PF1 Project - Chess

Developer Guide

Section 1: Overview

Thank you for reading the developer guide of this project. Please read all sections carefully in order to understand the source and functionalities of this program.

This project is done in the programming language Racket (Advanced Student) by following *a functional programming structure of the code* – meaning only functions can be found inside the code and no references of objects or classes.

The code consists of total **27** functions which are mainly driven and executed by the **big-bang** function. The following libraries need to be included in the code in order for it to be functional:

(**require** racket/base)	Used for the command struct-copy to copy an already existing struct and change certain field values of the struct if needed. Mainly used to avoid the use of the command **!set**.
(**require** 2htdp/image)	Used for visualizing chessboard elements and providing the ability to draw particular two-dimensional figures on a given scene.
(**require** 2htdp/universe)	Used for the ability to implement the functionality of creating interactive and graphical programs, such as chess. This library is the heart of the entire program.

Done by:
- Genaro Di Stefano
- Harkeerat Singh Sawhney
- Bojan Lazarevski

Section 2: Functions Overview

The program consists of 3 types of function, each type having its own usages:

- Main Functions
- Auxiliary Functions
- Calculating Functions

The *main functions* are those that are crucial for the effective and correct implementation and functionality of the program. The most important one is the **big-bang** function that runs the other main functions: **draw-appstate**, **handle-key** and **handle-mouse**. All these functions are built from other *auxiliary functions* that help finalize the complete look and feel of the chess game. These are mainly the drawing auxiliary functions that focus on correct piece placement on the chessboard, right positioning of the piece selector, winning/losing announcement messages, etc. All logical and mathematical algorithms behind the movements of the pieces on the chessboard are covered by the *calculating functions.* These can also be clarified as main functions because every calculating function has some interconnections to another calculating function due to the fact that in chess every move is dependent on the next move as well as to the previous one. These cover the functions that determine whether a player can move a certain piece to the specified position, if a player can take the opponent's piece with his next move, whether the next move will stay inside the chessboard and so on.

All functions depend on each other. Removing or changing a certain function may result in an unwanted behavior of the chess program.

Every function is individually explained in the design recipe of the program.

Done by:
- Genaro Di Stefano
- Harkeerat Singh Sawhney
- Bojan Lazarevski

Section 3: Main Functions

The functionality of the program is secured by the main functions. The most important one is the big-bang that is triggered by several events in the AppState:

Trigger Event:	Calls Function:
to-draw	**draw-appstate** Draws the entire chessboard recursively including all chessboard elements. At the beginning of the game, it shows a menu and at the end of the game it shows who won the game. The selector on the chessboard is drawn on the location that the mouse points at or is selected by the space key on the keyboard.
on-key	**handle-key** Responsible for the controls of the game. It triggers different events depending on the keys pushed. Player one plays with buttons WASD and the other player uses the arrows to navigate. Clicking on the space button confirms a move and places the chosen piece on the selected location.
on-mouse	**handle-mouse** This function *exploits* the handle-key function. Depending on a certain mouse event a corresponding key event is triggered. This is implemented for easier playing experience. To slightly reduce the lag delays, this function can be deleted (the program remains intact).

Section 4: Calculating Functions

Due to the high complexity of the chess algorithms, many calculating functions can be found in the code. Mainly, these are focused on the logic behind all chessboard elements such as the pieces, the selectors, board navigation and positioning, etc. Reading the code, it might seem that there is a lot of repetition, which is somewhat accurate, but this is due to the fact that the movements between the two players are mirrored both from the x-axis and y-axis. Thus, for every move, 2 separate conditions need to be coded to capture the movements of both players. The most important calculating functions are the following:

in-board?	Calculates whether the next move will be out of the chessboard borders by checking the x and y axis of the next movement
can-release?	Determines if the next move can be placed on the selected position. For example, the pawn can be moved two blocks up if it is on the starting position. The rook can move only horizontally or vertically, etc. It blocks unallowed movements and unselects the currently selected piece.
control-selector	Moves the chessboard selector by 1 block respectively to the given direction by the keys, arrows or mouse drag.
can-take?	Calculates if a certain move will take an opponent piece. A few positioning conditions have to be met for this to happen.
take-piece	If the above function returns #true, the next move captures the opponent piece and removes that piece from the list of pieces and from the appstate structure.

Done by:
- Genaro Di Stefano
- Harkeerat Singh Sawhney
- Bojan Lazarevski

Section 5: Known Bugs

We are aware of some of the bugs that are appearing in our code. However, if these bugs are not exploited, the game can function normally and effectively. The current bugs that are present in the game are:

- The pieces can jump over other pieces of same side. This applies to pieces that have free board selection such as rook, bishop and queen.
- If using the mouse, the function can-release? does not recognize some of the events. For instance, the rook can be placed at any location in the board. However, if you select the rook and use the keyboard to navigate, this will not happen.