Haskell Chess

Playing Haskell Chess:

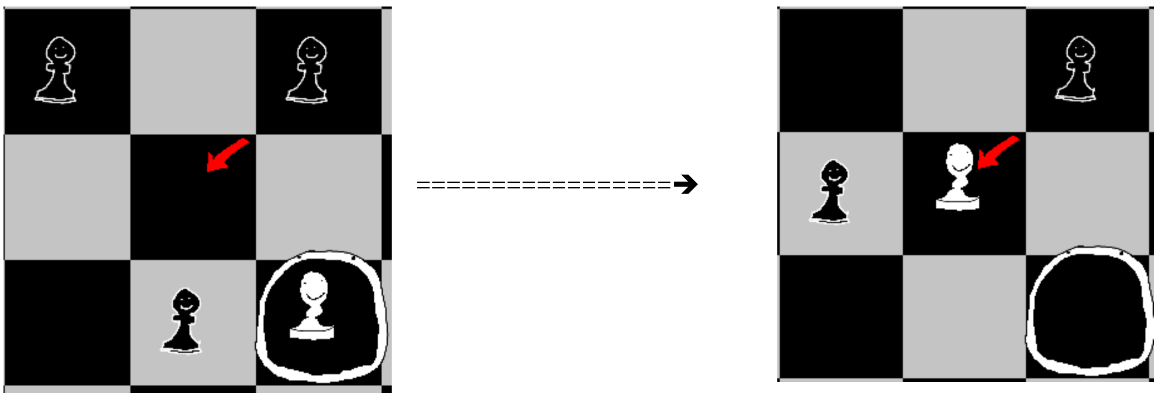
A two-player game of chess written in Haskell that allows for custom piece sprites and follows the main rules implemented by FIDE. The main way a user can interact with the program is via the use of the WASD keys and the arrow keys to move around two pointers:

The circular pointer selects the piece the user would like to move, whilst the red arrow points to where the player wants to move the piece to. The player can then press the “q” key to perform the move if it is valid. Validation checks are done to ensure that the moves made are in fact valid by comparing the piece, old position, and the new position to determine if a move made by a piece follows the rules.

Other rules of chess are implemented as follows:

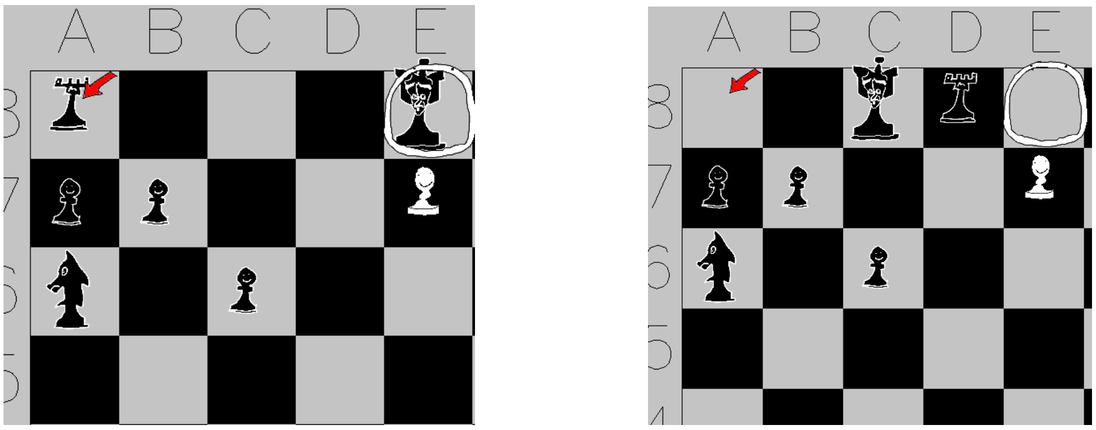
En Passant

This rule is performed when an opposing pawn that has performed a two step move forward can be taken by our pawn by capturing the space behind the two-step pawn. Example image is shown below:



Castling

Move can be performed in order to change the position of both a rook and a king granted that both pieces are in their initial state. To be performed a player must put the red pointer on the rook they would like to castle with and the circle selector on their king like so:



Pawn Promotion

Whilst pawn promotion is included in the game the promotion automatically promotes the pawn to a queen when it reaches the 7th rank of the board.

Important Notice: The program does not check for a player in check or checkmate. It is up to the players to decide whether they are in check. Although, this could be included in the future as a potential addition by checking the state of the entire board and creating a map of board positions that are currently controlled by the opposing side.

Main Libraries Used and Architectural Choices:

For the bulk of this monadic based program I relied solely on the Gloss implementation of the pure Game interface. This made integrating the game using the "play" monad significantly easier. This was because it allowed me to break down the game into sections which made it more intuitive to work on different aspects of the program at a time.

The "play" function in "Graphics.Gloss.Interface.Pure.Game" takes in a few main parameters, the most important being:

A function to define initial game state.

A function that takes a game state and outputs it as an image to the display.

A function that handles an event performed by the user.

A function that updates the game state based on said action from above.

In order to represent the game state, I used a map from the Data.Map library. This was mainly done as I could map board coordinates to each respective piece. For example, a White Pawn could be mapped with the key (1, 1). Meaning if I looked up that key within the map, I would get the White Pawn back. It was also down to convenience as the map included useful functions such as Map.lookup, Map.insert and Map.to/fromList which made world adjustments a lot simpler. Another benefit was down to the O(1) access time of using a map. This was because each piece was mapped via a unique position as defined by the PiecePosition data type which was a tuple of Integers. Fast access times allowed map adjustments to be made rather quickly. Although, this is hardly noticeable as we are only working with an 8x8 grid either way.

Personal Experience

Working with gloss and mapping elements to their correct positions using translate and scale was very tedious and required a lot of micro-management. However, I believe that the inclusion of custom sprites was well worth the work. Furthermore, gloss made outputting the game state using the "play" function a lot easier to deal with.

Working with the Data.Map library reminded me of working in the hash map library from OOP/Procedural languages like Java. It was a very simple data structure to use and made the game state update process a lot simpler to deal with.

Resources Used:

Hackage page for Gloss Pure Game library:

https://hackage.haskell.org/package/gloss-1.8.0.1/docs/Graphics-Gloss-Interface-Pure-Game.html#t%3aEvent

(This helped a lot when finding event keys)

All sprites in game were drawn in the GNU: GIMP 2.0:

https://www.gimp.org/

A guide to getting started with the Pure Game Interface in Gloss:

https://andrew.gibiansky.com/blog/haskell/haskell-gloss/

(Uses Pong as an example)