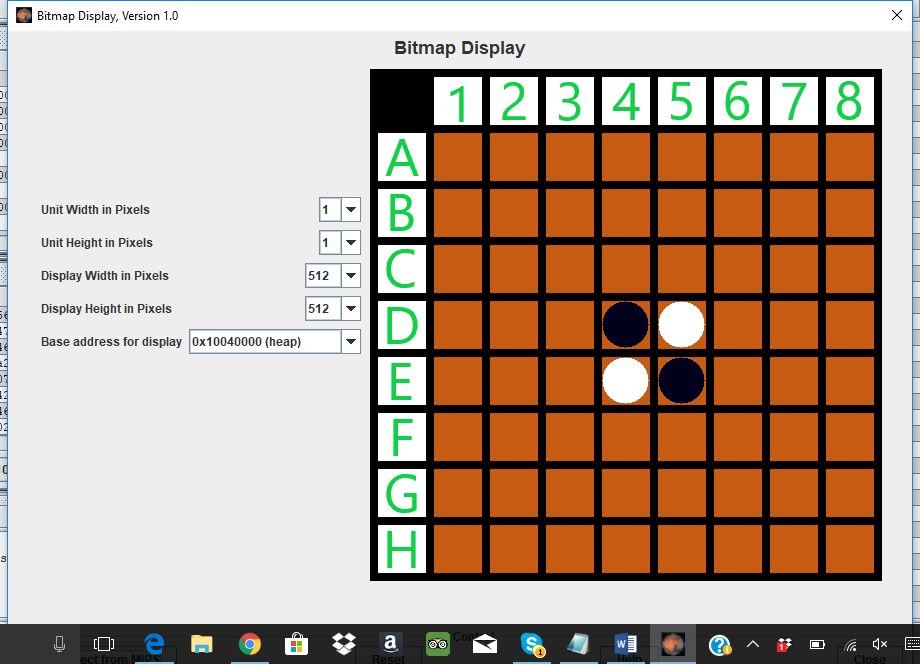
Reversi Game Manual



Team Ex-Caliber

Jiashuai Lu, Khoa Ho, Hector Martinez, Zeby Poycattle

CS 3340.501

Fall 2017

Contents

[Getting Started 3](#_Toc498701916)

[Machine Specific Preparation 3](#_Toc498701917)

[Game Board Preparation 4](#_Toc498701918)

[Assemble the Program 6](#_Toc498701919)

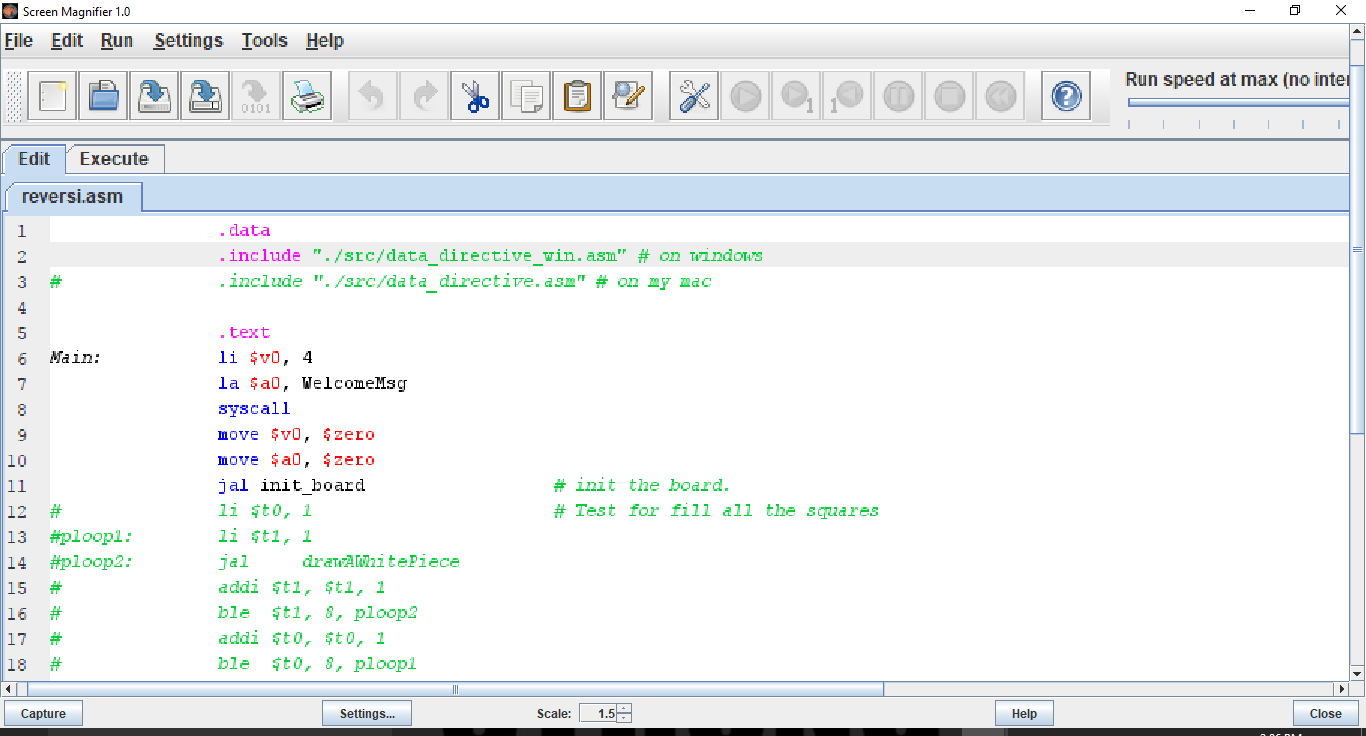
[Run the Program 6](#_Toc498701920)

[Gameplay Instructions 6](#_Toc498701921)

# Getting Started

1. Reversi is a board game that involves 2 players placing game pieces on an 8x8 board. The game has been implemented utilizing MIPS assembly language. To execute the game, one will need to have **MARS (MIPS Assembler and Runtime Simulator)** installed on their machine. **MARS** can be found here: <http://courses.missouristate.edu/KenVollmar/mars/>
2. Reversi contains several modules that need to be stored in a directory on the machine in the way specified below in order to be executed. The name of the modules are: **reversi.asm** (the main module)**, cal\_profit.asm, check\_around.asm, data\_directive.asm, data\_directive\_win.asm, draw\_pieces.asm, init\_board.asm, one\_step.asm, read\_bmp.asm, user\_input.asm,** as well as all **16 labels** that are required for the game board to be displayed. The 16 labels are the numbers **1-8 inclusive** and the letters **A-H inclusive.**
3. The main module, **reversi.asm**, can be stored in a directory chosen by the user, e.g. “C:\reversi\”. The remaining modules will then need to be stored in a sub-folder named “src” within the directory of the main reversi.asm module, e.g. “C:\reversi\src”. And the image label (.bmp) files will be placed in a subfolder named Labels within the “src” folder (e.g. e.g. “C:\reversi\src\Labels”). As in our current package, all files and modules have been placed in their correct directory within the encapsulating compressed (.zip/.rar) file.

# Machine Specific Preparation

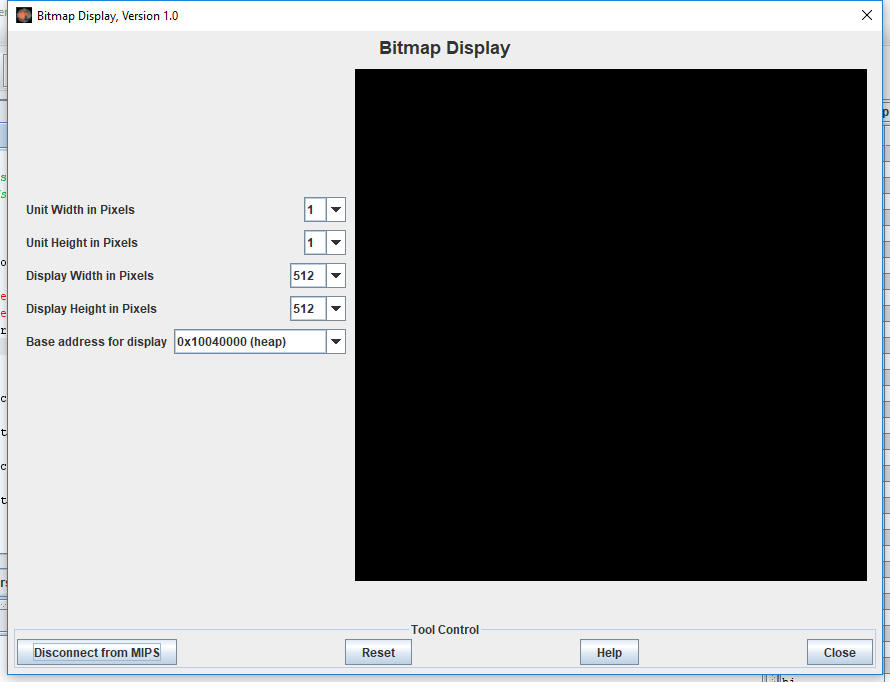
1. Reversi has been implemented to run on Unix, Linux, or Windows supported machines. A minor modification is required to ensure the machine executing the program is supported correctly.
2. Open **Reversi.asm** module in **MARS.** 
   1. If machine is Unix or Linux supported: **Comment out** the 2nd line “.include "./src/data\_directive\_win.asm" # on windows” by inserting a **#** at the beginning of the line.

**Figure 1: Reversi.asm Module, Line 2 or 3 are Commented Out**

* 1. If machine is Windows supported: **Comment out** the **3rd** **line** “.include "./src/data\_directive.asm" # on my mac” by inserting a **#** at the beginning of the line.

# Game Board Preparation

1. The 16 labels (numbers 1-8 and letters A-H) are utilized as the row and column labels of the gameboard.
2. The game board is displayed using the **MARS Bitmap Display** tools.
3. Select **Tools** in the toolbar and then **Bitmap Display**.
4. Ensure **Unit Width in Pixels** is set to **1**, **Unit Height in Pixels** is set to **1**, **Display Width in Pixels** is set to 512, **Display Height in Pixels** is set to 512, and **Base Address for Display** is set to **0x10040000 (heap)**.
5. Select **Connect to MIPS** in the bottom left corner of the window.



**Figure 4: Bitmap Display With All Parameters Set and Connected to MIPS**

# Assemble the Program

1. At this point, the program can be assembled and executed. Navigate to the **Reversi.asm** module within **MARS**.
2. Click on the wrench and screwdriver icon or select **Run** and then **Assemble** in the toolbar.

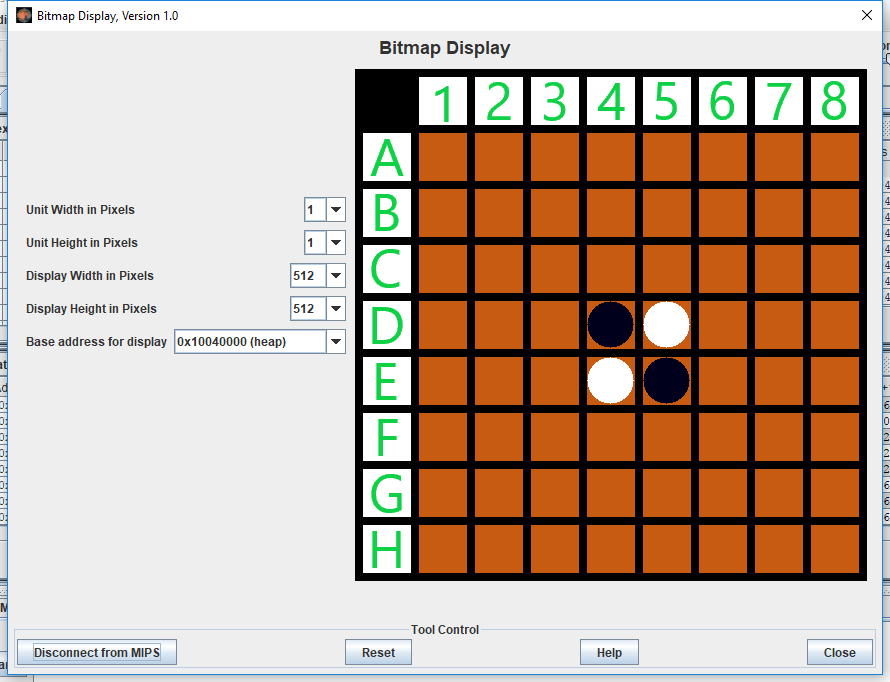
NOTE: If the user is in another module and attempts to assemble the program, assembly will fail. The program can only be assembled correctly from the **Reversi.asm** module.

# Run the Program

1. After successful assembly, the program can be run by selecting the icon with the “play” icon in a green circle, or select select **Run** and then **Go** in the toolbar.

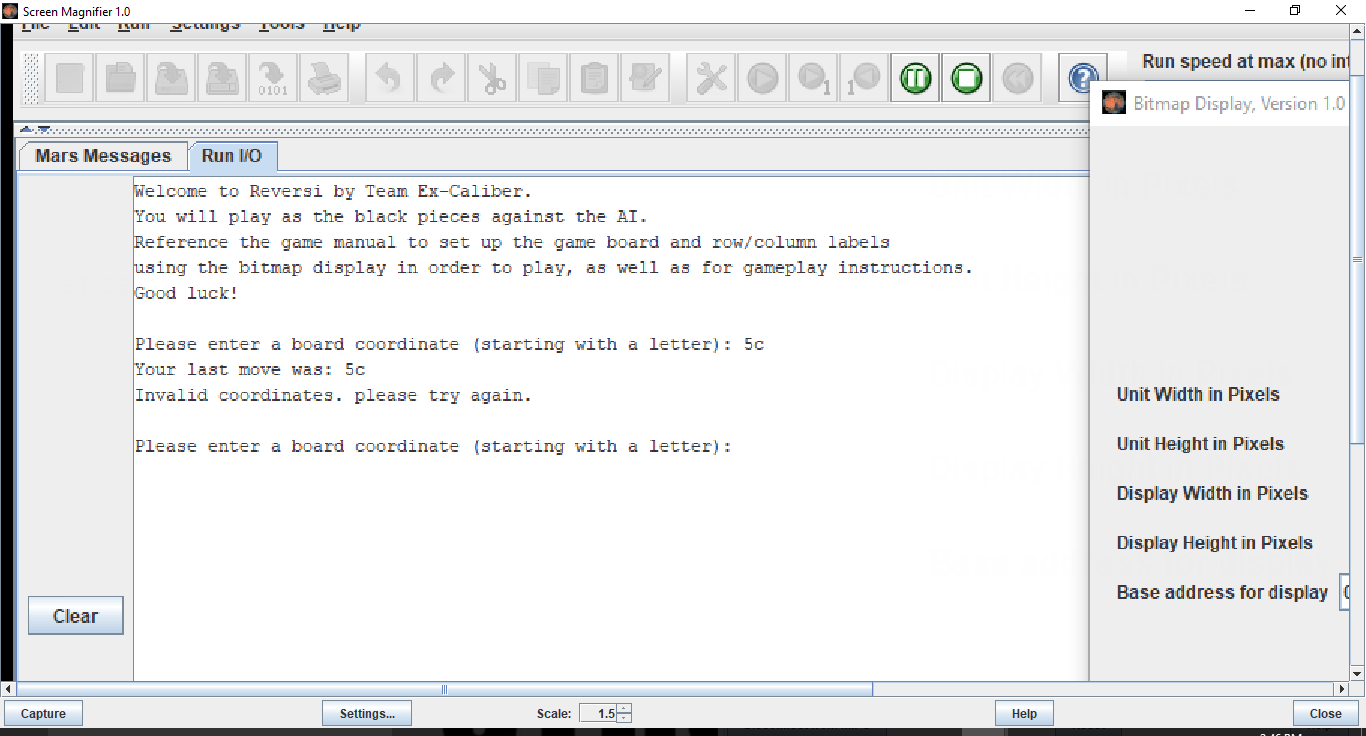
# Gameplay Instructions

1. The objective of Reversi is to capture as many squares of the board by occupying them with a game piece of the player’s color.
2. Pieces are captured by bounding pieces of the other color in succession by pieces of the player’s color.
3. For example, consider the starting board. The user (black) can place a piece on any of the following squares: C5, D6, E3, or F4.



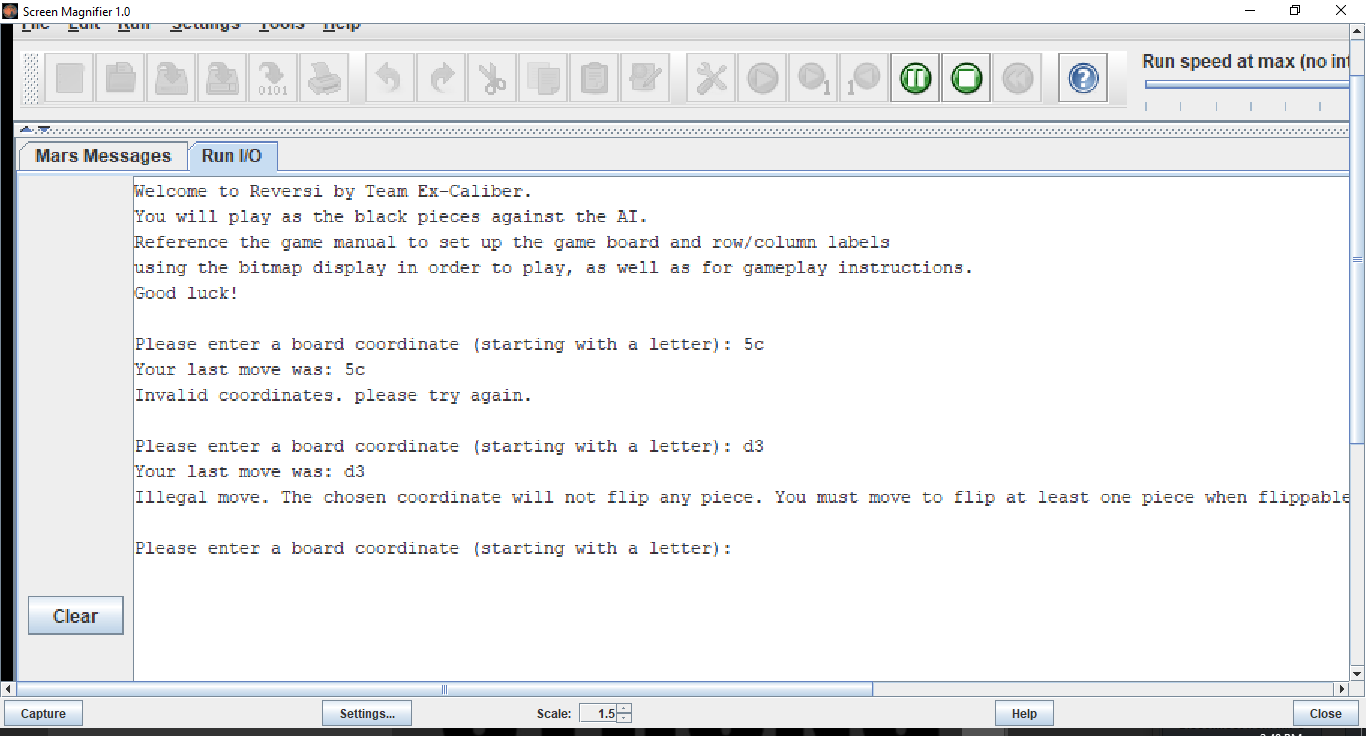
**Figure 5: Initial Game Board**

1. Successfully placing a piece requires typing the desired game board square into the MARS I/O. A piece can be placed by typing the letter (row) and then number (column) without spaces, irrespective of case.
2. If a game board square is input that is syntactically incorrect, the following message will print: “**Invalid coordinates. Please try again.”** The user will be prompted to enter a game board square again.

"

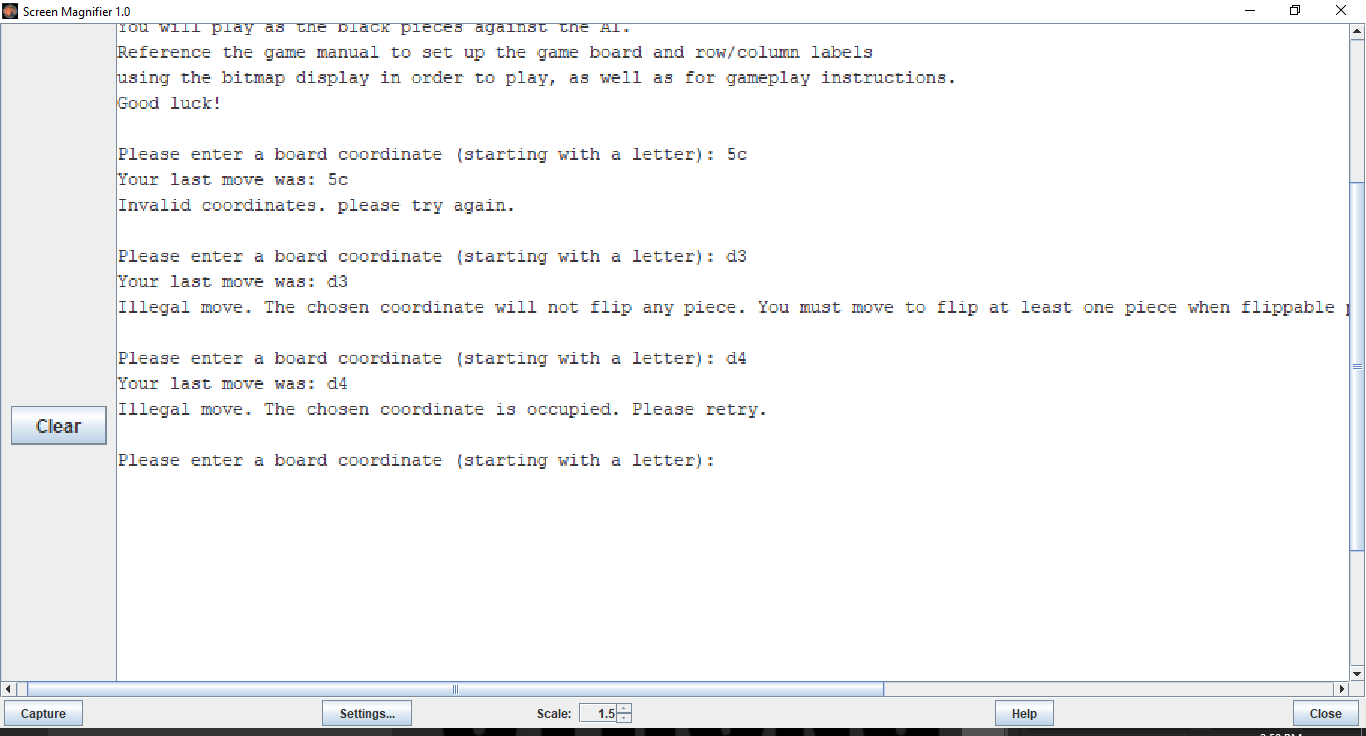
**Figure 6: Invalid Syntax Error Message**

1. If a game board square is input that can not be played because it does not capture any opponent pieces, the following message will print: “**Illegal move. The chosen coordinates will not flip any piece. You must move to flip at least one piece when flappable piece(s) exist.”** The user will be prompted to enter a game board square again.



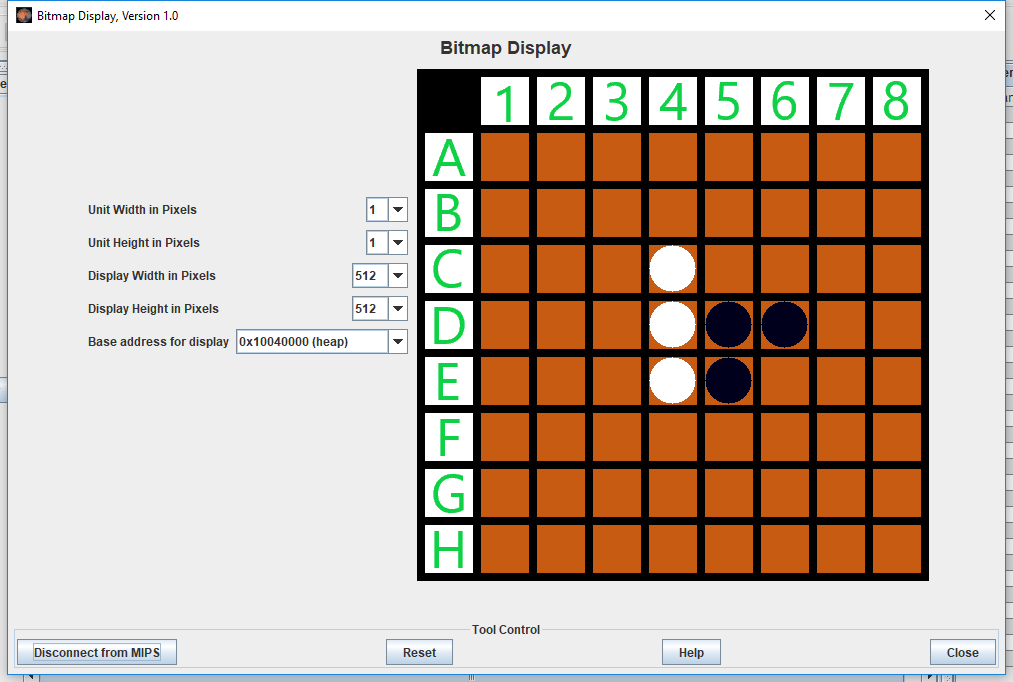
**Figure 7: Illegal Move-No Pieces Change Error Message**

1. If a game board piece is input that is already occupied, the following message will print: “**Illegal move. The chosen coordinate is occupied. Please retry.**”



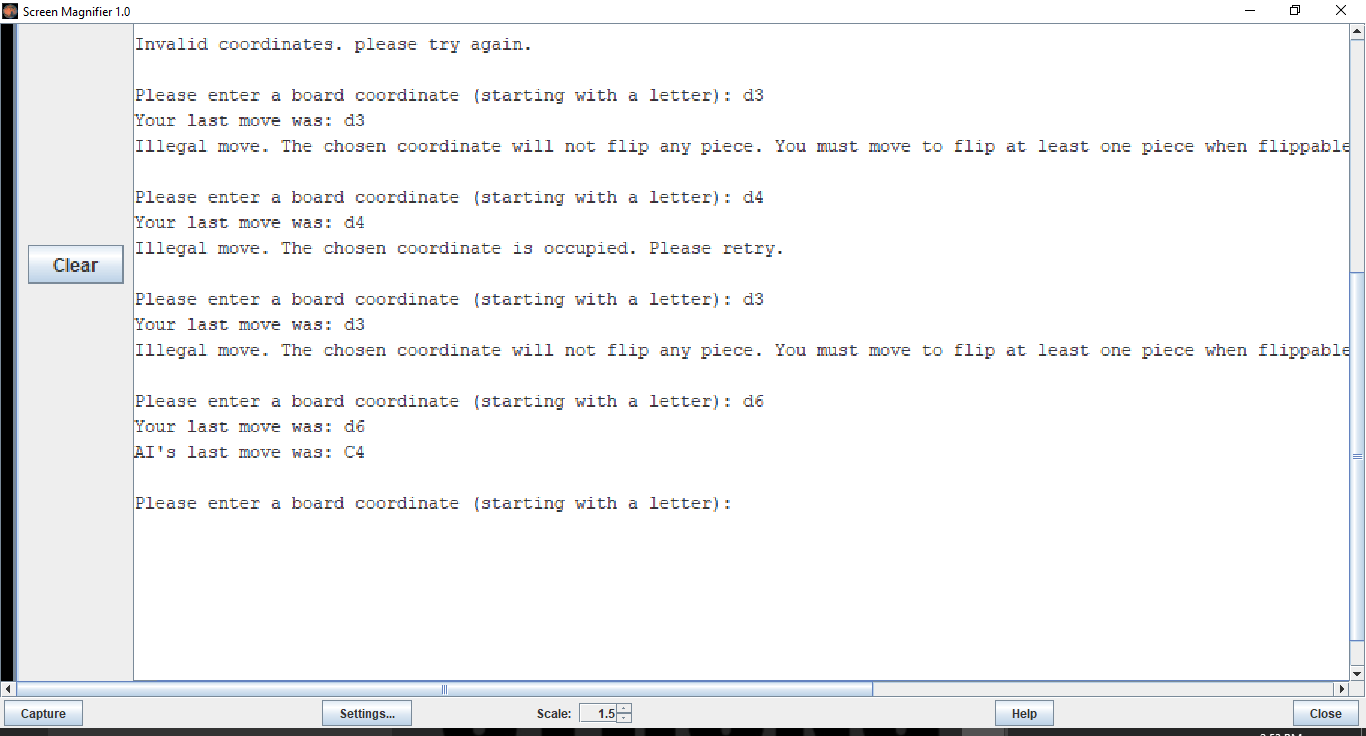
**Figure 8: Illegal Move-Already Occupied Square Error Message**

1. When a valid game board piece is input, the game board will update to reflect the result of the move.
2. The AI will then automatically place the game piece that captures the most user pieces for that turn. The game board will once again update to reflect the result of the move.



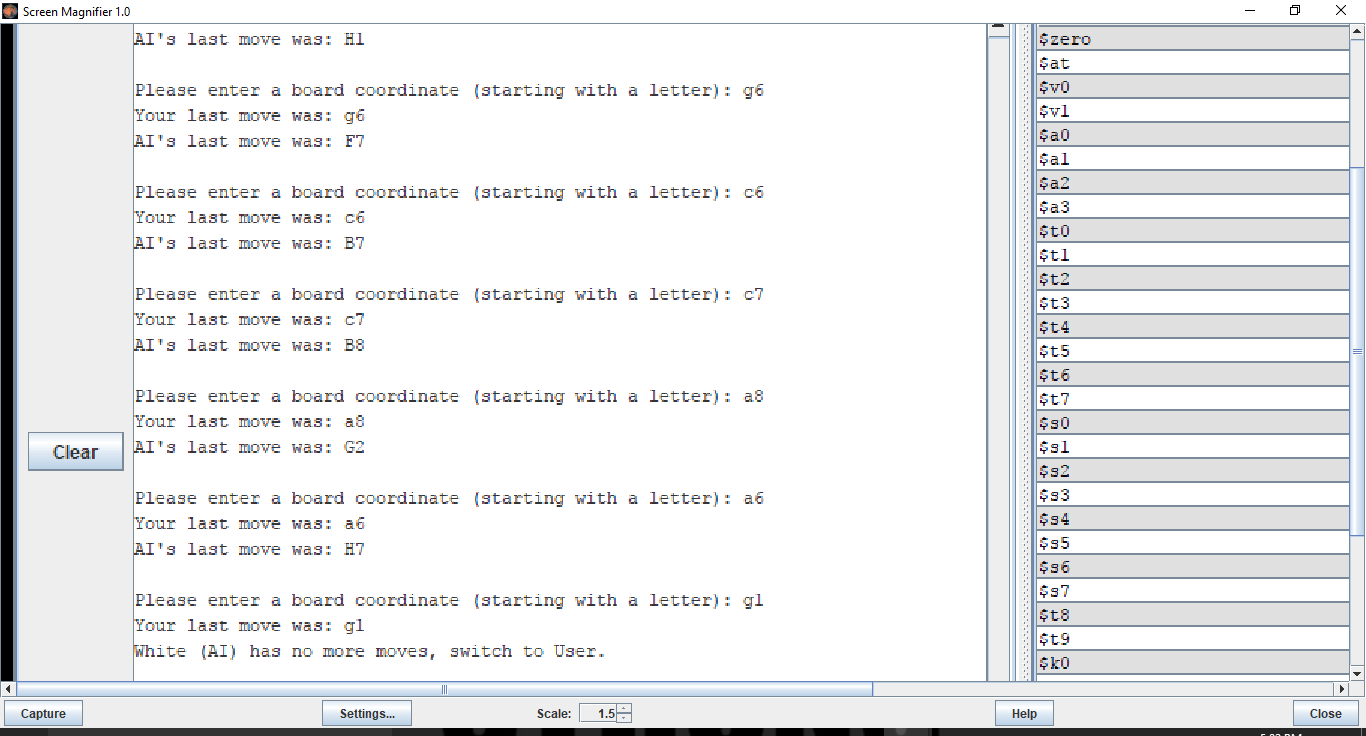
**Figure 9: Game Board After User/AI Turn**

1. The program will also output the **user’s last move** and the **AI’s last move** before prompting the user for their next turn.

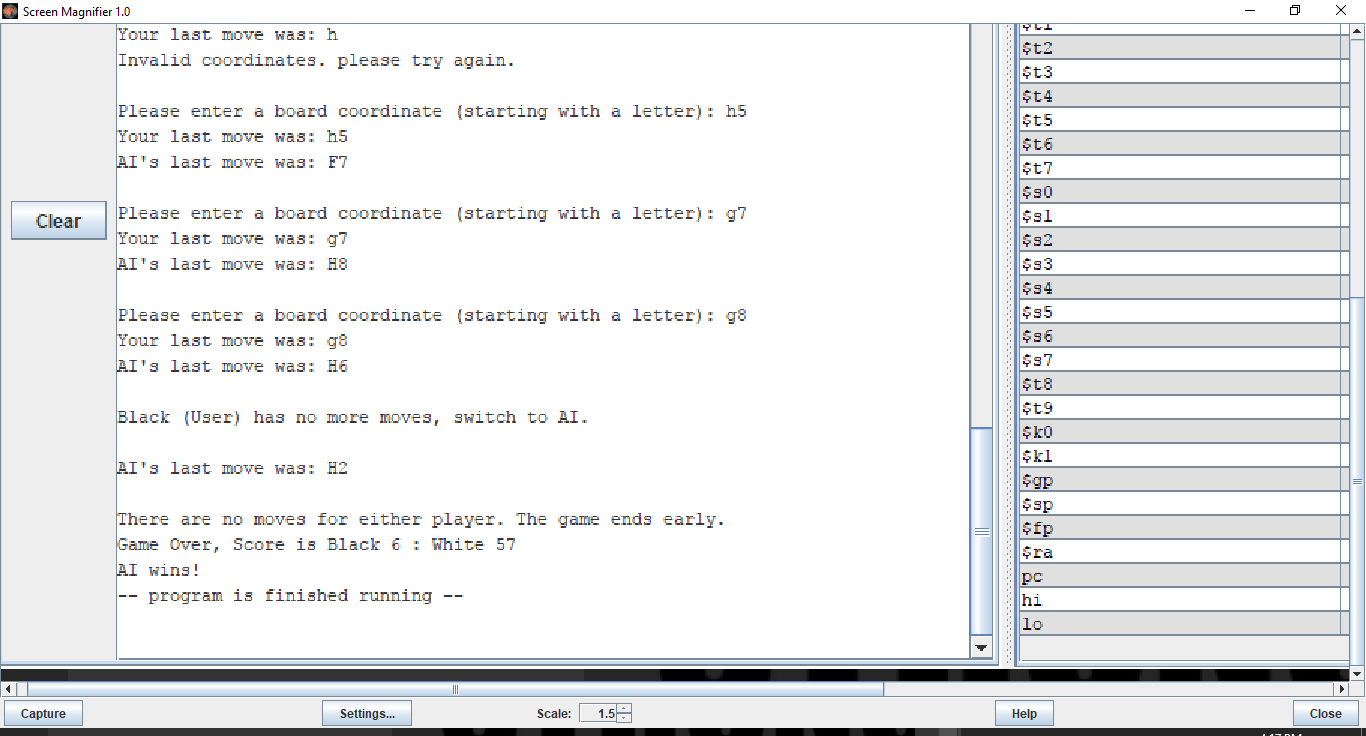


**Figure 10: Move Summary After User/AI Turn Output Message**

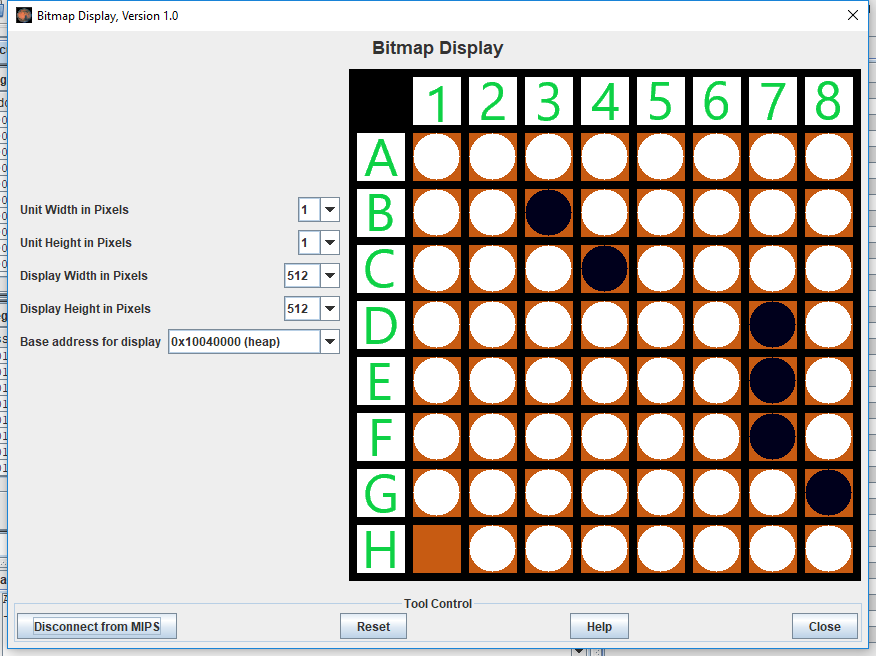
1. The user and computer will continue to alternate turns in this fashion.
2. At some point in the game, a situation could arise where either the user or computer can not make a valid move given the state of the board. Should this happen, the program will switch to the other player’s turn and print out either: “**White (AI) has no more moves, switch to User.**” Or “**Black (User) has no more moves, switch to AI.**”



**Figure 11: No More Moves for Player Output Message**

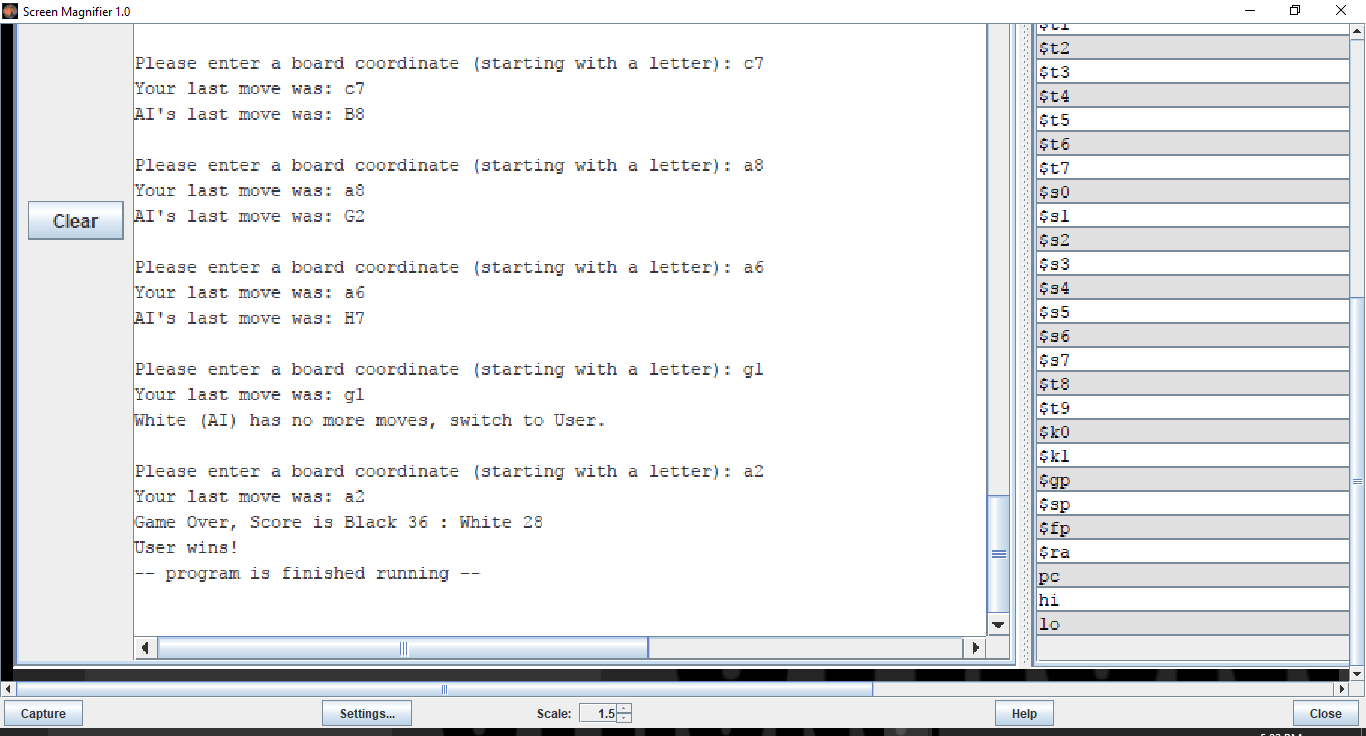
1. If the situation arises that neither player can make a valid move given the state of the board, the game will end and the following message will print: “**There are no moves for either player. The game ends early.**” 

**Figure 12: Game Ends Early Output Message**

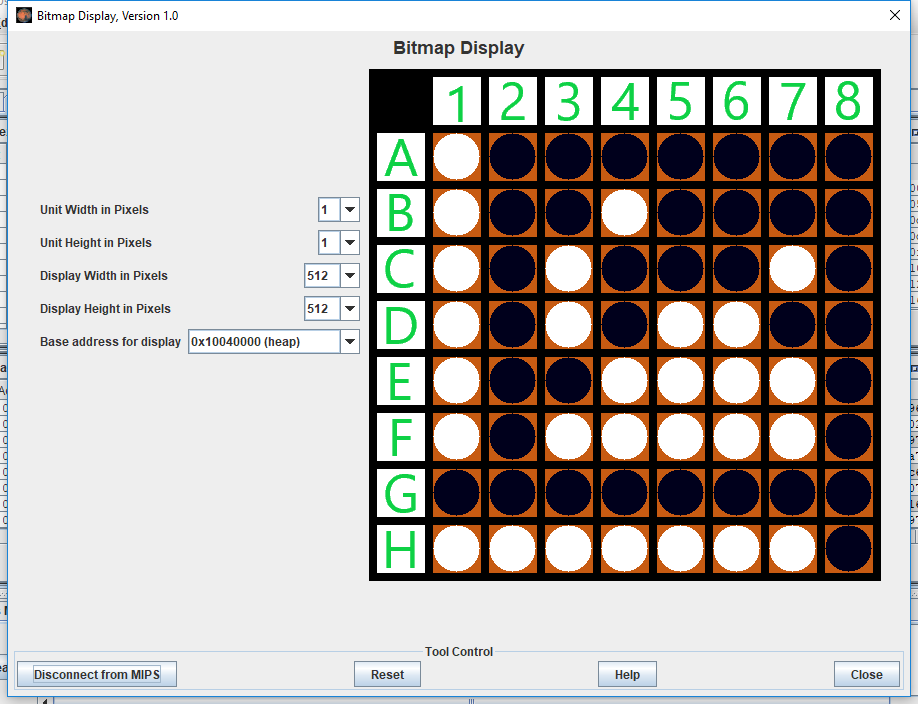


**Figure 13: Sample Scenario Where Game Ends Early**

1. Any time the game ends, whether early or when all the pieces of the board have been occupied, the program will print the score and a message indicating who won, or if the game ended in a tie.



**Figure 14: End of Full Game Output**



**Figure 15: Sample Full Game Board**