EG1002 – Reversi Assignment

Jake Dixon (12828309)

Reversi [2] is a strategy game where two players play on an 8x8 grid board. Each player takes a turn by placing a disc on the board with their colour facing up. During the game any disc of the opponent's colour that are in a line between the ally pieces are flipped over to the ally's respective colour. The objective of the game is to have a majority of the discs displaying your colour when the game ends.

Our task for this assignment was to implement the board game into Matlab. We were required to; develop an 'umpire' that enforces the games rules, create an AI that can play against a human or another AI and design a graphical user interface to allow the game to be played graphically as opposed to text based.

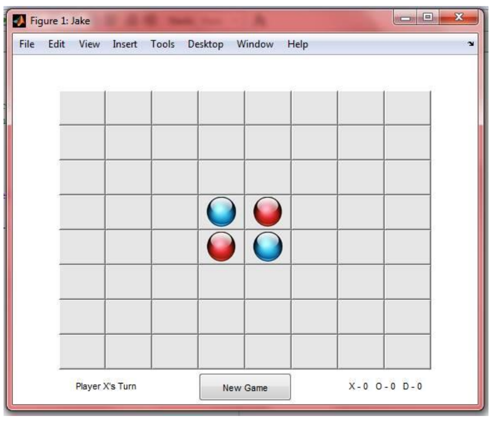
In week 7 we had worked with tic-tac-toe [1] and created an AI for it. I planned to make use of the existing code I had written and optimise it for reversi. I figured that reversi was just tic-tac-toe on an 8x8 board with a few different rules and thus I thought that it would be easy enough to adapt my current code to this. This approach however turned out to be quite annoying and in the end I abandoned this approach after the code became so incredibly disorganized that even I couldn't figure out how to get it working again. Thus in week 11 I scrapped this method and decided to start from ground 0. In this 2nd approach I decided I would write the code from the ground up but design it in a way that the functions I used resembled tic-tac-toe [1] in that they used the same input and outputs. I figured that this would make it easier to convert the text based version of the game into a GUI as I would be able to optimise the solution to week 10's lab task that was creating a GUI for tic-tac-toe [1]. I planned it out that I would only borrow minimal amounts of code such as setting up the board and displaying the text based version of the board from tic-tac-toe and write the whole game loop from scratch.

This approach turned out to be much easier and allowed me to organise my code better as I wasn't just placing lines of code in existing lines of code and cluttering it up. eventually in one weekend and roughly ten hours of work I had a fully working text based version of reversi with an AI that passed all 200 of the bot test that were placed on learn JCU. My GUI was next. The planning of my GUI started with one phrase 'Clean and Easy' [3] a design philosophy used by many GUI designers today such as those who work for apple or Microsoft. The phrase essentially describes that a GUI should be invisible to the user doesn't get in the way. I didn't want fancy colours and scrolling text with millions of buttons on one screen obscuring the beautiful background [4]. I wanted something that only showed you what you needed and only had the buttons you would use regularly, everything else would be placed elsewhere. This was a key element in how I shaped my GUI and how it ended up in my final version.

My umpire was designed to control the game. The umpire 's main operations occur in the 'buttonpress' call back and it allows me to make sure no illegal moves are made and also controls how the AI behaves (whether it is on or off). It is also in charge of score keeping and monitoring the output text. The basic design started out as an oversized tic-tac-toe and evolved from there with more checking and more if loops to make sure no illegal moves or bugs occurred. It once had trouble finding out if the game ended and thus was re-written once from the ground up on the 26th. The 'drawScreen' function is another function that ties in close to the umpire function as it allows for hints to be displayed as well as printing the player's pieces. My umpire relies strongly on the 'move_is_valid' function which requires an input of the board, the move and the player number and outputs if it is valid, a vector containing the number of tiles it can flip in each direction and a board (the board can be ignored by using the '~' symbol). It finds a valid move by analysing tiles in all 8 possible directions for another one of the player's pieces. If there is a possible move it sets the first output to '1' then it will place how many tiles between these tiles in the places matrix '[U D L R UR UL DR DL]' and then it calls the flipper function which takes in this places matrix and flips all those directions and then the move_is_valid function ends. I found this method in determining valid moves to be easy to manage and allows this one function to be able to do a lot of things such as calculate where the possible moves are for the hint function. This function also made the development of my AI much easier.

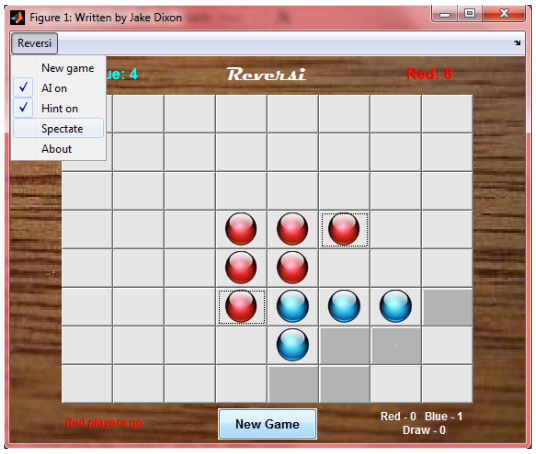
My AI was designed to find the maximum number of tiles it could flip in a single move and then put a move there. The move_is_valid function was copied into the AI as a sub function used to find where the valid moves were then store the places matrix and document which tile it corresponded to. The main function would then find the sum of each places matrix and out of all the sums it would find the largest, take it and find out which tile it was. It would then take that tile and set it as the move and output a two element vector to the umpire as its final move. If there are two max moves that can flip the same amount of tiles it randomly selects between them. I found this strategy to be the easiest way to implement a 'decent' AI. While it may not be the smartest AI as I can still beat it occasionally it will still put up a fight. The AI itself is also extremely fast and thus I saw no point in implementing a time limit as it can calculate its move in less than half a second, which is faster, way faster than I could ever make a move. The whole AI was started and completed on the 19th of May and passed 200/200 bot tests on its first try. The AI may not play a perfect game, but then I didn't program it to play a perfect game, I want it to be human like and make mistakes occasionally as I'm sure I can't play a perfect game.

My GUI early on was very similar to the week 10 solutions [1] as those screen casts helped a lot in getting the basic function of the GUI to work. I took the screencasts and optimised everything for an 8x8 board and once I got the basic functions working, mind you with lots of bugs, it started to become its own thing. One of the first things I did was replacing the 'X' and 'O' with pretty red and blue buttons, the red and blue colours were chosen as a reference to the forgotten Reversi for windows program bundled with Windows 1.01. I changed the colour of the buttons to a more windows colour of grey and made the buttons look pretty too. After I had gotten the buttons to work properly and



1. Original Design

the game to run without many bugs I worked on making the text boxes pretty and more reversi like instead of tic-tac-toe [1]. I make the text bigger and for the status textbox made the text change colour depending on whose go it was. I also added tile counters up the top and at once stage had my hint button on the board, this however made the board look cluttered and went against my design philosophy and so was move. I added a pretty title to the board in a retro font that made it look classy I think and added a beautiful wood grain background to the mix. This however presented problems as the textboxes weren't transparent. After about an hour I found the 'text' function and replaced all my text with that function because it supported transparency. I then added my own menu bar which is where I put all my options such as my hit button, AI toggle and spectator mode. The menu bar allowed me to make more options for the game without cluttering the main board which was great. In my hint function I decided against layering a picture over the tiles for hints as this could come across as confusing, instead I opted for a slightly less annoying approach and opted to just change the colour shade of the hint buttons to a darker shade to make them stand out, I experimented with colours but it ended up making the whole game look childish and un-professional, for example when have you ever seen an orange button in windows? Overall I'm really happy with the way my GUI turned out. In my opinion my design goal was reached and I think that the design looks much cleaner than some of the designs I've seen with more than one figure window open all the time storing extra buttons, not saying I don't like there designs it's just that it doesn't fit my design philosophy . If I had one problem with my GUI it would be that the scoreboard down the bottom is not much to look at. Lastly a special 'feature' was suggested by nick Stewart… try clicking the top left button 20 times for a nice surprise… you can also select a background music file by pressing the top left button and then the bottom right button this is also a hidden feature. please note that if the game loads slowly clear your workspace, for some reason an uncleared workspace can slow this down.



2. Near Final Design

The hint function is an extra feature I added to aid people who are only just learning to play. It works by simply shading the buttons can would cause a valid move to a darker shade of grey. Under the bonnet though, the move_is_valid function is used to find these buttons. A temporary board is made of zeros and each valid position is marked with a 1. The main if statement in the drawScreen function then interprets the 1's in the temporary board as places where the button shading needs to be changed. The hint function works with 2 players as well if you turn off the AI in the menu. So it is not strictly limited to player 1.

My spectator function allows me to pit my AI up against another AI via a file selection. The selection process is graphical and I borrowed 3 or 4 lines from the AI bot test function from learn JCU to make sure that the name of the bot complies with the specifications in the assignment task. If this didn't match it displays an error dialogue and then cancels the operation of selecting a bot and returns to 1 player. If a valid bot is selected the two bots will battle out until the game ends, at which time you can only select new game which will return the game back to 1 player.

My high score function allows you to view the highest score and clear the score if you want. A high score is achieved if you win and the number of tiles you have on the board is greater than the previous high score. You enter your name and then it saves the file as highscore.mat. Lastly my background music [5] function allows you to play the stop bkg1.wav file or you could replace it. This was just an idea and is not playing by default because I couldn't find music that suited the game the hidden wav selector was hidden for the same reason I was just testing the wav player, I thought it was cool so I kept it there, but hidden.

Bibliography:

[1] EG1002 2013 YouTube, Solution to week 10 lab, available at: http://www.youtube.com/playlist?list=PLlOkj3KHnhWIIJAH4R9r9ljceFy_bNicF

[2] Reversi Wikipedia page, available at: http://en.wikipedia.org/wiki/Reversi

[3] Kyle Sollenberger aug 7, 2012, 10 user interface design fundamentals, available at: http://blog.teamtreehouse.com/10-user-interface-design-fundamentalshttp://blog.teamtreehouse.com/10-user-interface-design-fundamentals

[4] Mathworks solution center, feb 5, 2013, 'adding backgrounds to GUI windows, available at http://www.mathworks.com.au/support/solutions/en/data/1-19J7T/

[5] Peter Bezemer, 12 mar, 2010, 'strategy game music demo' available at: https://www.youtube.com/watch?v=Rf3SxV4x7KA