Haskell 2

Intro

For this practical we had to implement the board game Othello in Haskell, extending the design given in the specification.

Features implemented

We have fully completed the basic specification, implementing functions for making moves, drawing the world, handling input and having a working AI.

The extensions we followed are:

- Command line options for AI level, board size and start type (Random or normal) and colour of player
- An undo button to go back to your previous move
- Alternative starting position (random start)
- We have an in game options menu for displaying the current value of and changing the start type, AI level and board size (board size is changed on new game)
- We have put in bitmaps for displaying all images in the games
- We have put in the ability to save the state of a program
- We also put in multiple AI's with different strategies

Using our program

The command line arguments for our program are as follows:

- '-l' -> Large board
- '-s' -> Small board
- '-h' -> Hard AI
- '-e' -> Easy AI
- '-w' -> Start as white
- '-r' -> Random start

If you give no arguments our program starts with you as black in the centre of 8x8 board with average AI.

In game options are as follows:

- 'n' -> New game
- 'h' -> Hints
- 'u' -> Undo
- 'o' -> Options menu
- 's' -> Save current state
- 'l' -> Load saved state

If you bring up the options menu the controls are described on the screen along with the current values.

As a side note if you change the size of the board, the actual change happens when you start a new game.

Design

Main.hs

In Main.hs we load the pictures we are going to use and parse the arguments given to the function. We load the pictures in main as opposed to in draw as loading the pictures is an IO operation with is quite costly in time. Having to load all of them every 1/5th of a second would be both unnecessary and slow down the program substantially. When we parse the argument we do a string comparison between the argument and the supported input. This is potentially inefficient but we couldn't think of a better way to do it. Also as it is a one-time operation when the program starts and the number of arguments we support is limited, the difference compared to a faster method would be negligible.

Board.hs

The first changes we made to the file was adding some functions to initialise a board of the correct size. The function called depended on whether we required a random input or just a normal board

We have extended the world data constructor to contain the state of various options, a pointer to the previous world for undo and the flags to make the world when new game is called.

We then have each new type an instance of binary so that we can encode it and save the state.

We made a flags type to define the conditions of the world when a new game is made.

We have a makeWorld function to get the right board, depending on whether the user has chosen to make it random or not, and then calls the initWorld function with the board to make the correct starting world. As getting the board could possibly be an IO operation we have to return an IO world and so we separating the making of a world into two functions.

This meant we had to change the game to an IO game which involved changing the method used to play to playIO so now all my functions called by main had to return IO types.

For the makeMove method we first check that the position given is within the board and that there isn't another piece there. We then call the flipMove function. This function works by starting with the current board and Boolean value set to false. The Boolean value represents whether any of the functions called has passed. We have defined a new operator '?>', this operator calls a function of type (Board -> Maybe Board). If it passes the function, it changes the board it is maintaining to the new board and changes the value of the Boolean to true to show it passes a function. Else it keeps the old value of both the board and the Boolean. If it does pass a test we add the pieces to the new board and this is the board we return.

The move it calls is dirMove, which moves in the direction of the given operators, checking it satisfies the conditions to flip pieces and flipping as it goes. If the condition means it should keep the flips it returns a new board with the correct pieces flipped else it returns nothing.

Input.hs

In Input.hs we have 2 functions. My first function, handle input is quite extensive and serves a number of functions. The first function it serves is, if on the board screen, handles the pressing on the screen to place a piece. Next it if in the options menu, it deals with the various inputs that can be input to change the value of options, if not handles the inputs that can pressed within the game

The other function is to determine the coordinates of the board pressed from the x and y coordinates pressed on the screen.

Draw.hs

Displays the game over screen, options menu, or the board depending on whether the state of the world. There are quite a few magic numbers in these functions, we tried to deal with them where possible but for some we felt unnecessary as they were just a position on the screen.

AI.hs

In AI.hs if it is the AI's go, we call make move with a different move depending on whether the AI selected. If the AI is set to easy, we get a random move. If it is medium we get the move that will get the best value for the evaluation function 1 moves down. If it is hard it will get the move that will result in the best evaluation function 4 moves down.

The evaluation function is simply (AI's pieces–users pieces)

To evaluate 4 moves down it use a min-max search. We chose 4 moves as after 4 moves the game runs too slowly to use easily. When we call the getBestMove function we give it a depth to go down to. It goes called maxPos followed by minPos on all the available moves until depth = 0. At this point or when it reaches a position there are no moves, it returns the value of the evaluated board along with the position that would cause it to reach that position in a tuple. Returning the maximum and minimum value when appropriate. It eventually returns the move that will result in the best position in 4 moves time.

Problems and Improvements

There are a couple of improvements we could make to our AI. At the moment we can only evaluate 4 layers deep before the program starts to run slowly. This could be due to my low quality generating function. A generation function that was more specific like possibly taking the top 5 moves and evaluating those would be much more efficient.

Another improvement to the AI would be to implement a better evaluation function. Currently our program is relatively easy to beat with even basic strategies. An improved evaluation function could account for these strategies and make the AI a lot stronger. The improvement we was considering would be to raise the values of the side squares and especially the corners while decreasing the value of the squares adjacent to the corners.

If you try and undo, and there has previously been a pass, the undo won't work correctly. It currently is implemented to go two world states back, but if there has been a pass it won't go back to your previous move. If the computer passed, it will go back to the last move the computer made. If you passed it will go back to the first move by the computer before your pass. There is enough time for you to press undo twice if necessary as we have lowered the frame rate to update every 5 times a second as opposed to 10.

Another improvement would be to make an interface for the user to enter a name for a save file to create a save file with a unique name so that the user can have multiple save files. Currently you can only save one state, any subsequent saves will overwrite the current save.

Currently if you save the game, you lose all the previous moves you made so you can't undo to before the save if you reload the game. This is because each world state has a pointer to the previous world state within it, with the initial board pointing to itself. If we was to try to save the previous states it would try to recursively save itself, a never ending number of times. To allow this we could possibly change the previous world states as a list.

Work system

Basic

Board.hs
AI.hs

Extensions

Undo (in collaboration)
Alternate start
Options (in collaboration)
Save
Multiple AI’s

Evaluation

I definitely feel there was a division in work done for each respective practical between the group members. We all played to our strengths, increasingly so as the deadlines drew nearer. I mainly did the Haskell whereas Ben mainly did the Python with Sam somewhere in between. We would still talk about how each of the two half’s were doing, bouncing ideas off each other.

Overall we worked reasonable well as a group, with good communication and tolerable differences in work done.