens when a player is checked and has no legal moves. "Draw by stalemate" happens when a player is not in check and has no legal moves. "Draw by insufficient material" happens in the four following cases: king against king, king against king and bishop, king against king and knight, and king and bishop against king and bishop where both bishops reside on squares of the same colour.

AI opponent Ensure that the configuration of your game allows the user to choose between a multiplayer game (with two human players) or a singleplayer game (against an AI opponent). For this assignment, the term "Artificial Intelligence" is used rather generously, as it refers to a simple decision algorithm based on a bounded exploration, using the well-known "minimax" algorithm[4]. This algorithm operates as follows:

1. First, the AI opponent has to explore all the possible sequences of moves (made by both the human player and the AI) up until reaching the targeted depth. For instance, an exploration depth of 3 means that the AI must make its decision based on all the possible sequences of: AI moves, player moves, followed by AI moves. This results into a bounded tree-like structure where each node represents the choices that both players have to make in alternation. Note that the depth of the tree may not be uniform as moves resulting in an endgame should not have any successors.

[2]Because castling (both queenside and kingside) is an illegal move for the king in normal circumstances, it can be encoded as a king move. For instance, `e1g1` (resp. `e1c1`) may encode kingside (resp. queenside) castling for the white player.

[3]To simplify the player interaction, we assume that each promoted pawn is automatically converted to a queen.

[4]`https://en.wikipedia.org/wiki/Minimax`

Pawn	1 point
Knight	3 points
Bishop	3 points
Rook	5 points
Queen	9 points

Table 1: A possible valuation of chess pieces.

2. The AI then uses a *fitness function* to estimate the "value" of the outcome for each leaf node in the exploration tree. If the AI plays white, it should attempt to maximize the fitness function; if the AI plays black, it should attempt to minimize the fitness function. For this assignment, the fitness function simply corresponds to the difference of material present on the board using the relative value of pieces shown in Table 1. For instance, if white has the king and a queen (9 points), while black has the king and two rooks (5 points each), then the value of the fitness function will be -1 (implying that black is slightly favored). Of course, checkmating the opponent is the ultimate goal and must outweigh any other situation. Therefore, the fitness function is worth $+\infty$ when white checkmates black and $-\infty$ when black checkmates white. In case of stalemate, the value of the fitness function is 0 regardless of the material present on the board.

3. After assigning a value to each leaf node in the exploration tree using the fitness function, the AI works its way back to the root of the tree. It assigns nodes a value based on the value of its children as follows (assuming the AI plays white): if a node represents an AI move, it is given the highest (i.e., max) of its children's values; if a node represents a human player's move, it is given the lowest (i.e., min) of its children's values. At the last step, the AI picks the move on the optimal path.

As an optimization, **ensure that your implementation avoids recomputing parts of the exploration tree between moves when possible**. Indeed, although the value of each node has to be recomputed at every turn, the moves that were already computed for the previous turn can be cached and reused. More precisely, upon performing a move, the root of the exploration tree must be updated accordingly, and the remaining leafs are used to compute the next possible moves to maintain the depth of the exploration tree.