Jethjera Silasant
Crystal Chongphakdee
Jeff Huynh

MAIN PROJECT - REVERSI

Information on data segment...

- An 8x8 array will be used to store and access the values of 0, 1, and 2.

(One dimensional array of size 64 that stores a word per index)

Values in the array and their corresponding piece in the bitmap:

1 – White piece

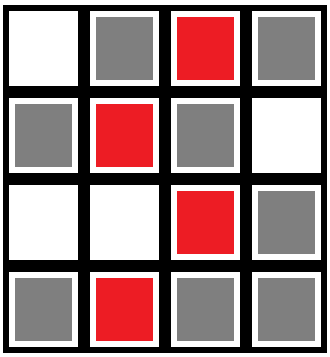
2 – Black piece

0 – Empty

The array will be initialized to have 2 pieces each for the player and user for the starting point of the game.

- The number of pieces for both the player and the A.I. to be updated throughout the game to determine the victor at the end of the game.
- Player input (a move/placement on the board) to be validated and used as an argument to determine whether the placement was a valid one.

Displaying the board on MARS using BITMAP tool...



Note: (Only a representation of the cells, not actual reversi)

Using the bitmap tool included in MARS, this will be our pixel representation of the board, having an individual cell be 8x8 pixels, 1 pixel width boarder and spacer, and 4x4 internal token; replicate this individual token for the remaining cells of the 8x8 board. The dimension of the bitmap board will be 512x512 pixels.

Display: [512x512]pixel:[1x1]base address: $gp . This will most likely be the setup for the bitmap display.

Sounds to express victory or loss as well as background music is in the works. The sound will be one of the hardest parts about this project because we would have to create a midi file and then convert

it to binary to be read by MIPS. We have an example on how to do it, but we would also have to create our own extension for the conversion to binary.

For accessing the array (horizontal, vertical, and diagonal movement) ...

Index 0 is the beginning of the array, and every 8 elements is a row of the board. To access the elements directly above or below it, +/- 8 words. To access the elements to the left or right, +/- 1 word. To access the elements diagonal, +/- 7 or 9 depending on which diagonal (\ or /).

To detect the edge of the board, check with x mod 8; if remainder is 0, then the checking functions terminate since the search would travel past a horizontal boundary. Any index < 0 or >= 64 is out of bounds vertically (I.e escapes the array size).

Validating a move and flipping...

Validating a move will be done by traveling along the 8 directions from the position specified by the user. In each direction in which all pieces traversed are the opposite piece or value, and the end of the traversal ends on a corresponding piece, all pieces traversed will be flipped. In the case that no direction was found to have those conditions, a message will be displayed indicating to the user that the move was invalid and asks to provide another move until a valid one is provided.

A separate array will be used to store the index or offset of all pieces traversed and will serve as a way to access and change the values in our board. Updates will be done to each number of pieces for the player and the AI. Once the array has been updated, the corresponding values will be outputted to the bitmap by writing the corresponding pixel values to that position.

Computer AI...

The A.I. in our program will run a greedy algorithm to choose a position which would provide the most pieces to be flipped. Different ways of making the A.I. a more formidable opponent is being discussed. A greedy algorithm as well as prioritizing edge/corner spaces is in consideration.

Calculating end results and restarting game...

When the array has reached full capacity, the program will compare the number of pieces for both the player and A.I. There will be a win message or loss message depending on whether the player has more pieces. Afterwards, a prompt will ask the user if they would like to play again or end the game. In the case that the player would like to continue playing, all registers will be cleared and the program will loop back to the beginning until user commands otherwise.