Learning to Program with F#
Exercises
Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

0.1 Chess

0.1.1 Teacher's guide

Emne Classes, inheritance and UML diagrams

Sværhedsgrad Svær

0.1.2 Introduction

Sporring, "Learning to program with F#", 2017, Chapter 21.4 describes a simplified version of Chess with only Kings and Rooks, and which we here will call Simplechess, and which is implemented in 3 files: `chess.fs`, `pieces.fs`, and `chessApp.fsx`. In this assignment you are to work with this implementation.

0.1.3 Exercise(s)

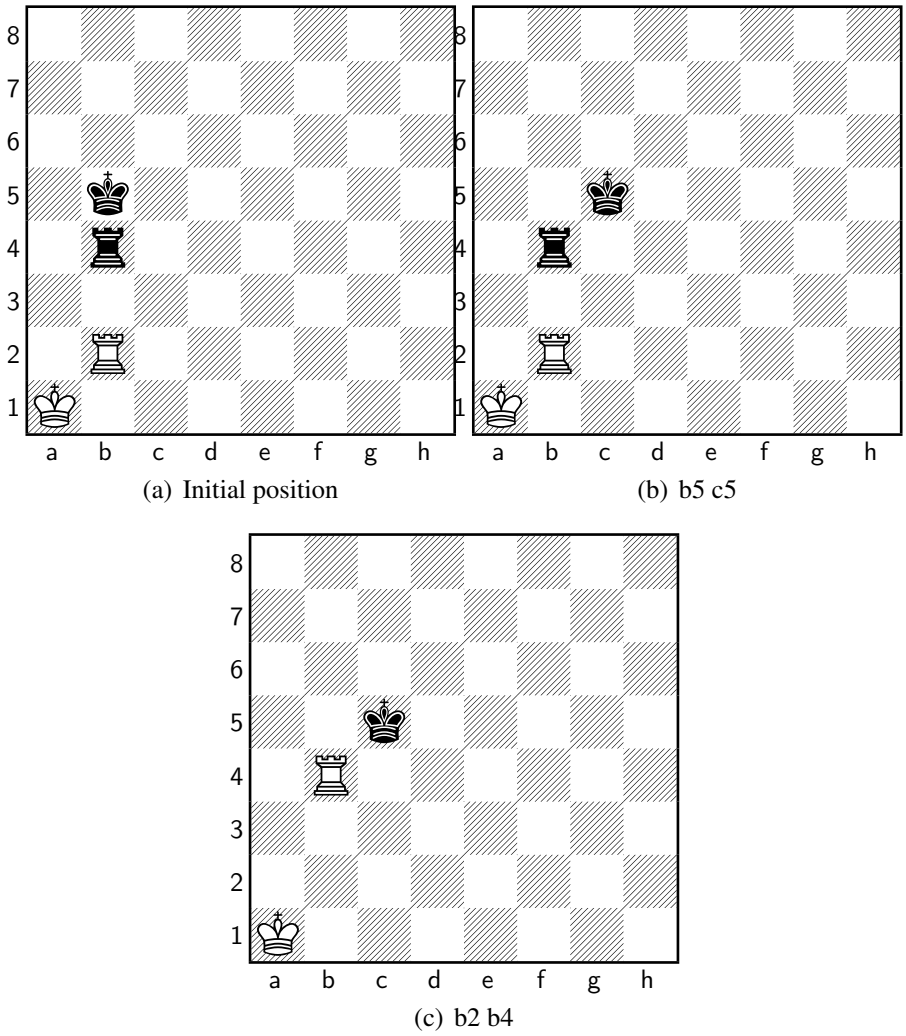
0.1.3.1: Produce a UML diagram describing the design presented of Simple Chess in the book.

0.1.3.2: The implementation of `availableMoves` for the King is flawed, since the method will list a square as available, even though it can be hit by an opponents piece at next turn. Correct `availableMoves`, such that threatened squares no longer are part of the list of vacant squares.

0.1.3.3: Extend the implementation with a class `Player` and two derived classes `Human` and `Computer`. The derived classes must have a method `nextMove`, which returns a legal movement as a codestring or the string "quit". A codestring is a string of the name of two squares separated by a space. E.g., if the white king is placed at a4, and a5 is an available move for the king, then a legal codestring for moving the king to a5 is "a4 a5". The player can be either a human or the computer. If the player is human, then the codestring is obtained by a text dialogue with the user. If the player is the computer, then the codestring must be constructed from a random selection of available move of one of its pieces.

0.1.3.4: Extend the implementation with a class `Game`, which includes a method `run`, and which allows two players to play a game. The class must be instantiated with two player objects either human or computer, and `run` must loop through each turn and ask each player object for their next move, until one of the players quits by typing "quit".

0.1.3.5: Extend `Player` with an artificial intelligence (AI), which simulate all possible series of moves at least $n \geq 0$ turns ahead or until a King is strikken. Each series should be given a fitness, and the AI should pick the move, which is the beginning of a series with the largest fitness. If there are several moves which have series with same fitness, then the AI should pick randomly among them. The fitness number must be calculated as the sum of the fitness of each move. A move, which does not strike any pieces gets value 0, if an opponent's rook is strikken, then the move has value 3. If the opponent strikes the player's rook, then the value of the move is -3. The king has in the same manner value ± 100. As an example, consider the series of 2 moves starting from Figure 1(a), and it is black's turn to move. The illustrated series is `["b5 b6"; "b2 b4"]`, the fitness of the corresponding moves are `[0;-3]`, and the fitness of the series is `-3`. Another



(a) Initial position (b) b5 c5

(c) b2 b4

Figure 1: Starting at the left and moving white rook to b4.

series among all possible is `["b5 b6"; "b2 c2"]`, which has fitness `0`. Thus, of the moves considered, `"b5 b6"` has the maximum fitness of `0` and is the top candidate for a move by the AI. Note that a rook has at maximum 14 possible squares to move to, and a king 8, so for a game where each player has a rook and a king each, then the number of series looking n turns ahead is $\mathcal{O}(22^n)$.

0.1.3.6: Make an extended UML diagram showing the final design including all the extending classes.