“Free Folks”

Final MIPS Project

Devon Hayworth, Max Chen, Shreyas Ganesh, Rohan Arya

Project Report

1. **A description of the program:**

This program is an implementation of Reversi (aka Othello) in MIPS. Reversi is a two-player game where the objective is to have more pieces on the board than the opponent when the game ends. The board is an 8x8 board with four pieces set in the middle in a diagonal pattern.

The rules of the game are as follows:

1. The ‘X’ player (dark player) must move first.
2. X must place a piece in a position that there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another X piece, with one or more subsequent O pieces between them.
3. After successfully placing a piece, the X player changes the O piece in-between the new piece and another X piece into an X piece.
4. A valid move is one that reverses a piece.
5. Players take alternate turns. If one player cannot make a valid move, the turn is passed back to the other player.
6. If no player can make a valid move, the game ends. (This means the game can end without the board being filled)
7. The player with the most pieces on the board wins.

In our version of the game, the second player (‘O’ Player) is the computer, and we have developed an algorithm that allows the computer to make valid moves based on the moves of the player. The algorithm will be discussed in more detail in part d.

The way we presented the game was through printing an ASCII text board. To make a move, the program prompts the user to enter the desired cell coordinate. Throughout the game the number of X and O pieces are updated along with the board after each turn. When an invalid move is entered, the program prints a descriptive error message and allows the user to enter another move. When there are no valid moves left on the board, the game ends and prints out the winner (whoever has more pieces).

1. **Project Challenges:**

One of the largest challenges that we faced as a team was how to match everyone group member’s availability. However, after we downloaded the group messaging app, “GroupMe”, we were able to have meetings which included most of the team members.

Another challenge was to understand each other’s programming styles including how they liked to use formatting or commenting. By working alongside each other and giving and receiving feedback, we were able to communicate to each other our own styles and how our program was written and functions.

Personally, a challenging part of the project was how I went about converting our template C++ algorithm code into MIPS assembly language. I was able to obtain help from working side by side with my team members when I encountered a problem.

1. **What you have learned by doing the project:**

For this project I learned many things about how to go about planning a comprehensive project. In beginning of the semester, we organized a meet up discussing our approach to completing the project. We all listed and contributed to creating a base skeleton outline with required features for the product to have. This outline was very helpful for us to keep track of the main goal and objectives.

As discussed previously, when I did my part of converting the C++ algorithm into assembly, I learned how to implement the different logic and loop code structures of C++ into MIPS. The concept of subroutines and the call stack were also necessary to understand and learn for me to implement the C++ algorithm into MIPS.

1. **A discussion of algorithms and techniques used in the program:**

The following abstraction summarizes our implementation of the game:

Starting with our main:

* The board is implemented with ascii text. Its stored in 64 bytes of memory.
* At the start of the program, every byte on the board is initialized to ‘ ‘, then the initial ‘x’s and ‘o’s are placed on the board.

Now we begin the game:

* The game starts with the player turn
* It then alternates between computer and player turns
* If a player can’t play, the other player gets another move
* Between the turns, it checks if there are any more valid moves
* Ends the game if there are not
* Prints out the result of the match

Printing board:

* The column numbers are displayed on the top of the game board
* The row letters are displayed on the left of the game board
* Horizontal lines are placed in between each row
* The board piece characters from our array are placed in each row
* Vertical lines are placed in between each board piece character
* The number of pieces each player owns is printed below the board

Player’s turn:

* First, it checks if the player has a valid move available to them, if not it skips the player's turn
* Then it reads the user input for the player’s move
* It places the piece on the board and flips any potential pieces
* Then it updates and reprints the ascii board

Computer’s turn (computer AI):

* The **best move** is defined as a valid move that flips the largest number of pieces
* The computer prioritizes playing along the edge of the board as it makes it harder to flip the computer’s pieces
* The computer checks all possible moves on the edge of the board
* If there are valid edge moves, it selects the best edge move.
* If there are no valid edge moves, it selects the best move in the interior of the board.
* If there are no valid interior moves, the computer’s turn is skipped
* When a best move has been found
  + The program places the piece on the board and flips any potential pieces
  + The program then updates and reprint the ascii board

1. **Peer Evaluation:**

Devon Hayworth:

* Designed the C++ algorithm and engine for our Reversi game
  + Developed a large part of the computer AI
* Did a large part of the debugging for the algorithms
* Was responsible for combining Max’s and Rohan’s assembly language functions into a complete program.
  + The subroutines that he implemented in MIPS were:
    - printBoard();
    - placePlayerPiece(int);
    - placeComputerPiece(int);
    - playerMove();
    - computerMove();
    - playerValidMoveExists();
    - isPlayerMoveValid(int);
    - readMove();
    - convertToBoardLocation(string);
    - getComputerMove();
    - getComputerEdgeMove();
    - getComputerInsideMove();

Max Chen:

* Helped Devon develop the move prioritization of the computer AI
* Was responsible for converting the player check and flip subroutines from C++ algorithm into MIPS
  + The subroutines that I implemented in MIPS were:
    - playerFlipUp(int);
    - playerFlipRight(int);
    - playerFlipDown(int);
    - playerFlipLeft(int);
    - playerFlipUpLeft(int);
    - playerFlipUpRight(int);
    - playerFlipDownRight(int);
    - playerFlipDownLeft(int);
    - playerCheckUp(int);
    - playerCheckRight(int);
    - playerCheckDown(int);
    - playerCheckLeft(int);
    - playerCheckUpLeft(int);
    - playerCheckUpRight(int);
    - playerCheckDownRight(int);
    - playerCheckDownLeft(int);

Shreyas Ganesh:

* Helped write the introduction of the project report
* Created the short video that describes our program in action
* Created a user manual on how to run and use the program
* Helped in playtesting for edge cases, and finding bugs in game

Rohan Arya:

* Was responsible for converting the computer check and flip subroutines from C++ algorithm into MIPS
  + The subroutines that he implemented in MIPS were:
    - computerFlipUp(int);
    - computerFlipRight(int);
    - computerFlipDown(int);
    - computerFlipLeft(int);
    - computerFlipUpLeft(int);
    - computerFlipUpRight(int);
    - computerFlipDownRight(int);
    - computerFlipDownLeft(int);
    - getComputerNumFlips(int);
    - getComputerNumFlipsUp(int);
    - getComputerNumFlipsRight(int);
    - getComputerNumFlipsDown(int);
    - getComputerNumFlipsLeft(int);
    - getComputerNumFlipsUpLeft(int);
    - getComputerNumFlipsUpRight(int);
    - getComputerNumFlipsDownRight(int);
    - getComputerNumFlipsDownLeft(int);