\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SAN DIEGO STATE UNIVERSITY \*

\* DUC M LE 132485155 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DECLATION

I, DUC M LE, declare that the work presented in this final project submission is entirely my original work. I have not copied or used any unauthorized sources or assistance beyond what is acknowledged in the accompanying documentation. All code, documentation, and supporting files have been produced solely by me, and any external resources or references have been cited correctly in the report. I understand that any violation of the academic integrity policies will result in appropriate sanctions as per institutional guidelines.



Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_April 4, 2025\_\_

Connect4TWS

**Date Progress: Jan 22, 2025**

**Description:** This game is a beginner-friendly C programming project to enhance foundational programming skills. The game features an 8x8 grid array where players compete to win with the following elements:

1. **Player Setup**: The user can choose the number of players for the game.
2. **Menu**: There will be a menu for user selection, either for coin flip or exit the game
3. **Unique Player Symbols**: The program uses x and o as a standard for better visualization.
4. **Wormholes and Traps**: There will be three traps and three wormhole spawns independently in each row and column, minus the 1 row and column.
   * **Wormhole**: Transports the player’s disc to another row.
   * **Trap**: Trap with another player disc.
5. **Snakes**: These are randomly spawned small cubes that remove a player’s disc from the grid when encountered.
6. **Coin Flip**: using the coin flip to randomly start for player 1 or 2, in a coin flip animation.
7. **Exit Option**: Players can type "exit" to quit the game anytime.
8. **Winner Recording**: The program outputs the winner(s) to a text file for record-keeping.
9. **Core Programming Concepts**:
   * Uses pointers and pass-by-value arrays for efficient memory handling.
   * Employs file I/O to save game results.
10. **Error Handling**: Provides user-friendly error messages for invalid inputs, such as choosing the same symbol or invalid commands.

**List of Core Functions**

**C code**

 **main()**

* Seeds the random number generator.
* Initiates the program by calling the menu function.

 **ShowMenu()**

* Clears the screen and displays a welcome banner.
* Presents a menu with two options: Start Game and Exit.
* Handles user input and directs flow based on choice:
  + On choosing Start, it initializes the game board, spawns challenge, performs a coin flip, and starts the main game loop.
  + Choosing Exit terminates the program.

 **coinFlipAnimation()**

* Simulates a coin-flip animation by rapidly displaying coin faces.
* Performs a final coin flip to decide which player will start the game.
* Returns 1 for Heads (Player 1) or 2 for Tails (Player 2).

 **getDebugFileName() & writeDebugLog()**

* **getDebugFileName()**: Constructs a filename based on the current date for logging purposes.
* **writeDebugLog()**: Writes debug messages to the generated log file.

 **logBoardState()**

* Logs the current state of the game board (both in a formatted grid and raw memory addresses) into the debug file for troubleshooting.

 **displayBoard()**

* Clears the screen.
* Prints the current board with row and column headers, showing the positions of discs and challenges.

 **Exit()**

* Provides a clean exit from the game when the user chooses to terminate the session.

 **Welcome()**

* Initializes an 8x8 game board (allocating a 2D array).
* Sets up the players’ symbols (typically 'x' and 'o').
* Displays an initialization message.

 **checkWinner()**

* Evaluates the board after each move to check for winning conditions horizontally, vertically, and diagonally.
* Returns a win status if four consecutive matching discs are found.

 **spawnChallenges()**

* Randomly places a specified number of traps ('T') and wormholes ('W') on the board.
* Ensures that challenges are not placed in row 0 and are placed only on empty cells.

 **setChallenge()**

* Places a particular challenge symbol at a random, valid location on the board.
* Logs the placement details for debugging.

 **teleportDisc()**

* Moves (teleports) a disc from its current location to a random empty cell.
* Logs and displays the board state after teleportation.

 **logPlayerDiscAddresses()**

* Logs the memory addresses of all player discs currently on the board.
* Useful for debugging memory allocation and board updates.

 **handleTrapLogic()**

* Implements the logic when a player lands on a trap ('T'):
  + Searches for the opponent’s discs and, if possible, places the current player's disc above them.
  + Logs the trap event and displays the updated board.

 **recordWinner()**

* Writes the game-winning event into a file with a timestamp.
* Records which player won and when.

 **snakeAttack()**

* Triggers a snake attack which randomly removes two of the current player's discs.
* Displays snake ASCII art and delays before resuming the game.
* Ensures that the removal is permanent for that game session.

 **MainGame()**

* Contains the main game loop:
  + Displays the board.
  + Handles user input for dropping discs.
  + Checks for snake attacks based on elapsed game time.
  + Applies trap and wormhole logic.
  + Checks for win conditions after each move.
  + Alternates between players and updates the game state.
  + Frees allocated memory at the end of the game.

**Mips Code**

### **Main Function**

* **main:**
  + Purpose: Entry point of the program. It calls ShowMenu to display the game menu and then exits.
  + Actions:
    - jal ShowMenu: Jumps to the ShowMenu procedure to display the menu and handle user input.
    - li $v0, 10 and syscall: Exits the program.

### **ShowMenu Procedure**

* **ShowMenu:**
  + Purpose: Displays the game menu and processes user input to either start the game or exit.
  + **Loops and Actions:**
    - **show\_menu\_loop:**
      * Displays the welcome message (welcome\_msg), menu options (menu\_prompt), and prompts for user input (get\_userinput).
      * Reads an integer input (li $v0, 5; syscall) and stores it in $t0.
      * Branches based on input:
        + beq $t0, 1, start\_game: If input is 1, jumps to start\_game.
        + beq $t0, 2, exit\_program: If input is 2, jumps to exit\_program.
        + Otherwise, displays invalid\_msg and loops back.
    - **start\_game:**
      * Initializes the 8x8 game board with spaces:
        + **init\_board\_loop:**

Iterates 64 times (size of the board) using $t2 as a counter.

Stores a space character (' ') in each board position (sb $t3, 0($t1)).

Increments the board pointer ($t1) and decrements the counter ($t2) until zero.

* + - * Calls coinFlipAnimation to determine the starting player and stores the result in $s0.
      * Calls MainGame to start the game.
      * Jumps to end\_show\_menu.
    - **exit\_program:**
      * Displays exit\_msg and exits the program.
    - **end\_show\_menu:**
      * Restores the return address ($ra) and returns to main.

### **Coin Flip Animation Procedure**

* **coinFlipAnimation:**
  + Purpose: Simulates a coin flip animation to decide which player goes first (Player 1 or Player 2).
  + **Loops and Actions:**
    - Initializes random seed using system time (li $v0, 30; syscall) and sets it (li $v0, 40).
    - **flip\_loop:**
      * Runs 5 times ($s0 as counter) to simulate coin flipping animation.
      * Generates a random number (0 or 1) using li $v0, 42; li $a1, 2; syscall.
      * If $t1 is 0, jumps to draw\_tails\_loop; otherwise, calls drawHeads.
      * **draw\_tails\_loop:**
        + Calls drawTails to draw the tails side of the coin.
      * **flip\_continue:**
        + Displays a sequence of coin states with delays:

drawEdgeVertical (vertical edge), delay 500ms.

drawTails, delay 500ms.

drawEdgeHorizontal (horizontal edge), delay 500ms.

drawHeads, delay 500ms.

* + - * + Decrements $s0 and loops until zero.
    - Final flip:
      * Generates a final random number to determine the result.
      * If 0, calls drawTails, displays tails\_result, and sets $v0 to 2 (Player 2 starts).
      * If 1, calls drawHeads, displays heads\_result, and sets $v0 to 1 (Player 1 starts).
    - **end\_coin\_flip:**
      * Restores registers and returns $v0 (1 or 2).

### **Main Game Procedure**

* **MainGame:**
  + Purpose: Manages the core game loop, including player turns, disc placement, wormholes, traps, and win checking.
  + **Loops and Actions:**
    - **game\_loop:**
      * Displays the board (jal displayBoard).
      * Prompts the current player (e.g., "Player 1x, enter column...") using $s1 (0 or 1) and the player symbol from players.
      * Reads user input into buffer:
        + If input is 'e', jumps to end\_game.
        + Converts input to a column number ($t0 - '0') and validates (0-7); otherwise, jumps to invalid\_input.
      * **find\_spot\_loop:**
        + Searches for the lowest empty spot in the chosen column ($s2):

Starts from row 7 ($t0 = 7) and moves up.

Calculates board index ($t1 = $t0 \* 8 + $s2).

Checks if the spot is empty (' '); if so, jumps to place\_disc.

If no empty spot is found (row < 0), jumps to invalid\_input.

* + - * **place\_disc:**
        + Places the player’s symbol ('x' or 'o') at the found position.
        + Calls checkWormholesAndTrap with row ($t0), column ($s2), and symbol ($t4):

If $v0 = 1, wormhole activated; jumps to wormhole\_handled.

If $v0 = 2, trap activated; jumps to trap\_handled.

* + - * + Displays the updated board.
        + Calls checkWinner with row ($s3) and column ($s2):

If $v0 = 1, a win is detected; displays the winner and ends the game.

* + - * **continue\_game:**
        + Toggles the current player ($s1 = ($s1 + 1) % 2) and loops back.
      * **invalid\_input:**
        + Displays invalid\_col and loops back to prompt again.
      * **end\_game:**
        + Restores registers and returns.

### **Display Board Procedure**

* **displayBoard:**
  + Purpose: Prints the current state of the 8x8 game board with row/column labels.
  + **Loops and Actions:**
    - **col\_header\_loop:**
      * Prints column numbers (0-7) with spacing (space\_two and space).
      * Iterates $t0 from 0 to 7.
    - **row\_loop:**
      * Iterates over rows (0-7) using $t0:
        + Prints the row number ($t0) and a '|' character.
        + **col\_loop:**

Iterates over columns (0-7) using $t1:

Prints spacing (space), the board character at board[$t0\*8 + $t1], and a '|' character.

Prints a newline after each row.

* + - Returns after printing the full board.

### **Check Wormholes and Trap Procedure**

* **checkWormholesAndTrap:**
  + Purpose: Checks if a placed disc lands on a wormhole entry or trap position and handles the action.
  + **Actions:**
    - Compares the placed position ($a0, $a1) with wormhole entries:
      * **check\_w1\_col:** If matches wormhole1\_entry (2,3), jumps to wormhole1\_activate.
      * **check\_w2\_col:** If matches wormhole2\_entry (1,5), jumps to wormhole2\_activate.
      * **check\_w3\_col:** If matches wormhole3\_entry (4,1), jumps to wormhole3\_activate.
    - **wormhole1\_activate:**, **wormhole2\_activate:**, **wormhole3\_activate:**
      * Calls handleWormhole with the corresponding exit position.
      * If checkWinner returns 1, sets $v0 = 1 (wormhole activated) and ends.
    - **check\_trap:**
      * If matches trap\_pos (3,4), jumps to trap\_activate.
    - **trap\_activate:**
      * Calls removeTrapDisc to clear the trap position.
      * If trap is activated ($v0 = 1), calls trapOpponentDiscs and sets $v0 = 2.
    - **no\_special:**
      * Sets $v0 = 0 if no wormhole or trap is triggered.
    - Returns $v0 (0, 1, or 2).

### **Handle Wormhole Procedure**

* **handleWormhole:**
  + Purpose: Teleports a disc from the entry position to the exit position if the exit is empty.
  + **Actions:**
    - Calculates the entry index ($s0 \* 8 + $s1) and loads the disc ($t7).
    - Loads the exit position from $a0 (row $t3, col $t4) and calculates its index.
    - If the exit is empty (' '), moves the disc:
      * Clears the entry position (sb ' ', 0($t2)).
      * Places the disc at the exit (sb $t7, 0($t6)).
      * Displays wormhole\_msg.
      * Checks for a win at the exit position (jal checkWinner).
    - Returns.

### **Remove Trap Disc Procedure**

* **removeTrapDisc:**
  + Purpose: Removes the disc at the trap position if it hasn’t been used yet.
  + **Actions:**
    - Checks if the trap is already used (trap\_used != 0); if so, returns 0.
    - Verifies the position matches trap\_pos (3,4).
    - If matched and unused:
      * Clears the position (sb ' ', 0($t2)).
      * Sets trap\_used = 1.
      * Returns $v0 = 1.
    - Otherwise, returns $v0 = 0.

### **Trap Opponent Discs Procedure**

* **trapOpponentDiscs:**
  + Purpose: Places the current player’s disc above or below each opponent’s disc if possible.
  + **Loops and Actions:**
    - **trap\_row\_loop:**
      * Iterates over rows (0-7) using $t0.
      * **trap\_col\_loop:**
        + Iterates over columns (0-7) using $t1.
        + Checks if the position contains the opponent’s disc ($t4 = $s1).
        + If found:

**try\_below:** Checks below (row + 1); if empty, places the player’s disc (sb $s0).

**try\_above:** Checks above (row - 1); if empty, places the player’s disc.

* + - * + Skips if no empty spot is available.
    - Displays trap\_msg and returns.

### **Check Winner Procedure**

* **checkWinner:**
  + Purpose: Checks for exactly 4 consecutive matching pieces in any direction from the placed disc.
  + **Loops and Actions:**
    - Loads the placed disc’s symbol ($s3).
    - **Horizontal Check:**
      * **horizontal\_loop:** Counts left from $s1 until a mismatch or boundary.
      * **check\_right:** Counts right until a mismatch or boundary.
      * Wins if total count ($t3) equals 4.
    - **Vertical Check:**
      * **vertical\_loop:** Counts up from $s0.
      * **check\_down:** Counts down.
      * Wins if total is 4.
    - **Forward Diagonal (Top-Left to Bottom-Right):**
      * **diagonal\_forward\_loop:** Counts up-left.
      * **check\_down\_right:** Counts down-right.
      * Wins if total is 4.
    - **Backward Diagonal (Top-Right to Bottom-Left):**
      * **diagonal\_backward\_loop:** Counts up-right.
      * **check\_down\_left:** Counts down-left.
      * Wins if total is 4.
    - Returns $v0 = 1 and $v1 = 1 (Player 1) or 2 (Player 2) if a win is detected; otherwise, $v0 = 0, $v1 = 0.

### **Draw Coin Flip Procedures**

* **drawHeads:**
  + Purpose: Draws the "heads" side of the coin (circle with an 'H').
  + **Loops:**
    - **clear\_heads\_loop:** Clears an 8x8 area with black.
    - **heads\_row\_loop:** and **heads\_col\_loop:**
      * Draws a silver circle with a yellow edge (radius ~3.5).
      * Adds a red 'H' in the middle row (cols 3 and 4).
* **drawTails:**
  + Purpose: Draws the "tails" side (circle with a 'T').
  + **Loops:**
    - **clear\_tails\_loop:** Clears the area.
    - **tails\_row\_loop:** and **tails\_col\_loop:**
      * Draws a silver circle with a yellow edge.
      * Adds a green 'T' in the middle row (col 3).
* **drawEdgeVertical:**
  + Purpose: Draws a vertical yellow line (coin edge).
  + **Loops:**
    - **clear\_edge\_v\_loop:** Clears the area.
    - **edge\_v\_loop:** Draws a vertical line at x=15.
* **drawEdgeHorizontal:**
  + Purpose: Draws a horizontal yellow line (coin edge).
  + **Loops:**
    - **clear\_edge\_h\_loop:** Clears the area.
    - **edge\_h\_loop:** Draws a horizontal line at y=15.

high-level description of your C code expressed in pseudo-code

**// MAIN PROGRAM FLOW**

**MAIN:**

**seed random number generator using current time**

**call ShowMenu()**

**exit program**

**// SHOW MENU**

**ShowMenu:**

**LOOP forever:**

**clear screen**

**display large welcome banner and menu options:**

**"1) Start Game"**

**"2) Exit"**

**prompt user for choice and read input**

**IF choice == 1 THEN:**

**clear screen**

**// Game Initialization**

**call Welcome() to:**

**- allocate an 8x8 board (2D array)**

**- initialize all cells to a blank space**

**- set up the players' symbols ('x' and 'o')**

**call spawnChallenges() to randomly place traps ('T') and wormholes ('W')**

**delay (simulate processing time)**

**startingPlayer = call coinFlipAnimation() // simulate coin flip and determine who goes first**

**delay (additional pause)**

**call MainGame(board, rows, cols, players, startingPlayer)**

**break out of the loop // game ends after MainGame finishes**

**ELSE IF choice == 2 THEN:**

**display "Exiting the program. Goodbye!"**

**exit program**

**ELSE:**

**display "Invalid choice. Please enter 1 or 2."**

**delay (allow user to read message)**

**continue loop**

**// COIN FLIP ANIMATION**

**coinFlipAnimation:**

**FOR a fixed number of flips:**

**generate random value (0 or 1)**

**clear screen**

**display "Flipping the coin..." and show corresponding ASCII art (heads or tails)**

**delay (simulate animation speed)**

**PERFORM final coin flip:**

**generate final random result (0 or 1)**

**clear screen**

**display final coin face (heads or tails)**

**RETURN value indicating starting player:**

**1 for Heads (Player 1)**

**2 for Tails (Player 2)**

**// WELCOME AND BOARD INITIALIZATION**

**Welcome:**

**display a message about initializing an 8x8 board for two players**

**allocate memory for a 2D board (8 rows × 8 columns)**

**initialize each board cell with a space (' ')**

**allocate memory for a players array and assign:**

**players[0] = 'x'**

**players[1] = 'o'**

**return board, row count, column count, and players**

**// MAIN GAME LOOP**

**MainGame(board, r, c, players, startingPlayer):**

**set currentPlayer based on startingPlayer (adjust index accordingly)**

**record the start time of the game**

**initialize snake attack timer (random interval between 10 and 30 seconds)**

**WHILE game is NOT over:**

**IF current time - last snake attack time >= snake interval AND snake count < limit:**

**call snakeAttack(board, r, c, currentPlayer’s disc)**

**update snake attack timer and count**

**continue to next iteration (skip regular turn)**

**call logBoardState(board, r, c) // optional debug logging**

**call logPlayerDiscAddresses(board, r, c) // optional debug logging**

**call displayBoard(board, r, c) to show current board**

**prompt current player:**

**"Player {number} ({disc}), enter column (0 to c-1) to drop your disc, or 'e' to quit: "**

**read input**

**IF input == 'e' THEN:**

**display exit message for current player**

**set game over flag to true**

**CONTINUE to exit loop**

**convert input to integer as column number**

**IF column number is invalid (not in range 0 to c-1):**

**display error "Invalid column! Choose between 0 and c-1."**

**continue loop (retry current player's turn)**

**find the lowest available row in the selected column:**

**FOR row from bottom (r - 1) upward:**

**IF board cell is empty OR contains a trap ('T') OR wormhole ('W'):**

**mark this row as the target**

**break loop**

**IF no valid row found (i.e., column is full):**

**display "Column is full! Try again."**

**continue loop**

**IF the target cell contains a trap ('T'):**

**call handleTrapLogic(board, r, c, row, col, currentPlayer’s disc)**

**ELSE IF the target cell contains a wormhole ('W'):**

**remove wormhole marker from board cell**

**call teleportDisc(board, r, c, row, col, currentPlayer’s disc)**

**ELSE:**

**place currentPlayer’s disc at the target cell**

**IF checkWinner(board, r, c, row, col, currentPlayer’s disc) returns TRUE THEN:**

**clear screen**

**display winning message indicating which player won**

**call recordWinner() to log the win event with timestamp**

**display final board state**

**set game over flag to true**

**IF game is not over:**

**switch currentPlayer to the other player (toggle between 0 and 1)**

**// End of game loop**

**call logBoardState(board, r, c) // final board logging**

**free allocated memory for board and players**

**display final thank-you message for playing**

**// ADDITIONAL FUNCTIONS**

**checkWinner:**

**check for four consecutive matching discs in:**

**- horizontal direction**

**- vertical direction**

**- diagonal directions (both \ and /)**

**return TRUE if a winning condition is met, otherwise FALSE**

**spawnChallenges:**

**randomly place a specified number of traps ('T') and wormholes ('W') on the board**

**ensure:**

**- traps and wormholes are not placed in row 0**

**- placement occurs only in empty cells**

**for wormholes, ensure they are placed in unique rows and columns**

**handleTrapLogic:**

**when a trap is activated:**

**search the board for opponent's discs**

**if found and an empty cell is available above, place the current player’s disc there**

**otherwise, place the disc in the trap cell**

**log the trap event for debugging**

**teleportDisc:**

**when a wormhole is encountered:**

**search for a random empty cell on the board**

**place the current player’s disc in that cell**

**log the teleportation event**

**display the board after teleportation**

**snakeAttack:**

**collect positions of current player's discs on the board**

**IF there are at least two discs:**

**randomly choose two distinct disc positions**

**remove (set to empty) those discs from the board**

**clear screen and display updated board with snake ASCII art**

**delay for a few seconds to show the effect**

**// The removed discs are permanently gone**

**recordWinner:**

**record the winning event (player number, disc, timestamp) in a log file for future reference**

**logBoardState, displayBoard, logPlayerDiscAddresses:**

**support debugging by printing/logging the current board state and memory details**

**C Code**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SAN DIEGO STATE UNIVERSITY \*

\* DUC M LE 132485155 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-------------------------- END-OF-HEADER -----------------------------

\*

\* Connect4TWS.c

\*

\*

\* Created on: Sep 6, 2024

\* Update on: Mar 2, 2025

\* Author: DUC M LE

\*

\* General: This game is a mashup of Wormhole, Tic-Tac Toe, Snake and Connect4. Whenever the user drops the disc into a wormhole,

\* it teleports the disc into a random column and row. Whenever the user drop the disc into trap , it will trap other player. It also

\* has snake spawn to take away player disc. There are 3 wormhole and one trap for standard interaction.

\*

\* Function: Uses board pointers to initialize and manage the game board.

\*

\* Feature added : Menu selection, coin flip.

\*

\* Note: This is a finish C for this game, there is not anymore to add for feature all the feature is consideration. Focus on MIPS. Clean the C program to look nice

\* and professional.

\*/

// Standard headers

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h>

**#include** <string.h>

**#include** <stddef.h>

**#include** <unistd.h>

**#include** <sys/types.h>

**#include** <sys/stat.h>

// Custom headers

**#include** "Header.h"

/\* ----------------------------------------------------------------

\* MAIN

\* - Shows a menu with two options: Start or Exit

\* - If user chooses Start -> coin flip -> game

\* - If user chooses Exit -> quits the program

\* ----------------------------------------------------------------

\*/

**int** **main**() {

// Seed the random number generator (for coinFlip)

**srand**((**unsigned** **int**)**time**(NULL));

ShowMenu(); // This will handle the user’s menu selection

**return** 0; // We'll usually never reach here if "Exit" is chosen

}

/\* ----------------------------------------------------------------

\* ShowMenu

\* - Clears screen

\* - Prints a large ASCII banner: "Welcome to Connect 4 TrapWormhole"

\* - Offers two options: 1) Start 2) Exit

\* - If user chooses 1, does coin-flip, then starts the game

\* - If user chooses 2, exits the program

\* ----------------------------------------------------------------

\*/

**void** **ShowMenu**(**void**) {

**int** choice = 0;

**while** (1) {

**system**(CLEAR\_SCREEN);

**printf**("=======================================================\n");

**printf**(" WELCOME TO CONNECT 4 TRAPWORMHOLE! \n");

**printf**("=======================================================\n");

**printf**(" A Twist on the Classic Game \n");

**printf**(" With Wormholes & Traps \n");

**printf**("=======================================================\n\n");

// Menu

**printf**("Menu:\n");

**printf**(" 1) Start Game\n");

**printf**(" 2) Exit\n\n");

**printf**("Enter your choice (1 or 2): ");

**scanf**("%d", &choice);

**if** (choice == 1) {

// Clear screen once user picks start

**system**(CLEAR\_SCREEN);

// 1) Initialize board

**char** \*\*board = NULL;

**int** r = 0, c = 0;

**char** \*players;

Welcome(&board, &r, &c, &players);

// 2) Spawn 1 traps & 3 wormholes (not in row 0)

spawnChallenges(board, r, c, 1, 3);

// 3) Simulate a small delay (1200 ms)

SLEEP\_MS(1200);

// 4) Coin flip -> decide who starts

**int** flipResult = coinFlipAnimation();

**if** (flipResult == 1) {

**printf**("Heads! Player 1 goes first.\n");

} **else** {

**printf**("Tails! Player 2 goes first.\n");

}

**int** startingPlayer = flipResult - 1; // (1->0 => P1, 2->1 => P2)

// Add 1000ms delay before showing the board

SLEEP\_MS(1900);

// 5) Launch the main game loop

MainGame(board, r, c, players, startingPlayer);

// After game ends, break out

**break**;

}

**else** **if** (choice == 2) {

**printf**("Exiting the program. Goodbye!\n");

**exit**(0);

}

**else** {

**printf**("Invalid choice. Please enter 1 or 2.\n");

SLEEP\_MS(1200);

}

}

}

/\* ----------------------------------------------------------------

\* coinFlipAnimation

\* - Displays a quick coin-flip animation (~2 seconds)

\* - Then does a final flip:

\* 1 => Heads => Player 1

\* 2 => Tails => Player 2

\* ----------------------------------------------------------------

\*/

**int** **coinFlipAnimation**(**void**) {

**const** **int** flips = 10;

**const** **int** delayMs = 200;

**const** **char**\* headsArt =

" \_\_\_\_\_\_\_\n"

" / \\ \n"

" | HEADS |\n"

" \\\_\_\_\_\_\_\_/\n";

**const** **char**\* tailsArt =

" \_\_\_\_\_\_\_\n"

" / \\ \n"

" | TAILS |\n"

" \\\_\_\_\_\_\_\_/\n";

**for** (**int** i = 0; i < flips; i++) {

**int** showHeads = **rand**() % 2; // 0 or 1

**system**(CLEAR\_SCREEN);

**printf**("Flipping the coin...\n\n");

**if** (showHeads) {

**printf**("%s\n", headsArt);

} **else** {

**printf**("%s\n", tailsArt);

}

SLEEP\_MS(delayMs);

}

// Final REAL flip

**int** finalFlip = **rand**() % 2;

**system**(CLEAR\_SCREEN);

**printf**("\n\n");

**if** (finalFlip == 1) {

**printf**("%s\n", headsArt);

**return** 1; // Heads → Player 1

} **else** {

**printf**("%s\n", tailsArt);

**return** 2; // Tails → Player 2

}

}

/\* ----------------------------------------------------------------

\* getDebugFileName

\* -Save file debug as a text to show detail log.

\* ----------------------------------------------------------------

\*/

**void** **getDebugFileName**(**char** \*filename) {

time\_t t = **time**(NULL);

**struct** tm tm = \***localtime**(&t);

sprintf(filename, "Debug%02d%02d%04d.txt", tm.tm\_mon + 1, tm.tm\_mday, tm.tm\_year + 1900);

}

/\* ----------------------------------------------------------------

\* writeDebugLog

\* -write debug log

\* ----------------------------------------------------------------

\*/

**void** **writeDebugLog**(**const** **char** \*message) {

**char** filename[20];

getDebugFileName(filename); // e.g. "DebugMMDDYYYY.txt"

FILE \*file = **fopen**(filename, "a"); // Open in append mode

**if** (file == NULL) {

**perror**("Error opening debug file");

**return**;

}

**fprintf**(file, "%s\n", message); // Write the debug message

**fclose**(file);

}

/\* ----------------------------------------------------------------

\* logBoardState

\* -Show board state for each time player drop the disc and write into text file

\* -Write into debug log for the board in the text file that is already open

\* -With address and address of player disc.

\* ----------------------------------------------------------------

\*/

**void** **logBoardState**(**char** \*\*board, **int** r, **int** c) {

**char** logMessage[DEBUG\_LOG\_SIZE];

sprintf(logMessage, "\nCurrent Board State:\n");

writeDebugLog(logMessage);

**for** (**int** i = 0; i < r; i++) {

sprintf(logMessage, "%2d |", i);

**for** (**int** j = 0; j < c; j++) {

sprintf(logMessage + **strlen**(logMessage), " %c |", board[i][j]);

}

writeDebugLog(logMessage);

}

sprintf(logMessage, " ");

**for** (**int** j = 0; j < c; j++) {

sprintf(logMessage + **strlen**(logMessage), " %d ", j);

}

writeDebugLog(logMessage);

// Log raw memory addresses (in the debug file)

writeDebugLog("\nDEBUG: Board Memory Layout (Address and Values)\n");

**for** (**int** i = 0; i < r; i++) {

**for** (**int** j = 0; j < c; j++) {

sprintf(logMessage, "Row %d, Col %d -> Address: %p, Value: '%c'",

i, j, (**void**\*)&board[i][j], board[i][j]);

writeDebugLog(logMessage);

}

}

}

/\* ----------------------------------------------------------------

\* displayBoard

\* -Display the board in the console.

\* ----------------------------------------------------------------

\*/

**void** **displayBoard**(**char** \*\*board, **int** r, **int** c) {

**system**(CLEAR\_SCREEN); // Clear screen before displaying the board

**printf**(" ");

**for** (**int** j = 0; j < c; j++) {

**printf**(" %d ", j);

}

**printf**("\n");

**for** (**int** i = 0; i < r; i++) {

**printf**("%2d |", i);

**for** (**int** j = 0; j < c; j++) {

**printf**(" %c |", board[i][j]);

}

**printf**("\n");

}

**printf**(" ");

**for** (**int** j = 0; j < c; j++) {

**printf**(" %d ", j);

}

**printf**("\n");

}

/\* ----------------------------------------------------------------

\* Exit()

\* -Exit the game when user hit e

\* ----------------------------------------------------------------

\*/

**void** **Exit**() {

**printf**("Exiting the game...\n");

**#ifdef** \_\_APPLE\_\_

**system**("osascript -e 'tell application \"Terminal\" to close first window' &");

**#endif**

}

/\* ----------------------------------------------------------------

\* Welcome

\* -Welome player and do the setup of the board and setting up the player symbol x and o

\* ----------------------------------------------------------------

\*/

**void** **Welcome**(**char** \*\*\*board, **int** \*r, **int** \*c, **char** \*\*players) {

// Print out a message about initializing the board

**printf**("\nInitializing 8x8 board for 2 players (x and o)...\n");

\*r = 8;

\*c = 8;

\*board = (**char** \*\*)**malloc**((\*r) \* **sizeof**(**char** \*));

**if** (\*board == NULL) {

**printf**("Memory allocation failed for board.\n");

**exit**(1);

}

**for** (**int** i = 0; i < \*r; i++) {

(\*board)[i] = (**char** \*)**malloc**((\*c) \* **sizeof**(**char**)); // avoid using malloc memory allocation using fix array size

**if** ((\*board)[i] == NULL) {

**printf**("Memory allocation failed for board row %d.\n", i);

**exit**(1);

}

}

**for** (**int** i = 0; i < \*r; i++) {

**for** (**int** j = 0; j < \*c; j++) {

(\*board)[i][j] = ' ';

}

}

\*players = (**char** \*)**malloc**(2 \* **sizeof**(**char**));

**if** (\*players == NULL) {

**printf**("Memory allocation failed for players array.\n");

**exit**(1);

}

(\*players)[0] = 'x';

(\*players)[1] = 'o';

}

/\* ----------------------------------------------------------------

\* checkWinner

\* -Check winner horizontal, vertical and diagonally

\* ----------------------------------------------------------------

\*/

**int** **checkWinner**(**char** \*\*board, **int** r, **int** c, **int** row, **int** col, **char** disc) {

**int** count;

// Check horizontally

count = 0;

**for** (**int** j = 0; j < c; j++) {

**if** (board[row][j] == disc) {

count++;

**if** (count == 4) **return** 1;

} **else** {

count = 0;

}

}

// Check vertically

count = 0;

**for** (**int** i = 0; i < r; i++) {

**if** (board[i][col] == disc) {

count++;

**if** (count == 4) **return** 1;

} **else** {

count = 0;

}

}

// Check diagonally (\)

count = 0;

{

**int** i = row, j = col;

**while** (i >= 0 && j >= 0 && board[i][j] == disc) { i--; j--; }

i++; j++;

**while** (i < r && j < c && board[i][j] == disc) {

count++;

i++; j++;

**if** (count == 4) **return** 1;

}

}

// Check diagonally (/)

count = 0;

{

**int** i = row, j = col;

**while** (i >= 0 && j < c && board[i][j] == disc) { i--; j++; }

i++; j--;

**while** (i < r && j >= 0 && board[i][j] == disc) {

count++;

i++; j--;

**if** (count == 4) **return** 1;

}

}

**return** 0;

}

/\* ----------------------------------------------------------------

\* spawnChallenges

\* - Places 'numTraps' traps ('T') and 'numWormholes' wormholes ('W')

\* in random positions on the board, excluding row 0.

\* - Board cells must be ' ' (empty).

\* ----------------------------------------------------------------

\*/

**void** **spawnChallenges**(**char** \*\*board, **int** r, **int** c, **int** numTraps, **int** numWormholes) {

// Place 'T' for each trap

**for** (**int** i = 0; i < numTraps; i++) {

**int** row, col;

**do** {

row = **rand**() % r;

col = **rand**() % c;

} **while** (row == 0 || board[row][col] != ' ');

board[row][col] = 'T';

}

// Place 'W' for each wormhole ensuring they are in different rows and columns

**int** usedRows[r];

**int** usedCols[c];

memset(usedRows, 0, **sizeof**(usedRows));

memset(usedCols, 0, **sizeof**(usedCols));

**for** (**int** i = 0; i < numWormholes; i++) {

**int** row, col;

**do** {

row = **rand**() % r;

col = **rand**() % c;

} **while** (row == 0 || board[row][col] != ' ' || usedRows[row] || usedCols[col]);

board[row][col] = 'W';

usedRows[row] = 1;

usedCols[col] = 1;

}

}

/\* ----------------------------------------------------------------

\* setChallenge

\*

\* - Set challenge of the wormhole and trap and write into debug file

\*

\* ----------------------------------------------------------------

\*/

**void** **setChallenge**(**char** \*\*board, **int** r, **int** c, **char** symbol, **int** \*outRow, **int** \*outCol) {

**int** row, col;

**do** {

row = **rand**() % r;

col = **rand**() % c;

} **while** (board[row][col] != ' ');

\*outRow = row;

\*outCol = col;

board[row][col] = symbol;

**char** debugMessage[DEBUG\_LOG\_SIZE];

sprintf(debugMessage, "DEBUG: '%c' placed at row %d, column %d (board[%d][%d])",

symbol, \*outRow, \*outCol, row, col);

writeDebugLog(debugMessage);

}

/\* ----------------------------------------------------------------

\* teleportDisc

\*

\* - Telelporting player disc if the disc is landing on the W for teleporting.

\*

\* - Teleporting to random row and col with delay and banner and log that in to debug file

\*

\* ----------------------------------------------------------------

\*/

**void** **teleportDisc**(**char** \*\*board, **int** r, **int** c, **int** \*row, **int** \*col, **char** disc) {

**int** newRow, newCol;

**int** maxAttempts = r \* c, attempts = 0;

**int** playerNumber = (disc == 'x') ? 1 : 2; // Determine player number

// Display teleportation banner and clear screen

**system**(CLEAR\_SCREEN);

**printf**("\n=============================================\n");

**printf**(" Teleporting Player %d's disc now... \n", playerNumber);

**printf**("=============================================\n");

SLEEP\_MS(1000); // 1000ms delay

**do** {

newRow = **rand**() % r;

newCol = **rand**() % c;

attempts++;

} **while** (board[newRow][newCol] != ' ' && attempts < maxAttempts);

**if** (board[newRow][newCol] == ' ') {

board[newRow][newCol] = disc;

\*row = newRow;

\*col = newCol;

// Log teleportation event

**char** debugMessage[DEBUG\_LOG\_SIZE];

sprintf(debugMessage, "DEBUG: Player %d's Disc teleported to Row %d, Column %d\n", playerNumber, \*row, \*col);

writeDebugLog(debugMessage);

}

// Show the current board after teleportation

displayBoard(board, r, c);

}

/\* ----------------------------------------------------------------

\* logPlayerDiscAddresses

\*

\* - Log player address for debug purposes.

\*

\* ----------------------------------------------------------------

\*/

**void** **logPlayerDiscAddresses**(**char** \*\*board, **int** r, **int** c) {

**char** filename[20];

getDebugFileName(filename);

FILE \*file = **fopen**(filename, "a");

**if** (file == NULL) {

**perror**("Error opening debug file");

**return**;

}

**fprintf**(file, "\nDEBUG: Player Disc Memory Addresses\n");

**for** (**int** i = 0; i < r; i++) {

**for** (**int** j = 0; j < c; j++) {

**if** (board[i][j] == 'x' || board[i][j] == 'o') {

**fprintf**(file, "Row %d, Col %d -> Address: %p, Value: '%c'\n",

i, j, (**void**\*)&board[i][j], board[i][j]);

}

}

}

**fclose**(file);

}

/\* ----------------------------------------------------------------

\* handleTrapLogic

\*

\* - To handle trap when player disc is landing on the T

\*

\* - Showing banner

\*

\* - Trap other player disc by searching the board for player x or o and trap them with perspectively player turn.

\*

\* - Log the trap logic into text file

\*

\* ----------------------------------------------------------------

\*/

**void** **handleTrapLogic**(**char** \*\*board, **int** r, **int** c, **int** row, **int** col, **char** disc) {

**char** opponentDisc = (disc == 'x') ? 'o' : 'x';

**int** placed = 0;

**int** playerNumber = (disc == 'x') ? 1 : 2; // Determine player number

// Display trap banner and clear screen

**system**(CLEAR\_SCREEN);

**printf**("\n=============================================\n");

**printf**(" Oh, Oh! Player %d landed on a TRAP! \n", playerNumber);

**printf**("=============================================\n");

SLEEP\_MS(1000); // 1000ms delay

// Log trap activation

**char** filename[20];

getDebugFileName(filename);

FILE \*file = **fopen**(filename, "a");

**if** (file != NULL) {

**fprintf**(file, "DEBUG: Player %d (%c) landed on a Trap at Row %d, Col %d\n", playerNumber, disc, row, col);

}

// Search entire row for opponent's discs and place player's disc on top of each

**for** (**int** j = 0; j < c; j++) {

**for** (**int** i = 0; i < r; i++) {

**if** (board[i][j] == opponentDisc && i > 0 && board[i - 1][j] == ' ') {

board[i - 1][j] = disc;

placed = 1;

**if** (file != NULL) {

**fprintf**(file, "DEBUG: Disc placed at Row %d, Col %d (above opponent at Row %d)\n", i - 1, j, i);

}

}

}

}

// Search entire column for opponent's discs and place player's disc on top of each

**for** (**int** i = 0; i < r; i++) {

**if** (board[i][col] == opponentDisc && i > 0 && board[i - 1][col] == ' ') {

board[i - 1][col] = disc;

placed = 1;

**if** (file != NULL) {

**fprintf**(file, "DEBUG: Disc placed at Row %d, Col %d (above opponent at Row %d)\n", i - 1, col, i);

}

}

}

// If no opponent disc was found, place at the trap location

**if** (!placed) {

board[row][col] = disc;

**if** (file != NULL) {

**fprintf**(file, "DEBUG: No opponent found, placed at trap location Row %d, Col %d\n", row, col);

}

}

// Remove the trap after it's used

board[row][col] = ' ';

// Close the debug file

**if** (file != NULL) {

**fclose**(file);

}

// Show the current board after trap logic

displayBoard(board, r, c);

}

**void** **recordWinner**(**int** playerNumber, **char** disc) {

**const** **char** \*filePath = "/Users/admin/Desktop/Compe160-271/Connect4Wormhole/winner\_record.txt";

FILE \*file = **fopen**(filePath, "a");

**if** (file == NULL) {

**perror**("Error opening file");

**return**;

}

time\_t rawtime;

**struct** tm \* timeinfo;// avoid using struct

**char** timeStr[80];

**time**(&rawtime);

timeinfo = **localtime**(&rawtime);

**strftime**(timeStr, **sizeof**(timeStr), "%Y-%m-%d %H:%M:%S", timeinfo);

**fprintf**(file, "Player %d (%c) won the game at %s\n", playerNumber, disc, timeStr);

**fclose**(file);

}

/\*\*

\* snakeAttack - Permanently removes two discs from the current player

\* (either 'x' or 'o') and displays only the current board.

\*

\* board, r, c: The game board & dimensions

\* disc: The current player's disc ('x' or 'o')

\*

\* Steps:

\* 1) Gather all positions of this player's discs.

\* 2) If < 2 discs exist, do nothing (just return).

\* 3) Otherwise pick 2 random discs, remove them from the board (set ' ').

\* 4) Clear screen, display the updated board once, show snake ASCII, and wait 3s.

\* 5) Discs are NOT restored.

\*/

**void** **snakeAttack**(**char** \*\*board, **int** r, **int** c, **char** disc) {

// 1) Collect the player's disc positions

**int** positions[1000][2];

**int** count = 0;

**for** (**int** i = 0; i < r; i++) {

**for** (**int** j = 0; j < c; j++) {

**if** (board[i][j] == disc) {

positions[count][0] = i;

positions[count][1] = j;

count++;

}

}

}

// If fewer than 2 discs => can't remove 2, do nothing

**if** (count < 2) {

**return**;

}

// 2) Pick two distinct random positions from that player's discs

**int** idx1 = **rand**() % count;

**int** idx2;

**do** {

idx2 = **rand**() % count;

} **while** (idx2 == idx1);

// row/col of the two discs

**int** row1 = positions[idx1][0];

**int** col1 = positions[idx1][1];

**int** row2 = positions[idx2][0];

**int** col2 = positions[idx2][1];

// 3) Permanently remove them

board[row1][col1] = ' ';

board[row2][col2] = ' ';

// 4) Clear screen and display updated board ONCE

**#ifdef** \_WIN32

system("cls");

**#else**

**system**("clear");

**#endif**

**printf**("The Snake is about to grab TWO of Player %c's discs!\n\n", disc);

displayBoard(board, r, c); // Show only the current board with discs removed

// 5) Print snake ASCII

**printf**("\nThe Snake Slithers In and Strikes!!!\n\n");

**printf**(" \_\_,\_\_\n"

" .--. .--.\n"

" / \\ \\ \n"

"| >\\/7 |\n"

"| >\\/7 ,--'\n"

" \\ / /\n"

" `--' `--' \n"

" || || \n"

" || || \n"

" ~~ ~~ \n\n");

**fflush**(stdout);

// 6) Wait ~3 seconds

SLEEP\_MS(3000);

// 7) Do NOT restore discs => they remain gone

// No additional board display here; the game loop will handle it

}

/\* ------------------------------------------------------

\* MainGame

\* - Plays the Connect4 + Wormhole + Trap variant.

\* - startingPlayer => 0 => P1, 1 => P2

\* - Ensures only the current board is displayed after each action.

\* ------------------------------------------------------

\*/

**void** **MainGame**(**char** \*\*board, **int** r, **int** c, **char** \*players, **int** startingPlayer) {

**int** currentPlayer = startingPlayer;

**char** disc;

**int** col, row;

**char** input[10];

**int** gameOver = 0;

time\_t gameStartTime = **time**(NULL); // Record game start time

// Initialize first random snake interval (10-30 seconds)

**if** (lastSnakeTime == 0) {

lastSnakeTime = gameStartTime;

nextSnakeInterval = (**rand**() % 21) + 10; // Random between 10 and 30 seconds

}

**while** (!gameOver) {

// Check for snake attack based on elapsed time

time\_t currentTime = **time**(NULL);

**if** (snakeCount < 3 && (currentTime - lastSnakeTime) >= nextSnakeInterval) {

snakeCount++;

disc = players[currentPlayer];

snakeAttack(board, r, c, disc); // Snake updates and shows current board

// Reset for next snake attack

lastSnakeTime = currentTime;

nextSnakeInterval = (**rand**() % 21) + 10; // New random interval (10-30s)

// Optional: Log the event

**char** debugMessage[DEBUG\_LOG\_SIZE];

sprintf(debugMessage, "DEBUG: Snake attacked at %ld seconds into game",

currentTime - gameStartTime);

writeDebugLog(debugMessage);

// Skip the regular board display this turn since snakeAttack already showed it

**continue**;

}

// Log board state (optional, for debugging)

logBoardState(board, r, c);

logPlayerDiscAddresses(board, r, c);

// Display the current board for player turn

displayBoard(board, r, c);

// Prompt for player's action

disc = players[currentPlayer];

**printf**("Player %d (%c), enter column (0-%d) to drop your disc, or 'e' to quit: ",

currentPlayer + 1, disc, c - 1);

**scanf**("%s", input);

// Check if user wants to exit

**if** (**strcmp**(input, "e") == 0) {

gameOver = 1;

**printf**("Player %d (%c) has exited the game.\n", currentPlayer + 1, disc);

**continue**;

}

// Otherwise, assume user typed a column number

col = **atoi**(input);

**if** (col < 0 || col >= c) {

**printf**("Invalid column! Choose between 0 and %d.\n", c - 1);

**continue**;

}

// Find where to place the disc (from bottom up)

row = -1;

**for** (**int** i = r - 1; i >= 0; i--) {

**if** (board[i][col] == ' ' || board[i][col] == 'T' || board[i][col] == 'W') {

row = i;

**break**;

}

}

**if** (row == -1) {

**printf**("Column %d is full! Try again.\n", col);

**continue**;

}

// Handle traps correctly

**if** (board[row][col] == 'T') {

handleTrapLogic(board, r, c, row, col, disc);

}

// If there's a wormhole, teleport disc

**else** **if** (board[row][col] == 'W') {

board[row][col] = ' '; // remove wormhole

teleportDisc(board, r, c, &row, &col, disc);

}

// Otherwise, place disc normally

**else** {

board[row][col] = disc;

}

// Check if the move results in a win

**if** (checkWinner(board, r, c, row, col, disc)) {

**system**(CLEAR\_SCREEN); // Clear screen before showing win message

**printf**("Player %d (%c) wins!\n", currentPlayer + 1, disc);

recordWinner(currentPlayer + 1, disc);

displayBoard(board, r, c); // Show final board state

gameOver = 1;

}

// Switch players if not game over

**if** (!gameOver) {

currentPlayer = (currentPlayer + 1) % 2;

}

}

// Final board state (already shown if a win occurred)

logBoardState(board, r, c);

// Free memory

**for** (**int** i = 0; i < r; i++) {

**free**(board[i]);

}

**free**(board);

**free**(players); // avoid using free

**printf**("Thank you for playing Wormhole Connect4!\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SAN DIEGO STATE UNIVERISTY \*

\* DUC M LE 132485155 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-------------------------- END-OF-HEADER -----------------------------

\*

\* Header.h

\*

\* Created on: Oct 16, 2024

\* Author: Duc Le

\*/

**#ifndef** HEADER\_H\_

**#define** HEADER\_H\_

// Function Prototypes

**void** **ShowMenu**(**void**);

**void** **Exit**();

**void** **Welcome**(**char** \*\*\*board, **int** \*r, **int** \*c, **char** \*\*players);

**void** **MainGame**(**char** \*\*board, **int** r, **int** c, **char** \*players, **int** startingPlayer);

**int** **checkWinner**(**char** \*\*board, **int** r, **int** c, **int** row, **int** col, **char** disc);

**void** **setChallenge**(**char** \*\*board, **int** r, **int** c, **char** symbol, **int** \*outRow, **int** \*outCol);

**void** **teleportDisc**(**char** \*\*board, **int** r, **int** c, **int** \*row, **int** \*col, **char** disc);

**void** **recordWinner**(**int** playerNumber, **char** disc);

**void** **spawnChallenges**(**char** \*\*board, **int** r, **int** c, **int** numTraps, **int** numWormholes);

**void** **handleTrapLogic**(**char** \*\*board, **int** r, **int** c, **int** row, **int** col, **char** disc);

**void** **getDebugFileName**(**char** \*filename);

**void** **writeDebugLog**(**const** **char** \*message);

**void** **logBoardState**(**char** \*\*board, **int** r, **int** c);

**int** **coinFlipAnimation**(**void**);

**void** **logPlayerDiscAddresses**(**char** \*\*board, **int** r, **int** c);

**void** **snakeAttack**(**char** \*\*board, **int** r, **int** c, **char** disc);

**void** **displayBoard**(**char** \*\*board, **int** r, **int** c);

// End Function Prototypes

// Prototypes

**#define** DEBUG\_LOG\_SIZE 1024

// custom Clear Screen

**#ifdef** \_WIN32

**#include** <windows.h> // For Sleep(ms) on Windows

**#define** CLEAR\_SCREEN "cls"

**#define** SLEEP\_MS(ms) Sleep(ms)

**#else**

**#define** CLEAR\_SCREEN "clear"

**#define** SLEEP\_MS(ms) usleep((ms) \* 1000)

**#endif**

// Custom Global Declaration

**static** time\_t lastSnakeTime = 0; // Time of the last snake attack

**static** **int** nextSnakeInterval = 0; // Seconds until next snake attack

**static** **int** snakeCount = 0; // Existing variable to limit snake appearances

// End Prototypes

**#endif** /\* HEADER\_H\_ \*/

**A black rectangle with white text

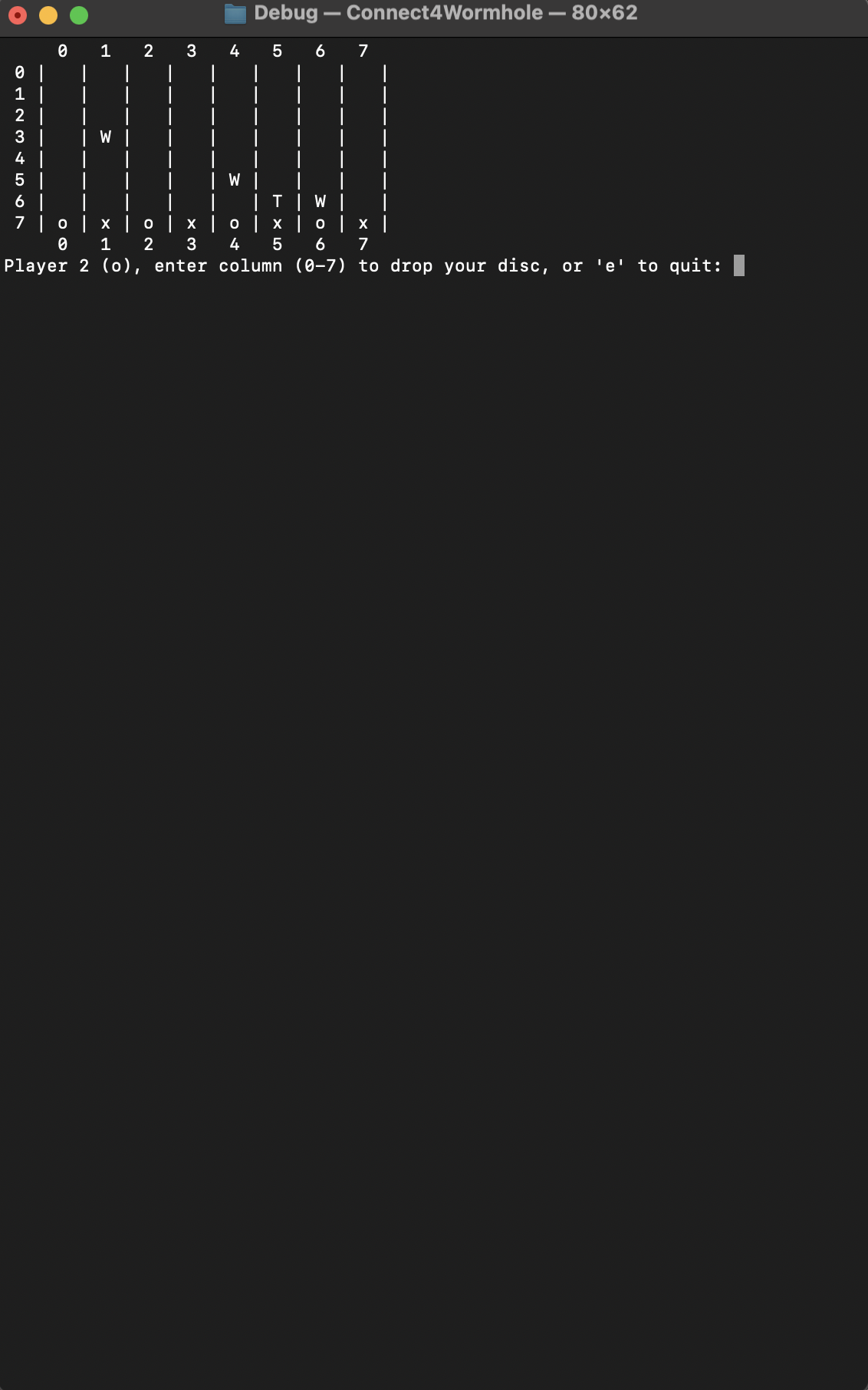
AI-generated content may be incorrect.Screenshot of C Game**

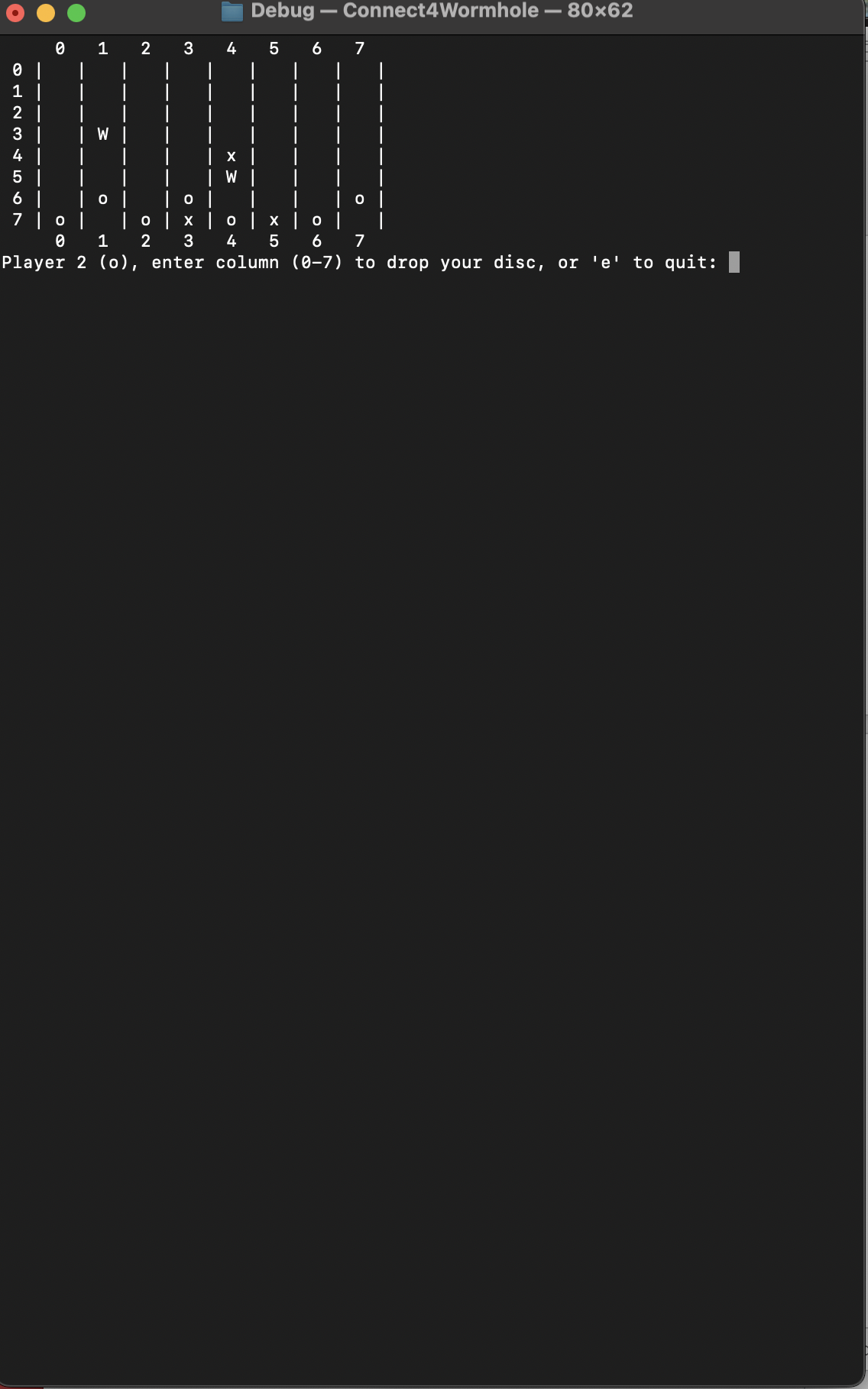
**A screenshot of a computer

AI-generated content may be incorrect.A black rectangular object with white text

AI-generated content may be incorrect.**

**A black rectangular object with a white border

AI-generated content may be incorrect.**

**A black rectangular object with white text

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

Task Report

Debug: Jan 22, 2025. For now, there are bugs with the address misplaced when the player places the disc in a certain number for the row to give an error. It is from the set trap and set wormhole function that I set to have multiple traps per row and column.

Debug Jan 23, 2025: The Trap and Wormhole should not be at index 0 of the array, which makes the row act complete when the player can’t drop the disc. Fix it, have the row -1 and the column -1

Debug: Jan 27, 2025: Finding out that the trap and wormhole did not function as expected, finding solution for it, might be caused by the addressing ram that could not let the function work correctly, or might be because of indexing -1

Debug: Jan 30, 2025: I discovered that the address is not set for T and W. When setting the trap, I thought it said it was activated at row 151587338, col 707406378. Just a column error. It should start from 0 to 7, not 1 to 8. Now it works well to go to the trap in the correct format, to where it should be. Now, to check for the logic function that the trap and wormhole are supposed to be.

Debug: Feb 6, 2025. Add the logic trap function and the head and tail player starter.

Debug: Feb 12, 2025: Implemented coin flip function, and every function seems to be working fine. Now, a snake as a little cue, to have an animation of a long cue moving and taking away the player disc. Winer has been recorded as a text file, and the debugging has been done for each memory, which also shows the array.

Debug: Feb 16, 2025. The trap function is not working as expected, but there is an address for each player disc. However, the address changes since using the function free in C programming changes each time. If able to manipulate the address to where it is searching the player disc, place an opponent disc on it for the whole row and column using the search function. Will be able to add a snake to take away the player disc randomly. The professor wants us to use the pointer array, not struct or malloc. Wanna asked him about that?

Debug: Feb 17, 2025. I figured out how to get the trap to work for searching the whole row and column, and I also added an invisible trap and wormhole so that if a player lands on it, it will have a banner for trapping and teleporting. Might add the snake to spawn randomly to take

Debug: Feb 18, 2025. Added a snake spawning function that takes away two discs of a player. It needs to be changed since it cannot do a time interval background for the spawning.

Debug Feb 27, 2025. Implementing it into MIPS to figure out the project, such as implementing seeding to generate a random number for the flip coin to give the player a fair chance to play. Although it is progress, it should work as a bare minimum. I also included the MIPS code for work as the backbone of the progress provided in the screenshot.

Debug March 16, 2025. Adding Trap and Wormhole to the board, and it is invisible to make the game look interesting. If either player lands on either wormhole or trap, it has interesting logic to transport and trap. Since the snake takes away the player’s disc, I have not implemented it yet, but I will add it to the game, and the check winner already has a function to check the winner board in multiple directions. I hope the game is turning out to be good and exciting for the players to play.

Debug March 24, 2025. Adding macro\_file for the sound and bitmap.

Debug March 25, 2025. Adding a block to cover the trap and wormhole instead of the invisible. It makes more games more fun rather than a boring way of doing things. Also, implement the welcome and spawn challenge to spawn random boards for wormhole and trap and random exit. Complete the snake attack to align with the C program. Update teleporting disc to have banner and trap to have banner as well. Also before exiting prompt user if they want to play again or exit using recursive.

Debug March 27, 2025. We need to fix the diagonal when four pieces are detected. It seems that it does not detect four pieces.

Debug: April 4, 2025. Fix the game and everything is final good to goes.

MIPS CODE

Requirement:

* **Conditional Branches (if-else Equivalents)**
* **Use of a 2-D Array**
* **Subroutine Call Depth (At Least 3 Levels)**
* **Use of the Stack for Register Storage**
* **Additional Features and Notes**
* **Use of the stack function**

CODE

#/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* SAN DIEGO STATE UNIVERSITY \*

#\* DUC M LE 132485155

#\* 44 55 43 20 4D 49 4E 48 20 4C 45 0A

#\* 31 33 32 34 38 35 31 35 35 0A \*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* SETUP BOARD PLAYER

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

board: .space 64 # 8x8 grid

rows: .word 8 # Number of rows

cols: .word 8 # Number of columns

players: .byte 'x', 'o' # Player 1 = 'x', Player 2 = 'o'

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* MESSAGE FEED BACK

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

welcome\_msg: .asciiz "=======================================================\n WELCOME TO CONNECT 4 TRAPWORMHOLE! \n=======================================================\n"

wormhole\_msg: .asciiz "Wormhole activated! Disc teleported.\n"

trap\_msg: .asciiz "Trap activated! Opponent's disc trapped.\n"

win\_msg: .asciiz " wins! Game Over.\n"

invalid\_msg: .asciiz "Invalid choice. Please enter 1 or 2.\n"

exit\_msg: .asciiz "Exiting the program. Goodbye!\n"

invalid\_col: .asciiz "Invalid column number! Try again.\n"

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* PROMPT USER

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

menu\_prompt: .asciiz "Menu:\n 1) Start Game\n 2) Exit\n"

player\_prompt:.asciiz "Player "

enter\_col: .asciiz ", enter column (0-7) to drop your disc, or 'e' to quit: "

get\_userinput:.asciiz "Please enter number to start game: "

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* FLIPPING COIN RESULT

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

heads\_result: .asciiz "Heads! Player 1 goes first.\n"

tails\_result: .asciiz "Tails! Player 2 goes first.\n"

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* ROW AND COL SPACING

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

newline: .asciiz "\n"

space: .asciiz " " # 4 spaces for wider spacing

space\_two: .asciiz " " # 5 spaces for centering header numbers

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* TRAP AND WORMHOLE POSITION

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

wormhole1\_entry: .word 2, 3 # Row 2, Col 3

wormhole1\_exit: .word 5, 3 # Row 5, Col 3

wormhole2\_entry: .word 1, 5 # Row 1, Col 5

wormhole2\_exit: .word 6, 2 # Row 6, Col 2

wormhole3\_entry: .word 4, 1 # Row 4, Col 1

wormhole3\_exit: .word 3, 6 # Row 3, Col 6

trap\_pos: .word 3, 4 # Row 3, Col 4

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* MISC MESSAGE

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

debug\_msg: .asciiz "DEBUG: Checking win at position (row,col): "

comma\_msg: .asciiz ","

debug\_rand: .asciiz "DEBUG: Random value = "

buffer: .space 10 # Input buffer

trap\_used: .word 0 # Flag to track if trap has been used (0 = not used, 1 = used)

debug\_msg\_t0\_init: .asciiz "DEBUG: $t0 after initialization: "

debug\_msg\_t0\_loop: .asciiz "DEBUG: $t0 in find\_spot\_loop: "

debug\_msg\_t0\_after\_wormhole: .asciiz "DEBUG: $t0 after checkWormholesAndTrap: "

debug\_msg\_a3: .asciiz "DEBUG: $a3 value: "

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* BIT MAP CONSTANTS

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Bitmap constants

.eqv WIDTH 32 # 256 / 8 = 32 pixels wide

.eqv HEIGHT 32 # 256 / 8 = 32 pixels high

.eqv BLACK 0x00000000

.eqv YELLOW 0x00FFFF00 # For coin edge and flipping states

.eqv SILVER 0x00C0C0C0 # For coin face

.eqv RED 0x00FF0000 # For 'H'

.eqv GREEN 0x0000FF00 # For 'T'

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* INCLUDE FILES

.include "macro\_file.asm" # Coin flip and utility macros

.include "MIDImacro\_file.asm" # MIDI out macro

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* MAIN FUNCTION

main:

jal ShowMenu # jump a link into ShowMenu

li $v0, 10

syscall

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* SHOW MENU PROCEDURE

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ShowMenu:

addi $sp, $sp, -4

sw $ra, 0($sp)

show\_menu\_loop:

li $v0, 4

la $a0, welcome\_msg

syscall

li $v0, 4

la $a0, menu\_prompt

syscall

li $v0, 4

la $a0, get\_userinput

syscall

li $v0, 5

syscall

move $t0, $v0

beq $t0, 1, start\_game

beq $t0, 2, exit\_program

li $v0, 4

la $a0, invalid\_msg

syscall

j show\_menu\_loop

start\_game:

# Initialize board

la $t1, board

li $t2, 64

li $t3, ' '

init\_board\_loop:

sb $t3, 0($t1)

addi $t1, $t1, 1

addi $t2, $t2, -1

bnez $t2, init\_board\_loop

jal coinFlipAnimation

move $s0, $v0 # $s0 holds starting player (1 for Player 1, 2 for Player 2)

jal MainGame

j end\_show\_menu

exit\_program:

li $v0, 4

la $a0, exit\_msg

syscall

li $v0, 10

syscall

end\_show\_menu:

lw $ra, 0($sp)

addi $sp, $sp, 4

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* COIN FLIP ANIMATION PROCEDURE

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

coinFlipAnimation:

addi $sp, $sp, -8

sw $ra, 0($sp)

sw $s0, 4($sp)

li $v0, 30

syscall

move $a1, $a0

li $v0, 40

li $a0, 1

syscall

li $s0, 5

flip\_loop:

li $v0, 42

li $a0, 1

li $a1, 2

syscall

move $t1, $a0

beqz $t1, draw\_tails\_loop

jal drawHeads

j flip\_continue

draw\_tails\_loop:

jal drawTails

flip\_continue:

delay(500000)

jal drawEdgeVertical

delay(500000)

jal drawTails

delay(500000)

jal drawEdgeHorizontal

delay(500000)

jal drawHeads

delay(500000)

addi $s0, $s0, -1

bnez $s0, flip\_loop

li $v0, 30

syscall

move $a1, $a0

li $v0, 40

li $a0, 1

syscall

li $v0, 42

li $a0, 1

li $a1, 2

syscall

move $t1, $a0

beqz $t1, final\_tails

jal drawHeads

li $v0, 4

la $a0, heads\_result

syscall

li $v0, 1

j end\_coin\_flip

final\_tails:

jal drawTails

li $v0, 4

la $a0, tails\_result

syscall

li $v0, 2

end\_coin\_flip:

lw $ra, 0($sp)

lw $s0, 4($sp)

addi $sp, $sp, 8

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* MAIN GAME PROCEDURE

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MainGame:

addi $sp, $sp, -24

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

sw $s2, 12($sp)

sw $s3, 16($sp)

sw $s4, 20($sp)

la $a0, board

lw $a1, rows

lw $a2, cols

la $a3, players

move $s4, $a3

move $s1, $s0 # $s1 = current player (1 or 2)

subi $s1, $s1, 1 # Adjust to 0-based index for player array

game\_loop:

jal displayBoard

li $v0, 4

la $a0, player\_prompt

syscall

li $v0, 1

addi $a0, $s1, 1 # Display player number (1 or 2)

syscall

li $v0, 11

addu $t2, $s4, $s1

lb $a0, 0($t2) # Display player symbol ('x' or 'o')

syscall

li $v0, 4

la $a0, enter\_col

syscall

li $v0, 8

la $a0, buffer

li $a1, 10

syscall

lb $t0, buffer

beq $t0, 'e', end\_game

subi $t0, $t0, '0'

blt $t0, 0, invalid\_input

bge $t0, 8, invalid\_input

move $s2, $t0 # $s2 = column

li $t0, 7 # Start from bottom row

find\_spot\_loop:

mul $t1, $t0, 8

add $t1, $t1, $s2

la $t2, board

add $t2, $t2, $t1

lb $t3, 0($t2)

beq $t3, ' ', place\_disc

addi $t0, $t0, -1

bgez $t0, find\_spot\_loop

j invalid\_input

place\_disc:

addu $t4, $s4, $s1

lb $t4, 0($t4) # Load player symbol

sb $t4, 0($t2) # Place disc

move $s3, $t0 # Save row in $s3

move $a0, $t0 # Row

move $a1, $s2 # Column

move $a2, $t4 # Player's piece

jal checkWormholesAndTrap

beq $v0, 1, wormhole\_handled

beq $v0, 2, trap\_handled

# jal displayBoard #from this

# li $v0, 4

# la $a0, newline

# syscall

# li $v0, 4

# la $a0, debug\_msg

# syscall

# li $v0, 1

# move $a0, $s3

# syscall # the commend part is for debug purposes

# li $v0, 4

# la $a0, comma\_msg

# syscall

# li $v0, 1

# move $a0, $s2

# syscall

# li $v0, 4

# la $a0, newline

# syscall #To This

move $a0, $s3

move $a1, $s2

jal checkWinner

beqz $v0, continue\_game

jal displayBoard # Display the final board state

li $v0, 4

la $a0, player\_prompt

syscall

li $v0, 1

move $a0, $v1 # Winning player number

syscall

li $v0, 11

addu $t2, $s4, $s1

lb $a0, 0($t2) # Winning player symbol

syscall

li $v0, 4

la $a0, newline

syscall

li $v0, 4

la $a0, win\_msg

syscall

play\_victory\_tune #sound for the winner declare

j end\_game

continue\_game:

addi $s1, $s1, 1

andi $s1, $s1, 1 # Toggle between 0 and 1

j game\_loop

wormhole\_handled:

addi $s1, $s1, 1

andi $s1, $s1, 1

j game\_loop

trap\_handled:

addi $s1, $s1, 1

andi $s1, $s1, 1

j game\_loop

invalid\_input:

li $v0, 4

la $a0, invalid\_col

syscall

j game\_loop

end\_game:

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

lw $s2, 12($sp)

lw $s3, 16($sp)

lw $s4, 20($sp)

addi $sp, $sp, 24

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* DISPLAY BOARD PROCEDURE

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

displayBoard:

addi $sp, $sp, -4

sw $ra, 0($sp)

li $t0, 0

col\_header\_loop:

li $v0, 4

la $a0, space\_two

syscall

li $v0, 1

move $a0, $t0

syscall

li $v0, 4

la $a0, space

syscall

addi $t0, $t0, 1

blt $t0, 8, col\_header\_loop

li $v0, 4

la $a0, newline

syscall

li $t0, 0

row\_loop:

li $v0, 1

move $a0, $t0

syscall

li $v0, 11

li $a0, '|'

syscall

li $t1, 0

col\_loop:

li $v0, 4

la $a0, space

syscall

mul $t2, $t0, 8

add $t2, $t2, $t1

la $t3, board

add $t3, $t3, $t2

lb $a0, 0($t3)

li $v0, 11

syscall

li $v0, 4

la $a0, space

syscall

li $v0, 11

li $a0, '|'

syscall

addi $t1, $t1, 1

blt $t1, 8, col\_loop

li $v0, 4

la $a0, newline

syscall

addi $t0, $t0, 1

blt $t0, 8, row\_loop

lw $ra, 0($sp)

addi $sp, $sp, 4

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* CHECK WORMHOLE AND TRAP PROCEDURE

#\* Inputs:

#\* $a0: Row of the move

#\* $a1: Column of the move

#\* $a2: Player's piece ('x' or 'o')

#\* Outputs:

#\* $v0: 0 (no special action), 1 (wormhole activated), 2 (trap activated)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

checkWormholesAndTrap:

addi $sp, $sp, -16

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

sw $s2, 12($sp)

move $s0, $a0

move $s1, $a1

move $s2, $a2

lw $t4, wormhole1\_entry

lw $t5, wormhole1\_entry+4

beq $a0, $t4, check\_w1\_col

j check\_wormhole2

check\_w1\_col:

beq $a1, $t5, wormhole1\_activate

j check\_wormhole2

wormhole1\_activate:

la $a0, wormhole1\_exit

jal handleWormhole

beq $v0, 1, win\_detected\_special

li $v0, 1

j end\_check\_wormholes

check\_wormhole2:

lw $t4, wormhole2\_entry

lw $t5, wormhole2\_entry+4

beq $a0, $t4, check\_w2\_col

j check\_wormhole3

check\_w2\_col:

beq $a1, $t5, wormhole2\_activate

j check\_wormhole3

wormhole2\_activate:

la $a0, wormhole2\_exit

jal handleWormhole

beq $v0, 1, win\_detected\_special

li $v0, 1

j end\_check\_wormholes

check\_wormhole3:

lw $t4, wormhole3\_entry

lw $t5, wormhole3\_entry+4

beq $a0, $t4, check\_w3\_col

j check\_trap

check\_w3\_col:

beq $a1, $t5, wormhole3\_activate

j check\_trap

wormhole3\_activate:

la $a0, wormhole3\_exit

jal handleWormhole

beq $v0, 1, win\_detected\_special

li $v0, 1

j end\_check\_wormholes

check\_trap:

lw $t4, trap\_pos

lw $t5, trap\_pos+4

beq $a0, $t4, check\_trap\_col

j no\_special

check\_trap\_col:

beq $a1, $t5, trap\_activate

j no\_special

trap\_activate:

# Call removeTrapDisc to remove the disc at the trap position

move $a0, $s0 # Row of the move

move $a1, $s1 # Column of the move

jal removeTrapDisc

# Check if the trap was activated

beq $v0, $zero, no\_special # If $v0 == 0, trap wasn't activated

# Call trapOpponentDiscs to trap opponent discs

move $a0, $s2 # Player's piece

jal trapOpponentDiscs

# Set return value to indicate trap activation

li $v0, 2

j end\_check\_wormholes

no\_special:

li $v0, 0

win\_detected\_special:

j end\_check\_wormholes

end\_check\_wormholes:

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

lw $s2, 12($sp)

addi $sp, $sp, 16

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* HANDLE WORMHOLE PROCEDURE

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

handleWormhole:

addi $sp, $sp, -12

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

mul $t1, $s0, 8

add $t1, $t1, $s1

la $t2, board

add $t2, $t2, $t1

lb $t7, 0($t2)

lw $t3, 0($a0)

lw $t4, 4($a0)

mul $t5, $t3, 8

add $t5, $t5, $t4

la $t6, board

add $t6, $t6, $t5

lb $t8, 0($t6)

bne $t8, ' ', skip\_teleport

li $t9, ' '

sb $t9, 0($t2)

sb $t7, 0($t6)

li $v0, 4

la $a0, wormhole\_msg

syscall

move $a0, $t3

move $a1, $t4

jal checkWinner

beq $v0, 1, win\_detected\_wormhole

skip\_teleport:

j end\_handle\_wormhole

win\_detected\_wormhole:

j end\_handle\_wormhole

end\_handle\_wormhole:

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

addi $sp, $sp, 12

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* REMOVE TRAP DISC PROCEDURE

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

removeTrapDisc:

# Save registers

addi $sp, $sp, -12

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

# Move arguments to saved registers

move $s0, $a0 # Row of the current move

move $s1, $a1 # Column of the current move

# Check if trap has already been used

lw $t0, trap\_used

bne $t0, $zero, end\_remove\_trap # If trap\_used != 0, skip trap logic

# Check if the current move is at the trap position (row 3, col 4)

la $t1, trap\_pos

lw $t2, 0($t1) # Load trap row (3)

lw $t3, 4($t1) # Load trap col (4)

bne $s0, $t2, end\_remove\_trap # If current row != trap row, skip

bne $s1, $t3, end\_remove\_trap # If current col != trap col, skip

# Load the player's piece at the trap position (for reference)

mul $t1, $s0, 8

add $t1, $t1, $s1

la $t2, board

add $t2, $t2, $t1

lb $t7, 0($t2) # Load player's piece ('x' or 'o')

# Remove the disc at the trap position

li $t5, ' '

sb $t5, 0($t2) # Set the trap position to space

# Mark the trap as used

li $t0, 1

sw $t0, trap\_used # Mark trap as used

# Return 1 to indicate the trap was activated

li $v0, 1

j end\_remove\_trap

end\_remove\_trap:

# If trap wasn't activated, return 0

beq $v0, 1, skip\_return\_zero

li $v0, 0

skip\_return\_zero:

# Restore registers

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

addi $sp, $sp, 12

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* TRAP OPPONENT DISCS PROCEDURE

#\* Scans the entire board (row by row, column by column) and traps

#\* every opponent disc found by placing the current player's disc

#\* above or below (if there's an empty space), without removing the opponent's disc.

#\* Inputs:

#\* $a0: Player's piece ('x' or 'o')

#\* Outputs:

#\* None (modifies the board directly)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

trapOpponentDiscs:

# Save registers

addi $sp, $sp, -12

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

# Store player's piece

move $s0, $a0 # $s0 = player's piece ('x' or 'o')

# Determine opponent's piece

li $s1, 'x' # $s1 = opponent's piece

beq $s0, 'o', opponent\_set

li $s1, 'o'

opponent\_set:

# Print trap activation message

li $v0, 4

la $a0, trap\_msg

syscall

# Initialize row counter

li $t0, 0 # $t0 = row counter

trap\_row\_loop:

li $t1, 0 # $t1 = column counter

trap\_col\_loop:

# Calculate board index: (row \* 8) + col

mul $t2, $t0, 8

add $t2, $t2, $t1

la $t3, board

add $t3, $t3, $t2 # $t3 = address of board[row][col]

lb $t4, 0($t3) # $t4 = piece at board[row][col]

# Check if the piece is the opponent's

bne $t4, $s1, skip\_trap # If not opponent's piece, skip

# Found an opponent's disc, try to trap it by placing player's disc above or below

# First, try to place player's disc above (row - 1)

blt $t0, 1, try\_below # If row == 0, can't place above, try below

# Check the spot above (row - 1, col)

subi $t5, $t0, 1 # $t5 = row - 1

mul $t6, $t5, 8

add $t6, $t6, $t1

la $t7, board

add $t7, $t7, $t6 # $t7 = address of board[row-1][col]

lb $t8, 0($t7) # $t8 = piece at board[row-1][col]

beq $t8, ' ', place\_above # If spot above is empty, place player's disc

try\_below:

# If can't place above, try below (row + 1)

bge $t0, 7, skip\_trap # If row == 7, can't place below, skip

# Check the spot below (row + 1, col)

addi $t5, $t0, 1 # $t5 = row + 1

mul $t6, $t5, 8

add $t6, $t6, $t1

la $t7, board

add $t7, $t7, $t6 # $t7 = address of board[row+1][col]

lb $t8, 0($t7) # $t8 = piece at board[row+1][col]

beq $t8, ' ', place\_below # If spot below is empty, place player's disc

j skip\_trap # If neither above nor below is empty, skip

place\_above:

# Place player's disc above

sb $s0, 0($t7) # Place player's piece at board[row-1][col]

j skip\_trap

place\_below:

# Place player's disc below

sb $s0, 0($t7) # Place player's piece at board[row+1][col]

skip\_trap:

# Move to the next column

addi $t1, $t1, 1

blt $t1, 8, trap\_col\_loop

# Move to the next row

addi $t0, $t0, 1

blt $t0, 8, trap\_row\_loop

end\_trap\_opponents:

# Restore registers

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

addi $sp, $sp, 12

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* CHECK WINNER PROCEDURE

#\* Checks for a win after placing a disc. Declares a win if the placed

#\* disc results in exactly 4 consecutive matching pieces in any direction

#\* (horizontal, vertical, forward diagonal, backward diagonal).

#\* Inputs:

#\* $a0: Row of the placed disc

#\* $a1: Column of the placed disc

#\* Outputs:

#\* $v0: 1 if a win is detected, 0 otherwise

#\* $v1: 1 if Player 1 ('x') wins, 2 if Player 2 ('o') wins, 0 if no win

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

checkWinner:

addi $sp, $sp, -20

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

sw $s2, 12($sp)

sw $s3, 16($sp)

move $s0, $a0 # Row of the placed disc

move $s1, $a1 # Column of the placed disc

la $s2, board # Base address of the board

mul $t2, $s0, 8

add $t2, $t2, $s1

add $t2, $t2, $s2

lb $s3, 0($t2) # Load the symbol at the placed position ('x' or 'o')

beq $s3, ' ', no\_win # If the position is empty, no win possible

# Horizontal Check

li $t3, 1 # Total count (start with the placed disc)

move $t1, $s1 # Column iterator for left

move $t5, $s1 # Column iterator for right

horizontal\_loop:

# Check left

addi $t1, $t1, -1

blt $t1, 0, check\_right

mul $t6, $s0, 8

add $t6, $t6, $t1

add $t6, $t6, $s2

lb $t7, 0($t6)

bne $t7, $s3, check\_right

addi $t3, $t3, 1

j horizontal\_loop

check\_right:

# Check right

addi $t5, $t5, 1

bge $t5, 8, horizontal\_done

mul $t6, $s0, 8

add $t6, $t6, $t5

add $t6, $t6, $s2

lb $t7, 0($t6)

bne $t7, $s3, horizontal\_done

addi $t3, $t3, 1

j check\_right

horizontal\_done:

beq $t3, 4, win\_detected # Win if exactly 4 pieces are found

# Vertical Check

li $t3, 1 # Total count (start with the placed disc)

move $t1, $s0 # Row iterator for up

move $t5, $s0 # Row iterator for down

vertical\_loop:

# Check up

addi $t1, $t1, -1

blt $t1, 0, check\_down

mul $t6, $t1, 8

add $t6, $t6, $s1

add $t6, $t6, $s2

lb $t7, 0($t6)

bne $t7, $s3, check\_down

addi $t3, $t3, 1

j vertical\_loop

check\_down:

# Check down

addi $t5, $t5, 1

bge $t5, 8, vertical\_done

mul $t6, $t5, 8

add $t6, $t6, $s1

add $t6, $t6, $s2

lb $t7, 0($t6)

bne $t7, $s3, vertical\_done

addi $t3, $t3, 1

j check\_down

vertical\_done:

beq $t3, 4, win\_detected # Win if exactly 4 pieces are found

# Forward Diagonal (Top-Left to Bottom-Right)

li $t3, 1 # Total count (start with the placed disc)

move $t1, $s0 # Row iterator for up-left

move $t5, $s1 # Column iterator for up-left

move $t8, $s0 # Row iterator for down-right

move $t9, $s1 # Column iterator for down-right

diagonal\_forward\_loop:

# Check up-left

addi $t1, $t1, -1

addi $t5, $t5, -1

blt $t1, 0, check\_down\_right

blt $t5, 0, check\_down\_right

mul $t6, $t1, 8

add $t6, $t6, $t5

add $t6, $t6, $s2

lb $t7, 0($t6)

bne $t7, $s3, check\_down\_right

addi $t3, $t3, 1

j diagonal\_forward\_loop

check\_down\_right:

# Check down-right

addi $t8, $t8, 1

addi $t9, $t9, 1

bge $t8, 8, diagonal\_forward\_done

bge $t9, 8, diagonal\_forward\_done

mul $t6, $t8, 8

add $t6, $t6, $t9

add $t6, $t6, $s2

lb $t7, 0($t6)

bne $t7, $s3, diagonal\_forward\_done

addi $t3, $t3, 1

j check\_down\_right

diagonal\_forward\_done:

beq $t3, 4, win\_detected # Win if exactly 4 pieces are found

# Backward Diagonal (Top-Right to Bottom-Left)

li $t3, 1 # Total count (start with the placed disc)

move $t1, $s0 # Row iterator for up-right

move $t5, $s1 # Column iterator for up-right

move $t8, $s0 # Row iterator for down-left

move $t9, $s1 # Column iterator for down-left

diagonal\_backward\_loop:

# Check up-right

addi $t1, $t1, -1

addi $t5, $t5, 1

blt $t1, 0, check\_down\_left

bge $t5, 8, check\_down\_left

mul $t6, $t1, 8

add $t6, $t6, $t5

add $t6, $t6, $s2

lb $t7, 0($t6)

bne $t7, $s3, check\_down\_left

addi $t3, $t3, 1

j diagonal\_backward\_loop

check\_down\_left:

# Check down-left

addi $t8, $t8, 1

addi $t9, $t9, -1

bge $t8, 8, diagonal\_backward\_done

blt $t9, 0, diagonal\_backward\_done

mul $t6, $t8, 8

add $t6, $t6, $t9

add $t6, $t6, $s2

lb $t7, 0($t6)

bne $t7, $s3, diagonal\_backward\_done

addi $t3, $t3, 1

j check\_down\_left

diagonal\_backward\_done:

beq $t3, 4, win\_detected

j no\_win

no\_win:

li $v0, 0

li $v1, 0

j end\_check\_winner

win\_detected:

li $v0, 1

beq $s3, 'x', player\_1\_win

li $v1, 2

j end\_check\_winner

player\_1\_win:

li $v1, 1

end\_check\_winner:

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

lw $s2, 12($sp)

lw $s3, 16($sp)

addi $sp, $sp, 20

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\* DRAW COIN FLIP PROCEDURE

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

drawHeads:

addi $sp, $sp, -4

sw $ra, 0($sp)

li $t0, 12

li $t1, 12

li $t2, 0

clear\_heads\_loop:

li $t3, 0

clear\_heads\_col:

add $t4, $t0, $t3

add $t5, $t1, $t2

draw\_pixel($t4, $t5, BLACK)

addi $t3, $t3, 1

blt $t3, 8, clear\_heads\_col

addi $t2, $t2, 1

blt $t2, 8, clear\_heads\_loop

li $t0, 12

li $t1, 12

li $t2, 0

heads\_row\_loop:

li $t3, 0

heads\_col\_loop:

add $t4, $t0, $t3

add $t5, $t1, $t2

subi $t6, $t4, 15

subi $t7, $t5, 15

mul $t6, $t6, $t6

mul $t7, $t7, $t7

add $t8, $t6, $t7

li $t9, 12

bgt $t8, $t9, heads\_next\_col

beq $t8, $t9, heads\_edge

beq $t2, 4, heads\_h

draw\_pixel($t4, $t5, SILVER)

j heads\_next\_col

heads\_edge:

draw\_pixel($t4, $t5, YELLOW)

j heads\_next\_col

heads\_h:

beq $t3, 3, heads\_h\_draw

beq $t3, 4, heads\_h\_draw

draw\_pixel($t4, $t5, SILVER)

j heads\_next\_col

heads\_h\_draw:

draw\_pixel($t4, $t5, RED)

heads\_next\_col:

addi $t3, $t3, 1

blt $t3, 8, heads\_col\_loop

addi $t2, $t2, 1

blt $t2, 8, heads\_row\_loop

lw $ra, 0($sp)

addi $sp, $sp, 4

jr $ra

drawTails:

addi $sp, $sp, -4

sw $ra, 0($sp)

li $t0, 12

li $t1, 12

li $t2, 0

clear\_tails\_loop:

li $t3, 0

clear\_tails\_col:

add $t4, $t0, $t3

add $t5, $t1, $t2

draw\_pixel($t4, $t5, BLACK)

addi $t3, $t3, 1

blt $t3, 8, clear\_tails\_col

addi $t2, $t2, 1

blt $t2, 8, clear\_tails\_loop

li $t0, 12

li $t1, 12

li $t2, 0

tails\_row\_loop:

li $t3, 0

tails\_col\_loop:

add $t4, $t0, $t3

add $t5, $t1, $t2

subi $t6, $t4, 15

subi $t7, $t5, 15

mul $t6, $t6, $t6

mul $t7, $t7, $t7

add $t8, $t6, $t7

li $t9, 12

bgt $t8, $t9, tails\_next\_col

beq $t8, $t9, tails\_edge

beq $t2, 4, tails\_t

draw\_pixel($t4, $t5, SILVER)

j tails\_next\_col

tails\_edge:

draw\_pixel($t4, $t5, YELLOW)

j tails\_next\_col

tails\_t:

beq $t3, 3, tails\_t\_draw

draw\_pixel($t4, $t5, SILVER)

j tails\_next\_col

tails\_t\_draw:

draw\_pixel($t4, $t5, GREEN)

tails\_next\_col:

addi $t3, $t3, 1

blt $t3, 8, tails\_col\_loop

addi $t2, $t2, 1

blt $t2, 8, tails\_row\_loop

lw $ra, 0($sp)

addi $sp, $sp, 4

jr $ra

drawEdgeVertical:

addi $sp, $sp, -4

sw $ra, 0($sp)

li $t0, 12

li $t1, 12

li $t2, 0

clear\_edge\_v\_loop:

li $t3, 0

clear\_edge\_v\_col:

add $t4, $t0, $t3

add $t5, $t1, $t2

draw\_pixel($t4, $t5, BLACK)

addi $t3, $t3, 1

blt $t3, 8, clear\_edge\_v\_col

addi $t2, $t2, 1

blt $t2, 8, clear\_edge\_v\_loop

li $t0, 15

li $t1, 12

li $t2, 0

edge\_v\_loop:

add $t5, $t1, $t2

draw\_pixel($t0, $t5, YELLOW)

addi $t2, $t2, 1

blt $t2, 8, edge\_v\_loop

lw $ra, 0($sp)

addi $sp, $sp, 4

jr $ra

drawEdgeHorizontal:

addi $sp, $sp, -4

sw $ra, 0($sp)

li $t0, 12

li $t1, 12

li $t2, 0

clear\_edge\_h\_loop:

li $t3, 0

clear\_edge\_h\_col:

add $t4, $t0, $t3

add $t5, $t1, $t2

draw\_pixel($t4, $t5, BLACK)

addi $t3, $t3, 1

blt $t3, 8, clear\_edge\_h\_col

addi $t2, $t2, 1

blt $t2, 8, clear\_edge\_h\_loop

li $t0, 12

li $t1, 15

li $t2, 0

edge\_h\_loop:

add $t4, $t0, $t2

draw\_pixel($t4, $t1, YELLOW)

addi $t2, $t2, 1

blt $t2, 8, edge\_h\_loop

lw $ra, 0($sp)

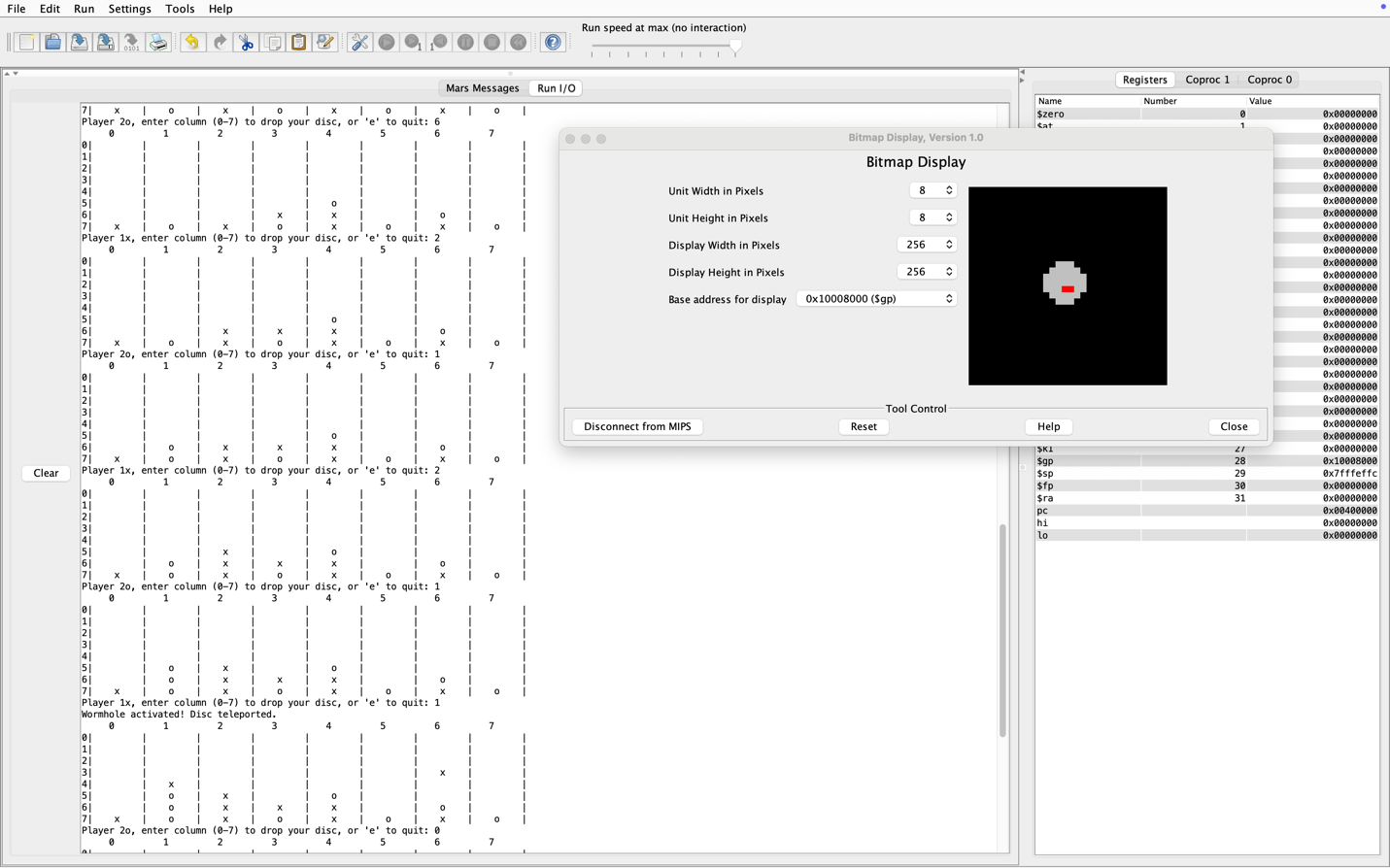
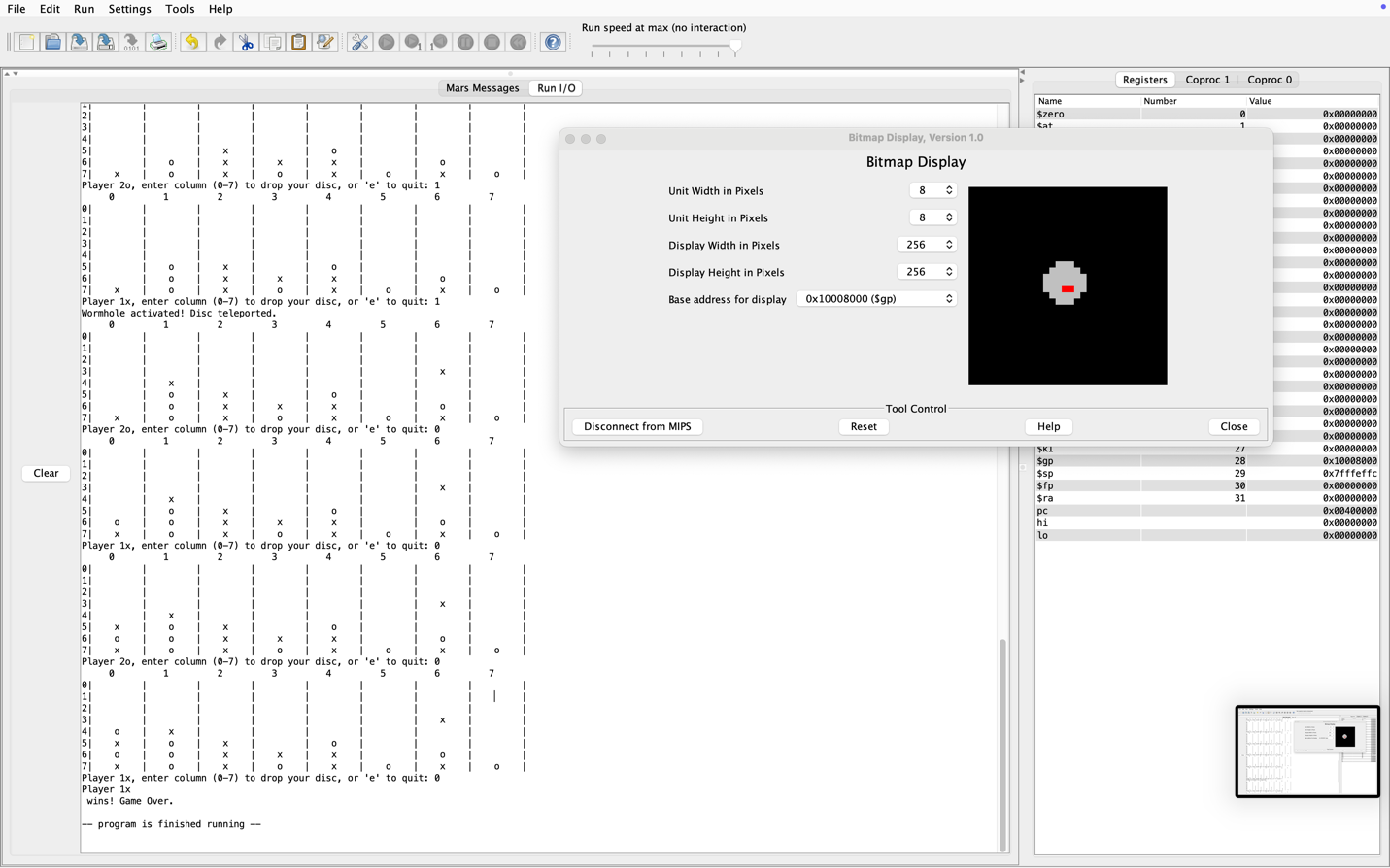
addi $sp, $sp, 4

jr $ra

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A screenshot of a computer

AI-generated content may be incorrect.



CONCLUSION

In short, if I had to do it over again, I would add more features to it, making sure that the code has more interesting options. Although it is a simple game, I do enjoy making and learning from it. I understand how MIPS works to better understand machine language. I hope that from this project, I can do more than just a game implementing finding hard drive serial numbers or finding a sort of sequential number.

TOTAL HOURS: ~ 2500 hours