Zeby Poycattle

Zkp170030

CS3340.501

Team Ex-Caliber

Reversi Project Report

1. The program is a MIPS Assembly Language implementation of the Reversi board game. The program displays an 8x8 game board with row and column labels utilizing the MARS Bitmap Display tool. The program places the initial 4 pieces in the center of the board and then prompts the user to input where they would like to place a game piece via the keyboard. The program determines whether a syntactically valid input was made, and then if the input results in a valid game move. If the input is syntactically invalid or an illegal game move, the user is told what the error was and prompted again for an input. After a valid move is input by the user, the game board is updated to reflect the result of the move. The computer then places a piece by determining what move will result in the greatest gain during that turn. The program updates the game board and prints out the last user and computer move after each computer move. The program alternates between user and computer turns, skipping turns when a valid move is not available and printing the game score and winner once the game ends.
2. We had several challenges arise during the project, but were able to overcome them as a team. One of our team members was able to code much of the logic very early in working on the project, before we learned about register convention. Thus, we had written almost all our subroutines without the implementation of register convention. To fix this, we decided to work through all the subroutines together, determining which registers needed to be saved and adjusting the subroutines so that the implementation of register convention was correct. We decided to work on this together in the same place, opposed to independently, because we thought it would be easier to see when registers would need to be saved and talk through the appropriate course of action. While fixing the register convention, we had issues arise in which our previously working game logic failed. To fix this, we tried many test cases and noted when errors would arise. Together, we were able to debug our code to ensure that the game was working correctly for all cases. Our last major challenge was determining a way to include row and column labels on our game board. This was much more difficult than drawing the board and game pieces as each letter/number is unique. We researched ways to implement this in MIPS and settled on utilizing .bmp files. We had to calculate new dimensions for the rows, columns, and game pieces to account for the additional row and column containing the labels. Together we solved a system of linear equations to determine the required dimensions for the 512x512 pixel bitmap.
3. Doing this project, I learned how to think more like the computer. Programming in assembly language required me to think very methodically and sequentially. I had to take the objective I was trying to achieve and break it down into steps that the computer understood, instead of higher level problem solving concepts. I also learned how to work on a large development project as a team. I am a graduate student transitioning into Computer Science from another field, so I have never had to work on a project like this. I was able to utilize my project management experience in this framework to keep our project on track.
4. To draw the game board, first the row and column labels are printed. Then, each square of the game board is drawn using orange pixels, as well as black pixels to split squares. Black and white pixels are used to draw circular game pieces. For a user turn, the program first uses the algorithm designed for a computer turn to see if any valid move exists. If the maximum calculated profit is returned 0, then no valid move exists and the program switches turns to the computer. If it returns something other than 0, a valid move exists and the user is prompted for an input. The input provided by the user is determined to by syntactically correct. The input can be case-insensitive. To determine whether the input is syntactically correct, the first character is determined to be within the ascii values of a-h or A-H and the second character is determined to be between 1-8. If not, the user is prompted again. To determine if a piece is valid to play, the designated square is found in an array of all the squares on the board, with occupied pieces having a 0 or 1 value for the color of the piece. If the square is already occupied in the array, the user is prompted to input a new square because the square is already occupied. The algorithm counts in each of the 8 directions of the board, square by square, advancing if a piece of the other color occupies that square. If it runs into an occupied square of the same color, it stops and proceeds in the next direction, and stores the pieces that will change color in another array that is used to update the board after the turn. If when counting in a direction the piece advances to the end of the board without finding a same colored square, or encounters an empty square, no gain is calculated in that direction. If in every direction there is no gain, the user is prompted again to pick a new square because the move is invalid. For a computer turn, the same algorithm is used, but the computer will choose the best available move by storing the maximum calculated profit for each available square. Before a user input or computer selection, if the maximum calculated profit of all available moves is 0, the program switches turns to the other player. The user move and computer move are stored in variables and printed after the computer’s turn to maintain a game history. The program checks to see if all the squares in the array are occupied or if both the user and computer return 0 as maximum profit to determine if the game should end. To print the score, the 0 or 1 values in the pieces array are summed corresponding to the user and computer. The values are compared to determined a winner.
5. We worked very well together as a team. We held weekly meetings beginning in October and did much of the development together.
   1. Jiashuai Lu- Jiashuai was our lead developer. He coded most of the logic as soon as the project was assigned, even before our first meeting. We worked together to adapt it to meet the project requirements. He also was responsible for the code to display the gameboard with row/column labels using the Bitmap Display.
   2. Hector Martinez- Hector was responsible for the user input module and the row and column labels of the bitmap display.
   3. Khoa Ho- Khoa was responsible for the various error messages when invalid game pieces were played, as well as the move history displaying the user and computer previous turns.
   4. Zeby Poycattle- I was responsible for beginning/end of game messages and logic for switching players when no moves remained. I also wrote the game manual and communicated to organize meetings.

Aside from these individual projects, we worked on adjusting our original code to utilize the appropriate register convention together. We ran into several logic issues after changing the register convention, which we debugged together. Lastly, we worked together to determine how to include labels in our game board. I am very pleased to say we all worked together extremely well and pulled our own weights. The project would not be as successful as it was were it not for the commitment of all the team members. Everyone attended every meeting, communicated effectively through group message, and promptly made changes on GitHub so that all members could complete their work appropriately.