REVERSI GAME IMPLEMENTATION USING MIPS ASSEMBLY LANGUAGE

PROJECT REPORT

Prepared for:

Nhut Nguyen, Ph. D. Department of Computer Science,

The University of Texas at Dallas

Prepared by:

Lan Vu

Tung Duc Vu

Keyur Savjani

Roman Chernov

DATE

November 30, 2017

Table of Contents

Page

List of illustrations iii

Abstract 1

I. Description of the program 1

II. Additional features 4

A. Graphical game board 4

B. winning strategy and difficulty 5

C. sounds 6

D. human vs. human mode 6

E. timer, players name, and statistics 7

F. prompt to play another game 7

III. The challenges 8

A. logical design 8

B. board layout 9

C. graphics 10

D. sounds 10

IV. lessons learned 10

V. algorithms and techniques used 10

VI. peer evaluation 10

VII. Suggestions 11

VIII. Conclusion 11

List of illustrations

Page

Figures

Figure 1. Multiplayer selection 1

Figure 2. Player’s name 1

Figure 3. Initial disk positions 2

Figure 4. Move combination 2

Figure 5. Invalid move 3

Figure 6. Game termination 3

Figure 7. Board drawing process 4

Figure 8. “Cost” of the move 5

Figure 9. Sound generation 6

Figure 10. Two players option 6

Figure 11. Statistical information 7

Figure 12. Another game 7

Figure 13. Game logic diagram 8

Figure 14. Board layout 9

Abstract

This Reversi game is a fully functional computer game developed to run utilizing MARS engine. The code for the game was written by Rembler group in MIPS assembly language. The game complies with minimum requirements outlined by the assignment specifications and has many additional features. This report describes the challenges our group had encountered during the development process as well as the lessons we have learned trying to overcome said challenges. This report is also covering algorithms and techniques our group utilized in the process of developing this game.

I. Description of the program

At the beginning of the game, the program prompts the player to make a selection for the game type. There are a human versus a computer and human versus human types of the game available. See Fig.1.

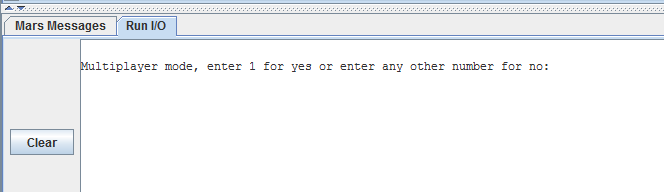


Fig. 1 Multiplayer selection.

Black disk always starts the game. Next, the program asks a player or players to enter their names. The name(s) will be used for messages during the game and for statistical information at the end of the game. See Fig.2.

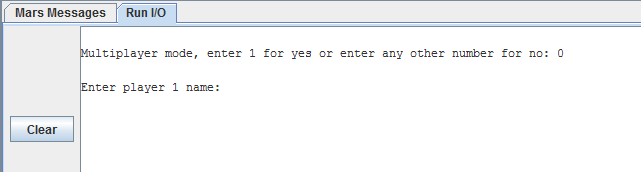


Fig. 2 Player’s name.

As the next step, the program performs a reset procedure setting up all the initial values for all parameters and makes a time stamp for the game start.

The game board is drawn next where player(s) can see an initial disk positions. See Fig.3. Black disk start the game.

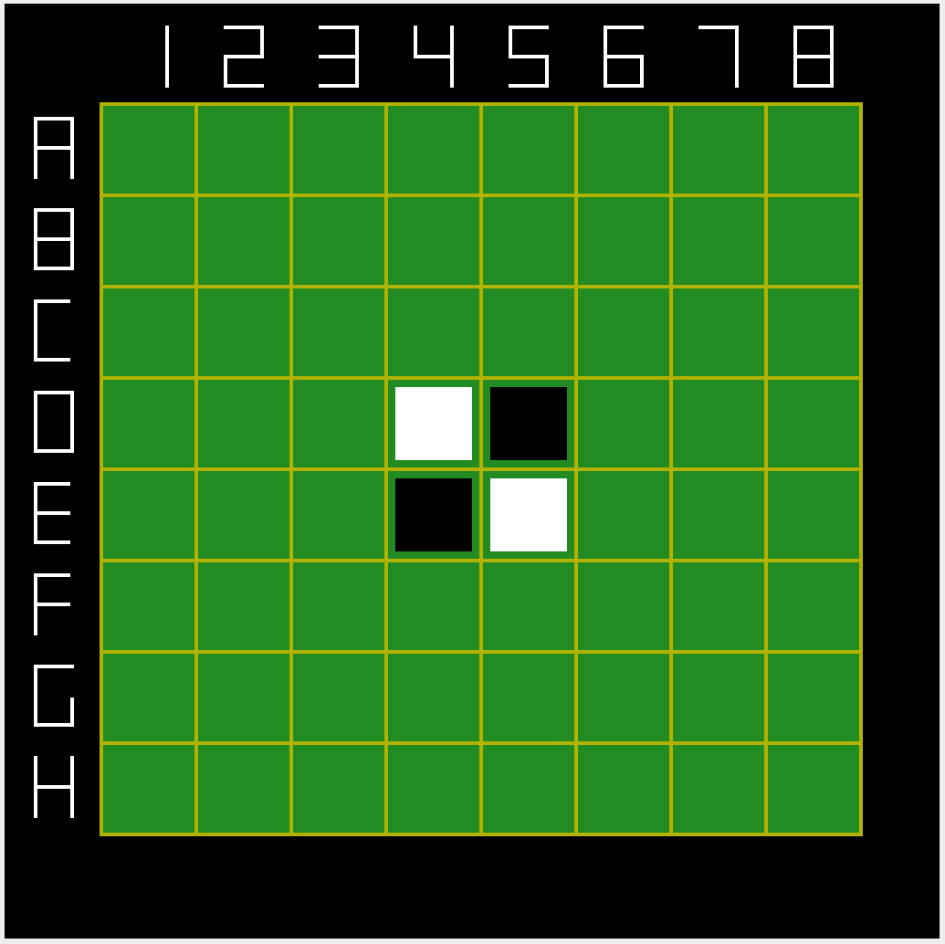


Fig. 3 Initial disk positions.

MIPS has no interactive graphic capability, so even though the players can see the board, they must use the program console to enter their next move. The location of the move on the board should be specified in letter plus number combination (ex. A5). See Fig.4. The letters are not case sensitive.

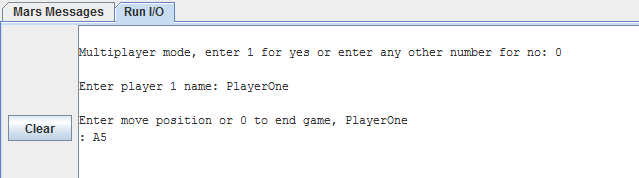


Fig. 4 Move combination.

After the desired move has been entered by a player, the program checks if that move is possible and the check has one of four outcomes. If the move is not possible, the player will be shown the message that the move cannot be performed along with the explanation why.

If the move is possible, the game will continue through. If there are no moves left for this color, the game will continue for the opposite color, and the game will end if both colors have no possible moves left.

After the move has been determined possible, the next stage is to check that selected move is valid according to the game rules. If the move is not valid, the player will be shown a message accordingly specifying what rule of the game has been violated and will be prompted to enter a new move again. See Fig.5.

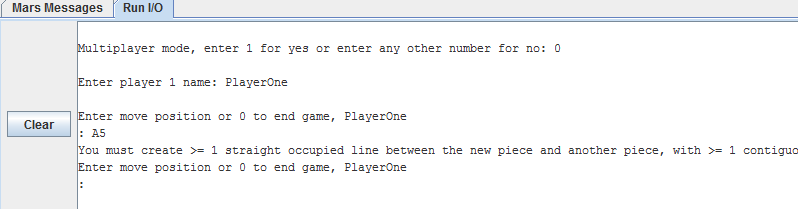


Fig. 5 Invalid move.

During computer’s turn, computer needs to decide on its own move, so computer produces an array of possible moves, checks their validity and goes through additional stages where those moves are analyzed for their “cost” and the best moves are strategically picked from the array. After that, a random algorithm is applied to make a final selection for the computer’s move.

For the next step, the program updates an array with a new move and redraws the board to reflect the change. After that, the turn is changed to the opposite color and the game continues for the opponent.

Entering zero at any moment terminates the current game and prompts the player to either play another round or quit the program completely. See Fig.6.

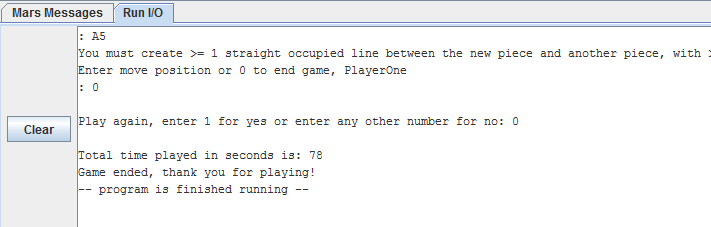


Fig. 6 Game termination.

II. Additional features

While the game conforms to minimum requirements outlined by the assignment specifications provided to us, our group decided to implement additional features into the game to enhance player’s experience and to develop our programming skills further.

A. Graphical game board

Instead of an ASCII board representation, our program draws a player-friendly graphical eight-by-eight game board with numbers on the top and letters on the left. Each move performed by a player must correspond to a letter plus a number combination according to the board drawing. Game disks are represented by white and black squares and the game board is redrawn after each move to reflect the changes.

Since MIPS has no graphical capabilities, the game board elements are drawn pixel by pixel. See Fig.7. This process involves a lot of mathematical calculations, careful loop orchestration, and a great deal of patience.

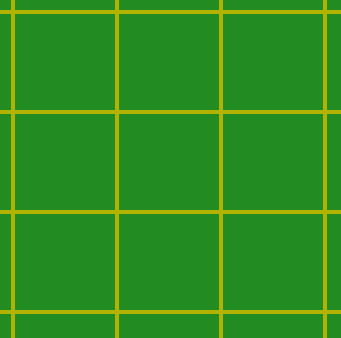


Fig. 7 Board drawing process.

B. winning strategy and difficulty

Our board logic design allowed our group to assign a numeric values to each of the play positions on the board. We called those values a “cost” of the move. See Fig.8. The higher the “cost” value, the less desirable move it is for a computer. The “cost” values are assigned strategically throughout the board utilizing a zone concept. When a computer needs to decide on its own move, it produces an array of possible moves, checks their validity and goes through additional stages to analyze the moves for their “cost.” The best moves are strategically picked from the array. After that, a random algorithm is applied to make a final selection for the computer’s move.

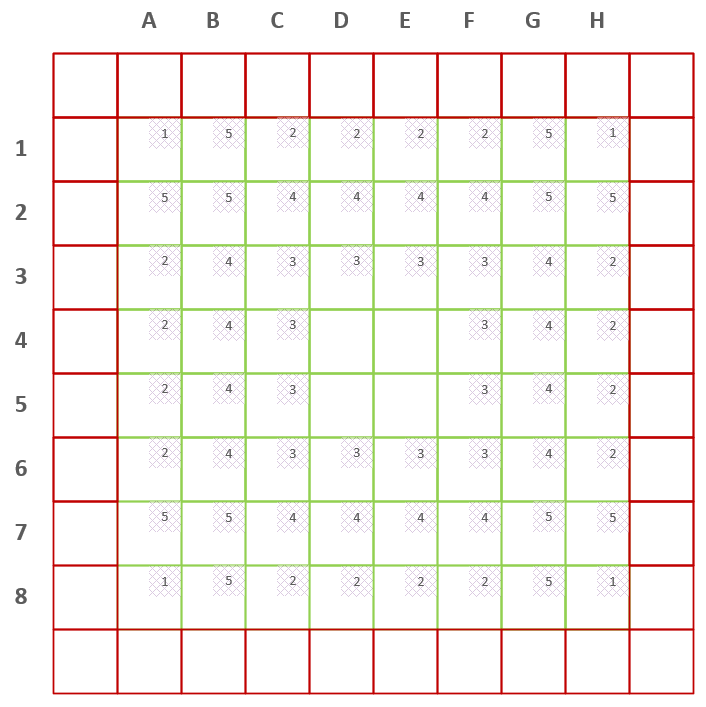


Fig. 8 “Cost” of the move.

C. sounds

As an additional program feature, our group decided to utilize sounds to enhance player’s experience. We have developed five different tunes: game start, correct move, incorrect move, game over with a win, and game over with a loss. All of those tunes are a combinations of individual sounds produced sequentially with a different pitch, duration, instrument, and volume programmatically. See Fig.9

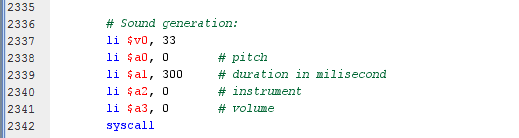


Fig. 9 Sound generation.

D. human vs. human mode

We implemented an option at the beginning of the game where a player can choose whether he or she would like to play against a computer or with another player. If a player chooses to play a game with another player, the players must take their turns. See Fig.10. In this instance, the logic for computer to pick a next best move is omitted.

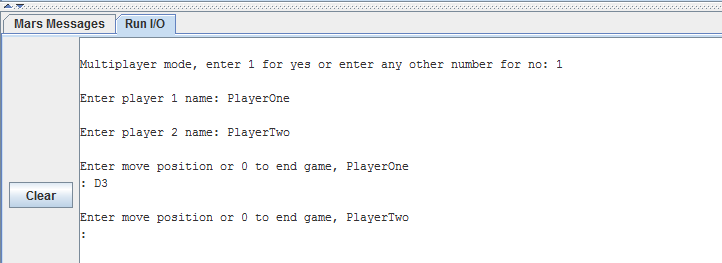


Fig. 10 Two players option.

E. timer, players name, and statistics

At the start of the game, computer takes a note of the current system time. Upon the completion of the game, computer calculates the difference between the current system time and the start time to find out the duration of the game. Also at the end of the game, the board array gets analyzed for how many black and how many white figurines are present on the board. This will become a game score. All of this information is put together and displayed to the player upon the completion of the game. See Fig. 11 for an example.

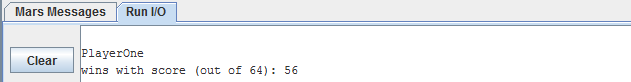


Fig. 11 Statistical information.

F. prompt to play another game

Lastly, as an additional feature, we implemented the message at the end of a game that prompts the player to enter “1” to play another round or any other number to terminate the program. This is a convenience feature – this way the player does not need to re-setup the graphical settings and re-assemble the game in order to play another round. See Fig. 12.

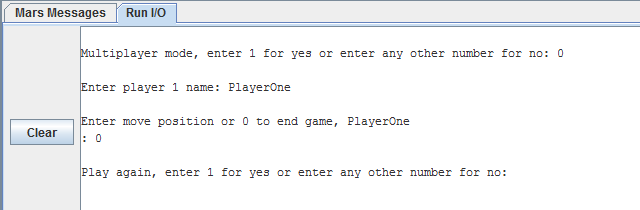


Fig. 12 Another game.

III. The challenges

During the design and the development stages of the game, our team encountered a few challenges. We met every Sunday for the duration of the project and we believe that constant communication and collaboration among all team members were vital to the success of this project.

A. logical design

It took us multiple weeks to come up with a good logical design of the game, as we believed that it would be a foundation for success and will minimize the number of bugs in the code. We wanted to simplify the logic while preserving the integrity and complexity of the game and make the design as efficient as possible. At the end we came up with what we feel is the best design and we are very proud of that. See Fig. 13.

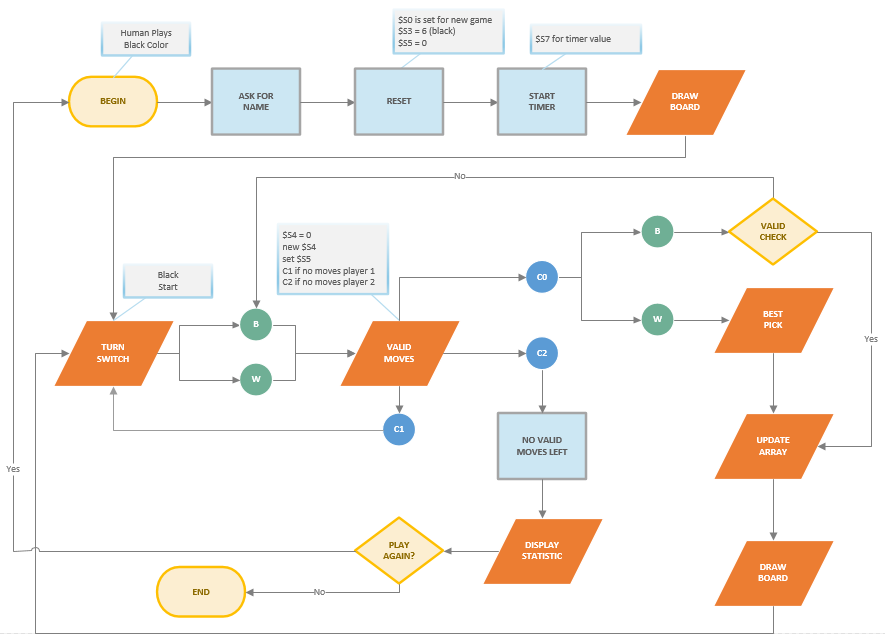


Fig. 13 Game logic diagram.

B. board layout

Our logical programming implementation design implied that we had a specific board layout developed for that purpose. This was the next challenge our group had encountered. Obviously, the game board for the player remained to be eight by eight pieces with the letters and numbers along sides. However, for a computer to interact with the board, the board had to be designed differently. We developed a ten by ten board, which contains a regular eight by eight playing field alongside with an extra row or column on each side. This design allowed a computer to understand where the edge of the normal game board is and helped it to make better decisions about the next move. See Fig. 14.

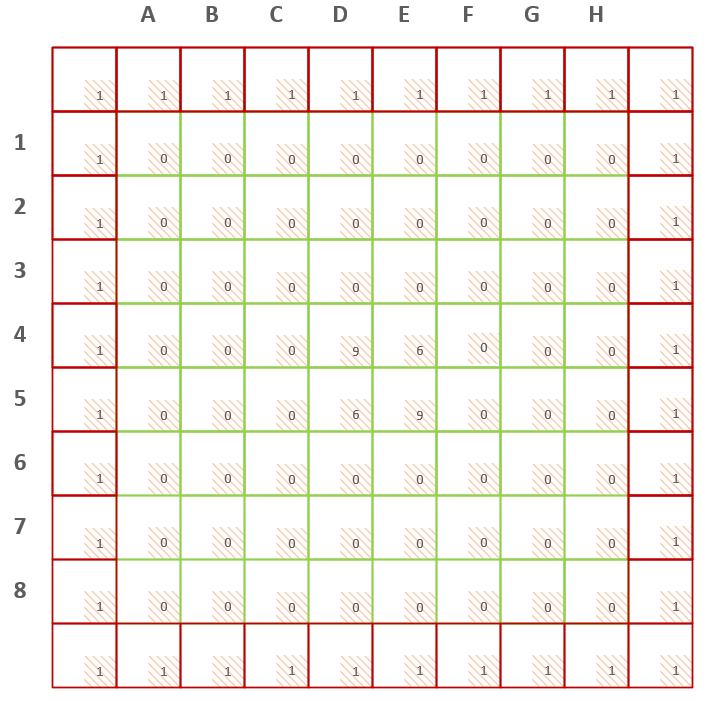


Fig. 14 Board layout.

C. graphics

Another monumental challenge was to make a graphical representation of the game board. Since MIPS has no graphical capability, the board has to be drawn pixel by pixel. This presented a challenge because the whole board had to be depicted as a pixel map, so the values for every line, every square, every letter, every number, and the game pieces had to be mathematically derived for the game board to look correctly. Implementing this style of graphics was very tedious and required a lot of attention to detail.

D. sounds

Additional challenge came with sound implementation. We spent a considerable amount of time first trying to figure out how to produce a sound using a code and then how to make a tune by putting multiple different sounds together. We came up with sounds for the game start, correct move, incorrect move, gave over with a win, and game over with a loss.

IV. lessons learned

This section to be completed individually.

V. algorithms and techniques used

This section to be completed individually.

VI. peer evaluation

This section carefully evaluates the performance of each member of Rembler group, excluding myself, over the period of the group project.

5 – Outstanding 4 – Good 3 – Satisfactory 2 – Poor 1 – Unacceptable

|  |  |  |  |
| --- | --- | --- | --- |
| Name: |  |  |  |
| Did his/her fair share of the work that was required. |  |  |  |
| Cooperated with other group members, agreed on task assignment. |  |  |  |
| Contributed to ideas and planning. |  |  |  |
| Was available for communication. |  |  |  |
| Shared responsibilities and did not try to take charge inappropriately. |  |  |  |
| Completed his share of the work on schedule. |  |  |  |
| Always submitted his best  effort. |  |  |  |
| Communicated thoughts and feelings effectively. |  |  |  |
| Was always well prepared for group meetings. |  |  |  |
| Participated in, and contributed to, all relevant discussions. |  |  |  |
| Attended group meetings when required to do so. |  |  |  |
| I would choose this person to be in the same group with me in the future. |  |  |  |
| Contributed to overall project success. |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| The average for this person (1 to 5): |  |  |  |

(Round average for each group member to two decimal places, e.g. 4.25)

VII. Suggestions

Let’s discuss this on Tuesday meeting.

VIII. Conclusion

The programmatic design and implementation of Reversi game in MIPS assembly language discovers the capabilities and limitations of the assembly language itself. Although it is much more convenient to utilize one of the high level languages to implement this project, doing this in assembly language was challenging and fun at the same time and gave us an ability to see an interworking of a computer system on a lower level.